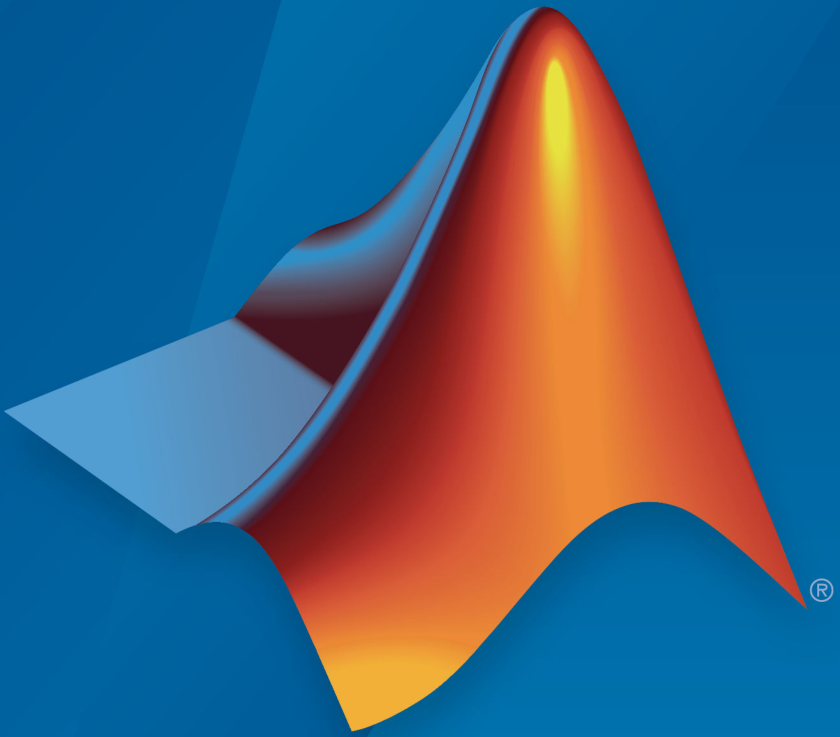
WLAN System Toolbox™

Reference



MATLAB®

R2017b

MathWorks®

WLAN System Toolbox™ Reference

Revision History

October 2015	Online only	New for Version 1.0 (R2015b)
March 2016	Online only	Revised for Version 1.1 (Release 2016a)
September 2016	Online only	Revised for Version 1.2 (Release 2016b)
March 2017	Online only	Revised for Version 1.3 (Release 2017a)
May 2017	Online only	Revised for Version 1.4 (Release 2017b)

Contents

functions — Alphabetical List

wlanBCCDecode

Convolutionally decode input data

Syntax

```
y = wlanBCCDecode(sym,rate)
y = wlanBCCDecode(sym,rate,decType)
y = wlanBCCDecode(sym,rate,tDepth)
y = wlanBCCDecode(sym,rate,decType,tDepth)
```

Description

`y = wlanBCCDecode(sym,rate)` convolutionally decodes the input `sym` using a binary convolutional code (BCC) at the specified `rate`. The BCC is defined in IEEE® 802.11™-2012 Sections 18.3.5.6 and 20.3.11.6.

`y = wlanBCCDecode(sym,rate,decType)` specifies the decoding type of the Viterbi decoding algorithm.

`y = wlanBCCDecode(sym,rate,tDepth)` specifies the traceback depth of the Viterbi decoding algorithm.

`y = wlanBCCDecode(sym,rate,decType,tDepth)` speficies the decoding type and the traceback depth. `decType` and `tDepth` can be placed in any order after `rate`.

Examples

BCC-Decode Two Encoded Streams

Decode two encoded streams of soft bits by using a BCC of rate 1/2.

Create the sequence of data bits.

```
dataBits = randi([0 1],100,1,'int8');
```

Parse the data bits as defined in IEEE® 802.11™-2012 Section 20.3.11.5 and IEEE® 802.11ac™-2013 Section 22.3.10.5.2. `numES` is the number of encoded streams.

```
numES = 2;
parsedData = reshape(dataBits,numES,[]).';
```

BCC-encode the parsed sequence.

```
encodedData = wlanBCCEncode(parsedData,'1/2');
```

Convert the encoded bits to soft bits (i.e. LLR demodulation).

```
demodData = double(1-2*encodedData);
```

BCC-decode the demodulated data.

```
decodedData = wlanBCCDecode(demodData,'1/2');
```

Deparse the decoded data.

```
deparsedData = reshape(decodedData.',[],1);
```

Verify that the decoded data matches the original data.

```
isequal(dataBits,deparsedData)
```

```
ans = logical
   1
```

BCC-Decode Soft Bits

Decode a sequence of soft bits by using a BCC of rate 3/4 and a traceback depth of 60.

Create the sequence of data bits.

```
dataBits = randi([0 1],300,1);
```

BCC-encode the sequence of bits.

```
encodedData = wlanBCCEncode(dataBits,3/4);
```

Convert the encoded bits to soft bits (i.e. LLR demodulation).

```
demodData = 1-2*encodedData;
```

BCC-decode the demodulated bits.

```
tDepth = 60;
decodedData = wlanBCCDecode(demodData,3/4,tDepth);
```

Verify that the decoded data matches the original data.

```
isequal(dataBits,decodedData)
```

```
ans = logical
   1
```

BCC-Decode Hard Bits

Decode a sequence of hard bits by using a BCC of rate 3/4 and a traceback depth of 45.

Create the sequence of data bits.

```
dataBits = randi([0 1],300,1,'int8');
```

BCC-encode the sequence of bits.

```
encodedData = wlanBCCEncode(dataBits,'2/3');
```

Perform hard BCC decoding on the encoded bits. Specify a traceback depth 45.

```
tDepth = 45;
decodedBits = wlanBCCDecode(encodedData,'2/3','hard',tDepth);
```

Verify that the decoded bits match the original bits.

```
isequal(dataBits,decodedBits)
```

```
ans = logical
   1
```

Input Arguments

`sym` — Input sequence
matrix

Input sequence of symbols to decode, specified as a numeric matrix of integers. The number of columns must be the number of encoded streams. Each stream is encoded separately. When `decType` is `'soft'` or not specified, `sym` must be a real matrix with log-likelihood ratios. Positive values represent a logical 0 and negative values represent a logical 1.

Data Types: `double` | `int8`

`rate` — Code rate
1/2 | 2/3 | 3/4 | 5/6

Code rate of the binary convolutional code (BCC), specified as a scalar, character array, or string scalar. `rate` must be a numeric value equal to 1/2, 2/3, 3/4, or 5/6, or a character vector or string scalar equal to `'1/2'`, `'2/3'`, `'3/4'`, or `'5/6'`.

Example: `'3/4'`

Data Types: `double` | `char` | `string`

`decType` — Decoding type
`'soft'` (default) | `'hard'`

Decoding type of the binary convolutional code (BCC), specified as a character vector or a string scalar. It can be `'hard'` for a hard input Viterbi algorithm, or `'soft'` for a soft input Viterbi algorithm without any quantization.

Data Types: `char` | `string`

`tDepth` — Traceback depth
positive integer

Traceback depth of the Viterbi decoding algorithm, specified as a positive integer less than or equal to the number of input symbols in `sym`.

Example: `y = wlanBCCDecode(sym,'1/2','hard',50)`

Data Types: `double`

Output Arguments

`y` — Binary convolutionally decoded output
matrix

Binary convolutionally decoded output, returned as a binary matrix of integers. The number of rows of `y` is equal to the number of rows of input `sym` multiplied by `rate`, rounded to the next integer. The number of columns of `y` is equal to the number of columns of `sym`.

Data Types: `int8`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`vitdec` | `wlanBCCEncode`

Introduced in R2017b

wlanBCCDeinterleave

Deinterleave binary convolutionally interleaved input

Syntax

```
y = wlanBCCDeinterleave(bits,type,numCBPSSI,cbw)
y = wlanBCCDeinterleave(bits,type,numCBPSSI)
```

Description

`y = wlanBCCDeinterleave(bits,type,numCBPSSI,cbw)` outputs the binary convolutionally deinterleaved input `bits` for a specified interleaver `type`, as defined in IEEE 802.11-2012 Section 18.3.5.7, IEEE 802.11ac™-2013 Section 22.3.10.8, and IEEE 802.11ah™ Section 24.3.9.8. `numCBPSSI` specifies the number of coded bits per OFDM symbol per spatial stream per interleaver block and `cbw` speficies the channel bandwidth.

`y = wlanBCCDeinterleave(bits,type,numCBPSSI)` outputs the deinterleaved input `bits` for the non-HT interleaver `type`.

Examples

Interleave and Deinterleave VHT Data Field

Perform BCC interleaving and deinterleaving for the VHT interleaving type.

Define the input parameters. Set the number of coded bits per OFDM symbol per spatial stream per interleaver block to 52, the channel bandwidth to 20Mhz and the number of spatial streams, named as numSS, to 4.

```
numCBPSSI = 52;
chanBW = 'CBW20';
numSS = 4;
```

Create a sequence of bits for two OFDM symbols, four spatial streams, and one segment.

```
bits = randi([0 1],(2*numCBPSSI),numSS,1);
```

Perform BCC interleaving on the bits.

```
intBits = wlanBCCInterleave(bits,'VHT',numCBPSSI,chanBW);
```

Perform BCC deinterleaving on the interleaved bits.

```
out = wlanBCCDeinterleave(intBits,'VHT',numCBPSSI,chanBW);
```

Verify that the deinterleaved data matches the original data.

```
isequal(bits,out)
```

```
ans = logical
   1
```

Interleave and Deinterleave Non-HT Data Field

Perform BCC interleaving and deinterleaving for the non-HT interleaving type.

Define the input parameters. Set the number of coded bits per OFDM symbol per spatial stream per interleaver block to 48.

```
numCBPSSI = 48;
```

Create a sequence of random bits for one OFDM symbol, one spatial stream, and one segment.

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

Perform BCC interleaving on the bits.

```
intBits = wlanBCCInterleave(bits,'Non-HT',numCBPSSI);
```

Perform BCC deinterleaving on the interleaved bits.

```
out = wlanBCCDeinterleave(intBits,'Non-HT',numCBPSSI);
```

Verify that the deinterleaved data matches the original data.

```
isequal(bits,out)
```

```
ans = logical
   1
```

Input Arguments

`bits` — **Input sequence**
matrix | 3-D array

Input sequence containing binary convolutionally interleaved data, specified as an (N_{CBPSSI}×N_{SYM})-by-N_{SS}-by-N_{SEG} array, where:

- N_{CBPSSI} is the number of coded bits per OFDM symbol per spatial stream per interleaver block.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.
  - If `type`= `'Non-HT'`, then N_{SS} must be 1.
  - If `type`= `'VHT'`, then N_{SS} must be from 1 to 8.
- N_{SEG} is the number of segments.

Data Types: `double`

`type` — **Type of interleaving**
`'VHT'` | `'Non-HT'`

The type of interleaving, specified as `'VHT'` or `'Non-HT'`.

Data Types: `char` | `string`

`numCBPSSI` — **Number of coded bits per OFDM symbol per spatial stream per interleaver block**
positive integer

Number of coded bits per OFDM symbol per spatial stream per interleaver block specified as a positive integer. As defined in IEEE 802.11ac-2013 Table 22-6, the value of `numCBPSSI` depends on the interleaving type:

`'Non-HT'`	N_{SD}×N_{BPSCS}

`'VHT'`	$N_{SD} \times N_{BPSCS} / N_{SEG}$

where:

- N_{SD} is the number of data subcarriers.
- N_{BPSCS} is the number of coded bits per subcarrier per spatial stream, specified as 1, 2, 4, 6, or 8.
- N_{SEG} is the number of segments.

When `type= 'Non-HT'`, `numCBPSSI` can be 48, 96, 192, 288, and 384, since $N_{CBPSSI} = 48 \times N_{BPSCS}$.

When `type= 'VHT'`, `numCBPSSI` can be 24, 48, 96, 144, and 192, since $N_{CBPSSI} = 24 \times N_{BPSCS}$.

Data Types: `double`

`cbw` — Channel bandwidth
`'CBW1'` | `'CBW2'` | `'CBW4'` | `'CBW8'` | `'CBW10'` | `'CBW16` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW1'`,`'CBW2'`, `'CBW4'`,`'CBW8'`, `'CBW10'`, `'CBW16'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`. When the interleaver type is set to `'Non-HT'`, then `cbw` is optional.

Data Types: `char` | `string`

Output Arguments

`y` — Deinterleaved output
matrix | 3-D array

Deinterleaved output, returned as an ($N_{CBPSSI} \times N_{SYM}$)-by-$N_{SS}$-by-$N_{SEG}$ array, where:

- N_{CBPSSI} is the number of coded bits per OFDM symbol per spatial stream per interleaver block.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.
- N_{SEG} is the number of segments.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`convdeintrlv` | `wlanBCCInterleave`

Introduced in R2017b

wlanBCCEncode

Convolutionally encode binary data

Syntax

```
y = wlanBCCEncode(bits,rate)
```

Description

`y = wlanBCCEncode(bits,rate)` convolutionally encodes the binary input `bits` using a binary convolutional code (BCC) at the specified `rate`. The BCC is defined in IEEE 802.11-2012 Sections 18.3.5.6 and 20.3.11.6.

Examples

BCC-Encode Bits

Encode a sequence of data bits by using a BCC of rate 3/4.

Create the sequence of data bits.

```
dataBits = randi([0 1],300,1);
```

BCC-encode the data bits.

```
encodedData = wlanBCCEncode(dataBits,'3/4');
size(encodedData)
```

```
ans =

   400     1
```

BCC-Encode Two Streams

Encode two streams of data bits by using a BCC of rate 1/2.

Create the sequence of data bits.

```
dataBits = randi([0 1],100,1,'int8');
```

Parse the sequence of bits as defined in IEEE® 802.11™-2012 Section 20.3.11.5 and IEEE® 802.11ac™-2013 Section 22.3.10.5.2. `numES` is the number of encoded streams.

```
numES = 2;
parsedData = reshape(dataBits,numES,[]).';
```

BCC-encode the parsed sequence.

```
encodedData = wlanBCCEncode(parsedData,1/2);
size(encodedData)
```

```
ans =

   100     2
```

Input Arguments

`bits` — **Input sequence**
matrix

Input sequence with data bits to encode, specified as a binary matrix. The number of columns must equal the number of encoded streams. Each stream is encoded separately.

Data Types: `double` | `int8`

`rate` — **Code rate**
1/2 | 2/3 | 3/4 | 5/6

Code rate of the binary convolutional code (BCC), specified as a scalar, character array, or string scalar. `rate` must be a numeric value equal to 1/2, 2/3, 3/4, or 5/6, or a character vector or string scalar equal to `'1/2'`, `'2/3'`, `'3/4'`, or `'5/6'`.

Example: `'1/2'`

Data Types: `double` | `char` | `string`

Output Arguments

`y` — Binary convolutionally encoded output
matrix

Binary convolutionally encoded output, returned as a binary matrix of the same type of `bits`. The number of rows of `y` is the result of dividing the number of rows of input `bits` by `rate`, rounded to the next integer. The number of columns of `y` is equal to the number of columns of `bits`.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also
`convenc` | `wlanBCCDecode`

Introduced in R2017b

wlanBCCInterleave

Interleave binary convolutionally encoded input

Syntax

```
y = wlanBCCInterleave(bits,type,numCBPSSI,cbw)
y = wlanBCCInterleave(bits,type,numCBPSSI)
```

Description

`y = wlanBCCInterleave(bits,type,numCBPSSI,cbw)` outputs the interleaved binary convolutionally encoded (BCC) input `bits` for a specified interleaver `type`, as defined in IEEE 802.11-2012 Section 18.3.5.7, IEEE 802.11ac-2013 Section 22.3.10.8, and IEEE 802.11ah Section 24.3.9.8. `numCBPSSI` specifies the number of coded bits per OFDM symbol per spatial stream per interleaver block and `cbw` speficies the channel bandwidth.

`y = wlanBCCInterleave(bits,type,numCBPSSI)` outputs the interleaved input `bits` for the non-HT interleaver `type`.

Examples

Interleave VHT Data Field

Perform BCC interleaving for the 'VHT' interleaving type.

Define the input parameters. Set the number of coded bits per OFDM symbol per spatial stream per interleaver block to 52, the channel bandwidth to 20Mhz and the number of spatial streams, named as `numSS,` to 4.

```
numCBPSSI = 52;
cbw = 'CBW20';
numSS = 4;
```

Create a sequence of bits for two OFDM symbols, four spatial streams, and one segment.

```
inBits = randi([0 1],(2*numCBPSSI),numSS,1,'int8');
```

Perform BCC interleaving on the bits.

```
out = wlanBCCInterleave(inBits,'VHT',numCBPSSI,cbw);
```

Interleave Non-HT Data Field

Perform BCC interleaving for the non-HT interleaving type.

Define the input parameters. Set the number of coded bits per OFDM symbol per spatial stream per interleaver block to 48.

```
numCBPSSI = 48;
```

Create a sequence of random bits for one OFDM symbol, one spatial stream, and one segment.

```
inBits = randi([0 1],numCBPSSI,1);
```

Perform BCC interleaving on the bits.

```
out = wlanBCCInterleave(inBits,'Non-HT',numCBPSSI);
```

Compare the original sequence with the interleaved one.

```
[inBits out]
```

```
ans =

     1     1
     1     0
     0     0
     1     1
     1     1
     0     0
     0     0
     1     1
     1     0
     1     1
```

Interleave Sequence

Get the interleaving sequence of a non-HT interleaver type.

Define the input parameters. Set the number of coded bits per OFDM symbol per spatial stream per interleaver block to 192.

```
numCBPSSI = 192;
```

Create a numeric sequence from 1 to `numCBPSSI`.

```
seq = (1:numCBPSSI).';
```

Perform BCC interleaving on the numeric sequence.

```
intSeq = wlanBCCInterleave(seq,'Non-HT',numCBPSSI);
intSeq(1:10)
```

```
ans =

     1
    17
    33
    49
    65
    81
    97
   113
   129
   145
```

Input Arguments

`bits` **— Input sequence**
matrix | 3-D array

Input sequence containing binary convolutionally encoded (BCC) data, specified as an (N_{CBPSSI}×N_{SYM})-by-N_{SS}-by-N_{SEG} array, where:

- N_{CBPSSI} is the number of coded bits per OFDM symbol per spatial stream per interleaver block.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.
  - If `type`= `'Non-HT'`, then N_{SS} must be 1.
  - If `type`= `'VHT'`, then N_{SS} must be from 1 to 8.
- N_{SEG} is the number of segments.

Data Types: `double` | `int8`

`type` — Type of interleaving
`'VHT'` | `'Non-HT'`

The type of interleaving, specified as `'VHT'` or `'Non-HT'`.

Data Types: `char` | `string`

`numCBPSSI` — Number of coded bits per OFDM symbol per spatial stream per interleaver block
positive integer

Number of coded bits per OFDM symbol per spatial stream per interleaver block specified as a positive integer. As defined in IEEE 802.11ac-2013 Table 22-6, the value of `numCBPSSI` depends on the interleaving type:

`'Non-HT'`	$N_{\mathrm{SD}} \times N_{\mathrm{BPSCS}}$
`'VHT'`	$N_{\mathrm{SD}} \times N_{\mathrm{BPSCS}} / N_{\mathrm{SEG}}$

where:

- N_{SD} is the number of data subcarriers.
- N_{BPSCS} is the number of coded bits per subcarrier per spatial stream, specified as 1, 2, 4, 6, or 8.
- N_{SEG} is the number of segments.

When `type`= `'Non-HT'`, `numCBPSSI` can be 48, 96, 192, 288, and 384, since $N_{\mathrm{CBPSSI}} = 48 \times N_{\mathrm{BPSCS}}$.

When `type`= `'VHT'`, `numCBPSSI` can be 24, 48, 96, 144, and 192, since $N_{\mathrm{CBPSSI}} = 24 \times N_{\mathrm{BPSCS}}$.

Data Types: `double`

`cbw` — Channel bandwidth
`'CBW1'` | `'CBW2'` | `'CBW4'` | `'CBW8'` | `'CBW10'` | `'CBW16` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW1'`,`'CBW2'`, `'CBW4'`,`'CBW8'`, `'CBW10'`, `'CBW16'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`. When the interleaver type is set to `'Non-HT'`, then `cbw` is optional.

Data Types: `char` | `string`

Output Arguments

`y` — Interleaved output
matrix | 3-D array

Interleaved output, returned as an (N_{CBPSSI}×N_{SYM})-by-N_{SS}-by-N_{SEG} array, where:

- N_{CBPSSI} is the number of coded bits per OFDM symbol per spatial stream per interleaver block.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.
- N_{SEG} is the number of segments.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`convintrlv` | `wlanBCCDeinterleave`

Introduced in R2017b

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 1-28[1] samples and channel bandwidth.

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

Examples

Coarse Estimate of CFO for Non-HT Waveform

Create a non-HT configuration object.

```
nht = wlanNonHTConfig;
```

Generate a non-HT waveform.

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

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

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

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

```
tgacChan = 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 = tgacChan(txstf);
rxstf = noise(rxstfNoNoise);
```

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

```
pfOffset = comm.PhaseFrequencyOffset('SampleRate',fs, ...
    'FrequencyOffsetSource','Input port');
rxstf = pfOffset(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(tgacChan)
tgacChan.DelayProfile = 'Model-E';
```

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

```
rxstfNoNoise = tgacChan(txstf);
rxstf = noise(rxstfNoNoise);
rxstf = pfOffset(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 = pfOffset(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 format configuration object.

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

Generate the transmit waveform.

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

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

```
tgnChan = 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 = tgnChan(txSig);
rxSig = noise(rxSigNoNoise);
```

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

```
pfOffset = comm.PhaseFrequencyOffset('SampleRate',fs,'FrequencyOffsetSource','Input por
rxSig = pfOffset(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 =

   2.0003e+03
```

Correct for the estimated offset.

```
rxSigCorr1 = pfOffset(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 =

    6.5375
```

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

Correct for the remaining offset.

```
rxSigCorr2 = pfOffset(rxSigCorr1,-foffset2);
```

Determine the frequency offset of the twice corrected signal.

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

```
deltaFreq =

  -1.6112e-13
```

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`

`cbw` — Channel bandwidth

`'CBW5'` | `'CBW10'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as: `'CBW5'`, `'CBW10'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Data Types: `char` | `string`

`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. The duration of the short training symbol varies with bandwidth. For more information, see “L-STF” on page 1-28.

Data Types: `double`

Output Arguments

`fOffset` — Frequency offset

real scalar

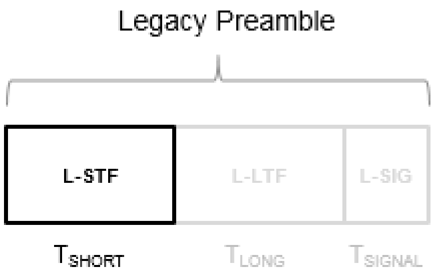
Frequency offset in Hz, returned as a real scalar.

Data Types: `double`

Definitions

L-STF

The legacy short training field (L-STF) is the first field of the 802.11 OFDM PLCP legacy preamble. The L-STF is a component of VHT, HT, and non-HT PPDUs.



The L-STF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	L-STF Duration ($T_{SHORT} = 10 \times T_{FFT} / 4$)
20, 40, 80, and 160	312.5	3.2 μs	8 μs
10	156.25	6.4 μs	16 μs
5	78.125	12.8 μs	32 μs

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 per 20 MHz channel bandwidth segment. For 5 MHz, 10 MHz, and 20 MHz bandwidths, the number of channel bandwidths segments is 1.

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

[3] 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.

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

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

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

Introduced in R2015b

wlanConstellationDemap

Constellation demapping

Syntax

```
y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS)
y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS,demapType)
y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS,phase)
y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS,demapType,phase)
```

Description

`y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS)` demaps the received input `sym` using the soft-decision approximate LLR method for the specified number of coded bits per subcarrier per spatial stream `numBPSCS`. The received symbols must be generated with one of these modulations:

- BPSK, QPSK, 16QAM, or 64QAM, as per IEEE 802.11-2012, Section 18.3.5.8
- 256QAM, as per IEEE 802.11ac-2012, Section 22.3.10.9.1
- 1024QAM, as per IEEE 802.11-16/0922r2

`y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS,demapType)` specifies the demapping type.

`y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS,phase)` derotates the symbols clockwise before demapping by the number of radians specified in `phase`.

`y = wlanConstellationDemap(sym,noiseVarEst,numBPSCS,demapType,phase)` specifies the demapping type and the phase rotation.

Examples

256QAM Demapping

Perform a 256QAM demapping, as defined in IEEE® 802.11ac™-2013, Section 22.3.10.9.1.

Create the sequence of data bits.

```
bits = randi([0 1],416,1,'int8');
```

Perform the constellation mapping on the data bits by using a 256QAM modulation. The size of the output returned equals the size of the input sequence divided by eight.

```
numBPSCS = 8;
mappedData = wlanConstellationMap(bits,numBPSCS);
size(mappedData)
```

```
ans =

    52     1
```

Perform the 256QAM constellation demapping. Because the default demapping type is soft, the ouput is a vector of soft bits.

```
noiseVar = 0;
demappedData = wlanConstellationDemap(mappedData,noiseVar,numBPSCS);
size(demappedData)
```

```
ans =

   416     1
```

Constellation Demapping with Hard Demodulation

Perform a 256QAM demapping by using hard demodulation. The demapping is defined in IEEE® 802.11™-2012 Section 18.3.5.8

Create the sequence of data bits.

```
 bits = randi([0 1],416,1);
```

Perform the constellation mapping on the data bits by using a 256QAM constellation.

```
numBPSCS = 8;
mappedData = wlanConstellationMap(bits,numBPSCS);
```

Perform the hard 256QAM constellation demapping. Because it is a hard demapping, the estimated noise variance is ignored.

```
noiseVar = 0;
demapType = 'hard';
demappedData = wlanConstellationDemap(mappedData,noiseVar,numBPSCS,demapType);
```

Verify that the demapped data matches the original data.

```
isequal(bits,demappedData)
```

```
ans = logical
   1
```

BPSK and QBPSK Demapping for VHT-SIG-A Field

BPSK and QBPSK demapping for different OFDM symbols for the VHT-SIG-A field by using a soft demodulation. The demapping is defined in IEEE® 802.11ac™-2013 Section 22.3.8.3.3

Create the sequence of data bits. Specify the two OFDM symbols in columns.

```
 bits = randi([0 1],48,2,'int8');
```

Perform constellation mapping on the data bits. Specify the size of the constellation rotation as the number in columns of the input sequence. The first column is mapped with a BPSK modulation. The second column is modulated with a QBPSK modulation.

```
numBPSCS = 1;
phase = [0 pi/2];
mappedData = wlanConstellationMap(bits,numBPSCS,phase);
```

Perform the constellation demapping with an estimated variance noise equal to zero (no added noise). To derotate the constellation, specify the same phase as in the mapping function. The output is a vector of soft bits ready to be the input of a convolutional decoder.

```
noiseVar = 0;
demappedData = wlanConstellationDemap(mappedData,noiseVar,numBPSCS,phase);
```

Verify that the demapped data matches the original data. Because no noise is present, you can recover the original data without errors by assigning the negative values to a logical 1 and the positive values to a logical 0. In other words, you can convert the soft bits into hard bits.

```
demappedBits = int8((demappedData<=0));
isequal(bits,demappedBits)
```

```
ans = logical
   1
```

4-D Array Demapping

QBPSK demapping on a four-dimensional array by using hard demodulation..

Create the sequence of data bits as an array of four dimensions, with 416 coded bits per subcarrier per spatial stream per interleaver block, four OFDM smbols, two spatial streams, and two segments.

```
numCBPSSI = 416;
numSym = 4;
numSS = 2;
numSeg = 2;
bits = randi([0 1],numCBPSSI,numSym,numSS,numSeg);
size(bits)
```

```
ans =

   416     4     2     2
```

Perform QBPSK constellation mapping on the data bits with a rotation of $\frac{\pi}{2}$ radians.

```
numBPSCS = 1;
phase = pi/2;
mappedData = wlanConstellationMap(bits,numBPSCS,phase);
size(mappedData)
```

```
ans =

   416     4     2     2
```

Perform hard QBPSK constellation demapping. To de-rotate the constellation, specify the same phase as in the mapping function. Because it is a hard demapping, the estimated noise variance is ignored.

```
noiseVar = 0;
demapType = 'hard';
demappedData = wlanConstellationDemap(mappedData,noiseVar,numBPSCS,demapType);
```

Verify that the demapped data matches the original data.

```
isequal(bits,demappedData)
```

```
ans = logical
   1
```

Input Arguments

`sym` — Input sequence

vector | matrix | multidimensional array

Input sequence of received symbols, specified as a numeric vector, matrix, or multidimensional array of integers.

Data Types: `double`
Complex Number Support: Yes

`noiseVarEst` — Noise variance estimate

nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar. When the demapping type is set to `'hard'`, the noise variance estimate is not required and therefore is ignored.

Example: `0.7071`

Data Types: `double`

`numBPSCS` — Number of coded bits per subcarrier per spatial stream
1 | 2 | 4 | 6 | 8 | 10

Number of coded bits per subcarrier per spatial stream, specified as log2(*M*), where *M* is the modulation order. Therefore, `numBPSCS` must equal:

- 1 for a BPSK modulation
- 2 for a QPSK modulation
- 4 for a 16QAM modulation
- 6 for a 64QAM modulation
- 8 for a 256QAM modulation
- 10 for a 1024QAM modulation

Example: `4`

Data Types: `double`

`demapType` — Demapping type
`'soft'` (default) | `'hard'`

Demapping type, specified as a character vector or a string scalar. It can be `'hard'` for hard-decision demapping or `'soft'` for the soft-decision approximate LLR method.

Data Types: `double`

`phase` — Constellation rotation
scalar | vector | multidimensional array

Constellation rotation in radians, specified as a scalar, vector, or multidimensional array. The size of `phase` must be compatible with the size of the input `sym`. `phase` and `sym` have compatible sizes if, for each corresponding dimension, the dimension sizes are either equal or one of them is 1. When one of the dimensions of `sym` is equal to 1, and the corresponding dimension of `phase` is larger than 1, then the output dimensions have the same size as the dimensions of `phase`.

Example: `pi*(0:size(bits,1)/numBPSCS-1).'/2;`

Data Types: `double`

Output Arguments

`y` — Demapped symbols
vector | matrix | multidimensional array

Demapped symbols, returned as a numeric vector, matrix, or multidimensional array of integers. `y` has the same size as `sym` except for the number of rows, which is equal to the number of rows of `sym`, multiplied by `numBPSCS`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanConstellationMap`

Introduced in R2017b

wlanConstellationMap

Constellation mapping

Syntax

```
y = wlanConstellationMap(bits,numBPSCS)
y = wlanConstellationMap(bits,numBPSCS,phase)
```

Description

`y = wlanConstellationMap(bits,numBPSCS)` maps the input sequence `bits` using the number of coded bits per subcarrier per spatial stream, `numBPSCS`, to one of the following modulations:

- BPSK, QPSK, 16QAM, or 64QAM, as per IEEE 802.11-2012, Section 18.3.5.8
- 256QAM, as per IEEE 802.11ac-2012, Section 22.3.10.9.1
- 1024QAM, as per IEEE 802.11-16/0922r2

The constellation mapping is performed column-wise.

`y = wlanConstellationMap(bits,numBPSCS,phase)` rotates the constellation points counterclockwise by the number of radians specified in `phase`.

Examples

256QAM Mapping

Perform a 256QAM mapping, as defined in IEEE® 802.11ac™-2013 Section 22.3.10.9.1.

Create the sequence of data bits.

```
bits = randi([0 1],416,1,'int8');
```

Perform the constellation mapping on the data bits with a 256QAM modulation.

```
numBPSCS = 8;
mappedData = wlanConstellationMap(bits,numBPSCS);
```

The size of the output returned by this modulation equals the size of the input sequence divided by eight.

```
size(mappedData)
```

```
ans =

    52     1
```

π/2-BPSK Mapping

Perform a $\frac{\pi}{2}$-BPSK mapping on a sequence of data bits as defined in IEEE® 802.11ad™-2012 Section 21.6.3.2.4.

Create the sequence of data bits.

```
bits = randi([0 1],512,1);
```

Perform the BPSK mapping on the data bits with a rotation of $\frac{\pi}{2}$ radians. Note that the size of the constellation rotation `phase` is equal to the size of input sequence.

```
numBPSCS = 1;
phase = pi*(0:size(bits,1)/numBPSCS-1).'/2;
mappedData = wlanConstellationMap(bits,numBPSCS,phase);
```

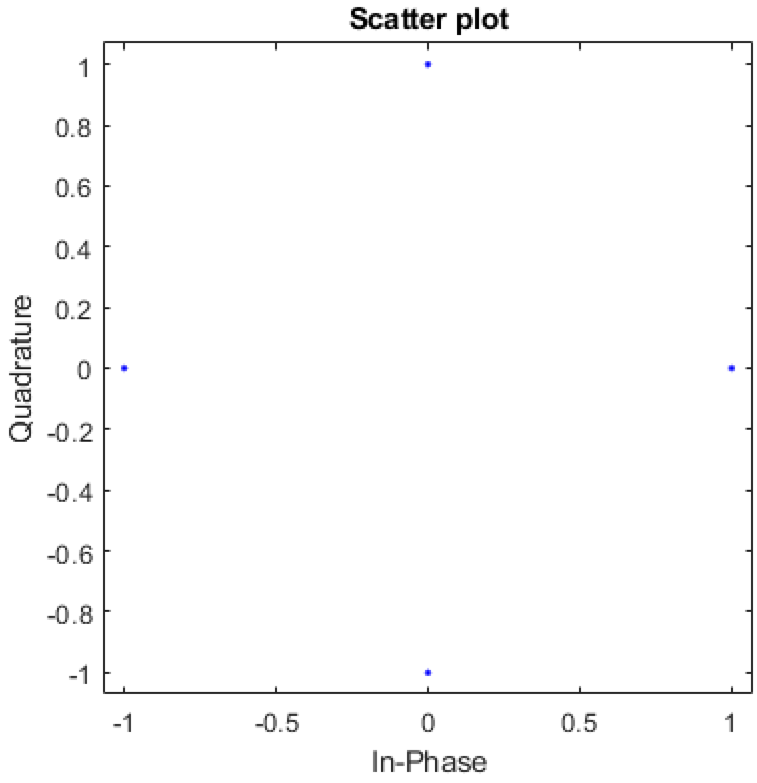
As we performed a BPSK mapping, the number of symbols per bit is one, therefore the size of the output is equal to the size of the original sequence.

```
size(mappedData)
```

```
ans =

   512     1
```

Display the modulated signal constellation using the `scatterplot` function.

```
scatterplot(mappedData);
```



BPSK and QBPSK Mapping for VHT-SIG-A field

Perform BPSK and QBPSK demapping for different OFDM symbols for the VHT-SIG-A field by using a soft demodulation. The mapping is defined in IEEE® 802.11ac™-2013 Section 22.3.8.3.3 for the VHT-SIG-A field.

Create the sequence of data bits. Place the two OFDM symbols in columns.

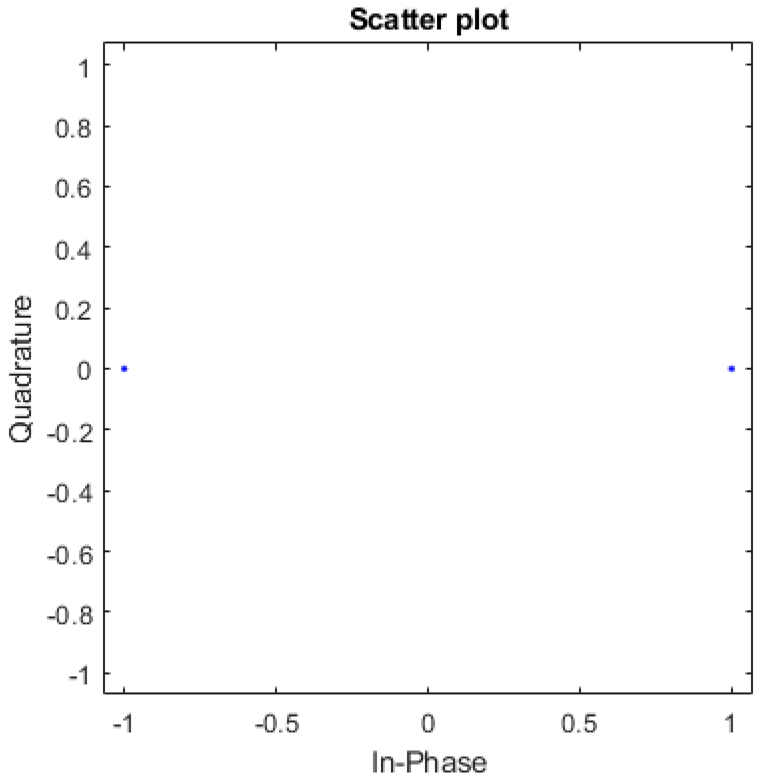
```
bits = randi([0 1],48,2,'int8');
```

Perform constellation mapping on the data bits. Specify the size of constellation rotation `phase` as the number of columns in the input sequence. The first column is mapped with a BPSK modulation. The second column is modulated with a QBPSK modulation.

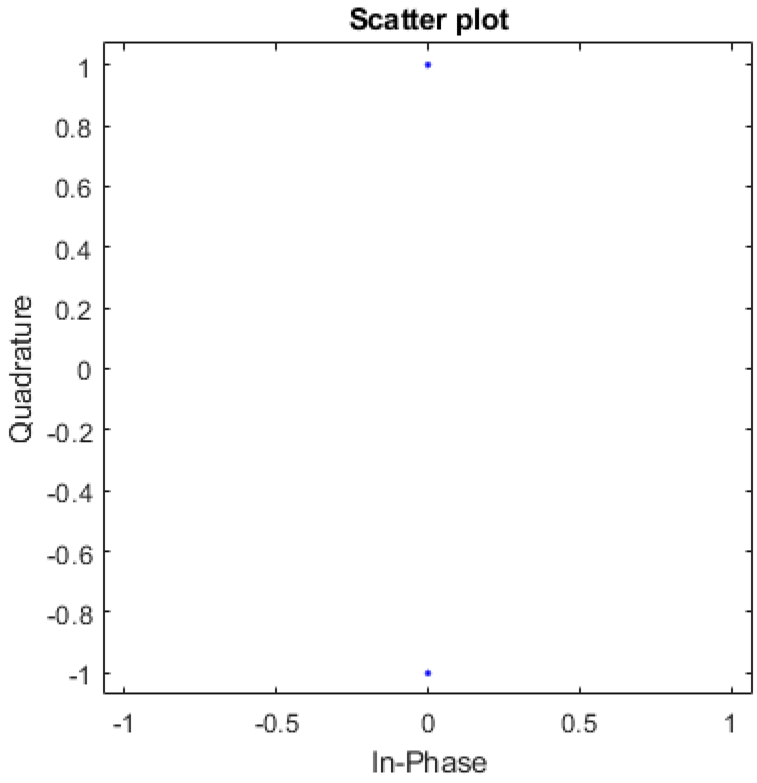
```
numBPSCS = 1;
phase = [0 pi/2];
mappedData = wlanConstellationMap(bits,numBPSCS,phase);
```

Display the modulated signal constellation by using the `scatterplot` function. The first plot shows the data after the BPSK modulation, and the second plot shows the QBPSK-modulated symbols.

```
scatterplot(mappedData(:,1))
```



```
scatterplot(mappedData(:,2))
```



Input Arguments

`bits` — Input sequence

vector | matrix | multidimensional array

Input sequence of bits to map into symbols, specified as a binary vector, matrix, or multidimensional array.

Data Types: `double` | `int8`

`numBPSCS` — Number of coded bits per subcarrier per spatial stream

1 | 2 | 4 | 6 | 8 | 10

Number of coded bits per subcarrier per spatial stream, specified as log2(*M*), where *M* is the modulation order. Therefore, `numBPSCS` must equal:

- 1 for a BPSK modulation
- 2 for a QPSK modulation
- 4 for a 16QAM modulation
- 6 for a 64QAM modulation
- 8 for a 256QAM modulation
- 10 for a 1024QAM modulation

Example: `4`

Data Types: `double`

`phase` — Constellation rotation
scalar | vector | multidimensional array

Constellation rotation in radians, specified as a scalar, vector, or multidimensional array. The size of `phase` must be compatible with the size of the input `bits`. `phase` and `bits` have compatible sizes if, for each corresponding dimension, the dimension sizes are either equal or one of them is 1. When one of the dimensions of `bits` is equal to 1, and the corresponding dimension of `phase` is larger than 1, then the output dimensions have the same size as the dimensions of `phase`.

Example: `pi*(0:size(bits,1)/numBPSCS-1).'/2;`

Data Types: `double`

Output Arguments

`y` — Mapped symbols
vector | matrix | multidimensional array

Mapped symbols, returned as a complex vector, matrix, or multidimensional array. `y` has the same size as `bits`, except for the number of rows, which is equal to the number of rows of `bits` divided by `numBPSCS`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanConstellationDemap`

Introduced in R2017b

wlanDMGConfig

Create DMG format configuration object

Syntax

```
cfgDMG = wlanDMGConfig
cfgDMG = wlanDMGConfig(Name,Value)
```

Description

`cfgDMG = wlanDMGConfig` creates a configuration object that initializes parameters for an IEEE 802.11 directional multi-gigabit (DMG) format “PPDU” on page 1-52.

`cfgDMG = wlanDMGConfig(Name,Value)` creates a DMG 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 DMG Configuration Object with Default Settings

```
cfgDMG = wlanDMGConfig
```

```
cfgDMG = 

  wlanDMGConfig with properties:

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

```
                 Turnaround: 0
```

Create DMG Configuration Object and Modify Default Settings

Create a DMG configuration object and use `Name,Value` pairs to override default settings.

```
dtpgrouppairs = (randperm(42)-1)';
cfgDMG = wlanDMGConfig('MCS',13,'TonePairingType','Dynamic', ...
    'DTPGroupPairIndex',dtpgrouppairs)
```

```
cfgDMG = 

  wlanDMGConfig with properties:

                      MCS: 13
           TrainingLength: 0
          TonePairingType: 'Dynamic'
        DTPGroupPairIndex: [42x1 double]
             DTPIndicator: 0
               PSDULength: 1000
    ScramblerInitialization: 2
            AggregatedMPDU: 0
                  LastRSSI: 0
                Turnaround: 0
```

Create DMG Configuration Object and Return DMG PHY Type

Create DMG configuration objects and change the default property settings by using dot notation. Use the `phyType` object function to access the DMG PHY modulation type.

Create a DMG configuration object and return the DMG PHY modulation type. By default, the configuration object creates properties to model the DMG control PHY.

```
dmg = wlanDMGConfig;
phyType(dmg)
```

```
ans = 
'Control'
```

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

```
dmg.MCS = 5;
phyType(dmg)
```

```
ans =
'SC'
```

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: `'MCS','13','TrainingLength',4` specifies a modulation and coding scheme of 13, which indicates OFDM PHY modulation and code rate of 1/2. Also, a PPDU with four training fields is specified for the DMG format packet.

`MCS` — Modulation and coding scheme index

`0` (default) | integer from 0 to 24

Modulation and coding scheme index, specified as an integer from 0 to 24. The `MCS` index indicates the modulation and coding scheme used in transmitting the current packet.

- Modulation and coding scheme for control PHY

MCS Index	Modulation	Coding Rate	Comment
0	DBPSK	1/2	Code rate and data rate might be lower due to codeword shortening.

- Modulation and coding schemes for single-carrier modulation

MCS Index	Modulation	Coding Rate	N_{CBPS}	Repetition
1	π/2 BPSK	1/2	1	2

MCS Index	Modulation	Coding Rate	N_{CBPS}	Repetition
2		1/2		1
3		5/8		
4		3/4		
5		13/16		
6	π/2 QPSK	1/2	2	
7		5/8		
8		3/4		
9		13/16		
10	π/2 16QAM	1/2	4	
11		5/8		
12		3/4		

N_{CBPS} is the number of coded bits per symbol.

- Modulation and coding schemes for OFDM modulation

MCS Index	Modulation	Coding Rate	N_{BPSC}	N_{CBPS}	N_{DBPS}
13	SQPSK	1/2	1	336	168
14		5/8			210
15	QPSK	1/2	2	672	336
16		5/8			420
17		3/4			504
18	16QAM	1/2	4	1344	672
19		5/8			840
20		3/4			1008
21		13/16			1092
22	64QAM	5/8	6	2016	1260
23		3/4			1512
24		13/16			1638

MCS Index	Modulation	Coding Rate	N_{BPSC}	N_{CBPS}	N_{DBPS}
N_{BPSC} is the number of coded bits per single carrier.					
N_{CBPS} is the number of coded bits per symbol.					
N_{DBPS} is the number of data bits per symbol.					

Data Types: `double`

`TrainingLength` — Number of training fields
`0` (default) | integer from 0 to 64

Number of training fields, specified as an integer from 0 to 64. `TrainingLength` must be a multiple of four.

Data Types: `double`

`PacketType` — Packet training field type
`'TRN-R'` (default) | `'TRN-T'`

Packet training field type, specified as `'TRN-R'` or `'TRN-T'`. This property applies when `TrainingLength` > `0`.

`'TRN-R'` indicates that the packet includes or requests receive-training subfields and `'TRN-T'` indicates that the packet includes transmit-training subfields.

Data Types: `char` | `string`

`BeamTrackingRequest` — Request beam tracking
`false` (default) | `true`

Request beam tracking, specified as a logical. Setting `BeamTrackingRequest` to `true` indicates that beam tracking is requested. This property applies when `TrainingLength` > `0`.

Data Types: `logical`

`TonePairingType` — Tone pairing type
`'Static'` (default) | `'Dynamic'`

Tone pairing type, specified as `'Static'` or `'Dynamic'`. This property applies when `MCS` is from 13 to 17. Specifically, `TonePairingType` applies when using OFDM and either SQPSK or QPSK modulation.

Data Types: `char` | `string`

`DTPGroupPairIndex` — DTP group pair index
42-by-1 integer vector

DTP group pair index, specified as a 42-by-1 integer vector for each pair. Element values must be from 0 to 41, with no duplicates. This property applies when `MCS` is from 13 to 17 and when `TonePairingType` is `'Dynamic'`.

Data Types: `double`

`DTPIndicator` — DTP update indicator
`false` (default) | `true`

DTP update indicator, specified as a logical. Toggle `DTPIndicator` between packets to indicate that the dynamic tone pair mapping has been updated. This property applies when `MCS` is from 13 to 17 and when `TonePairingType` is `'Dynamic'`.

Data Types: `logical`

`PSDULength` — Number of bytes carried in the user payload
`1000` (default) | integer from 1 to 262,143

Number of bytes carried in the user payload, specified as an integer from 1 to 262,143.

Data Types: `double`

`ScramblerInitialization` — Initial scrambler state
`2` (default) | integer from 1 to 127

Initial scrambler state of the data scrambler for each packet generated, specified as an integer from 1 to 127. When `MCS` is `0`, the initial scrambler state is limited to values from 1 to 15. The default value of 2 is the example state given in IEEE Std 802.11-2012, Amendment 3, Section L.5.2.

Data Types: `double` | `int8`

`AggregatedMPDU` — MPDU aggregation indicator
`false` (default) | `true`

MPDU aggregation indicator, specified as a logical. Setting `AggregatedMPDU` to `true` indicates that the current packet uses A-MPDU aggregation.

Data Types: `logical`

`LastRSSI` — Received power level of the last packet
`0` (default) | integer from 0 to 15

Received power level of the last packet, specified as an integer from 0 to 15.

When transmitting a response frame immediately following a short interframe space (SIFS) period, a DMG STA sets the `LastRSSI` as specified in IEEE 802.11ad™-2012, Section 9.3.2.3.3, to map to the *TXVECTOR* parameter *LAST_RSSI* of the response frame to the power that was measured on the received packet, as reported in the RCPI field of the frame that elicited the response frame. The encoding of the value for *TXVECTOR* is as follows:

- Power values equal to or above –42 dBm are represented as the value 15.
- Power values between –68 dBm and –42 dBm are represented as round((power – (–71 dBm))/2).
- Power values less than or equal to –68 dBm are represented as the value of 1.
- For all other cases, the DMG STA shall set the TXVECTOR parameter LAST_RSSI of the transmitted frame to 0.

The *LAST_RSSI* parameter in *RXVECTOR* maps to `LastRSSI` and indicates the value of the *LAST_RSSI* field from the PCLP header of the received packet. The encoding of the value for *RXVECTOR* is as follows:

- A value of 15 represents power greater than or equal to –42 dBm.
- Values from 2 to 14 represent power levels (–71+*`value`*×2) dBm.
- A value of 1 represents power less than or equal to –68 dBm.
- A value of 0 indicates that the previous packet was not received during the SIFS period before the current transmission.

For more information, see IEEE 802.11ad-2012, Section 21.2.

Data Types: `double`

`Turnaround` — Turnaround indication
`false` (default) | `true`

Turnaround indication, specified as a logical. Setting `Turnaround` to `true` indicates that the STA is required to listen for an incoming PPDU immediately following the

transmission of the PPDU. For more information, see IEEE 802.11ad-2012, Section 9.3.2.3.3.

Data Types: `logical`

Output Arguments

`cfgDMG` — DMG PPDU configuration
`wlanDMGConfig` object

DMG "PPDU" on page 1-52 configuration, returned as a `wlanDMGConfig` object. The properties of `cfgDMG` are described in wlanDMGConfig.

Definitions

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 802.11ad™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanDMGConfig.phyType` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanS1GConfig` | `wlanVHTConfig` | `wlanWaveformGenerator`

Topics

"Packet Size and Duration Dependencies"

Introduced in R2017a

wlanDMGConfig.phyType

Return DMG PHY modulation type

Syntax

```
type = phyType(cfg)
```

Description

`type = phyType(cfg)` returns the DMG physical layer (PHY) modulation method, based on the configuration of the DMG object.

Input Arguments

`cfg` — DMG PPDU configuration

`wlanDMGConfig` object

DMG PPDU configuration, specified as a `wlanDMGConfig` object.

Output Arguments

`type` — DMG PHY modulation type

`Control` | `SC` | `OFDM`

DMG PHY modulation type, specified as '`Control`', '`SC`', or '`OFDM`'.

Examples

Create DMG Configuration Object and Return DMG PHY Type

Create DMG configuration objects and change the default property settings by using dot notation. Use the `phyType` object function to access the DMG PHY modulation type.

Create a DMG configuration object and return the DMG PHY modulation type. By default, the configuration object creates properties to model the DMG control PHY.

```
dmg = wlanDMGConfig;
phyType(dmg)
```

```
ans =
'Control'
```

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

```
dmg.MCS = 5;
phyType(dmg)
```

```
ans =
'SC'
```

See Also

`wlanDMGConfig`

Introduced in R2017b

wlanDMGDataBitRecover

Recover data bits from DMG data field

Syntax

```
DataBits = wlanDMGDataBitRecover(rxDataSig,noiseVarEst,cfg)
DataBits = wlanDMGDataBitRecover(rxDataSig,noiseVarEst,csi,cfg)
DataBits = wlanDMGDataBitRecover(___,Name,Value)
```

Description

`DataBits = wlanDMGDataBitRecover(rxDataSig,noiseVarEst,cfg)` recovers the data bits given the data field from a DMG transmission (OFDM, single-carrier, or control PHY), the noise variance estimate, and the DMG configuration object.

`DataBits = wlanDMGDataBitRecover(rxDataSig,noiseVarEst,csi,cfg)` uses the channel state information specified in `csi` to enhance the demapping of OFDM subcarriers.

`DataBits = wlanDMGDataBitRecover(___,Name,Value)` specifies additional options in name-value pair arguments, using the inputs from preceding syntaxes. When a name-value pair is not specified, its default value is used.

Examples

Recover Data Field from DMG SC PHY

Recover data information bits from the DMG data field of single-carrier (SC) PHY.

Transmitter

Create the DMG configuration object with a modulation and coding scheme (MCS) for the SC PHY.

```
cfgDMG = wlanDMGConfig('MCS',10);
```

Create the input sequence of data bits, specifying it as a column vector with `cfgDMG.PSDULength*8` elements. Generate the DMG transmission waveform.

```
txBits = randi([0 1],cfgDMG.PSDULength*8,1,'int8');
tx = wlanWaveformGenerator(txBits,cfgDMG);
```

AWGN Channel

Set an SNR of 10 dB, calculate the noise power (noise variance), and add AWGN to the transmission waveform by using the `awgn` function.

```
SNR = 10;
nVar = 10^(-SNR/10);
rx = awgn(tx,SNR);
```

Receiver

Extract the data field by using the `wlanFieldIndices` function to generate the PPDU field indices.

```
ind = wlanFieldIndices(cfgDMG);
rxData = rx(ind.DMGData(1):ind.DMGData(2));
```

Reshape the received data waveform into blocks. Set the data block size to 512 and the guard interval length to 64. Remove the last guard interval from the received data waveform. The resulting data waveform is a 512-by-`Nblks` matrix, where `Nblks` is the number of DMG data blocks.

```
blkSize = 512;
Ngi = 64;
rxData = rxData(1:end-Ngi);
rxData = reshape(rxData,blkSize,[]);
```

Remove the guard interval from each block. The resulting signal is a 448-by-`Nblks` matrix, as expected for a time-domain DMG data field signal for SC PHY.

```
rxSym = rxData(Ngi+1:end,:);
size(rxSym)
```

```
ans =

   448     9
```

Recover the PSDU from the DMG data field.

```
rxBits = wlanDMGDataBitRecover(rxSym,nVar,cfgDMG);
```

Compare it against the original information bits.

```
disp(isequal(txBits,rxBits));
```

```
     1
```

Recover Data Field from DMG OFDM PHY

Recover data information bits of the DMG data field of the OFDM PHY.

Transmitter

Create the DMG configuration object with a modulation and coding scheme (MCS) for the OFDM PHY.

```
cfgDMG = wlanDMGConfig('MCS',14);
```

Create the input sequence of data bits, specifying it as a column vector with `cfgDMG.PSDULength*8` elements. Generate the DMG transmission waveform.

```
txBits = randi([0 1],cfgDMG.PSDULength*8,1,'int8');
tx = wlanWaveformGenerator(txBits,cfgDMG);
```

Channel

Transmit the signal through a channel with no noise (zero noise variance).

```
rx = tx;
nVar = 0;
```

Receiver

Extract the data field, using the `wlanFieldIndices` function to generate the PPDU field indices.

```
ind = wlanFieldIndices(cfgDMG);
rxData = rx(ind.DMGData(1):ind.DMGData(2));
```

Set the FFT length to 512 and the cyclic prefix length to 128 for the OFDM demodulation.

```
Nfft = 512;
Ncp = 128;
```

Perform the OFDM demodulation. Reshape the received waveform to have the OFDM symbols per column and remove cyclic prefix. Then, scale the sequence by the active tone 352 and extract the *frequency domain* symbols.

```
ofdmSym = reshape(rxData,Nfft+Ncp,[]);
dftSym = ofdmSym(Ncp+1:end,:);
dftSym = dftSym/(Nfft/sqrt(352));
freqSym = fftshift(fft(dftSym,[],1),1);
```

Extract data-carrying subcarriers and discard the pilots. Set the highest subcarrier index to 177.

```
pilotSCIndex = [-150; -130; -110; -90; -70; -50; -30; -10; 10; 30; 50; 70; 90; 110; 130
noDataSCIndex = [pilotSCIndex; [-1; 0; 1]];
Nsr = 177;
dataSCIndex = setdiff((-Nsr:Nsr).',sort(noDataSCIndex));
rxSym = freqSym(dataSCIndex+(Nfft/2+1),:);
```

Recover the PSDU from the DMG data field. Assume a CSI estimation of all ones.

```
csi = ones(length(dataSCIndex),1);
rxBits = wlanDMGDataBitRecover(rxSym,nVar,csi,cfgDMG);
```

Compare it against the original information bits.

```
disp(isequal(txBits,rxBits));
```

```
   1
```

Recover Data Field from DMG Control PHY

Recover data information bits from the DMG data field of the control PHY.

Transmitter

Create the DMG configuration object with a modulation and coding scheme (MCS) for the control PHY.

```
cfgDMG = wlanDMGConfig('MCS',0);
```

Create the input sequence of data bits, specifying it as a column vector with `cfgDMG.PSDULength*8` elements. Generate the DMG transmission waveform.

```
txBits = randi([0 1],cfgDMG.PSDULength*8,1,'int8');
tx = wlanWaveformGenerator(txBits,cfgDMG);
```

Channel

Transmit the signal through a channel with no noise (zero noise variance).

```
rx = tx;
nVar = 0;
```

Receiver

Extract the header and the data field by using the `wlanFieldIndices` function.

```
ind = wlanFieldIndices(cfgDMG);
rxSym = rx(ind.DMGHeader(1):ind.DMGData(2));
```

De-rotate the received signal by pi/2 and despread it with a spreading factor of 32. Use the `wlanGolaySequence` function to generate the Golay sequence.

```
rxSym = rxSym.*exp(-1i*pi/2*(0:size(rxSym,1)-1).');
SF = 32;
Ga = wlanGolaySequence(SF);
rxSymDespread = (reshape(rxSym,SF,length(rxSym)/SF)'*Ga)/SF;
```

Recover the PSDU from the DMG data field.

```
rxBits = wlanDMGDataBitRecover(rxSymDespread,nVar,cfgDMG);
```

Compare it against the original information bits.

```
disp(isequal(txBits,rxBits));
```

```
   1
```

Input Arguments

`rxDataSig` — Received DMG data field signal
real or complex matrix

Received DMG data signal, specified as a real or complex matrix. The contents and size of `rxDataSig` depend on the physical layer (PHY):

- Single-carrier PHY — `rxDataSig` is the time-domain DMG data field signal, specified as a 448-by-N_{BLKS} matrix of real or complex values. The value 448 is the number of symbols in a DMG data symbol and N_{BLKS} is the number of DMG data blocks.
- OFDM PHY — `rxDataSig` is the demodulated DMG data field OFDM symbols, specified as a 336-by-N_{SYM} matrix of real or complex values. The value 336 is the number of data subcarriers in the DMG data field and N_{SYM} is the number of OFDM symbols.
- Control PHY — `rxDataSig` is the time-domain signal containing the header and data fields, specified as an N_{B}-by-1 column vector of real or complex values, where N_{B} is the number of despread symbols.

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` — DMG PPDU configuration
`wlanDMGConfig` object

DMG PPDU configuration, specified as a `wlanDMGConfig` object. The `wlanDMGDataBitRecover` function uses the following object properties:

`MCS` — Modulation and coding scheme index
`0` (default) | integer from 0 to 24

Modulation and coding scheme index, specified as an integer from 0 to 24. The `MCS` index indicates the modulation and coding scheme used in transmitting the current packet.

- Modulation and coding scheme for control PHY

MCS Index	Modulation	Coding Rate	Comment
0	DBPSK	1/2	Code rate and data rate might be lower due to codeword shortening.

- Modulation and coding schemes for single-carrier modulation

MCS Index	Modulation	Coding Rate	N_{CBPS}	Repetition
1	π/2 BPSK	1/2	1	2
2		1/2		1
3		5/8		
4		3/4		
5		13/16		
6	π/2 QPSK	1/2	2	
7		5/8		
8		3/4		
9		13/16		
10	π/2 16QAM	1/2	4	
11		5/8		
12		3/4		
N_{CBPS} is the number of coded bits per symbol.				

- Modulation and coding schemes for OFDM modulation

MCS Index	Modulation	Coding Rate	N_{BPSC}	N_{CBPS}	N_{DBPS}
13	SQPSK	1/2	1	336	168
14		5/8			210
15	QPSK	1/2	2	672	336
16		5/8			420
17		3/4			504
18	16QAM	1/2	4	1344	672
19		5/8			840
20		3/4			1008
21		13/16			1092
22	64QAM	5/8	6	2016	1260
23		3/4			1512

MCS Index	Modulation	Coding Rate	N_{BPSC}	N_{CBPS}	N_{DBPS}
24		13/16			1638
N_{BPSC} is the number of coded bits per single carrier. N_{CBPS} is the number of coded bits per symbol. N_{DBPS} is the number of data bits per symbol.					

Data Types: `double`

`TrainingLength` — **Number of training fields**
`0` (default) | integer from 0 to 64

Number of training fields, specified as an integer from 0 to 64. `TrainingLength` must be a multiple of four.

Data Types: `double`

`PacketType` — **Packet training field type**
`'TRN-R'` (default) | `'TRN-T'`

Packet training field type, specified as `'TRN-R'` or `'TRN-T'`. This property applies when `TrainingLength` > `0`.

`'TRN-R'` indicates that the packet includes or requests receive-training subfields and `'TRN-T'` indicates that the packet includes transmit-training subfields.

Data Types: `char` | `string`

`BeamTrackingRequest` — **Request beam tracking**
`false` (default) | `true`

Request beam tracking, specified as a logical. Setting `BeamTrackingRequest` to `true` indicates that beam tracking is requested. This property applies when `TrainingLength` > `0`.

Data Types: `logical`

`TonePairingType` — **Tone pairing type**
`'Static'` (default) | `'Dynamic'`

Tone pairing type, specified as `'Static'` or `'Dynamic'`. This property applies when `MCS` is from 13 to 17. Specifically, `TonePairingType` applies when using OFDM and either SQPSK or QPSK modulation.

Data Types: `char` | `string`

`DTPGroupPairIndex` — DTP group pair index
42-by-1 integer vector

DTP group pair index, specified as a 42-by-1 integer vector for each pair. Element values must be from 0 to 41, with no duplicates. This property applies when `MCS` is from 13 to 17 and when `TonePairingType` is `'Dynamic'`.

Data Types: `double`

`DTPIndicator` — DTP update indicator
`false` (default) | `true`

DTP update indicator, specified as a logical. Toggle `DTPIndicator` between packets to indicate that the dynamic tone pair mapping has been updated. This property applies when `MCS` is from 13 to 17 and when `TonePairingType` is `'Dynamic'`.

Data Types: `logical`

`PSDULength` — Number of bytes carried in the user payload
`1000` (default) | integer from 1 to 262,143

Number of bytes carried in the user payload, specified as an integer from 1 to 262,143.

Data Types: `double`

`ScramblerInitialization` — Initial scrambler state
`2` (default) | integer from 1 to 127

Initial scrambler state of the data scrambler for each packet generated, specified as an integer from 1 to 127. When `MCS` is `0`, the initial scrambler state is limited to values from 1 to 15. The default value of 2 is the example state given in IEEE Std 802.11-2012, Amendment 3, Section L.5.2.

Data Types: `double` | `int8`

`AggregatedMPDU` — MPDU aggregation indicator
`false` (default) | `true`

MPDU aggregation indicator, specified as a logical. Setting `AggregatedMPDU` to `true` indicates that the current packet uses A-MPDU aggregation.

Data Types: `logical`

`LastRSSI` — Received power level of the last packet
`0` (default) | integer from 0 to 15

Received power level of the last packet, specified as an integer from 0 to 15.

When transmitting a response frame immediately following a short interframe space (SIFS) period, a DMG STA sets the `LastRSSI` as specified in IEEE 802.11ad-2012, Section 9.3.2.3.3, to map to the *TXVECTOR* parameter *LAST_RSSI* of the response frame to the power that was measured on the received packet, as reported in the RCPI field of the frame that elicited the response frame. The encoding of the value for *TXVECTOR* is as follows:

- Power values equal to or above –42 dBm are represented as the value 15.
- Power values between –68 dBm and –42 dBm are represented as round((power – (–71 dBm))/2).
- Power values less than or equal to –68 dBm are represented as the value of 1.
- For all other cases, the DMG STA shall set the TXVECTOR parameter LAST_RSSI of the transmitted frame to 0.

