WLAN System Toolbox™

User's Guide

MATLAB®

R2015b

MathWorks®

How to Contact MathWorks

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

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

WLAN System Toolbox™ User's Guide

Revision History

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

Contents

1 Tutorials

Tutorials

WLAN Parameterization

WLAN System Toolbox configuration objects initialize, store, and validate configuration properties. The functions in the toolbox use these properties to initialize parameter settings that define the characteristics of waveforms generated and to define the recovery process.

Configuration Objects in WLAN System Toolbox

The configuration objects are designed specifically as containers to store properties. They also provide some level of data validation for the function inputs that they maintain. Functions perform further data validation across input settings based on runtime conditions.

The configuration objects are optimized for iterative computations that process large streams of data, such as communications systems. For more information on MATLAB® objects, see “Using Objects”.

WLAN System Toolbox configuration objects define and configure format-specific and function-specific properties. The property page of each object contains descriptions, valid settings, ranges, and other information about the object properties.

- `wlanVHTConfig` — The VHT configuration object defines and configures very high throughput (VHT) transmission PPDUs. See wlanVHTConfig Properties.
- `wlanHTConfig` — The HT configuration object defines and configures high throughput (HT) transmission PPDUs. See wlanHTConfig Properties.
- `wlanNonHTConfig` — The non-HT configuration object defines and configures non-high throughput (non-HT) transmission PPDUs. See wlanNonHTConfig Properties.
- `wlanGeneratorConfig` — The generator configuration object defines and configures waveform generation characteristics for transmission of WLAN PPDUs. See wlanGeneratorConfig Properties.
- `wlanRecoveryConfig` — The recovery configuration object defines and configures information recovery characteristics for received WLAN transmission PPDUs. See wlanRecoveryConfig Properties.

See Also

“WLAN Packet Structure” on page 1-3 | “Create Configuration Objects” | “What Is WLAN?”

WLAN Packet Structure

Physical Layer Conformance Procedure Protocol Data Unit

IEEE® 802.11™[1][2] is a packet-based protocol. Each physical layer conformance procedure (PLCP) protocol data unit (PPDU) contains preamble and data fields. The preamble field contains the transmission vector format information. The data field contains the user payload and higher layer headers, such as MAC fields and CRC. The transmission vector format and the PPDU packet structure vary depending on the 802.11 version being configured for transmission. The transmission vector (*TXVECTOR*) format parameter is classified as:

- *VHT* to specify a very high throughput PHY implementation. VHT refers to preamble fields formatted for association with 802.11ac data. IEEE 802.11ac-2013 [3], Section 22 defines and describes the VHT PHY layer and PPDU.
- *HT* to specify a high throughput PHY implementation. HT refers to preamble fields formatted for association with 802.11n data. IEEE 802.11-2012 [2], Section 20 defines and describes the HT PHY layer and PPDU. The standard defines two HT formats:
  - *HT_MF* indicates the HT-mixed format and contains a preamble compatible with HT and non-HT receivers. Support for HT-mixed format is mandatory.
  - *HT_GF* indicates the HT-greenfield format and does not contain a non-HT compatible part. WLAN System Toolbox does not support HT_GF format.
- *non-HT* to specify a PHY implementation that is not HT and is not VHT. Non-HT refers to preamble fields formatted for association with pre-802.11n™ data. IEEE 802.11-2012 [2], Section 18 defines and describes the OFDM PHY layer and PPDU for non-HT transmission. In addition to supporting non-HT synchronization, the non-HT preamble fields are used in support of HT and VHT synchronization.

The *TXVECTOR* parameters, as defined in IEEE 802.11ac-2013 [3], Table 22-1, determine the structure of PPDUs transmitted by a VHT STA. For a VHT STA, the *FORMAT* parameter determines the overall structure of the PPDU and enables:

- Non-HT format (*NON_HT*), based on Section 18 and including non-HT duplicate format.

1. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.
2. IEEE Std 802.11ac™-2013 Adapted and reprinted with permission from IEEE. Copyright IEEE 2013. All rights reserved.

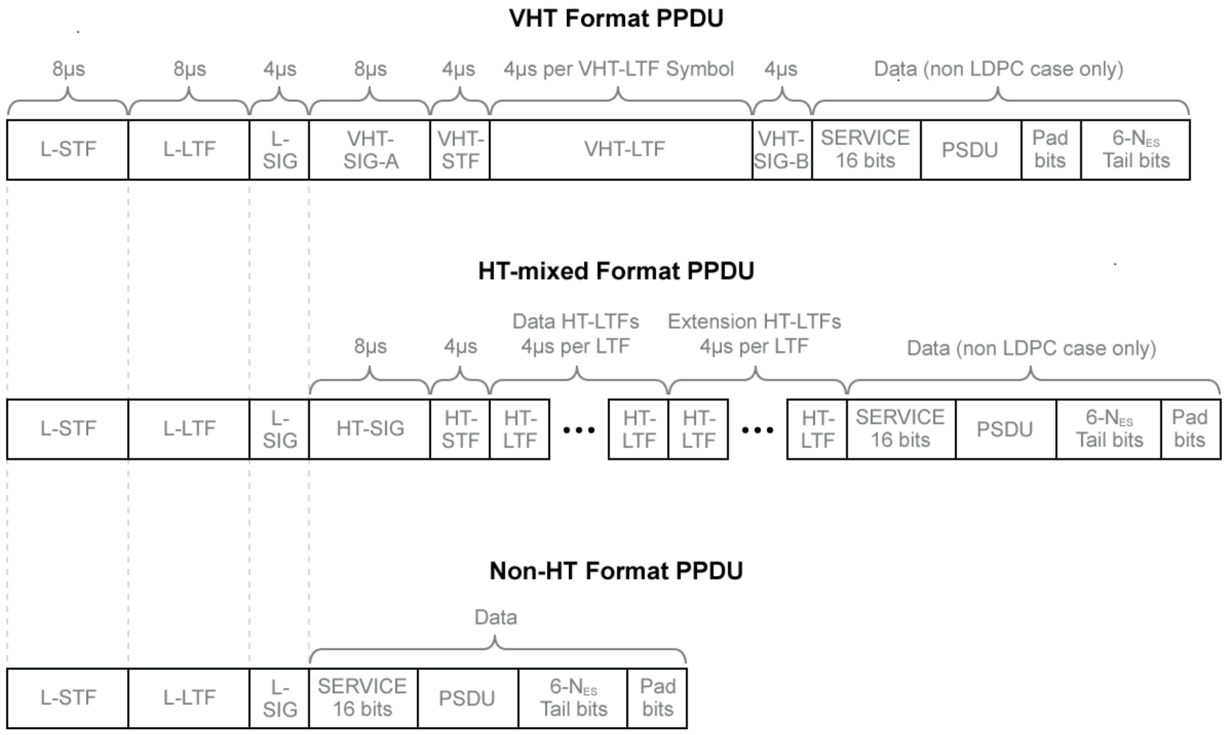
- HT-mixed format (*HT_MF*), as specified in Section 20.
- HT-greenfield format (*HT_GF*), as specified in Section 20. WLAN System Toolbox does not support HT_GF format.
- VHT format (*VHT*), as specified in Section 22. The VHT format PPDUs contain a preamble compatible with Section 18 and Section 20 STAs. The non-VHT portions of the VHT format preamble (the parts of VHT preamble preceding the VHT-SIG-A field) are defined to enable decoding of the PPDU by VHT STAs.

The table shows 802.11 versions that the toolbox supports, along with the supported *TXVECTOR* options and associated modulation formats.

802.11 version	**Transmission Vector Format**	**Modulation format**	**Bandwidths (MHz)**
802.11b	non-HT	DSSS/CCK	20
802.11a	non-HT	OFDM only	20
802.11g	non-HT	OFDM and DSSS/CCK	20
802.11n	HT_MF, Non-HT	OFDM only	20, 40
802.11ac	VHT, HT_MF, Non-HT	OFDM only	20, 40, 80, 160

WLAN System Toolbox configuration objects define the properties that enable creation of PPDUs and waveforms for the specified 802.11 transmission format. See wlanVHTConfig Properties, wlanHTConfig Properties, wlanNonHTConfig Properties, and wlanGeneratorConfig Properties.

The field structure for VHT, HT, and non-HT PPDUs consist of preamble and data portions. The legacy preamble fields (L-STF, L-LTF, and L-SIG) are common to VHT, HT, and non-HT format preambles. VHT and HT format preamble fields include additional format-specific training and signaling fields. Each format defines a data field for transmission of user payload data.



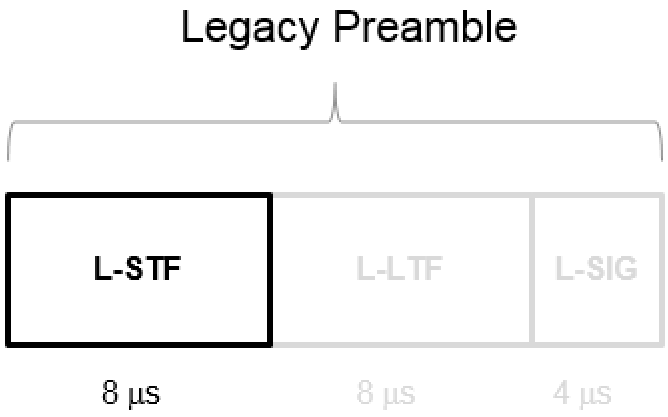
PPDU Field Abbreviation	Description
L-STF	Non-HT Short Training field
L-LTF	Non-HT Long Training field
L-SIG	Non-HT SIGNAL field
HT-SIG	HT SIGNAL field
HT-STF	HT Short Training field
HT-LTF	HT Long Training field, multiple HT-LTFs are transmitted as indicated by the MCS
VHT-SIG-A	VHT Signal A field
VHT-STF	VHT Short Training field
VHT-LTF	VHT Long Training field
VHT-SIG-B	VHT Signal B field

PPDU Field Abbreviation	Description
Data	VHT, HT, and non-HT Data fields include the service bits, PSDU, tail bits, and pad bits

See IEEE 802.11-2012 [2], Section 20.3.2 for more information.

Non-HT (Legacy) Short Training Field

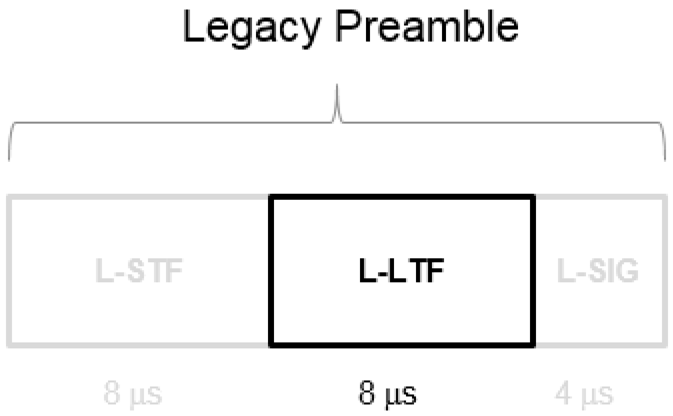
The legacy short training field (L-STF) is the first field of the legacy preamble for 802.11a/g.



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

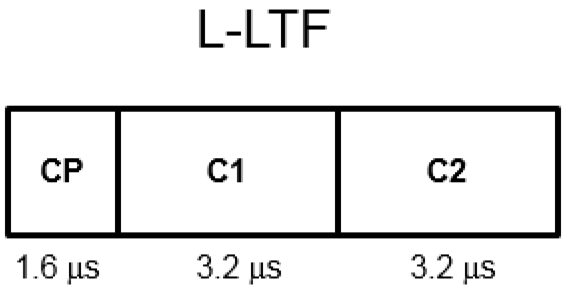
Non-HT (Legacy) Long Training Field

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



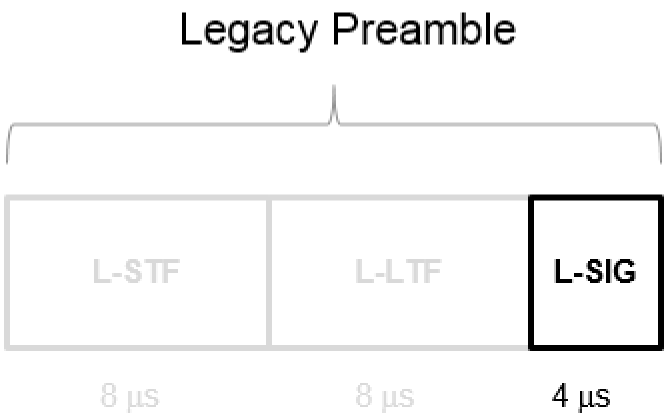
It is also the second field of the HT-mixed format preamble used in 802.11n and the VHT format preamble used in 802.11ac. Channel estimation, frequency offset estimation, and time synchronization rely on the L-LTF. The long OFDM training symbol consists of 53 subcarriers. IEEE Std 802.11-2012, Equation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

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

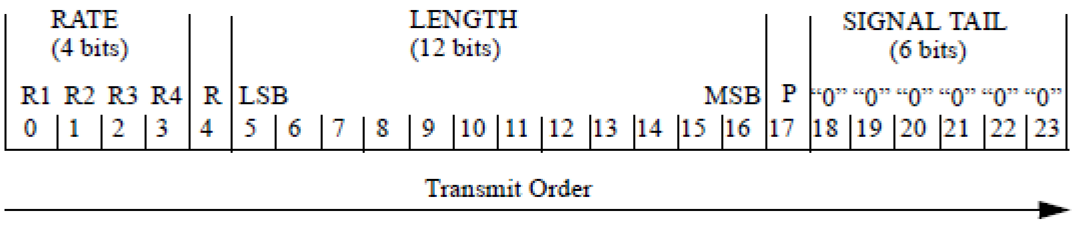


Non-HT (Legacy) Signal Field

The legacy signal (L-SIG) field is part of the legacy preamble. It consists of 24 bits that contain rate, length, and parity information. The L-SIG is a component of VHT, HT, and non-HT PPDUs. It is transmitted using BPSK modulation with rate 1/2 binary convolutional coding (BCC).