The *LAST_RSSI* parameter in *RXVECTOR* maps to `LastRSSI` and indicates the value of the *LAST_RSSI* field from the PCLP header of the received packet. The encoding of the value for *RXVECTOR* is as follows:

- A value of 15 represents power greater than or equal to –42 dBm.
- Values from 2 to 14 represent power levels (–71+*`value`*×2) dBm.
- A value of 1 represents power less than or equal to –68 dBm.
- A value of 0 indicates that the previous packet was not received during the SIFS period before the current transmission.

For more information, see IEEE 802.11ad-2012, Section 21.2.

Data Types: `double`

`Turnaround` — Turnaround indication
`false` (default) | `true`

Turnaround indication, specified as a logical. Setting `Turnaround` to `true` indicates that the STA is required to listen for an incoming PPDU immediately following the transmission of the PPDU. For more information, see IEEE 802.11ad-2012, Section 9.3.2.3.3.

Data Types: `logical`

`csi` — Channel State Information
real column vector

Channel state information, specified as a 336-by-1 real column vector. The value 336 specifies the number of data subcarriers in the DMG data field. `csi` is required only for OFDM PHY.

Data Types: `double`

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: `'MaximumLDPCIterationCount','12','EarlyTermination','false'` specifies a maximum of 12 decoding iterations for the LDPC and disables early termination of LDPC decoding so that it completes the 12 iterations.

`MaximumLDPCIterationCount` — Maximum number of LDPC decoding iterations
`12` | positive scalar integer

Maximum number of LDPC decoding iterations, specified as the comma-separated pair consisting of '`MaximumLDPCIterationCount`' and a positive integer.

Data Types: `double`

`EarlyTermination` — Enable early termination of LDPC decoding
`false` (default) | `true`

Enable early termination of LDPC decoding, specified as the comma-separated pair consisting of '`EarlyTermination`' and a logical.

- When set to `false` — LDPC decoding completes the number of iterations specified by `MaximumLDPCIterationCount`, regardless of parity check status.
- When set to `true` — LDPC decoding terminates when all parity checks are satisfied.

Output Arguments

`DataBits` — Recovered information bits in the DMG data field

1 | 0 | column vector

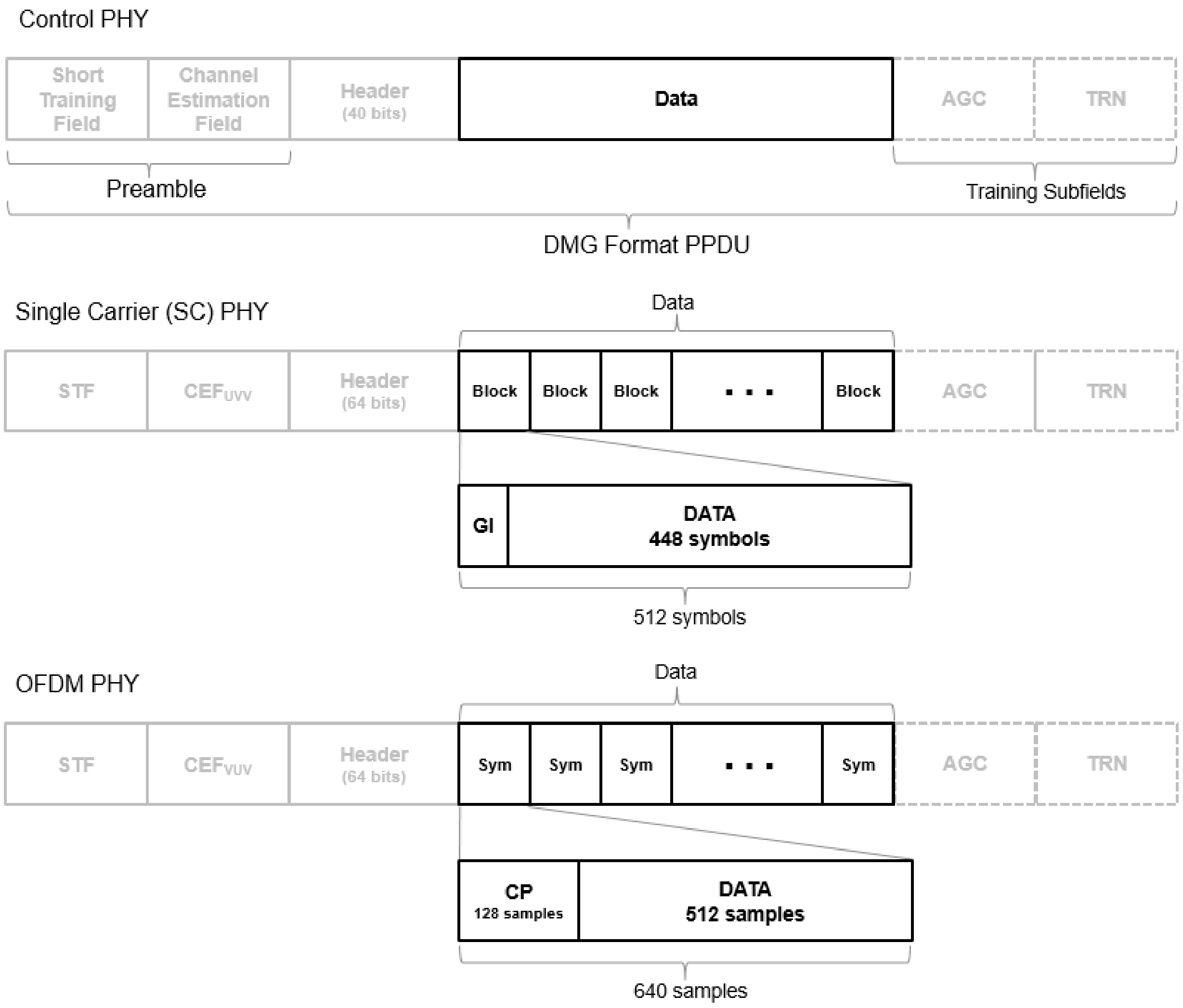
Recovered information bits from the DMG data field, returned as a column vector of length 8 × `cfgDMG.PSDULength`. See `wlanDMGConfig` for `PSDULength` details.

Data Types: `int8`

Definitions

DMG Data Field

The DMG format supports three physical layer (PHY) modulation schemes: control, single carrier, and OFDM. The data field is variable in length. It serves the same function for the three PHYs and carries the user data payload.



For SC PHY, each block in the data field is 512-symbols long and with a guard interval (GI) of 64 symbols with the Golay Sequence. For OFDM, each OFDM symbol in the data field are 640 samples long and with a cyclic prefix (CP) of 128 samples to prevent intersymbol interference.

IEEE 802.11ad-2012 specifies the common aspects of the DMG PPDU packet structure in Section 21.3. The PHY modulation-specific aspects of the data field structure are specified in these sections:

- The DMG control PHY packet structure is specified in Section 21.4.
- The DMG OFDM PHY packet structure is specified in Section 21.5.
- The DMG SC PHY packet structure is specified in Section 21.6.

References

[1] IEEE Std 802.11ad™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanDMGConfig` | `wlanDMGHeaderBitRecover`

Introduced in R2017b

wlanDMGHeaderBitRecover

Recover header bits from DMG header field

Syntax

```
[headerBits,failHCS] = wlanDMGHeaderBitRecover(rxHeader,noiseVarEst,
cfg)
headerBits = wlanDMGHeaderBitRecover(rxHeader,noiseVarEst,csi,cfg)
headerBits = wlanDMGHeaderBitRecover( ___ ,Name,Value)
```

Description

`[headerBits,failHCS] = wlanDMGHeaderBitRecover(rxHeader,noiseVarEst,cfg)` recovers the header information bits and tests the header check sequence (HCS) given the header field from a DMG transmission (OFDM, single-carrier, or control PHY), the noise variance estimate, and the DMG configuration object.

`headerBits = wlanDMGHeaderBitRecover(rxHeader,noiseVarEst,csi,cfg)` uses the channel state information specified in `csi` to enhance the demapping of OFDM subcarriers.

`headerBits = wlanDMGHeaderBitRecover( ___ ,Name,Value)` specifies additional options using name-value pair arguments, using the inputs from preceding syntaxes. When a name-value pair is not specified, its default value is used.

Examples

Recover Header Field from DMG SC PHY

Recover header bits from the DMG header field of the single-carrier (SC) PHY.

Transmitter

Create the DMG configuration object with a modulation and coding scheme (MCS) for the SC PHY.

```
cfgDMG = wlanDMGConfig('MCS',10);
```

Create the input sequence of bits, specifying it as a column vector with `cfgDMG.PSDULength*8` elements. Generate the DMG transmission waveform.

```
txBits = randi([0 1],cfgDMG.PSDULength*8,1,'int8');
tx = wlanWaveformGenerator(txBits,cfgDMG);
```

AWGN Channel

Set an SNR of 10 dB, calculate the noise power (noise variance), and add AWGN to the transmission waveform by using the `awgn` function.

```
SNR = 10;
nVar = 10^(-SNR/10);
rx = awgn(tx,SNR);
```

Receiver

Extract the header field by using the `wlanFieldIndices` function.

```
ind = wlanFieldIndices(cfgDMG);
rxHeader = rx(ind.DMGHeader(1):ind.DMGHeader(2));
```

Reshape the received waveform into blocks. Set the data block size to 512 and the guard interval length to 64. Remove the last guard interval from the received header waveform. The resulting signal is a 448-by-2 matrix.

```
blkSize = 512;
rxHeader = reshape(rxHeader,blkSize,[]);
Ngi = 64;
rxSym = rxHeader(Ngi+1:end,:);
size(rxSym)
```

```
ans =

   448     2
```

Recover header bits from DMG header field.

```
[rxBits,failHCS] = wlanDMGHeaderBitRecover(rxSym,nVar,cfgDMG);
```

Display the HCS check on the recovered header bits.

```
disp(failHCS);
```

```
   0
```

Recover Header Field from DMG OFDM PHY

Recover header information bits from the DMG header field of the OFDM PHY.

Transmitter

Create the DMG configuration object with a modulation and coding scheme (MCS) for the OFDM PHY.

```
cfgDMG = wlanDMGConfig('MCS',14);
```

Create the input sequence of data bits, specifying it as a column vector with `cfgDMG.PSDULength*8` elements. Generate the DMG transmission waveform.

```
txBits = randi([0 1],cfgDMG.PSDULength*8,1,'int8');
tx = wlanWaveformGenerator(txBits,cfgDMG);
```

Channel

Transmit the signal through a channel with no noise (zero noise variance).

```
rx = tx;
nVar = 0;
```

Receiver

Extract data field using the `wlanFieldIndices` function.

```
ind = wlanFieldIndices(cfgDMG);
rxHeader = rx(ind.DMGHeader(1):ind.DMGHeader(2));
```

Set the FFT length to 512 and the cyclic prefix length to 128 for the OFDM demodulation.

```
Nfft = 512;
Ncp = 128;
```

Perform the OFDM demodulation. Remove cyclic prefix, scale the sequence by the active tone 352, and extract the *frequency domain* symbols.

```
dftSym = rxHeader(Ncp+1:end,:);
dftSym = dftSym/(Nfft/sqrt(352));
freqSym = fftshift(fft(dftSym,[],1),1);
```

Extract data-carrying subcarriers and discard the pilots. Set the highest subcarrier index to 177.

```
pilotSCIndex = [-150; -130; -110; -90; -70; -50; -30; -10; 10; 30; 50; 70; 90; 110; 130
noDataSCIndex = [pilotSCIndex; [-1; 0; 1]];
Nsr = 177;
dataSCIndex = setdiff((-Nsr:Nsr).',sort(noDataSCIndex));
rxSym = freqSym(dataSCIndex+(Nfft/2+1),:);
```

Recover the header bits from the DMG header field. Assume a CSI estimation of all ones.

```
csi = ones(length(dataSCIndex),1);
[rxBits,failHCS] = wlanDMGHeaderBitRecover(rxSym,nVar,csi,cfgDMG);
```

Display the HCS check on the recovered header bits.

```
disp(failHCS);
```

```
   0
```

Recover Header Field from DMG Control PHY

Recover header information bits of the DMG header field from the control PHY.

Transmitter

Create the DMG configuration object with a modulation and coding scheme (MCS) for the control PHY.

```
cfgDMG = wlanDMGConfig('MCS',0);
```

Create the input sequence of data bits, specifying it as a column vector with `cfgDMG.PSDULength*8` elements. Generate the DMG transmission waveform.

```
txBits = randi([0 1],cfgDMG.PSDULength*8,1,'int8');
tx = wlanWaveformGenerator(txBits,cfgDMG);
```

Channel

Transmit the signal through a channel with no noise (zero noise variance).

```
rx = tx;
nVar = 0;
```

Receiver

Extract the header field by using the `wlanFieldIndices` function.

```
ind = wlanFieldIndices(cfgDMG);
rxHeader = rx(ind.DMGHeader(1):ind.DMGHeader(2));
```

De-rotate the received signal by pi/2 and despread it with a spreading factor of 32. Use the `wlanGolaySequence` function to generate the Golay sequence.

```
rxSym = rxHeader.*exp(-1i*pi/2*(0:size(rxHeader,1)-1).');
SF = 32;
Ga = wlanGolaySequence(SF);
rxDespread = reshape(rxSym,SF,length(rxSym)/SF)'*Ga/SF;
```

Recover the header bits from the DMG header field.

```
[rxBits,failHCS] = wlanDMGHeaderBitRecover(rxDespread,nVar,cfgDMG);
```

Display the HCS check on the recovered header bits.

```
disp(failHCS);
```

```
   0
```

Input Arguments

`rxHeader` — Received DMG header field signal
matrix

Received DMG header field signal, specified as a real or complex matrix. The contents and size of `rxHeader` depends on the physical layer (PHY):

- Single-Carrier PHY — `rxHeader` is the time-domain DMG header field signal, specified as a 448-by-N_{BLKS} matrix of real or complex values. The value 448 is the number of symbols in a DMG header symbol and N_{BLKS} is the number of DMG header blocks.

- OFDM PHY — `rxHeader` is the frequency-domain signal, specified as a 336-by-1 column vector of real or complex values. The value 336 is the number of data subcarriers in the DMG header field.
- Control PHY — `rxHeader` is the time-domain signal containing the header field, specified as an N_B-by-1 column vector of real or complex values. N_B is the number of despread symbols.

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` — DMG PPDU configuration

`wlanDMGConfig` object

DMG PPDU configuration, specified as a `wlanDMGConfig` object. The `wlanDMGDataBitRecover` function uses the following object properties:

`MCS` — Modulation and coding scheme index

`0` (default) | integer from 0 to 24

Modulation and coding scheme index, specified as an integer from 0 to 24. The `MCS` index indicates the modulation and coding scheme used in transmitting the current packet.

- Modulation and coding scheme for control PHY

MCS Index	Modulation	Coding Rate	Comment
0	DBPSK	1/2	Code rate and data rate might be lower due to codeword shortening.

- Modulation and coding schemes for single-carrier modulation

MCS Index	Modulation	Coding Rate	N_{CBPS}	Repetition
1	π/2 BPSK	1/2	1	2
2		1/2		1
3		5/8		
4		3/4		
5		13/16		
6	π/2 QPSK	1/2	2	
7		5/8		
8		3/4		
9		13/16		
10	π/2 16QAM	1/2	4	
11		5/8		
12		3/4		
N_{CBPS} is the number of coded bits per symbol.				

- Modulation and coding schemes for OFDM modulation

MCS Index	Modulation	Coding Rate	N_{BPSC}	N_{CBPS}	N_{DBPS}
13	SQPSK	1/2	1	336	168
14		5/8			210
15	QPSK	1/2	2	672	336
16		5/8			420
17		3/4			504
18	16QAM	1/2	4	1344	672
19		5/8			840
20		3/4			1008
21		13/16			1092
22	64QAM	5/8	6	2016	1260
23		3/4			1512
24		13/16			1638

MCS Index	Modulation	Coding Rate	N_{BPSC}	N_{CBPS}	N_{DBPS}
N_{BPSC} is the number of coded bits per single carrier. N_{CBPS} is the number of coded bits per symbol. N_{DBPS} is the number of data bits per symbol.					

Data Types: `double`

`TrainingLength` — Number of training fields
`0` (default) | integer from 0 to 64

Number of training fields, specified as an integer from 0 to 64. `TrainingLength` must be a multiple of four.

Data Types: `double`

`PacketType` — Packet training field type
`'TRN-R'` (default) | `'TRN-T'`

Packet training field type, specified as `'TRN-R'` or `'TRN-T'`. This property applies when `TrainingLength` > `0`.

`'TRN-R'` indicates that the packet includes or requests receive-training subfields and `'TRN-T'` indicates that the packet includes transmit-training subfields.

Data Types: `char` | `string`

`BeamTrackingRequest` — Request beam tracking
`false` (default) | `true`

Request beam tracking, specified as a logical. Setting `BeamTrackingRequest` to `true` indicates that beam tracking is requested. This property applies when `TrainingLength` > `0`.

Data Types: `logical`

`TonePairingType` — Tone pairing type
`'Static'` (default) | `'Dynamic'`

Tone pairing type, specified as `'Static'` or `'Dynamic'`. This property applies when `MCS` is from 13 to 17. Specifically, `TonePairingType` applies when using OFDM and either SQPSK or QPSK modulation.

Data Types: `char` | `string`

`DTPGroupPairIndex` — DTP group pair index
42-by-1 integer vector

DTP group pair index, specified as a 42-by-1 integer vector for each pair. Element values must be from 0 to 41, with no duplicates. This property applies when `MCS` is from 13 to 17 and when `TonePairingType` is `'Dynamic'`.

Data Types: `double`

`DTPIndicator` — DTP update indicator
`false` (default) | `true`

DTP update indicator, specified as a logical. Toggle `DTPIndicator` between packets to indicate that the dynamic tone pair mapping has been updated. This property applies when `MCS` is from 13 to 17 and when `TonePairingType` is `'Dynamic'`.

Data Types: `logical`

`PSDULength` — Number of bytes carried in the user payload
`1000` (default) | integer from 1 to 262,143

Number of bytes carried in the user payload, specified as an integer from 1 to 262,143.

Data Types: `double`

`ScramblerInitialization` — Initial scrambler state
`2` (default) | integer from 1 to 127

Initial scrambler state of the data scrambler for each packet generated, specified as an integer from 1 to 127. When `MCS` is `0`, the initial scrambler state is limited to values from 1 to 15. The default value of 2 is the example state given in IEEE Std 802.11-2012, Amendment 3, Section L.5.2.

Data Types: `double` | `int8`

`AggregatedMPDU` — MPDU aggregation indicator
`false` (default) | `true`

MPDU aggregation indicator, specified as a logical. Setting `AggregatedMPDU` to `true` indicates that the current packet uses A-MPDU aggregation.

Data Types: `logical`

`LastRSSI` — Received power level of the last packet
`0` (default) | integer from 0 to 15

Received power level of the last packet, specified as an integer from 0 to 15.

When transmitting a response frame immediately following a short interframe space (SIFS) period, a DMG STA sets the `LastRSSI` as specified in IEEE 802.11ad-2012, Section 9.3.2.3.3, to map to the *TXVECTOR* parameter *LAST_RSSI* of the response frame to the power that was measured on the received packet, as reported in the RCPI field of the frame that elicited the response frame. The encoding of the value for *TXVECTOR* is as follows:

- Power values equal to or above –42 dBm are represented as the value 15.
- Power values between –68 dBm and –42 dBm are represented as round((power – (–71 dBm))/2).
- Power values less than or equal to –68 dBm are represented as the value of 1.
- For all other cases, the DMG STA shall set the TXVECTOR parameter LAST_RSSI of the transmitted frame to 0.

The *LAST_RSSI* parameter in *RXVECTOR* maps to `LastRSSI` and indicates the value of the *LAST_RSSI* field from the PCLP header of the received packet. The encoding of the value for *RXVECTOR* is as follows:

- A value of 15 represents power greater than or equal to –42 dBm.
- Values from 2 to 14 represent power levels (–71+*`value`*×2) dBm.
- A value of 1 represents power less than or equal to –68 dBm.
- A value of 0 indicates that the previous packet was not received during the SIFS period before the current transmission.

For more information, see IEEE 802.11ad-2012, Section 21.2.

Data Types: `double`

`Turnaround` — Turnaround indication
`false` (default) | `true`

Turnaround indication, specified as a logical. Setting `Turnaround` to `true` indicates that the STA is required to listen for an incoming PPDU immediately following the

transmission of the PPDU. For more information, see IEEE 802.11ad-2012, Section 9.3.2.3.3.

Data Types: `logical`

`csi` — Channel State Information
real column vector

Channel state information, specified as a 336-by-1 real column vector. The value 336 specifies the number of data subcarriers in the DMG data field. `csi` is required only for OFDM PHY.

Data Types: `double`

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: `'MaximumLDPCIterationCount','12','EarlyTermination','false'` specifies a maximum of 12 decoding iterations for the LDPC and disables early termination of LDPC decoding so that it completes the 12 iterations.

`MaximumLDPCIterationCount` — Maximum number of LDPC decoding iterations
`12` | positive scalar integer

Maximum number of LDPC decoding iterations, specified as the comma-separated pair consisting of '`MaximumLDPCIterationCount`' and a positive integer.

Data Types: `double`

`EarlyTermination` — Enable early termination of LDPC decoding
`false` (default) | `true`

Enable early termination of LDPC decoding, specified as the comma-separated pair consisting of '`EarlyTermination`' and a logical.

- When set to `false` — LDPC decoding completes the number of iterations specified by `MaximumLDPCIterationCount`, regardless of parity check status.

- When set to `true` — LDPC decoding terminates when all parity checks are satisfied.

Output Arguments

`headerBits` — Recovered header information bits
1 | 0 | column vector

Recovered header information bits, returned as a column vector of 64 elements for OFDM and single-carrier PHYs and a column vector of 40 elements for control PHYs.

Data Types: `int8`

`failHCS` — HCS check
`false` | `true`

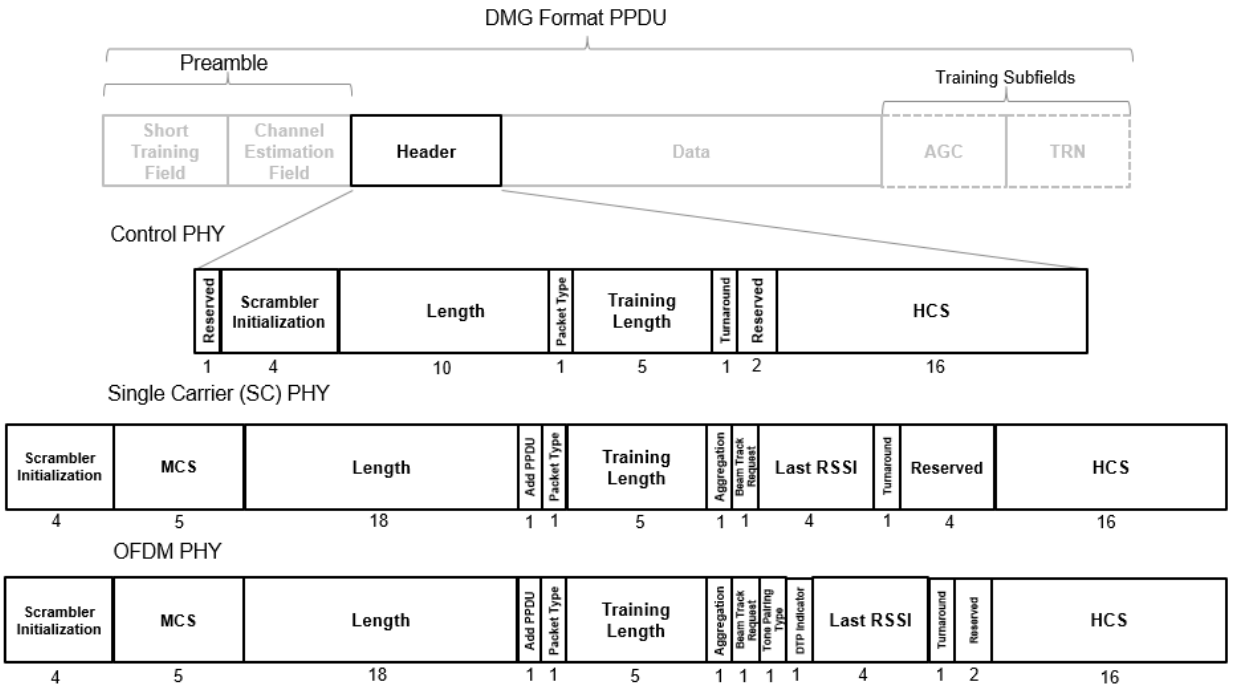
HCS check, returned as a logical. When `headerBits` fails the HCS check, `failHCS` is `true`.

Data Types: `logical`

Definitions

DMG Header Field

In the DMG format, the header field is different in size and content for every supported physical layer (PHY) modulation scheme. This field contains additional important information for the receiver.



The total size of the header field is 40 bits for control PHYs and 64 bits for SC and OFDM PHYs.

The most important fields common for the three PHY modes are:

- *Scrambler initialization* — Specifies the initial state for the scrambler.
- *MCS* — Specifies the modulation and coding scheme used in the data field. It is not present in control PHY.
- *Length (data)* — Specifies the length of the data field.
- *Packet type* — Specifies whether the beamforming training field is intended for the receiver or the transmitter.
- *Training length* — Specifies whether a beamforming training field is used and if so, its length.
- *HCS* — Provides a checksum per CRC for the header.

IEEE 802.11ad-2012 specifies the detailed aspects of the DMG header field structure. In particular, the PHY modulation-specific aspects of the header field are specified in these sections:

- The DMG control PHY header structure is specified in Section 21.4.3.2.
- The DMG OFDM PHY header structure is specified in Section 21.5.3.1.
- The DMG SC PHY header structure is specified in Section 21.6.3.1.

References

[1] IEEE Std 802.11ad™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanDMGConfig` | `wlanDMGDataBitRecover`

Introduced in R2017b

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 1-90[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 configuration object.

```
nht = wlanNonHTConfig;
```

Generate a non-HT waveform.

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

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

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

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

```
tgacChan = 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 = tgacChan(txltf);
rxltf = noise(rxltfNoNoise);
```

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

```
pfoffset = comm.PhaseFrequencyOffset('SampleRate',fs,'FrequencyOffsetSource','Input por
rxltf = pfoffset(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 = pfoffset(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 format configuration object.

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

Generate the transmit waveform.

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

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

```
tgnChan = 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 = tgnChan(txSig);
rxSig = noise(rxSigNoNoise);
```

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

```
pfOffset = comm.PhaseFrequencyOffset('SampleRate',fs,'FrequencyOffsetSource','Input por
rxSig = pfOffset(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 =

   2.0003e+03
```

Correct for the estimated offset.

```
rxSigCorr1 = pfOffset(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 =

    6.5375
```

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

Correct for the remaining offset.

```
rxSigCorr2 = pfOffset(rxSigCorr1,-foffset2);
```

Determine the frequency offset of the twice corrected signal.

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

```
deltaFreq =

  -1.6112e-13
```

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`

`cbw` — Channel bandwidth
`'CBW5'` | `'CBW10'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW5'`, `'CBW10'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Data Types: `char` | `string`

`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 duration of the short training symbol varies with bandwidth. For more information, see "L-LTF" on page 1-90.

Data Types: `double`

Output Arguments

`fOffset` — Frequency offset
real scalar

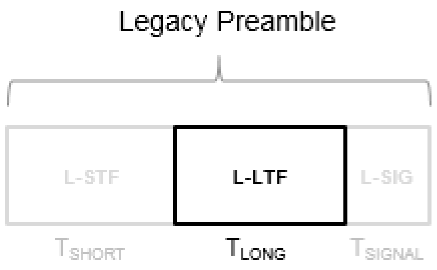
Frequency offset in Hz, returned as a real scalar.

Data Types: `double`

Definitions

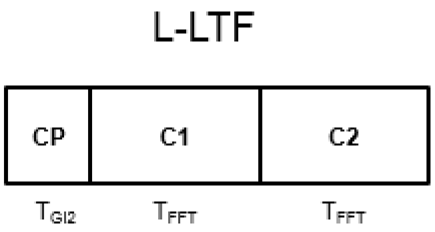
L-LTF

The legacy long training field (L-LTF) is the second field in the 802.11 OFDM PLCP legacy preamble. The L-LTF is a component of VHT, HT, and non-HT PPDUs.



Channel estimation, fine frequency offset estimation, and fine symbol timing offset estimation rely on the L-LTF.

The L-LTF is composed of a cyclic prefix (CP) followed by two identical long training symbols (C1 and C2). The CP consists of the second half of the long training symbol.



The L-LTF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration ($T_{GI2} = T_{FFT} / 2$)	L-LTF Duration ($T_{LONG} = T_{GI2} + 2 \times T_{FFT}$)
20, 40, 80, and 160	312.5	3.2 μs	1.6 μs	8 μs
10	156.25	6.4 μs	3.2 μs	16 μs
5	78.125	12.8 μs	6.4 μs	32 μs

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

[3] 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.

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

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-103[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 and performs frequency smoothing over the specified filter span. For more information, see "Frequency Smoothing" on page 1-105.

This syntax supports input options from prior syntaxes.

Examples

Estimate SISO Channel Using L-LTF

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

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

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

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.

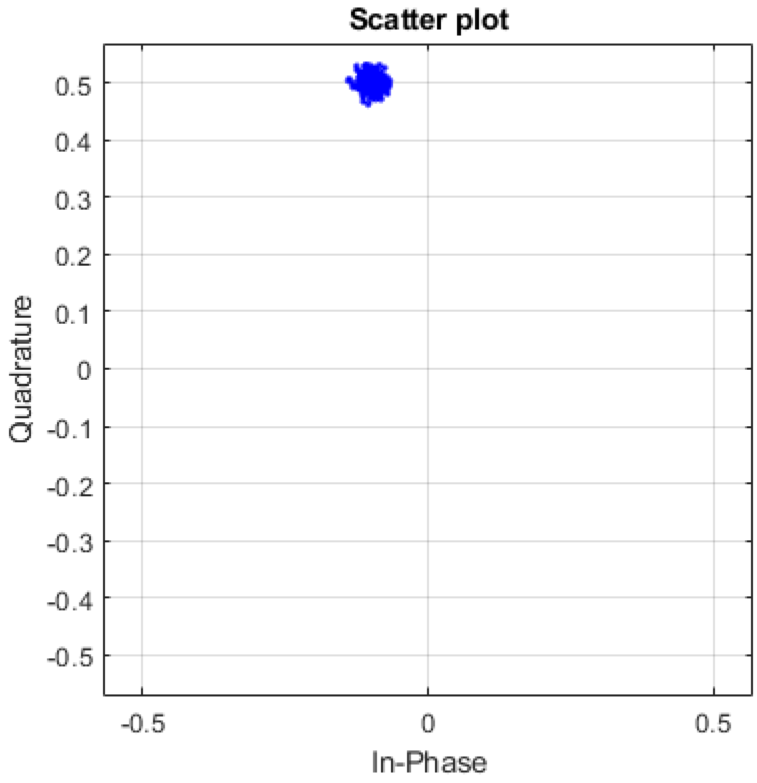
```
rxWaveform = awgn(txWaveform*(-0.1+0.5i),30);
```

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.

L-LTF Channel Estimation After TGn Channel

Generate a time domain waveform for an 802.11n HT packet, pass it through a TGn fading channel and perform L-LTF channel estimation. Trailing zeros are added to the waveform to allow for TGn channel delay.

Create the HT packet configuration and transmit waveform.

```
cfgHT = wlanHTConfig;
txWaveform = wlanWaveformGenerator([1;0;0;1],cfgHT);
```

Configure a TGn channel with 20 MHz bandwidth.

```
tgnChannel = wlanTGnChannel;
tgnChannel.SampleRate = 20e6;
```

Pass the waveform through the TGn channel, adding trailing zeros to allow for channel delay.

```
rxWaveform = tgnChannel([txWaveform; zeros(15,1)]);
```

Skip the first four samples to synchronize the received waveform for channel delay.

```
rxWaveform = rxWaveform(5:end,:);
```

Extract the L-LTF and perform channel estimation.

```
idnLLTF = wlanFieldIndices(cfgHT,'L-LTF');
sym = wlanLLTFDemodulate(rxWaveform(idnLLTF(1):idnLLTF(2),:),cfgHT);
est = wlanLLTFChannelEstimate(sym,cfgHT);
```

Estimate 80 MHz SISO Channel Using L-LTF

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

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

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

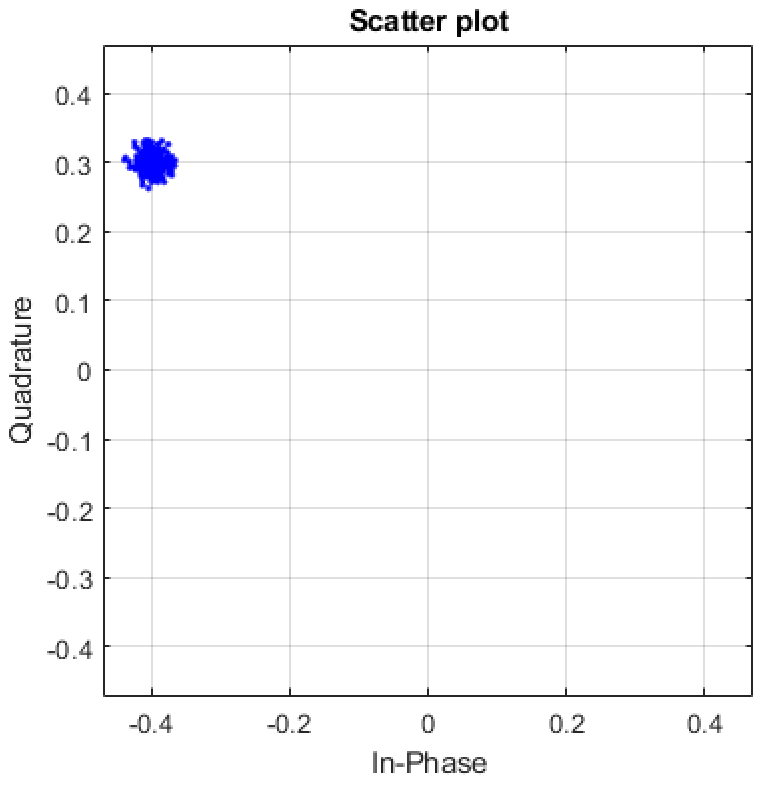
```
rxWaveform = awgn(txWaveform*(-0.4+0.3i),30);
```

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 a VHT format configuration object. Generate a time-domain waveform for an 802.11ac VHT packet.

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

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.

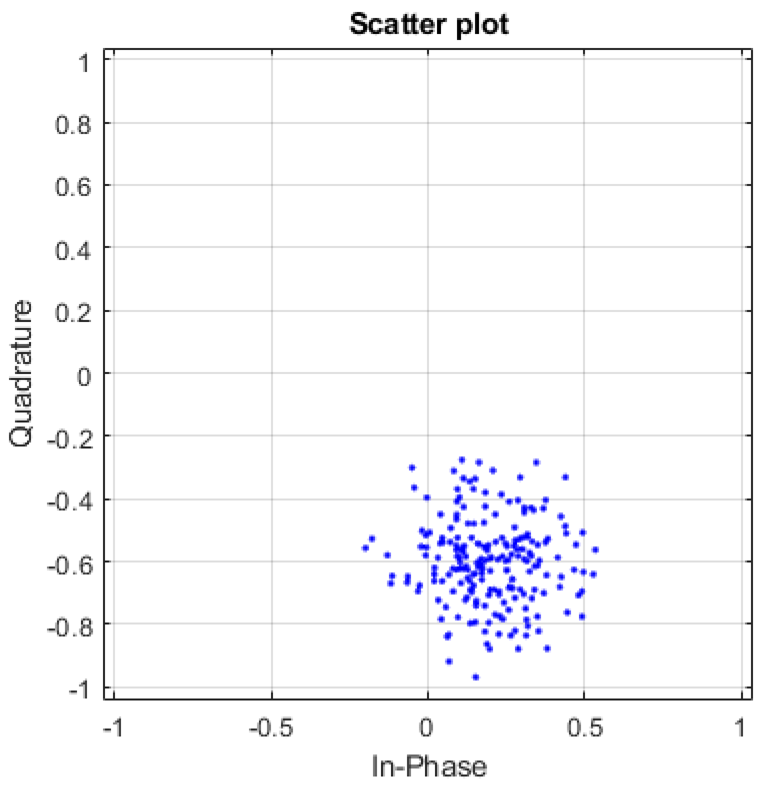
```
idxLLTF = wlanFieldIndices(vht, 'L-LTF');
lltfDemodSig = wlanLLTFDemodulate(rxWaveform(idxLLTF(1):idxLLTF(2),:),vht);
```

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

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

Plot the channel estimate.

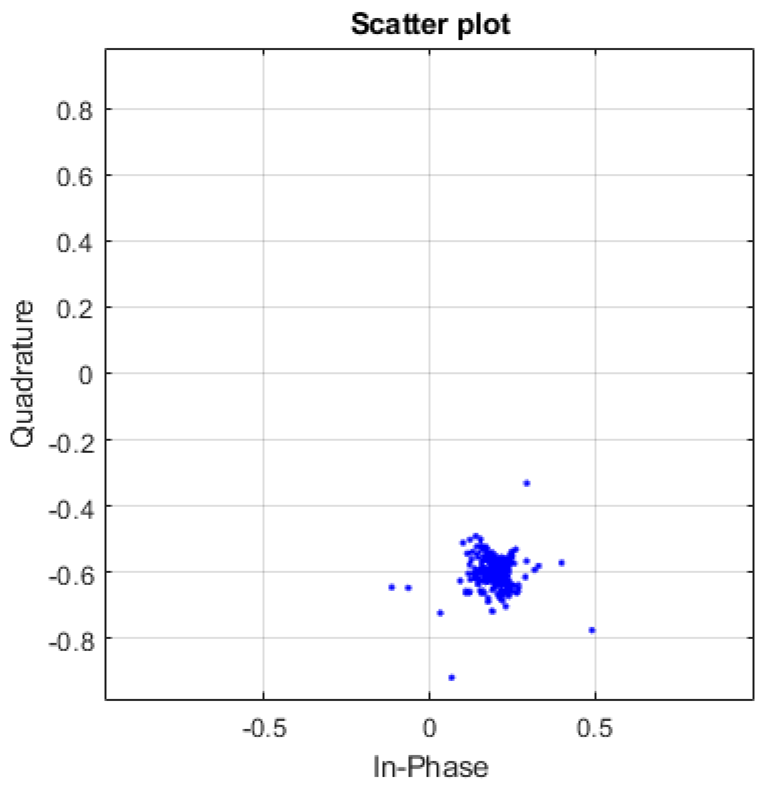
```
scatterplot(est)
grid
```



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 and a Model-A delay profile. Pass the transmitted L-LTF and VHT-SIG-A signals through the channel.

```
tgacChan = wlanTGacChannel('SampleRate',80e6,'ChannelBandwidth','CBW80', ...
    'DelayProfile','Model-A');

rxLLTFNoNoise = tgacChan(txLLTF);
rxSigNoNoise = tgacChan(txSig);
```

Create an AWGN noise channel with an SNR = 15 dB. Add the AWGN noise to L-LTF and VHT-SIG-A signals.

```
chNoise = comm.AWGNChannel('NoiseMethod','Signal to noise ratio (SNR)', ...
    'SNR',15);

rxLLTF = chNoise(rxLLTFNoNoise);
rxSig = chNoise(rxSigNoNoise);
```

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);
awgnChan = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);

rxLLTF = awgnChan(rxLLTF);
rxSig = awgnChan(rxSig);
```

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 =

  logical
```

```
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

`'CBW5'` | `'CBW10'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth of the packet transmission waveform, specified as:

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

PPDU Transmission Format	Valid Channel Bandwidth
non-HT	`'CBW5'`, `'CBW10'`, or `'CBW20'` (default)

Data Types: `char` | `string`

`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 1-105.

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

Data Types: `double`

Output Arguments

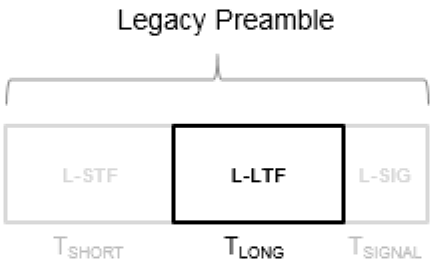
`chEst` — Channel estimate
3-D array

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.

Definitions

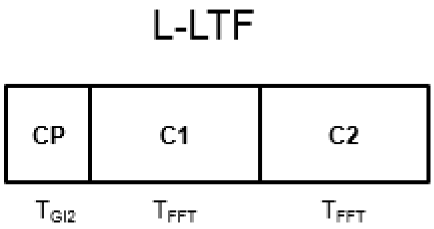
L-LTF

The legacy long training field (L-LTF) is the second field in the 802.11 OFDM PLCP legacy preamble. The L-LTF is a component of VHT, HT, and non-HT PPDUs.



Channel estimation, fine frequency offset estimation, and fine symbol timing offset estimation rely on the L-LTF.

The L-LTF is composed of a cyclic prefix (CP) followed by two identical long training symbols (C1 and C2). The CP consists of the second half of the long training symbol.



The L-LTF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period (T_{FFT} = 1 / Δ_F)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration (T_{GI2} = T_{FFT} / 2)	L-LTF Duration (T_{LONG} = T_{GI2} + 2 × T_{FFT})
20, 40, 80, and 160	312.5	3.2 μs	1.6 μs	8 μs

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period (T_{FFT} = 1 / Δ_F)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration (T_{GI2} = T_{FFT} / 2)	L-LTF Duration (T_{LONG} = T_{GI2} + 2 × T_{FFT})
10	156.25	6.4 μs	3.2 μs	16 μs
5	78.125	12.8 μs	6.4 μs	32 μs

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., O. Edfors, M. Sandell, S. K. Wilson, and P. O. Borjesson. “On Channel Estimation in OFDM Systems." Vehicular Technology Conference, IEEE 45th, Volume 2, IEEE, 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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-113[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);
```

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

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

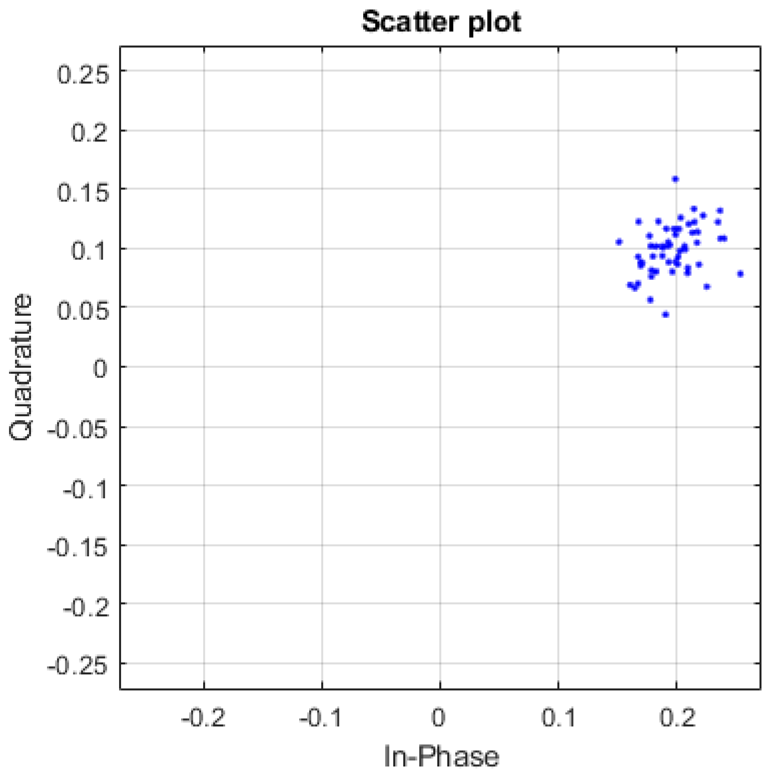
```
rxSig = awgn(txSig*(0.2+0.1i),30);
demodSig = wlanHTLTFDemodulate(rxSig,cfg);
```

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. Transmit a complete HT waveform.

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

Pass the transmitted waveform through a 2x2 TGn channel.

```
tgnChan = wlanTGnChannel('SampleRate',20e6, ...
    'NumTransmitAntennas',2, ...
    'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
rxWaveformNoNoise = tgnChan(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);
awgnChan = comm.AWGNChannel('NoiseMethod','Variance', ...
    'Variance',nVar);
```

Pass the signal through the AWGN channel.

```
rxWaveform = awgnChan(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`

`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` | `string`

`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`
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

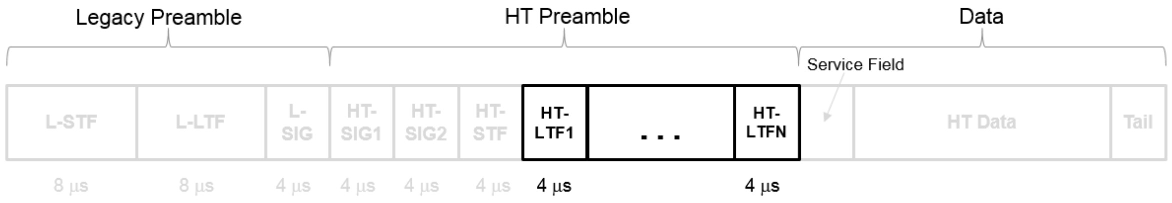
Channel estimate between all combinations of space-time streams and receive antennas, returned as an N_{ST}-by-(N_{STS}+N_{ESS})-by-N_R array. N_{ST} is the number of occupied subcarriers, N_{STS} is the number of space-time streams. N_{ESS} is the number of extension spatial streams. N_R is the number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

Data Types: `double`

Definitions

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 $N_{HTLTF} = 5$.

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanHTConfig` | `wlanHTLTF` | `wlanHTLTFDemodulate`

Introduced in R2015b

wlanVHTLTFChannelEstimate

Channel estimation using VHT-LTF

Syntax

```
chEst = wlanVHTLTFChannelEstimate(demodSig,cfg)
chEst = wlanVHTLTFChannelEstimate(demodSig,cbw,numSTS)
chEst = wlanVHTLTFChannelEstimate(___,span)
```

Description

`chEst = wlanVHTLTFChannelEstimate(demodSig,cfg)` returns the channel estimate, using the demodulated "VHT-LTF" on page 1-124[5] signal, `demodSig`, given the parameters specified in `wlanVHTConfig` object `cfg`.

`chEst = wlanVHTLTFChannelEstimate(demodSig,cbw,numSTS)` returns the channel estimate for the specified channel bandwidth, `cbw`, and the number of space-time streams, `numSTS`.

`chEst = wlanVHTLTFChannelEstimate(___,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`.

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

```
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.

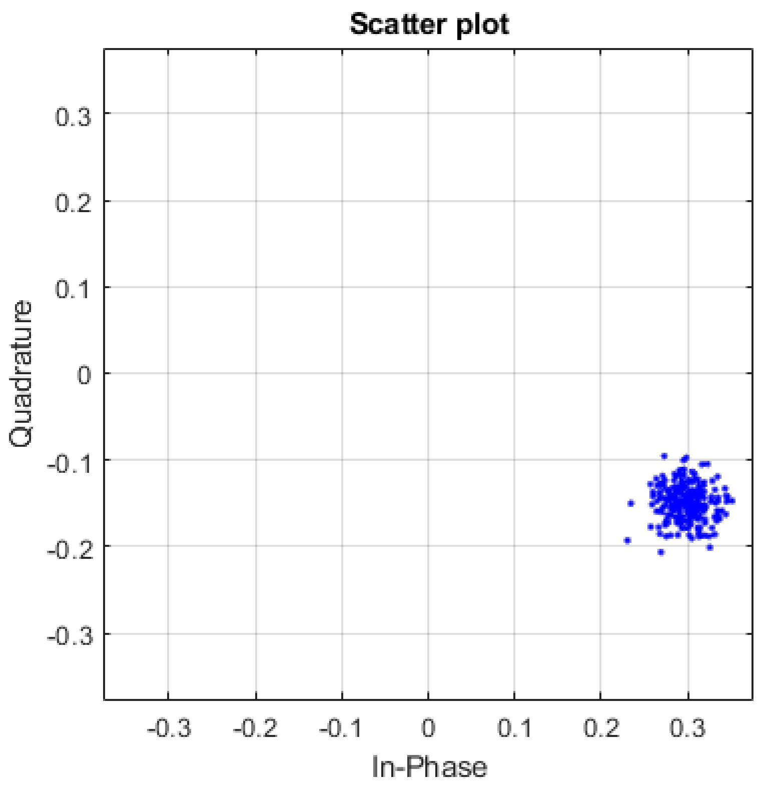
```
rxSig = awgn(txSig*(0.3-0.15i),30);
demodSig = wlanVHTLTFDemodulate(rxSig,cfg);
```

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. Transmit a complete VHT waveform.

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

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

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

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

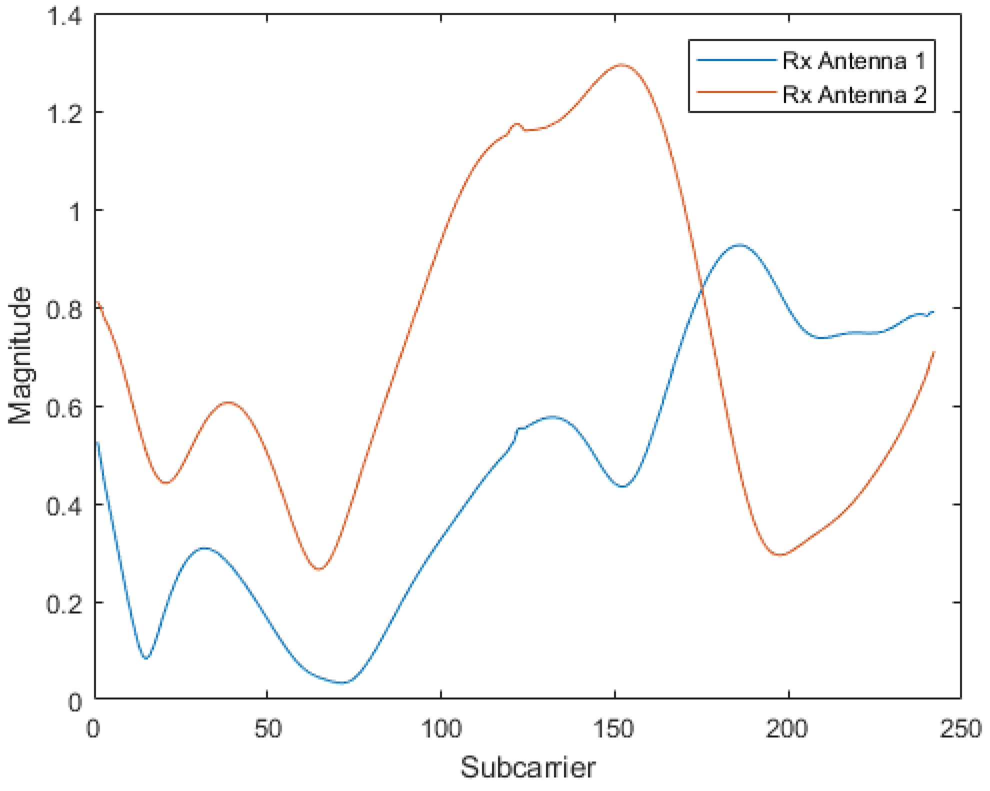
```
indVHTLTF = wlanFieldIndices(cfg,'VHT-LTF');
ltfDemodSig = wlanVHTLTFDemodulate(rxWaveform(indVHTLTF(1):indVHTLTF(2),:), cfg);
```

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')
```



Recover VHT-Data Field in MU-MIMO Channel

Recover VHT-Data field bits for a multiuser transmission using channel estimation on a VHT-LTF field over a quasi-static fading channel.

Create a VHT configuration object having a 160 MHz channel bandwidth, two users, and four transmit antennas. Assign one space-time stream to the first user and three space-time streams to the second user.

```
cbw = 'CBW160';
numSTS = [1 3];
```

```
vht = wlanVHTConfig('ChannelBandwidth',cbw,'NumUsers',2, ...
    'NumTransmitAntennas',4,'NumSpaceTimeStreams',numSTS);
```

Because there are two users, the PSDU length is a 1-by-2 row vector.

```
psduLen = vht.PSDULength
```

```
psduLen =

        1050        3156
```

Generate multiuser input data. This data must be in the form of a 1-by- *N* cell array, where *N* is the number of users.

```
txDataBits{1} = randi([0 1],8*vht.PSDULength(1),1);
txDataBits{2} = randi([0 1],8*vht.PSDULength(2),1);
```

Generate VHT-LTF and VHT-Data field signals.

```
txVHTLTF  = wlanVHTLTF(vht);
txVHTData = wlanVHTData(txDataBits,vht);
```

Pass the data field for the first user through a 4x1 channel because it consists of a single space-time stream. Pass the second user's data through a 4x3 channel because it consists of three space-time streams. Apply white Gaussian noise to each user signal.

```
snr = 15;
H1 = 1/sqrt(2)*complex(randn(4,1),randn(4,1));
H2 = 1/sqrt(2)*complex(randn(4,3),randn(4,3));

rxVHTData1 = awgn(txVHTData*H1,snr,'measured');
rxVHTData2 = awgn(txVHTData*H2,snr,'measured');
```

Repeat the process for the VHT-LTF fields.

```
rxVHTLTF1  = awgn(txVHTLTF*H1,snr,'measured');
rxVHTLTF2  = awgn(txVHTLTF*H2,snr,'measured');
```

Calculate the received signal power for both users and use it to estimate the noise variance.

```
powerDB1 = 10*log10(var(rxVHTData1));
noiseVarEst1 = mean(10.^(0.1*(powerDB1-snr)));
```

```
powerDB2 = 10*log10(var(rxVHTData2));
noiseVarEst2 = mean(10.^(0.1*(powerDB2-snr)));
```

Estimate the channel characteristics using the VHT-LTF fields.

```
demodVHTLTF1 = wlanVHTLTFDemodulate(rxVHTLTF1,cbw,numSTS);
chanEst1 = wlanVHTLTFChannelEstimate(demodVHTLTF1,cbw,numSTS);

demodVHTLTF2 = wlanVHTLTFDemodulate(rxVHTLTF2,cbw,numSTS);
chanEst2 = wlanVHTLTFChannelEstimate(demodVHTLTF2,cbw,numSTS);
```

Recover VHT-Data field bits for the first user and compare against the original payload bits.

```
rxDataBits1 = wlanVHTDataRecover(rxVHTData1,chanEst1,noiseVarEst1,vht,1);
[~,ber1] = biterr(txDataBits{1},rxDataBits1)
```

```
ber1 =

    0.4983
```

Determine the number of bit errors for the second user.

```
rxDataBits2 = wlanVHTDataRecover(rxVHTData2,chanEst2,noiseVarEst2,vht,2);
[~,ber2] = biterr(txDataBits{2},rxDataBits2)
```

```
ber2 =

    0.0972
```

The bit error rates are quite high because there is no precoding to mitigate the interference between streams. This is especially evident for the user 1 receiver because it receives energy from the three streams intended for user 2. The example is intended to show the workflow and proper syntaxes for the LTF demodulate, channel estimation, and data recovery functions.

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`

`cfg` — Format configuration

`wlanVHTConfig`

Format configuration, specified as a `wlanVHTConfig` object.

`cbw` — Channel bandwidth

`'CBW20'` | `'CBW40'` | `'CBW80'` | `'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.

Data Types: `char` | `string`

`numSTS` — Number of space-time streams

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]` indicates that one space-time stream is assigned to user 1, three space-time streams are assigned to user 2, and two space-time streams are assigned to user 3.

Note The sum of the space-time stream vector elements must not exceed eight.

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 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

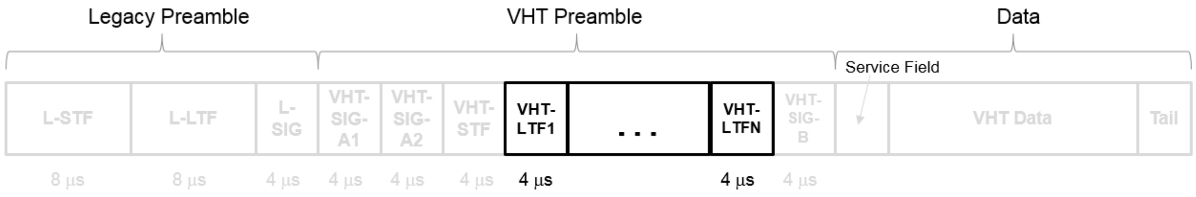
Channel estimate between all combinations of space-time streams and receive antennas, returned as an N_{ST}-by-$N_{STS,total}$-by-N_R array. N_{ST} is the number of occupied subcarriers. $N_{STS,total}$ is the total number of space-time streams for all users. For the single-user case, $N_{STS,total}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

Data Types: `double`

Definitions

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanVHTConfig` | `wlanVHTDataRecover` | `wlanVHTLTFDemodulate`

Introduced in R2015b

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 baseband PPDU waveform, given a format configuration object.

Note For non-HT format, this function only supports generation of field indices for OFDM modulation.

`ind = wlanFieldIndices(cfg,field)` returns the start and stop indices for the specified field type in the rows of an N-by-2 matrix.

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. Generate the corresponding VHT waveform.

```
cfg = wlanVHTConfig('MCS',8,'ChannelBandwidth','CBW160','NumTransmitAntennas',2,'NumSpa
txSig = wlanWaveformGenerator([1;0;0;1],cfg);
```

Determine the component PPDU field indices for the VHT format.

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

```
ind = 

  struct with fields:

       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 format object with two paths.

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

Generate a random PSDU and create the corresponding VHT waveform.

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

Pass the signal through a TGac 2x2 MIMO channel.

```
tgacChan = wlanTGacChannel('NumTransmitAntennas',2,'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
rxSigNoNoise = tgacChan(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);
awgnChan = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
rxSig = awgnChan(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

`wlanDMGConfig` | `wlanS1GConfig` | `wlanVHTConfig` | `wlanHTConfig` | `wlanNonHTConfig`

Transmission format, specified as a `wlanDMGConfig`, `wlanS1GConfig`, `wlanVHTConfig`, `wlanHTConfig`, or `wlanNonHTConfig` configuration object.

Example: `txformat = wlanVHTConfig`

`field` — PPDU fieldname

character vector

PPDU fieldname, specified as a character vector. The valid set of `field` values depends on the transmission format specified in `cfg`.

Transmission Format (`cfg`)	Valid Fieldname Values (`field`)
`wlanDMGConfig`	`'DMG-STF'`, `'DMG-CE'`, `'DMG-Header'`, and `'DMG-Data'` are common for all DMG PHY configurations.
	When the `wlanDMGConfig` property `'TrainingLength'` > `0`, additional valid fields include: `'DMG-AGC'`, `'DMG-AGCSubfields'`, `'DMG-TRN'`, `'DMG-TRNCE'`, and `'DMG-TRNSubfields'`. • `'DMG-AGCSubfields'` is returned in a matrix with `TrainingLength` rows • `'DMG-TRNCE'` is returned in a matrix with `TrainingLength`/4 rows • `'DMG-TRNSubfields'` is returned in a matrix with `TrainingLength` rows

Transmission Format (`cfg`)	Valid Fieldname Values (`field`)
`wlanS1GConfig`	`'S1G-STF'`, `'S1G-LTF1'`, and `'S1G-Data'` are common for all S1G configurations.
	For a 1MHz, or ≥ 2MHz short preamble configuration, additional valid fields include `'S1G-SIG'`, or `'S1G-LTF2N'`.
	For ≥ 2MHz long preamble configuration, additional valid fields include `'S1G-SIG-A'`, `'S1G-DSTF'`, `'S1G-DLTF'`, or `'S1G-SIG-B'`.
`wlanVHTConfig`	`'L-STF'`, `'L-LTF'`, `'L-SIG'`, `'VHT-SIG-A'`, `'VHT-STF'`, `'VHT-LTF'`, `'VHT-SIG-B'`, or `'VHT-Data'`
`wlanHTConfig`	`'L-STF'`, `'L-LTF'`, `'L-SIG'`, `'HT-SIG'`, `'HT-STF'`, `'HT-LTF'`, or `'HT-Data'`
`wlanNonHTConfig`	`'L-STF'`, `'L-LTF'`, `'L-SIG'`, or `'NonHT-Data'`

Data Types: `char` | `string`

Output Arguments

`ind` — Start and stop indices

structure | matrix

Start and stop indices, returned as a structure or a matrix. The indices correspond to the start and stop indices of fields included in the baseband waveform defined by the specified WLAN format configuration object.

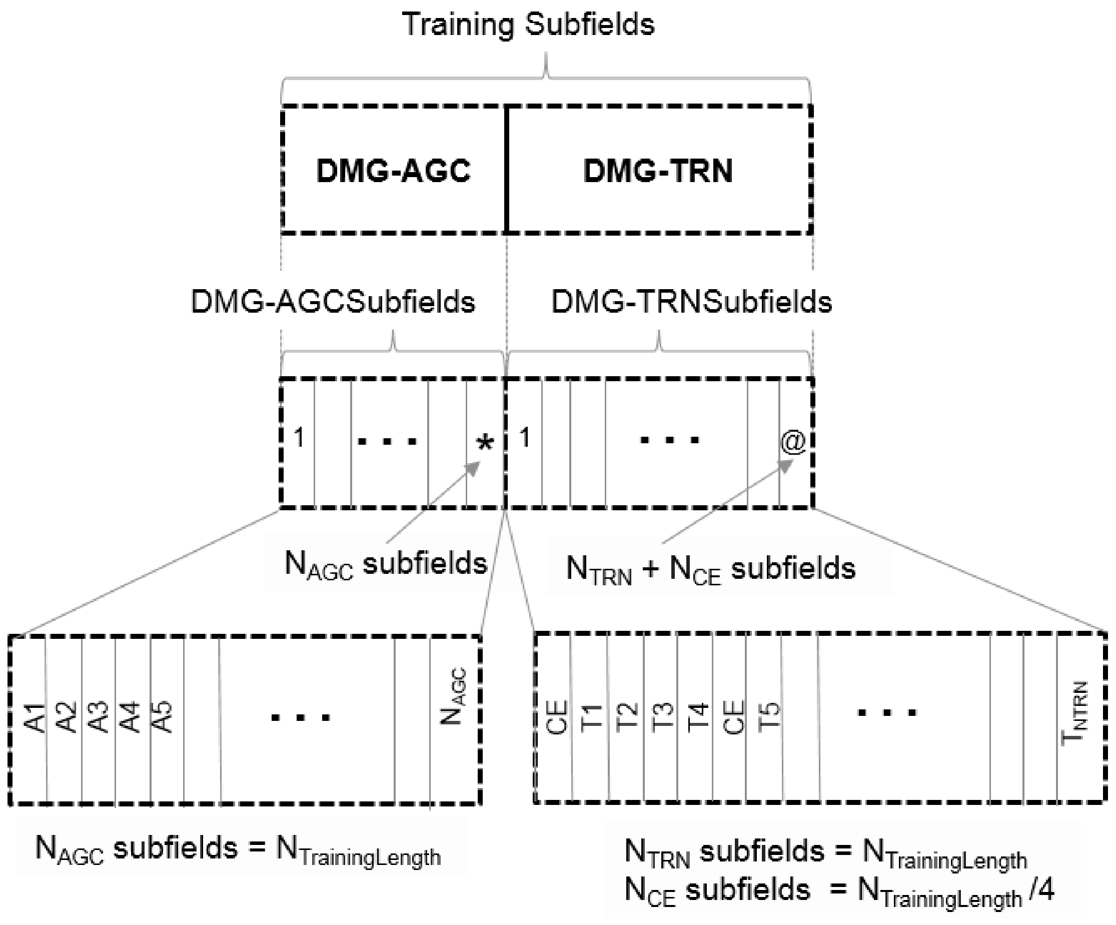
If you specify an input `field`, the function returns `ind` as an *N*-by-2 matrix of `uint32` values, consisting of the start and stop indices of the PPDU field, where *N* is the number of rows.

This table outlines the *N* dimension of the *N*-by-2 matrix that is returned based on the specific format and configuration.

Format	Configuration	`ind` or Specific Field Dimension
non-HT	—	1-by-2
HT	—	1-by-2
	When null data packet (NDP) mode, if `PSDULength` = 0	empty matrix
VHT	—	1-by-2
	When null data packet (NDP) mode, if `APEPLength` = 0	empty matrix
S1G	—	1-by-2
	When null data packet (NDP) mode, if `APEPLength` = 0	empty matrix
DMG	—	1-by-2
	When the `wlanDMGConfig` object property `'TrainingLength'` > 0	`'DMG-AGC'` is a 1-by-2 matrix
		`'DMG-TRN'` is a 1-by-2 matrix
		`'DMG-AGCSubfields'` is a `TrainingLength`-by-2 matrix
		`'DMG-TRNSubfields'` is a `TrainingLength`-by-2 matrix
		`'DMG-TRNCE'` is a (`TrainingLength`/4)-by-2 matrix
	When the `wlanDMGConfig` property `'TrainingLength'` = 0	`'DMG-AGC'` is a 0-by-2 matrix
		`'DMG-TRN'` is a 0-by-2 matrix

Format	Configuration	`ind` or Specific Field Dimension
		`'DMG-AGCSubfields'` is a 0-by-2 matrix
		`'DMG-TRNSubfields'` is a 0-by-2 matrix
		`'DMG-TRNCE'` is a 0-by-2 matrix

The `'DMG-AGC'` field contains $N_{\mathrm{TrainingLength}}$ subfields, where $N_{\mathrm{TrainingLength}}$ is 0 to 64 subfields. The `'DMG-TRN'` field contains $N_{\mathrm{TrainingLength}} + (N_{\mathrm{TrainingLength}}/4)$ subfields. As shown here, the indices for `'DMG-AGC'` and `'DMG-TRN'` overlap with those of their respective subfields, `'DMG-AGCSubfields'` and `'DMG-TRNSubfields'`.


Training Subfields
DMG-AGC
DMG-TRN
DMG-AGCSubfields
DMG-TRNSubfields
1
...
*
1
...
@
N_{AGC} subfields
N_{TRN} + N_{CE} subfields
A1
A2
A3
A4
A5
...
N_{AGC}
CE
T1
T2
T3
T4
CE
T5
...
T_{NTRN}
N_{AGC} subfields = $N_{TrainingLength}$
N_{TRN} subfields = $N_{TrainingLength}$
N_{CE} subfields = $N_{TrainingLength}/4$

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.

[3] IEEE Std 802.11ad™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanDMGConfig` | `wlanGeneratorConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanS1GConfig` | `wlanVHTConfig` | `wlanWaveformGenerator`

Introduced in R2015b

wlanFormatDetect

Packet format detection

Syntax

```
format = wlanFormatDetect(rxSig,chEst,noiseVarEst,cbw)
format = wlanFormatDetect(rxSig,chEst,noiseVarEst,cbw,cfgRec)
```

Description

`format = wlanFormatDetect(rxSig,chEst,noiseVarEst,cbw)` detects and returns the packet format for the specified received signal. Inputs include the received signal, the channel estimate, the noise variance estimate, and the channel bandwidth. For more information, see "Format Detection Processing" on page 1-142.

`format = wlanFormatDetect(rxSig,chEst,noiseVarEst,cbw,cfgRec)` uses `cfgRec` to specify algorithm options for information bit recovery.

Examples

Detect HT-MF Format Waveform

Perform format detection on a WLAN high throughput mixed format (HT-MF) waveform.

Generate an HT-MF waveform and add noise to the transmitted waveform.

```
cbw = 'CBW20';
cfgTx = wlanHTConfig('ChannelBandwidth',cbw);
tx = wlanWaveformGenerator([1;0;0;1],cfgTx);
snr = 10;
rxSig = awgn(tx,snr);
```

Demodulate Received Signal and Perform Channel Estimation

- Determine indices for the L-LTF for the 20 MHz bandwidth waveform. For this calculation, define local variables for the sample rate and duration of the L-STF and L-LTF fields in seconds.
- Demodulate the L-LTF.
- Perform channel estimation using the L-LTF.
- Estimate the noise variance.

```
sr = 20e6;
Tlstf = 8e-6;
Tlltf = 8e-6;

idxlltf = Tlstf*sr+(1:Tlltf*sr);

lltfDemod = wlanLLTFDemodulate(rxSig(idxlltf,:),cbw);
chEst = wlanLLTFChannelEstimate(lltfDemod,cbw);
noiseVarEst = 10^(-snr/20);
```

Detect Signal Format

- Determine indices for the three symbols following the L-LTF. For a 20 MHz bandwidth waveform, the duration for three symbols is 12 μs.
- Perform format detection.

```
idxDetectionSymbols = (Tlstf+Tlltf)*sr+(1:12e-6*sr);

in = rxSig(idxDetectionSymbols,:);
format = wlanFormatDetect(in,chEst,noiseVarEst,cbw)
```

```
format =

    'HT-MF'
```

Detect VHT Format Waveform After Adjusting Recovery Algorithm

Perform format detection on a WLAN very high throughput (VHT) waveform. Use the recovery configuration object to adjust the default recovery algorithm settings.

Generate an VHT waveform and add noise to the transmitted waveform.

```
cbw = 'CBW80';
cfgTx = wlanVHTConfig('ChannelBandwidth',cbw);
tx = wlanWaveformGenerator([1;0;0;1],cfgTx);
snr = 10;
rxSig = awgn(tx,snr);
```

Received signal demodulation and channel estimation

- Determine indices for the L-LTF for the 80 MHz bandwidth waveform. For this calculation, define local variables for the sample rate and duration of the L-STF and L-LTF fields in seconds.
- Demodulate the L-LTF.
- Perform channel estimation using the L-LTF.
- Estimate the noise variance.

```
sr = 80e6;
Tlstf = 8e-6;
Tlltf = 8e-6;

idxlltf = Tlstf*sr+(1:Tlltf*sr);

lltfDemod = wlanLLTFDemodulate(rxSig(idxlltf,:),cbw);
chEst = wlanLLTFChannelEstimate(lltfDemod,cbw);
noiseVarEst = 10^(-snr/20);
```

Format detection

- Determine indices for the three symbols following the L-LTF. For an 80 MHz bandwidth waveform, the duration for three symbols is 12 μs.
- Adjust the default recovery settings.
- Perform format detection using modified recovery settings.

```
TdetectionSymbols = 12e-6;
idxDetectionSymbols = (Tlstf+Tlltf)*sr+(1:TdetectionSymbols*sr);
in = rxSig(idxDetectionSymbols,:);
cfgRec = wlanRecoveryConfig('OFDMSymbolOffset',0.5,...
    'PilotPhaseTracking','None')
format = wlanFormatDetect(in,chEst,noiseVarEst,cbw,cfgRec)
```

```
cfgRec =
  wlanRecoveryConfig with properties:

             OFDMSymbolOffset: 0.5000
           EqualizationMethod: 'MMSE'
           PilotPhaseTracking: 'None'
    MaximumLDPCIterationCount: 12
             EarlyTermination: 0


format =

    'VHT'
```

Input Arguments

`rxSig` — Received time-domain signal
matrix

Received time-domain signal containing the three OFDM symbols immediately following the L-LTF, specified as an N_S-by-N_R matrix. N_S represents the number of time-domain samples in three OFDM symbols. N_R is the number of receive antennas.

Note If N_S is greater than three OFDM symbols, additional samples after the first three symbols are not used.

Data Types: `double`
Complex Number Support: Yes

`chEst` — Channel estimation
matrix | 3-D array

Channel estimation for data and pilot subcarriers based on the L-LTF, specified as a matrix or array of size N_{ST}-by-1-by-N_R. N_{ST} is the number of occupied subcarriers. The second dimension corresponds to the single transmitted stream in the L-LTF. If multiple transmit antennas are used, the single transmitted stream includes the combined cyclic shifts. N_R is the number of receive antennas.

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
`'CBW5'` | `'CBW10'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW5'`, `'CBW10'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

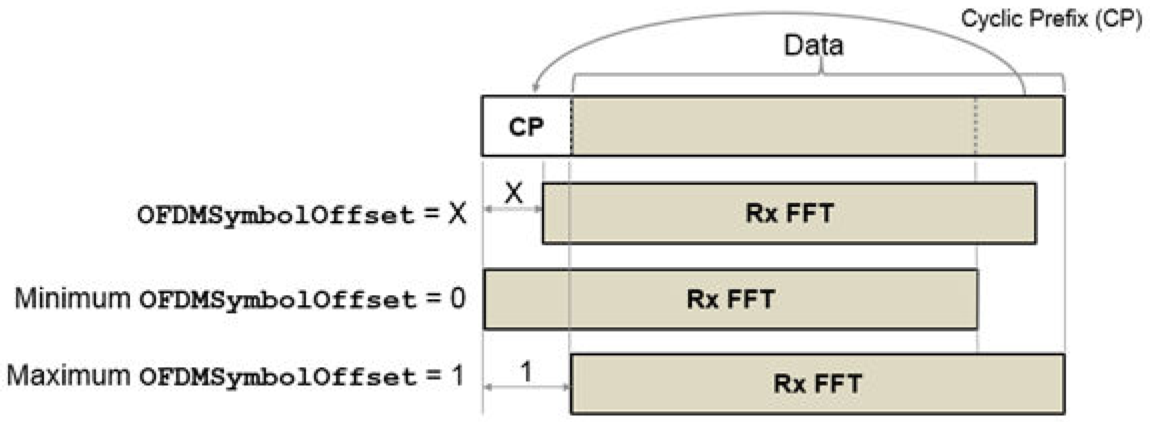
Data Types: `char`

`cfgRec` — Algorithm parameters
`wlanRecoveryConfig` object

Algorithm parameters containing properties used during data recovery, specified as a `wlanRecoveryConfig` object. The configurable properties include the OFDM symbol sampling offset, the equalization method, and the type of pilot phase tracking. If you do not specify a `cfgRec` object, the default object property values described in wlanRecoveryConfig are used in the data recovery.

`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` | `string`

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

Pilot phase tracking, specified as `'PreEQ'` or `'None'`.

- `'PreEQ'` — Enables pilot phase tracking, which is performed before any equalization operation.
- `'None'` — Pilot phase tracking does not occur.

Data Types: `char` | `string`

Output Arguments

`format` — Packet format
`'Non-HT'` | `'HT-MF'` | `'HT-GF'` | `'VHT'`

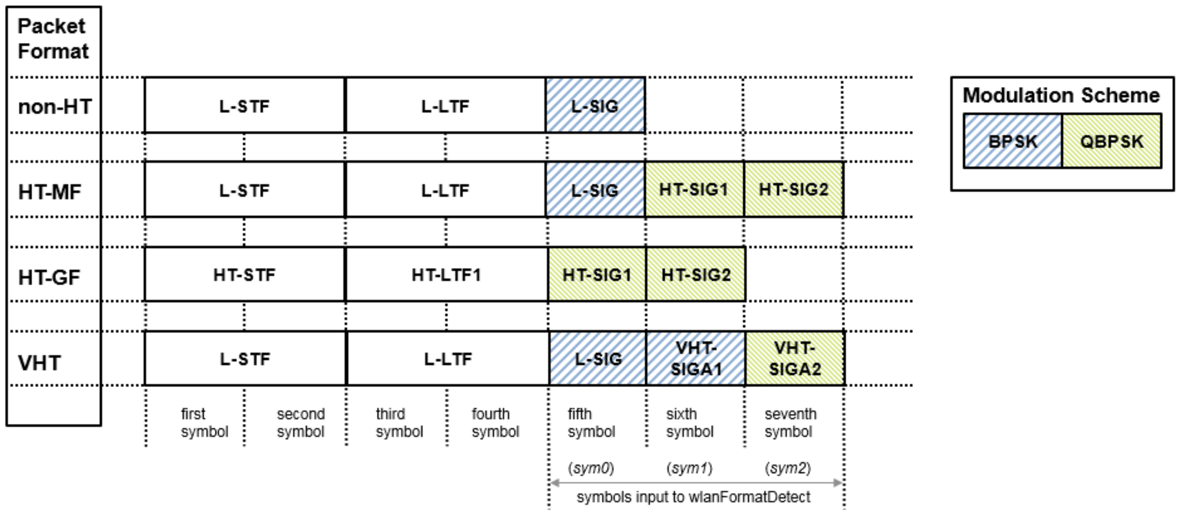
Packet format, returned as `'Non-HT'`, `'HT-MF'`, `'HT-GF'`, or `'VHT'`.

Algorithms

Format Detection Processing

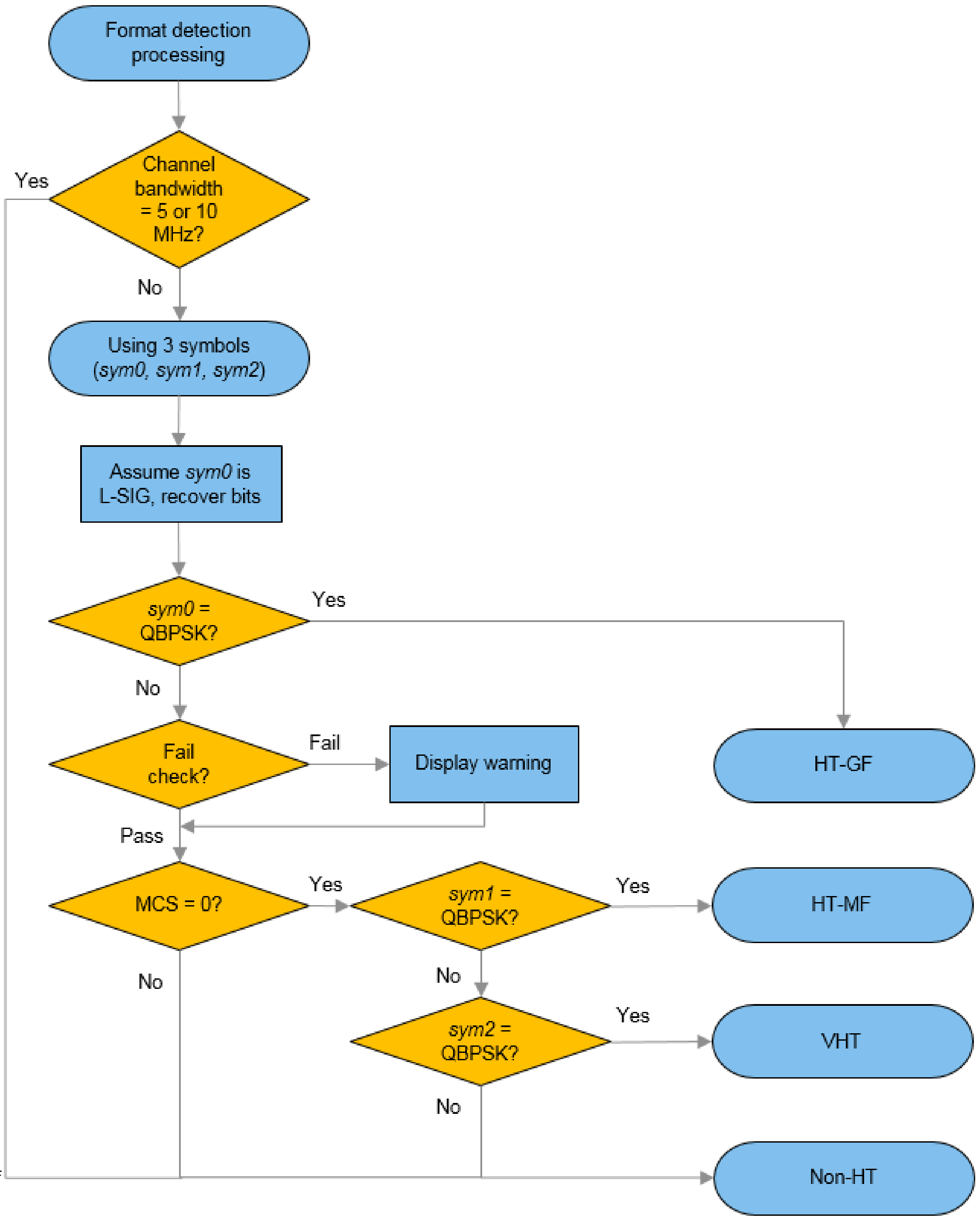
The format detection processing algorithm determines the packet format by detecting the modulation scheme of three symbols. Specifically, the input waveform, `rxSig`, should include three symbols, beginning with the first sample of the fifth symbol and ending with the last sample of the seventh symbol. Additional samples after the last sample of symbol seven are not used.

- If the packet is non-HT, HT-MF, or VHT format, these are the three symbols following the L-LTF symbol.
- If the packet is HT-GF format, these are the three symbols following the HT-LTF1 symbol.



Prior to demodulating any packet symbols, the `wlanFormatDetect` function checks the channel bandwidth input. If the channel bandwidth is 5 MHz or 10 MHz, the algorithm processing concludes and the function returns `non-HT` as the detected packet format. The channel estimate, noise variance estimate, and channel bandwidth are used in the recovery of L-SIG field bits from the fifth symbol, and in the demodulation and equalization of the sixth and seventh symbols.

The logic associated with format detection confirms the modulation scheme by using three consecutive symbols, beginning with the first signaling symbol (L-SIG or HT-SIG1) in sequence. As shown, the packet format prediction is made based on which symbols are BPSK or QBPSK modulated. This logic flow chart identifies the fifth, sixth, and seventh symbols of the packet as *sym0*, *sym1*, and *sym2*, respectively.


Format detection processing
Channel bandwidth = 5 or 10 MHz?
Yes
No
Using 3 symbols (sym0, sym1, sym2)
Assume sym0 is L-SIG, recover bits
sym0 = QBPSK?
Yes
No
Fail check?
Fail
Display warning
Pass
HT-GF
MCS = 0?
Yes
No
sym1 = QBPSK?
Yes
No
HT-MF
sym2 = QBPSK?
Yes
No
VHT
Non-HT

- If *sym0* is QBPSK, the packet format is HT-GF.
- If *sym0* is BPSK and the L-SIG parity check fails, a warning is issued. The format detection processing continues because the L-SIG parity check does not conclusively indicate an error in the MCS determination.
  - If the MCS is not zero, the packet format is non-HT.
  - If the MCS is zero, the modulation scheme of *sym1* is detected.
    - If *sym1* is QBPSK, the packet format is HT-MF.
    - If *sym1* is BPSK, *sym2* is detected.
      - If *sym2* is QBPSK, the packet format is VHT.
      - If *sym2* is BPSK, the packet format is non-HT.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanLLTFChannelEstimate` | `wlanLSIGRecover` | `wlanRecoveryConfig`

Introduced in R2016b

wlanGeneratorConfig

(Not recommended) Create waveform generator configuration object

Note To override default waveform generator configuration values, use the `wlanWaveformGenerator(bits,cfgFormat,Name1,Value1,...,NameN,ValueN)` syntax.

Use of `wlanGeneratorConfig` is not recommended. Therefore, use of the `wlanWaveformGenerator(bits,cfgFormat,cfgWaveGen)` syntax is discouraged as well.

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.

Examples

Waveform Generator Parameterization

Waveform generation using the waveform generator configuration object as shown in Method 1 is no longer supported. Instead, use `Name,Value` pairs when creating the waveform as shown in Method 2.

Create VHT Format Configuration Object

Create a format configuration object for an 802.11ac VHT transmission. Specify generation of a waveform with 10 packets and a 20 microsecond idle period between packets. Use a random scrambler initial value for each packet.

```
cfgVHT = wlanVHTConfig;
numPackets = 10;
idleTime = 20e-6;
scramblerInit = randi([1 127],numPackets, 1);
```

Method 1: (Not Supported) Use wlanGeneratorConfig object in wlanWaveformGenerator

```
%cfgWaveGen = wlanGeneratorConfig('NumPackets',numPackets,...
%    'IdleTime',idleTime,'ScramblerInitialization', scramblerInit);
```

To supress the warning associated with use of `wlanGeneratorConfig`, execute "`warning('off','wlan:wlanGeneratorConfig:Deprecation');`" in your MATLAB® command window prior to the running code.

Method 2: Use Name-Value pair syntax in wlanWaveformGenerator

```
txWaveform2 = wlanWaveformGenerator([1;0;0;1],cfgVHT,'NumPackets',...
    numPackets,'IdleTime',idleTime,'ScramblerInitialization',scramblerInit);
```

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. The default value is 0. If `IdleTime` is not set to the default value, it must be:

- ≥ 1e-06 seconds for DMG format
- ≥ 2e-06 seconds for VHT, HT-mixed, non-HT formats

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` — Request windowing between consecutive OFDM symbols

`true` (default) | `false`

Request windowing between consecutive OFDM symbols. Setting `Windowing` to `false` indicates that beam windowing will not be applied between OFDM symbols.

Data Types: `logical`

`WindowTransitionTime` — Duration of the window transition

nonnegative scalar

Duration of the window transition applied to each OFDM symbol, specified in seconds as a nonnegative scalar. No windowing is applied if `WindowTransitionTime` = 0. The default and maximum value permitted is shown for the various formats, type of guard interval, and channel bandwidth.

	Maximum Permitted `WindowTransitionTime` (seconds)						
	DMG	S1G	VHT	HT-mixed	non-HT		
	2640 MHz	1, 2, 4, 8, or 16 MHz	20, 40, 80, or 160 MHz	20 or 40 MHz	20 MHz	10 MHz	5 MHz
Default	6.0606e-09 (= 16/2640e6)	1.0e-07	1.0e-07	1.0e-07	1.0e-07	1.0e-07	1.0e-07
Maximum	9.6969e-08 (= 256/2640e6)	–	–	–	–	–	–

	Maximum Permitted **`WindowTransitionTime`** (seconds)						
	DMG	S1G	VHT	HT-mixed	non-HT		
	2640 MHz	1, 2, 4, 8, or 16 MHz	20, 40, 80, or 160 MHz	20 or 40 MHz	20 MHz	10 MHz	5 MHz
Maximum Permitted for Long Guard Interval	–	1.6e-05	1.6e-06	1.6e-06	1.6e-06	3.2e-06	6.4e-06
Maximum Permitted for Short Guard Interval	–	8.0e-06	8.0e-07	8.0e-07	–	–	–

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.

See Also

`wlanWaveformGenerator`

Introduced in R2015b

wlanGolaySequence

Generate Golay sequence

Syntax

```
[Ga,Gb] = wlanGolaySequence(len)
```

Description

`[Ga,Gb] = wlanGolaySequence(len)` returns the Golay sequences `Ga` and `Gb` for a specified sequence length. The sequences are defined in IEEE 802.11ad-2012, Section 21.11.

Examples

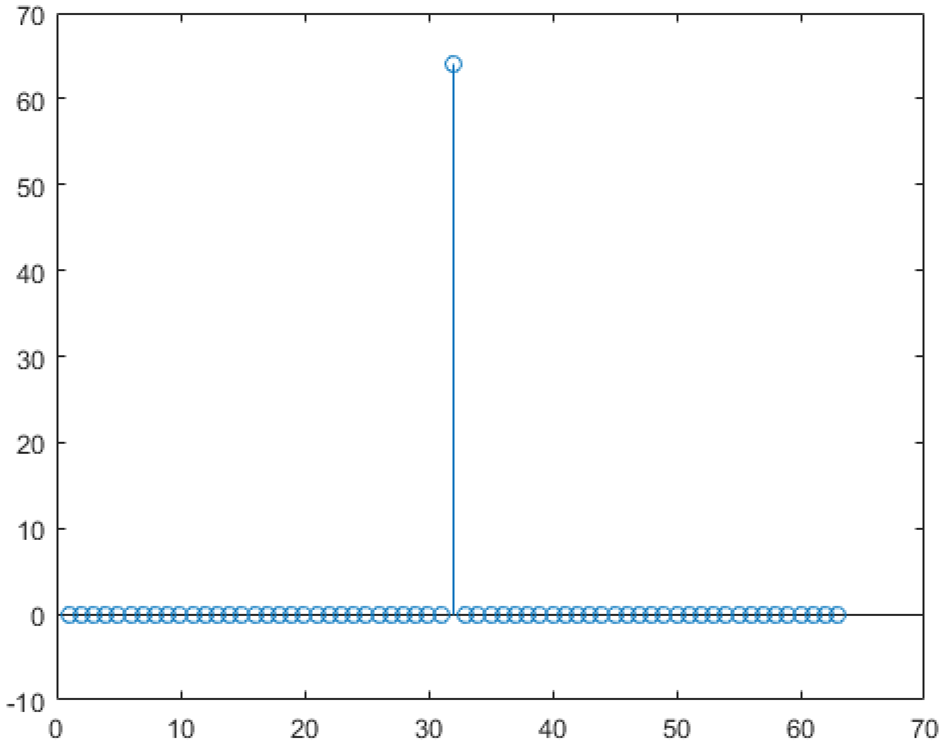
Generate Golay Sequences

Generate complementary 32-length Golay sequences.

```
[Ga,Gb] = wlanGolaySequence(32);
```

The sum of the autocorrelations is a dirac delta function.

```
figure
stem(xcorr(Ga)+xcorr(Gb))
```



Input Arguments

`len` — Sequence length

32 | 64 | 128

Sequence length, specified as 32, 64, or 128.

Data Types: `double`

Output Arguments

`Ga` — Golay sequence

column vector of integers

Golay sequence, returned as a column vector of integers of length `len`.

`Gb` — Complementary Golay sequence

column vector of integers

Complementary Golay sequence, returned as a column vector of integers of length `len`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanDMGConfig` | `wlanDMGDataBitRecover`

Introduced in R2017b

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 mixed (HT-mixed) format “PPDU” on page 1-160.

`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. After creating the object update the number of transmit antennas and space-time streams.

```
cfgHT = wlanHTConfig
```

```
cfgHT = 

  wlanHTConfig with properties:

       ChannelBandwidth: 'CBW20'
```

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

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
         AggregatedMPDU: 0
     RecommendSmoothing: 1
```

Create wlanHTConfig Object

Create a `wlanHTConfig` object 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
         AggregatedMPDU: 0
      RecommendSmoothing: 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: `'ChannelBandwidth','CBW40','NumTransmitAntennas',2`

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

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

Data Types: `char` | `string`

`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` | `string`

`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` | `string`

`ChannelCoding` — **Type of forward error correction coding**
`'BCC'` (default) | `'LDPC'`

Type of forward error correction coding for the data field, specified as `'BCC'` (default) or `'LDPC'`. `'BCC'` indicates binary convolutional coding, and `'LDPC'` indicates low density parity check coding.

Data Types: `char` | `cell` | `string`

`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`

`AggregatedMPDU` — **MPDU aggregation indicator**
`false` (default) | `true`

MPDU aggregation indicator, specified as a logical. Setting `AggregatedMPDU` to `true` indicates that the current packet uses A-MPDU aggregation.

Data Types: `logical`

`RecommendSmoothing` — **Recommend smoothing for channel estimation**
`true` (default) | `false`

Recommend smoothing for channel estimation, specified as a logical.

- If the frequency profile is nonvarying across the channel , the receiver sets this property to `true`. In this case, frequency-domain smoothing is recommended as part of channel estimation.
- If the frequency profile varies across the channel, the receiver sets this property to `false`. In this case, frequency-domain smoothing is not recommended as part of channel estimation.

Data Types: `logical`

Output Arguments

`cfgHT` — HT PPDU configuration
`wlanHTConfig` object

HT “PPDU” on page 1-160 configuration, returned as a `wlanHTConfig` object. The properties of `cfgHT` are described in wlanHTConfig.

Definitions

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanDMGConfig` | `wlanHTDataRecover` | `wlanNonHTConfig` | `wlanS1GConfig` | `wlanVHTConfig` | `wlanWaveformGenerator`

Topics

"Packet Size and Duration Dependencies"

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 1-169[6] time-domain waveform for the input PLCP service data unit, `psdu`, and specified configuration object, `cfg`. See "HT-Data Field Processing" on page 1-170 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 =
```

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

```
  wlanHTConfig with properties:

       ChannelBandwidth: 'CBW40'
     NumTransmitAntennas: 2
     NumSpaceTimeStreams: 2
          SpatialMapping: 'Direct'
                     MCS: 12
           GuardInterval: 'Long'
           ChannelCoding: 'BCC'
              PSDULength: 1024
          AggregatedMPDU: 0
      RecommendSmoothing: 1
```

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 1-170), 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` | `string`

`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` | `string`

`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` | `string`

`ChannelCoding` — Type of forward error correction coding
`'BCC'` (default) | `'LDPC'`

Type of forward error correction coding for the data field, specified as `'BCC'` (default) or `'LDPC'`. `'BCC'` indicates binary convolutional coding and `'LDPC'` indicates low density parity check coding. Providing a character vector or a single cell character vector defines the channel coding type for a single user or all users in a multiuser transmission. By providing a cell array different channel coding types can be specified per user for a multiuser transmission.

Data Types: `char` | `cell` | `string`

`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.

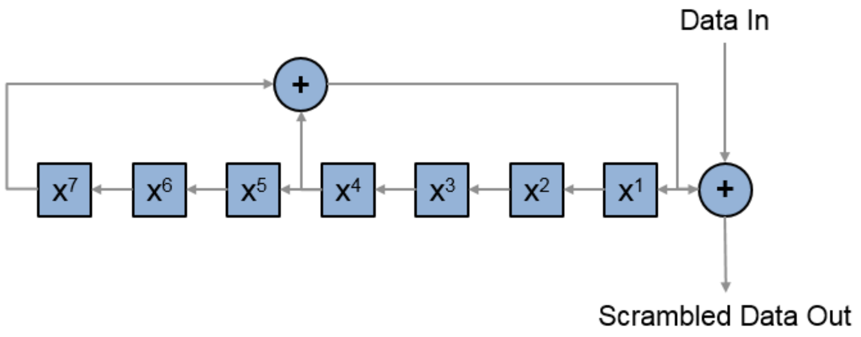
Example: `512`

Data Types: `double`

`scramInit` — Scrambler initialization state
93 (default) | integer from 1 to 127 | binary vector

Scrambler initialization state for each packet generated, specified as an integer from 1 to 127 or as the corresponding binary 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.

The scrambler initialization used on the transmission data follows the process described in IEEE Std 802.11-2012, Section 18.3.5.5 and IEEE Std 802.11ad-2012, Section 21.3.9. The header and data fields that follow the scrambler initialization field (including data padding bits) are scrambled by XORing each bit with a length-127 periodic sequence generated by the polynomial $S(x) = x^7+x^4+1$. The octets of the PSDU (Physical Layer Service Data Unit) are placed into a bit stream, and within each octet, bit 0 (LSB) is first and bit 7 (MSB) is last. The generation of the sequence and the XOR operation are shown in this figure:



Conversion from integer to bits uses left-MSB orientation. For the initialization of the scrambler with decimal `1`, the bits are mapped to the elements shown.

Element	X^7	X^6	X^5	X^4	X^3	X^2	X^1
Bit Value	0	0	0	0	0	0	1

To generate the bit stream equivalent to a decimal, use `de2bi`. For example, for decimal `1`:

```
de2bi(1,7,'left-msb')
ans =

     0     0     0     0     0     0     1
```

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

Data Types: `double` | `int8`

Output Arguments

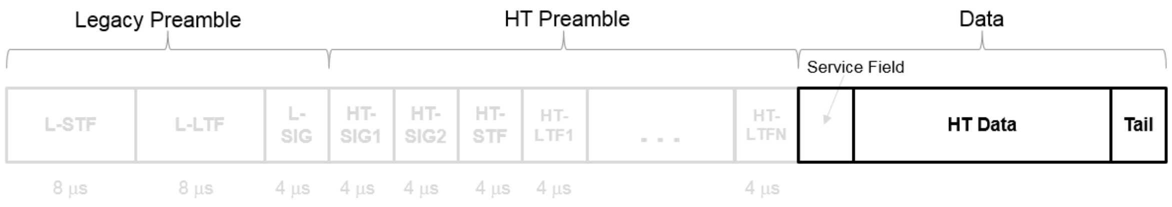
y — HT-Data field time-domain waveform
matrix

"HT-Data field" on page 1-169 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.

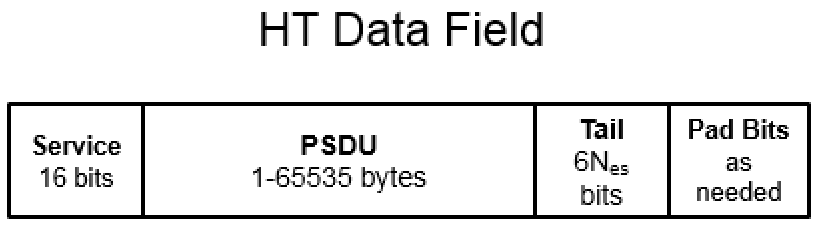
Definitions

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.

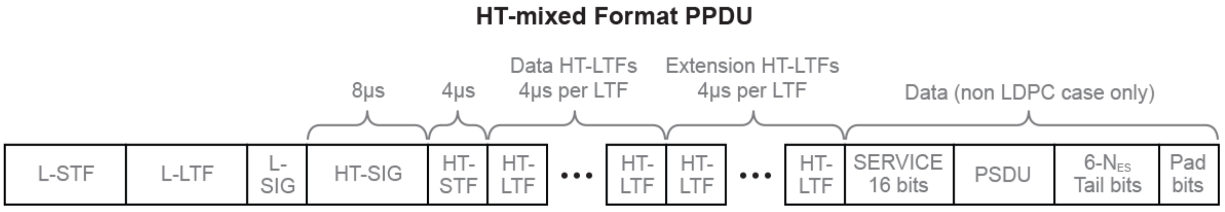
PSDU

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

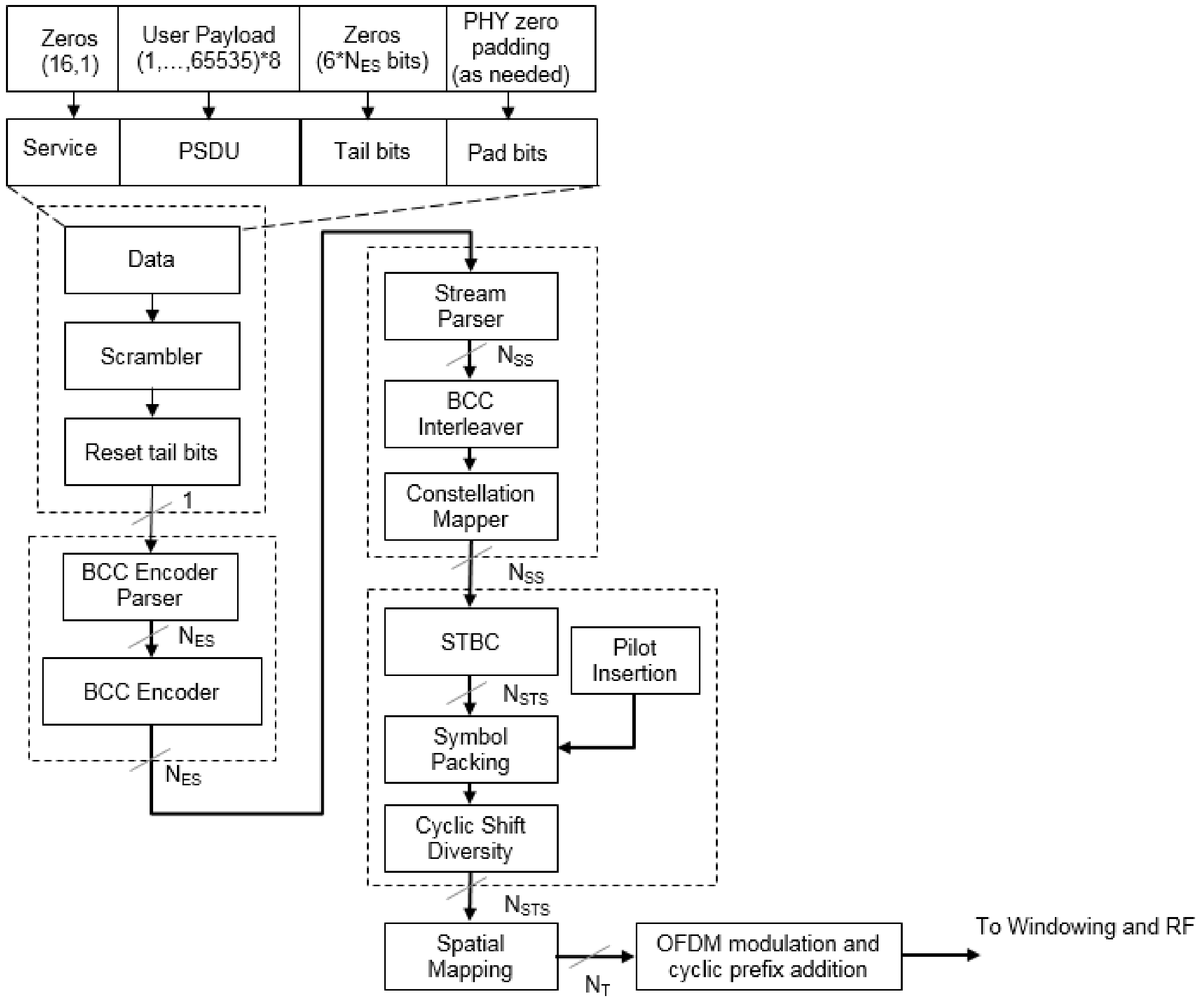
HT-Data Field Processing

The “HT-Data field” on page 1-169 follows the last HT-LTF in the packet structure.



The “HT-Data field” on page 1-169 includes the user payload in the PSDU, plus 16 service bits, $6 \times N_{ES}$ 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 1-169 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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-181[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 1-182 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

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

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.

```
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`

`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`

`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` | `string`

`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`
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` | `string`

`ChannelCoding` — Type of forward error correction coding
`'BCC'` (default) | `'LDPC'`

Type of forward error correction coding for the data field, specified as `'BCC'` (default) or `'LDPC'`. `'BCC'` indicates binary convolutional coding and `'LDPC'` indicates low density parity check coding. Providing a character vector or a single cell character vector defines the channel coding type for a single user or all users in a multiuser transmission. By providing a cell array different channel coding types can be specified per user for a multiuser transmission.

Data Types: `char` | `cell` | `string`

`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`

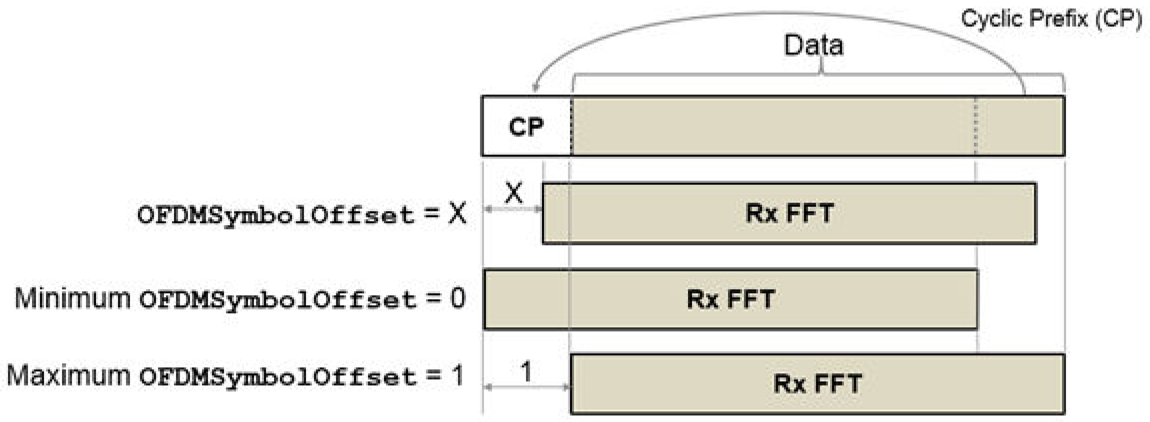
Data Types: `double`

`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` | `string`

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

Pilot phase tracking, specified as `'PreEQ'` or `'None'`.

- `'PreEQ'` — Enables pilot phase tracking, which is performed before any equalization operation.
- `'None'` — Pilot phase tracking does not occur.

Data Types: `char` | `string`

`MaximumLDPCIterationCount` — Maximum number of decoding iterations in LDPC
12 (default) | positive scalar integer

Maximum number of decoding iterations in LDPC, specified as a positive scalar integer. This parameter is applicable when channel coding is set to LDPC. For information on channel coding options, see `wlanVHTConfig` or `wlanHTConfig` for 802.11 format of interest.

Data Types: `double`

`EarlyTermination` — Enable early termination of LDPC decoding
`false` (default) | `true`

Enable early termination of LDPC decoding, specified as a logical. This parameter is applicable when channel coding is set to LDPC.

- When set to `false`, LDPC decoding completes the number of iterations specified by `MaximumLDPCIterationCount`, regardless of parity check status.
- When set to `true`, LDPC decoding terminates when all parity-checks are satisfied.

For information on channel coding options, see `wlanVHTConfig` or `wlanHTConfig` for 802.11 format of interest.

Output Arguments

`recData` — Recovered binary output data
binary column vector

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

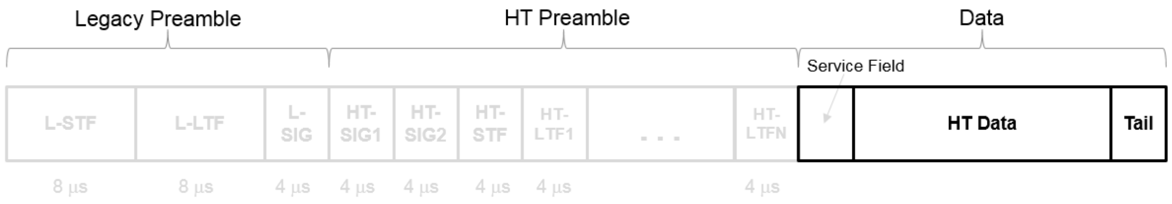
`cpe` — Common phase error
column vector

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.

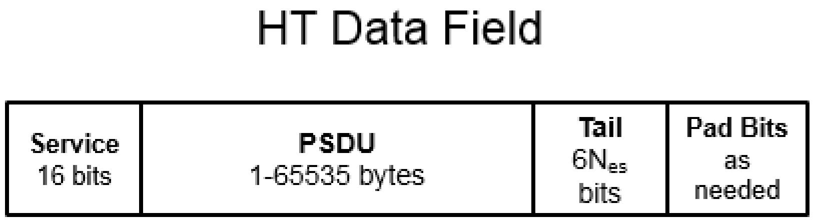
Definitions

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-187[8] time-domain waveform for “HT-mixed” on page 1-188 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
         AggregatedMPDU: 0
     RecommendSmoothing: 1
```

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` | `string`

`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` | `string`

`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

matrix

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 1-187. 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`

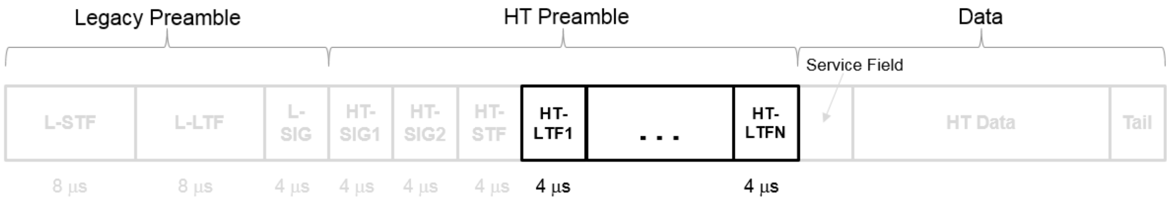
Determination of the number of N_{HTLTF} is described in "HT-LTF" on page 1-187.

Data Types: `double`

Definitions

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-194[9], `y`, given received HT-LTF `x`. The input signal is a component of the “HT-mixed” on page 1-195 format “PPDU” on page 1-195. 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.

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

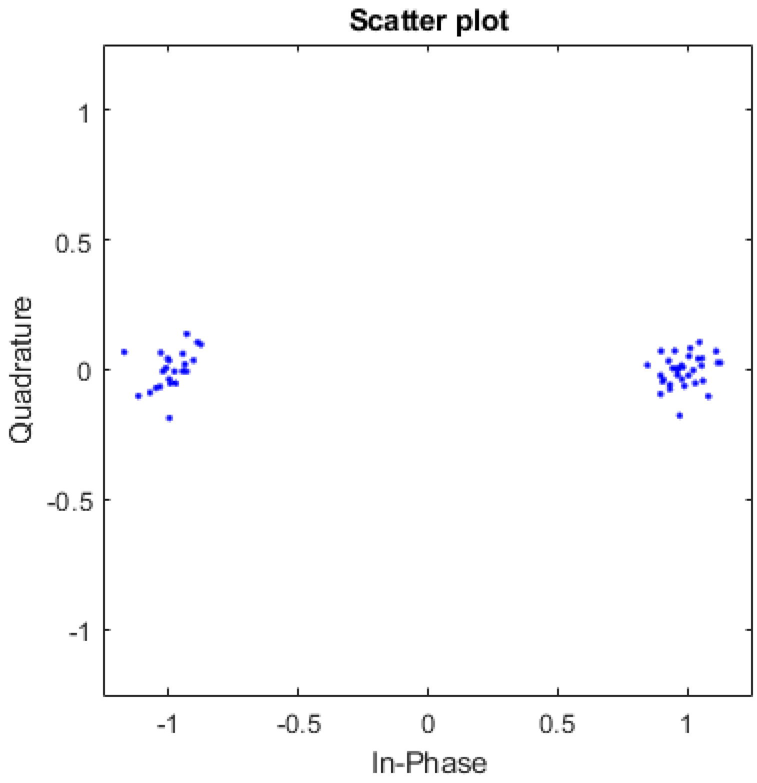
```
y = awgn(x,20);
```

Demodulate the received signal.

```
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 1-194, 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`

`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` | `string`

`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`

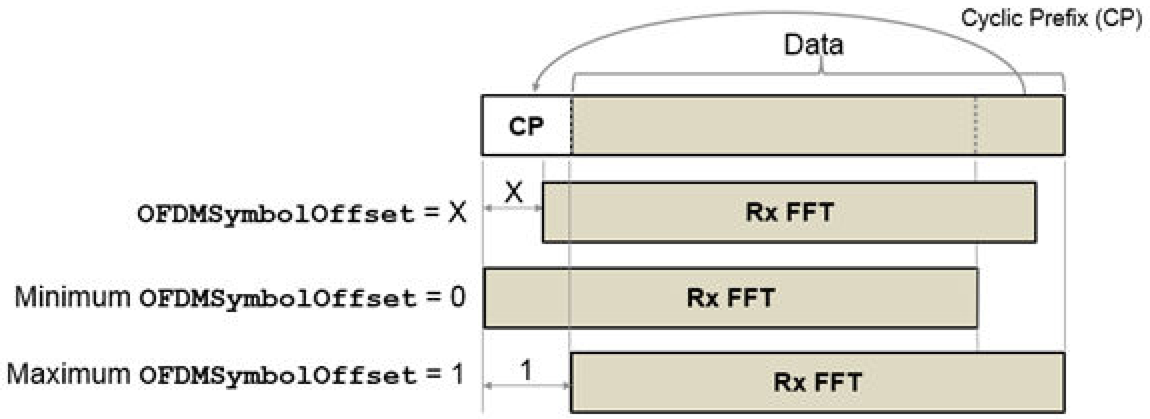
`NumExtensionStreams` — **Number of extension spatial streams**
0 (default) | 1 | 2 | 3

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

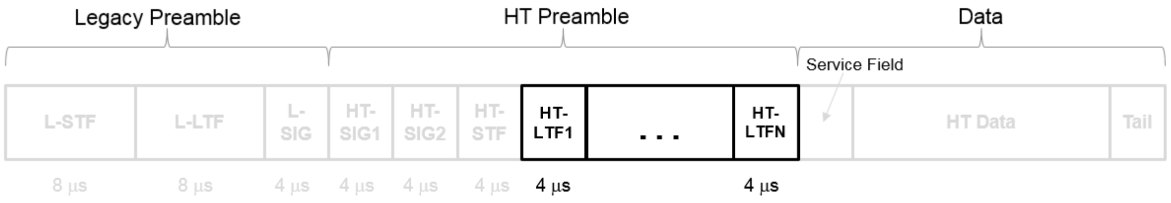
Demodulated HT-LTF signal for an HT-Mixed PPDU, returned as an N_{ST}-by-N_{SYM}-by-N_R matrix or array. N_{ST} is the number of data and pilot subcarriers. N_{SYM} is the number of OFDM symbols in the HT-LTF. N_R is the number of receive antennas.

Data Types: `double`

Definitions

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

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-202[10] time-domain waveform for "HT-mixed" on page 1-204 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 =
```

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

```
  wlanHTConfig with properties:

       ChannelBandwidth: 'CBW40'
    NumTransmitAntennas: 1
    NumSpaceTimeStreams: 1
         SpatialMapping: 'Direct'
                    MCS: 0
          GuardInterval: 'Long'
          ChannelCoding: 'BCC'
             PSDULength: 1024
        AggregatedMPDU: 0
     RecommendSmoothing: 1
```

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 =

  int8

   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 =

  int8

   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`
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` | `string`

`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` — Recommend smoothing for channel estimation
`true` (default) | `false`

Recommend smoothing for channel estimation, specified as a logical.

- If the frequency profile is nonvarying across the channel , the receiver sets this property to `true`. In this case, frequency-domain smoothing is recommended as part of channel estimation.
- If the frequency profile varies across the channel, the receiver sets this property to `false`. In this case, frequency-domain smoothing is not recommended as part of channel estimation.

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) | `'LDPC'`

Type of forward error correction coding for the data field, specified as `'BCC'` (default) or `'LDPC'`. `'BCC'` indicates binary convolutional coding and `'LDPC'` indicates low density parity check coding. Providing a character vector or a single cell character vector defines the channel coding type for a single user or all users in a multiuser transmission. By providing a cell array different channel coding types can be specified per user for a multiuser transmission.

Data Types: `char` | `cell` | `string`

`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` | `string`

`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`

`bits` — HT-SIG information bits
vector

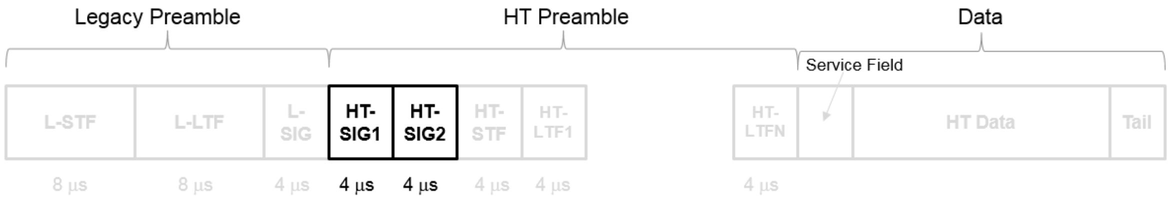
HT-SIG information bits, returned as a 48-by-1 vector.

Data Types: `int8`

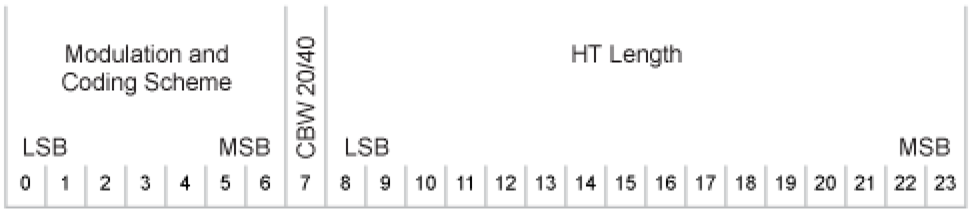
Definitions

HT-SIG

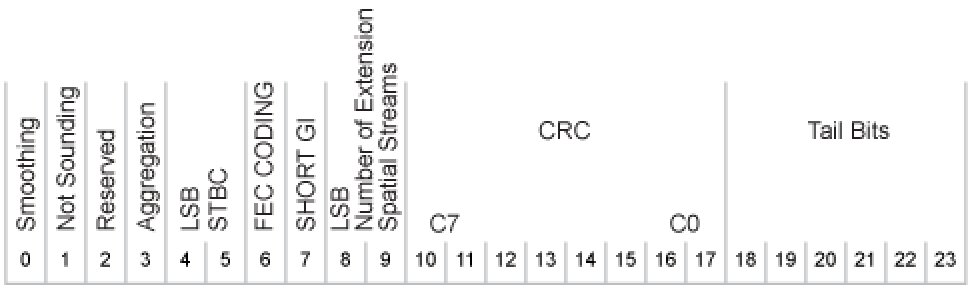
The high throughput signal (HT-SIG) field is located between the L-SIG field and HT-STF and is part of the HT-mixed format preamble. It is composed of two symbols, HT-SIG$_1$ and HT-SIG$_2$.



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.

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-212[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.

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

```
rxSig = awgnChan(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;
tgnChan = 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(tgnChan(txLLTF),20);
rxHTSIG = awgn(tgnChan(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`.

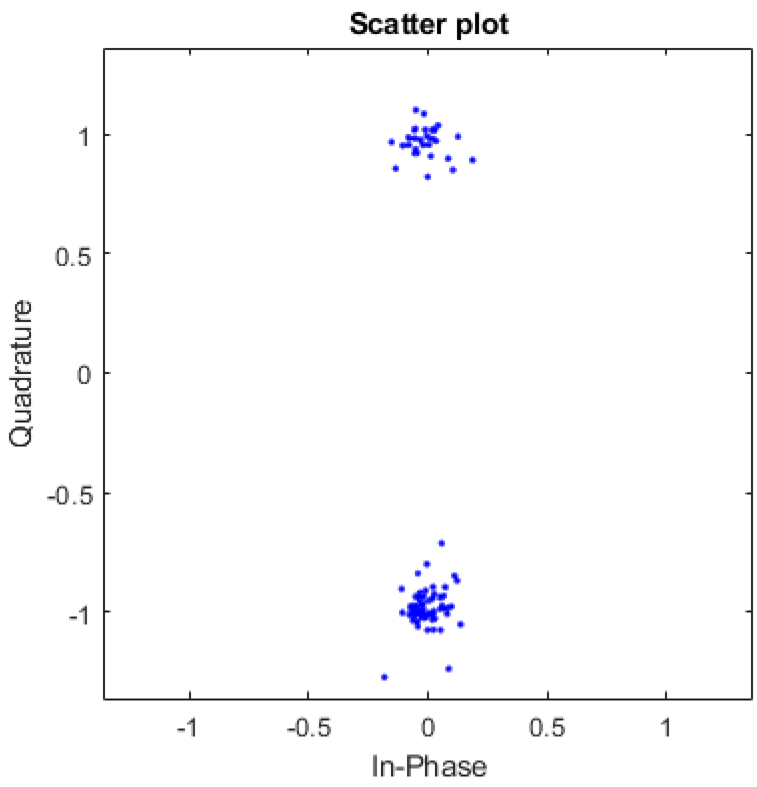
```
failCRC
```

```
failCRC =

  logical

   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
`'CBW40'`	320

N_R is the number of receive antennas.

Data Types: `double`

`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 1-214.

`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 `'CBW20'` or `'CBW40'`.

Data Types: `char` | `string`

`cfgRec` — Algorithm parameters
`wlanRecoveryConfig` object

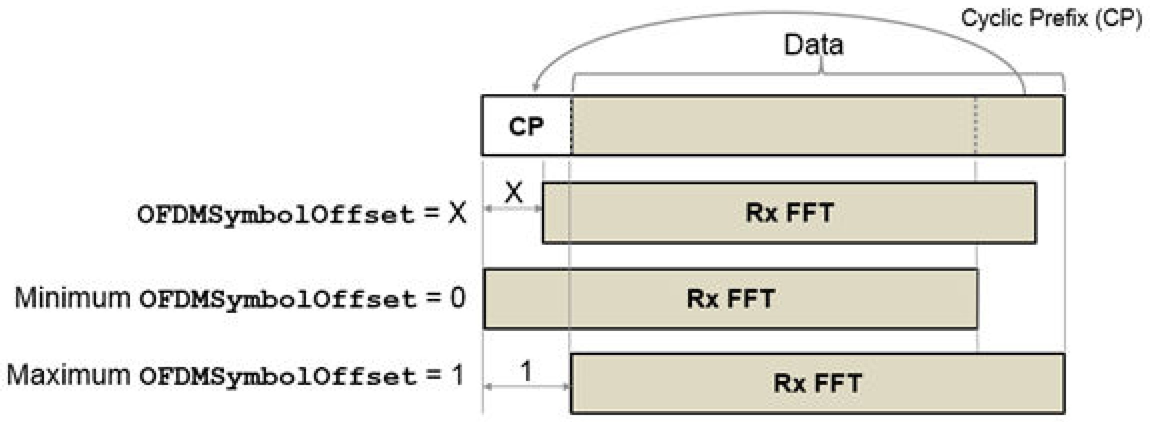
Algorithm parameters, specified as a `wlanRecoveryConfig` object. The function uses these properties.

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` | `string`

`PilotPhaseTracking` — Pilot phase tracking

`'PreEQ'` (default) | `'None'`

Pilot phase tracking, specified as `'PreEQ'` or `'None'`.

- `'PreEQ'` — Enables pilot phase tracking, which is performed before any equalization operation.
- `'None'` — Pilot phase tracking does not occur.

Data Types: `char` | `string`

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.

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

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

`eqSym` — Equalized symbols
matrix

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

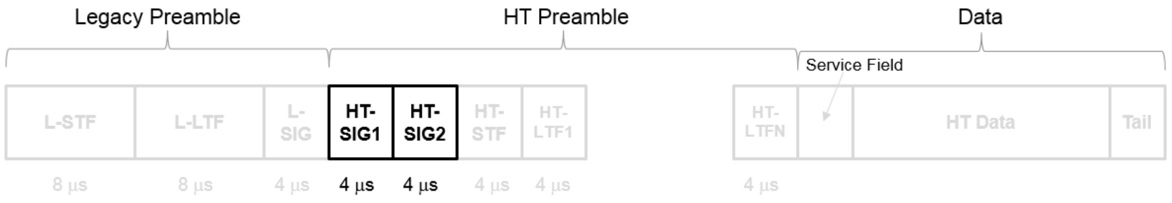
`cpe` — Common phase error
column vector

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

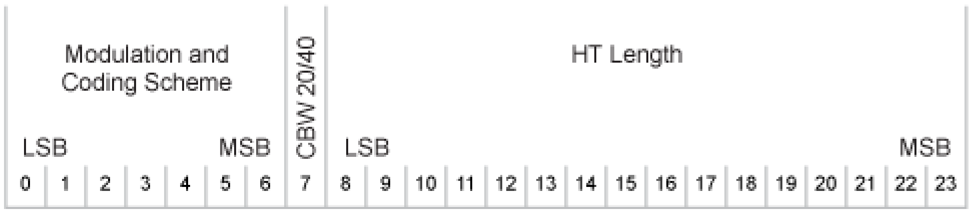
Definitions

HT-SIG

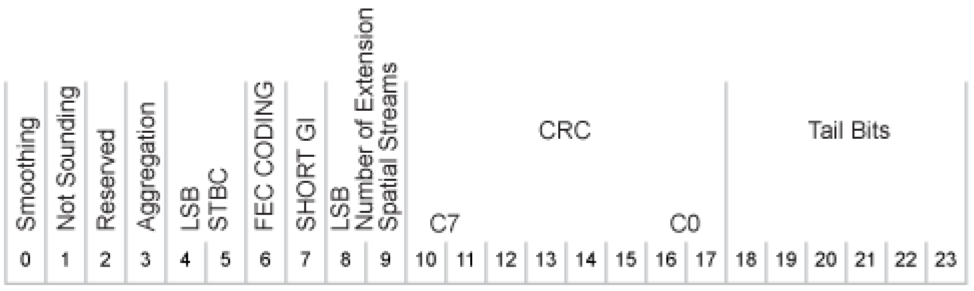
The high throughput signal (HT-SIG) field is located between the L-SIG field and HT-STF and is part of the HT-mixed format preamble. It is composed of two symbols, HT-SIG_1 and HT-SIG_2.



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$_1$

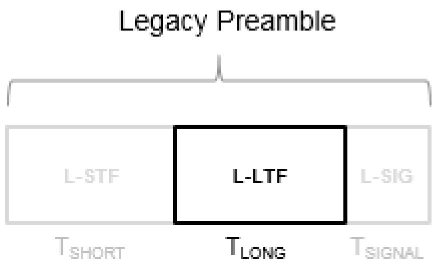


HT-SIG$_2$

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

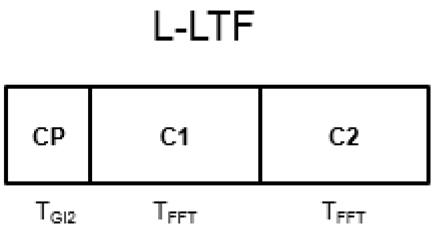
L-LTF

The legacy long training field (L-LTF) is the second field in the 802.11 OFDM PLCP legacy preamble. The L-LTF is a component of VHT, HT, and non-HT PPDUs.



Channel estimation, fine frequency offset estimation, and fine symbol timing offset estimation rely on the L-LTF.

The L-LTF is composed of a cyclic prefix (CP) followed by two identical long training symbols (C1 and C2). The CP consists of the second half of the long training symbol.