The L-SIG consists of one 4 µs symbol, which is a 3.2 µs OFDM symbol prepended with a 0.8 µs cyclic prefix. It contains packet information for the received configuration,



- Bits 0 through 3 specify the data rate (modulation and coding rate) for the non-HT format.

Rate (bits 0–3)	Modulation	Coding Rate (R)	Data Rate (20 MHz BW) (Mb/s)
1101	BPSK	1/2	6
1111	BPSK	3/4	9
0101	QPSK	1/2	12
0111	QPSK	3/4	18
1001	16-QAM	1/2	24
1011	16-QAM	3/4	36
0001	64-QAM	2/3	48
0011	64-QAM	3/4	54

For HT and VHT formats, the L-SIG rate bits are set to `'1 1 0 1'`. Data rate information for HT and VHT formats is signaled in format-specific signaling fields.

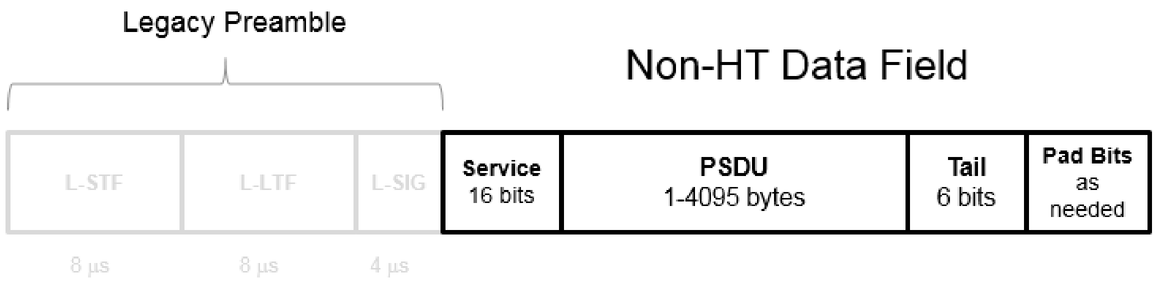
- Bit 4 is reserved for future use.
- Bits 5 through 16:
  - For non-HT, specify the data length (amount of data transmitted in octets) as described in IEEE Std 802.11-2012, Table 18-1 and Section 9.23.4.
  - For HT-mixed, specify the transmission time as described in IEEE Std 802.11-2012, Section 20.3.9.3.5 and Section 9.23.4.
  - For VHT, specify the transmission time as described in IEEE Std 802.11ac-2013, Section 22.3.8.2.4.
- Bit 17 has the even parity of bits 0 through 16.
- Bits 18 through 23 contain all zeros for the signal tail bits.

Note: Signaling fields added for HT (`wlanHTSIG`) and VHT (`wlanVHTSIGA`, `wlanVHTSIGB`) formats provide data rate and configuration information for those formats.

- IEEE Std 802.11-2012, Section 20.3.9.4.3 describes HT-SIG bit settings for the HT-mixed format.
- IEEE Std 802.11ac-2013, Section 22.3.8.3.3 and Section 22.3.8.3.6 describe bit settings for VHT-SIG-A and VHT-SIG-B, respectively, for the VHT format.

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.

Processing of an 802.11a™ data field is defined in IEEE 802.11-2012 [2], Section 18.3.5.

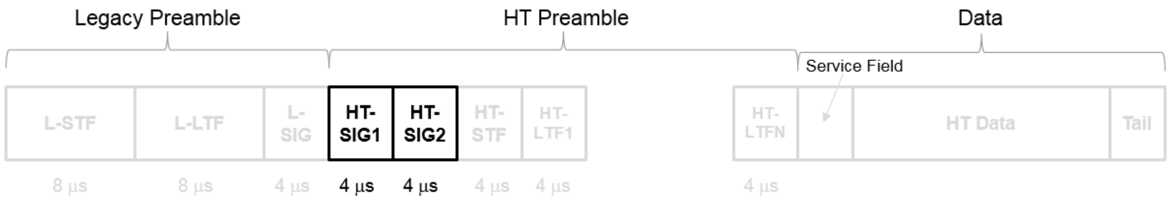
The six tail bits are set to zero after a 127-bit scrambling sequence has been applied to the full data field. The receiver uses the first seven bits of the service field to determine the initial state of the scrambler. Rate 1/2 BCC encoding is performed on the scrambled data. The zeroed tail bits cause the BCC encoder to return to a zero state. Puncturing is applied as needed for the selected rate.

The coded data is grouped into several bits per symbol, and two permutations of block interleaving are applied to each group of data. The groups of bits are then modulated to the selected rate (BPSK, QPSK, 16-QAM, or 64-QAM) and the complex symbols are then mapped onto corresponding subcarriers. For each symbol, the pilot subcarriers are inserted. An IFFT is used to transform each symbol group to the time domain and the cyclic prefix is prepended.

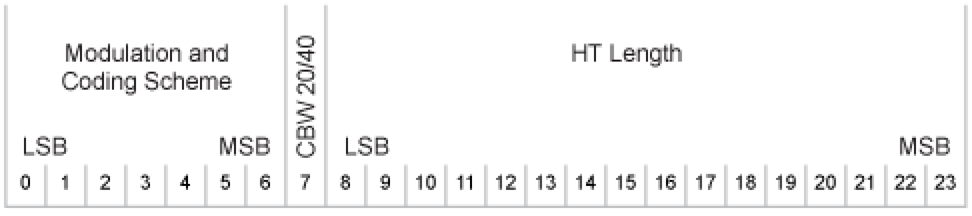
The final processing preceding DAC up-conversion to RF and the power amplifier is to apply a pulse shaping filter on the data to smooth transitions between symbols. The standard provides an example pulse shaping function but does not specifically require one.

High Throughput Signal Field

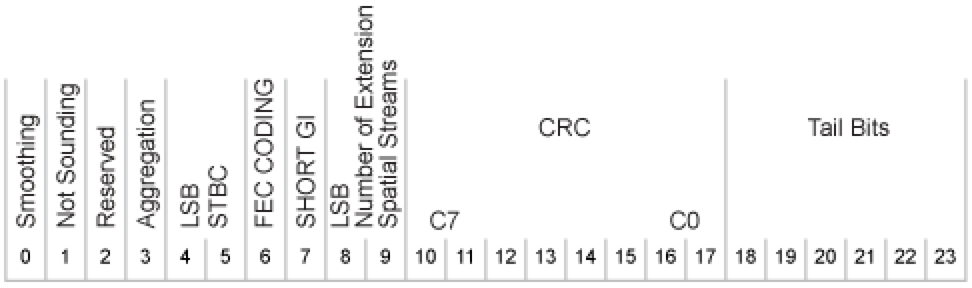
The high throughput signal (HT-SIG) field is located between the L-SIG field and HT-STF and is part of the HT-mixed format preamble. It is composed of two symbols, HT-SIG$_1$ and HT-SIG$_2$.



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$_1$

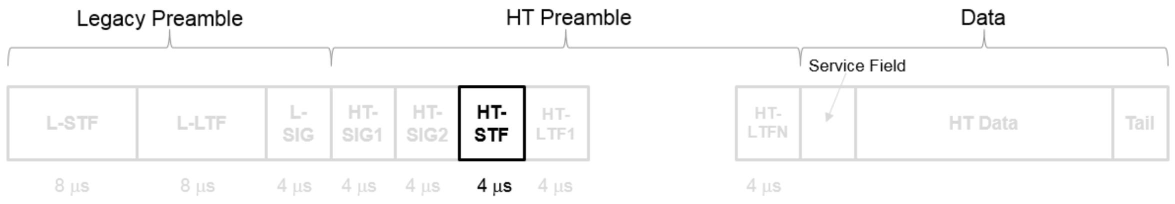


HT-SIG$_2$

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

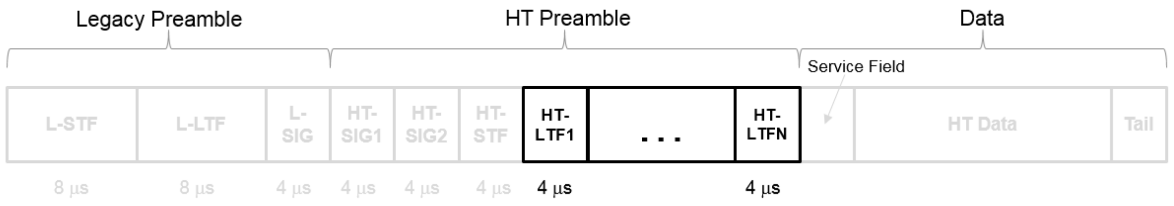
High Throughput Short Training Field

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.



High Throughput Long Training Fields

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	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

N_{STS} Determination	N_{HTDLTF} Determination	N_{HTELTF} Determination
number of spatial streams (N_{SS}) from the MCS and the STBC field.		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

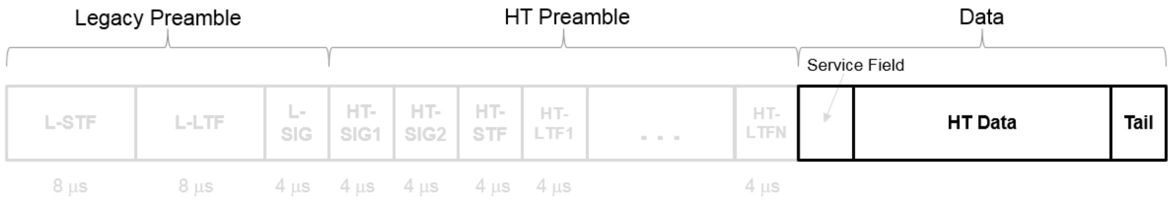
N_{ESS}	N_{HTELTF}
0	0
1	1
2	2
3	4

Additional constraints include:

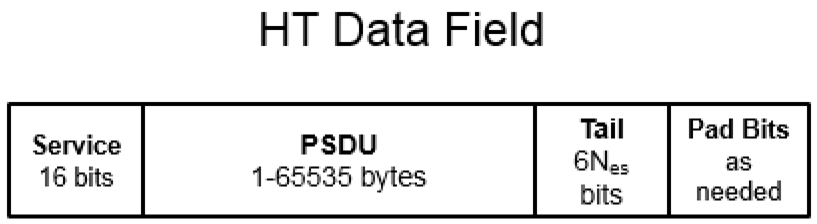
- $N_{HTLTF} = N_{HTDLTF} + N_{HTELTF} \leq 5$.
- $N_{STS} + N_{ESS} \leq 4$.
  - When $N_{STS} = 3$, N_{ESS} cannot exceed one.
  - If $N_{ESS} = 1$ when $N_{STS} = 3$ then $N_{HTLTF} = 5$.

HT 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.

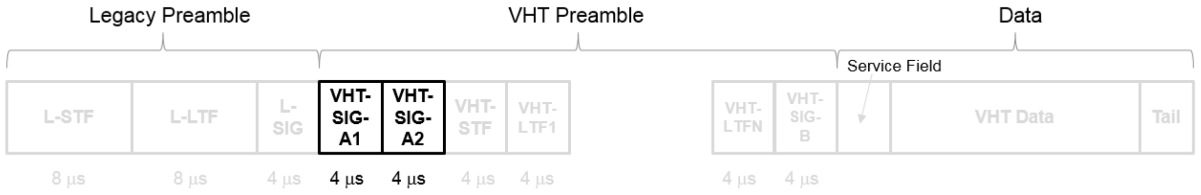


- **Service field** — Contains 16 zeros to initialize the data scrambler.
- **PSDU** — Variable-length field containing the PLCP service data unit (PSDU). In 802.11, the PSDU can consist of an aggregate of several MAC service data units.
- **Tail** — Tail bits required to terminate a convolutional code. 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.