The L-LTF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration ($T_{GI2} = T_{FFT} / 2$)	L-LTF Duration ($T_{LONG} = T_{GI2} + 2 \times T_{FFT}$)
20, 40, 80, and 160	312.5	3.2 μs	1.6 μs	8 μs
10	156.25	6.4 μs	3.2 μs	16 μs
5	78.125	12.8 μs	6.4 μs	32 μs

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-219[12] time-domain waveform for "HT-mixed" on page 1-219 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
```

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

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

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

`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` | `string`

`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

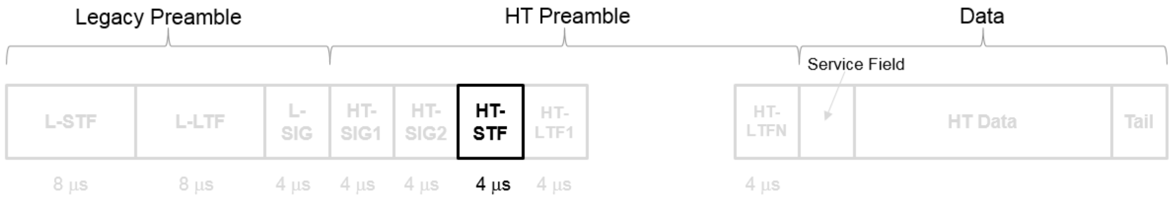
HT-STF waveform, returned as an N_S-by-N_T matrix. N_S is the number of samples, and N_T is the number of transmit antennas.

Data Types: `double`

Definitions

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-224[13] time-domain waveform for the specified configuration object.

Examples

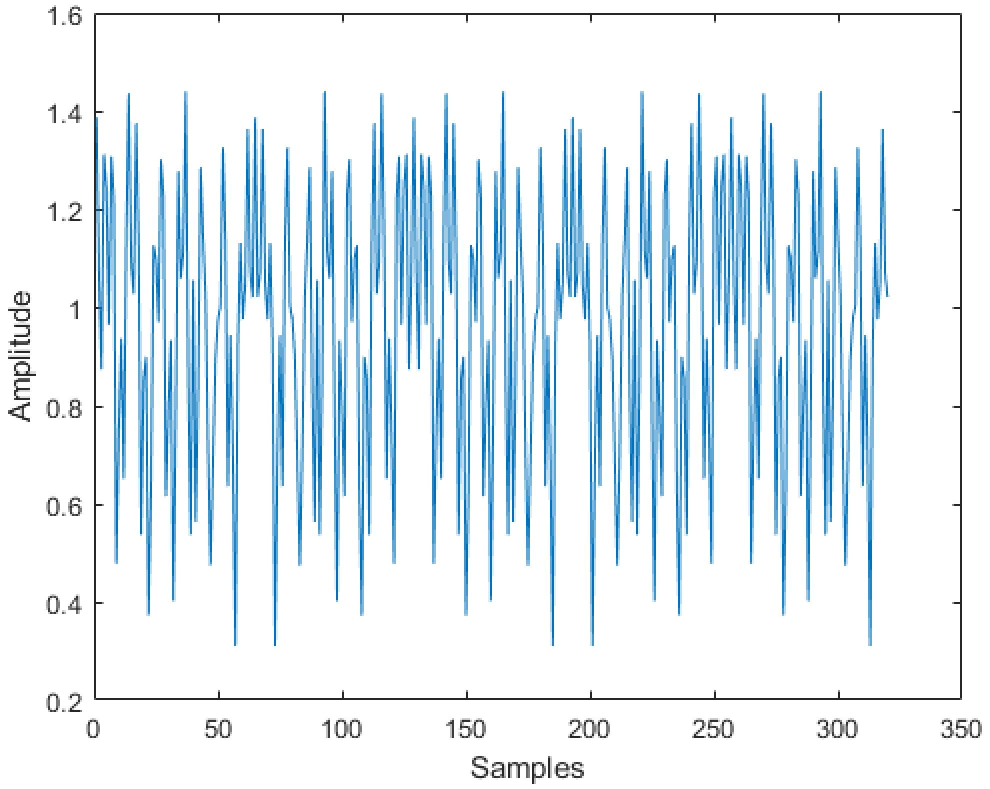
Generate L-LTF Waveform

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	Configuration Object	Applicable Object Properties
VHT	`wlanVHTConfig`	`ChannelBandwidth`, `NumTransmitAntennas`
HT	`wlanHTConfig`	`ChannelBandwidth`, `NumTransmitAntennas`
non-HT See note “1” on page 1-223	`wlanNonHTConfig`	`ChannelBandwidth`, `NumTransmitAntennas`
Note: **1** For non-HT format, when channel bandwidth is 5 MHz or 10 MHz, `NumTransmitAntennas` is not applicable because only one transmit antenna is permitted.		

Example: `wlanVHTConfig`

Output Arguments

`y` — L-LTF time-domain waveform

matrix

“L-LTF” on page 1-224 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.

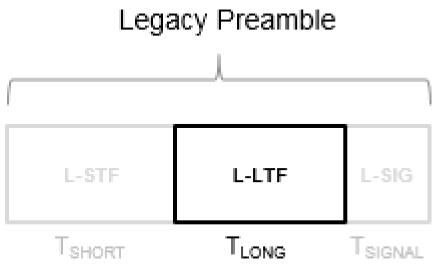
`ChannelBandwidth`	N_S
`'CBW5'`, `'CBW10'`, `'CBW20'`	`160`
`'CBW40'`	`320`
`'CBW80'`	`640`
`'CBW160'`	`1280`

Data Types: `double`
Complex Number Support: Yes

Definitions

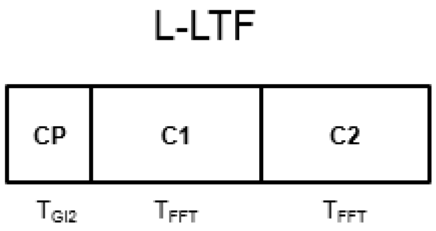
L-LTF

The legacy long training field (L-LTF) is the second field in the 802.11 OFDM PLCP legacy preamble. The L-LTF is a component of VHT, HT, and non-HT PPDUs.



Channel estimation, fine frequency offset estimation, and fine symbol timing offset estimation rely on the L-LTF.

The L-LTF is composed of a cyclic prefix (CP) followed by two identical long training symbols (C1 and C2). The CP consists of the second half of the long training symbol.



The L-LTF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration ($T_{GI2} = T_{FFT} / 2$)	L-LTF Duration ($T_{LONG} = T_{GI2} + 2 \times T_{FFT}$)
20, 40, 80, and 160	312.5	3.2 μs	1.6 μs	8 μs
10	156.25	6.4 μs	3.2 μs	16 μs
5	78.125	12.8 μs	6.4 μs	32 μs

Algorithms

The “L-LTF” on page 1-224 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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-230[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 1-232, 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.

N_S is proportional to the channel bandwidth. The time-domain waveform consists of two symbols.

`ChannelBandwidth`	N_S
`'CBW5'`, `'CBW10'`, `'CBW20'`	`160`
`'CBW40'`	`320`
`'CBW80'`	`640`
`'CBW160'`	`1280`

Data Types: `double`

`cbw` — Channel bandwidth

`'CBW5'` | `'CBW10'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW5'`, `'CBW10'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Data Types: `char` | `string`

`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 OFDM symbol 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.

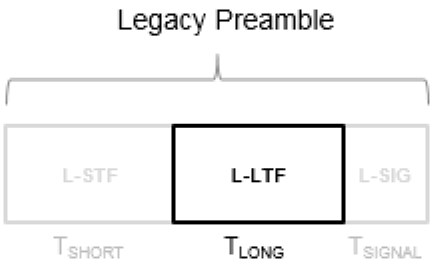
N_{ST} varies with channel bandwidth.

`ChannelBandwidth`	Number of Occupied Subcarriers (N_{ST})
`'CBW20'`,`'CBW10'`,`'CBW5'`	`52`
`'CBW40'`	`104`
`'CBW80'`	`208`
`'CBW160'`	`416`

Definitions

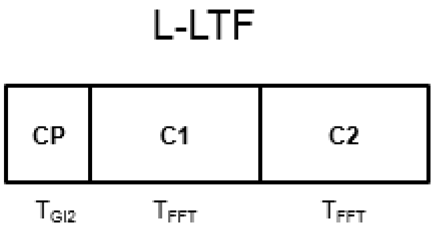
L-LTF

The legacy long training field (L-LTF) is the second field in the 802.11 OFDM PLCP legacy preamble. The L-LTF is a component of VHT, HT, and non-HT PPDUs.



Channel estimation, fine frequency offset estimation, and fine symbol timing offset estimation rely on the L-LTF.

The L-LTF is composed of a cyclic prefix (CP) followed by two identical long training symbols (C1 and C2). The CP consists of the second half of the long training symbol.



The L-LTF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration ($T_{GI2} = T_{FFT} / 2$)	L-LTF Duration ($T_{LONG} = T_{GI2} + 2 \times T_{FFT}$)
20, 40, 80, and 160	312.5	3.2 μs	1.6 μs	8 μs

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration ($T_{GI2} = T_{FFT} / 2$)	L-LTF Duration ($T_{LONG} = T_{GI2} + 2 \times T_{FFT}$)
10	156.25	6.4 µs	3.2 µs	16 µs
5	78.125	12.8 µs	6.4 µs	32 µs

PPDU

The PLCP protocol data unit (PPDU) is the complete “PLCP” on page 1-232 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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-238[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
```

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

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.

```
cfg = wlanNonHTConfig
```

```
cfg = 

  wlanNonHTConfig with properties:

              Modulation: 'OFDM'
        ChannelBandwidth: 'CBW20'
                     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 = 

  4x1 int8 column vector

   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'
       ChannelBandwidth: 'CBW20'
                    MCS: 7
             PSDULength: 1000
    NumTransmitAntennas: 1


rateBits =

  4x1 int8 column vector

   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	Configuration Object	Applicable Object Properties
VHT	`wlanVHTConfig`	`ChannelBandwidth`, `NumUsers`, `NumTransmitAntennas`, `NumSpaceTimeStreams`, `STBC`, `MCS`, `ChannelCoding`, `APEPLength`, `GuardInterval`
HT	`wlanHTConfig`	`ChannelBandwidth`, `NumTransmitAntennas`, `NumSpaceTimeStreams`, `MCS`, `GuardInterval`, `ChannelCoding`, `PSDULength`
non-HT See note “1” on page 1-237 and “2” on page 1-237	`wlanNonHTConfig`	`ChannelBandwidth`, `Modulation`, `MCS`, `PSDULength`, `NumTransmitAntennas`
Note: 1 Only OFDM modulation is supported for a `wlanNonHTConfig` object input. 2 For non-HT format, when channel bandwidth is 5 MHz or 10 MHz, `NumTransmitAntennas` is not applicable because only one transmit antenna is permitted.		

Example: `wlanVHTConfig`

Output Arguments

y — L-SIG time-domain waveform

matrix

“L-SIG” on page 1-238 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.

`ChannelBandwidth`	N_S
`'CBW5','CBW10','CBW20'`	`80`
`'CBW40'`	`160`
`'CBW80'`	`320`
`'CBW160'`	`640`

Data Types: `double`
Complex Number Support: Yes

`bits` — Signaling bits
column vector

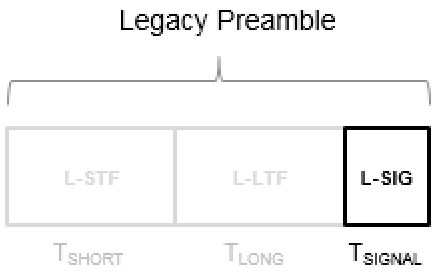
Signaling bits from the legacy signal field, returned as a 24-by-1 bit column vector. See "L-SIG" on page 1-238 for the bit field description.

Data Types: `int8`

Definitions

L-SIG

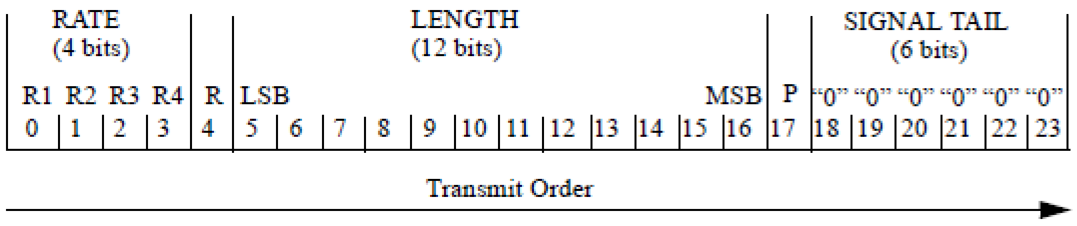
The legacy signal (L-SIG) field is the third field of the 802.11 OFDM PLCP legacy preamble. It consists of 24 bits that contain rate, length, and parity information. The L-SIG is a component of VHT, HT, and non-HT PPDUs. It is transmitted using BPSK modulation with rate 1/2 binary convolutional coding (BCC).



The L-SIG is one OFDM symbol with a duration that varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier frequency spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) period ($T_{FFT} = 1 / \Delta_F$)	Guard Interval (GI) Duration ($T_{GI} = T_{FFT} / 4$)	L-SIG duration ($T_{SIGNAL} = T_{GI} + T_{FFT}$)
20, 40, 80, and 160	312.5	3.2 μs	0.8 μs	4 μs
10	156.25	6.4 μs	1.6 μs	8 μs
5	78.125	12.8 μs	3.2 μs	16 μs

The L-SIG 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 (Mb/s)		
			20 MHz channel bandwidth	10 MHz channel bandwidth	5 MHz channel bandwidth
1101	BPSK	1/2	6	3	1.5
1111	BPSK	3/4	9	4.5	2.25
0101	QPSK	1/2	12	6	3
0111	QPSK	3/4	18	9	4.5
1001	16-QAM	1/2	24	12	6
1011	16-QAM	3/4	36	18	9
0001	64-QAM	2/3	48	24	12

Rate (bits 0–3)	Modulation	Coding rate (*R*)	Data Rate (Mb/s)		
			20 MHz channel bandwidth	10 MHz channel bandwidth	5 MHz channel bandwidth
0011	64-QAM	3/4	54	27	13.5

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.

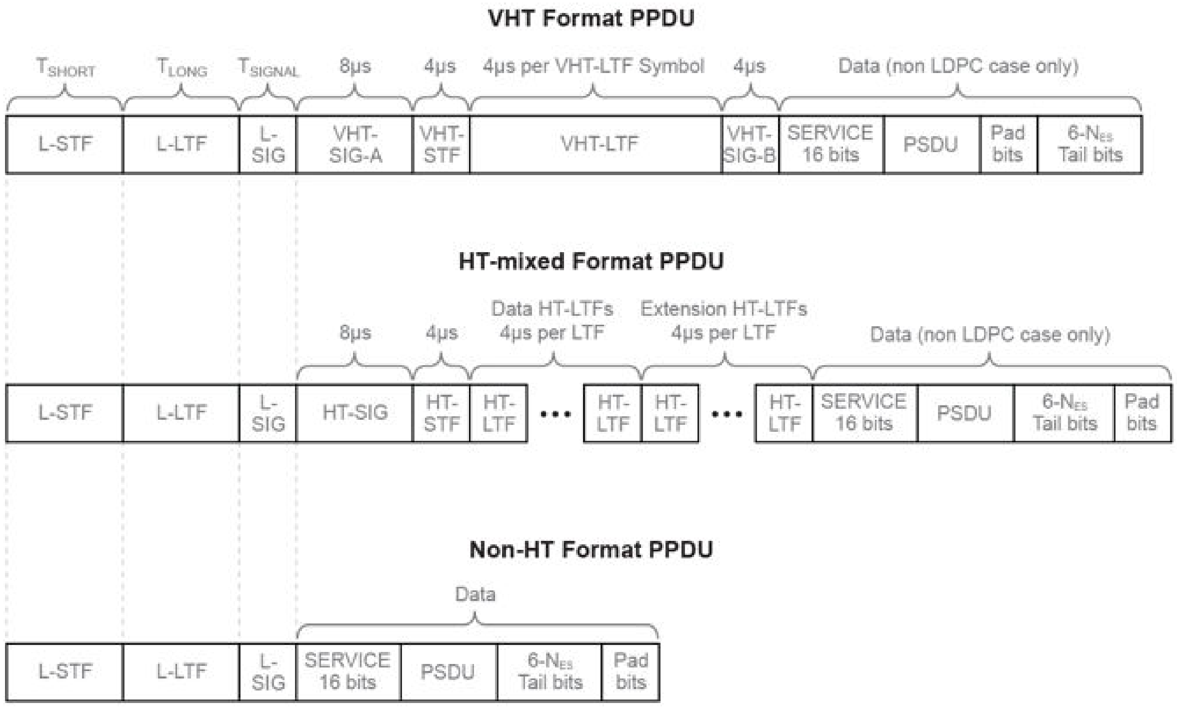
- Bit 4 is reserved for future use.
- Bits 5 through 16:
  - For non-HT, specify the data length (amount of data transmitted in octets) as described in IEEE Std 802.11-2012, Table 18-1 and Section 9.23.4.
  - For HT-mixed, specify the transmission time as described in IEEE Std 802.11-2012, Section 20.3.9.3.5 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.

- For the HT-mixed format, IEEE Std 802.11-2012, Section 20.3.9.4.3 describes HT-SIG bit settings.
- For the VHT 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.

Algorithms

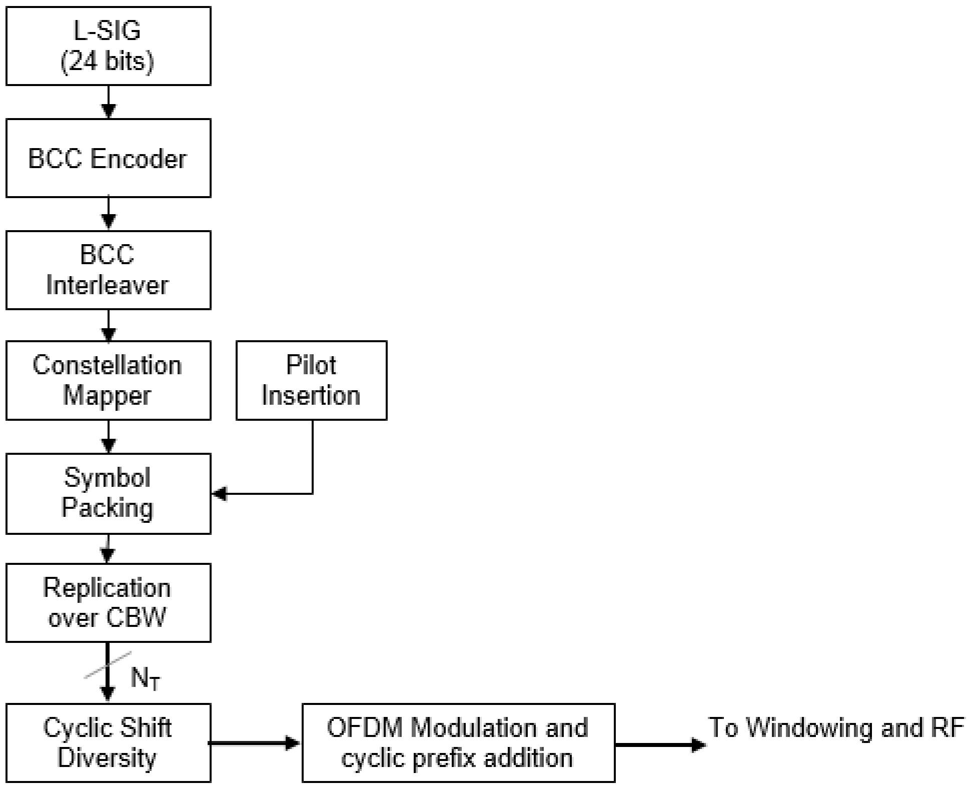
The “L-SIG” on page 1-238 follows the L-STF and L-LTF of the preamble in the packet structure.



For “L-SIG” on page 1-238 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 1-238 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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-253[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);
```

Create a 2x2 TGac channel and an AWGN channel with an SNR=10 dB.

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

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

Pass the signals through the noisy 2x2 multipath fading channel.

```
rxLLTF = chNoise(tgacChan(txLLTF));
rxLSIG = chNoise(tgacChan(txLSIG));
```

Add additional white noise corresponding to a receiver with a 9 dB noise figure. The noise variance is equal to $k*T*B*F$, where k is Boltzmann's constant, T is the ambient temperature, B is the channel bandwidth (sample rate), and F is the receiver noise figure.

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

```
rxLLTF = rxNoise(rxLLTF);
rxLSIG = rxNoise(rxLSIG);
```

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.

```
pfOffset = 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(pfOffset(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
```

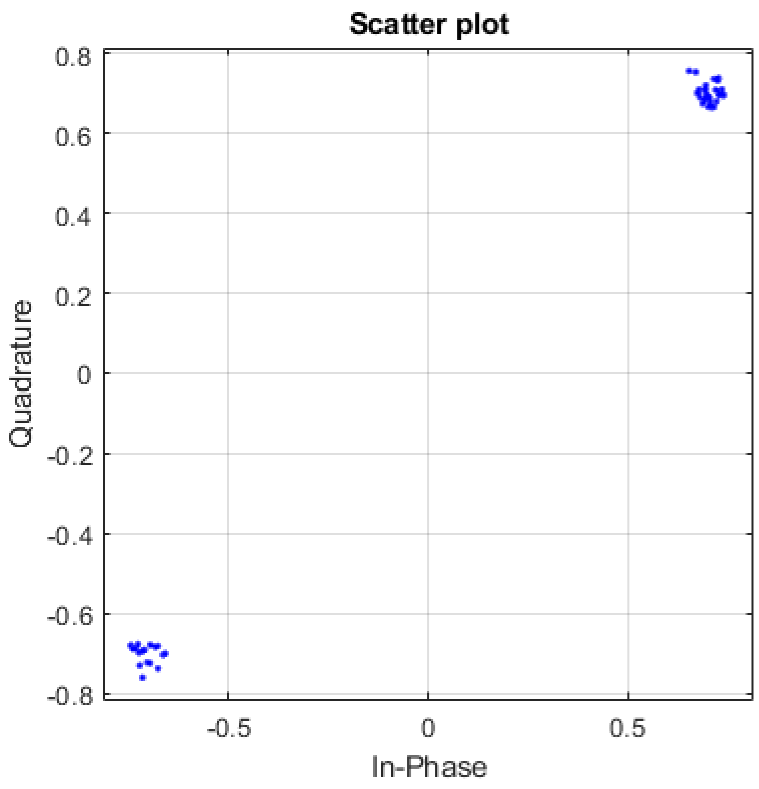
Verify that the L-SIG passed the failure checks.

```
failCheck
```

```
failCheck =

  logical

   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.

N_S is proportional to the channel bandwidth.

`ChannelBandwidth`	N_S
`'CBW5'`, `'CBW10'`, `'CBW20'`	`80`
`'CBW40'`	`160`
`'CBW80'`	`320`
`'CBW160'`	`640`

Data Types: `double`

`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_{ST}
`'CBW5'`, `'CBW10'`, `'CBW20'`	52
`'CBW40'`	104
`'CBW80'`	208
`'CBW160'`	416

Data Types: `double`

`noiseVarEst` — Noise variance estimate
nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar.

Data Types: `double`

`cbw` — Channel bandwidth
`'CBW5'` | `'CBW10'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW5'`, `'CBW10'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Example: `'CBW80'` corresponds to a channel bandwidth of 80 MHz

Data Types: `char` | `string`

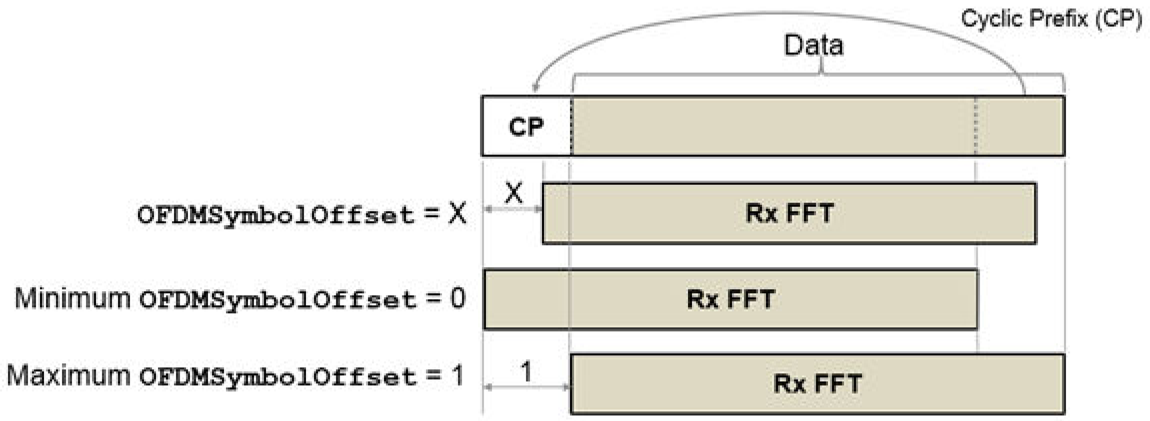
`cfgRec` — Algorithm parameters
`wlanRecoveryConfig` object

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` | `string`

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

Pilot phase tracking, specified as `'PreEQ'` or `'None'`.

- `'PreEQ'` — Enables pilot phase tracking, which is performed before any equalization operation.
- `'None'` — Pilot phase tracking does not occur.

Data Types: `char` | `string`

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`

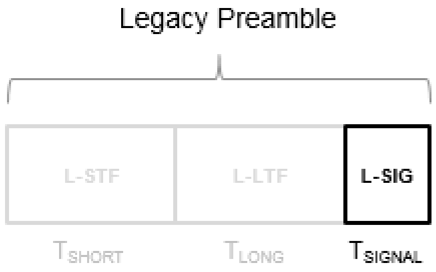
cpe — Common phase error
column vector

Common phase error in radians, returned as a scalar.

Definitions

L-SIG

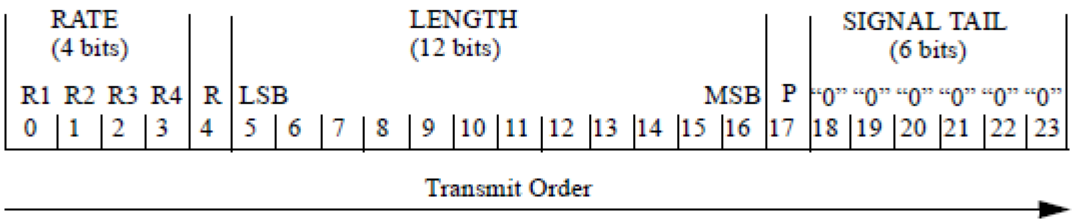
The legacy signal (L-SIG) field is the third field of the 802.11 OFDM PLCP legacy preamble. It consists of 24 bits that contain rate, length, and parity information. The L-SIG is a component of VHT, HT, and non-HT PPDUs. It is transmitted using BPSK modulation with rate 1/2 binary convolutional coding (BCC).



The L-SIG is one OFDM symbol with a duration that varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier frequency spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) period (T_{FFT} = 1 / Δ_F)	Guard Interval (GI) Duration (T_{GI} = T_{FFT} / 4)	L-SIG duration (T_{SIGNAL} = T_{GI} + T_{FFT})
20, 40, 80, and 160	312.5	3.2 μs	0.8 μs	4 μs
10	156.25	6.4 μs	1.6 μs	8 μs
5	78.125	12.8 μs	3.2 μs	16 μs

The L-SIG 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 (Mb/s)		
			20 MHz channel bandwidth	10 MHz channel bandwidth	5 MHz channel bandwidth
1101	BPSK	1/2	6	3	1.5
1111	BPSK	3/4	9	4.5	2.25
0101	QPSK	1/2	12	6	3
0111	QPSK	3/4	18	9	4.5
1001	16-QAM	1/2	24	12	6
1011	16-QAM	3/4	36	18	9
0001	64-QAM	2/3	48	24	12
0011	64-QAM	3/4	54	27	13.5

  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.

- For the HT-mixed format, IEEE Std 802.11-2012, Section 20.3.9.4.3 describes HT-SIG bit settings.
- For the VHT 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.

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-259[17] time-domain waveform using the specified configuration object.

Examples

Generate L-STF Waveform

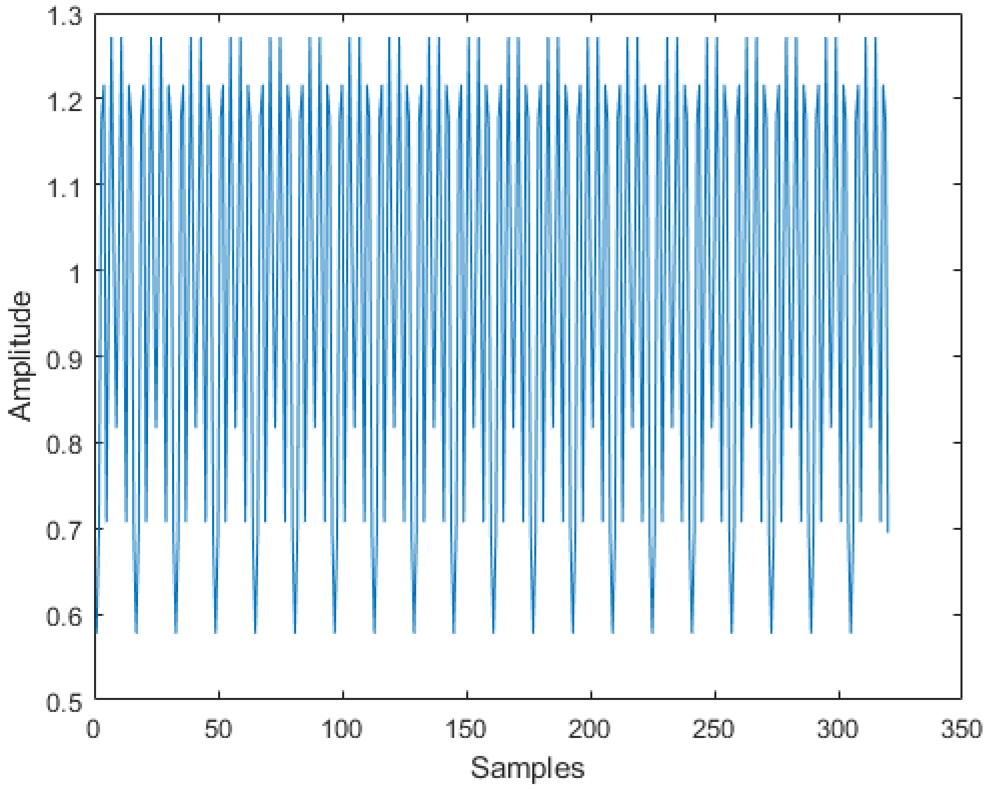
Generate the L-STF waveform for a 40 MHz single antenna VHT packet.

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 See note “1” on page 1-258	`ChannelBandwidth`, `NumTransmitAntennas`
Note: 1 For non-HT format, when channel bandwidth is 5 MHz or 10 MHz, `NumTransmitAntennas` is not applicable because only one transmit antenna is permitted.	

Example: `wlanVHTConfig`

Output Arguments

`y` — L-STF time-domain waveform

matrix

(“L-STF” on page 1-259) 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.

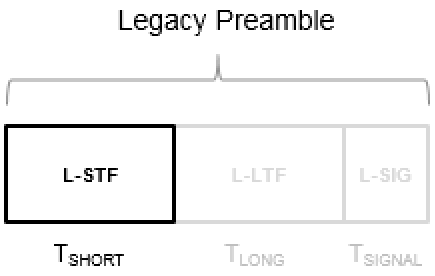
`ChannelBandwidth`	N_S
`'CBW5'`, `'CBW10'`, `'CBW20'`	`160`
`'CBW40'`	`320`
`'CBW80'`	`640`
`'CBW160'`	`1280`

Data Types: `double`
Complex Number Support: Yes

Definitions

L-STF

The legacy short training field (L-STF) is the first field of the 802.11 OFDM PLCP legacy preamble. The L-STF is a component of VHT, HT, and non-HT PPDUs.



The L-STF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period (T_{FFT} = 1 / Δ_F)	L-STF Duration (T_{SHORT} = 10 × T_{FFT} / 4)
20, 40, 80, and 160	312.5	3.2 µs	8 µs
10	156.25	6.4 µs	16 µs
5	78.125	12.8 µs	32 µs

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 per 20 MHz channel bandwidth segment. For 5 MHz, 10 MHz, and 20 MHz bandwidths, the number of channel bandwidths segments is 1.

Algorithms

The “L-STF” on page 1-259 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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-268.

For non-HT, subcarrier spacing and subcarrier allocation have channel bandwidth dependencies. For more information, see “OFDM PLCP Timing Parameters” on page 1-266.

`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. After creating the object update the number of transmit antennas.

```
cfgNHT = wlanNonHTConfig
```

```
cfgNHT =
  wlanNonHTConfig with properties:

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

Update the number of transmit antennas to two.

```
cfgNHT.NumTransmitAntennas = 2
```

```
cfgNHT = 

  wlanNonHTConfig with properties:

             Modulation: 'OFDM'
       ChannelBandwidth: 'CBW20'
                    MCS: 0
             PSDULength: 1000
    NumTransmitAntennas: 2
```

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'
       ChannelBandwidth: 'CBW20'
                    MCS: 0
             PSDULength: 2048
```

```
    NumTransmitAntennas: 1
```

Create Non-HT Format Configuration Object for DSSS Modulation

Create a `wlanNonHTConfig` object for DSSS operation for a PSDU length of 2048 bytes.

```
cfgNHT = wlanNonHTConfig('Modulation','DSSS','PSDULength',2048)
```

```
cfgNHT = 

  wlanNonHTConfig with properties:

      Modulation: 'DSSS'
        DataRate: '1Mbps'
    LockedClocks: 1
      PSDULength: 2048
```

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` | `string`

`ChannelBandwidth` — **Channel bandwidth**
`'CBW20'` (default) | `'CBW10'` | `'CBW5'`

Channel bandwidth in MHz for OFDM, specified as `'CBW20'`, `'CBW10'`, or `'CBW5'`. The default value of `'CBW20'` sets the channel bandwidth to 20 MHz.

When channel bandwidth is 5 MHz or 10 MHz, only one transmit antenna is permitted and `NumTransmitAntennas` is not applicable.

Data Types: `char` | `string`

`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	Coded bits per subcarrier (N_{BPSC})	Coded bits per OFDM symbol (N_{CBPS})	Data bits per OFDM symbol (N_{DBPS})	Data Rate (Mbps)		
						20 MHz channel bandwidth	10 MHz channel bandwidth	5 MHz channel bandwidth
0	BPSK	1/2	1	48	24	6	3	1.5
1	BPSK	3/4	1	48	36	9	4.5	2.25
2	QPSK	1/2	2	96	48	12	6	3
3	QPSK	3/4	2	96	72	18	9	4.5
4	16QAM	1/2	4	192	96	24	12	6
5	16QAM	3/4	4	192	144	36	18	9
6	64QAM	2/3	6	288	192	48	24	12
7	64QAM	3/4	6	288	216	54	27	13.5

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` | `string`

`Preamble` — DSSS modulation preamble type

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

DSSS modulation preamble type, 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` | `string`

`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 for OFDM, specified as a scalar integer from 1 to 8.

When channel bandwidth is 5 MHz or 10 MHz, `NumTransmitAntennas` is not applicable because only one transmit antenna is permitted.

Data Types: `double`

Output Arguments

`cfgNonHT` — Non-HT PPDU configuration
`wlanNonHTConfig` object

Non-HT "PPDU" on page 1-268 configuration, returned as a `wlanNonHTConfig` object. The properties of `cfgNonHT` are specified in wlanNonHTConfig.

Definitions

OFDM PLCP Timing Parameters

IEEE Std 802.11™-2012 [1], Section 18[18] specifies OFDM PLCP 20 MHz, 10 MHz, and 5 MHz channel bandwidth operation.

Timing parameters associated with the OFDM PLCP are listed in IEEE Std 802.11™-2012 [1], Table 18-5.

Parameter	Value	20 MHz channel bandwidth	10 MHz channel bandwidth	5 MHz channel bandwidth
N_{SD}: Number of data subcarriers	48	48	48	48
N_{SP}: Number of pilot subcarriers	4	4	4	4
N_{ST}: Number of subcarriers, total	$N_{SD} + N_{SP}$	52	52	52
Δ_F: Subcarrier frequency spacing	(Channel BW in MHz) / 64	0.3125 MHz (= 20 / 64)	0.15625 MHz (= 10 / 64)	0.078125 MHz (= 5 / 64)
T_{FFT}: Inverse Fast Fourier Transform (IFFT) / Fast Fourier Transform (FFT) period	1 / Δ_F	3.2 µs	6.4 µs	12.8 µs
$T_{PREAMBLE}$: PLCP preamble duration	$T_{SHORT} + T_{LONG}$	16 µs	32 µs	64 µs
T_{SIGNAL}: Duration of the L-SIG symbol	$T_{GI} + T_{FFT}$	4.0 µs	8.0 µs	16.0 µs
T_{GI}: GI duration	$T_{FFT}/4$	0.8 µs	1.6µs	3.2 µs
T_{GI2}: Training symbol GI duration	$T_{FFT}/2$	1.6 µs	3.2µs	6.4 µs
T_{SYM}: Symbol interval	$T_{GI} + T_{FFT}$	4 µs	8 µs	16 µs

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

Parameter	Value	20 MHz channel bandwidth	10 MHz channel bandwidth	5 MHz channel bandwidth
T_{SHORT}: L-STF duration	$10 \times T_{FFT} /4$	8 μs	16 μs	32 μs
T_{LONG}: L-LTF duration	$T_{GI2} + 2 \times T_{FFT}$	8 μs	16 μs	32 μs

Note The standard refers to operation at:

- 10 MHz as “half-clocked”.
- 5 MHz as “quarter-clocked”.

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanDMGConfig` | `wlanHTConfig` | `wlanS1GConfig` | `wlanVHTConfig` | `wlanWaveformGenerator`

Topics

“Packet Size and Duration Dependencies”

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 1-274[19] time-domain waveform for the input “PSDU” on page 1-274 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)
```

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

```
ans =

        4480           1
```

Input Arguments

`psdu` — PLCP service data unit
vector

PLCP service data unit (“PSDU” on page 1-274), specified as an N_{bits}-by-1 vector, where N_{bits} = **`PSDULength`** × 8. “PSDU” on page 1-274 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

`ChannelBandwidth` — Channel bandwidth
`'CBW20'` (default) | `'CBW10'` | `'CBW5'`

Channel bandwidth in MHz for OFDM, specified as `'CBW20'`, `'CBW10'`, or `'CBW5'`. The default value of `'CBW20'` sets the channel bandwidth to 20 MHz.

When channel bandwidth is 5 MHz or 10 MHz, only one transmit antenna is permitted and `NumTransmitAntennas` is not applicable.

Data Types: `char` | `string`

`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	Coded bits per subcarrier (N_{BPSC})	Coded bits per OFDM symbol (N_{CBPS})	Data bits per OFDM symbol (N_{DBPS})	Data Rate (Mbps)		
						20 MHz channel bandwidth	10 MHz channel bandwidth	5 MHz channel bandwidth
0	BPSK	1/2	1	48	24	6	3	1.5
1	BPSK	3/4	1	48	36	9	4.5	2.25
2	QPSK	1/2	2	96	48	12	6	3
3	QPSK	3/4	2	96	72	18	9	4.5
4	16QAM	1/2	4	192	96	24	12	6
5	16QAM	3/4	4	192	144	36	18	9
6	64QAM	2/3	6	288	192	48	24	12
7	64QAM	3/4	6	288	216	54	27	13.5

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 for OFDM, specified as a scalar integer from 1 to 8.

When channel bandwidth is 5 MHz or 10 MHz, `NumTransmitAntennas` is not applicable because only one transmit antenna is permitted.

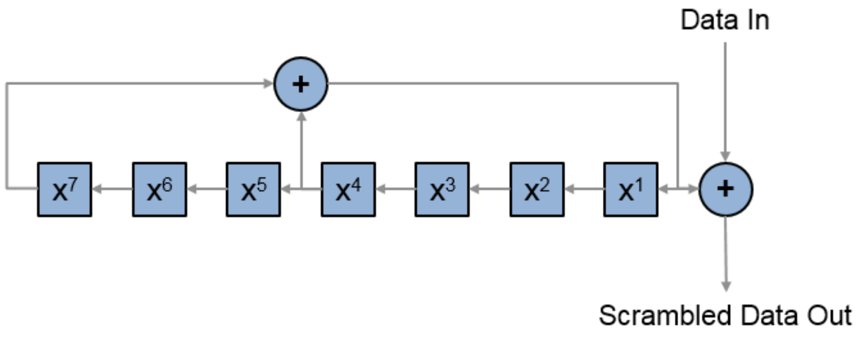
Data Types: `double`

`scramInit` — Scrambler initialization state

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

Scrambler initialization state for each packet generated, specified as an integer from 1 to 127 or as the corresponding binary 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.

The scrambler initialization used on the transmission data follows the process described in IEEE Std 802.11-2012, Section 18.3.5.5 and IEEE Std 802.11ad-2012, Section 21.3.9. The header and data fields that follow the scrambler initialization field (including data padding bits) are scrambled by XORing each bit with a length-127 periodic sequence generated by the polynomial $S(x) = x^7+x^4+1$. The octets of the PSDU (Physical Layer Service Data Unit) are placed into a bit stream, and within each octet, bit 0 (LSB) is first and bit 7 (MSB) is last. The generation of the sequence and the XOR operation are shown in this figure:



Conversion from integer to bits uses left-MSB orientation. For the initialization of the scrambler with decimal `1`, the bits are mapped to the elements shown.

Element	X^7	X^6	X^5	X^4	X^3	X^2	X^1
Bit Value	0	0	0	0	0	0	1

To generate the bit stream equivalent to a decimal, use `de2bi`. For example, for decimal `1`:

```
de2bi(1,7,'left-msb')
ans =

     0     0     0     0     0     0     1
```

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

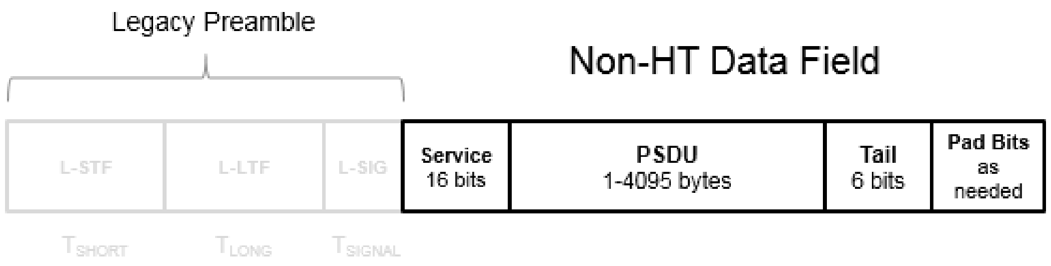
Definitions

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.

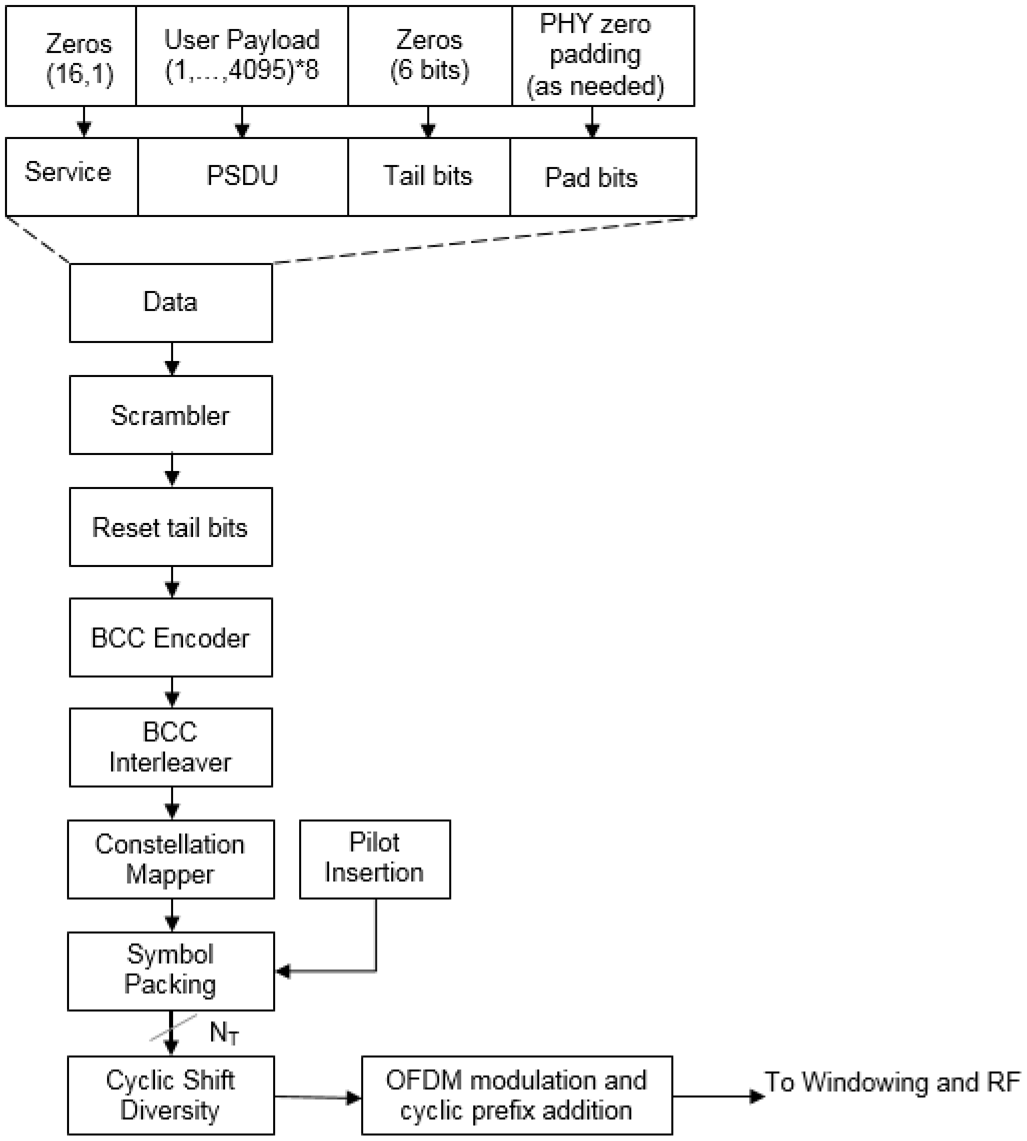


- **Service field** — Contains 16 zeros to initialize the data scrambler.
- **PSDU** — Variable-length field containing the PLCP service data unit (PSDU).
- **Tail** — Tail bits required to terminate a convolutional code. The field uses six zeros for the single encoding stream.
- **Pad Bits** — Variable-length 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 1-274 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 1-274 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 1-274 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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-285[20] bits, given received signal `rxSig`, channel estimate data `chEst`, noise variance estimate `noiseVarEst`, and `wlanNonHTConfig` object `cfg`.

Note This function only supports data recovery for OFDM modulation.

`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

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

Recover Non-HT Data Bits

Create a non-HT configuration object having a PSDU length of 2048 bytes. Generate the corresponding data sequence.

```
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`

`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`

`noiseVarEst` — Noise variance estimate

nonnegative scalar

Estimate of the noise variance, specified as a nonnegative scalar.

Example: 0.7071

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	Coded bits per subcarrier (N_{BPSC})	Coded bits per OFDM symbol (N_{CBPS})	Data bits per OFDM symbol (N_{DBPS})	Data Rate (Mbps)		
						20 MHz channel bandwidth	10 MHz channel bandwidth	5 MHz channel bandwidth
0	BPSK	1/2	1	48	24	6	3	1.5
1	BPSK	3/4	1	48	36	9	4.5	2.25
2	QPSK	1/2	2	96	48	12	6	3
3	QPSK	3/4	2	96	72	18	9	4.5
4	16QAM	1/2	4	192	96	24	12	6
5	16QAM	3/4	4	192	144	36	18	9
6	64QAM	2/3	6	288	192	48	24	12
7	64QAM	3/4	6	288	216	54	27	13.5

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.

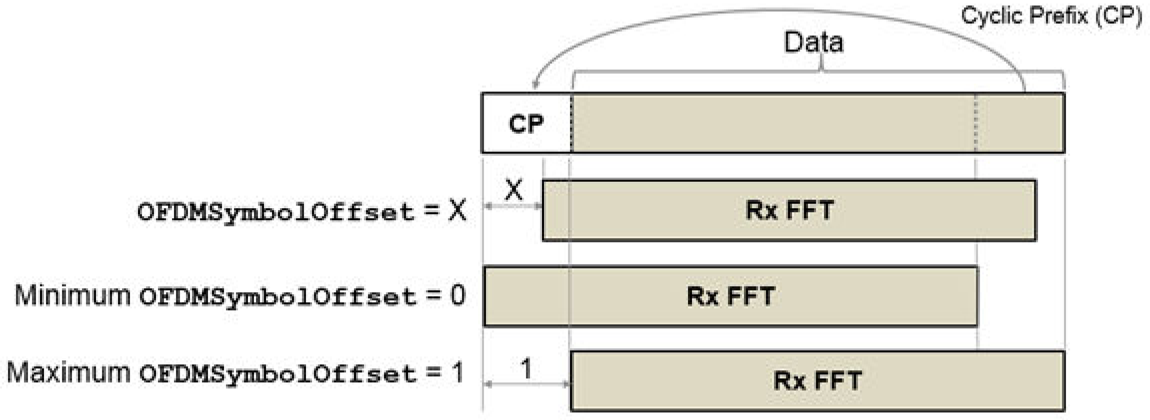
Data Types: `double`

`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` | `string`

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

Pilot phase tracking, specified as `'PreEQ'` or `'None'`.

- `'PreEQ'` — Enables pilot phase tracking, which is performed before any equalization operation.
- `'None'` — Pilot phase tracking does not occur.

Data Types: `char` | `string`

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 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`

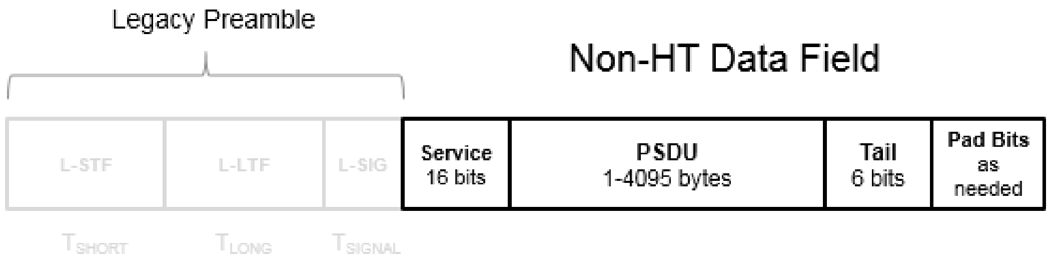
`cpe` **— Common phase error**
column vector

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 1-285.

Definitions

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanNonHTConfig` | `wlanNonHTData` | `wlanRecoveryConfig`

Introduced in R2015b

wlanPacketDetect

OFDM packet detection using L-STF

Syntax

```
startOffset = wlanPacketDetect(rxSig,cbw)
startOffset = wlanPacketDetect(rxSig,cbw,offset)
startOffset = wlanPacketDetect(rxSig,cbw,offset,threshold)
[startOffset,M] = wlanPacketDetect(___)
```

Description

`startOffset = wlanPacketDetect(rxSig,cbw)` returns the offset from the start of the input waveform to the start of the detected preamble, given a received time-domain waveform and the channel bandwidth. For more information, see “Packet Detection Processing” on page 1-294.

Note This function supports packet detection of OFDM modulated signals only.

`startOffset = wlanPacketDetect(rxSig,cbw,offset)` specifies an offset from the start of the received waveform and indicates where the autocorrelation processing begins. The returned `startOffset` is relative to the input `offset`.

`startOffset = wlanPacketDetect(rxSig,cbw,offset,threshold)` specifies the threshold which the decision statistic must meet or exceed to detect a packet.

`[startOffset,M] = wlanPacketDetect(___)` also returns the decision statistics of the packet detection algorithm for the received time-domain waveform, using any of the input arguments in the previous syntaxes.

Examples

Detect 802.11n Packet

Detect a received 802.11n packet at a signal-to-noise ratio (SNR) of 20 dB.

Create an HT configuration object and TGn channel object. Generate a transmit waveform.

```
cfgHT = wlanHTConfig;
tgn = wlanTGnChannel('LargeScaleFadingEffect','None');

txWaveform = wlanWaveformGenerator([1;0;0;1],cfgHT);
```

Pass the waveform through the TGn channel with an SNR of 20 dB. Detect the start of the packet.

```
snr = 20;
fadedSig = tgn(txWaveform);
rxWaveform = awgn(fadedSig,snr,0);

startOffset = wlanPacketDetect(rxWaveform,cfgHT.ChannelBandwidth)
```

```
startOffset =

     0
```

The packet is detected at the first sample of the received waveform, specifically the returned `startOffset` indicates an offset of zero samples from the start of the received waveform.

Detect Delayed 802.11ac Packet

Detect a received 802.11ac packet that has been delayed. Specify an offset of 25 to begin the autocorrelation process.

Create an VHT configuration object and generate the transmit waveform.

```
cfgVHT = wlanVHTConfig;

txWaveform = wlanWaveformGenerator([1;0;0;1],cfgVHT,...
    'WindowTransitionTime',0);
```

Delay the signal by appending zeros at the start. Specify an offset of 25 for the beginning of autocorrelation processing. Detect the start of the packet.

```
rxWaveform = [zeros(100,1);txWaveform];
offset = 25;
startOffset = wlanPacketDetect(rxWaveform,cfgVHT.ChannelBandwidth,offset)
```

```
startOffset =

    48
```

Calculate the detected packet offset by adding the returned `startOffset` and the input `offset`.

```
pktOffset = offset + startOffset
```

```
pktOffset =

    73
```

The offset from the first sample of the received waveform to the start of the packet is detected to be 73 samples. This coarse approximation of the packet-start offset is useful for determining where to begin autocorrelation for the first packet and for subsequent packets when a multipacket waveform is transmitted.

Detect Delayed 802.11a Packet

Detect a received 802.11a packet that has been delayed. No channel impairments are added. Set the input offset to 5 and use a threshold setting very close to 1.

Create an non-HT configuration object. Generate the transmit waveform.

```
cfgNonHT = wlanNonHTConfig;

txWaveform = wlanWaveformGenerator([1;0;0;1],cfgNonHT,...
    'WindowTransitionTime',0);
```

Delay the signal by appending zeros at the start. Set an initial offset of 5 and a threshold very close to 1. Detect the delayed packet.

```
rxWaveform = [zeros(20,1);txWaveform];

offset = 5;
threshold = 1-10*eps;
startOffset = wlanPacketDetect(rxWaveform,...
    cfgNonHT.ChannelBandwidth,offset,threshold)
```

```
startOffset =

    15
```

Calculate the detected packet offset by adding the returned `startOffset` and the input `offset`.

```
totalOffset = offset + startOffset
```

```
totalOffset =

    20
```

Using a threshold close to 1 and an undistorted received waveform increases the accuracy of the packet detect location. The detected offset from the first sample of the received waveform to the start of the packet is determined to be 20 samples.

Generate WLAN Packet Decision Statistics

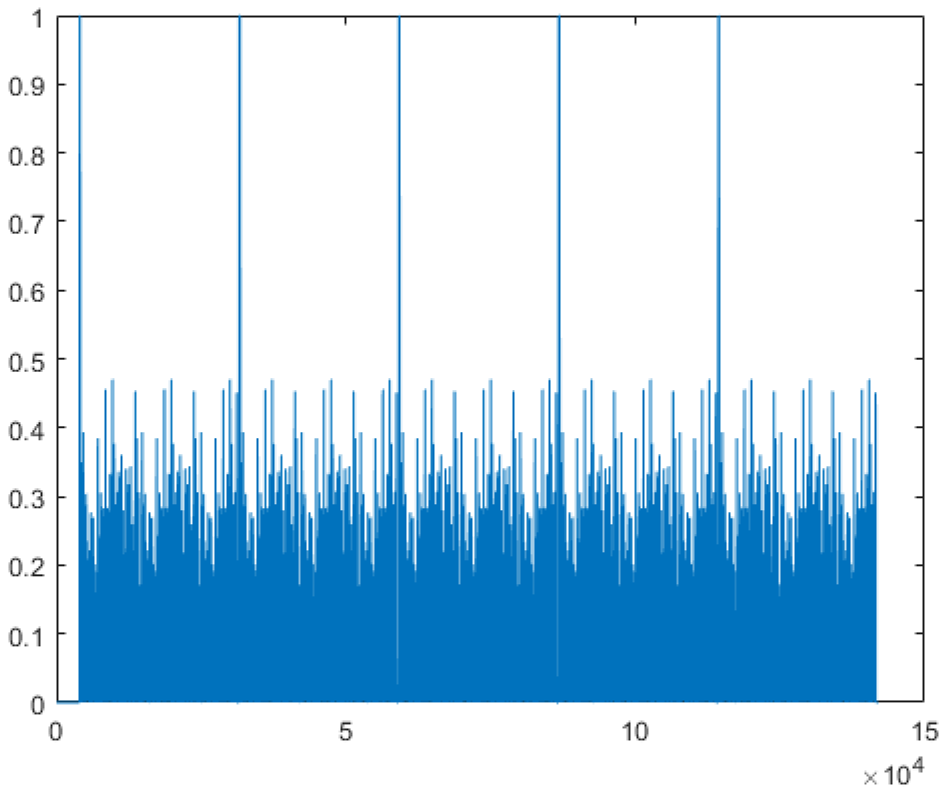
Return the decision statistics of a WLAN waveform that consists of five 802.11a packets.

Create a non-HT configuration object and a five-packet waveform. Delay the waveform by 4000 samples.

```
cfgNonHT = wlanNonHTConfig;
txWaveform = wlanWaveformGenerator([1;0;0;1],cfgNonHT, ...
    'NumPackets',5,'IdleTime',20e-6);

rxWaveform = [zeros(4000,1);txWaveform];
```

Setting the threshold input to 1, generates packet decision statistics for the entire waveform and suppresses the `startOffset` output. Plot the decision statistics, `M`.

```
offset = 0;
threshold = 1;
[startOffset,M] = wlanPacketDetect(rxWaveform,cfgNonHT.ChannelBandwidth,...
    offset,threshold);
plot(M)
```



Since `threshold` = 1, the decision statistics for the entire waveform are included in the output `M`. The decision statistics show five peaks. The peaks corresponds to the first sample of each packet detected. View `startOffset`.

```
startOffset
```

```
startOffset =
     []
```

The returned `startOffset` is empty because `threshold` was set to 1.

Input Arguments

`rxSig` — Received time-domain signal

matrix

Received time-domain signal, specified as an N_S-by-N_R matrix. N_R is the number of receive antennas. N_S represents the number of time-domain samples in the received signal.

Data Types: `double`
Complex Number Support: Yes

`cbw` — Channel bandwidth

`'CBW5'` | `'CBW10'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW5'`, `'CBW10'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Data Types: `char` | `string`

`offset` — Number of samples offset

0 (default) | nonnegative integer

Number of samples offset from the beginning of the received waveform, specified as a nonnegative integer. `offset` defines the starting sample for the autocorrelation process. `offset` is useful for advancing through and detecting the `startOffset` sample for successive packets in multipacket waveforms.

Note Since the packet detection searches forward in time, the first packet will not be detected if the initial setting for `offset` is beyond the first “L-STF” on page 1-293.

Data Types: `double`

`threshold` — Decision statistic threshold
0.5 (default) | real scalar | from >0 to 1

Decision statistic threshold that must be met or exceeded to detect a packet, specified as a real scalar greater than 0 and less than or equal to 1.

Data Types: `double`

Output Arguments

`startOffset` — Number of samples offset to the start of packet
nonnegative integer | [ ]

Number of samples offset to the start of packet, returned as a nonnegative integer. This value, shifted by `offset`, indicates the detected start of a packet from the first sample of `rxSig`.

- An empty value, `[ ]`, is returned if no packet is detected or if `threshold` is set to 1.
- Zero is returned if there is no delay, specifically the packet is detected at the first sample of the waveform.

`M` — Decision statistics
vector

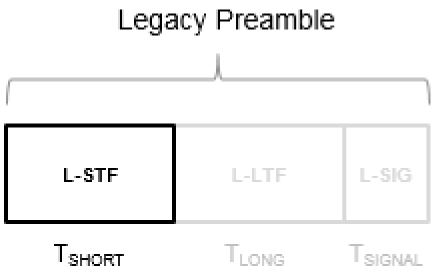
Decision statistics based on autocorrelation of the input waveform, returned as an N-by-1 real vector. The length of N depends on the starting location of the autocorrelation process and the number of samples until a packet is detected. When `threshold` is set to 1, `M` returns the decision statistics of the full waveform and `startOffset` returns empty.

For more information, see “Packet Detection Processing” on page 1-294.

Definitions

L-STF

The legacy short training field (L-STF) is the first field of the 802.11 OFDM PLCP legacy preamble. The L-STF is a component of VHT, HT, and non-HT PPDUs.



The L-STF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	L-STF Duration ($T_{SHORT} = 10 \times T_{FFT} / 4$)
20, 40, 80, and 160	312.5	3.2 μs	8 μs
10	156.25	6.4 μs	16 μs
5	78.125	12.8 μs	32 μs

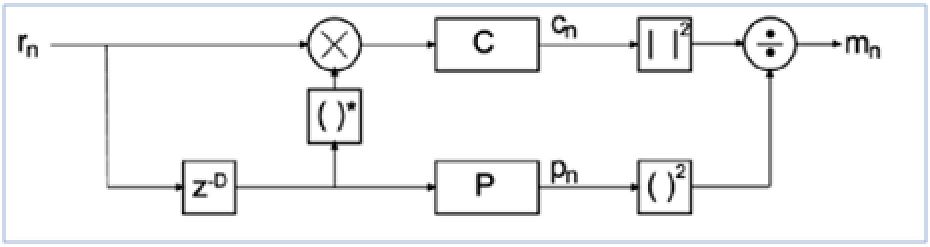
Because the sequence has good correlation properties, it is used for start-of-packet detection, for coarse frequency correction, and for setting the AGC. The sequence uses 12 of the 52 subcarriers that are available per 20 MHz channel bandwidth segment. For 5 MHz, 10 MHz, and 20 MHz bandwidths, the number of channel bandwidths segments is 1.

Algorithms

Packet Detection Processing

The packet detection algorithm is implemented as a double sliding window as described in OFDM Wireless LANs [1], Chapter 2. The autocorrelation of “L-STF” on page 1-293 short training symbols is used to return an estimated packet-start offset. In a robust system, the next stage will refine this estimate with symbol timing detection using the L-LTF.

As shown in the figure, the received signal, r_n, is delayed then correlated in two sliding windows independently. The packet detection processing output provides decision statistics (m_n) of the received waveform.



- Window C autocorrelates between the received signal and the delayed version, c_n.

$$c_n = \sum_{l=1}^{N_R} \sum_{K=0}^{D-1} r_{n+k,l} r^{*}_{n+k+D,l}$$

- Window P calculates the energy received in the autocorrelation window, p_n.

$$p_n = \sum_{l=1}^{N_R} \sum_{k=0}^{D-1} | r_{n+k+D,l} |^2$$

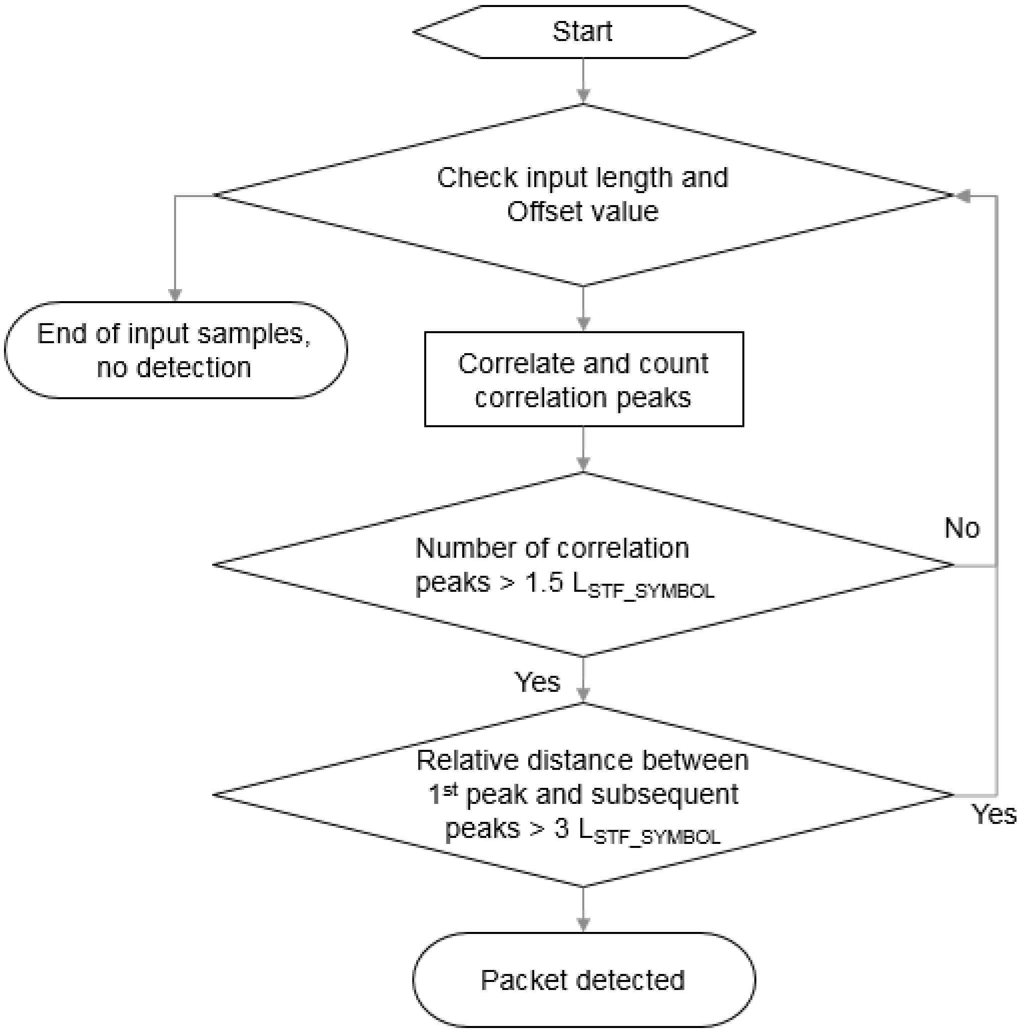
- The decision statistics, m_n, normalize the autocorrelation by p_n so that the decision statistic is not dependent on the absolute received power level.

$$m_n = \frac{| c_n |^2}{(p_n)^2}$$

  The decision statistics provide visual information resulting from the autocorrelation process that is useful when selecting the appropriate threshold value for the input waveform. The recommended default value of 0.5 for `threshold` favors false detections over missed detections considering a range of SNRs and various antenna configurations.

In the sliding window calculations, D is the period of the “L-STF” on page 1-293 short training symbols and N_R is the number of receive antennas.

Packet detection processing follows this flow chart:


Start
Check input length and Offset value
End of input samples, no detection
Correlate and count correlation peaks
Number of correlation peaks > 1.5 L_STF_SYMBOL
No
Yes
Relative distance between 1st peak and subsequent peaks > 3 L_STF_SYMBOL
Yes
Packet detected

$L_{\text{STF\_SYMBOL}}$ is the length of an "L-STF" on page 1-293 symbol.

Note This function supports packet detection of OFDM modulated signals only.

References

[1] Terry, J., and J. Heiskala. *OFDM Wireless LANs: A Theoretical and Practical Guide*. Indianapolis, IN: Sams, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanCoarseCFOEstimate` | `wlanFieldIndices`

Introduced in R2016b

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: 0.5000
          EqualizationMethod: 'ZF'
          PilotPhaseTracking: 'PreEQ'
   MaximumLDPCIterationCount: 12
            EarlyTermination: 0
```

Input Arguments

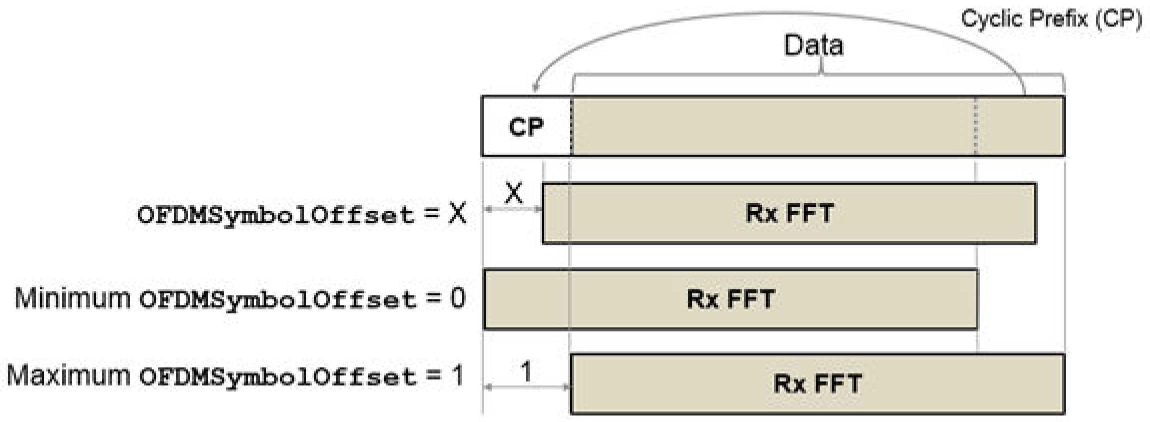
Name-Value Pair Arguments

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` | `string`

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

Pilot phase tracking, specified as `'PreEQ'` or `'None'`.