Very High Throughput SIG-A Field

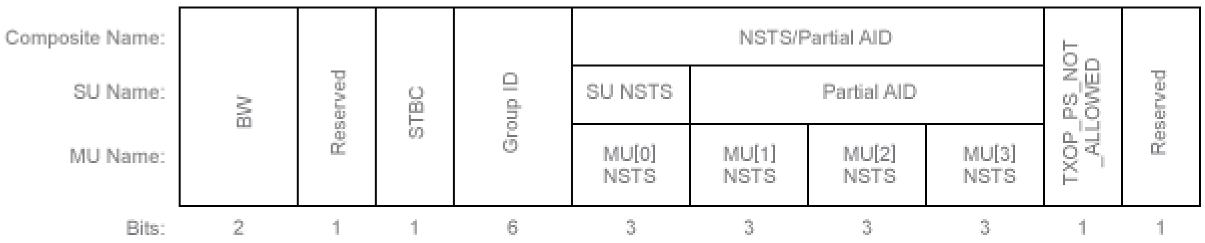
The very high throughput signal A (VHT-SIG-A) field contains information required to interpret VHT format packets. Similar to the non-HT signal (L-SIG) field for the non-HT OFDM format, this field stores the actual rate value, channel coding, guard interval, MIMO scheme, and other configuration details for the VHT format packet. Unlike the HT-SIG field, this field does not store the packet length information. Packet length information is derived from L-SIG and is captured in the VHT-SIG-B field for the VHT format.

The VHT-SIG-A field consists of two symbols: VHT-SIG-A1 and VHT-SIG-A2. These symbols are located between the L-SIG and the VHT-STF portion of the VHT format PPDU.

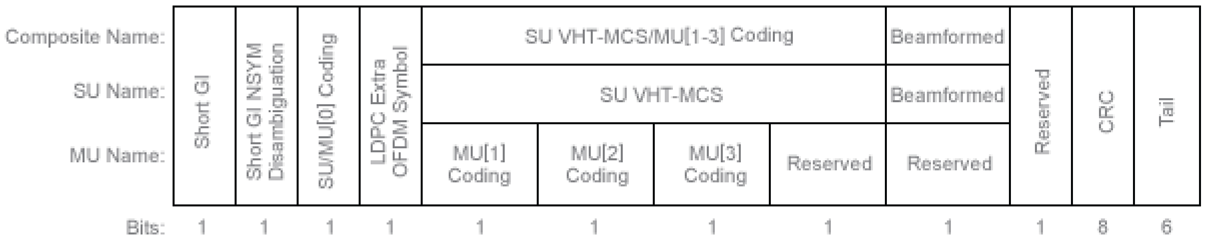


The VHT-SIG-A field is defined in IEEE Std 802.11ac-2013, Section 22.3.8.3.3.

VHT-SIG-A1 Structure



VHT-SIG-A2 Structure



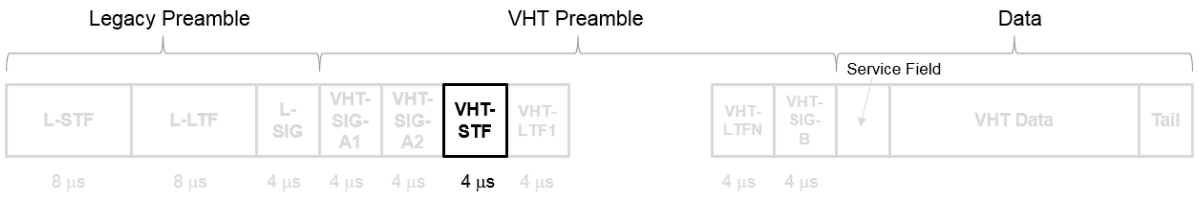
The VHT-SIG-A field includes these components. The bit field structures for VHT-SIG-A1 and VHT-SIG-A2 vary for single user or multi-user transmissions.

- **BW** — A two-bit field that indicates 0 for 20 MHz, 1 for 40 MHz, 2 for 80 MHz, or 3 for 160 MHz.
- **STBC** — A bit that indicates the presence of space-time block coding.
- **Group ID** — A six-bit field that indicates the group and user position assigned to a STA.
- **N_{STS}** — A three-bit field for a single user or 4 three-bit fields for a multi-user scenario, that indicates the number of space-time streams per user.
- **Partial AID** — An identifier that combines the association ID and the BSSID.
- **TXOP_PS_NOT_ALLOWED** — An indicator bit that shows if client devices are allowed to enter dose state. This bit is set to false when the VHT-SIG-A structure is populated, indicating that the client device is allowed to enter dose state.

- **Short GI** — A bit that indicates use of the 400 ns guard interval.
- **Short GI NSYM Disambiguation** — A bit that indicates if an extra symbol is required when the short GI is used.
- **SU/MU[0] Coding** — A bit field that indicates if convolutional or LDPC coding is used for a single user or for user MU[0] in a multi-user scenario.
- **LDPC Extra OFDM Symbol** — A bit that indicates if an extra OFDM symbol is required to transmit the data field.
- **MCS** — A four-bit field.
  - For a single user scenario, it indicates the modulation and coding scheme used.
  - For a multi-user scenario, it indicates use of convolutional or LDPC coding and the MCS setting is conveyed in the VHT-SIG-B field.
- **Beamformed** — An indicator bit set to 1 when a beamforming matrix is applied to the transmission.
- **CRC** — An eight-bit field used to detect errors in the VHT-SIG-A transmission.
- **Tail** — A six-bit field used to terminate the convolutional code.

Very High Throughput Short Training Field

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.

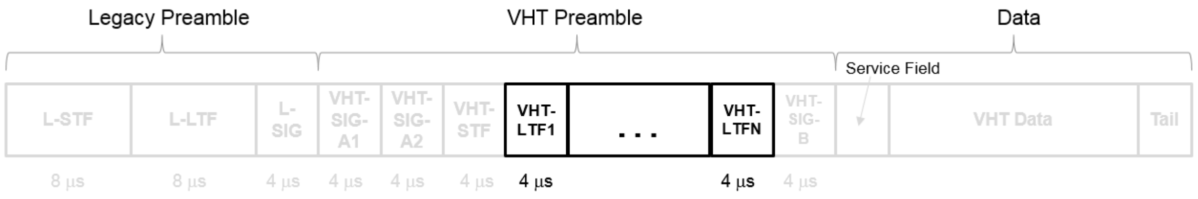


The frequency domain sequence used to construct the VHT-STF for a 20 MHz transmission is identical to the L-STF sequence. Duplicate L-STF sequences are frequency shifted and phase rotated to support VHT transmissions for the 40 MHz, 80 MHz, and 160 MHz channel bandwidths. As such, the L-STF and HT-STF are subsets of the VHT-STF.

The VHT-STF is defined in IEEE Std 802.11ac-2013, Section 22.3.8.3.4.

Very High Throughput Long Training Fields

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



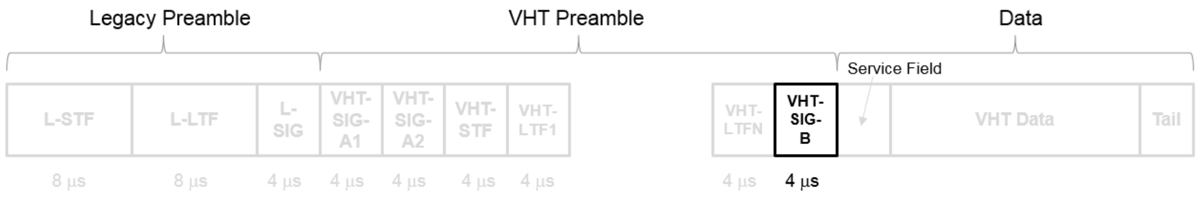
It is used for MIMO channel estimation and pilot subcarrier tracking. The VHT-LTF includes one VHT long training symbol for each spatial stream indicated by the selected MCS. Each symbol is 4 µs long. A maximum of eight symbols are permitted in the VHT-LTF.

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

Very High Throughput SIG-B Field

The very high throughput signal B field (VHT-SIG-B) is used for multi-user scenario to set up the data rate and to fine-tune MIMO reception. It is modulated using MCS 0 and is transmitted in a single OFDM symbol.

The VHT-SIG-B field consists of a single OFDM symbol located between the VHT-LTF and the data portion of the VHT format PPDU.



The very high throughput signal B (VHT-SIG-B) field contains the actual rate and A-MPDU length value per user. The VHT-SIG-B is defined in IEEE Std 802.11ac-2013, Section 22.3.8.3.6, and Table 22–14. The number of bits in the VHT-SIG-B field varies with the channel bandwidth and the assignment depends on whether single user or multi-user scenario in allocated. For single user configurations, the same information is available in the L-SIG field but the VHT-SIG-B field is included for continuity purposes.

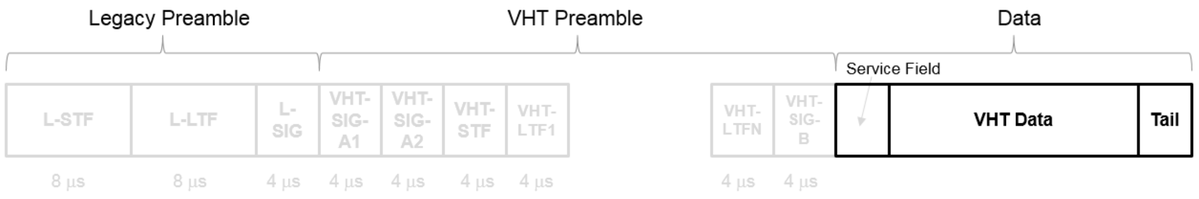
Field	VHT MU PPDU Allocation (bits)			VHT SU PPDU Allocation (bits)			Description
	20 MHz	40 MHz	80 MHz, 160 MHz	20 MHz	40 MHz	80 MHz, 160 MHz	
VHT-SIG-B	B0-15 (16)	B0-16 (17)	B0-18 (19)	B0-16 (17)	B0-18 (19)	B0-20 (21)	A variable-length field that indicates the size of the data payload in four-byte units. The length of the field depends on the channel bandwidth.
VHT-MCS	B16-19 (4)	B17-20 (4)	B19-22 (4)	N/A	N/A	N/A	A four-bit field that is included for multi-user scenarios only.
Reserved	N/A	N/A	N/A	B17–19 (3)	B19-20 (2)	B21-22 (2)	All ones
Tail	B20-25 (6)	B21-26 (6)	B23-28 (6)	B20-25 (6)	B21-26 (6)	B23-28 (6)	Six zero-bits

Field	VHT MU PPDU Allocation (bits)			VHT SU PPDU Allocation (bits)			Description
	20 MHz	40 MHz	80 MHz, 160 MHz	20 MHz	40 MHz	80 MHz, 160 MHz	
							used to terminate the convolutional code.
Total # bits	26	27	29	26	27	29	
Bit field 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.*	

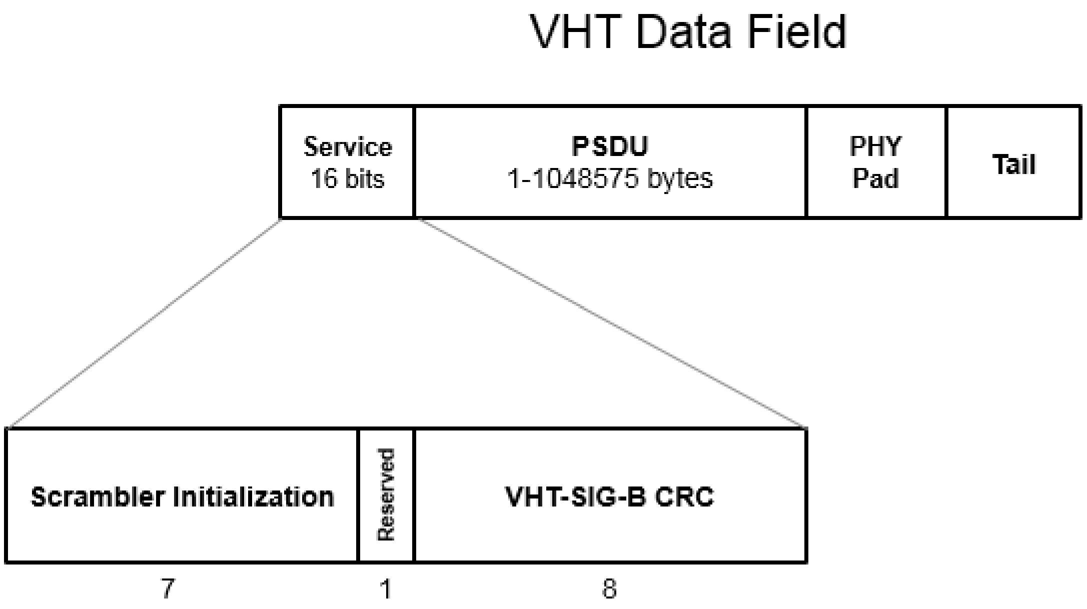
For a null data packet (NDP), the VHT-SIG-B bits are set according to IEEE Std 802.11ac-2013, Table 22-15.

VHT 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.

See Also

"Waveform Generation" | "What Is WLAN?"

Build VHT PPDU

This example shows how to build VHT PPDUs by using the waveform generator function or by building each field individually.

Waveform Generator

Create a VHT configuration object and a waveform generator object.

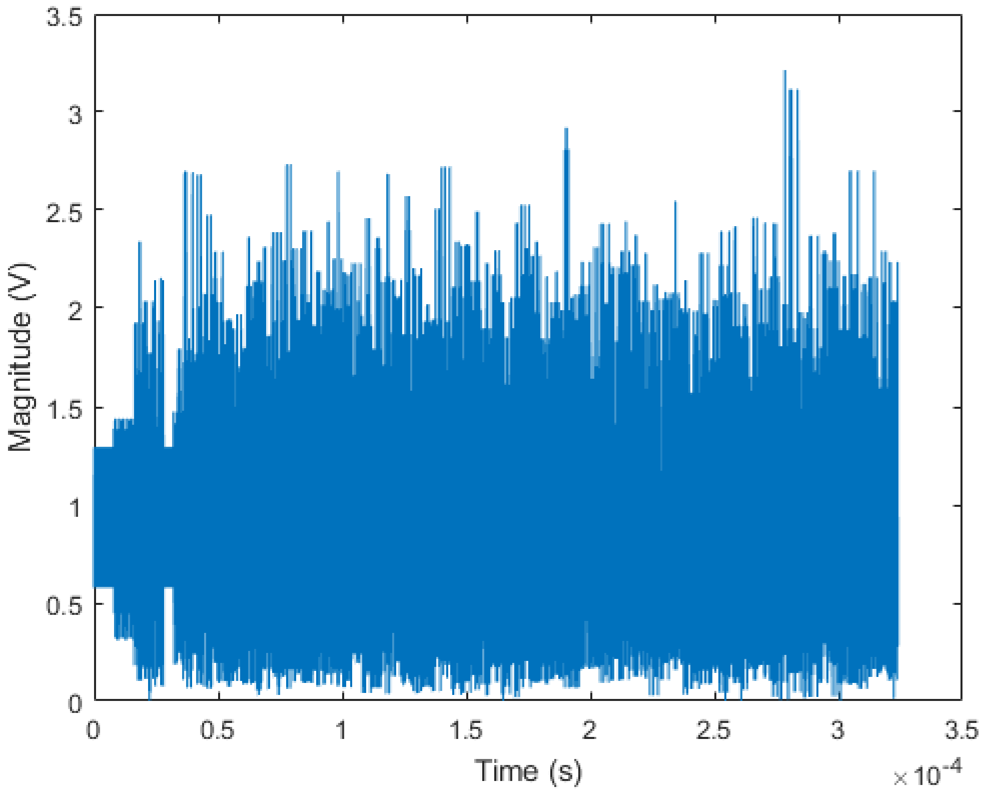
```
vht = wlanVHTConfig;
wgc = wlanGeneratorConfig('Windowing',false);
```

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

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

Plot the magnitude of the waveform.

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



Individual PPDU Fields

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

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

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

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

Concatenate the individual fields to create a single PPDU.

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

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

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

```
ans =

     1
```

Related examples:

- "Build HT PPDU"
- "Build Non-HT PPDU"

Build HT PPDU

This example shows how to build HT PPDUs by using the waveform generator function or by building each field individually.

Waveform Generator

Create an HT configuration object and a waveform generator object.

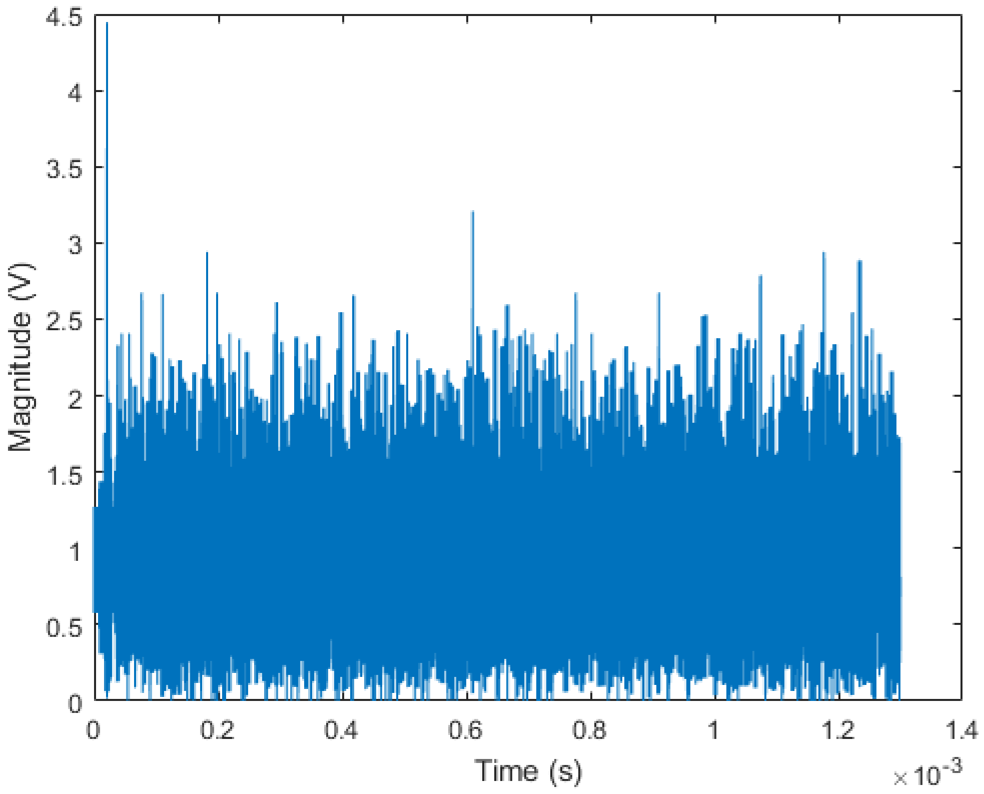
```
ht = wlanHTConfig;
wgc = wlanGeneratorConfig('Windowing',false);
```

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

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

Plot the magnitude of the waveform.

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



Individual PPDU Fields

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

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

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

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

Concatenate the individual fields to create a single PPDU.

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

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

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

```
ans =

     1
```

Related examples:

- “Build VHT PPDU”
- “Build Non-HT PPDU”

Build Non-HT PPDU

This example shows how to build non-HT PPDUs by using the waveform generator function or by building each field individually.

Waveform Generator

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

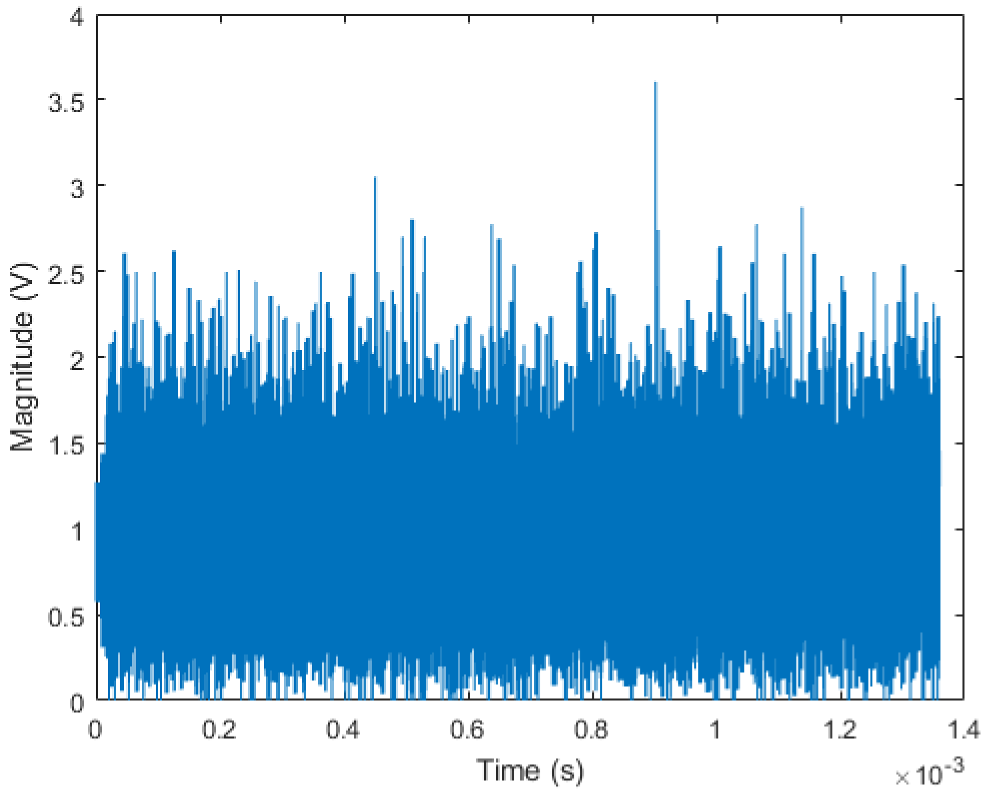
```
nht = wlanNonHTConfig;
wgc = wlanGeneratorConfig('Windowing',false);
```

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

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

Plot the magnitude of the waveform.

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



Individual PPDU Fields

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

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

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

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

Concatenate the individual fields to create a single PPDU.

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

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

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

```
ans =

     1
```

Related examples:

- “Build VHT PPDU”
- “Build HT PPDU”

Transmit and Recover L-SIG, VHT-SIG-A, VHT-SIG-B in Fading Channel

This example shows how to transmit a VHT waveform through a noisy MIMO channel. It also shows how to extract the L-SIG, VHT-SIG-A, and VHT-SIG-B fields and verify that they were correctly recovered.

Set the parameters used throughout the example.

```
cbw = 'CBW40';                      % Channel bandwidth
fs = 40e6;                          % Sample rate (Hz)
ntx = 2;                            % Number of transmit antennas
nsts = 2;                           % Number of space-time streams
nrx = 3;                            % Number of receive antennas
```

Create a VHT configuration object that supports a 2x2 MIMO transmission and has an APEP length of 2000. Create a generator configuration object.

```
vht = wlanVHTConfig('ChannelBandwidth',cbw,'APEPLength',2000, ...
    'NumTransmitAntennas',ntx,'NumSpaceTimeStreams',nsts, ...
    'SpatialMapping','Direct','STBC',false);
wgc = wlanGeneratorConfig;
```

Generate a VHT waveform containing a random PSDU.

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

Create a 2x2 TGac channel and an AWGN channel.

```
chTG = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',cbw, ...
    'NumTransmitAntennas',ntx,'NumReceiveAntennas',nrx, ...
    'LargeScaleFadingEffect','Pathloss and shadowing', ...
    'DelayProfile','Model-C');
chAWGN = comm.AWGNChannel('NoiseMethod','Variance', ...
    'VarianceSource','Input port');
```

Calculate the noise variance for a receiver with a 9 dB noise figure. The noise variance is equal to $kTBF$, where k is Boltzmann's constant, T is the ambient temperature, B is the bandwidth (sample rate), and F is the receiver noise figure. Pass the transmitted waveform through the noisy TGac channel.