- `'PreEQ'` — Enables pilot phase tracking, which is performed before any equalization operation.
- `'None'` — Pilot phase tracking does not occur.

Data Types: `char` | `string`

`MaximumLDPCIterationCount` — Maximum number of decoding iterations in LDPC
12 (default) | positive scalar integer

Maximum number of decoding iterations in LDPC, specified as a positive scalar integer. This parameter is applicable when channel coding is set to LDPC. For information on channel coding options, see `wlanVHTConfig` or `wlanHTConfig` for 802.11 format of interest.

Data Types: `double`

`EarlyTermination` — Enable early termination of LDPC decoding
`false` (default) | `true`

Enable early termination of LDPC decoding, specified as a logical. This parameter is applicable when channel coding is set to LDPC.

- When set to `false`, LDPC decoding completes the number of iterations specified by `MaximumLDPCIterationCount`, regardless of parity check status.
- When set to `true`, LDPC decoding terminates when all parity-checks are satisfied.

For information on channel coding options, see `wlanVHTConfig` or `wlanHTConfig` for 802.11 format of interest.

Output Arguments

`cfgRec` — Data recovery configuration
`wlanRecoveryConfig` object

Data recovery configuration, returned as a `wlanRecoveryConfig` object. The properties of `cfgRec` are specified in wlanRecoveryConfig.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanHTDataRecover` | `wlanHTSIGRecover` | `wlanLSIGRecover` | `wlanNonHTDataRecover` | `wlanVHTDataRecover` | `wlanVHTSIGARecover` | `wlanVHTSIGBRecover`

Introduced in R2015b

wlanS1GConfig

Create S1G format configuration object

Syntax

```
cfgS1G = wlanS1GConfig
cfgS1G = wlanS1GConfig(Name,Value)
```

Description

`cfgS1G = wlanS1GConfig` creates a configuration object that initializes parameters for an IEEE 802.11 sub 1 GHz (S1G) format "PPDU" on page 1-313.

`cfgS1G = wlanS1GConfig(Name,Value)` creates an S1G 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 wlanS1GConfig Object for Single User

Create an S1G configuration object with default settings for a single user. Override the default by specifying a 4 MHz channel bandwidth and short preamble configuration.

```
cfgS1G = wlanS1GConfig;
cfgS1G.ChannelBandwidth = 'CBW4';
cfgS1G.Preamble = 'Short';
cfgS1G
```

```
cfgS1G =
  wlanS1GConfig with properties:

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

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

Create wlanS1GConfig Object for Two Users

Create an S1G configuration object that assigns a 2 MHz bandwidth and two users. Use a combination of Name,Value pairs and in-line initialization to change default settings. In vector-valued properties, each element applies to a specific user.

```
cfgMU = wlanS1GConfig('ChannelBandwidth','CBW2', ...
    'Preamble','Long', ...
    'NumUsers',2, ...
    'GroupID',2, ...
    'NumTransmitAntennas', 2);
cfgMU.NumSpaceTimeStreams = [1 1];
cfgMU.MCS = [4 8];
cfgMU.APEPLength = [1024 2048];
cfgMU
```

```
cfgMU =
  wlanS1GConfig with properties:

       ChannelBandwidth: 'CBW2'
               Preamble: 'Long'
               NumUsers: 2
          UserPositions: [0 1]
    NumTransmitAntennas: 2
    NumSpaceTimeStreams: [1 1]
         SpatialMapping: 'Direct'
                    MCS: [4 8]
             APEPLength: [1024 2048]
          GuardInterval: 'Long'
                GroupID: 2
        TravelingPilots: 0
     ResponseIndication: 'None'

   Read-only properties:
          ChannelCoding: 'BCC'
             PSDULength: [1031 2065]
```

`NumUsers` is set to 2 and the user-dependent properties are two-element vectors.

Create wlanS1GConfig Object and Return Packet Format

Create an S1G configuration object with default settings for a single user and change the default property settings by using dot notation. Use the `packetFormat` object function to access the S1G packet format of the object.

Create an S1G configuration object with default settings. By default, the configuration object creates properties to model the short S1G packet format.

```
cfgS1G = wlanS1GConfig;
packetFormat(cfgS1G)
```

```
ans = 
'S1G-Short'
```

Modify the defaults by using the dot notation to specify a long preamble.

```
cfgS1G.Preamble = 'Long';
packetFormat(cfgS1G)
```

```
ans = 
'S1G-Long'
```

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','CBW4','NumUsers',2` specifies a channel bandwidth of 4 MHz and two users for the S1G format packet.

`ChannelBandwidth` — Channel bandwidth
`'CBW2'` (default) | `'CBW1'` | `'CBW4'` | `'CBW8'` | `'CBW16'`

Channel bandwidth, specified as `'CBW1'`, `'CBW2'`, `'CBW4'`, `'CBW8'`, or `'CBW16'`. If the transmission has multiple users, the same channel bandwidth is applied to all users.

Example: `'CBW16'` sets the channel bandwidth to 16 MHz.

Data Types: `char` | `string`

`Preamble` — Preamble type
`'Short'` (default) | `'Long'`

Preamble type, specified as `'Short'` or `'Long'`. This property applies only when `ChannelBandwidth` is not `'CBW1'`.

Data Types: `char` | `string`

`NumUsers` — Number of users
1 (default) | 2 | 3 | 4

Number of users, specified as 1, 2, 3, or 4. (N_{Users})

Data Types: `double`

`UserPositions` — Position of users
[0 1] (default) | row vector of integers from 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` > 1.

Example: `[0 2 3]` indicates positions for three users, where the first user occupies position 0, the second user occupies position 2, and the third user occupies position 3.

Data Types: `double`

`NumTransmitAntennas` — Number of transmit antennas
1 (default) | integer from 1 to 4

Number of transmit antennas, specified as a scalar integer from 1 to 4.

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams
1 (default) | integer from 1 to 4 | 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. (N_{sts})

- For a single user, the number of space-time streams is an integer scalar from 1 to 4.
- For multiple users, the number of space-time streams is a 1-by-N_{Users} vector of integers from 1 to 4, where $N_{\text{Users}} \leq 4$. The sum total of space-time streams for all users, $N_{\text{sts\_Total}}$, must not exceed four.

Example: `[1 1 2]` indicates number of space-time streams for three users, where the first user gets 1 space-time stream, the second user gets 1 space-time stream, and the third user gets 2 space-time streams. The total number of space-time streams assigned is 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 of `'Direct'` applies when `NumTransmitAntennas` and `NumSpaceTimeStreams` are equal.

Data Types: `char` | `string`

`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})
`'CBW1'`	`26`	`24`	`2`
`'CBW2'`	`56`	`52`	`4`
`'CBW4'`	`114`	`108`	`6`
`'CBW8'`	`242`	`234`	`8`
`'CBW16'`	`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` — Enable beamforming in a long preamble packet

`true` (default) | `false`

Enable beamforming in a long preamble packet, specified as a logical. Beamforming is performed when this setting is `true`. This property applies for a long preamble (`Preamble` = `'Long'`) with `NumUsers` = 1 and `SpatialMapping = 'Custom'`. The `SpatialMappingMatrix` property specifies the beamforming steering matrix.

Data Types: `logical`

`STBC` — Enable space-time block coding
`false` (default) | `true`

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 10 | 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 10.
- For multiple users, MCS is a 1-by-N_{Users} vector of integers or a scalar with values from 0 to 10, where $N_{\text{Users}} \leq 4$.

MCS	Modulation	Coding Rate	Comment
0	`BPSK`	`1/2`	
1	`QPSK`	`1/2`	
2	`QPSK`	`3/4`	
3	`16QAM`	`1/2`	
4	`16QAM`	`3/4`	
5	`64QAM`	`2/3`	

MCS	Modulation	Coding Rate	Comment
6	`64QAM`	`3/4`	
7	`64QAM`	`5/6`	
8	`256QAM`	`3/4`	
9	`256QAM`	`5/6`	
10	`BPSK`	`1/2`	Applies only for `ChannelBandwidth = 'CBW1'`

Data Types: `double`

`APEPLength` — Number of bytes in the A-MPDU pre-EOF padding
256 (default) | integer from 0 to 65,535 | vector of integers

Number of bytes in the A-MPDU pre-EOF padding, specified as an integer scalar or vector.

- For a single user, `APEPLength` is a scalar integer from 0 to 65,535.
- For multiple users, `APEPLength` is a 1-by-N_{Users} vector of integers or a scalar with values from 0 to 65,535, where $N_{\text{Users}} \leq 4$.
- `APEPLength = 0` for a null data packet (NDP).

`APEPLength` is used internally to determine the number of OFDM symbols in the data field.

Note Only aggregated data transmission is supported.

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.

Note For S1G, the first OFDM symbol within the data field always has a long guard interval, even when `GuardInterval` is set to `'Short'`.

Data Types: `char` | `string`

`GroupID` — Group identification number
1 (default) | integer from 1 to 62

Group identification number, specified as an integer scalar from 1 to 62. The group identification number is signaled during a multi-user transmission. Therefore this property applies for a long preamble (`Preamble` = `'Long'`) and when `NumUsers` is greater than 1.

Data Types: `double`

`PartialAID` — Abbreviated indication of the PSDU recipient
37 (default) | integer from 0 to 511

Abbreviated indication of the PSDU recipient, specified as an integer scalar from 0 to 511.

- For an uplink transmission, the partial identification number is the last nine bits of the basic service set identifier (BSSID) and must be an integer from 0 to 511.
- 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 and must be an integer from 0 to 63.

For more information, see IEEE Std 802.11ac-2013, Table 22-1.

Data Types: `double`

`UplinkIndication` — Enable uplink indication
`false` (default) | `true`

Enable uplink indication, specified as a logical. Set `UplinkIndication` to `true` for uplink transmission or `false` for downlink transmission. This property applies when `ChannelBandwidth` is not `'CBW1'` and `NumUsers` = 1.

Data Types: `logical`

`Color` — Access point color identifier
0 (default) | integer scalar from 0 to 7

Access point (AP) color identifier, specified as an integer from 0 to 7. An AP includes a `Color` number for the basic service set (BSS). An S1G station (STA) can use the `Color` setting to determine if the transmission is within a BSS it is associated with. An S1G STA can terminate the reception process for transmissions received from a BSS that it is not associated with. This property applies when `ChannelBandwidth` is not `'CBW1'`, `NumUsers` = 1, and `UplinkIndication` = `false`.

Data Types: `double`

`TravelingPilots` — Enable traveling pilots
`false` (default) | `true`

Enable traveling pilots, specified as a logical. Set `TravelingPilots` to `true` for nonconstant pilot locations. Traveling pilots allow a receiver to track a changing channel due to Doppler spread.

Data Types: `logical`

`ResponseIndication` — Response indication type
`'None'` (default) | `'NDP'` | `'Normal'` | `'Long'`

Response indication type, specified as `'None'`, `'NDP'`, `'Normal'`, or `'Long'`. This information is used to indicate the presence and type of frame that will be sent a short interframe space (SIFS) after the current frame transmission. The response indication field is set based on the value of `ResponseIndication` and transmitted in;

- The SIG2 field of the S1G_SHORT preamble
- The SIG-A-2 field of the S1G_LONG preamble
- The SIG field of the S1G_1M preamble

Data Types: `char` | `string`

`RecommendSmoothing` — Recommend smoothing for channel estimation
`true` (default) | `false`

Recommend smoothing for channel estimation, specified as a logical.

- If the frequency profile is nonvarying across the channel , the receiver sets this property to `true`. In this case, frequency-domain smoothing is recommended as part of channel estimation.

- If the frequency profile varies across the channel, the receiver sets this property to `false`. In this case, frequency-domain smoothing is not recommended as part of channel estimation.

Data Types: `logical`

Output Arguments

`cfgS1G` — S1G PPDU configuration
`wlanS1GConfig` object

S1G “PPDU” on page 1-313 configuration, returned as a `wlanS1GConfig` object. The properties of `cfgS1G` are described in wlanS1GConfig.

Definitions

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanDMGConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanS1GConfig.packetFormat` | `wlanVHTConfig` | `wlanWaveformGenerator`

Topics

“Packet Size and Duration Dependencies”

Introduced in R2016b

wlanS1GConfig.packetFormat

Return S1G packet format

Syntax

```
format = packetFormat(cfg)
```

Description

`format = packetFormat(cfg)` returns the S1G packet format , based on the configuration of the S1G object.

Input Arguments

`cfg` — S1G PPDU configuration
`wlanS1GConfig` object

S1G PPDU configuration, specified as a `wlanS1GConfig` object.

Output Arguments

`format` — S1G packet format
`S1G-1M` | `S1G-Short` | `S1G-Long`

S1G packet format , specified as '`S1G-1M`', '`S1G-Short`', or '`S1G-Long`'.

Examples

Create wlanS1GConfig Object and Return Packet Format

Create an S1G configuration object with default settings for a single user and change the default property settings by using dot notation. Use the `packetFormat` object function to access the S1G packet format of the object.

Create an S1G configuration object with default settings. By default, the configuration object creates properties to model the short S1G packet format.

```
cfgS1G = wlanS1GConfig;
packetFormat(cfgS1G)
```

```
ans =
'S1G-Short'
```

Modify the defaults by using the dot notation to specify a long preamble.

```
cfgS1G.Preamble = 'Long';
packetFormat(cfgS1G)
```

```
ans =
'S1G-Long'
```

See Also

`wlanS1GConfig`

Introduced in R2017b

wlanSampleRate

Return the nominal sample rate

Syntax

```
fs = wlanSampleRate(cfgFormat)
```

Description

`fs = wlanSampleRate(cfgFormat)` returns the nominal sample rate for the specified format configuration object `cfgFormat`.

Examples

Sample Rate for VHT format

Get the sample rate for a VHT format configuration in samples per second.

```
cfgVHT = wlanVHTConfig;
fs = wlanSampleRate(cfgVHT)
```

```
fs = 80000000
```

Input Arguments

`cfgFormat` — Packet format configuration
`wlanDMGConfig` object | `wlanS1GConfig` object | `wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object

Packet format configuration, specified as a `wlanDMGConfig`, `wlanS1GConfig`, `wlanVHTConfig`, `wlanHTConfig`, or `wlanNonHTConfig` object. The type of `cfgFormat` object determines the nominal sample rate. For a description of the properties and valid settings for the various packet format configuration objects, see:

- wlanDMGConfig
- wlanS1GConfig
- wlanVHTConfig
- wlanHTConfig
- wlanNonHTConfig

Output Arguments

`fs` — Sample rate

scalar

Sample rate in samples per second, returned as an scalar.

See Also

`wlanDMGConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanS1GConfig` | `wlanVHTConfig`

Introduced in R2017b

wlanScramble

Scramble and descramble binary input sequence

Syntax

```
y = wlanScramble(bits,scramInit)
```

Description

`y = wlanScramble(bits,scramInit)` scrambles or descrambles the binary input `bits` for the specified initial scramble state, using a 127-length frame-synchronous scrambler. The frame-synchronous scrambler uses the generator polynomial defined in IEEE 802.11-2012, Section 18.3.5.5 and IEEE 802.11ad-2012, Section 21.3.9. The same scrambler is used to scramble bits at the transmitter and descramble bits at the receiver.

Examples

Scramble and Descramble bits

Create the scrambler initialization and the input sequence of random bits.

```
scramInit = 93;
bits = randi([0,1],1000,1);
```

Scramble and descramble the bits by using the scrambler initialization.

```
scrambledData = wlanScramble(bits,scramInit);
descrambledData = wlanScramble(scrambledData,scramInit);
```

Verify that the descrambled data matches the original data.

```
isequal(bits,descrambledData)
```

```
ans = logical
   1
```

Input Arguments

`bits` — Input sequence

column vector | matrix

Input sequence to be scrambled, specified as a binary column vector or matrix.

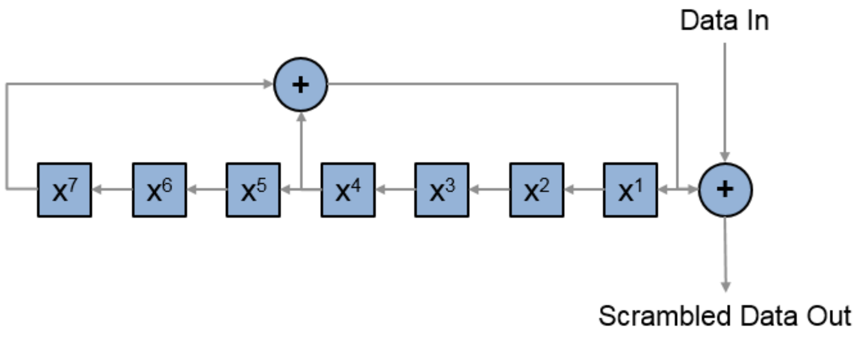
Data Types: `double` | `int8`

`scramInit` — Initial state of scrambler

integer from 1 to 127 | 7-by-1 binary column vector

Initial state of the scrambler, specified as an integer from 1 to 127, or a corresponding 7-by-1 column vector of binary bits.

The scrambler initialization used on the transmission data follows the process described in IEEE Std 802.11-2012, Section 18.3.5.5 and IEEE Std 802.11ad-2012, Section 21.3.9. The header and data fields that follow the scrambler initialization field (including data padding bits) are scrambled by XORing each bit with a length-127 periodic sequence generated by the polynomial $S(x) = x^7+x^4+1$. The octets of the PSDU (Physical Layer Service Data Unit) are placed into a bit stream, and within each octet, bit 0 (LSB) is first and bit 7 (MSB) is last. The generation of the sequence and the XOR operation are shown in this figure:



Conversion from integer to bits uses left-MSB orientation. For the initialization of the scrambler with decimal `1`, the bits are mapped to the elements shown.

Element	X^7	X^6	X^5	X^4	X^3	X^2	X^1
Bit Value	0	0	0	0	0	0	1

To generate the bit stream equivalent to a decimal, use `de2bi`. For example, for decimal `1`:

```
de2bi(1,7,'left-msb')
ans =

     0     0     0     0     0     0     1
```

Same `scramInit` is applied across all the columns of `bits` when the input is a matrix.

Example: `[0 0 0 0 0 0 1]'`

Data Types: `double`

Output Arguments

`y` — Scrambled or descrambled output
column vector | matrix

Scrambled or descrambled output, returned as a binary column vector or matrix with the same size and type as `bits`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`comm.Descrambler` | `comm.Scrambler` | `wlanWaveformGenerator`

Introduced in R2017b

wlanSegmentDeparseBits

Segment-deparse data bits

Syntax

```
y = wlanSegmentDeparseBits(bits,cbw,numES,numCBPS,numBPSCS)
```

Description

`y = wlanSegmentDeparseBits(bits,cbw,numES,numCBPS,numBPSCS)` performs the inverse operation of the segment parsing defined in IEEE 802.11ac-2013 Section 22.3.10.7 when `cbw` is `'CBW16'` or `'CBW160'`.

Note Segment deparsing of the bits applies only when the channel bandwidth is either 16 MHz or 160 MHz, and is bypassed for the remaining channel bandwidths (as stated in the aforementioned section of IEEE802.11ac-2013). Therefore, when `cbw` is any accepted value other than `'CBW16'` or `'CBW160'`, `wlanSegmentParseBits` returns the input unchanged.

Examples

Segment-Deparse Coded Bits in Two OFDM Symbols

Segment-deparse the coded bits for a VHT configuration (with a channel bandwidth of 160 MHz and three spatial streams) into two OFDM symbols.

Define the input parameters. Set the channel bandwidth to 160 MHz, the number of coded bits per OFDM symbol to 2808, the number of spatial streams to 3, the number of encoded streams to 1, the number of coded bits per subcarrier per spatial stream to 2, and the number of OFDM symbols to 2. Calculate the number of coded bits per OFDM symbol per spatial stream by dividing the number of coded bits per OFDM symbol by the number of spatial streams.

```
chanBW = 'CBW160';
numCBPS = 2808;
numSS = 3;
numES = 1;
numBPSCS = 2;
numSym = 2;
numCBPSS = numCBPS/numSS;
```

Create the input sequence of bits.

```
bits = randi([0 1],numCBPSS*numSym,numSS);
```

Perform segment parsing on the bits.

```
parsedBits = wlanSegmentParseBits(bits,chanBW,numES,numCBPS,numBPSCS);
size(parsedBits)
```

```
ans =

   936     3     2
```

Perform segment deparsing on the parsed bits.

```
 deparsedBits = wlanSegmentDeparseBits(parsedBits,chanBW,numES,numCBPS,numBPSCS);
 size(deparsedBits)
```

```
ans =

        1872           3
```

Verify that the deparsed data matches the original data.

```
isequal(bits,deparsedBits)
```

```
ans = logical
   1
```

Input Arguments

`bits` — Input sequence

matrix | 3-D array

Input sequence of deinterleaved bits, specified as an (N_{CBPSSI}×N_{SYM})-by-N_{SS}-by-N_{SEG} array, where:

- N_{CBPSSI} is the number of coded bits per OFDM symbol per spatial stream per interleaver block.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.
- N_{SEG} is the number of segments. When `cbw` is `'CBW16'` or `'CBW160'`, N_{SEG} must be 2. Otherwise it must be 1.

Data Types: `double` | `int8`

`cbw` — Channel bandwidth

`'CBW1'` | `'CBW2'` | `'CBW4'` | `'CBW8'` | `'CBW16` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW1'`,`'CBW2'`, `'CBW4'`,`'CBW8'`, `'CBW16'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Example: `'CBW160'`

Data Types: `char` | `string`

`numES` — Number of encoded streams

1 to 9 | 12

Number of encoded streams, specified as an integer from 1 to 9, or 12.

Data Types: `double`

`numCBPS` — Number of coded bits per OFDM symbol

positive integer

Number of coded bits per OFDM symbol, specified as a positive integer. When `cbw` is `'CBW16'` or `'CBW160'`, `numCBPS` must be an integer equal to 468×N_{BPSCS}×N_{SS}, where:

- N_{BPSCS} is the number of coded bits per subcarrier per spatial stream.
- N_{SS} is the number of spatial streams. It accounts for the number of columns (second dimension) of the input `bits`.

Data Types: `double`

`numBPSCS` — Number of coded bits per subcarrier per spatial stream
1 | 2 | 4 | 6 | 8

Number of coded bits per subcarrier per spatial stream, specified as log2(M), where M is the modulation order. Therefore, `numBPSCS` must equal:

- 1 for a BPSK modulation
- 2 for a QPSK modulation
- 4 for a 16QAM modulation
- 6 for a 64QAM modulation
- 8 for a 256QAM modulation

Data Types: `double`

Output Arguments

`y` — Merged segments of data
matrix

Merged segments of data, specified as an (N_{CBPSS}×N_{SYM})-by-N_{SS} matrix, where:

- N_{CBPSS} is the number of coded bits per OFDM symbol per spatial stream.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanSegmentParseBits`

Introduced in R2017b

wlanSegmentDeparseSymbols

Segment-deparse data subcarriers

Syntax

```
y = wlanSegmentDeparseSymbols(sym,cbw)
```

Description

`y = wlanSegmentDeparseSymbols(sym,cbw)` performs segment deparsing on the input `sym` as per IEEE 802.11ac-2013, Section 22.3.10.9.3, when `cbw` is `'CBW16'` or `'CBW160'`.

Note Segment deparsing of the data subcarriers applies only when the channel bandwidth is either 16 MHz or 160 MHz, and is bypassed for the remaining channel bandwidths (as stated in the aforementioned section of IEEE802.11ac-2013). Therefore, when `cbw` is any accepted value other than `'CBW16'` or `'CBW160'`, `wlanSegmentDeparseSymbols` returns the input unchanged.

Examples

Segment-Deparse Symbols

Segment-deparse the symbols in four OFDM symbols for a VHT configuration with a channel bandwidth of 16 MHz and 3 spatial streams.

Define the input parameters. Since the channel bandwidth is 16 MHz, set the number of data subcarriers to 468 and the number of frequency segments to two.

```
chanBW = 'CBW16';
numSD = 468;
numSym = 4;
```

```
numSS = 3;
numSeg = 2;
```

Create the input sequence of symbols.

```
data = randi([0 1],numSD/numSeg,numSym,numSS,numSeg);
```

Segment-deparse the symbols into data subcarriers. The first dimension of the parsed output accounts for the total number of data subcarriers.

```
deparsedData = wlanSegmentDeparseSymbols(data,chanBW);
size(deparsedData)
```

```
ans =

   468     4     3
```

Get Symbol Order for a VHT Configuration

Get the symbol order after stream deparsing a sequence for a VHT configuration with a channel bandwidth of 160 MHz and one spatial stream.

Define the input parameters. Since the channel bandwidth is 160 MHz, set the number of data subcarriers to 468 and the number of frequency segments to two.

```
chanBW = 'CBW160';
numSD = 468;
numSym = 1;
numSS = 1;
numSeg = 2;
```

Create the input sequence of symbols.

```
sequence = (1:numSD*numSym*numSS).';
inp = reshape(sequence, numSD/numSeg, numSym, numSS, numSeg);
```

Segment-deparse the symbols. The output is a column vector with the sequence order of the symbols.

```
deparsedData = wlanSegmentDeparseSymbols(inp, chanBW);
deparsedData(1:10)
```

```
ans =

     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
```

Input Arguments

`sym` — Input sequence

4-D array

Input sequence of frequency segments to deparse, specified as an (N_{SD}/N_{SEG})-by-$N_{SYM}$by-N_{SS}-by-N_{SEG} array, where:

- N_{SD} is the number of data subcarriers.
- N_{SEG} is the number of segments. When `cbw` is `'CBW16'` or `'CBW160'`, N_{SEG} is 2. Otherwise it is 1.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.

Data Types: `double`
Complex Number Support: Yes

`cbw` — Channel bandwidth

`'CBW1'` | `'CBW2'` | `'CBW4'` | `'CBW8'` | `'CBW16` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW1'`,`'CBW2'`, `'CBW4'`,`'CBW8'`, `'CBW16'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Example: `'CBW160'`

Data Types: `char` | `string`

Output Arguments

`y` — Deparsed frequency segments
3-D array

Deparsed frequency segments, specified as an N_{SD}-by-N_{SYM}-by-N_{SS} array, where:

- N_{SD} is the number of data subcarriers.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanSegmentParseSymbols`

Introduced in R2017b

wlanSegmentParseBits

Segment-parse data bits

Syntax

```
y = wlanSegmentParseBits(bits,cbw,numES,numCBPS,numBPSCS)
```

Description

`y = wlanSegmentParseBits(bits,cbw,numES,numCBPS,numBPSCS)` performs segment parsing on the input `bits` as per IEEE 802.11ac-2013, Section 22.3.10.7, when `cbw` is `'CBW16'` or `'CBW160'`.

Note Segment parsing of the bits applies only when the channel bandwidth is either 16 MHz or 160 MHz, and is bypassed for the remaining channel bandwidths (as stated in the aforementioned section of IEEE802.11ac-2013). Therefore, when `cbw` is any accepted value other than `'CBW16'` or `'CBW160'`, `wlanSegmentParseBits` returns the input unchanged.

Examples

Segment-Parse Bits in Two OFDM Symbols

Segment-parse coded bits for a VHT configuration (with a channel bandwidth of 160 MHz and three spatial streams) into two OFDM symbols.

Define the input parameters. Set the channel bandwidth to 160 MHz, the number of coded bits per OFDM symbol to 2808, the number of spatial streams to 3, the number of encoded streams to 1, the number of coded bits per subcarrier per spatial stream to 2, and the number of OFDM symbols to 2. Calculate the number of coded bits per OFDM symbol per spatial stream by dividing the number of coded bits per OFDM symbol by the number of spatial streams.

```
chanBW = 'CBW160';
numCBPS = 2808;
numSS = 3;
numES = 1;
numBPSCS = 2;
numSym = 2;
numCBPSS = numCBPS/numSS;
```

Create the input sequence of bits.

```
bits = randi([0 1],numCBPSS*numSym,numSS,'int8');
```

Perform segment parsing on the bits.

```
parsedBits = wlanSegmentParseBits(bits,chanBW,numES,numCBPS,numBPSCS);
```

The parsed sequence is a three-dimensional array of bits.

```
size(parsedBits)
```

```
ans =

   936     3     2

```

```
parsedBits(1:5,:,:)
```

```
ans = 5x3x2 int8 array
ans(:,:,1) =

   1   0   1
   0   1   1
   1   0   1
   0   0   0
   1   0   1


ans(:,:,2) =

   1   1   1
   1   1   1
   0   0   1
   1   1   0
   1   0   0
```

Get Bit Order of OFDM Symbol

Get the bit order after the segment parsing of an OFDM symbol of an S1G configuration with a channel bandwidth of 16 MHz, and two spatial streams.

Define the input parameters. Set the channel bandwidth to 16 MHz, the number of coded bits per OFDM symbol to 1872, the number of spatial streams to 2, the number of encoded streams to 1, the number of coded bits per subcarrier per spatial stream to 2 and the number of OFDM symbols to 2. Calculate the number of coded bits per OFDM symbol per spatial stream by dividing the number of coded bits per OFDM symbol by the number of spatial streams.

```
chanBW = 'CBW16';
numCBPS = 1872;
numSS = 2;
numES = 1;
numBPSCS = 2;
numSym = 1;
numCBPSS = numCBPS/numSS;
```

Create the input sequence.

```
sequence = (1:numCBPS*numSym).';
inp = reshape(sequence,numCBPSS*numSym,numSS);
```

Perform segment parsing on the sequence.

```
parsedSequence = wlanSegmentParseBits(inp,chanBW,numES,numCBPS,numBPSCS);
```

The parsed sequence is a three-dimensional array containing the corresponding bit order.

```
size(parsedSequence)
```

```
ans =

   468     2     2
```

Input Arguments

`bits` — Input sequence

matrix

Input sequence of stream-parsed bits, specified as an (N_{CBPSS}×N_{SYM})-by-N_{SS} matrix, where:

- N_{CBPSS} is the number of coded bits per OFDM symbol per spatial stream.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.

Data Types: `double` | `int8`

`cbw` — Channel bandwidth

`'CBW1'` | `'CBW2'` | `'CBW4'` | `'CBW8'` | `'CBW16` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW1'`,`'CBW2'`, `'CBW4'`,`'CBW8'`, `'CBW16'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Example: `'CBW160'`

Data Types: `char` | `string`

`numES` — Number of encoded streams

1 to 9 | 12

Number of encoded streams, specified as an integer from 1 to 9, or 12.

Data Types: `double`

`numCBPS` — Number of coded bits per OFDM symbol

positive integer

Number of coded bits per OFDM symbol, specified as a positive integer. When `cbw` is `'CBW16'` or `'CBW160'`, `numCBPS` must be an integer equal to 468×N_{BPSCS}×N_{SS}, where:

- N_{BPSCS} is the number of coded bits per subcarrier per spatial stream.
- N_{SS} is the number of spatial streams. It accounts for the number of columns (second dimension) of the input `bits`.

Data Types: `double`

`numBPSCS` — Number of coded bits per subcarrier per spatial stream
1 | 2 | 4 | 6 | 8

Number of coded bits per subcarrier per spatial stream, specified as log2(M), where M is the modulation order. Therefore, `numBPSCS` must equal:

- 1 for a BPSK modulation
- 2 for a QPSK modulation
- 4 for a 16QAM modulation
- 6 for a 64QAM modulation
- 8 for a 256QAM modulation

Data Types: `double`

Output Arguments

`y` — Segment-parsed bits
matrix | 3-D array

Segment-parsed bits, specified as an (N_{CBPSSI}×N_{SYM})-by-N_{SS}-by-N_{SEG} array, where:

- N_{CBPSSI} is the number of coded bits per OFDM symbol per spatial stream per interleaver block.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.
- N_{SEG} is the number of segments. When `cbw` is `'CBW16'` or `'CBW160'`, N_{SEG} is 2. Otherwise it is 1.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanSegmentDeparseBits`

Introduced in R2017b

wlanSegmentParseSymbols

Segment-parse data subcarriers

Syntax

```
y = wlanSegmentParseSymbols(sym,cbw)
```

Description

`y = wlanSegmentParseSymbols(sym,cbw)` performs the inverse operation of the segment deparsing on the input `sym` defined in IEEE 802.11ac-2013, Section 22.3.10.9.3, when `cbw` is `'CBW16'` or `'CBW160'`.

Note Segment parsing of the data subcarriers applies only when the channel bandwidth is either 16 MHz or 160 MHz, and is bypassed for the remaining channel bandwidths (as stated in the aforementioned section of IEEE802.11ac-2013). Therefore, when `cbw` is any accepted value other than `'CBW16'` or `'CBW160'`, `wlanSegmentParseSymbols` returns the input unchanged.

Examples

Segment-Parse Symbols

Segment-deparse and segment-parse the symbols in four OFDM symbols for a VHT configuration with a channel bandwidth of 160 MHz and two spatial streams.

Define the input parameters. Since the channel bandwidth is 160 MHz, set the number of data subcarriers to 468 and the number of frequency segments to two.

```
chanBW = 'CBW160';
numSD = 468;
numSym = 4;
```

```
numSS = 2;
numSeg = 2;
```

Create the input sequence of symbols.

```
data = randi([0 1],numSD/numSeg,numSym,numSS,numSeg);
```

Segment-deparse the symbols into data subcarriers. The first dimension of the parsed output accounts for the total number of data subcarriers.

```
deparsedData = wlanSegmentDeparseSymbols(data,chanBW);
size(deparsedData)
```

```
ans =

   468     4     2
```

Segment-parse the symbols into data subcarriers. The size of the output is equal to the size of the original sequence.

```
segments = wlanSegmentParseSymbols(deparsedData,chanBW);
size(segments)
```

```
ans =

   234     4     2     2
```

Input Arguments

`sym` — Input sequence

3-D array

Input sequence of equalized data to be segmented, specified as an N_{SD}-by-N_{SYM}-by-N_{SS} array, where:

- N_{SD} is the number of data subcarriers.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.

Data Types: `double`

Complex Number Support: Yes

`cbw` — Channel bandwidth
`'CBW1'` | `'CBW2'` | `'CBW4'` | `'CBW8'` | `'CBW16` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW1'`,`'CBW2'`, `'CBW4'`,`'CBW8'`, `'CBW16'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Example: `'CBW160'`

Data Types: `char` | `string`

Output Arguments

`y` — Frequency segments
4-D array

Frequency segments, specified as an (N_{SD}/N_{SEG})-by-$N_{SYM}$by-N_{SS}-by-N_{SEG} array, where:

- N_{SD} is the number of data subcarriers.
- N_{SEG} is the number of segments. When `cbw` is `'CBW16'` or `'CBW160'`, N_{SEG} is 2. Otherwise it is 1.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanSegmentDeparseSymbols`

Introduced in R2017b

wlanStreamDeparse

Stream-deparse binary input

Syntax

```
y = wlanStreamDeparse(bits,numES,numCBPS,numBPSCS)
```

Description

`y = wlanStreamDeparse(bits,numES,numCBPS,numBPSCS)` deparses the spatial streams specified in `bits` to form encoded streams. This operation is the inverse of the one defined in IEEE 802.11-2012 Section 20.3.11.8.2 and IEEE 802.11ac-2013 Section 22.3.10.6.

Examples

Stream-Deparse Input Bits

Stream-deparse five OFDM symbols with two spatial streams into one encoded stream.

Define the input parameters. Set the number of coded bits per OFDM symbol to 432, the number of coded bits per subcarrier per spatial stream to 2, the number of encoded streams to 1, the number of spatial streams to 2 and the number of OFDM symbols to 5.

```
numCBPS = 432;
numBPSCS = 2;
numES = 1;
numSS = 2;
numSym = 5;
```

Create a parsed input of hard bits.

```
parsed = randi([0 1],numCBPS/numSS*numSym,numSS)
parsed =

     1     0
     1     1
     0     1
     1     1
     1     1
     0     1
     0     1
     1     0
     1     1
     1     1
```

Stream-deparse the bits.

```
deparsed = wlanStreamDeparse(parsed,numES,numCBPS,numBPSCS)
```

```
deparsed =

     1
     0
     1
     1
     0
     1
     1
     1
     1
     1
```

Input Arguments

`bits` **— Input sequence**
matrix

Input sequence of stream-parsed data, specified as a (N_{CBPSS}×N_{SYM})-by-N_{SS} matrix, where:

- N_{CBPSS} is the number of coded bits per OFDM symbol per spatial stream.

- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.

Data Types: `double` | `int8`

`numES` — Number of encoded streams

integer from 1 to 9, 12

Number of encoded streams, specified as a integer from 1 to 9, or 12.

Data Types: `double`

`numCBPS` — Number of coded bits per OFDM symbol

positive integer

Number of coded bits per OFDM symbol, specified as an integer equal to ($N_{\mathrm{BPSCS}} \times N_{\mathrm{SS}} \times N_{\mathrm{SD}}$), where:

- N_{BPSCS} is the number of coded bits per subcarrier per spatial stream. See `numBPSCS`.
- N_{SS} is the number of spatial streams.
- N_{SD} is the number of complex data numbers per frequency segment, specified as 24, 52, 108, 234, or 468.

Data Types: `double`

`numBPSCS` — Number of coded bits per subcarrier per spatial stream

1 | 2 | 4 | 6 | 8

Number of coded bits per subcarrier per spatial stream, specified as 1, 2, 4, 6, or 8.

Data Types: `double`

Output Arguments

`y` — Stream-deparsed output

matrix

Stream-deparsed output data, returned as an ($N_{CBPS} \times N_{SYM}$)-by-N_{ES} matrix, where:

- N_{CBPS} is the number of coded bits per OFDM symbol.

- N_{SYM} is the number of OFDM symbols.
- N_{ES} is the number of encoded streams.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanStreamParse`

Introduced in R2017b

wlanStreamParse

Stream-parse binary input

Syntax

```
y = wlanStreamParse(bits,numSS,numCBPS,numBPSCS)
```

Description

`y = wlanStreamParse(bits,numSS,numCBPS,numBPSCS)` parses the encoded `bits` into spatial streams, as defined in IEEE 802.11-2012 Section 20.3.11.8.2 and IEEE 802.11ac-2013 Section 22.3.10.6.

Examples

Stream-Parse Input Bits

Stream-parse three OFDM symbols with two encoded streams into five spatial streams.

Define the input parameters. Set the number of coded bits per OFDM symbol to 3240, the number of coded bits per subcarrier per spatial stream to 6, the number of encoded streams to 2, the number of spatial streams to 5 and the number of OFDM symbols to 3.

```
numCBPS = 3240;
numBPSCS = 6;
numES = 2;
numSS = 5;
numSym = 3;
```

Create a random sequence of bits.

```
bits = randi([0 1],numCBPS*numSym/numES,numES,'int8');
```

Stream-parse the random bits.

```
parsedData = wlanStreamParse(bits,numSS,numCBPS,numBPSCS);
```

Verify the size of the parsed bits.

```
size(parsedData)
```

```
ans =

        1944           5
```

Get Bit Order After Stream Parsing

Get the bit order of an OFDM symbol after stream-parsing it from one encoded stream into three spatial streams.

Define the input parameters. Set the number of coded bits per OFDM symbol to 156, the number of coded bits per subcarrier per spatial stream to 1, the number of encoded streams to 1, the number of spatial streams to 3 and the number of OFDM symbols to 1.

```
numCBPS = 156;
numBPSCS = 1;
numES = 1;
numSS = 3;
numSym = 1;
```

Create an input sequence of ordered symbols with the proper dimensions.

```
sequence = (1:numCBPS*numSym).';
inp = reshape(sequence,numCBPS*numSym/numES,numES)
```

```
inp =

     1
     2
     3
     4
     5
     6
     7
     8
     9
```

```
    10
```

Stream-parse the symbols.

```
parsedData = wlanStreamParse(inp,numSS,numCBPS,numBPSCS)
```

```
parsedData =

     1     2     3
     4     5     6
     7     8     9
    10    11    12
    13    14    15
    16    17    18
    19    20    21
    22    23    24
    25    26    27
    28    29    30
```

Input Arguments

`bits` — Input sequence

matrix

Input sequence of encoded bits, specified as a (N_{CBPS}×N_{SYM}/N_{ES})-by-N_{ES} matrix, where:

- N_{CBPS} is the number of coded bits per OFDM symbol.
- N_{SYM} is the number of OFDM symbols.
- N_{ES} is the number of encoded streams.

Data Types: `double` | `int8`

`numSS` — Number of spatial streams

integer from 1 to 8

Number of spatial streams (N_{SS}), specified as an integer from 1 to 8.

Data Types: `double`

`numCBPS` — **Number of coded bits per OFDM symbol**
positive integer

Number of coded bits per OFDM symbol, specified as an integer equal to (N_{BPSCS}×N_{SS}×N_{SD}), where:

- N_{BPSCS} is the number of coded bits per subcarrier per spatial stream. See `numBPSCS`.
- N_{SS} is the number of spatial streams.
- N_{SD} is the number of complex data numbers per frequency segment, specified as 24, 52, 108, 234, or 468.

Data Types: `double`

`numBPSCS` — **Number of coded bits per subcarrier per spatial stream**
1 | 2 | 4 | 6 | 8

Number of coded bits per subcarrier per spatial stream, specified as 1, 2, 4, 6, or 8.

Data Types: `double`

Output Arguments

`y` — **Stream-parsed output**
matrix

Stream-parsed output data, returned as an (N_{CBPSS}×N_{SYM})-by-N_{SS} matrix, where:

- N_{CBPSS} is the number of coded bits per OFDM symbol per spatial stream.
- N_{SYM} is the number of OFDM symbols.
- N_{SS} is the number of spatial streams.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanStreamDeparse`

Introduced in R2017b

wlanSymbolTimingEstimate

Fine symbol timing estimate using L-LTF

Syntax

```
startOffset = wlanSymbolTimingEstimate(rxSig,cbw)
startOffset = wlanSymbolTimingEstimate(rxSig,cbw,threshold)
[startOffset,M] = wlanSymbolTimingEstimate( ___ )
```

Description

`startOffset = wlanSymbolTimingEstimate(rxSig,cbw)` returns the offset from the start of the input waveform to the estimated start of the “L-STF” on page 1-359 [21].

`startOffset = wlanSymbolTimingEstimate(rxSig,cbw,threshold)` specifies the threshold that the decision metric must meet or exceed to obtain a symbol timing estimate.

`[startOffset,M] = wlanSymbolTimingEstimate( ___ )` also returns the decision metric of the symbol timing algorithm for the received time-domain waveform, using any of the input arguments in the previous syntaxes.

Examples

Detect HT Packet and Estimate Symbol Timing

Detect a received 802.11n™ packet and estimate its symbol timing at 20 dB SNR.

Create an HT format configuration object and TGn channel configuration object.

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

```
cfgHT = wlanHTConfig;
tgn = wlanTGnChannel;
```

Generate a transmit waveform and add a delay at the start of the waveform.

```
txWaveform = wlanWaveformGenerator([1;0;0;1],cfgHT);
txWaveform = [zeros(100,1);txWaveform];
```

Pass the waveform through the TGn channel model and add noise.

```
SNR = 20; % In decibels
fadedSig = tgn(txWaveform);
rxWaveform = awgn(fadedSig,SNR,0);
```

Detect the packet. Extract the non-HT fields. Estimate the fine packet offset using the coarse detection for the first symbol of the waveform and the non-HT preamble field indices.

```
startOffset = wlanPacketDetect(rxWaveform,cfgHT.ChannelBandwidth);
ind = wlanFieldIndices(cfgHT);
nonHTFields = rxWaveform(startOffset+(ind.LSTF(1):ind.LSIG(2)),:);

startOffset = wlanSymbolTimingEstimate(nonHTFields, ...
    cfgHT.ChannelBandwidth)
```

```
startOffset =

     6
```

Detect HT Packet and Set Threshold When Estimating Symbol Timing

Impair an HT waveform by passing it through a TGn channel configured to model a large delay spread. Detect the waveform and estimate the symbol timing. Adjust the decision metric threshold and estimate the symbol timing again.

Create an HT format configuration object and TGn channel configuration object. Specify the Model-E delay profile, which introduces a large delay spread.

```
cfgHT = wlanHTConfig;
```

```
tgn = wlanTGnChannel;
tgn.DelayProfile = 'Model-E';
```

Generate a transmit waveform and add a delay at the start of the waveform.

```
txWaveform = wlanWaveformGenerator([1;0;0;1],cfgHT);
txWaveform = [zeros(100,1);txWaveform];
```

Pass the waveform through the TGn channel model and add noise.

```
SNR = 50; % In decibels
fadedSig = tgn(txWaveform);
rxWaveform = awgn(fadedSig,SNR,0);
```

Detect the packet. Extract the non-HT fields. Estimate the fine packet offset using the coarse detection for the first symbol of the waveform and the non-HT preamble field indices. Adjust the decision metric threshold and estimate the fine packet offset again.

```
startOffset = wlanPacketDetect(rxWaveform,cfgHT.ChannelBandwidth);
ind = wlanFieldIndices(cfgHT);
nonHTFields = rxWaveform(startOffset+(ind.LSTF(1):ind.LSIG(2)),:);

startOffset = wlanSymbolTimingEstimate(nonHTFields, ...
    cfgHT.ChannelBandwidth)
threshold = 0.1
startOffset = wlanSymbolTimingEstimate(nonHTFields, ...
    cfgHT.ChannelBandwidth,threshold)
```

```
startOffset =

     5


threshold =

    0.1000


startOffset =

     9
```

Detecting the correct timing offset is more challenging for a channel model with large delay spread. For large delay spread channels, you can try lowering the threshold setting to see if performance improves in an end-to-end simulation.

Estimate Symbol Timing of TGn-Impaired HT Waveform

Detect a received 802.11n™ packet and estimate its symbol timing at 15 dB SNR.

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

```
cfgHT = wlanHTConfig;
nAnt = 2;
cfgHT.NumTransmitAntennas = nAnt;
cfgHT.NumSpaceTimeStreams = nAnt;
```

Show the logic behind the MCS selection for BPSK modulation.

```
if cfgHT.NumSpaceTimeStreams == 1
    cfgHT.MCS = 0;
elseif cfgHT.NumSpaceTimeStreams == 2
    cfgHT.MCS = 8;
elseif cfgHT.NumSpaceTimeStreams == 3
    cfgHT.MCS = 16;
elseif cfgHT.NumSpaceTimeStreams == 4
    cfgHT.MCS = 24;
end
```

Generate a transmit waveform and add a delay at the start of the waveform.

```
txWaveform = wlanWaveformGenerator([1;0;0;1],cfgHT);
txWaveform = [zeros(100,cfgHT.NumTransmitAntennas);txWaveform];
```

Create a TGn channel configuration object for two transmit antennas and two receive antennas. Specify the Model-B delay profile. Pass the waveform through the TGn channel model and add noise.

```
tgn = wlanTGnChannel;
tgn.NumTransmitAntennas = nAnt;
tgn.NumReceiveAntennas = nAnt;
tgn.DelayProfile = 'Model-B';

SNR = 15; % In decibels
```

```
fadedSig = tgn(txWaveform);
rxWaveform = awgn(fadedSig,SNR,0);
```

Detect the packet. Extract the non-HT fields. Estimate the fine packet offset using the coarse detection for the first symbol of the waveform and the non-HT preamble field indices.

```
startOffset = wlanPacketDetect(rxWaveform,cfgHT.ChannelBandwidth);
ind = wlanFieldIndices(cfgHT);
nonHTFields = rxWaveform(startOffset+(ind.LSTF(1):ind.LSIG(2)),:);

startOffset = wlanSymbolTimingEstimate(nonHTFields, ...
    cfgHT.ChannelBandwidth)
```

```
startOffset =

     8
```

Estimate VHT Packet Symbol Timing

Return the symbol timing and decision metric of an 802.11ac™ packet without channel impairments.

Create a VHT format configuration object. Specify two transmit antennas and two space-time streams.

```
cfgVHT = wlanVHTConfig;
cfgVHT.NumTransmitAntennas = 2;
cfgVHT.NumSpaceTimeStreams = 2;
```

Generate a VHT format transmit waveform. Add a 50-sample delay at the start of the waveform.

```
txWaveform = wlanWaveformGenerator([1;0;0;1],cfgVHT);
txWaveform = [zeros(50,cfgVHT.NumTransmitAntennas); txWaveform];
```

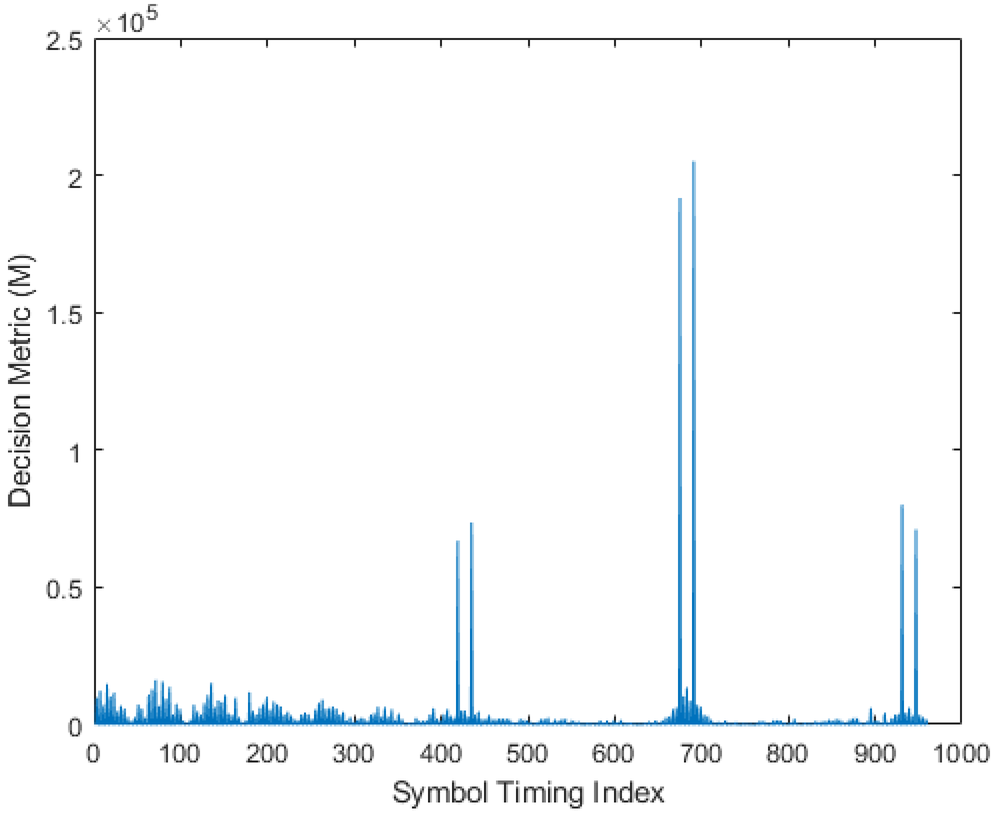
Extract the non-HT preamble fields. Obtain the timing offset estimate and decision metric.

```
ind = wlanFieldIndices(cfgVHT);
nonhtfields = txWaveform(ind.LSTF(1):ind.LSIG(2),:);
```

```
[startOffset,M] = wlanSymbolTimingEstimate(nonhtfields, ...
    cfgVHT.ChannelBandwidth);
```

Plot the returned decision metric for the non-HT preamble of the VHT format transmission waveform.

```
figure
plot(M)
xlabel('Symbol Timing Index')
ylabel('Decision Metric (M)')
```



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 time-domain samples in the L-LTF and N_R is the number of receive antennas.

Data Types: `double`

`cbw` — Channel bandwidth
`'CBW5'` | `'CBW10'` | `'CBW20'` | `'CBW40'` | `'CBW80'` | `'CBW160'`

Channel bandwidth in MHz, specified as `'CBW5'`, `'CBW10'`, `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Data Types: `char` | `string`

`threshold` — Decision threshold

1 (default) | real scalar from 0 to 1

Decision threshold, specified as a real scalar from 0 to 1.

You can try out different threshold to maximize the packet reception performance. For channels with small delay spread with respect to the cyclic prefix length, the default value is recommended. For a wireless channel with large delay spread with respect to the cyclic prefix length, such as TGn channel with `'Model E'` delay profile, a value of 0.5 is suggested.

By lowering the threshold setting, you add a non-negative corrector to the symbol timing estimate as compared to the estimate using the default threshold setting. The range of the timing corrector is [0, CSD ns/sampling duration]. For more information, see “Cyclic Shift Delay (CSD)” on page 1-361.

Data Types: `double`

Output Arguments

`startOffset` — Offset of L-STF start

integer

Offset of L-STF start, returned as an integer within the range [$–L$, $N_S–2L$], where L is the length of the L-LTF and N_S is the number of samples. Using the input channel bandwidth (`cbw`) to determine the range of symbol timing, `wlanSymbolTimingEstimate` estimates the offset to the start of L-STF by cross-correlating the received signal with a locally generated “L-LTF” on page 1-360 of the first antenna.

- `startOffset` is empty when $N_S < L$.
- `startOffset` is negative when the input waveform does not contain a complete “L-STF” on page 1-359.

`M` — Cross-correlation

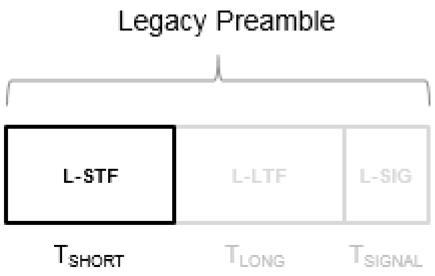
vector

Cross-correlation, returned as an (N_S–L+1)-by-1 vector. `M` is the cross-correlation between the received signal and the locally generated “L-LTF” on page 1-360 of the first transmit antenna.

Definitions

L-STF

The legacy short training field (L-STF) is the first field of the 802.11 OFDM PLCP legacy preamble. The L-STF is a component of VHT, HT, and non-HT PPDUs.



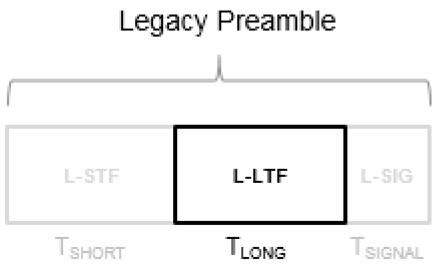
The L-STF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	L-STF Duration ($T_{SHORT} = 10 \times T_{FFT} / 4$)
20, 40, 80, and 160	312.5	3.2 μs	8 μs
10	156.25	6.4 μs	16 μs
5	78.125	12.8 μs	32 μs

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 per 20 MHz channel bandwidth segment. For 5 MHz, 10 MHz, and 20 MHz bandwidths, the number of channel bandwidths segments is 1.

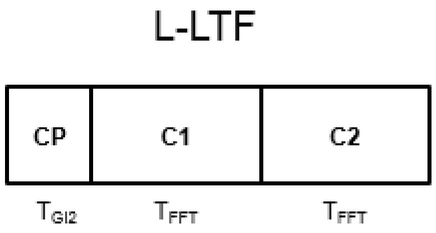
L-LTF

The legacy long training field (L-LTF) is the second field in the 802.11 OFDM PLCP legacy preamble. The L-LTF is a component of VHT, HT, and non-HT PPDUs.



Channel estimation, fine frequency offset estimation, and fine symbol timing offset estimation rely on the L-LTF.

The L-LTF is composed of a cyclic prefix (CP) followed by two identical long training symbols (C1 and C2). The CP consists of the second half of the long training symbol.



The L-LTF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration ($T_{GI2} = T_{FFT} / 2$)	L-LTF Duration ($T_{LONG} = T_{GI2} + 2 \times T_{FFT}$)
20, 40, 80, and 160	312.5	3.2 μs	1.6 μs	8 μs
10	156.25	6.4 μs	3.2 μs	16 μs
5	78.125	12.8 μs	6.4 μs	32 μs

Cyclic Shift Delay (CSD)

A CSD is added to the L-LTF for each transmit antenna, which causes multiple strong peaks in the correlation function *M*. The multiple peaks affect the accuracy of fine symbol timing estimation. For more information, see IEEE 802.11ac, Section 22.3.8.2.1 and Table 22-10.

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

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

Introduced in R2017a

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 1-371.

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

   Read-only properties:
              PSDULength: 1035
```

Update the channel bandwidth.

```
cfgVHT.ChannelBandwidth = 'CBW40'
```

```
cfgVHT = 

  wlanVHTConfig with properties:

        ChannelBandwidth: 'CBW40'
                NumUsers: 1
     NumTransmitAntennas: 1
     NumSpaceTimeStreams: 1
          SpatialMapping: 'Direct'
                    STBC: 0
                     MCS: 0
           ChannelCoding: 'BCC'
              APEPLength: 1024
           GuardInterval: 'Long'
                 GroupID: 63
              PartialAID: 275

   Read-only properties:
              PSDULength: 1030
```

Create wlanVHTConfig Object for Two Users

Create a VHT configuration object for a 20MHz two-user configuration and one antenna per user.

Create a `wlanVHTConfig` object using a combination of `Name,Value` pairs and in-line initialization to change default settings. Vector-valued properties apply user-specific settings.

```
cfgMU = wlanVHTConfig('ChannelBandwidth','CBW20','NumUsers',2, ...
    'GroupID',2,'NumTransmitAntennas',2);

cfgMU.NumSpaceTimeStreams = [1 1];
cfgMU.MCS = [4 8];
cfgMU.APEPLength = [1024 2048];
cfgMU.ChannelCoding = {'BCC' 'LDPC'}
```

```
cfgMU = 

  wlanVHTConfig with properties:

       ChannelBandwidth: 'CBW20'
               NumUsers: 2
          UserPositions: [0 1]
    NumTransmitAntennas: 2
    NumSpaceTimeStreams: [1 1]
         SpatialMapping: 'Direct'
                    MCS: [4 8]
          ChannelCoding: {'BCC'  'LDPC'}
             APEPLength: [1024 2048]
          GuardInterval: 'Long'
                GroupID: 2

   Read-only properties:
             PSDULength: [1030 2065]
```

The configuration object settings reflect the updates specified. Default values are used for properties that were not modified.

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` | `string`

`NumUsers` — Number of users

1 (default) | 2 | 3 | 4

Number of users, specified as 1, 2, 3, or 4. (N_{Users})

Data Types: `double`

`UserPositions` — Position of users

[0 1] (default) | row vector of integers from 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` > 1.