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

```
rxPPDU = step(chAWGN, step(chTG,txPPDU), nVar);
```

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

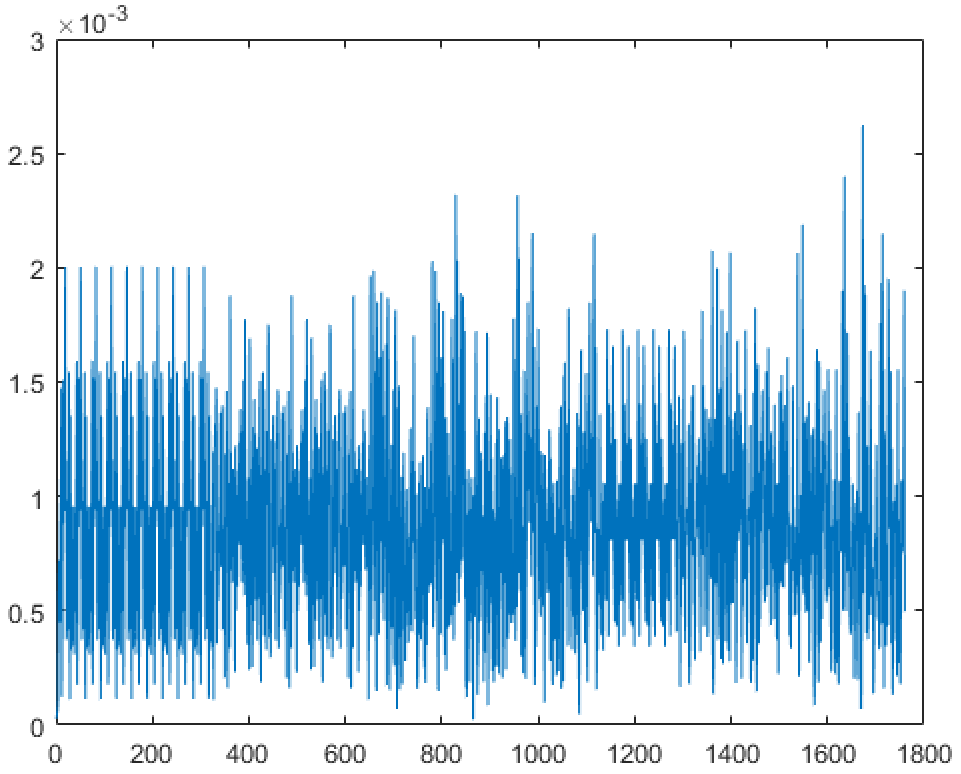
```
ind = wlanFieldIndices(vht)
```

```
ind = 

       LSTF: [1 320]
       LLTF: [321 640]
       LSIG: [641 800]
    VHTSIGA: [801 1120]
     VHTSTF: [1121 1280]
     VHTLTF: [1281 1600]
    VHTSIGB: [1601 1760]
    VHTData: [1761 25600]
```

The preamble is contained in the first 1760 symbols. Plot the preamble.

```
plot(abs(rxPPDU(1:1760)))
```



Extract the L-LTF from the received PPDU using the start and stop indices determined by the `wlanFieldIndices` function. Demodulate the L-LTF and estimate the channel coefficients.

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

Extract the L-SIG field from the received PPDU and recover its information bits.

```
rxLSIG = rxPPDU(ind.LSIG(1):ind.LSIG(2),:);
infoLSIG = wlanLSIGRecover(rxLSIG,chEstLLTF,nVar,cbw);
```

Inspect the L-SIG rate information and confirm that the sequence `[1 1 0 1]` is received. This sequence corresponds to a 6 MHz data rate, which is used for all VHT transmissions.

```
rate = infoLSIG(1:4)'
```

```
rate =

     1     1     0     1
```

Extract the VHT-SIG-A and confirm that the CRC check passed.

```
rxVHTSIGA = rxPPDU(ind.VHTSIGA(1):ind.VHTSIGA(2),:);
[infoVHTSIGA,failCRC] = wlanVHTSIGARecover(rxVHTSIGA, ...
    chEstLLTF,nVar,cbw);
failCRC
```

```
failCRC =

     0
```

Extract and demodulate the VHT-LTF. Use the demodulated signal to estimate the channel coefficients needed to recover the VHT-SIG-B field.

```
rxVHTLTF = rxPPDU(ind.VHTLTF(1):ind.VHTLTF(2),:);
demodVHTLTF = wlanVHTLTFDemodulate(rxVHTLTF,vht);
chEstVHTLTF = wlanVHTLTFChannelEstimate(demodVHTLTF,vht);
```

Extract and recover the VHT-SIG-B.

```
rxVHTSIGB = rxPPDU(ind.VHTSIGB(1):ind.VHTSIGB(2),:);
infoVHTSIGB = wlanVHTSIGBRecover(rxVHTSIGB,chEstVHTLTF,nVar,cbw);
```

Verify that the APEP length, contained in the first 19 bits of the VHT-SIG-B, corresponds to the specified length of 2000 bits.

```
pktLbits = infoVHTSIGB(1:19)';
pktLen = bi2de(double(pktLbits))*4
```

```
pktLen =
```

2000

End-to-End VHT Simulation with Frequency Correction

This example shows how to:

- Transmit a VHT waveform through a MIMO channel with AWGN
- Perform a two-stage process to estimate and correct for a frequency offset
- Estimate the channel response
- Recover the VHT data field
- Compare the transmitted and received PSDUs to determine if bit errors occurred

Set the parameters used throughout the example.

```
cbw = 'CBW160';                   % Channel bandwidth
fs = 160e6;                       % Sample rate (Hz)
ntx = 2;                          % Number of transmit antennas
nsts = 2;                         % Number of space-time streams
nrx = 2;                          % Number of receive antennas
```

Create a VHT configuration object that supports a 2x2 MIMO transmission and has an APEP length of 2000. Create a generator configuration object.

```
vht = wlanVHTConfig('ChannelBandwidth',cbw,'APEPLength',2000, ...
    'NumTransmitAntennas',ntx,'NumSpaceTimeStreams',nsts, ...
    'SpatialMapping','Direct','STBC',false);
wgc = wlanGeneratorConfig;
```

Generate a VHT waveform containing a random PSDU.

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

Create a 2x2 TGac channel and an AWGN channel.

```
chTG = wlanTGacChannel('SampleRate',fs,'ChannelBandwidth',cbw, ...
    'NumTransmitAntennas',ntx,'NumReceiveAntennas',nrx, ...
    'LargeScaleFadingEffect','Pathloss and shadowing', ...
    'DelayProfile','Model-C');
chAWGN = comm.AWGNChannel('NoiseMethod','Variance', ...
    'VarianceSource','Input port');
```

Create a phase/frequency offset object.

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

Calculate the noise variance for a receiver with a 9 dB noise figure. Pass the transmitted waveform through the noisy TGac channel.

```
nVar = 10^((-228.6 + 10*log10(290) + 10*log10(fs) + 9)/10);
rxPPDU = step(chAWGN, step(chTG,txPPDU), nVar);
```

Introduce a frequency offset of 500 Hz.

```
rxPPDUcfo = step(pfo,rxPPDU,500);
```

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

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

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

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

foffset1 = wlanCoarseCFOEstimate(rxLSTF,cbw);
rxPPDUcorr = step(pfo,rxPPDUcfo,-foffset1);
```

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

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

foffset2 = wlanFineCFOEstimate(rxLLTF,cbw);
rxPPDU2 = step(pfo,rxPPDUcorr,-foffset2);
```

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

```
rxVHTLTF = rxPPDU2(ind.VHTLTF(1):ind.VHTLTF(2),:);
dLTF = wlanVHTLTFDemodulate(rxVHTLTF,vht);
chEst = wlanVHTLTFChannelEstimate(dLTF,vht);
```

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

```
rxVHTData = rxPPDU2(ind.VHTData(1):ind.VHTData(2),:);
rxPSDU = wlanVHTDataRecover(rxVHTData,chEst,nVar,vht);
```

Calculate the number of bit errors in the received packet.

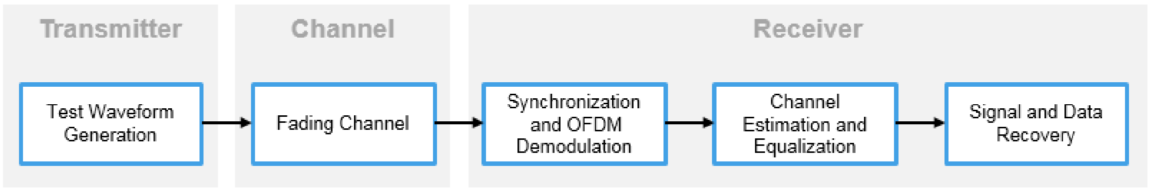
```
numErr = biterr(txPSDU,rxPSDU)
numErr =

     0
```

Transmit-Receive Chain

In this section...
"Transmit Processing Chain" on page 1-39
"Receiver Processing Chain" on page 1-44

WLAN System Toolbox functionality includes elements of a standard transmitter–channel–receiver processing chain.



- Transmitter functions enable simulation of the various IEEE 802.11 [3][4] formats. The simulated waveform includes preamble and data fields of the PPDU. You can use this waveform in link-level simulations. You can also use it as a test signal for test devices and equipment.
- Channel functions model various types of AWGN, fading, or moving channel environmental effects.
- Receiver functions recover the transmitted signal.

Transmit Processing Chain

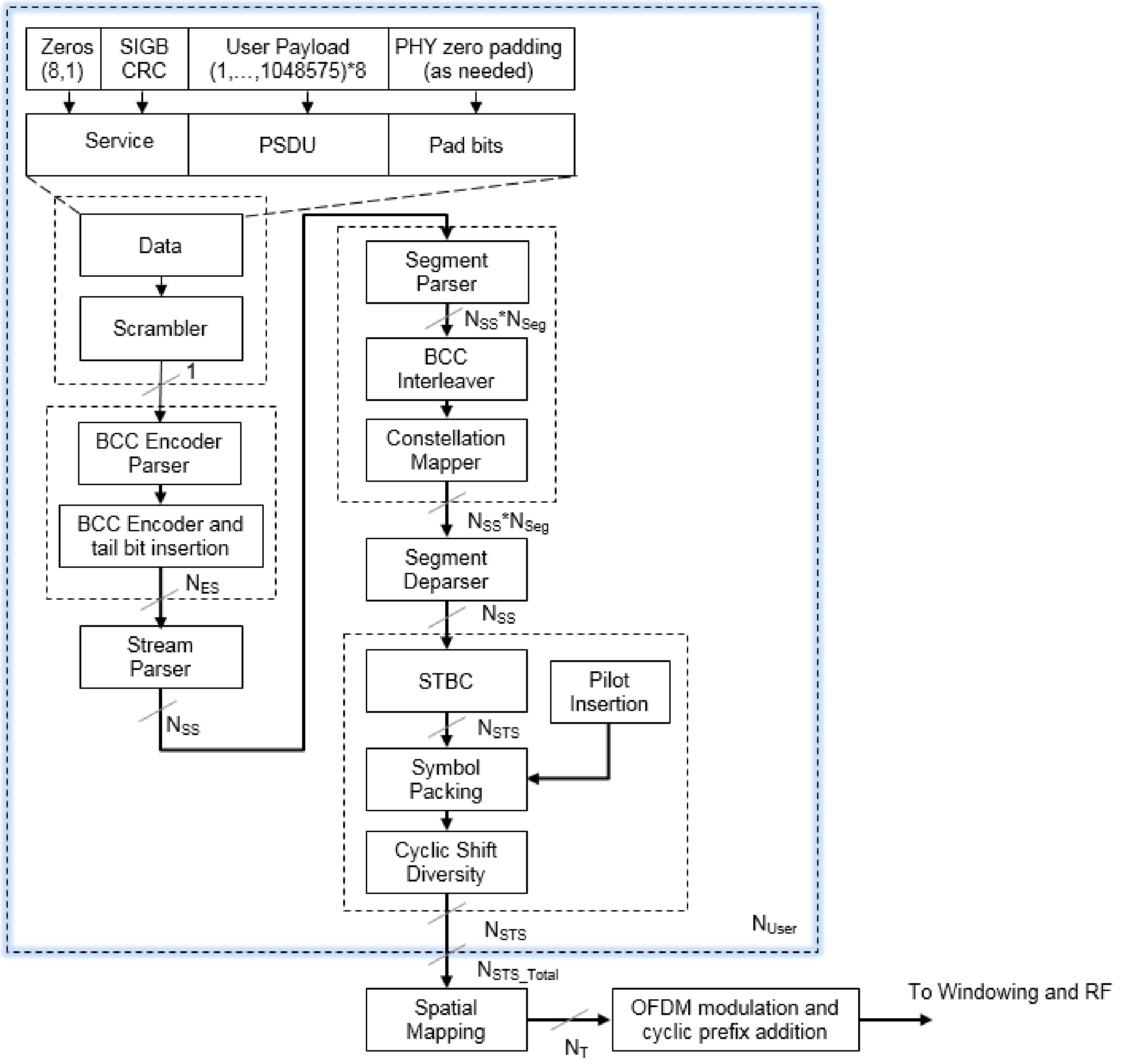
WLAN System Toolbox functions enable you to generate waveforms for a complete PPDU or for the individual fields of VHT, HT-mixed, and non-HT format PPDUs.

VHT Data Transmit Processing Chain

As described in IEEE 802.11ac-2013 [3], Section 22 specifies the PHY entity for a very high throughput (VHT) orthogonal frequency division multiplexing (OFDM) system. A

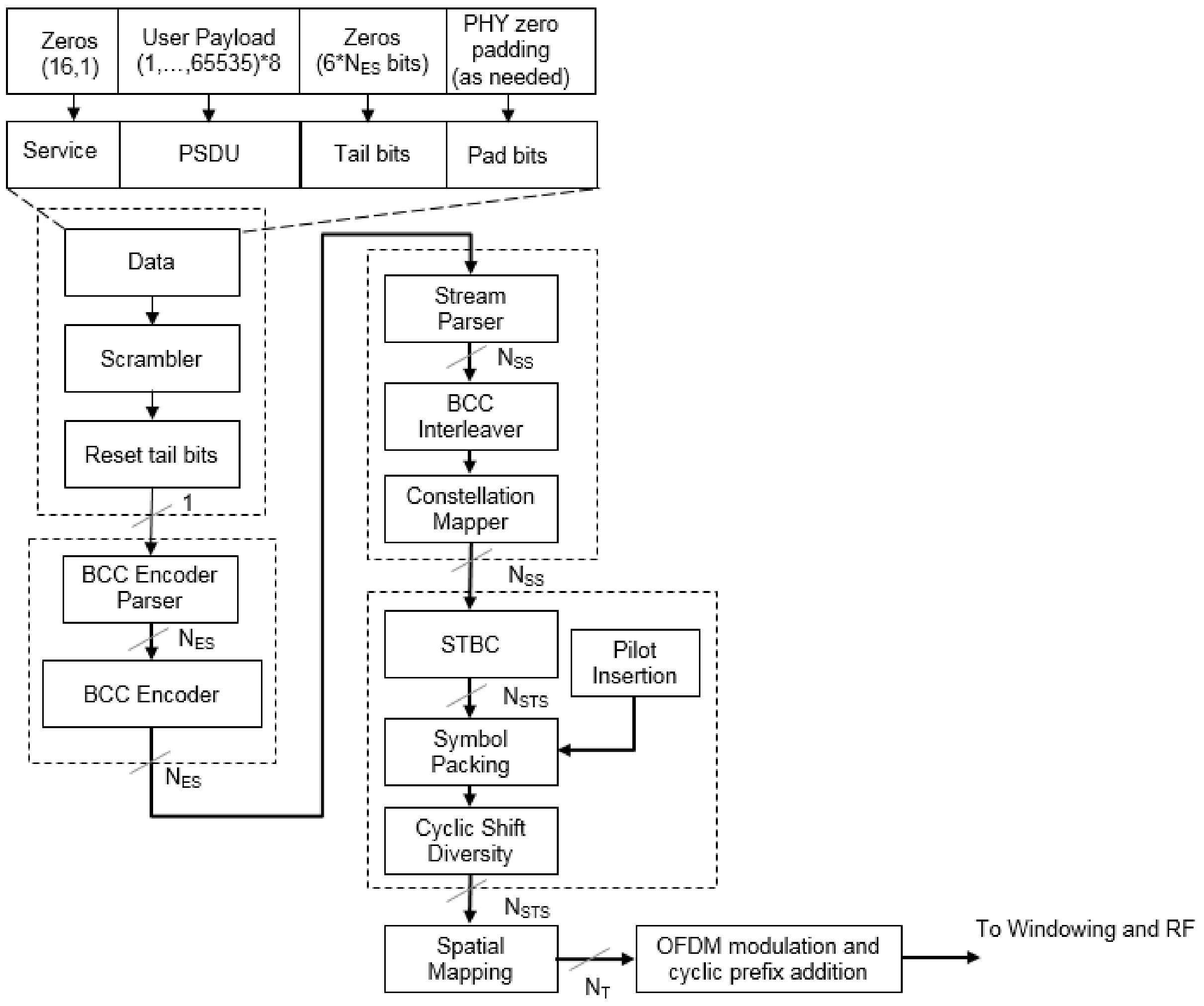
3. IEEE Std 802.11-2012 Adapted and reprinted with permission from IEEE. Copyright IEEE 2012. All rights reserved.
4. IEEE Std 802.11ac-2013 Adapted and reprinted with permission from IEEE. Copyright IEEE 2013. All rights reserved.

VHT STA must be capable of transmitting and receiving HT-PHY and non-HT-PHY-compliant PPDUs. Specifically, the VHT PHY is based on the HT PHY defined in Section 20, which in turn is based on the OFDM PHY defined in Section 18. The VHT PHY extends the maximum number of space-time streams supported to eight and provides support for downlink multi-user (MU) transmissions. A downlink MU transmission supports up to four users with up to four space-time streams per user, with the total number of space-time streams not exceeding eight.



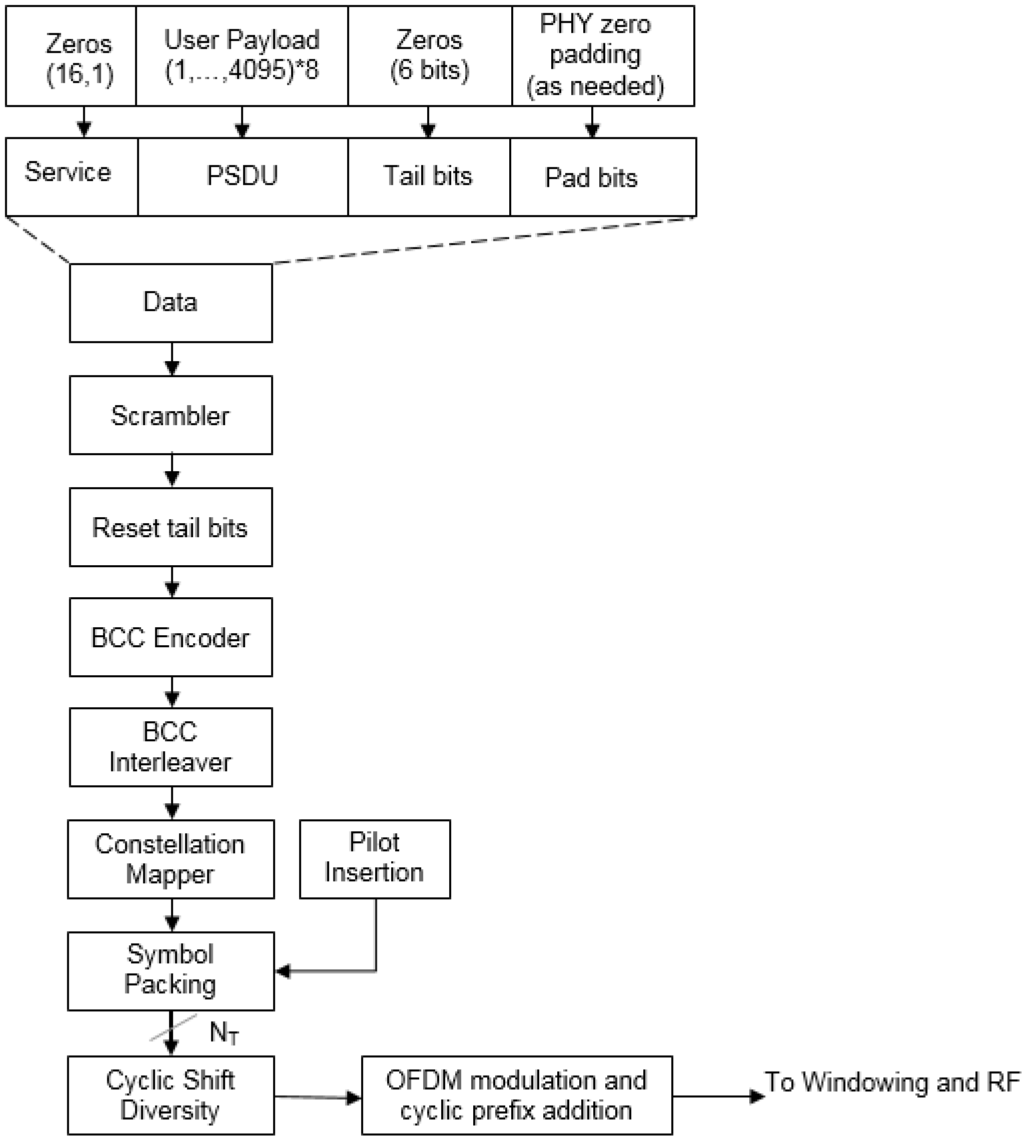
IEEE Std 802.11ac-2013 [3], Section 22 defines requirements for physical layer processing associated with each PPDU field for the VHT format.

HT Data Transmit Processing Chain



IEEE 802.11-2012 [2], Section 20 defines requirements for physical layer processing associated with each PPDU field for the HT-mixed format.