Example: `[0 2 3]` indicates positions for three users, where the first user occupies position 0, the second user occupies position 2, and the third user occupies position 3.

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` | `string`

`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` — Enable signaling of a transmission with beamforming

`true` (default) | `false`

Enable 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` — Enable space-time block coding

`false` (default) | `true`

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) | `'LDPC'`

Type of forward error correction coding for the data field, specified as `'BCC'` (default) or `'LDPC'`. `'BCC'` indicates binary convolutional coding and `'LDPC'` indicates low density parity check coding. Providing a character vector or a single cell character vector defines the channel coding type for a single user or all users in a multiuser transmission. By providing a cell array different channel coding types can be specified per user for a multiuser transmission.

Data Types: `char` | `cell` | `string`

`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` | `string`

`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 1-371 configuration, returned as a `wlanVHTConfig` object. The properties of `cfgVHT` are described in wlanVHTConfig.

Definitions

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanDMGConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanS1GConfig` | `wlanVHTDataRecover` | `wlanVHTLTFDemodulate` | `wlanWaveformGenerator`

Topics

“Packet Size and Duration Dependencies”

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 1-381[22] time-domain waveform from the input user data bits, `psdu`, for the specified configuration object, `cfg`. See “VHT-Data Field Processing” on page 1-383 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.

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

```
psdu = randi([0 1],cfgVHT.PSDULength*8,1);
y = wlanVHTData(psdu,cfgVHT);
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 1-382), 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` | `string`

`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` | `string`

`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` — **Enable space-time block coding**
`false` (default) | `true`

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) | `'LDPC'`

Type of forward error correction coding for the data field, specified as `'BCC'` (default) or `'LDPC'`. `'BCC'` indicates binary convolutional coding and `'LDPC'` indicates low density parity check coding. Providing a character vector or a single cell character vector defines the channel coding type for a single user or all users in a multiuser transmission. By providing a cell array different channel coding types can be specified per user for a multiuser transmission.

Data Types: `char` | `cell` | `string`

`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` | `string`

`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.
- When undefined, `PSDULength` is returned as an empty, `[]`. This can happen when the set of property values for the object are in an invalid state.

`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.

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.

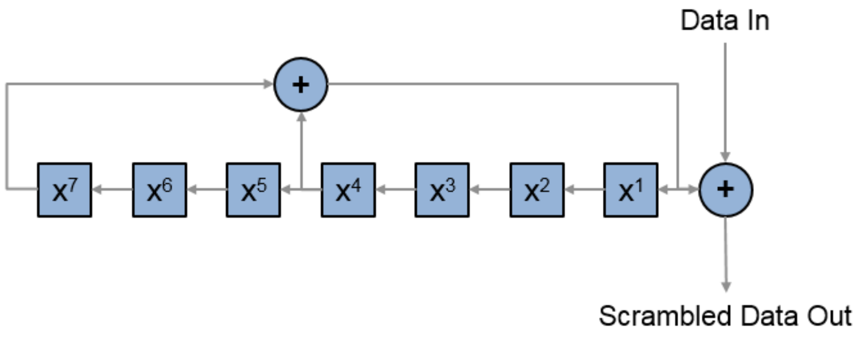
Data Types: `double`

`scramInit` — Scrambler initialization state

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

Initial scrambler state of the data scrambler for each packet generated, specified as an integer, a binary 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 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 vector.

The scrambler initialization used on the transmission data follows the process described in IEEE Std 802.11-2012, Section 18.3.5.5 and IEEE Std 802.11ad-2012, Section 21.3.9. The header and data fields that follow the scrambler initialization field (including data padding bits) are scrambled by XORing each bit with a length-127 periodic sequence generated by the polynomial $S(x) = x^7+x^4+1$. The octets of the PSDU (Physical Layer Service Data Unit) are placed into a bit stream, and within each octet, bit 0 (LSB) is first and bit 7 (MSB) is last. The generation of the sequence and the XOR operation are shown in this figure:



Conversion from integer to bits uses left-MSB orientation. For the initialization of the scrambler with decimal `1`, the bits are mapped to the elements shown.

Element	X^7	X^6	X^5	X^4	X^3	X^2	X^1
Bit Value	0	0	0	0	0	0	1

To generate the bit stream equivalent to a decimal, use `de2bi`. For example, for decimal `1`:

```
de2bi(1,7,'left-msb')
ans =

     0     0     0     0     0     0     1
```

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

Data Types: `double` | `int8`

Output Arguments

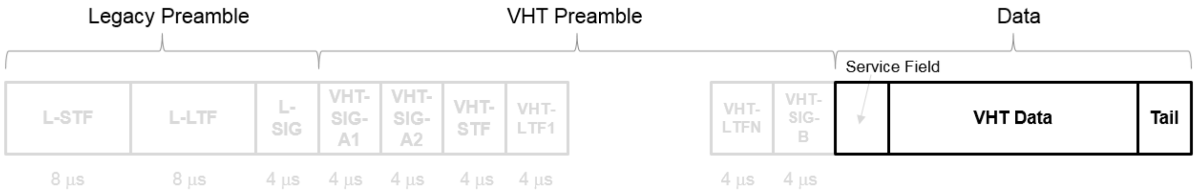
`y` — VHT-Data field time-domain waveform
matrix

“VHT-Data field” on page 1-381 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 1-383 for waveform generation details.

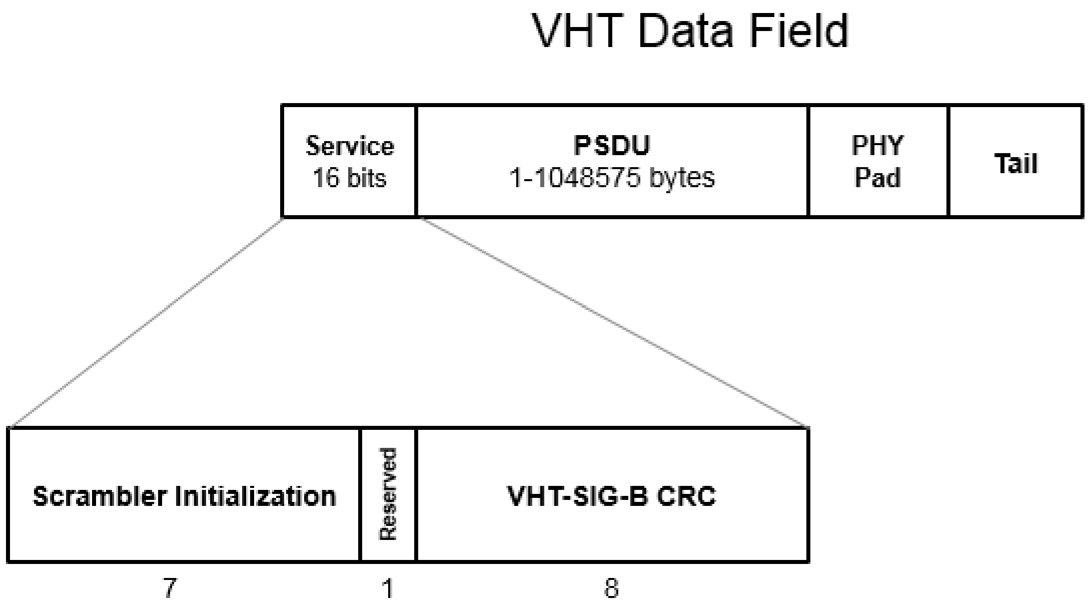
Definitions

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.

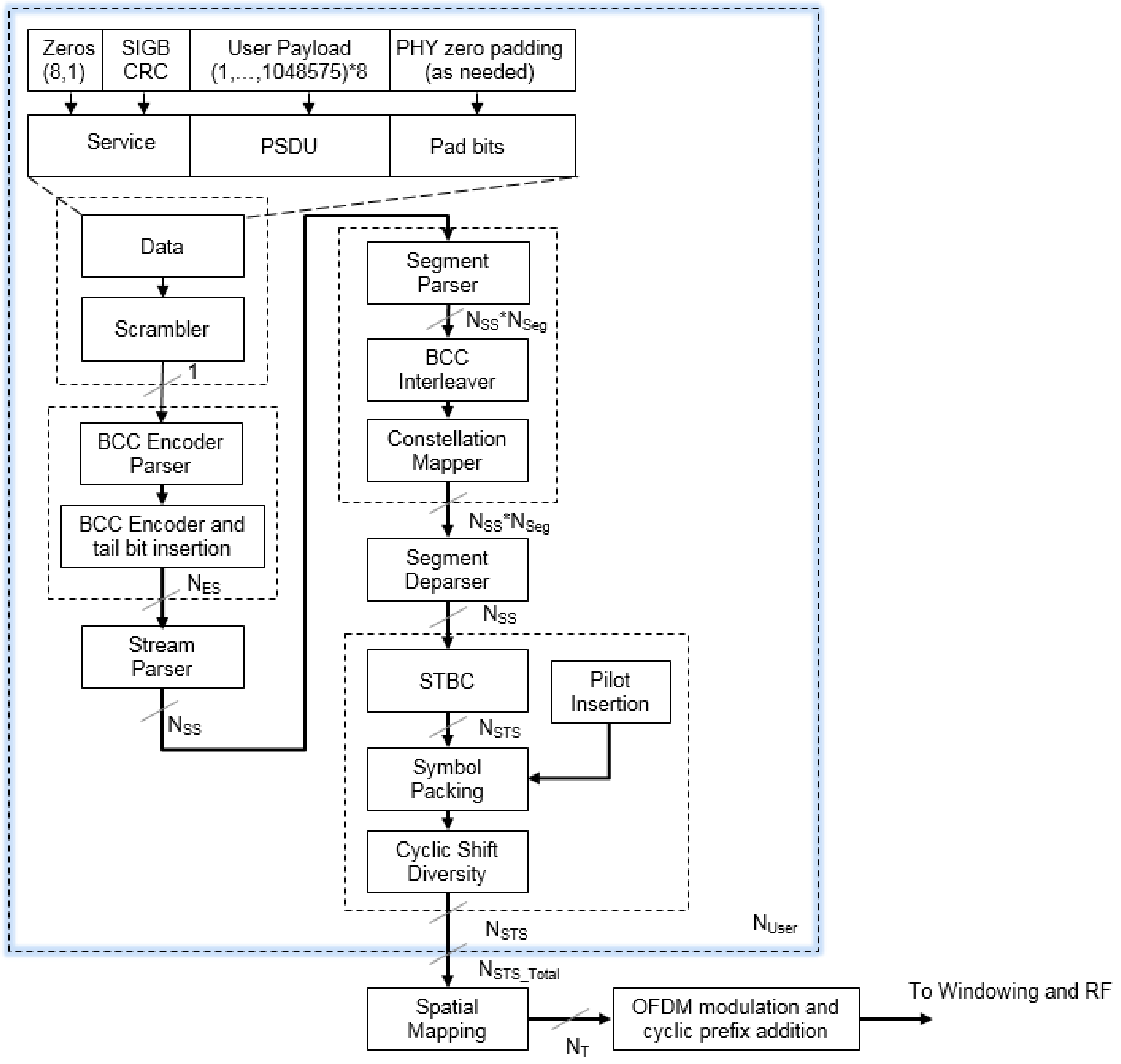
PSDU

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 1-381 encodes the service, “PSDU” on page 1-382, pad bits, and tail bits. The `wlanVHTData` function performs transmitter processing on the “VHT-Data field” on page 1-381 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.

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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,userNumber)
recBits = wlanVHTDataRecover(rxSig,chEst,noiseVarEst,cfg,userNumber,
numSTS)
recBits = wlanVHTDataRecover( ___ ,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 1-399[23] for a single-user transmission. Inputs include the received “VHT data field” on page 1-399 signal, the channel estimate, the noise variance estimate, and the format configuration object, `cfg`.

`recBits = wlanVHTDataRecover(rxSig,chEst,noiseVarEst,cfg,userNumber)` returns the recovered payload bits, in a multiuser transmission, for the user specified by `userNumber`.

`recBits = wlanVHTDataRecover(rxSig,chEst,noiseVarEst,cfg,userNumber, numSTS)` also specifies the number of space-time streams, `numSTS`, for a multiuser transmission.

`recBits = wlanVHTDataRecover( ___ ,cfgRec)` returns the recovered bits using the algorithm parameters specified in `cfgRec`.

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

`[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`.

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,'NumSpaceTimeStreams
```

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.

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

Recover VHT-Data Field in MU-MIMO Channel

Recover VHT-Data field bits for a multiuser transmission using channel estimation on a VHT-LTF field over a quasi-static fading channel.

Create a VHT configuration object having a 160 MHz channel bandwidth, two users, and four transmit antennas. Assign one space-time stream to the first user and three space-time streams to the second user.

```
cbw = 'CBW160';
numSTS = [1 3];
vht = wlanVHTConfig('ChannelBandwidth',cbw,'NumUsers',2, ...
    'NumTransmitAntennas',4,'NumSpaceTimeStreams',numSTS);
```

Because there are two users, the PSDU length is a 1-by-2 row vector.

```
psduLen = vht.PSDULength
```

```
psduLen =

        1050        3156
```

Generate multiuser input data. This data must be in the form of a 1-by- *N* cell array, where *N* is the number of users.

```
txDataBits{1} = randi([0 1],8*vht.PSDULength(1),1);
txDataBits{2} = randi([0 1],8*vht.PSDULength(2),1);
```

Generate VHT-LTF and VHT-Data field signals.

```
txVHTLTF  = wlanVHTLTF(vht);
txVHTData = wlanVHTData(txDataBits,vht);
```

Pass the data field for the first user through a 4x1 channel because it consists of a single space-time stream. Pass the second user's data through a 4x3 channel because it consists of three space-time streams. Apply white Gaussian noise to each user signal.

```
snr = 15;
H1 = 1/sqrt(2)*complex(randn(4,1),randn(4,1));
```

```
H2 = 1/sqrt(2)*complex(randn(4,3),randn(4,3));

rxVHTData1 = awgn(txVHTData*H1,snr,'measured');
rxVHTData2 = awgn(txVHTData*H2,snr,'measured');
```

Repeat the process for the VHT-LTF fields.

```
rxVHTLTF1  = awgn(txVHTLTF*H1,snr,'measured');
rxVHTLTF2  = awgn(txVHTLTF*H2,snr,'measured');
```

Calculate the received signal power for both users and use it to estimate the noise variance.

```
powerDB1 = 10*log10(var(rxVHTData1));
noiseVarEst1 = mean(10.^(0.1*(powerDB1-snr)));

powerDB2 = 10*log10(var(rxVHTData2));
noiseVarEst2 = mean(10.^(0.1*(powerDB2-snr)));
```

Estimate the channel characteristics using the VHT-LTF fields.

```
demodVHTLTF1 = wlanVHTLTFDemodulate(rxVHTLTF1,cbw,numSTS);
chanEst1 = wlanVHTLTFChannelEstimate(demodVHTLTF1,cbw,numSTS);

demodVHTLTF2 = wlanVHTLTFDemodulate(rxVHTLTF2,cbw,numSTS);
chanEst2 = wlanVHTLTFChannelEstimate(demodVHTLTF2,cbw,numSTS);
```

Recover VHT-Data field bits for the first user and compare against the original payload bits.

```
rxDataBits1 = wlanVHTDataRecover(rxVHTData1,chanEst1,noiseVarEst1,vht,1);
[~,ber1] = biterr(txDataBits{1},rxDataBits1)
```

```
ber1 =

    0.4983
```

Determine the number of bit errors for the second user.

```
rxDataBits2 = wlanVHTDataRecover(rxVHTData2,chanEst2,noiseVarEst2,vht,2);
[~,ber2] = biterr(txDataBits{2},rxDataBits2)
```

```
ber2 =
    0.0972
```

The bit error rates are quite high because there is no precoding to mitigate the interference between streams. This is especially evident for the user 1 receiver because it receives energy from the three streams intended for user 2. The example is intended to show the workflow and proper syntaxes for the LTF demodulate, channel estimation, and data recovery functions.

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. For multiuser transmissions, N_{STS} is the total number of space-time streams for all users. 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` | `string`

`NumUsers` — Number of users
1 (default) | 2 | 3 | 4

Number of users, specified as 1, 2, 3, or 4. (N_{Users})

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`

`STBC` — Enable space-time block coding
`false` (default) | `true`

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` | `string`

`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) | `'LDPC'`

Type of forward error correction coding for the data field, specified as `'BCC'` (default) or `'LDPC'`. `'BCC'` indicates binary convolutional coding and `'LDPC'` indicates low density parity check coding. Providing a character vector or a single cell character vector defines the channel coding type for a single user or all users in a multiuser transmission. By providing a cell array different channel coding types can be specified per user for a multiuser transmission.

Data Types: `char` | `cell` | `string`

`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

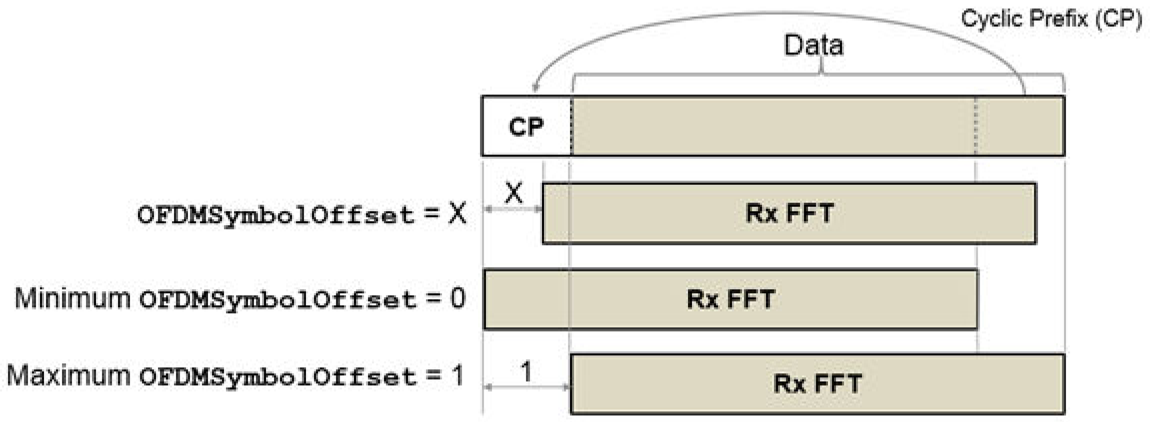
Algorithm parameters containing properties used during data recovery, specified as a `wlanRecoveryConfig` object. The configurable properties include OFDM symbol sampling offset, equalization method, and the type of pilot phase tracking. If you do not specify a `cfgRec` object, the default object property values as described in wlanRecoveryConfig 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` | `string`

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

Pilot phase tracking, specified as `'PreEQ'` or `'None'`.

- `'PreEQ'` — Enables pilot phase tracking, which is performed before any equalization operation.
- `'None'` — Pilot phase tracking does not occur.

Data Types: `char` | `string`

`MaximumLDPCIterationCount` — Maximum number of decoding iterations in LDPC
12 (default) | positive scalar integer

Maximum number of decoding iterations in LDPC, specified as a positive scalar integer. This parameter is applicable when channel coding is set to LDPC. For information on channel coding options, see `wlanVHTConfig` or `wlanHTConfig` for 802.11 format of interest.

Data Types: `double`

`EarlyTermination` — Enable early termination of LDPC decoding

`false` (default) | `true`

Enable early termination of LDPC decoding, specified as a logical. This parameter is applicable when channel coding is set to LDPC.

- When set to `false`, LDPC decoding completes the number of iterations specified by `MaximumLDPCIterationCount`, regardless of parity check status.
- When set to `true`, LDPC decoding terminates when all parity-checks are satisfied.

For information on channel coding options, see `wlanVHTConfig` or `wlanHTConfig` for 802.11 format of interest.

`userNumber` — Number of the user

integer from 1 to N_{Users}

Number of the user in a multiuser transmission, specified as an integer having a value from 1 to N_{Users}. N_{Users} is the total number of users.

`numSTS` — Number of space-time streams

1-by-N_{Users} vector of integers from 1 to 4

Number of space-time streams in a multiuser transmission, specified as a vector. The number of space-time streams is a 1-by-N_{Users} vector of integers from 1 to 4, where 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`

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 for `PSDULength` details. The output is for a single user as determined by `userNumber`.

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 assigned to the user. 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 1-399.

Limitations

`wlanVHTDataRecover` processing limitations, restrictions, and recommendations:

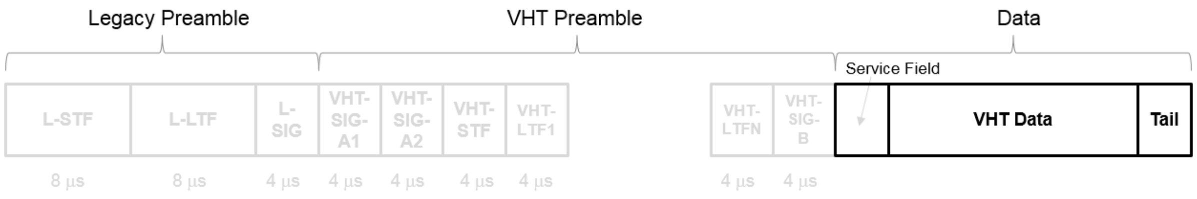
- If only VHT format PPDUs are processed, then `isa(cfgVHT, 'wlanVHTConfig')` must be `true`.

- 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`

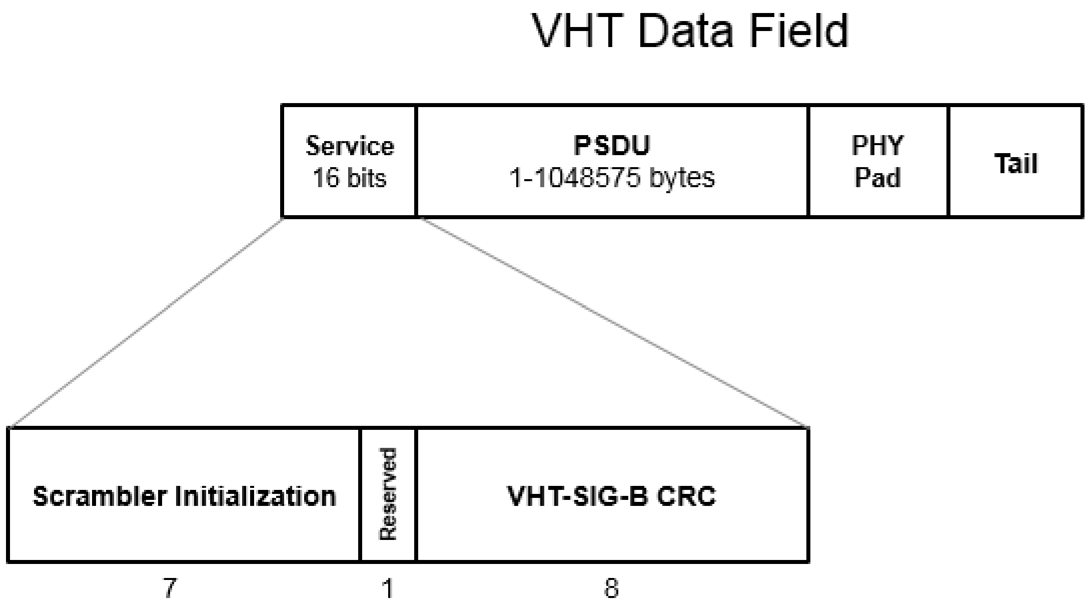
Definitions

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-405 [24] time-domain waveform for the specified configuration object. See “VHT-LTF Processing” on page 1-406 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
```

24. 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` | `string`

`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` | `string`

`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 1-405 time-domain waveform, returned as an (N_S × 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.

`ChannelBandwidth`	N_S
`'CBW20'`	`80`
`'CBW40'`	`160`
`'CBW80'`	`320`
`'CBW160'`	`640`

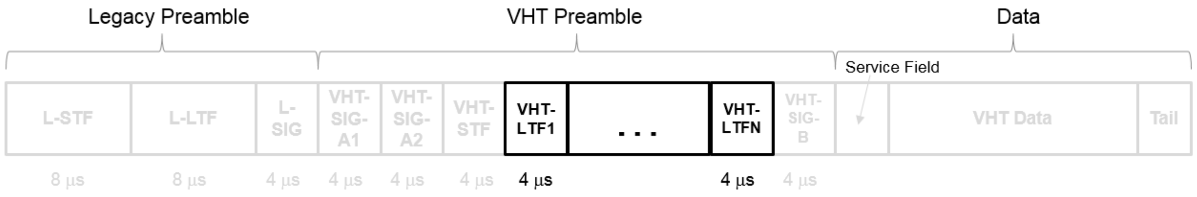
See “VHT-LTF Processing” on page 1-406 for waveform generation details.

Data Types: `double`
Complex Number Support: Yes

Definitions

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 1-405 is used for MIMO channel estimation and pilot subcarrier tracking. The number of OFDM symbols in the “VHT-LTF” on page 1-405 (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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanLLTF` | `wlanVHTConfig` | `wlanVHTData` | `wlanVHTLTFChannelEstimate` | `wlanVHTLTFDemodulate` | `wlanVHTSTF`

Introduced in R2015b

wlanVHTLTFDemodulate

Demodulate VHT-LTF waveform

Syntax

```
y = wlanVHTLTFDemodulate(x,cfg)
y = wlanVHTLTFDemodulate(x,cbw,numSTS)
y = wlanVHTLTFDemodulate( ___ ,OFDMSymbolOffset)
```

Description

`y = wlanVHTLTFDemodulate(x,cfg)` returns demodulated "VHT-LTF" on page 1-416[25] waveform `y` given time-domain input signal `x` and `wlanVHTConfig` object `cfg`.

`y = wlanVHTLTFDemodulate(x,cbw,numSTS)` demodulates the received signal for the specified channel bandwidth, `cbw`, and number of space-time streams, `numSTS`.

`y = wlanVHTLTFDemodulate( ___ ,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);
```

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

Add white noise to the signal.

```
rxVHTLTF = awgn(txVHTLTF,1);
```

Demodulate the received signal.

```
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.

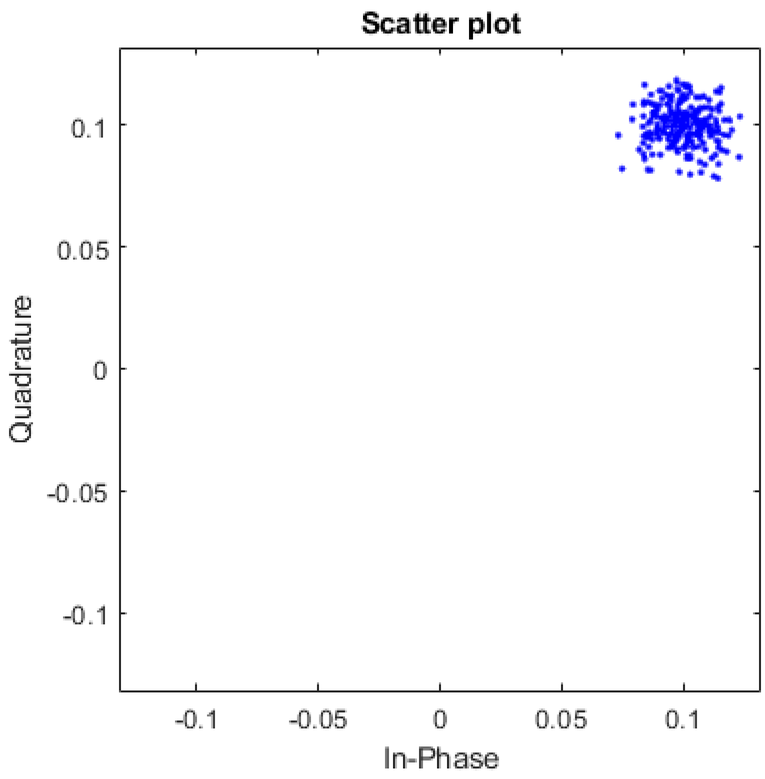
```
rxltfNoNoise = txltf * complex(0.1,0.1);
rxltf = awgn(rxltfNoNoise,20,'measured');
```

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 format object with two paths.

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

Generate a random PSDU and create the corresponding VHT waveform.

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

Pass the signal through a TGac 2x2 MIMO channel.

```
tgacChan = wlanTGacChannel('NumTransmitAntennas',2,'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
rxSigNoNoise = tgacChan(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);
awgnChan = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
rxSig = awgnChan(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
```

Recover VHT-Data Field in MU-MIMO Channel

Recover VHT-Data field bits for a multiuser transmission using channel estimation on a VHT-LTF field over a quasi-static fading channel.

Create a VHT configuration object having a 160 MHz channel bandwidth, two users, and four transmit antennas. Assign one space-time stream to the first user and three space-time streams to the second user.

```
cbw = 'CBW160';
numSTS = [1 3];
vht = wlanVHTConfig('ChannelBandwidth',cbw,'NumUsers',2, ...
    'NumTransmitAntennas',4,'NumSpaceTimeStreams',numSTS);
```

Because there are two users, the PSDU length is a 1-by-2 row vector.

```
psduLen = vht.PSDULength
```

```
psduLen =

        1050        3156
```

Generate multiuser input data. This data must be in the form of a 1-by- N cell array, where N is the number of users.

```
txDataBits{1} = randi([0 1],8*vht.PSDULength(1),1);
txDataBits{2} = randi([0 1],8*vht.PSDULength(2),1);
```

Generate VHT-LTF and VHT-Data field signals.

```
txVHTLTF  = wlanVHTLTF(vht);
txVHTData = wlanVHTData(txDataBits,vht);
```

Pass the data field for the first user through a 4x1 channel because it consists of a single space-time stream. Pass the second user's data through a 4x3 channel because it consists of three space-time streams. Apply white Gaussian noise to each user signal.

```
snr = 15;
H1 = 1/sqrt(2)*complex(randn(4,1),randn(4,1));
H2 = 1/sqrt(2)*complex(randn(4,3),randn(4,3));

rxVHTData1 = awgn(txVHTData*H1,snr,'measured');
rxVHTData2 = awgn(txVHTData*H2,snr,'measured');
```

Repeat the process for the VHT-LTF fields.

```
rxVHTLTF1  = awgn(txVHTLTF*H1,snr,'measured');
rxVHTLTF2  = awgn(txVHTLTF*H2,snr,'measured');
```

Calculate the received signal power for both users and use it to estimate the noise variance.

```
powerDB1 = 10*log10(var(rxVHTData1));
noiseVarEst1 = mean(10.^(0.1*(powerDB1-snr)));

powerDB2 = 10*log10(var(rxVHTData2));
noiseVarEst2 = mean(10.^(0.1*(powerDB2-snr)));
```

Estimate the channel characteristics using the VHT-LTF fields.

```
demodVHTLTF1 = wlanVHTLTFDemodulate(rxVHTLTF1,cbw,numSTS);
chanEst1 = wlanVHTLTFChannelEstimate(demodVHTLTF1,cbw,numSTS);

demodVHTLTF2 = wlanVHTLTFDemodulate(rxVHTLTF2,cbw,numSTS);
chanEst2 = wlanVHTLTFChannelEstimate(demodVHTLTF2,cbw,numSTS);
```

Recover VHT-Data field bits for the first user and compare against the original payload bits.

```
rxDataBits1 = wlanVHTDataRecover(rxVHTData1,chanEst1,noiseVarEst1,vht,1);
[~,ber1] = biterr(txDataBits{1},rxDataBits1)
```

```
ber1 =

    0.4983
```

Determine the number of bit errors for the second user.

```
rxDataBits2 = wlanVHTDataRecover(rxVHTData2,chanEst2,noiseVarEst2,vht,2);
[~,ber2] = biterr(txDataBits{2},rxDataBits2)
```

```
ber2 =

    0.0972
```

The bit error rates are quite high because there is no precoding to mitigate the interference between streams. This is especially evident for the user 1 receiver because it receives energy from the three streams intended for user 2. The example is intended to show the workflow and proper syntaxes for the LTF demodulate, channel estimation, and data recovery functions.

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`

`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` | `string`

`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`

`cbw` — Channel bandwidth

`'CBW20'` | `'CBW40'` | `'CBW80'` | `'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.

Data Types: `char` | `string`

`numSTS` — Number of space-time streams

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]` indicates that one space-time stream is assigned to user 1, three space-time streams are assigned to user 2, and two space-time streams are assigned to user 3.

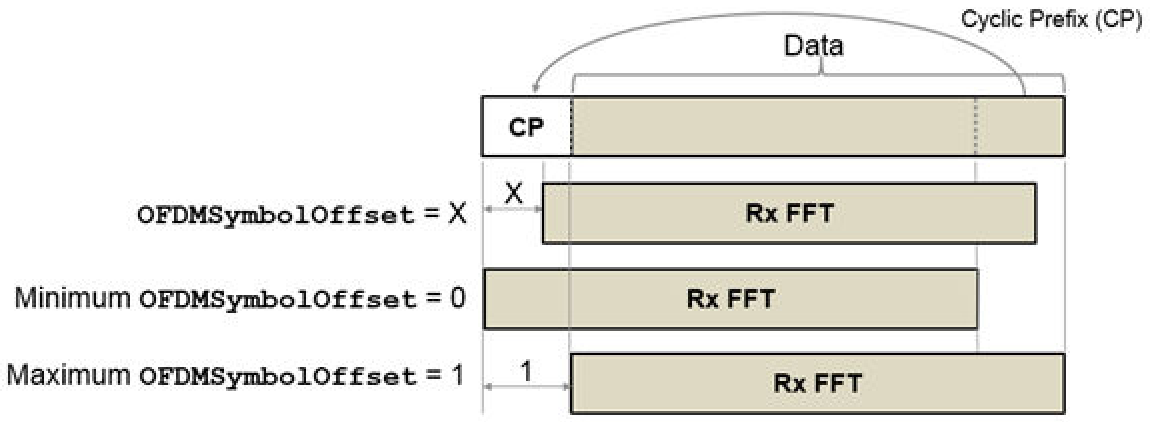
Note The sum of the space-time stream vector elements must not exceed eight.

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.

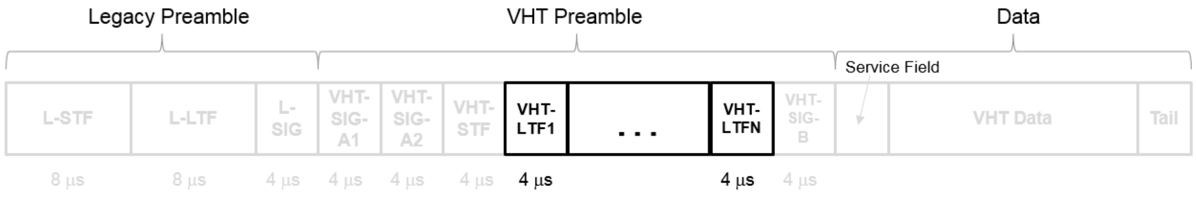
If the received VHT-LTF signal, `x`, is empty, then the output is also empty.

Data Types: `double`

Definitions

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-426[26] time-domain waveform for the specified configuration object. See “VHT-SIG-A Processing” on page 1-428 for waveform generation details.

`[y,bits] = wlanVHTSIGA(cfg)` also outputs “VHT-SIG-A” on page 1-426 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)
```

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

```
ans =

   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 =

  2x1 int8 column vector

   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` | `string`

`NumUsers` — Number of users

1 (default) | 2 | 3 | 4

Number of users, specified as 1, 2, 3, or 4. (N_{Users})

Data Types: `double`

`UserPositions` — Position of users

[0 1] (default) | row vector of integers from 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` > 1.

Example: `[0 2 3]` indicates positions for three users, where the first user occupies position 0, the second user occupies position 2, and the third user occupies position 3.

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` | `string`

`Beamforming` — Enable signaling of a transmission with beamforming

`true` (default) | `false`

Enable 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` — Enable space-time block coding

`false` (default) | `true`

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) | `'LDPC'`

Type of forward error correction coding for the data field, specified as `'BCC'` (default) or `'LDPC'`. `'BCC'` indicates binary convolutional coding and `'LDPC'` indicates low density parity check coding. Providing a character vector or a single cell character vector defines the channel coding type for a single user or all users in a multiuser transmission. By providing a cell array different channel coding types can be specified per user for a multiuser transmission.

Data Types: `char` | `cell` | `string`

`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` | `string`

`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 1-426 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.

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

See “VHT-SIG-A Processing” on page 1-428 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 1-426, returned as a 48-bit column vector.

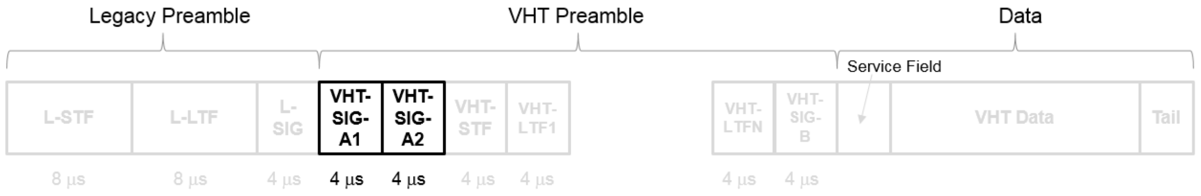
Data Types: `int8`

Definitions

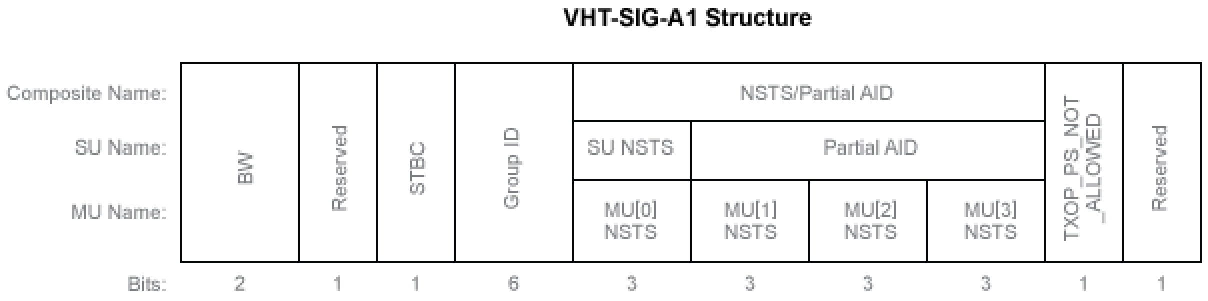
VHT-SIG-A

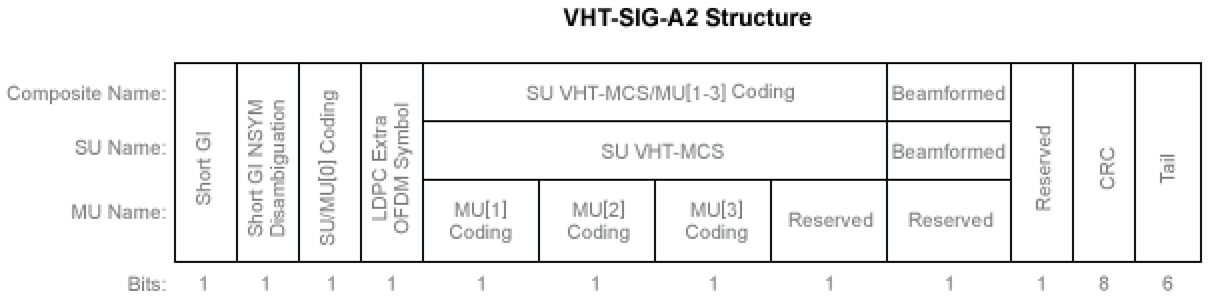
The very high throughput signal A (VHT-SIG-A) field contains information required to interpret VHT format packets. Similar to the non-HT signal (L-SIG) field for the non-HT OFDM format, this field stores the actual rate value, channel coding, guard interval, MIMO scheme, and other configuration details 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.





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.

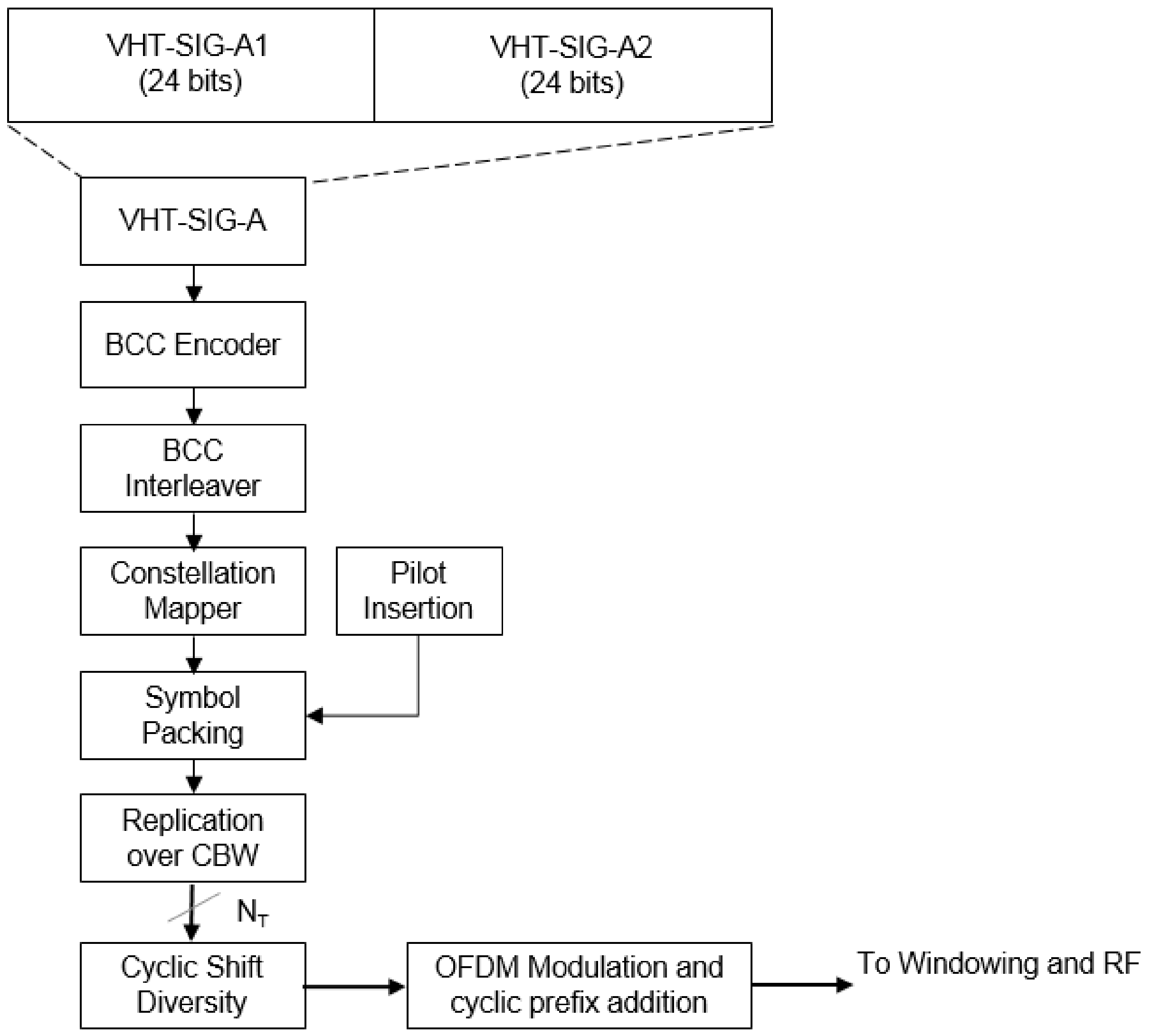
- **SU/MU[0] Coding** — A bit field that indicates if convolutional or LDPC coding is used for a single user or for user MU[0] in a multi-user scenario.
- **LDPC Extra OFDM Symbol** — A bit that indicates if an extra OFDM symbol is required to transmit the data field.
- **MCS** — A four-bit field.
  - 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 1-426 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 1-426 field and outputs the time-domain waveform.


VHT-SIG-A1 (24 bits)
VHT-SIG-A2 (24 bits)
VHT-SIG-A
BCC Encoder
BCC Interleaver
Constellation Mapper
Pilot Insertion
Symbol Packing
Replication over CBW
N_T
Cyclic Shift Diversity
OFDM Modulation and cyclic prefix addition
To Windowing and RF

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-440[27] field. Inputs include the received "VHT-SIG-A" on page 1-440 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`.

Examples

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

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 =

  logical

   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 =

  logical

   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.

```
awgnChan = comm.AWGNChannel('NoiseMethod','Variance','Variance',noiseVar);
rxLLTF = awgnChan(txLLTF);
rxSig = awgnChan(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 =

  logical

   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, 'NumSpaceT
```

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.

```
mimoChan = comm.MIMOChannel('SampleRate', 20e6);
rxLLTF = awgn(mimoChan(txLLTF), 15);
rxVHTSIGA = awgn(mimoChan(txVHTSIGA),15);
```

Demodulate the L-LTF signal. To generate a channel estimate, use the demodulated L-LTF.

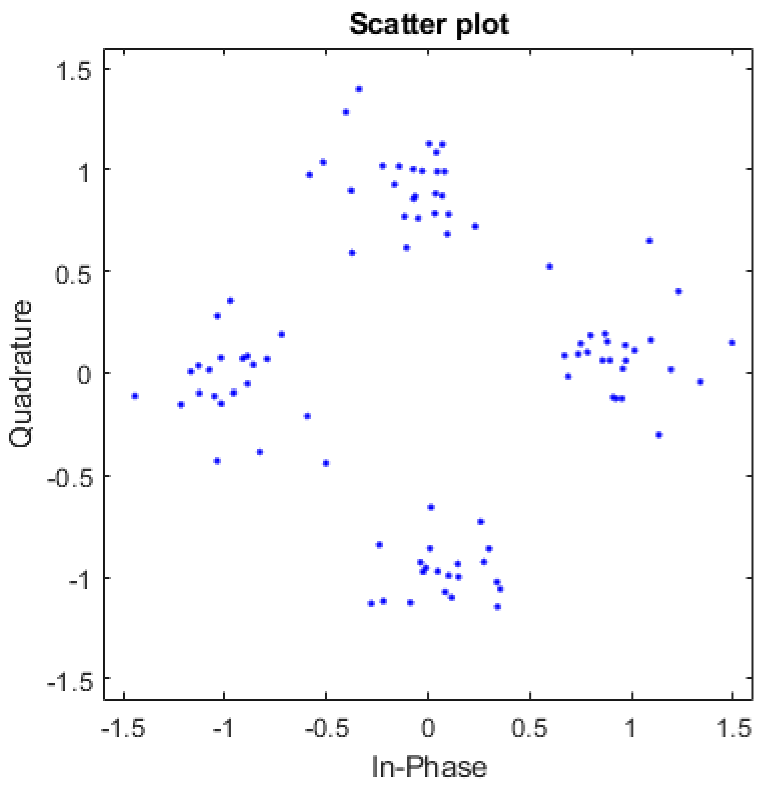
```
demodLLTF = wlanLLTFDemodulate(rxLLTF, chanBW, 1);
chanEst = wlanLLTFChannelEstimate(demodLLTF, chanBW);
```

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

Channel Bandwidth	N_S
`'CBW80'`	640
`'CBW160'`	1280

N_R is the number of receive antennas.

Data Types: `double`

`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 1-440.

Data Types: `double`

`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 `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

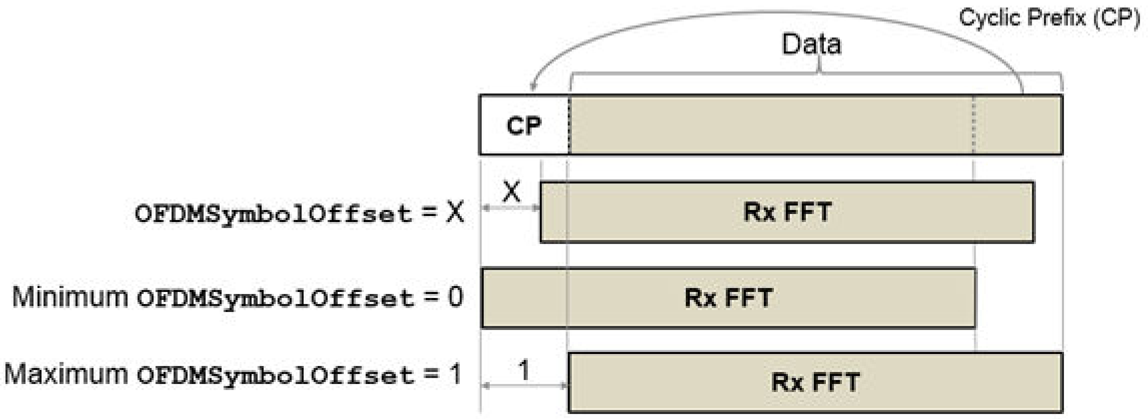
Data Types: `char` | `string`

`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` | `string`

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

Pilot phase tracking, specified as `'PreEQ'` or `'None'`.

- `'PreEQ'` — Enables pilot phase tracking, which is performed before any equalization operation.
- `'None'` — Pilot phase tracking does not occur.

Data Types: `char` | `string`

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 1-440 for more information.

`failCRC` — CRC failure check

`true` | `false`

CRC failure check, returned as `true` if the CRC check fails or `false` if the CRC check passes.

`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 1-440 field.

`cpe` — Common phase error

column vector

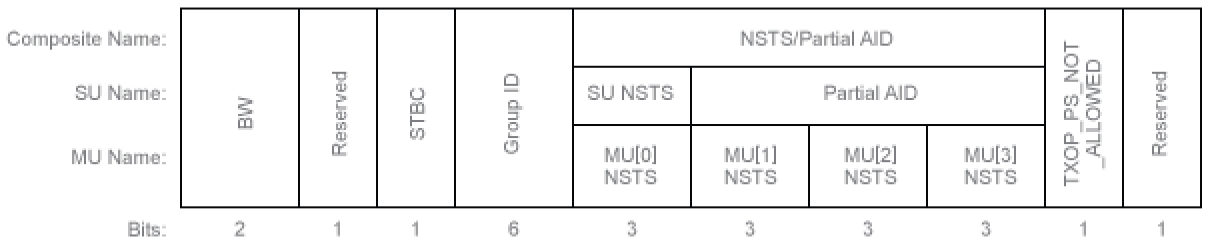
Common phase error in radians, returned as a 2-by-1 column vector.

Definitions

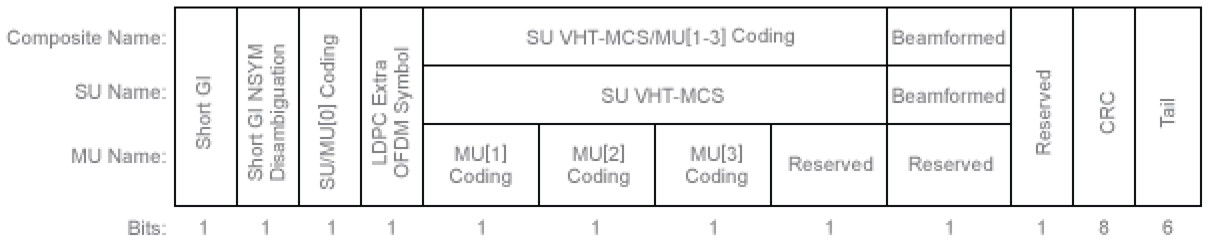
VHT-SIG-A

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



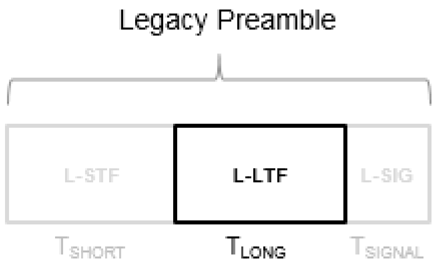
VHT-SIG-A2 Structure



For VHT-SIG-A field bit details, refer to IEEE Std 802.11ac-2013 [1], Table 22-12.

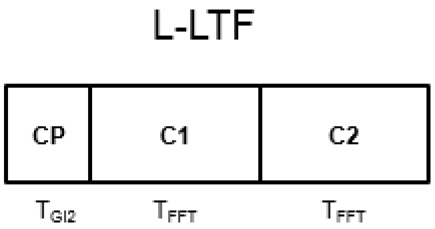
L-LTF

The legacy long training field (L-LTF) is the second field in the 802.11 OFDM PLCP legacy preamble. The L-LTF is a component of VHT, HT, and non-HT PPDUs.



Channel estimation, fine frequency offset estimation, and fine symbol timing offset estimation rely on the L-LTF.

The L-LTF is composed of a cyclic prefix (CP) followed by two identical long training symbols (C1 and C2). The CP consists of the second half of the long training symbol.



The L-LTF duration varies with channel bandwidth.

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration ($T_{GI2} = T_{FFT} / 2$)	L-LTF Duration ($T_{LONG} = T_{GI2} + 2 \times T_{FFT}$)
20, 40, 80, and 160	312.5	3.2 μs	1.6 μs	8 μs

Channel Bandwidth (MHz)	Subcarrier Frequency Spacing, Δ_F (kHz)	Fast Fourier Transform (FFT) Period ($T_{FFT} = 1 / \Delta_F$)	Cyclic Prefix or Training Symbol Guard Interval (GI2) Duration ($T_{GI2} = T_{FFT} / 2$)	L-LTF Duration ($T_{LONG} = T_{GI2} + 2 \times T_{FFT}$)
10	156.25	6.4 µs	3.2 µs	16 µs
5	78.125	12.8 µs	6.4 µs	32 µs

PPDU

PLCP protocol data unit

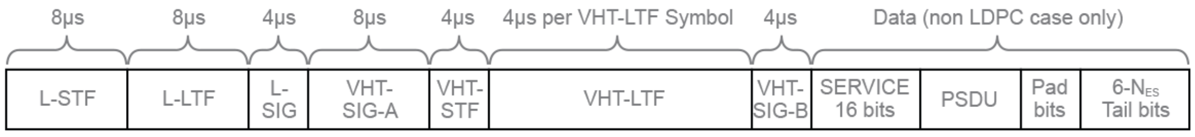
The PPDU is the complete PLCP frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

Algorithms

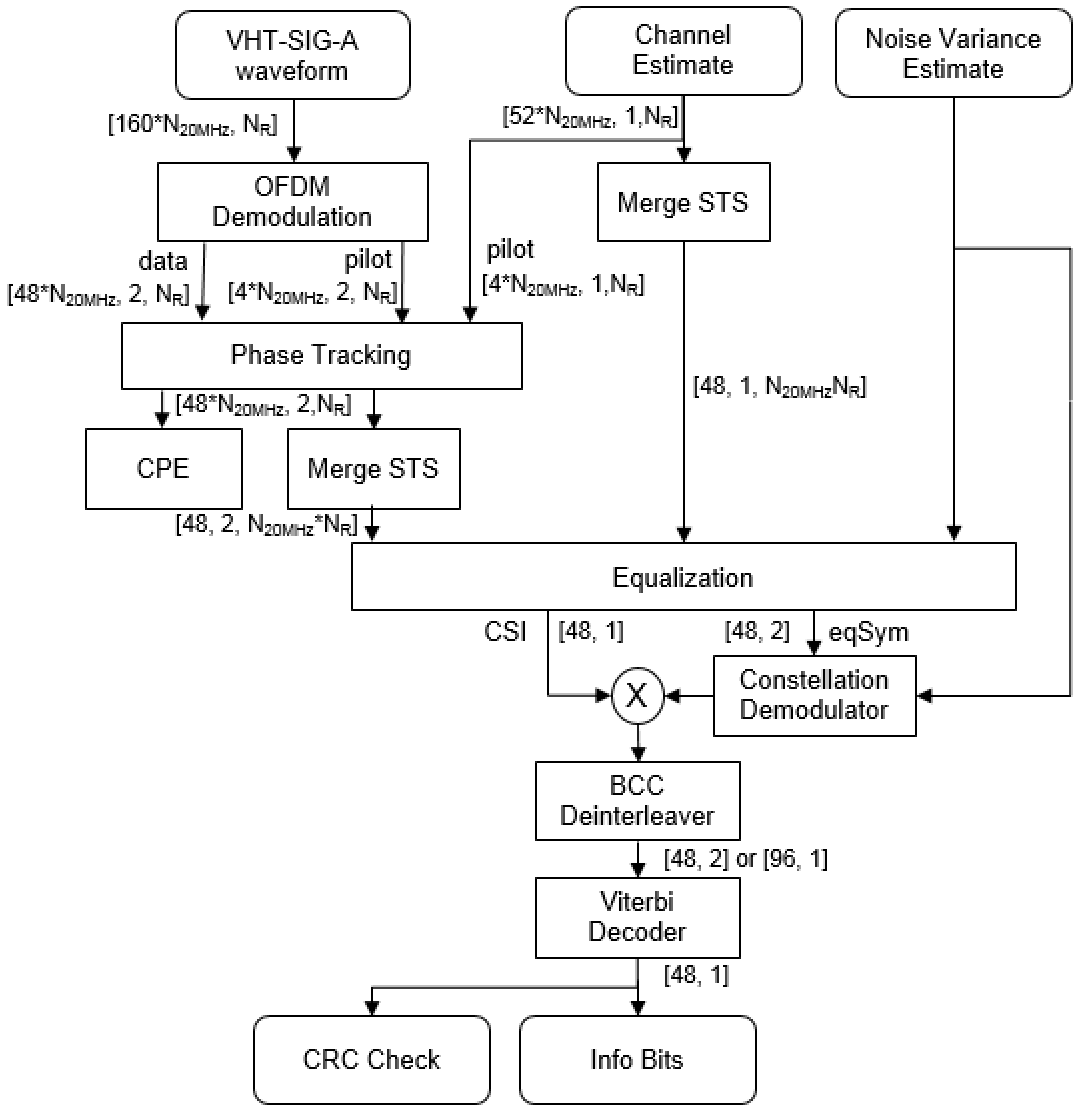
VHT-SIG-A Recovery

The “VHT-SIG-A” on page 1-440 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 1-442.

VHT Format PPDU



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 1-440 field.



For "VHT-SIG-A" on page 1-440 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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-450[28] time-domain waveform for the specified configuration object. See “VHT-SIG-B Processing” on page 1-452 for waveform generation details.

`[y,bits] = wlanVHTSIGB(cfg)` also outputs “VHT-SIG-B” on page 1-450 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 =
```

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

```
   320     1
```

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` | `string`

`NumUsers` — Number of users

1 (default) | 2 | 3 | 4

Number of users, specified as 1, 2, 3, or 4. (N_{Users})

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` | `string`

`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

`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`

MCS	Modulation	Coding Rate
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 1-450 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.

`ChannelBandwidth`	N_S
`'CBW20'`	`80`
`'CBW40'`	`160`
`'CBW80'`	`320`
`'CBW160'`	`640`

See “VHT-SIG-B Processing” on page 1-452. 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 1-450 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 1-452. for waveform generation details.

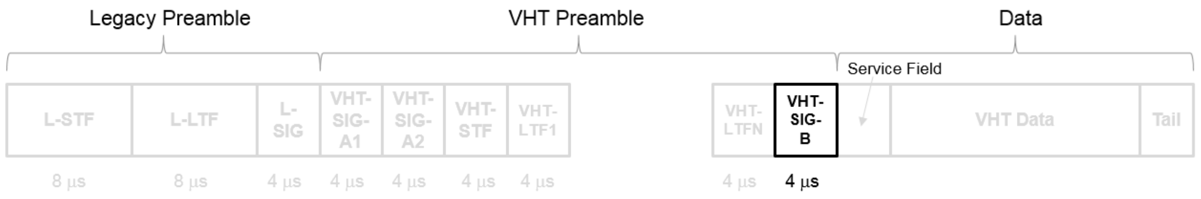
Data Types: `int8`

Definitions

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.

Field	VHT MU PPDU Allocation (bits)			VHT SU PPDU Allocation (bits)			Descriptio n
	20 MHz	40 MHz	80 MHz, 160 MHz	20 MHz	40 MHz	80 MHz, 160 MHz	
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 terminat e the convoluti onal code.
Total # bits	26	27	29	26	27	29	
Bit field repetitio n	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 1-450 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 1-450 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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanVHTConfig` | `wlanVHTData` | `wlanVHTLTF` | `wlanVHTSIGBRecover`

Introduced in R2015b

wlanVHTSIGBRecover

Recover VHT-SIG-B information bits

Syntax

```
recBits = wlanVHTSIGBRecover(rxSig,chEst,noiseVarEst,cbw)
recBits = wlanVHTSIGBRecover(rxSig,chEst,noiseVarEst,cbw,userNumber,
numSTS)
recBits = wlanVHTSIGBRecover(___,cfgRec)

[recBits,eqSym] = wlanVHTSIGBRecover(___)
[recBits,eqSym,cpe] = wlanVHTSIGBRecover(___)
```

Description

`recBits = wlanVHTSIGBRecover(rxSig,chEst,noiseVarEst,cbw)` returns the recovered information bits from the “VHT-SIG-B” on page 1-463[29] field for a single-user transmission. Inputs include the received “VHT-SIG-B” on page 1-463 field, the channel estimate, the noise variance estimate, and the channel bandwidth.

`recBits = wlanVHTSIGBRecover(rxSig,chEst,noiseVarEst,cbw,userNumber, numSTS)` returns the recovered information bits of a multiuser transmission for the user of interest, `userNumber`, and the number of space-time streams, `numSTS`.

`recBits = wlanVHTSIGBRecover(___,cfgRec)` specifies algorithm information using `wlanRecoveryConfig` object `cfgRec`.

`[recBits,eqSym] = wlanVHTSIGBRecover(___)` returns the equalized symbols, `eqSym`, using the arguments from previous syntaxes.

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

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

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.

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 =

  logical

   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.

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

rxSig = awgnChan(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 for an SNR=10 dB and a receiver that has a 9 dB noise figure. 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, ...
    'NumSpaceTimeStreams',2);
```

```
txVHTLTF = wlanVHTLTF(vht);
[txVHTSIGB,txVHTSIGBBits] = wlanVHTSIGB(vht);
```

Pass the VHT-LTF and VHT-SIG-B waveforms through a 2x2 TGac channel.

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

Add white noise for an SNR = 10dB.

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

rxVHTLTF = chNoise(rxVHTLTF);
rxVHTSIGB = chNoise(rxVHTSIGB);
```

Add additional white noise corresponding to a receiver with a 9 dB noise figure. The noise variance is equal to $k*T*B*F$, where k is Boltzmann's constant, T is the ambient temperature, B is the channel bandwidth (sample rate), and F is the receiver noise figure.