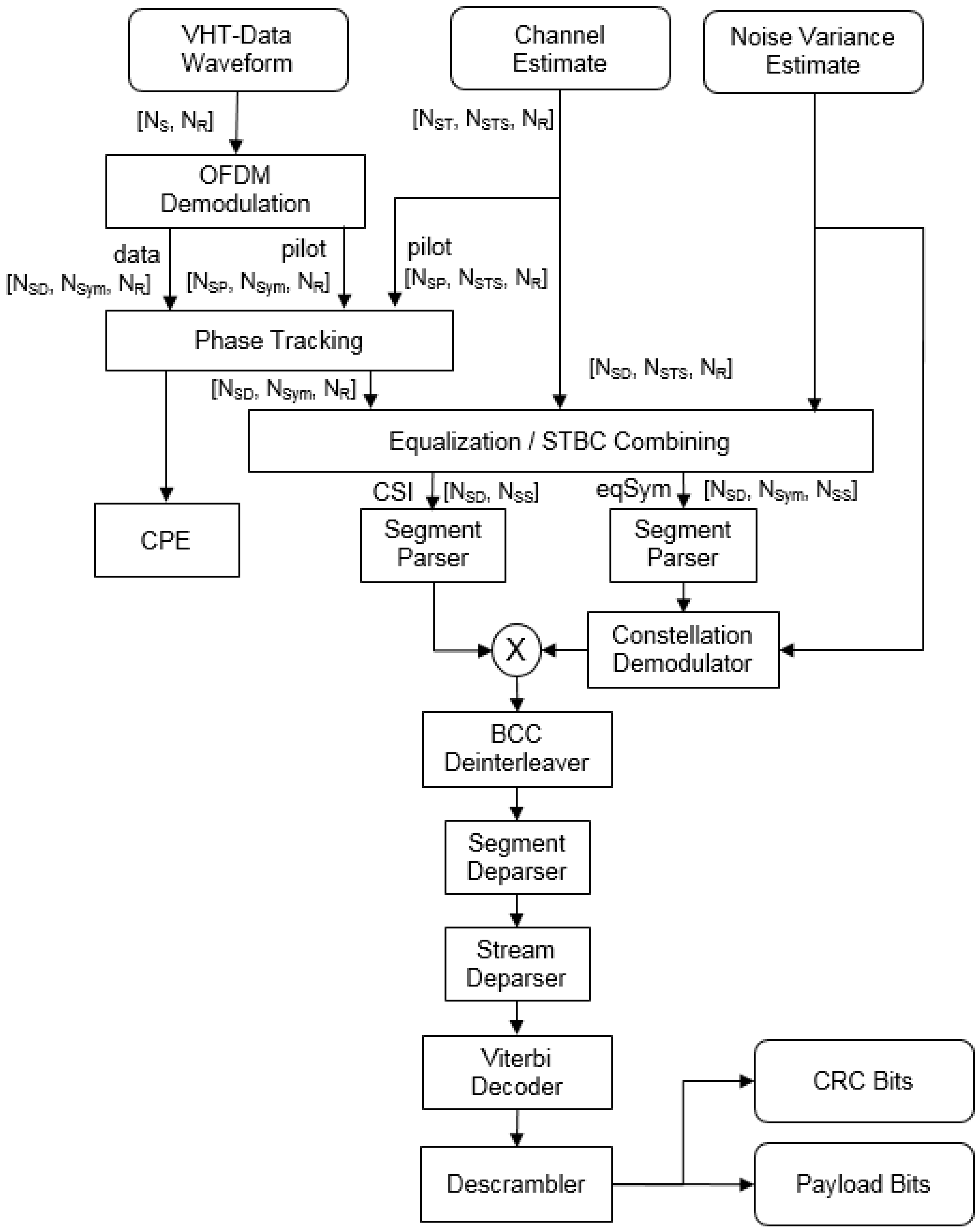
Non-HT Data Transmit Processing Chain



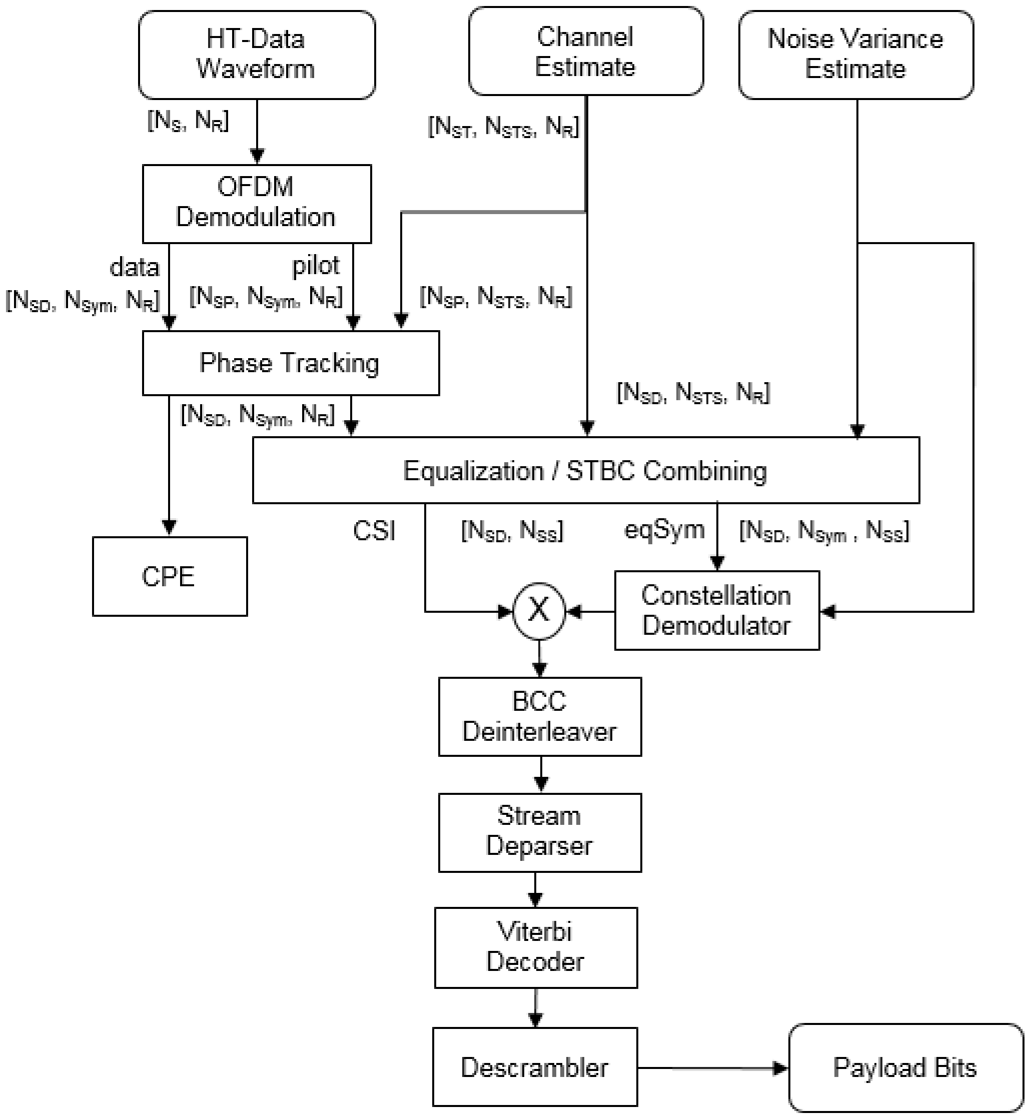
IEEE 802.11-2012 [2], Section 18 defines requirements for physical layer processing associated with each PPDU field for the OFDM modulation scheme. IEEE 802.11-2012 [2], Section 17, and Section 19 define requirements for physical layer processing associated with each PPDU field for the DSSS modulation scheme.

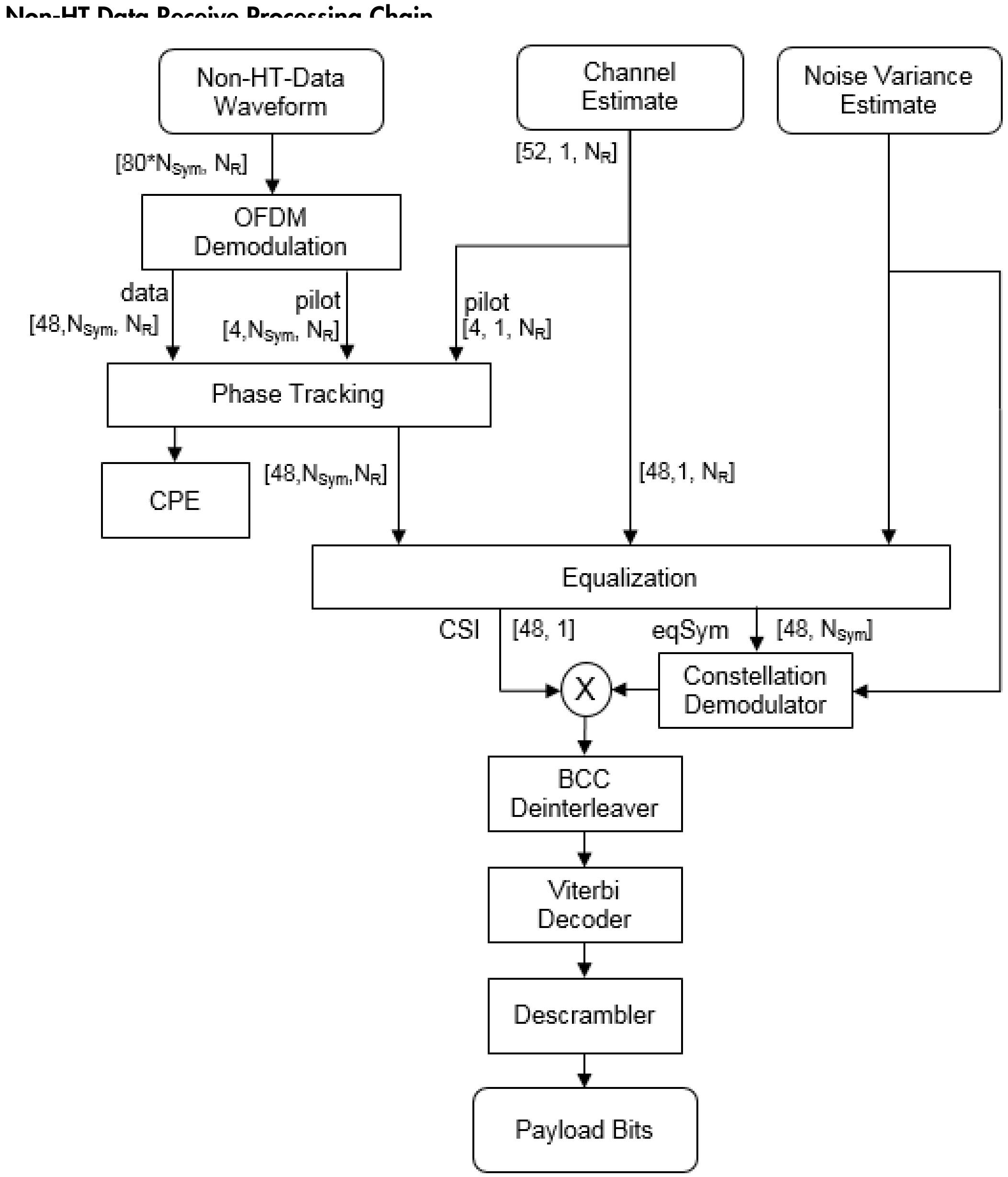
Receiver Processing Chain

WLAN System Toolbox functions enable you to recover transmitted VHT, HT-mixed, and non-HT format PPDUs. The receive processing chain includes synchronization, OFDM demodulation, channel estimation, equalization, and signal and data recovery.


VHT-Data Waveform
Channel Estimate
Noise Variance Estimate
$[N_S, N_R]$
$[N_{ST}, N_{STS}, N_R]$
OFDM Demodulation
data
$[N_{SD}, N_{Sym}, N_R]$
pilot
$[N_{SP}, N_{Sym}, N_R]$
pilot
$[N_{SP}, N_{STS}, N_R]$
Phase Tracking
$[N_{SD}, N_{Sym}, N_R]$
$[N_{SD}, N_{STS}, N_R]$
Equalization / STBC Combining
CPE
CSI
$[N_{SD}, N_{SS}]$
eqSym
$[N_{SD}, N_{Sym}, N_{SS}]$
Segment Parser
Segment Parser
X
Constellation Demodulator
BCC Deinterleaver
Segment Deparser
Stream Deparser
Viterbi Decoder
Descrambler
CRC Bits
Payload Bits

HT Data Receive Processing Chain




Non-HT Data Receive Processing Chain
Non-HT-Data Waveform
Channel Estimate
Noise Variance Estimate
[80*N_Sym, N_R]
[52, 1, N_R]
OFDM Demodulation
data
[48,N_Sym, N_R]
pilot
[4,N_Sym, N_R]
pilot
[4, 1, N_R]
Phase Tracking
CPE
[48,N_Sym,N_R]
[48,1, N_R]
Equalization
CSI
[48, 1]
eqSym
[48, N_Sym]
X
Constellation Demodulator
BCC Deinterleaver
Viterbi Decoder
Descrambler
Payload Bits

References

[1] IEEE 802.11™: Wireless LANs. http://standards.ieee.org/about/get/802/802.11.html

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

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

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

See Also

“Transmit and Recover L-SIG, VHT-SIG-A, VHT-SIG-B in Fading Channel” on page 1-31 | “End-to-End VHT Simulation with Frequency Correction” on page 1-36

Delay Profile and Fluorescent Lighting Effects

This example demonstrates the impact of changing the TGac delay profile, and it shows how fluorescent lighting affects the time response of the channel.

Delay Profile Effects

Create VHT and generator configuration objects. Set the sample rate to 80 MHz.

```
vht = wlanVHTConfig;
wgc = wlanGeneratorConfig;
fs = 80e6;
```

Generate random binary data and create a transmit waveform using the configuration objects.

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

Create a TGac channel object. Set the delay profile to `'Model-A'`, which corresponds to flat fading. Disable the large-scale fading effects.

```
tgac = wlanTGacChannel('SampleRate',fs, ...
    'ChannelBandwidth',vht.ChannelBandwidth, ...
    'DelayProfile','Model-A', ...
    'LargeScaleFadingEffect','None');
```

Pass the transmitted waveform through the TGac channel.

```
rxSigA = step(tgac,txSig);
```

Set the delay profile to `'Model-C'`. Model-C corresponds to a multipath channel having 14 distinct paths, with a 30 ns RMS delay spread. The maximum delay spread is 200 ns, which corresponds to a coherence bandwidth of 2.5 MHz.

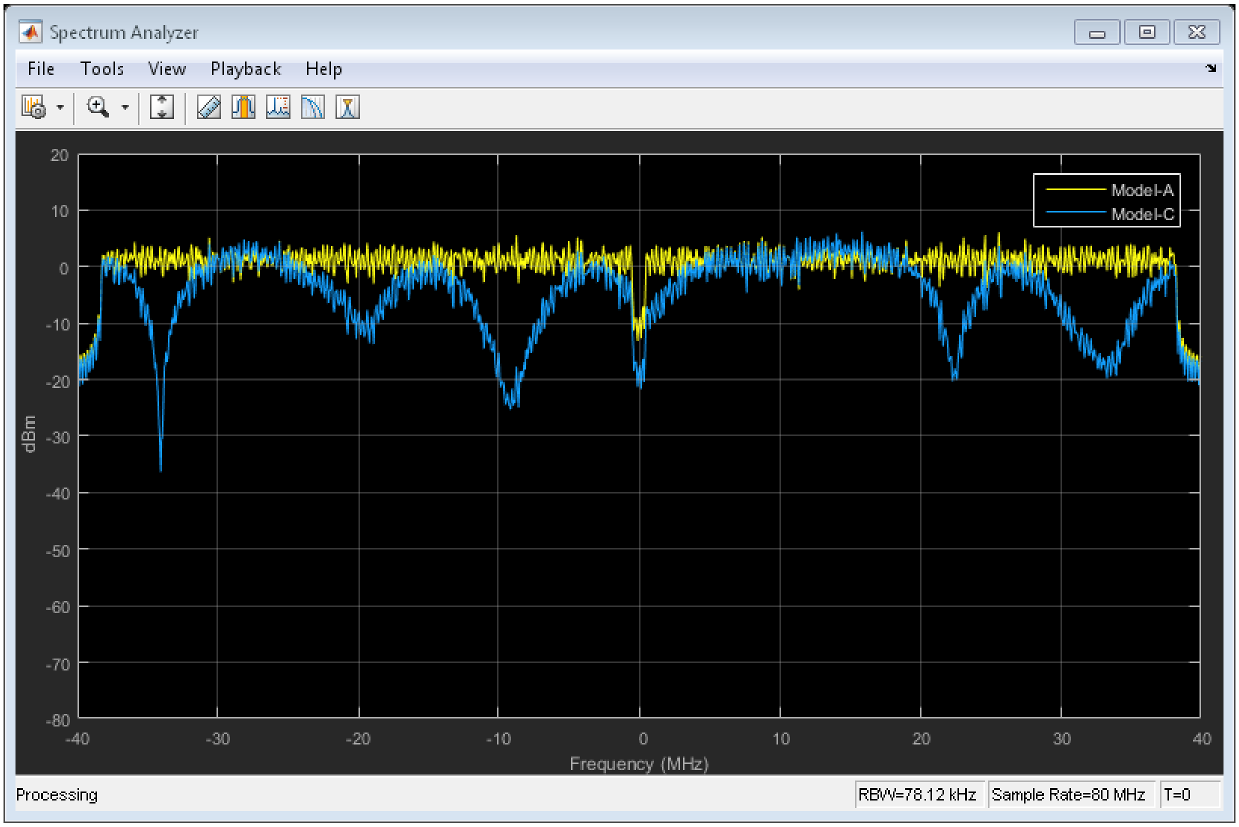
```
release(tgac)
tgac.DelayProfile = 'Model-C';
```

Pass the waveform through the model-C channel.

```
rxSigC = step(tgac,txSig);
```

Create a spectrum analyzer and use it to visualize the spectrum of the received signals.

```
sa = dsp.SpectrumAnalyzer('SampleRate',fs, ...
    'ShowLegend',true,'ChannelNames',{'Model-A','Model-C'}, ...
    'SpectralAverages',10);
step(sa,[rxSigA rxSigC])
```



As expected, the frequency response of the model-A signal is flat across the 80 MHz bandwidth. Conversely, the model-C frequency response varies because its coherence bandwidth is much smaller than the channel bandwidth.