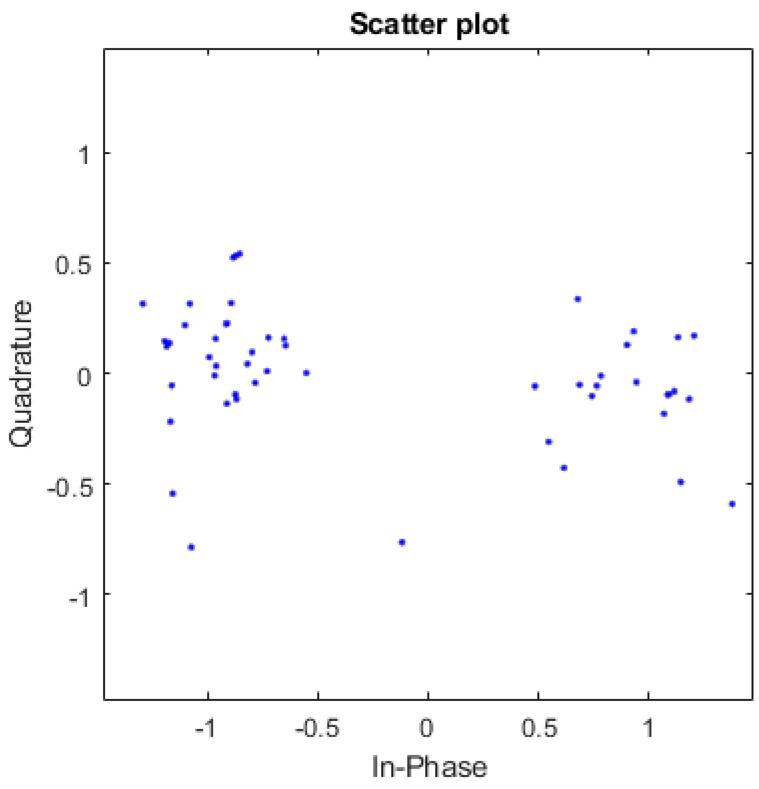
```
nVar = 10^((-228.6+10*log10(290)+10*log10(fs)+9)/10);
rxNoise = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);

rxVHTLTF = rxNoise(rxVHTLTF);
rxVHTSIGB = rxNoise(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 =

    0.0318
```

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`

`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. For multiuser transmissions, N_{STS} is the total number of space-time streams for all users . 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`

`ChannelBandwidth`	Number of Occupied Subcarriers (N_{ST})	Number of Data Subcarriers (N_{SD})	Number of Pilot Subcarriers (N_{SP})
`'CBW160'`	484	468	16

The channel estimate is based on the “VHT-LTF” on page 1-465.

`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 `'CBW20'`, `'CBW40'`, `'CBW80'`, or `'CBW160'`.

Data Types: `char` | `string`

`userNumber` — Number of the user
integer from 1 to N_{Users}

Number of the user in a multiuser transmission, specified as an integer having a value from 1 to N_{Users}. N_{Users} is the total number of users.

Data Types: `double`

`numSTS` — Number of space-time streams
1-by-N_{Users} vector of integers from 1 to 4

Number of space-time streams in a multiuser transmission, specified as a vector. The number of space-time streams is a 1-by-N_{Users} vector of integers from 1 to 4, where 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`

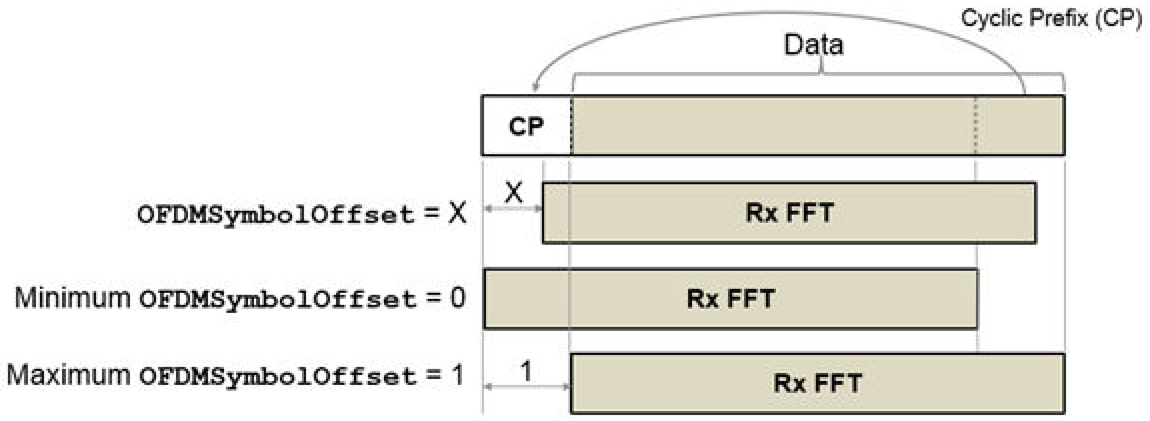
`cfgRec` — Algorithm parameters
`wlanRecoveryConfig` object

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` | `string`

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

Pilot phase tracking, specified as `'PreEQ'` or `'None'`.

- `'PreEQ'` — Enables pilot phase tracking, which is performed before any equalization operation.
- `'None'` — Pilot phase tracking does not occur.

Data Types: `char` | `string`

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 output is for a single user as determined by `userNumber`.

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 1-463 for information about the meaning of each bit in the field.

`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`

`cpe` — Common phase error
column vector

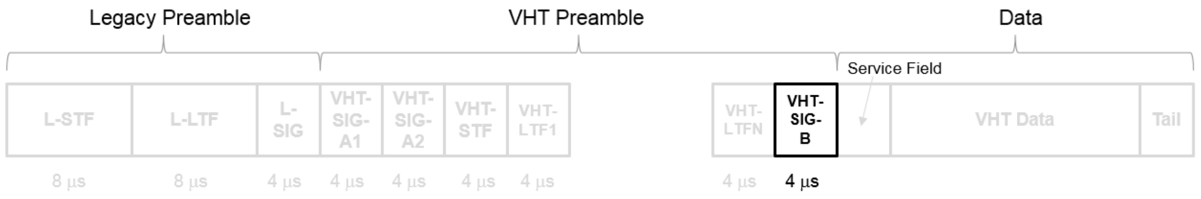
Common phase error in radians, returned as a scalar.

Definitions

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.

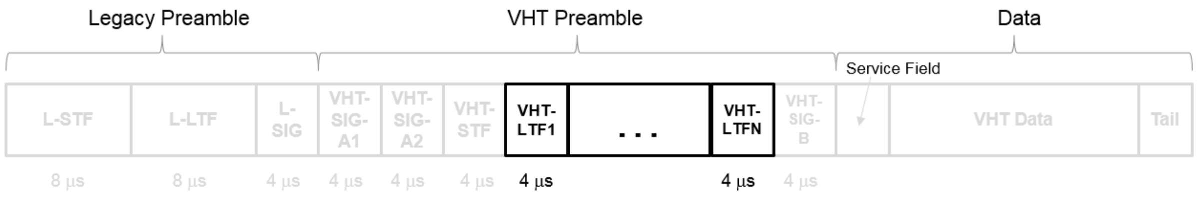
Field	VHT MU PPDU Allocation (bits)			VHT SU PPDU Allocation (bits)			Description
	20 MHz	40 MHz	80 MHz, 160 MHz	20 MHz	40 MHz	80 MHz, 160 MHz	
VHT-SIG-B	B0-15 (16)	B0-16 (17)	B0-18 (19)	B0-16 (17)	B0-18 (19)	B0-20 (21)	A variable-length field that indicates the size of the data payload in four-byte 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

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	
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 convolutional code.
Total # bits	26	27	29	26	27	29	
Bit field repetition	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

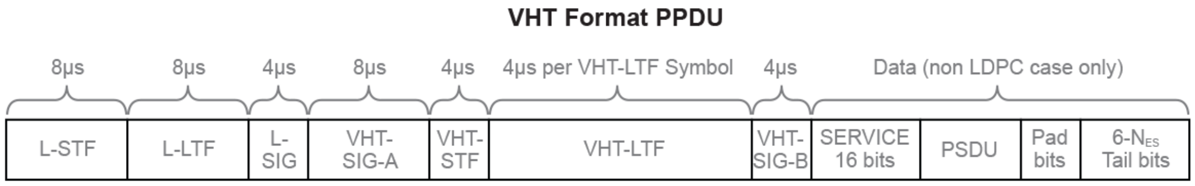
PLCP protocol data unit

The PPDU is the complete PLCP frame, including PLCP headers, MAC headers, the MAC data field, and the MAC and PLCP trailers.

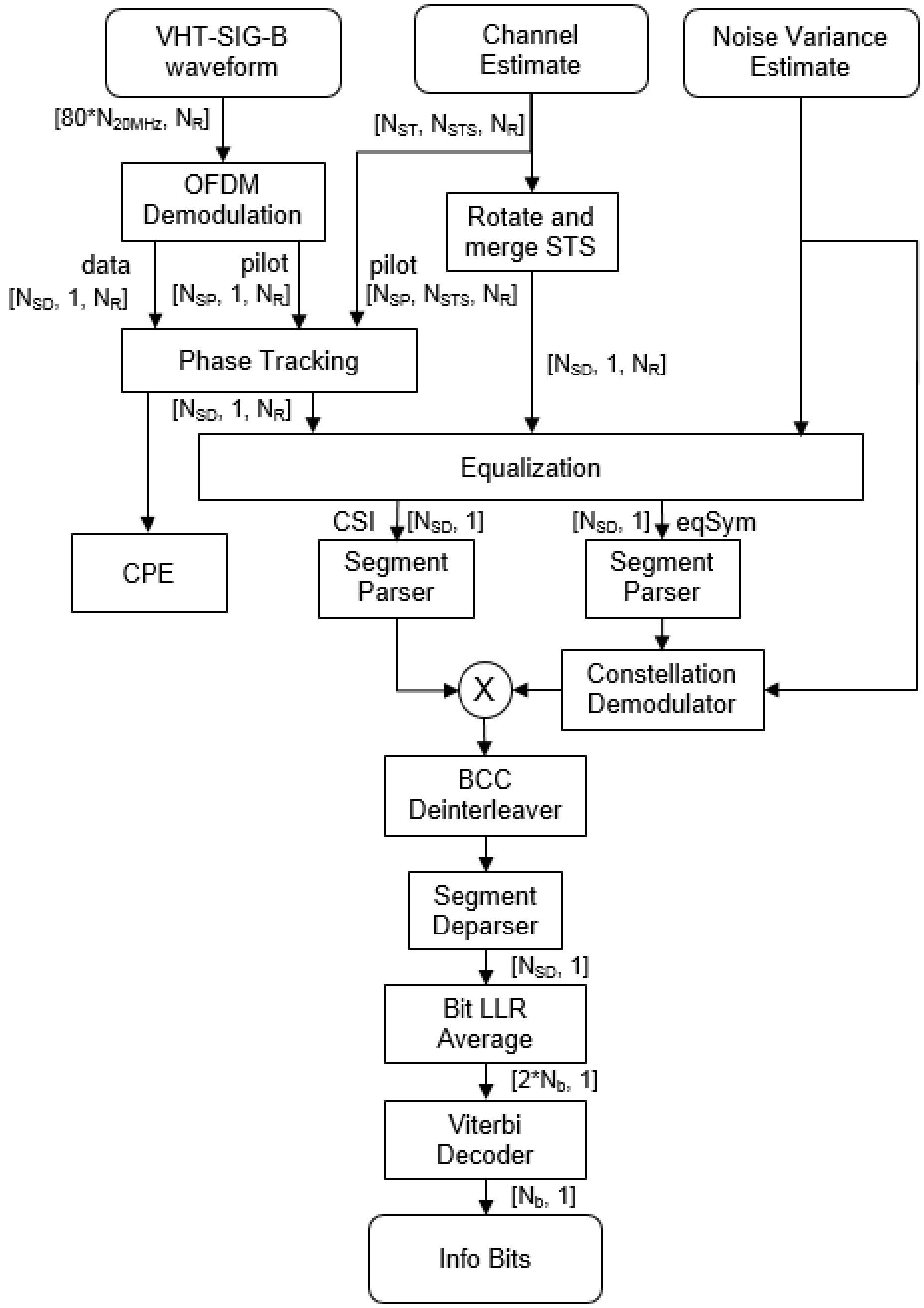
Algorithms

VHT-SIG-B Recovery

The “VHT-SIG-B” on page 1-463 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 1-463 field. For multiuser transmissions, recovering the VHT-SIG-B field provides packet length and MCS information for each user.


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
CSI
[NSD, 1]
[NSD, 1]
eqSym
CPE
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 1-463 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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

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 1-473[30] time-domain waveform for the specified configuration object. See “VHT-STF Processing” on page 1-474 for waveform generation details.

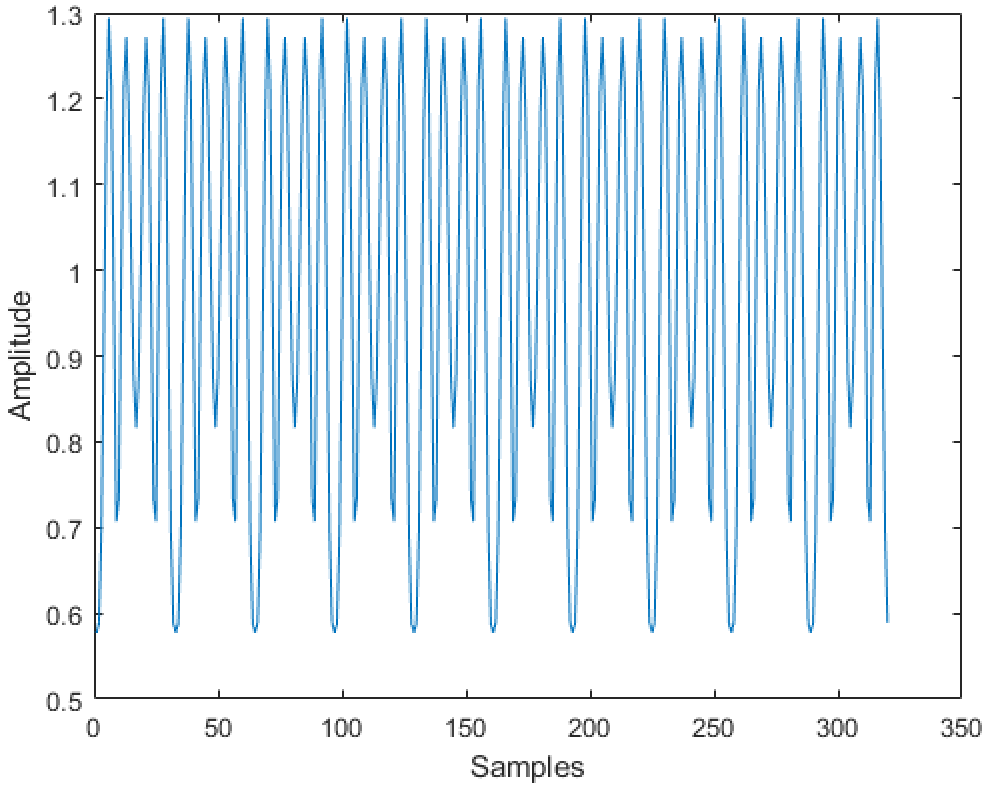
Examples

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')
```

30. 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` | `string`

`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` | `string`

`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 1-473 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.

`ChannelBandwidth`	N_S
`'CBW20'`	`80`
`'CBW40'`	`160`
`'CBW80'`	`320`
`'CBW160'`	`640`

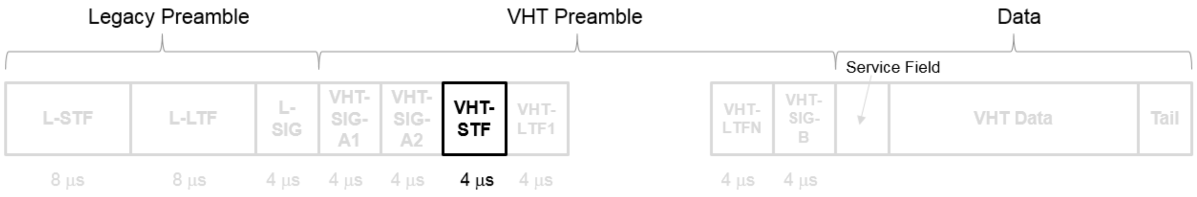
See "VHT-STF Processing" on page 1-474 for waveform generation details.

Data Types: `double`
Complex Number Support: Yes

Definitions

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 1-473 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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanLSTF` | `wlanVHTConfig` | `wlanVHTLTF` | `wlanVHTSIGA`

Introduced in R2015b

wlanWaveformGenerator

Generate WLAN waveform

Syntax

```
waveform = wlanWaveformGenerator(bits,cfgFormat)
waveform = wlanWaveformGenerator(bits,cfgFormat,Name,Value)
```

Description

`waveform = wlanWaveformGenerator(bits,cfgFormat)` generates a waveform for the specified information bits, and format configuration. For more information, see “IEEE 802.11 PPDU Format” on page 1-482.

`waveform = wlanWaveformGenerator(bits,cfgFormat,Name,Value)` overrides default generator configuration values using one or more `Name,Value` pair arguments.

Examples

Generate VHT Waveform

Generate a time-domain signal for an 802.11ac VHT transmission with one packet.

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;
```

Generate the transmission waveform.

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

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.

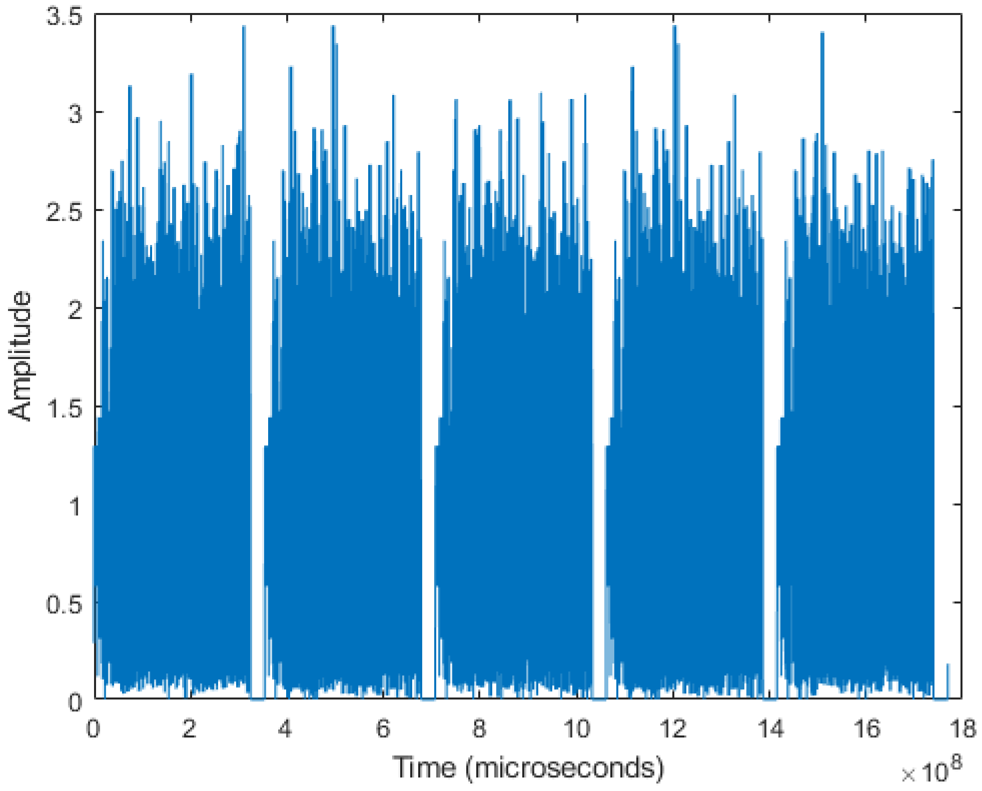
Create a VHT configuration object and confirm the channel bandwidth for scaling the *x*-axis of the plot.

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

```
ans =

    'CBW80'
```

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

```
numPkts = 5;
scramInit = randi([1 127],numPkts,1);
txWaveform = wlanWaveformGenerator([1;0;0;1],vht,'NumPackets',numPkts,'IdleTime',30e-6,
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 `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

`wlanDMGConfig` object | `wlanS1GConfig` object | `wlanVHTConfig` object | `wlanHTConfig` object | `wlanNonHTConfig` object

Packet format configuration, specified as a `wlanDMGConfig`, `wlanS1GConfig`, `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:

- wlanDMGConfig
- wlanS1GConfig
- wlanVHTConfig
- wlanHTConfig
- wlanNonHTConfig

The data rate and PSDU length of generated PPDUs is determined based on the properties of the packet format configuration object.

`Name,Value` — Name-Value Pair Arguments

`Name1,Value1,...,NameN,ValueN`

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. The default value is 0. If `IdleTime` is not set to the default value, it must be:

- ≥ 1e-06 seconds for DMG format
- ≥ 2e-06 seconds for VHT, HT-mixed, non-HT formats

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 [1], Section L.1.5.2 and applies for S1G, VHT, HT, and Non-HT OFDM formats. For the DMG format, specifying `ScramblerInitialization` with `wlanWaveformGenerator` overrides the value specified by the `wlanDMGConfig` configuration object. For more information, see "Scrambler Initialization" on page 1-494.

- 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.

Note `ScramblerInitialization` is not valid for non-HT DSSS.

Example: `[3 56 120]`

Data Types: `double` | `int8`

`WindowTransitionTime` — Duration of the window transition
nonnegative scalar

Duration of the window transition applied to each OFDM symbol, specified in seconds as a nonnegative scalar. No windowing is applied if `WindowTransitionTime` = 0. The default and maximum value permitted is shown for the various formats, type of guard interval, and channel bandwidth.

	Maximum Permitted `WindowTransitionTime` (seconds)						
	DMG	S1G	VHT	HT-mixed	non-HT		
	2640 MHz	1, 2, 4, 8, or 16 MHz	20, 40, 80, or 160 MHz	20 or 40 MHz	20 MHz	10 MHz	5 MHz
Default	6.0606e-09 (= 16/2640e6)	1.0e-07	1.0e-07	1.0e-07	1.0e-07	1.0e-07	1.0e-07

	Maximum Permitted `WindowTransitionTime` (seconds)						
	DMG	S1G	VHT	HT-mixed	non-HT		
	2640 MHz	1, 2, 4, 8, or 16 MHz	20, 40, 80, or 160 MHz	20 or 40 MHz	20 MHz	10 MHz	5 MHz
Maximum	9.6969e-08 (= 256/2640e6)	–	–	–	–	–	–
Maximum Permitted for Long Guard Interval	–	1.6e-05	1.6e-06	1.6e-06	1.6e-06	3.2e-06	6.4e-06
Maximum Permitted for Short Guard Interval	–	8.0e-06	8.0e-07	8.0e-07	–	–	–

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 1-482. Each packet can contain different information bits. Waveform packet windowing is enabled by setting `WindowTransitionTime` > 0. Windowing is enabled by default.

For more information, see "Waveform Sampling Rate" on page 1-487, "OFDM Symbol Windowing" on page 1-489, and "Waveform Looping" on page 1-491.

Data Types: `double`
Complex Number Support: Yes

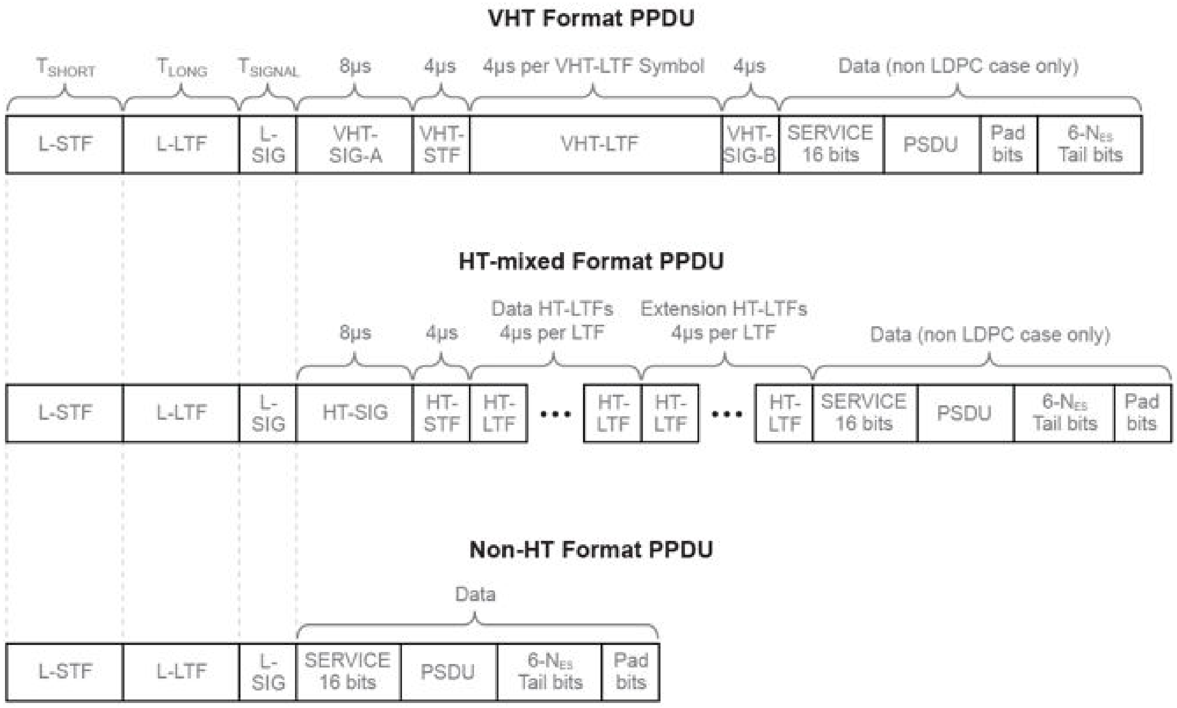
Definitions

IEEE 802.11 PPDU Format

IEEE 802.11[31][32] PPDU formats defined for transmission include VHT, HT, non-HT, S1G, and DMG. For all formats, the PPDU field structure includes preamble and data portions. The DMG format PPDU contains a header field and optional training fields.

VHT, HT-Mixed, and Non-HT Format PPDU Field Structures

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



Subcarrier duration varies with channel bandwidth for the legacy preamble fields.

Channel Bandwidth (MHz)	Preamble Field Duration		
	T_{SHORT}: L-STF Duration	T_{LONG}: L-LTF Duration	T_{SIGNAL}: Duration of the L-SIG Symbol
20, 40, 80, 160	8 µs	8 µs	4 µs
10	16 µs	16 µs	8 µs
5	32 µs	32 µs	16 µs

S1G Format PPDU Field Structure

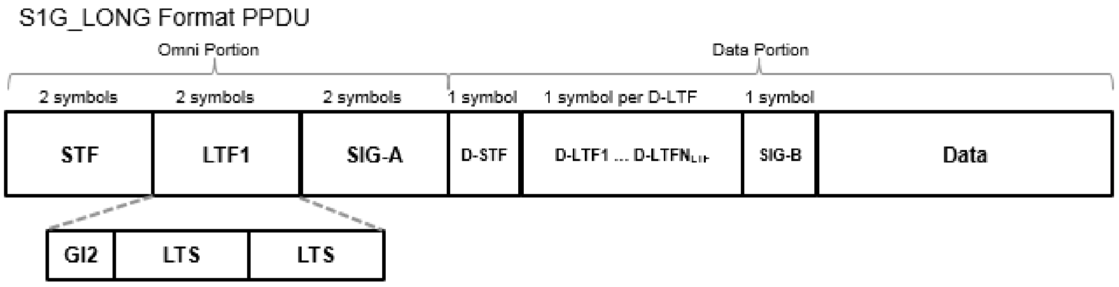
In S1G, there are three transmission modes:

- ≥2-MHz long preamble mode
- ≥2-MHz short preamble mode

- 1-MHz mode

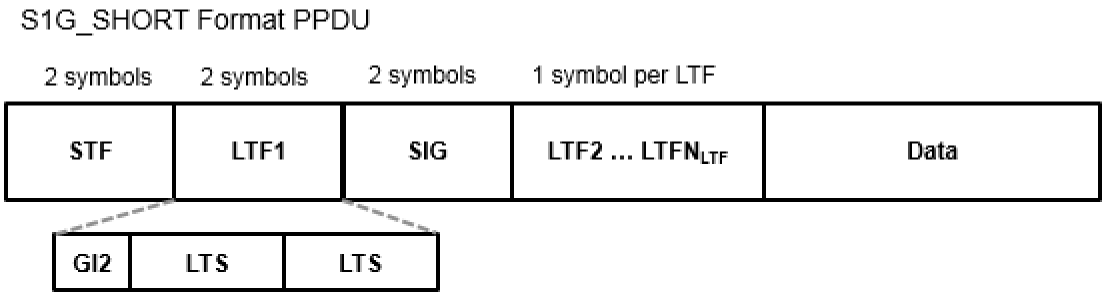
Each transmission mode has a specific PPDU preamble structure:

- An S1G ≥2-MHz long preamble mode PPDU supports single-user and multi-user transmissions. The long preamble PPDU consists of two portions; the omni-directional portion and the beam-changeable portion.



- The omni-directional portion is transmitted to all users without beamforming. It consists of three fields:
  - The short training field (STF) is used for coarse synchronization.
  - The long training field (LTF1) is used for fine synchronization and initial channel estimation.
  - The signal A field (SIG-A) is decoded by the receiver to determine transmission parameters relevant to all users.
- The data portion can be beamformed to each user. It consists of four fields:
  - The beamformed short training field (D-STF) is used by the receiver for automatic gain control.
  - The beamformed long training fields (D-LTF-N) are used for MIMO channel estimation.
  - The signal B field (SIG-B) in a multi-user transmission, signals the MCS for each user. In a single-user transmission, the MCS is signaled in the SIG-A field of the omni-directional portion of the preamble. Therefore, in a single-user transmission the SIG-B symbol transmitted is an exact repetition of the first D-LTF. This repetition allows for improved channel estimation.

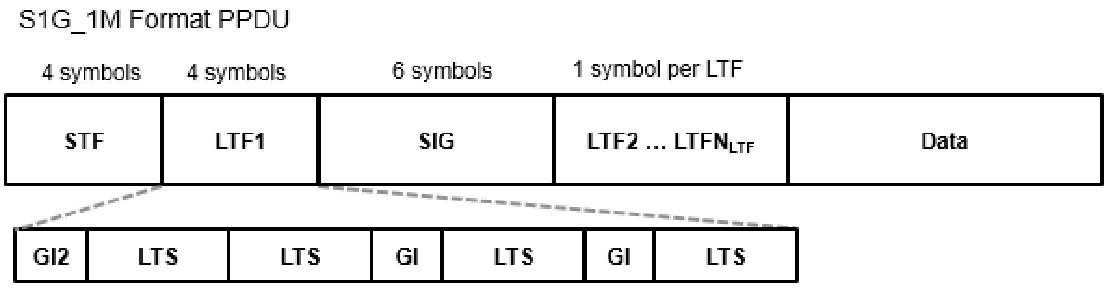
  - The data field is variable in length. It carries the user data payload.
- An S1G ≥2-MHz short preamble mode PPDU supports single-user transmissions. All fields in the PPDU can be beamformed.



The PPDU consists of these five fields:

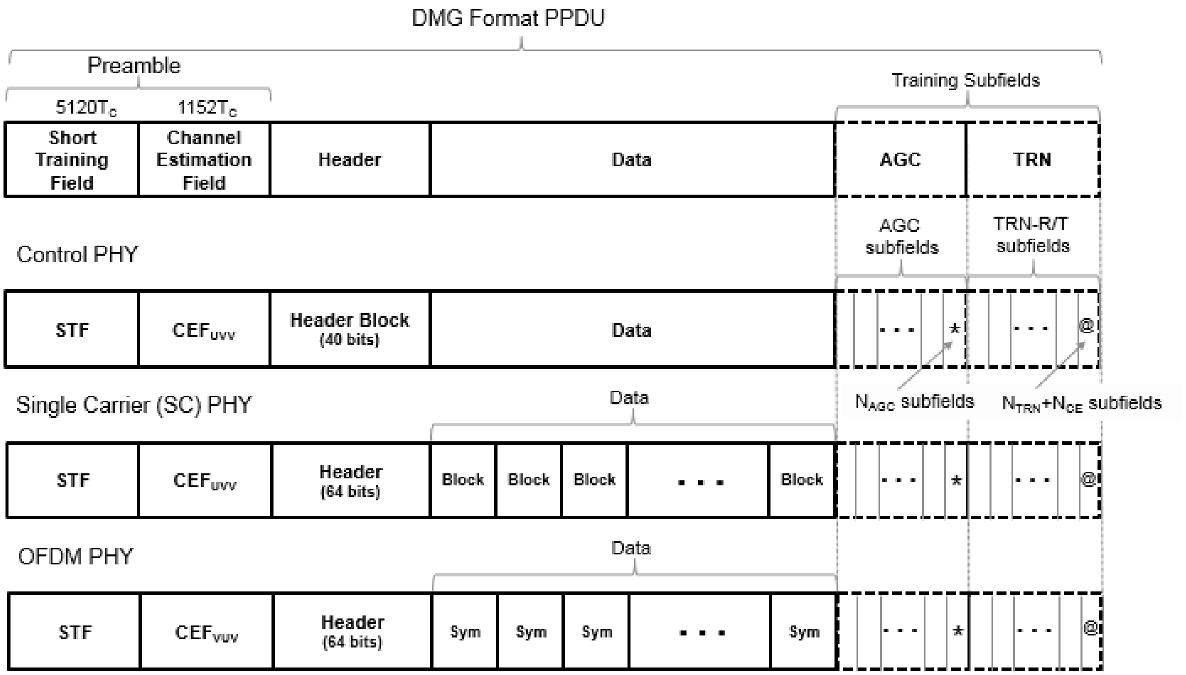
- The short training field (STF) is used for coarse synchronization.
- The first long training field (LTF1) is used for fine synchronization and initial channel estimation.
- The signaling field (SIG) is decoded by the receiver to determine transmission parameters.
- The subsequent long training fields (LTF2-N) are used for MIMO channel estimation. N_{SYMBOLS} = 1 per subsequent LTF
- The data field is variable in length. It carries the user data payload.

- An S1G 1-MHz mode PPDU supports single-user transmissions. It is composed of the same five fields as the S1G ≥2-MHz short preamble mode PPDU and all fields can be beamformed. An S1G 1-MHz mode PPDU has longer STF, LTF1, and SIG fields so this narrower bandwidth mode can achieve sensitivity that is similar to the S1G ≥2-MHz short preamble mode transmissions.



DMG Format PPDU Field Structure

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



The single-carrier chip timing, $T_C = 1/F_C = 0.57$ ns. For more information, see "Waveform Sampling Rate" on page 1-487.

The supported DMG format PPDU field structures each contain these fields:

- The preamble contains a short training field (STF) and channel estimation field (CEF). The preamble is used for packet detection, AGC, frequency offset estimation, synchronization, indication of modulation type (Control, SC, or OFDM), and channel estimation. The format of the preamble is common to the Control, SC, and OFDM PHY packets.
  - The STF is composed of Golay *Ga* sequences as specified in 802.11ad-2012 [2], Section 21.3.6.2.
  - The CEF is composed of Golay *Gu* and *Gv* sequences as specified in 802.11ad-2012 [2], Section 21.3.6.3.
    - When the header and data fields of the packet are modulated using a single carrier (control PHY and SC PHY), the Golay sequencing for the CEF waveform is shown in 802.11ad-2012 [2], Figure 21-5.
    - When the header and data fields of the packet are modulated using OFDM (OFDM PHY), the Golay sequencing for the CEF waveform is shown in 802.11ad-2012 [2], Figure 21-6.
- The header field is decoded by the receiver to determine transmission parameters.
- The data field is variable in length. It carries the user data payload.
- The training fields (AGC and TRN-R/T subfields) are optional. They can be included to refine beamforming.

IEEE 802.11ad-2012 [2] specifies the common aspects of the DMG PPDU packet structure in Section 21.3. The PHY modulation-specific aspects of the packet structure are specified in these sections:

- The DMG control PHY packet structure is specified in Section 21.4.
- The DMG OFDM PHY packet structure is specified in Section 21.5.
- The DMG SC PHY packet structure is specified in Section 21.6.

Waveform Sampling Rate

At the output of `wlanWaveformGenerator`, the generated waveform has a sampling rate equal to the channel bandwidth.

For all VHT, HT, and non-HT format OFDM modulation, the channel bandwidth is configured via the `ChannelBandwidth` property of the format configuration object.

For the DMG format modulation schemes, the channel bandwidth is always 2640 MHz and the channel spacing is always 2160 MHz These values are specified in IEEE 802.11ad-2012 [2], Section 21.3.4 and Annex E-1, respectively.

For the non-HT format DSSS modulation scheme, the chipping rate is always 11 MHz, as specified in IEEE 802.11-2012[1], Section 17.1.1.

This table indicates the waveform sampling rates associated with standard channel spacing for each configuration format prior to filtering.

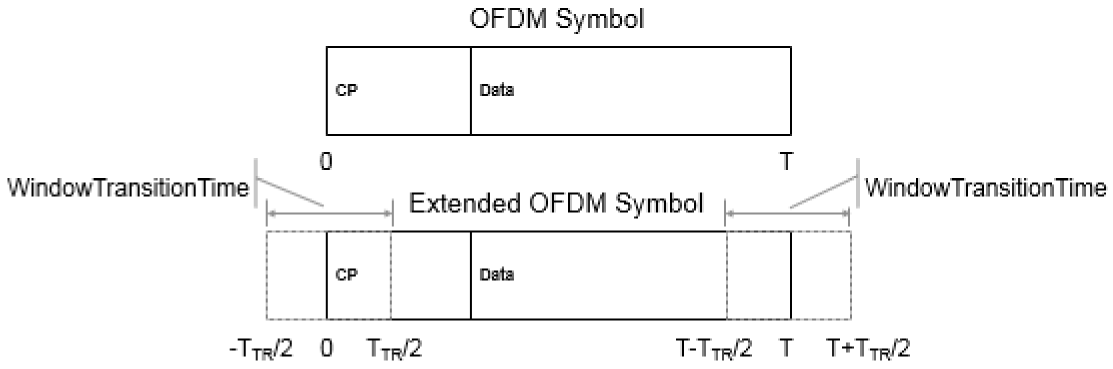
Configuration Object	**`Modulation`**	**`ChannelBandwidth`**	Channel Spacing (MHz)	Sampling Rate (MHz) (F_S, F_C)
`wlanDMGConfig`	Control PHY	For DMG, the channel bandwidth is fixed at 2640 MHz.	2160	$F_C = ⅔ F_S = 1760$
	SC			
	OFDM			$F_S = 2640$
`wlanS1GConfig`	OFDM	`'CBW1'`	1	$F_S = 1$
		`'CBW2'`	2	$F_S = 2$
		`'CBW4'`	4	$F_S = 4$
		`'CBW8'`	8	$F_S = 8$
		`'CBW16'`	16	$F_S = 16$
`wlanVHTConfig`	OFDM	`'CBW20'`	20	$F_S = 20$
		`'CBW40'`	40	$F_S = 40$
		`'CBW80'`	80	$F_S = 80$
		`'CBW160'`	160	$F_S = 160$
`wlanHTConfig`	OFDM	`'CBW20'`	20	$F_S = 20$
		`'CBW40'`	40	$F_S = 40$
`wlanNonHTConfig`	DSSS/CCK	Not applicable	11	$F_C = 11$
	OFDM	`'CBW5'`	5	$F_S = 5$
		`'CBW10'`	10	$F_S = 10$

Configuration Object	`Modulation`	`ChannelBandwidth`	Channel Spacing (MHz)	Sampling Rate (MHz) (F_S, F_C)
		`'CBW20'`	20	$F_S = 20$
F_S is the OFDM sampling rate. F_C is the chip rate for single carrier, control PHY, and DSSS/CCK modulations.				

OFDM Symbol Windowing

OFDM naturally lends itself to processing with Fourier transforms. A negative side effect of using an IFFT to process OFDM symbols is the resulting symbol-edge discontinuities. These discontinuities cause out-of-band emissions in the transition region between consecutive OFDM symbols. To smooth the discontinuity between symbols and reduce the intersymbol out-of-band emissions, you can use the `wlanWaveformGenerator` function to apply OFDM symbol windowing. To apply windowing, set `WindowTransitionTime` to greater than zero.

When windowing is applied, transition regions are added to the leading and trailing edge of the OFDM symbol by the `wlanWaveformGenerator`. Windowing extends the length of the OFDM symbol by `WindowTransitionTime` (T_{TR}).

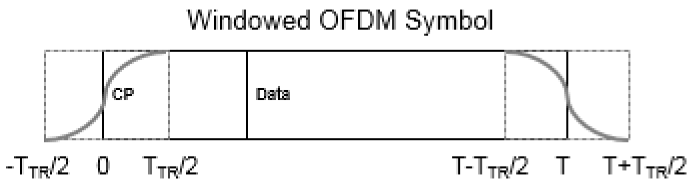


The extended waveform is windowed by pointwise multiplication in the time domain, using the windowing function specified in IEEE Std 802.11-2012 [1], Section 18.3.2.5:

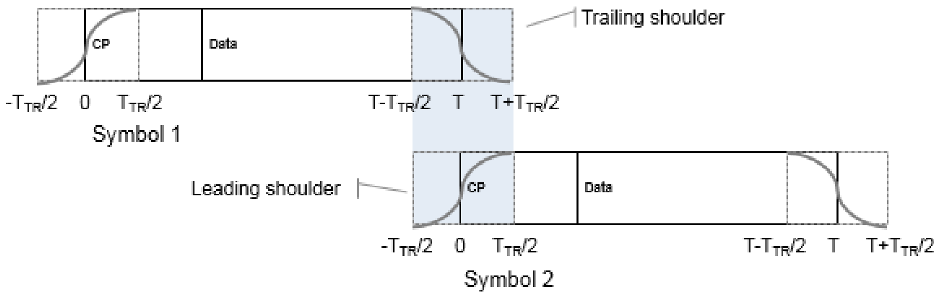
$$w_{\mathrm{T}}(t) = \begin{cases} \sin^2\left(\frac{\pi}{2}\left(0.5 + \frac{t}{T_{\mathrm{TR}}}\right)\right) & \left(-{T_{\mathrm{TR}}}/{2} < t < {T_{\mathrm{TR}}}/{2}\right) \\ 1 & \left({T_{\mathrm{TR}}}/{2} < t < T - {T_{\mathrm{TR}}}/{2}\right) \\ \sin^2\left(\frac{\pi}{2}\left(0.5 - \frac{t-T}{T_{\mathrm{TR}}}\right)\right) & \left(T - {T_{\mathrm{TR}}}/{2} < t < T + {T_{\mathrm{TR}}}/{2}\right) \end{cases}$$

The windowing function applies over the leading and trailing portion of the OFDM symbol:

- $-T_{\mathrm{TR}}/2$ to $T_{\mathrm{TR}}/2$
- $-T - T_{\mathrm{TR}}/2$ to $T + T_{\mathrm{TR}}/2$



After windowing is applied to each symbol, pointwise addition is used to combine the overlapped regions between consecutive OFDM symbols. Specifically, the trailing shoulder samples at the end of OFDM symbol 1 ($T - T_{\mathrm{TR}}/2$ to $T + T_{\mathrm{TR}}/2$) are added to the leading shoulder samples at the beginning of OFDM symbol 2 ($-T_{\mathrm{TR}}/2$ to $T_{\mathrm{TR}}/2$).

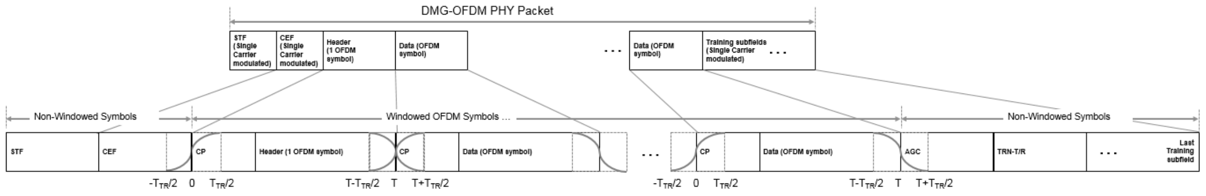


Smoothing the overlap between consecutive OFDM symbols in this manner reduces the out-of-band emissions. `wlanWaveformGenerator` applies OFDM symbol windowing between:

- Each OFDM symbol within a packet
- Consecutive packets within the waveform, considering the `IdleTime` between packets
- The last and the first packet of the generated waveform

Windowing DMG Format Packets

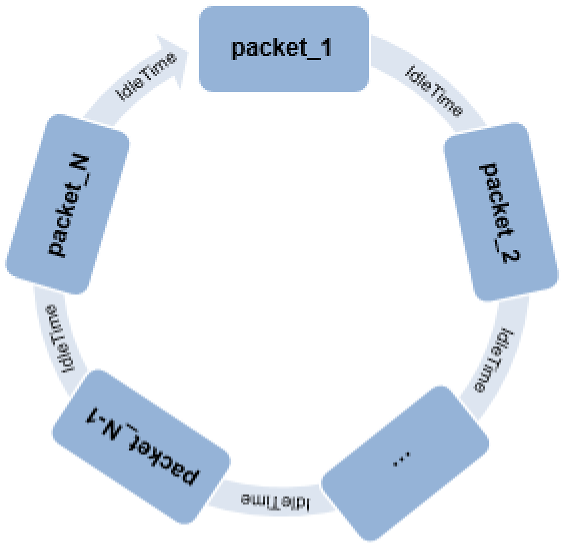
For DMG format, windowing is only applicable to packets transmitted using the OFDM PHY and is applied only to the OFDM modulated symbols. For OFDM PHY, only the header and data symbols are OFDM modulated. The preamble (STF and CEF) and the training fields are single carrier modulated and are not windowed. Similar to the out of band emissions experienced by consecutive OFDM symbols, as shown here the CEF and the first training subfield are subject to a nominal amount out of band emissions from the adjacent windowed OFDM symbol.



For more information on how `wlanWaveformGenerator` handles windowing for the consecutive packet `IdleTime` and for the last waveform packet, see “Waveform Looping” on page 1-491.

Waveform Looping

To produce a continuous input stream, you can have your code loop on a waveform from the last packet back to the first packet.

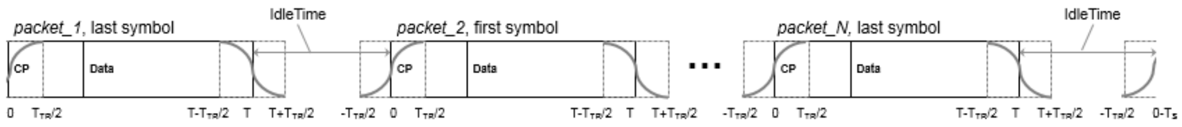


Applying windowing to the last and first OFDM symbols of the generated waveform smooths the transition between the last and first packet of the waveform. When `WindowTransitionTime` is greater than zero, `wlanWaveformGenerator` applies "OFDM Symbol Windowing" on page 1-489.

When looping a waveform, the last symbol of *packet_N* is followed by the first OFDM symbol of *packet_1*. If the waveform has only one packet, the waveform loops from the last OFDM symbol of the packet to the first OFDM symbol of the same packet.

When windowing is applied to the last OFDM symbol of a packet and the first OFDM of the next packet, the idle time between the packets factors into the windowing applied. Specify the idle time using the `IdleTime` property of `wlanWaveformGenerator`.

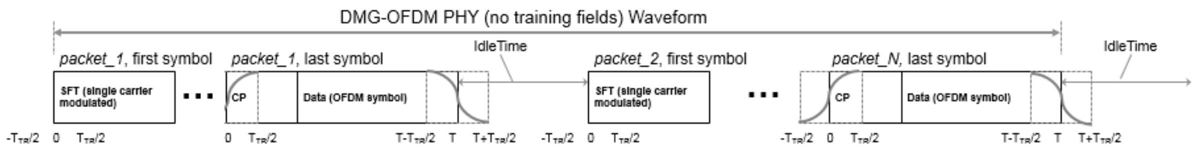
- If `IdleTime` is zero, "OFDM Symbol Windowing" on page 1-489 is applied as it would be for consecutive OFDM symbols within a packet.
- If the `IdleTime` is nonzero, the extended windowed portion of the first OFDM symbol in *packet_1* (from $-T_{TR}/2$ to $0-T_S$), is included at the end of the waveform. This extended windowed portion is applied for looping when computing the "OFDM Symbol Windowing" on page 1-489 between the last OFDM symbol of *packet_N* and the first OFDM symbol of *packet_1*. T_S is the sample time.



Looping DMG Format Waveforms

For DMG format waveforms there are three looping scenarios,

- The looping behavior for a waveform composed of DMG OFDM-PHY packets with no training subfields is similar to the general case outlined in “Waveform Looping” on page 1-491 but the first symbol of the waveform (and each packet) is not windowed.



  - If `IdleTime` is zero for the waveform, the windowed portion (from T to $T + T_{\mathrm{TR}}/2$) of the last data symbols is added to the start of the STF field.
  - If `IdleTime` is non-zero for the waveform, the `IdleTime` is appended at the end of the windowed portion (after $T + T_{\mathrm{TR}}/2$) of the last OFDM symbol.
- When a waveform composed of DMG OFDM-PHY packets includes training subfields, no windowing is applied to the single carrier modulated symbols the end of the waveform. The last sample of the last training subfield is followed by the first STF sample of the first packet in the waveform.

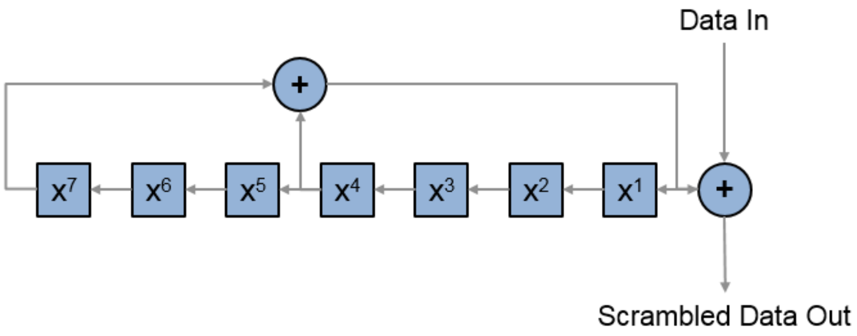
  - If `IdleTime` is zero for the waveform, there is no overlap.
  - If `IdleTime` is nonzero for the waveform, the value specifies the delay between the last sample of *packet_N* and the first sample of in *packet_1*.
- When a waveform is composed of DMG-SC or DMG-Control PHY packets, the end of the waveform is single carrier modulated, so no windowing is applied to the last waveform symbol. The last sample of the last training subfield is followed by the first STF sample of the first packet in the waveform.

  - If `IdleTime` is zero for the waveform, there is no overlap.
  - If `IdleTime` is nonzero for the waveform, the value specifies the delay between the last sample of *packet_N* and the first sample of in *packet_1*.

Note The same looping behavior applies for a waveform composed of DMG OFDM-PHY packets with training subfields, DMG-SC PHY packets, or DMG-Control PHY packets.

Scrambler Initialization

The scrambler initialization used on the transmission data follows the process described in IEEE Std 802.11-2012, Section 18.3.5.5 and IEEE Std 802.11ad-2012, Section 21.3.9. The header and data fields that follow the scrambler initialization field (including data padding bits) are scrambled by XORing each bit with a length-127 periodic sequence generated by the polynomial $S(x) = x^7+x^4+1$. The octets of the PSDU (Physical Layer Service Data Unit) are placed into a bit stream, and within each octet, bit 0 (LSB) is first and bit 7 (MSB) is last. The generation of the sequence and the XOR operation are shown in this figure:



Conversion from integer to bits uses left-MSB orientation. For the initialization of the scrambler with decimal `1`, the bits are mapped to the elements shown.

Element	X^7	X^6	X^5	X^4	X^3	X^2	X^1
Bit Value	0	0	0	0	0	0	1

To generate the bit stream equivalent to a decimal, use `de2bi`. For example, for decimal `1`:

```
de2bi(1,7,'left-msb')
ans =

     0     0     0     0     0     0     1
```

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 802.11ad™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Use in a MATLAB Function block is not supported.

See Also

`wlanDMGConfig` | `wlanHTConfig` | `wlanNonHTConfig` | `wlanS1GConfig` | `wlanVHTConfig`

Topics

“Packet Size and Duration Dependencies”

Introduced in R2015b

Classes — Alphabetical List

wlanDMGConfig Properties

Define parameter values for DMG format packet

Description

The `wlanDMGConfig` object specifies the transmission properties for the IEEE 802.11 directional multi-gigabit (DMG) format physical layer (PHY) packet.

After you create an object, use dot notation to change or access the object parameters. For example:

Create a `wlanDMGConfig` object. Then modify the default setting for the `MCS` property.

```
cfgDMG = wlanDMGConfig;
cfgDMG.MCS = 9;
```

Properties

DMG Format Configuration

`MCS` — Modulation and coding scheme index

`0` (default) | integer from 0 to 24

Modulation and coding scheme index, specified as an integer from 0 to 24. The `MCS` index indicates the modulation and coding scheme used in transmitting the current packet.

- Modulation and coding scheme for control PHY

MCS Index	Modulation	Coding Rate	Comment
0	DBPSK	1/2	Code rate and data rate might be lower due to codeword shortening.

- Modulation and coding schemes for single-carrier modulation

MCS Index	Modulation	Coding Rate	N_{CBPS}	Repetition
1	π/2 BPSK	1/2	1	2
2		1/2		1
3		5/8		
4		3/4		
5		13/16		
6	π/2 QPSK	1/2	2	
7		5/8		
8		3/4		
9		13/16		
10	π/2 16QAM	1/2	4	
11		5/8		
12		3/4		
N_{CBPS} is the number of coded bits per symbol.				

- Modulation and coding schemes for OFDM modulation

MCS Index	Modulation	Coding Rate	N_{BPSC}	N_{CBPS}	N_{DBPS}
13	SQPSK	1/2	1	336	168
14		5/8			210
15	QPSK	1/2	2	672	336
16		5/8			420
17		3/4			504
18	16QAM	1/2	4	1344	672
19		5/8			840
20		3/4			1008
21		13/16			1092
22	64QAM	5/8	6	2016	1260
23		3/4			1512
24		13/16			1638

MCS Index	Modulation	Coding Rate	N_{BPSC}	N_{CBPS}	N_{DBPS}
N_{BPSC} is the number of coded bits per single carrier.					
N_{CBPS} is the number of coded bits per symbol.					
N_{DBPS} is the number of data bits per symbol.					

Data Types: `double`

`TrainingLength` — Number of training fields
`0` (default) | integer from 0 to 64

Number of training fields, specified as an integer from 0 to 64. `TrainingLength` must be a multiple of four.

Data Types: `double`

`PacketType` — Packet training field type
`'TRN-R'` (default) | `'TRN-T'`

Packet training field type, specified as `'TRN-R'` or `'TRN-T'`. This property applies when `TrainingLength` > `0`.

`'TRN-R'` indicates that the packet includes or requests receive-training subfields and `'TRN-T'` indicates that the packet includes transmit-training subfields.

Data Types: `char` | `string`

`BeamTrackingRequest` — Request beam tracking
`false` (default) | `true`

Request beam tracking, specified as a logical. Setting `BeamTrackingRequest` to `true` indicates that beam tracking is requested. This property applies when `TrainingLength` > `0`.

Data Types: `logical`

`TonePairingType` — Tone pairing type
`'Static'` (default) | `'Dynamic'`

Tone pairing type, specified as `'Static'` or `'Dynamic'`. This property applies when `MCS` is from 13 to 17. Specifically, `TonePairingType` applies when using OFDM and either SQPSK or QPSK modulation.

Data Types: `char` | `string`

`DTPGroupPairIndex` — DTP group pair index
42-by-1 integer vector

DTP group pair index, specified as a 42-by-1 integer vector for each pair. Element values must be from 0 to 41, with no duplicates. This property applies when `MCS` is from 13 to 17 and when `TonePairingType` is `'Dynamic'`.

Data Types: `double`

`DTPIndicator` — DTP update indicator
`false` (default) | `true`

DTP update indicator, specified as a logical. Toggle `DTPIndicator` between packets to indicate that the dynamic tone pair mapping has been updated. This property applies when `MCS` is from 13 to 17 and when `TonePairingType` is `'Dynamic'`.

Data Types: `logical`

`PSDULength` — Number of bytes carried in the user payload
`1000` (default) | integer from 1 to 262,143

Number of bytes carried in the user payload, specified as an integer from 1 to 262,143.

Data Types: `double`

`ScramblerInitialization` — Initial scrambler state
`2` (default) | integer from 1 to 127

Initial scrambler state of the data scrambler for each packet generated, specified as an integer from 1 to 127. When `MCS` is `0`, the initial scrambler state is limited to values from 1 to 15. The default value of 2 is the example state given in IEEE Std 802.11-2012, Amendment 3, Section L.5.2.

Data Types: `double` | `int8`

`AggregatedMPDU` — MPDU aggregation indicator
`false` (default) | `true`

MPDU aggregation indicator, specified as a logical. Setting `AggregatedMPDU` to `true` indicates that the current packet uses A-MPDU aggregation.

Data Types: `logical`

`LastRSSI` — Received power level of the last packet
`0` (default) | integer from 0 to 15

Received power level of the last packet, specified as an integer from 0 to 15.

When transmitting a response frame immediately following a short interframe space (SIFS) period, a DMG STA sets the `LastRSSI` as specified in IEEE 802.11ad-2012, Section 9.3.2.3.3, to map to the *TXVECTOR* parameter *LAST_RSSI* of the response frame to the power that was measured on the received packet, as reported in the RCPI field of the frame that elicited the response frame. The encoding of the value for *TXVECTOR* is as follows:

- Power values equal to or above –42 dBm are represented as the value 15.
- Power values between –68 dBm and –42 dBm are represented as round((power – (–71 dBm))/2).
- Power values less than or equal to –68 dBm are represented as the value of 1.
- For all other cases, the DMG STA shall set the TXVECTOR parameter LAST_RSSI of the transmitted frame to 0.

The *LAST_RSSI* parameter in *RXVECTOR* maps to `LastRSSI` and indicates the value of the *LAST_RSSI* field from the PCLP header of the received packet. The encoding of the value for *RXVECTOR* is as follows:

- A value of 15 represents power greater than or equal to –42 dBm.
- Values from 2 to 14 represent power levels (–71+*`value`*×2) dBm.
- A value of 1 represents power less than or equal to –68 dBm.
- A value of 0 indicates that the previous packet was not received during the SIFS period before the current transmission.

For more information, see IEEE 802.11ad-2012, Section 21.2.

Data Types: `double`

`Turnaround` — Turnaround indication
`false` (default) | `true`

Turnaround indication, specified as a logical. Setting `Turnaround` to `true` indicates that the STA is required to listen for an incoming PPDU immediately following the

transmission of the PPDU. For more information, see IEEE 802.11ad-2012, Section 9.3.2.3.3.

Data Types: `logical`

References

[1] IEEE Std 802.11ad™-2012 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications — Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band.

See Also

`wlanDMGConfig` | `wlanWaveformGenerator`

Introduced in R2017a

wlanGeneratorConfig Properties

Define parameter values for waveform generation

Description

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;
```

Note To override default waveform generator configuration values, use the `wlanWaveformGenerator(bits,cfgFormat,Name1,Value1,...,NameN,ValueN)` syntax.

Use of `wlanGeneratorConfig` is not recommended. Therefore, use of the `wlanWaveformGenerator(bits,cfgFormat,cfgWaveGen)` syntax is discouraged as well.

Properties

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. The default value is 0. If `IdleTime` is not set to the default value, it must be:

- ≥ 1e-06 seconds for DMG format
- ≥ 2e-06 seconds for VHT, HT-mixed, non-HT formats

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`

`WindowTransitionTime` — Duration of the window transition
nonnegative scalar

Duration of the window transition applied to each OFDM symbol, specified in seconds as a nonnegative scalar. No windowing is applied if `WindowTransitionTime` = 0. The default and maximum value permitted is shown for the various formats, type of guard interval, and channel bandwidth.

	Maximum Permitted `WindowTransitionTime` (seconds)						
	DMG	S1G	VHT	HT-mixed	non-HT		
	2640 MHz	1, 2, 4, 8, or 16 MHz	20, 40, 80, or 160 MHz	20 or 40 MHz	20 MHz	10 MHz	5 MHz
Default	6.0606e-09 (= 16/2640e6)	1.0e-07	1.0e-07	1.0e-07	1.0e-07	1.0e-07	1.0e-07
Maximum	9.6969e-08 (= 256/2640e6)	–	–	–	–	–	–
Maximum Permitted for Long Guard Interval	–	1.6e-05	1.6e-06	1.6e-06	1.6e-06	3.2e-06	6.4e-06
Maximum Permitted for Short Guard Interval	–	8.0e-06	8.0e-07	8.0e-07	–	–	–

Data Types: `double`

References

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

See Also

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

Introduced in R2015b

wlanHTConfig Properties

Define parameter values for HT format packet

Description

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;
```

Properties

HT Format Configuration

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

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

Data Types: `char` | `string`

`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` | `string`

`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` | `string`

`ChannelCoding` — Type of forward error correction coding
`'BCC'` (default) | `'LDPC'`

Type of forward error correction coding for the data field, specified as `'BCC'` (default) or `'LDPC'`. `'BCC'` indicates binary convolutional coding, and `'LDPC'` indicates low density parity check coding.

Data Types: `char` | `cell` | `string`

`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` — Recommend smoothing for channel estimation
`true` (default) | `false`

Recommend smoothing for channel estimation, specified as a logical.

- If the frequency profile is nonvarying across the channel , the receiver sets this property to `true`. In this case, frequency-domain smoothing is recommended as part of channel estimation.

- If the frequency profile varies across the channel, the receiver sets this property to `false`. In this case, frequency-domain smoothing is not recommended as part of channel estimation.

Data Types: `logical`

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

Introduced in R2015b

wlanNonHTConfig Properties

Define parameter values for non-HT format packet

Description

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;
```

Properties

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` | `string`

`ChannelBandwidth` — Channel bandwidth
`'CBW20'` (default) | `'CBW10'` | `'CBW5'`

Channel bandwidth in MHz for OFDM, specified as `'CBW20'`, `'CBW10'`, or `'CBW5'`. The default value of `'CBW20'` sets the channel bandwidth to 20 MHz.

When channel bandwidth is 5 MHz or 10 MHz, only one transmit antenna is permitted and `NumTransmitAntennas` is not applicable.

Data Types: `char` | `string`

`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	Coded bits per subcarrier (N_{BPSC})	Coded bits per OFDM symbol (N_{CBPS})	Data bits per OFDM symbol (N_{DBPS})	Data Rate (Mbps)		
						20 MHz channel bandwidth	10 MHz channel bandwidth	5 MHz channel bandwidth
0	BPSK	1/2	1	48	24	6	3	1.5
1	BPSK	3/4	1	48	36	9	4.5	2.25
2	QPSK	1/2	2	96	48	12	6	3
3	QPSK	3/4	2	96	72	18	9	4.5
4	16QAM	1/2	4	192	96	24	12	6
5	16QAM	3/4	4	192	144	36	18	9
6	64QAM	2/3	6	288	192	48	24	12
7	64QAM	3/4	6	288	216	54	27	13.5

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` | `string`

`Preamble` **— DSSS modulation preamble type**
`'Long'` (default) | `'Short'`

DSSS modulation preamble type, 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` | `string`

`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 for OFDM, specified as a scalar integer from 1 to 8.

When channel bandwidth is 5 MHz or 10 MHz, `NumTransmitAntennas` is not applicable because only one transmit antenna is permitted.

Data Types: `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.

See Also

`wlanLLTF` | `wlanLLTFChannelEstimate` | `wlanLLTFDemodulate` | `wlanLSIG` | `wlanLSIGRecover` | `wlanLSTF` | `wlanNonHTConfig` | `wlanNonHTData` | `wlanNonHTDataRecover` | `wlanWaveformGenerator`

Introduced in R2015b

wlanS1GConfig Properties

Define parameter values for S1G format packet

Description

The `wlanS1GConfig` object specifies the transmission properties for the IEEE 802.11 sub 1 GHz (S1G) format physical layer (PHY) packet.

After you create an object, use dot notation to change or access the object parameters. For example:

Create a `wlanS1GConfig` object. Then modify the default setting for the `ChannelBandwidth` property.

```
cfgS1G = wlanS1GConfig;
cfgS1G.ChannelBandwidth = 'CBW2';
```

Properties

S1G Format Configuration

`ChannelBandwidth` — Channel bandwidth
`'CBW2'` (default) | `'CBW1'` | `'CBW4'` | `'CBW8'` | `'CBW16'`

Channel bandwidth, specified as `'CBW1'`, `'CBW2'`, `'CBW4'`, `'CBW8'`, or `'CBW16'`. If the transmission has multiple users, the same channel bandwidth is applied to all users.

Example: `'CBW16'` sets the channel bandwidth to 16 MHz.

Data Types: `char` | `string`

`Preamble` — Preamble type
`'Short'` (default) | `'Long'`

Preamble type, specified as `'Short'` or `'Long'`. This property applies only when `ChannelBandwidth` is not `'CBW1'`.

Data Types: `char` | `string`

`NumUsers` — Number of users

1 (default) | 2 | 3 | 4

Number of users, specified as 1, 2, 3, or 4. (N_{Users})

Data Types: `double`

`UserPositions` — Position of users

[0 1] (default) | row vector of integers from 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` > 1.

Example: `[0 2 3]` indicates positions for three users, where the first user occupies position 0, the second user occupies position 2, and the third user occupies position 3.

Data Types: `double`

`NumTransmitAntennas` — Number of transmit antennas

1 (default) | integer from 1 to 4

Number of transmit antennas, specified as a scalar integer from 1 to 4.

Data Types: `double`

`NumSpaceTimeStreams` — Number of space-time streams

1 (default) | integer from 1 to 4 | 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. (N_{sts})

- For a single user, the number of space-time streams is an integer scalar from 1 to 4.
- For multiple users, the number of space-time streams is a 1-by-N_{Users} vector of integers from 1 to 4, where $N_{\text{Users}} \leq 4$. The sum total of space-time streams for all users, $N_{\text{sts\_Total}}$, must not exceed four.

Example: `[1 1 2]` indicates number of space-time streams for three users, where the first user gets 1 space-time stream, the second user gets 1 space-time stream, and the third user gets 2 space-time streams. The total number of space-time streams assigned is 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 of 'Direct' applies when NumTransmitAntennas and NumSpaceTimeStreams are equal.

Data Types: char | string

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})
'CBW1'	26	24	2
'CBW2'	56	52	4
'CBW4'	114	108	6
'CBW8'	242	234	8
'CBW16'	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` — Enable beamforming in a long preamble packet
`true` (default) | `false`

Enable beamforming in a long preamble packet, specified as a logical. Beamforming is performed when this setting is `true`. This property applies for a long preamble (`Preamble` = `'Long'`) with `NumUsers` = 1 and `SpatialMapping = 'Custom'`. The `SpatialMappingMatrix` property specifies the beamforming steering matrix.

Data Types: `logical`

`STBC` — Enable space-time block coding
`false` (default) | `true`

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 10 | 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 10.

- For multiple users, MCS is a 1-by-N_{Users} vector of integers or a scalar with values from 0 to 10, where $N_{\text{Users}} \leq 4$.

MCS	Modulation	Coding Rate	Comment
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`	
10	`BPSK`	`1/2`	Applies only for `ChannelBandwidth = 'CBW1'`

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'`. Only binary convolutional coding is supported.

Data Types: `char` | `cell`

`APEPLength` — Number of bytes in the A-MPDU pre-EOF padding

256 (default) | integer from 0 to 65,535 | vector of integers

Number of bytes in the A-MPDU pre-EOF padding, specified as an integer scalar or vector.

- For a single user, `APEPLength` is a scalar integer from 0 to 65,535.

- For multiple users, `APEPLength` is a 1-by-N_{Users} vector of integers or a scalar with values from 0 to 65,535, where $N_{\text{Users}} \leq 4$.
- `APEPLength = 0` for a null data packet (NDP).

`APEPLength` is used internally to determine the number of OFDM symbols in the data field.

Note Only aggregated data transmission is supported.

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, specified as an integer scalar or vector. 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 65,535, where $N_{\text{Users}} \leq 4$.
- When undefined, `PSDULength` is returned as an empty, `[]`. This can happen when the set of property values for the object are in an invalid state.

`PSDULength` is calculated internally based on the `APEPLength` property and other coding-related properties. It is accessible only by direct property call.

Example: `[1031 2065]` is the PSDU length vector for a `wlanS1GConfig` object with two users, where the MCS for the first user is 4 and the MCS for the second user is 8.

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.

Note For S1G, the first OFDM symbol within the data field always has a long guard interval, even when `GuardInterval` is set to `'Short'`.

Data Types: `char` | `string`

`GroupID` — Group identification number
1 (default) | integer from 1 to 62

Group identification number, specified as an integer scalar from 1 to 62. The group identification number is signaled during a multi-user transmission. Therefore this property applies for a long preamble (`Preamble` = `'Long'`) and when `NumUsers` is greater than 1.

Data Types: `double`

`PartialAID` — Abbreviated indication of the PSDU recipient
37 (default) | integer from 0 to 511

Abbreviated indication of the PSDU recipient, specified as an integer scalar from 0 to 511.

- For an uplink transmission, the partial identification number is the last nine bits of the basic service set identifier (BSSID) and must be an integer from 0 to 511.
- 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 and must be an integer from 0 to 63.

For more information, see IEEE Std 802.11ac-2013, Table 22-1.

Data Types: `double`

`UplinkIndication` — Enable uplink indication
`false` (default) | `true`

Enable uplink indication, specified as a logical. Set `UplinkIndication` to `true` for uplink transmission or `false` for downlink transmission. This property applies when `ChannelBandwidth` is not `'CBW1'` and `NumUsers` = 1.

Data Types: `logical`

`Color` — Access point color identifier
0 (default) | integer scalar from 0 to 7

Access point (AP) color identifier, specified as an integer from 0 to 7. An AP includes a `Color` number for the basic service set (BSS). An S1G station (STA) can use the `Color` setting to determine if the transmission is within a BSS it is associated with. An S1G STA can terminate the reception process for transmissions received from a BSS that it is not associated with. This property applies when `ChannelBandwidth` is not `'CBW1'`, `NumUsers` = 1, and `UplinkIndication` = `false`.

Data Types: `double`

`TravelingPilots` — Enable traveling pilots
`false` (default) | `true`

Enable traveling pilots, specified as a logical. Set `TravelingPilots` to `true` for nonconstant pilot locations. Traveling pilots allow a receiver to track a changing channel due to Doppler spread.

Data Types: `logical`

`ResponseIndication` — Response indication type
`'None'` (default) | `'NDP'` | `'Normal'` | `'Long'`

Response indication type, specified as `'None'`, `'NDP'`, `'Normal'`, or `'Long'`. This information is used to indicate the presence and type of frame that will be sent a short interframe space (SIFS) after the current frame transmission. The response indication field is set based on the value of `ResponseIndication` and transmitted in;

- The SIG2 field of the S1G_SHORT preamble
- The SIG-A-2 field of the S1G_LONG preamble
- The SIG field of the S1G_1M preamble

Data Types: `char` | `string`

`RecommendSmoothing` — Recommend smoothing for channel estimation
`true` (default) | `false`

Recommend smoothing for channel estimation, specified as a logical.

- If the frequency profile is nonvarying across the channel , the receiver sets this property to `true`. In this case, frequency-domain smoothing is recommended as part of channel estimation.
- If the frequency profile varies across the channel, the receiver sets this property to `false`. In this case, frequency-domain smoothing is not recommended as part of channel estimation.

Data Types: `logical`

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` | `wlanNonHTConfig` | `wlanS1GConfig` | `wlanVHTConfig` | `wlanWaveformGenerator`

Introduced in R2016b

wlanRecoveryConfig Properties

Define parameter values for data recovery

Description

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;
```

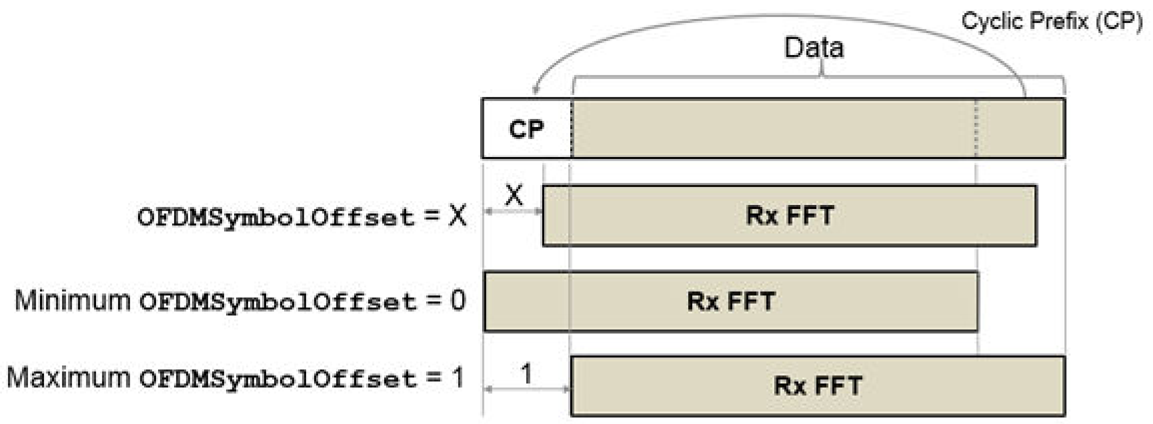
Properties

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` | `string`

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

Pilot phase tracking, specified as `'PreEQ'` or `'None'`.

- `'PreEQ'` — Enables pilot phase tracking, which is performed before any equalization operation.
- `'None'` — Pilot phase tracking does not occur.

Data Types: `char` | `string`

`MaximumLDPCIterationCount` — Maximum number of decoding iterations in LDPC
12 (default) | positive scalar integer

Maximum number of decoding iterations in LDPC, specified as a positive scalar integer. This parameter is applicable when channel coding is set to LDPC. For information on channel coding options, see `wlanVHTConfig` or `wlanHTConfig` for 802.11 format of interest.

Data Types: `double`

`EarlyTermination` — Enable early termination of LDPC decoding
`false` (default) | `true`

Enable early termination of LDPC decoding, specified as a logical. This parameter is applicable when channel coding is set to LDPC.

- When set to `false`, LDPC decoding completes the number of iterations specified by `MaximumLDPCIterationCount`, regardless of parity check status.
- When set to `true`, LDPC decoding terminates when all parity-checks are satisfied.

For information on channel coding options, see `wlanVHTConfig` or `wlanHTConfig` for 802.11 format of interest.

See Also

`wlanHTConfig` | `wlanNonHTConfig` | `wlanRecoveryConfig` | `wlanVHTConfig`

Introduced in R2015b

wlanVHTConfig Properties

Define parameter values for VHT format packet

Description

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';
```

Properties

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` | `string`

`NumUsers` — Number of users
1 (default) | 2 | 3 | 4

Number of users, specified as 1, 2, 3, or 4. (N_{Users})

Data Types: `double`

`UserPositions` — Position of users

[0 1] (default) | row vector of integers from 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` > 1.

Example: `[0 2 3]` indicates positions for three users, where the first user occupies position 0, the second user occupies position 2, and the third user occupies position 3.

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` | `string`

`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` — Enable signaling of a transmission with beamforming

`true` (default) | `false`

Enable 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` — Enable space-time block coding

`false` (default) | `true`

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`

MCS	Modulation	Coding Rate
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) | `'LDPC'`

Type of forward error correction coding for the data field, specified as `'BCC'` (default) or `'LDPC'`. `'BCC'` indicates binary convolutional coding and `'LDPC'` indicates low density parity check coding. Providing a character vector or a single cell character vector defines the channel coding type for a single user or all users in a multiuser transmission. By providing a cell array different channel coding types can be specified per user for a multiuser transmission.

Data Types: `char` | `cell` | `string`

`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.
- When undefined, `PSDULength` is returned as an empty, `[]`. This can happen when the set of property values for the object are in an invalid state.

`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.

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` | `string`

`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`

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

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`.

Note Alternatively, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

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 `'Model-A'`, `'Model-B'`, `'Model-C'`, `'Model-D'`, `'Model-E'`, or `'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 `'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 `'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'`.

`NormalizeChannelOutputs`

Normalize channel outputs

To normalize the channel outputs by the number of receive antennas, set this property to `true` (default).

`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.

`PathGainsOutputPort`

Enable path gain output

To enable computation of path gain output, set this property to `true`. The default value of this property is `false`.

Methods

info	Characteristic information about TGac Channel
reset	Reset states of the `wlanTGacChannel` object
step	Filter signal through 802.11ac multipath fading channel

Common to All System Objects	
`clone`	Create System object with same property values
`getNumInputs`	Expected number of inputs to a System object
`getNumOutputs`	Expected number of outputs of a System object
`isLocked`	Check locked states of a System object (logical)
`release`	Allow System object property value changes

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;
txSig = wlanWaveformGenerator(randi([0 1],1000,1),cfg);
```

Create a TGac SISO channel with path loss and shadowing enabled.

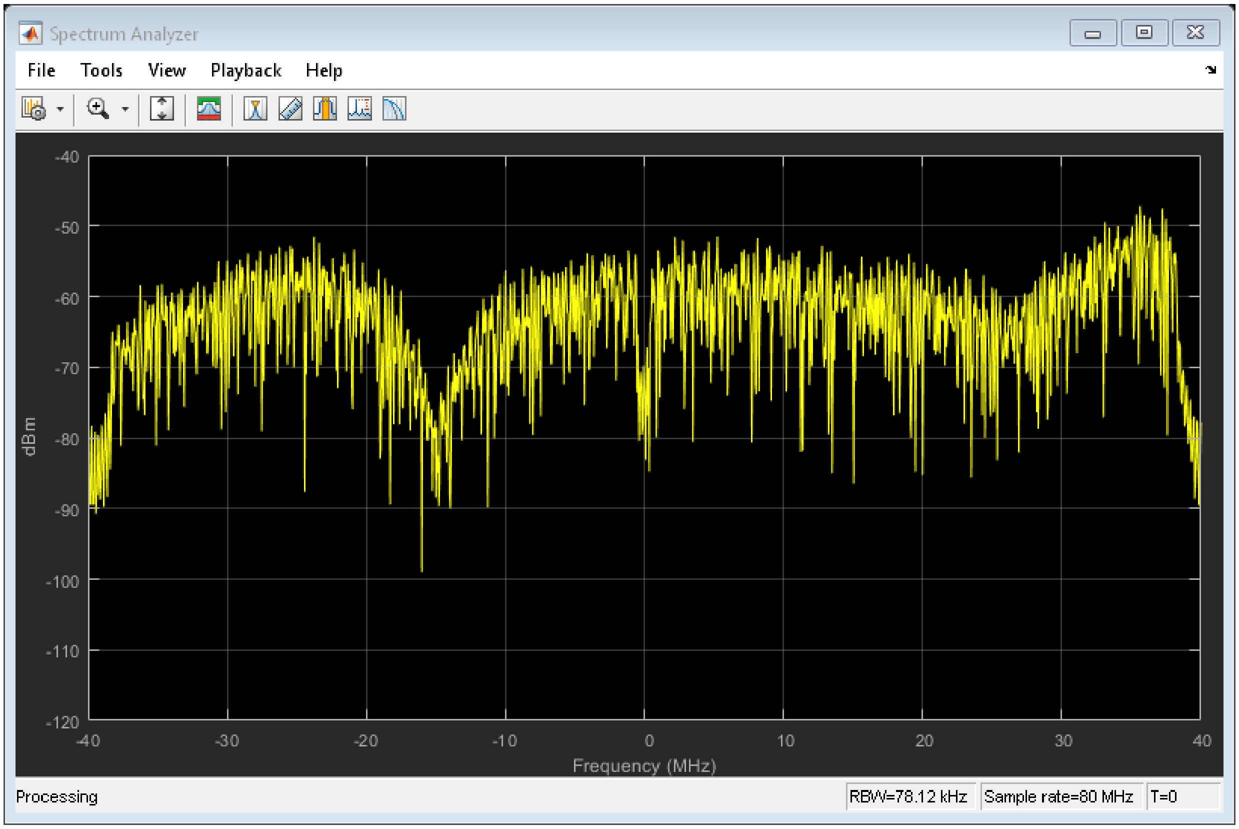
```
tgacChan = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',bw, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
```

Pass the VHT waveform through the channel.

```
rxSig = tgacChan(txSig);
```

Plot the spectrum of the received waveform.

```
saScope = dsp.SpectrumAnalyzer('SampleRate',fs,'YLimits',[-120 -40]);
saScope(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');
txSig = wlanWaveformGenerator([1;0;0;1],cfg);
```

Create a 4x2 MIMO TGac channel and disable large-scale fading effects.

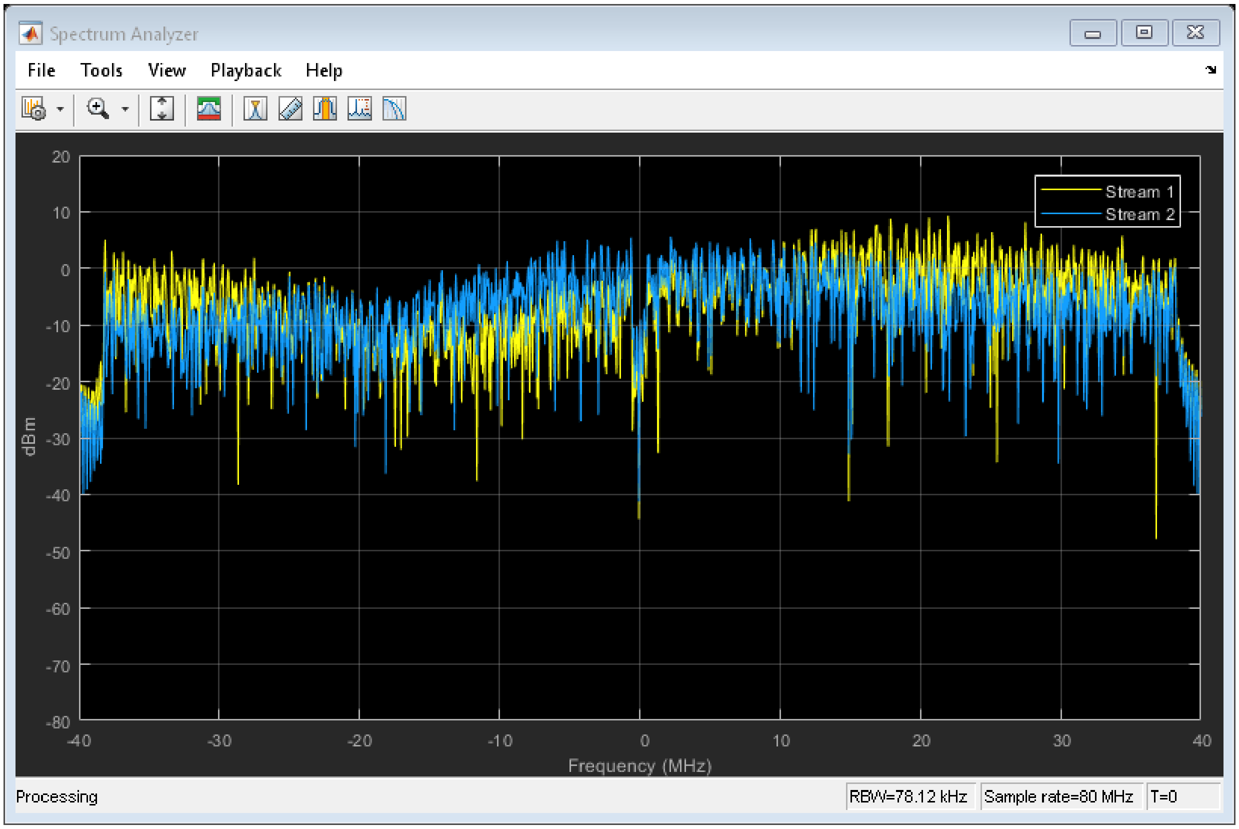
```
tgacChan = wlanTGacChannel('SampleRate',80e6,'ChannelBandwidth','CBW80', ...
    'NumTransmitAntennas',4,'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','None');
```

Pass the transmit waveform through the channel.

```
rxSig = tgacChan(txSig);
```

Display the spectrum of the two received space-time streams.

```
saScope = dsp.SpectrumAnalyzer('SampleRate',80e6, ...
    'ShowLegend',true, ...
    'ChannelNames',{'Stream 1','Stream 2'});
saScope(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.

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

Create an AWGN channel noise, setting SNR = 15 dB.

```
chNoise = comm.AWGNChannel('NoiseMethod','Signal to noise ratio (SNR)',...
    'SNR',15);
```

Pass the signals through the TGac channel and noise models.

```
rxLTF = chNoise(tgacChan(txLTF));
rxDataSig = chNoise(tgacChan(txDataSig));
```

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);
rxNoise = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
```

Pass the signals through the reciever noise model.

```
rxLTF = rxNoise(rxLTF);
rxDataSig = rxNoise(rxDataSig);
```

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

Bandwidth, *W*	Sampling Rate Expansion Factor	PDP Tap Spacing (ns)
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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

Use in a MATLAB Function block is not supported.

See Also

`comm.MIMOChannel` | `wlanTGahChannel` | `wlanTGnChannel`

Introduced in R2015b

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).

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

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`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 (MATLAB) 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;
txSig = wlanWaveformGenerator(randi([0 1],1000,1),cfg);
```

Create a TGac SISO channel with path loss and shadowing enabled.

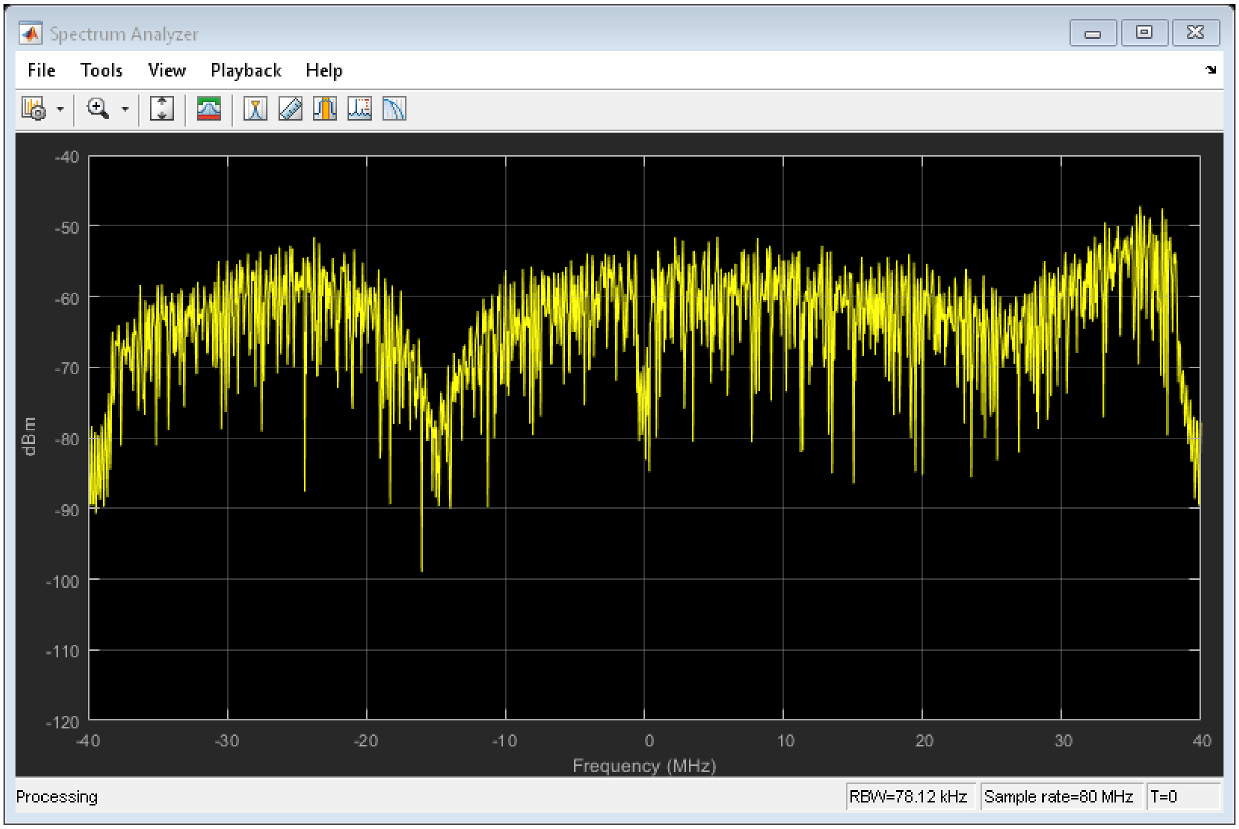
```
tgacChan = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',bw, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
```

Pass the VHT waveform through the channel.

```
rxSig = tgacChan(txSig);
```

Plot the spectrum of the received waveform.

```
saScope = dsp.SpectrumAnalyzer('SampleRate',fs,'YLimits',[-120 -40]);
saScope(rxSig)
```



Because path loss and shadowing are enabled, the mean received power across the spectrum is approximately -60 dBm.

wlanTGahChannel System object

Filter signal through 802.11ah multipath fading channel

Description

The `wlanTGahChannel` System object filters an input signal through an 802.11ah (TGah) indoor MIMO channel as specified in [1], following the MIMO modeling approach in [4].

The fading processing assumes the same parameters for all N_T-by-N_R links of the TGah 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 TGah multipath fading channel:

1. Define and set up your TGah channel object. See "Construction" on page 3-20.
2. Call `step` to filter the input signal through a TGah multipath fading channel according to the properties of `wlanTGahChannel`.

Note Alternatively, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`tgah = wlanTGahChannel` creates a TGah fading channel System object, `tgah`. This object filters a real or complex input signal through the TGah channel to obtain the channel-impaired signal.

`tgah = wlanTGahChannel(Name,Value)` creates a TGah channel object, `tgah`, 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

Sample rate of the input signal in Hz, specified as a real positive scalar. The default is `2e6`.

`DelayProfile`

Delay profile model

Delay profile model, specified as `'Model-A'`, `'Model-B'`, `'Model-C'`, `'Model-D'`, `'Model-E'`, or `'Model-F'`. The default is `'Model-B'`.

The table summarizes the models properties before the bandwidth reduction factor.

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, specified as `'CBW1'`, `'CBW2'`, `'CBW4'`, `'CBW8'`, or `'CBW16'`. The default is `'CBW2'`, which corresponds to a 2 MHz channel bandwidth.

As specified in *TGac Channel Model Addendum* [3], a reduction factor is applied to the multipath spacing of the power delay profile for channel bandwidths greater than 4 MHz.

The reduction factor applied to the multipath spacing is $2^{\text{ceil}(\log 2(BW/4))}$, where BW is the channel bandwidth in MHz.

`CarrierFrequency`

RF carrier frequency

RF carrier frequency in Hz, specified as a real positive scalar. The default is `915e6` Hz.

`TransmitReceiveDistance`

Distance between transmitter and receiver

Distance between the transmitter and receiver in meters, specified as a real positive scalar. The default is `3` meters.

`TransmitReceiveDistance` is used to compute the path loss, and to determine whether the channel has a line-of-sight (LOS) or no-line-of-sight (NLOS) condition. The path loss and standard deviation of shadow fading loss depend on the separation between the transmitter and the receiver.

`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 for single or multi-user scenario

User index, specified as a nonnegative integer scalar. `UserIndex` specifies the single user or a particular user in a multiuser scenario. The default is `0`.

A pseudorandom per-user angle-of-arrival (AoA) and angle-of-departure (AoD) rotation is applied to support a multi-user scenario. A value of `0` indicates a simulation scenario that does not require per-user angle diversity and assumes the TGn defined cluster AoAs and AoDs.

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 `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`.

`TransmitAntennaSpacing` supports uniform linear array only. This property applies only 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`.

`ReceiveAntennaSpacing` supports uniform linear array only. This property applies only 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'`.

FloorSeparation

Floor separation

Floor separation, specified as a real scalar, indicating the number of building floors between the transmitter and the receiver in order to account for the floor attenuation loss in the calculation of path loss in a multiple floor scenario. The default is `0`, which represents a communication link between a transmitter and a receiver located on the same floor.

The `FloorSeparation` property applies only when `DelayProfile` is `'Model-A'` or `'Model-B'`.

FluorescentEffect

Enable fluorescent effect

To include Doppler effects due to fluorescent lighting, set this property to `true` (default).

The `FluorescentEffect` property applies only when `DelayProfile` is `'Model-D'` or `'Model-E'`.

PowerLineFrequency

Frequency of the power line (Hz)

Frequency of the power line in Hz, specified as `'50Hz'` or `'60Hz'`. The default is `'60Hz'`.

The power line frequency is 60 Hz in USA and 50 Hz in Europe. This property applies only when you set `FluorescentEffect` to `true` and `DelayProfile` to `'Model-D'` or `'Model-E'`.

NormalizeChannelOutputs

Normalize channel outputs

To normalize the channel outputs by the number of receive antennas, set this property to `true` (default).

`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 stream, specified as a real, nonnegative integer scalar. The default is `73`.

This property applies only when you set the `RandomStream` property to `'mt19937ar with seed'`. The `Seed` property reinitializes the mt19937ar random number stream in the `reset` method.

`PathGainsOutputPort`

Enable path gain output

To enable computation of path gain output, set this property to `true`. The default is `false`.

Methods

info	Display information about TGah Channel object
reset	Reset states of the `wlanTGahChannel` object
step	Filter signal through 802.11ah multipath fading channel

Common to All System Objects	
`clone`	Create System object with same property values
`getNumInputs`	Expected number of inputs to a System object
`getNumOutputs`	Expected number of outputs of a System object
`isLocked`	Check locked states of a System object (logical)
`release`	Allow System object property value changes

Examples

Pass S1G Waveform Through TGah Channel

Filter an 802.11ah waveform through a TGah channel. Specify a seed value to produce a repeatable channel output.

Create an S1G configuration object and waveform.

```
cfgS1G = wlanS1GConfig;
txWaveform = wlanWaveformGenerator([1;0;0;1],cfgS1G);
```

Create a TGah channel object and adjust some default properties. Pass the S1G waveform through the channel by supplying it as an input to the TGah channel object.

```
tgah = wlanTGahChannel;
tgah.LargeScaleFadingEffect = 'PathLoss and shadowing';
tgah.FloorSeparation = 2;
tgah.RandomStream = 'mt19937ar with seed';
tgah.Seed = 10;
```

```
channelOutput = tgah(txWaveform);
```

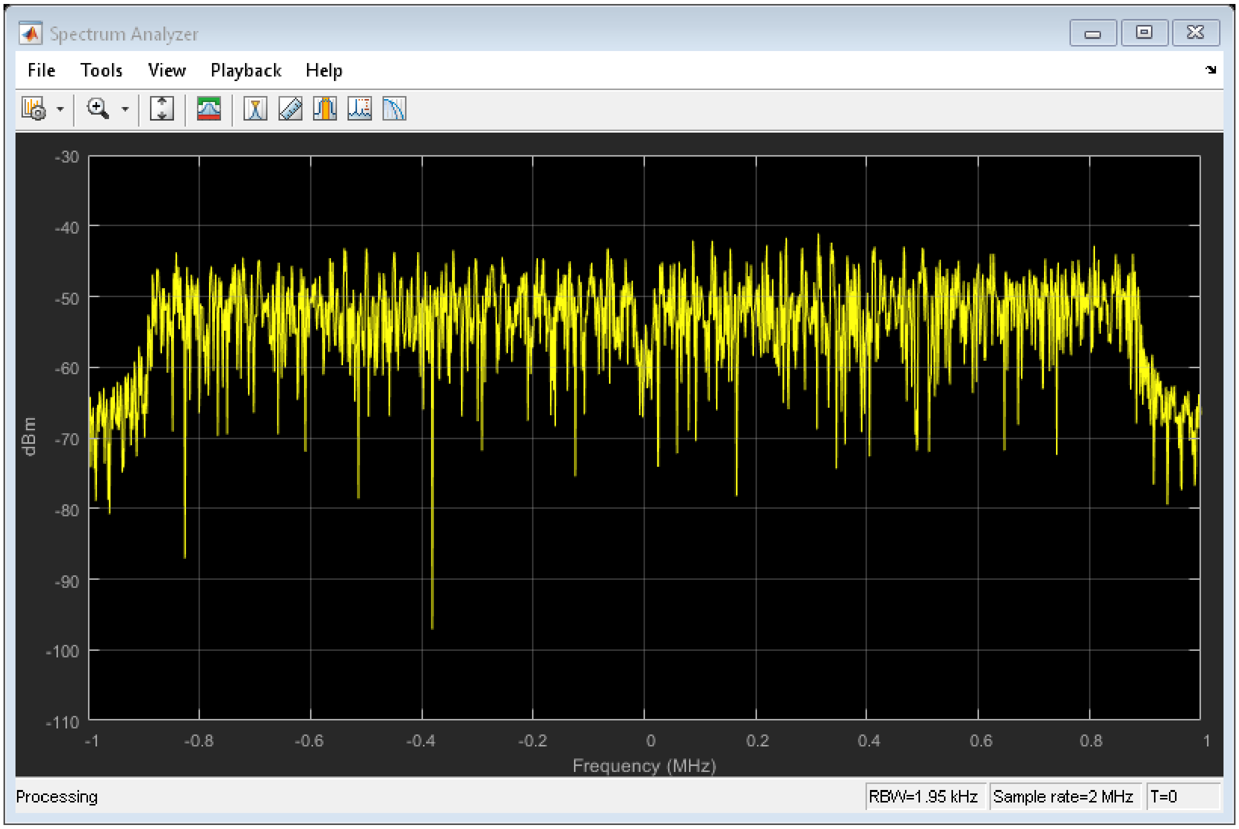
Confirm the channel bandwidth and set the corresponding sample rate.

```
cfgS1G.ChannelBandwidth
fs = 2e6;
```

```
ans =

    'CBW2'
```

Plot the spectrum of the channel output waveform.

```
saScope = dsp.SpectrumAnalyzer('SampleRate',fs,'YLimits',[-110 -30]);
saScope(channelOutput)
```



Across the spectrum, the mean power of the channel output waveform is approximately -50 dBm.

TGah Channel Model-B Delay Profile

Plot the delay profile for an impulse waveform passed through a TGah channel.

Create an impulse waveform.

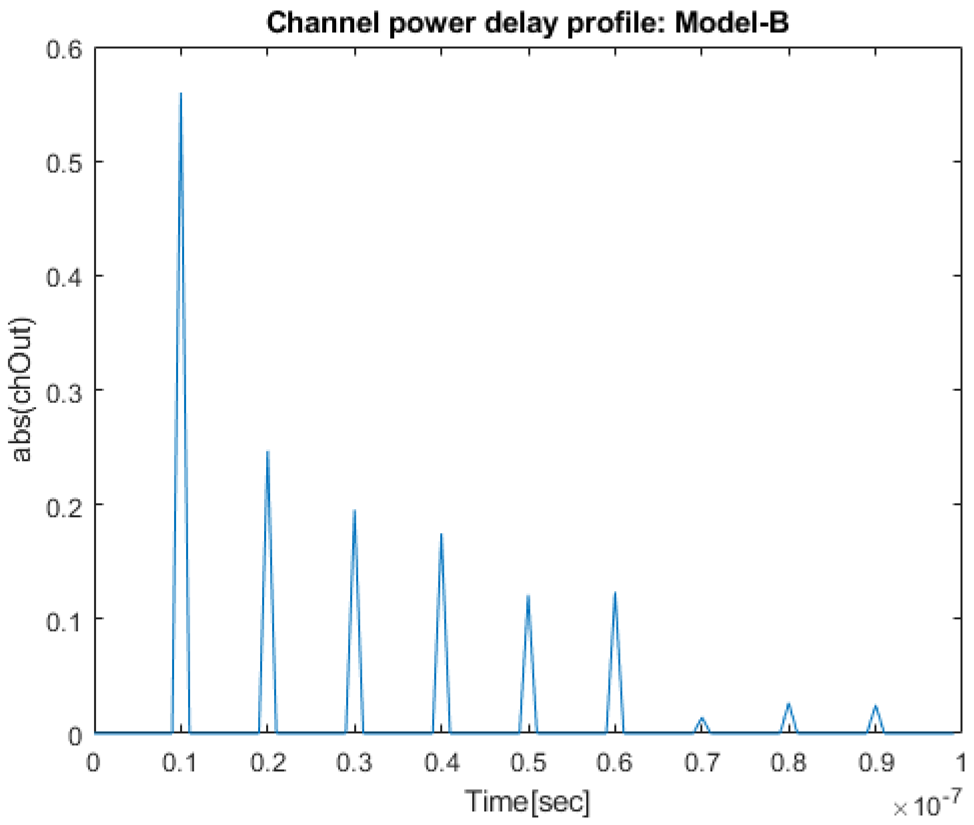
```
txWaveform = zeros(100,1);
% The impulse is delayed by 10 samples. This is equivalent to 10nsec in
% time.
txWaveform(11) = 1;
```

Create a TGah channel object. Here we specify the seed so that results can be replicated.

```
tgah = wlanTGahChannel;
tgah.RandomStream = 'mt19937ar with seed';
tgah.Seed = 10;
```

Set the sample rate so that sampling of the channel multipaths are integer multiples of integer sampling delay.

```
tgah.SampleRate = 1e9;

chOut = tgah(txWaveform);
plot((0:length(chOut)-1)*1/tgah.SampleRate,abs(chOut));
xlabel('Time[sec]'); ylabel('abs(chOut)');
title('Channel power delay profile: Model-B')
```



Transmit S1G Waveform Through 4-by-2 MIMO Channel

Create a S1G waveform generated using four transmit antennas and two spatial streams.

```
cfg = wlanS1GConfig('NumTransmitAntennas',4,'NumSpaceTimeStreams',2, ...
    'SpatialMapping','Fourier');
txSig = wlanWaveformGenerator([1;0;0;1],cfg);
```

Create a 4x2 MIMO TGah channel and disable large-scale fading effects.

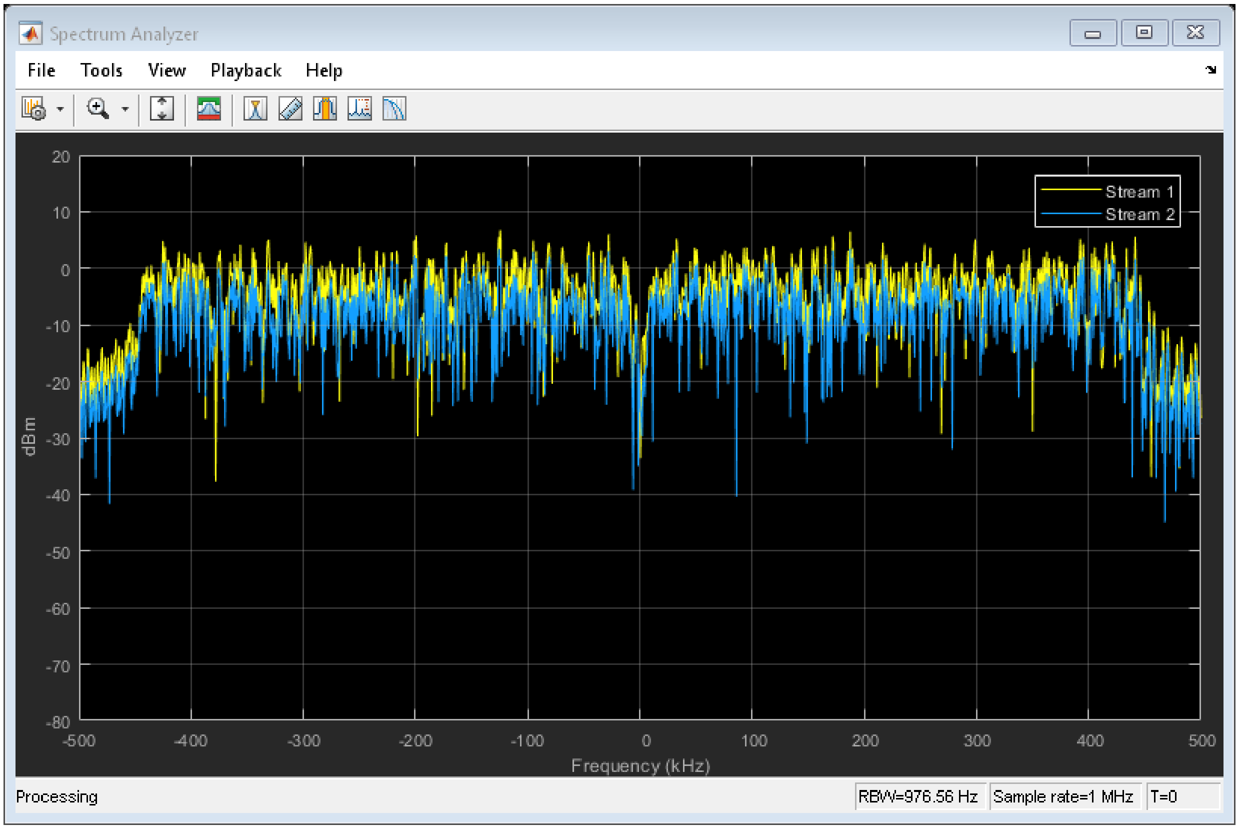
```
tgahChan = wlanTGahChannel('SampleRate',1e6,'ChannelBandwidth','CBW1', ...
    'NumTransmitAntennas',4,'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','None');
```

Pass the transmit waveform through the channel.

```
rxSig = tgahChan(txSig);
```

Display the spectrum of the two received space-time streams.

```
saScope = dsp.SpectrumAnalyzer('SampleRate',1e6, ...
    'ShowLegend',true, ...
    'ChannelNames',{'Stream 1','Stream 2'});
saScope(rxSig)
```



Algorithms

The algorithms used to model the TGah channel are based on those used for the TGn channel (as described in `wlanTGnChannel` and *TGn Channel Models* [2]) and the TGac channel (as described in `wlanTGacChannel` and *TGac Channel Model Addendum* [3]). Complete information on the changes required to support TGah channels can be found in *TGah Channel Model* [1]. The changes to support the TGah channel include lower bandwidths, floor separation attenuation, MIMO enhancements, and path loss and shadowing.

Lower Bandwidths

The TGah channel model supports channel bandwidths down to 1 MHz.

Floor Separation Attenuation

The TGah channel model includes floor separation attenuation effects. In the TGah channel, the path loss model used in the spatial correlation computation is updated to include floor separation attenuation effects. The `FloorSeparation` property applies only when `DelayProfile` is `'Model-A'` or `'Model-B'`, and `LargeScaleFadingEffect` is `'Pathloss'`, `'Shadowing'`, or `'Pathloss and shadowing'`. For more information, see *TGah Channel Model* [1].

MIMO Enhancements

The TGah channel model supports no more than 4x4 MIMO.

The TGah model also includes support for multiple users as simultaneous communication takes place between access points and user stations. Accordingly, the TGah model extends the concept of cluster angles of arrival and departure to account for point-to-multipoint transmission. For more information, see *Stochastic MIMO Radio Channel Model with Experimental Validation* [4].

Path Loss and Shadowing

TGah Channel Model [1], Table 2 defines path loss parameters that are slightly modified from those defined for TGn. Specifically, the shadow fading values corresponding to breakpoint distance are 1 dB less for all TGah channel models.

The path loss exponent and the standard deviation of the shadow fading loss characterize each model. The two parameters 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

Parameter	Model					
	A	B	C	D	E	F
Path loss exponent for $d \leq d_{BP}$	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 $d \leq d_{BP}$	2	2	2	2	2	2
Shadow fading σ (dB) for $d > d_{BP}$	3	3	4	4	5	5

References

[1] Porat R., S.K.. Yong, and K. Doppler. *TGah Channel Model*. IEEE 802.11-11/0968r4, March 2015.

[2] Erceg, V., L. Schumacher, P. Kyritsi, et al. *TGn Channel Models*. Version 4. IEEE 802.11-03/940r4, May 2004.

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

[4] 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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

Use in a MATLAB Function block is not supported.

See Also

System Objects

`comm.MIMOChannel` | `wlanTGacChannel` | `wlanTGnChannel`

Introduced in R2017a

info

System object: wlanTGahChannel

Display information about TGah Channel object

Syntax

Description

`s` = info(`obj` returns a structure containing information about the `wlanTGahChannel` object characteristics. The information structure contains:

- `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).

Introduced in R2017a

reset

System object: wlanTGahChannel

Reset states of the `wlanTGahChannel` object

Syntax

```
reset(obj)
```

Description

`reset(obj)` resets the states of the `wlanTGahChannel` System object.

If the `RandomStream` property of `obj` is set 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.

Introduced in R2017a

step

System object: wlanTGahChannel

Filter signal through 802.11ah multipath fading channel

Syntax

```
Y = step(obj,X)
[Y,pathGains] = step(obj,X)
```

Description

Note Alternatively, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(obj,X)` filters input signal `X` through 802.11ah (TGah) fading channel `obj` and returns the result in `Y`. The input `X` can be a scalar, vector, or N_S-by-$N_T$matrix with real or complex values. N_S is the number of samples and N_T is the number of transmit antennas. `Y` is returned as a N_S-by-N_R matrix of complex values. N_R is the number of receive antennas.

`[Y,pathGains] = step(obj,X)` also returns `pathGains` as a N_S-by-N_P-by-N_T-by-N_R matrix of complex values. N_P is the number of paths in the channel.

Note `obj` 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 (MATLAB) 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

Pass S1G Waveform Through TGah Channel

Filter an 802.11ah waveform through a TGah channel. Specify a seed value to produce a repeatable channel output.

Create an S1G configuration object and waveform.

```
cfgS1G = wlanS1GConfig;
txWaveform = wlanWaveformGenerator([1;0;0;1],cfgS1G);
```

Create a TGah channel object and adjust some default properties. Pass the S1G waveform through the channel by supplying it as an input to the TGah channel object.

```
tgah = wlanTGahChannel;
tgah.LargeScaleFadingEffect = 'PathLoss and shadowing';
tgah.FloorSeparation = 2;
tgah.RandomStream = 'mt19937ar with seed';
tgah.Seed = 10;

channelOutput = tgah(txWaveform);
```

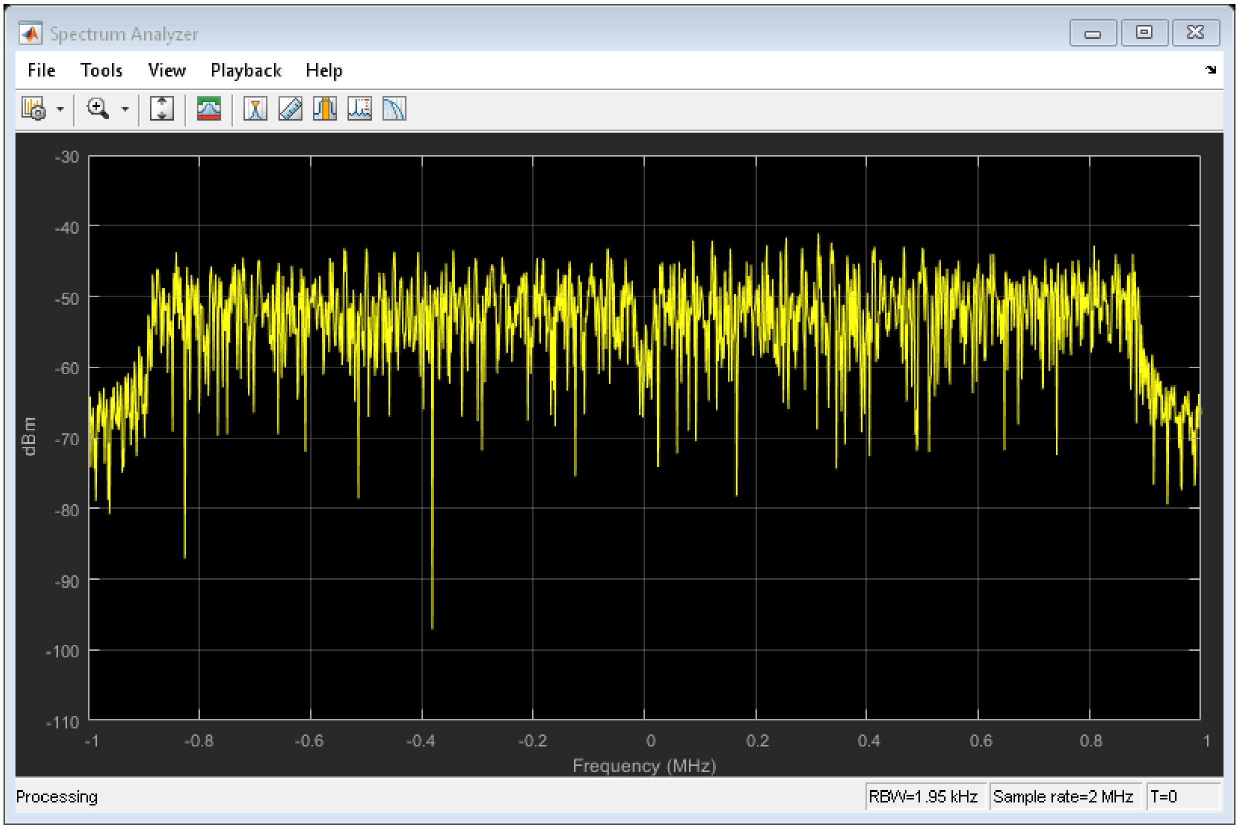
Confirm the channel bandwidth and set the corresponding sample rate.

```
cfgS1G.ChannelBandwidth
fs = 2e6;
```

```
ans =

    'CBW2'
```

Plot the spectrum of the channel output waveform.

```
saScope = dsp.SpectrumAnalyzer('SampleRate',fs,'YLimits',[-110 -30]);
saScope(channelOutput)
```



Across the spectrum, the mean power of the channel output waveform is approximately -50 dBm.

Introduced in R2017a

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 3-41.

2 Call `step` to filter the input signal through a TGn multipath fading channel according to the properties of `wlanTGnChannel`.

Note Alternatively, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

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 `'Model-A'`, `'Model-B'`, `'Model-C'`, `'Model-D'`, `'Model-E'`, or `'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 `'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'`.

NormalizeChannelOutputs

Normalize channel outputs

To normalize the channel outputs by the number of receive antennas, set this property to `true` (default).

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.

`PathGainsOutputPort`

Enable path gain output

To enable computation of path gain output, set this property to `true`. The default value of this property is `false`.

Methods

info	Characteristic information about TGn Channel
reset	Reset states of the `wlanTGnChannel` object
step	Filter signal through 802.11n multipath fading channel

Common to All System Objects	
`clone`	Create System object with same property values
`getNumInputs`	Expected number of inputs to a System object
`getNumOutputs`	Expected number of outputs of a System object
`isLocked`	Check locked states of a System object (logical)
`release`	Allow System object property value changes

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);
txSig = wlanWaveformGenerator(randi([0 1],1000,1),cfg);
```

Create a TGn SISO channel with path loss and shadowing enabled.

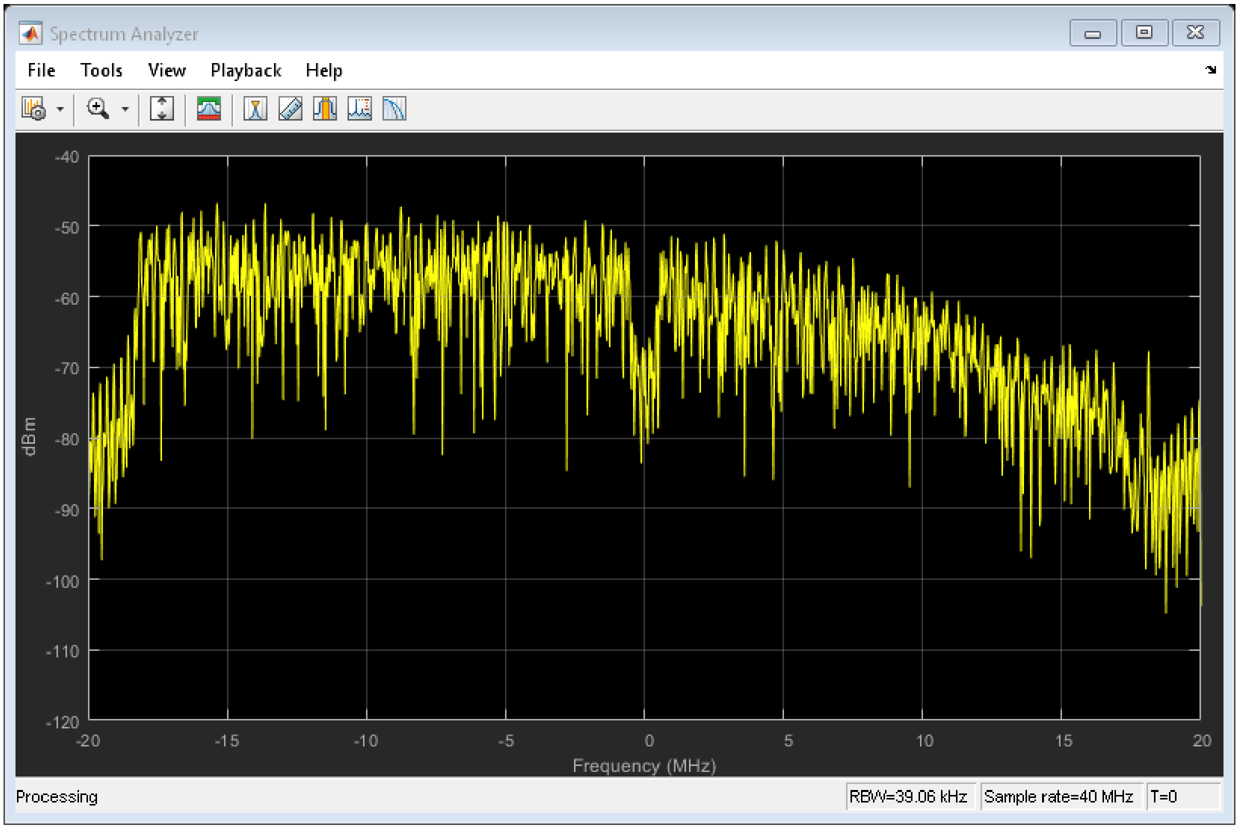
```
tgnChan = wlanTGnChannel('SampleRate',fs, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
```

Pass the HT waveform through the channel.

```
rxSig = tgnChan(txSig);
```

Plot the spectrum of the received waveform.

```
saScope = dsp.SpectrumAnalyzer('SampleRate',fs,'YLimits',[-120 -40]);
saScope(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');
txSig = wlanWaveformGenerator([1;0;0;1],cfg);
```

Create a 4x2 MIMO TGn channel and disable large-scale fading effects.

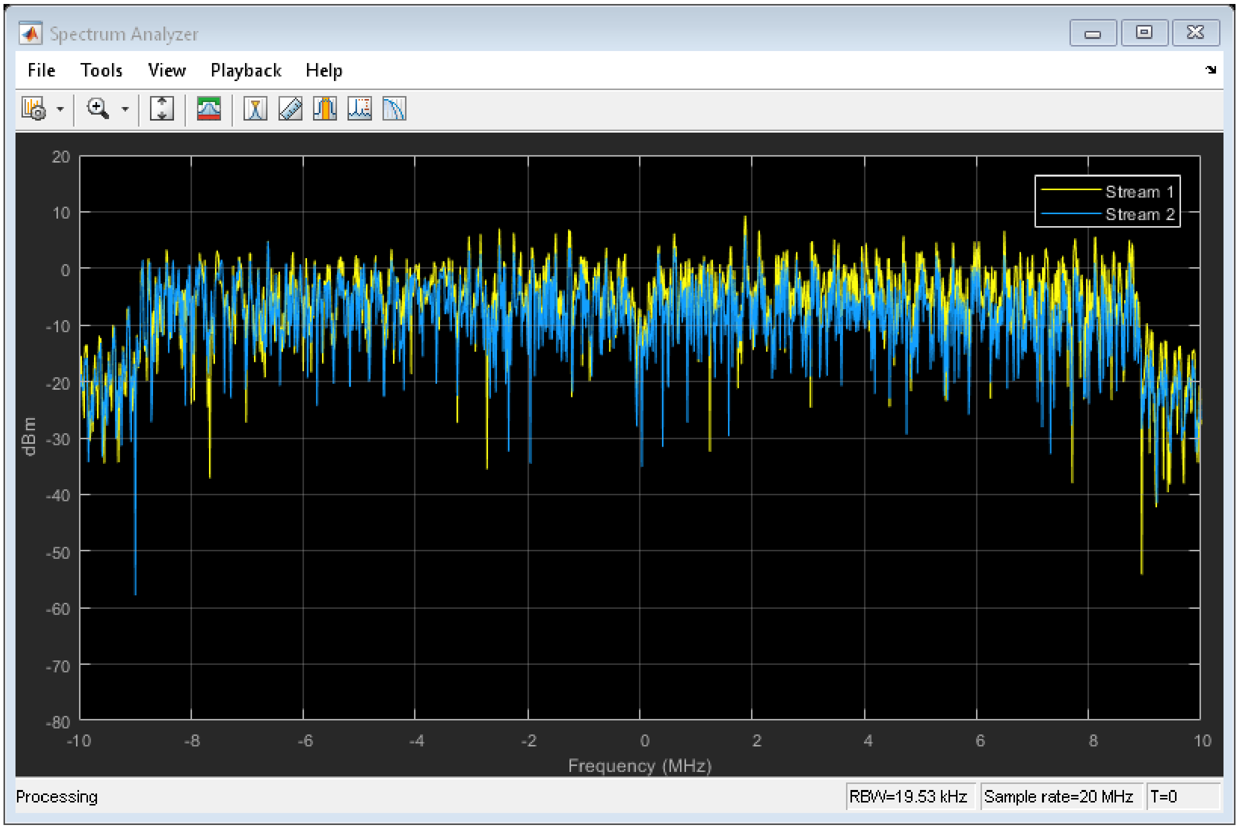
```
tgnChan = wlanTGnChannel('SampleRate',20e6, ...
    'NumTransmitAntennas',4, ...
    'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','None');
```

Pass the transmit waveform through the channel.

```
rxSig = tgnChan(txSig);
```

Display the spectrum of the two received space-time streams.

```
saScope = dsp.SpectrumAnalyzer('SampleRate',20e6, ...
    'ShowLegend',true, ...
    'ChannelNames',{'Stream 1','Stream 2'});
saScope(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.

```
tgnChan = wlanTGnChannel('SampleRate',fs, ...
    'NumTransmitAntennas',2,'NumReceiveAntennas',2, ...
    'LargeScaleFadingEffect','None');
```

Create AWGN channel noise, setting SNR = 15 dB.

```
chNoise = comm.AWGNChannel('NoiseMethod','Signal to noise ratio (SNR)',...
    'SNR',15);
```

Pass the signals through the TGn channel and noise models.

```
rxLTF = chNoise(tgnChan(txLTF));
rxDataSig = chNoise(tgnChan(txDataSig));
```

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);
awgnChan = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
```

Pass the signals through the channel.

```
rxLTF = awgnChan(rxLTF);
rxDataSig = awgnChan(rxDataSig);
```

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	
Power (dB)	0	–5.4	–10.8	–16.2	
Angle of arrival (°)	4.3	4.3	4.3	4.3	
Receiver angular spread (°)	14.4	14.4	14.4	14.4	
Angle of departure (°)	225.1	225.1	225.1	225.1	
Transmitter angular spread (°)	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, 2π]. 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 $d \leq d_{BP}$	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 $d \leq d_{BP}$	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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

Use in a MATLAB Function block is not supported.

See Also

`comm.MIMOChannel` | `wlanTGacChannel` | `wlanTGahChannel`

Introduced in R2015b

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).

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

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`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 (MATLAB) 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);
txSig = wlanWaveformGenerator(randi([0 1],1000,1),cfg);
```

Create a TGn SISO channel with path loss and shadowing enabled.

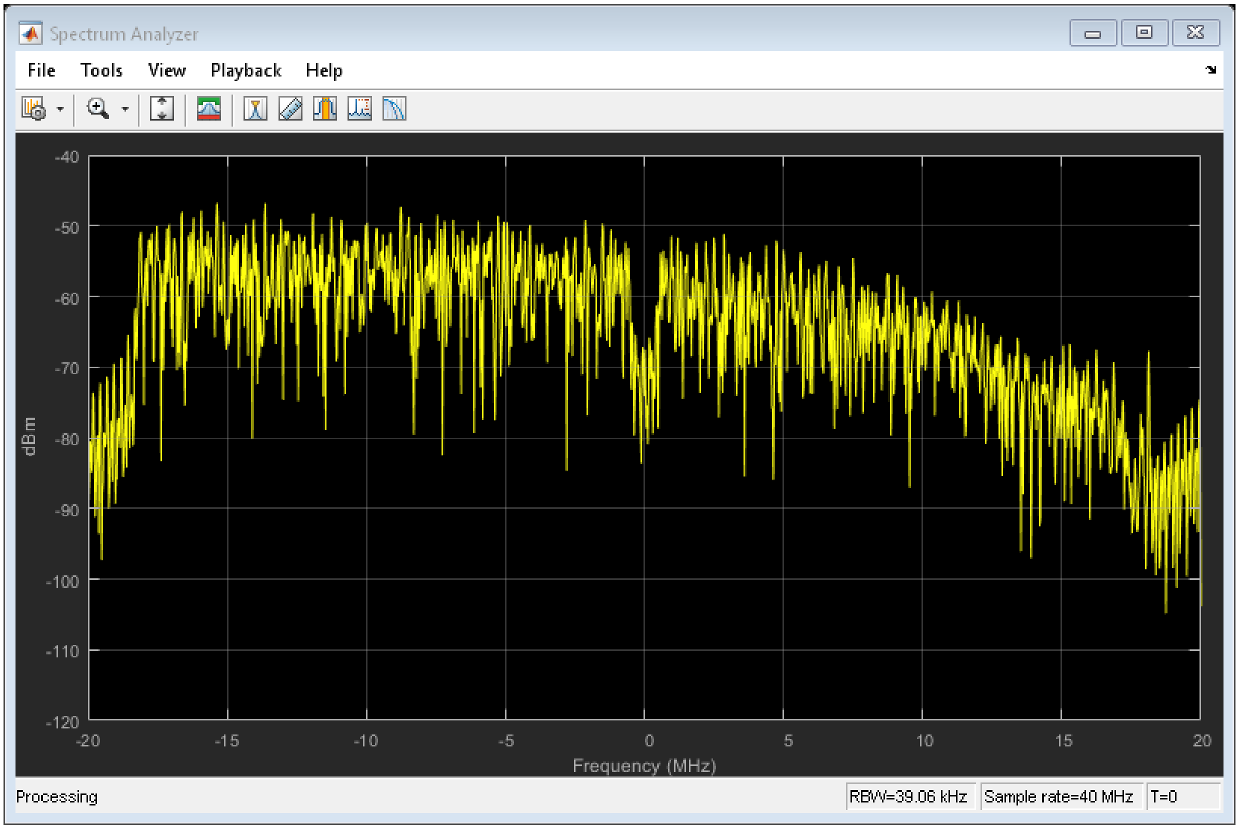
```
tgnChan = wlanTGnChannel('SampleRate',fs, ...
    'LargeScaleFadingEffect','Pathloss and shadowing');
```

Pass the HT waveform through the channel.

```
rxSig = tgnChan(txSig);
```

Plot the spectrum of the received waveform.

```
saScope = dsp.SpectrumAnalyzer('SampleRate',fs,'YLimits',[-120 -40]);
saScope(rxSig)
```



Because path loss and shadowing are enabled, the mean received power across the spectrum is approximately -60 dBm.