Fluorescent Effects

Release the TGac channel, and set its delay profile to `'Model-D'`. Disable the fluorescent lighting effect.

```
release(tgac)
```

```
tgac.DelayProfile = 'Model-D';
tgac.FluorescentEffect = false;
```

To better illustrate the Doppler effects of fluorescent lighting, change the bandwidth and sample rate of the channel. Generate a test waveform of all ones.

```
tgac.ChannelBandwidth = 'CBW20';
fs = 20e6;
tgac.SampleRate = fs;

txSig = ones(5e5,1);
```

To ensure repeatability, set the global random number generator to a fixed value.

```
rng(37)
```

Pass the waveform through the TGac channel.

```
rxSig0 = step(tgac,txSig);
```

Enable the fluorescent lighting effect. Reset the random number generator, and pass the waveform through the channel.

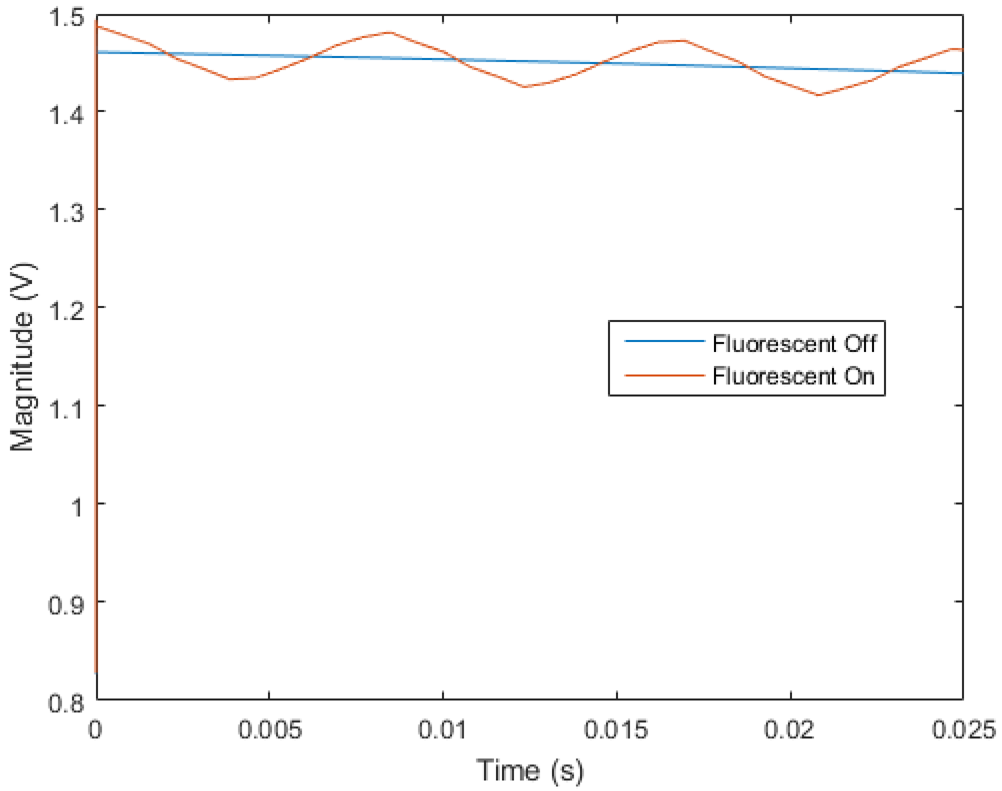
```
release(tgac)
tgac.FluorescentEffect = true;
rng(37)
rxSig1 = step(tgac,txSig);
```

Determine the time axis and channel filter delay.

```
t = ((1:size(rxSig0,1))'-1)/fs;
fDelay = tgac.info.ChannelFilterDelay;
```

Plot the magnitude of the received signals while accounting for the channel filter delay.

```
plot(t(fDelay+1:end),[abs(rxSig0(fDelay+1:end)) abs(rxSig1(fDelay+1:end))])
xlabel('Time (s)')
ylabel('Magnitude (V)')
legend('Fluorescent Off','Fluorescent On','location','best')
```



Fluorescent lighting introduces a Doppler component at twice the power line frequency (120 Hz in the U.S.).

Confirm that the peaks are separated by approximately 0.0083 s (inverse of 120 Hz) by measuring distance between the second and third peaks.

```
[~,loc] = findpeaks(abs(rxSig1(1e5:4e5)));
peakTimes = loc/fs;
peakSeparation = diff(peakTimes)
```

```
peakSeparation =

    0.0085
```

Generate Multi-User VHT Waveform

This example shows how to generate a multi-user VHT waveform from individual components. It also shows how to generate the same waveform by using the `wlanWaveformGenerator` function. The data fields from the two approaches are compared and shown to be identical.

Create a VHT configuration object having 3 users and 3 transmit antennas.

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

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

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

Set a different MCS value for each user.

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

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

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

```
vht = 

  wlanVHTConfig with properties:

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

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

```
vht.PSDULength
```

```
ans =

        2000        6008       12019
```

Display the field indices for the VHT waveform.

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

```
ind = 

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

Create the individual fields that comprise the VHT waveform.

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

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

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

     2
```

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

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

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

```
groupid =

     5
```

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

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

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

```
pktLen =

        2000
        1400
        1800
```

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

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

```
ans =

     1
```

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

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

```
mcs =

     0
     2
     4
```

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

Create three data sequences, one for each user.

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

Generate a VHT data field using these data sequences.

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

Create a generator configuration object in which windowing is disabled.

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

Generate a multi-user VHT waveform. Extract the data field from the waveform.

```
wv = wlanWaveformGenerator({d1 d2 d3},vht,wgc);

wvdata = wv(ind.VHTData(1):ind.VHTData(2),:);
```

Confirm that the two generation approaches produce identical results.

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

```
ans =

     1
```

Basic VHT Data Recovery Steps

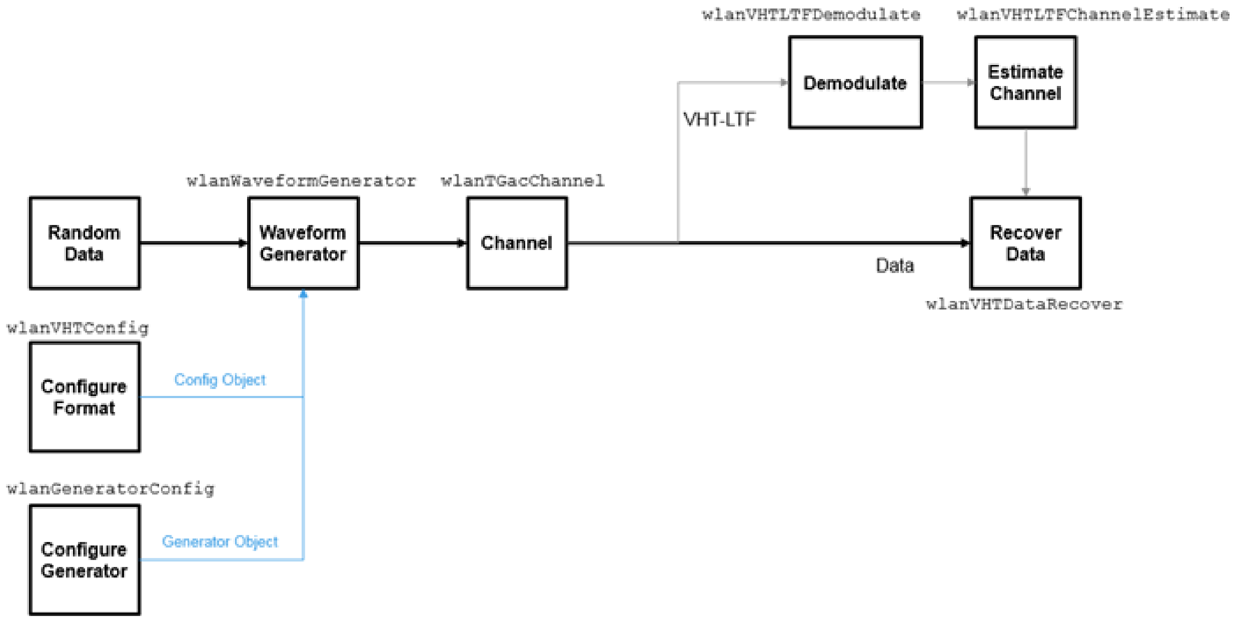
This example shows how to perform basic VHT data recovery. It also shows how to recover VHT data when the received signal has a carrier frequency offset. Similar procedures can be used to recover data with the HT and non-HT formats.

Basic Data Recovery

The WLAN System Toolbox™ provides functions to generate and recover IEEE® 802.11ac™ standards compliant waveforms. Data recovery is accomplished by these steps:

1. Generate a VHT waveform
2. Pass the waveform through a channel
3. Extract the VHT-LTF and demodulate
4. Estimate the channel by using the demodulated VHT-LTF
5. Extract the data field
6. Recover the data by using the channel and noise variance estimates

The block diagram shows these steps, along with their corresponding commands.



Create VHT format and waveform generator configuration objects.

```
vht = wlanVHTConfig;
wgc = wlanGeneratorConfig;
```

Create a VHT transmit waveform by using the VHT and waveform generator configuration objects. Set the data sequence to `[1;0;1;1]`. The data sequence is repeated to generate the specified number of packets.

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

Pass the received signal through an AWGN channel.

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

Determine the field indices of the waveform.

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

Extract the VHT-LTF from the received signal.

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

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

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

Extract the VHT data field.

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

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

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

rxBits(1:8)
```

```
ans =

     1
     0
     1
     1
     1
     0
     1
     1
```

Data Recovery with Frequency Correction

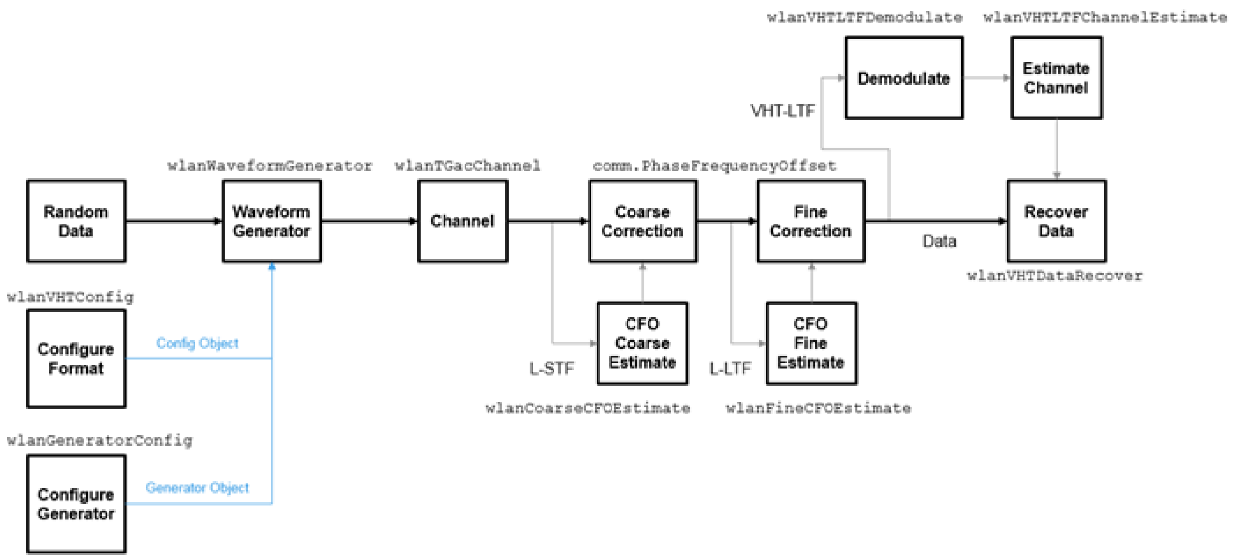
Data recovery when a carrier frequency offset is present is accomplished by these steps:

1. Generate a VHT waveform
2. Pass the waveform through a channel
3. Extract the L-STF and perform a coarse frequency offset estimate
4. Correct for the offset by using the coarse estimate
5. Extract the L-LTF and perform a fine frequency offset estimate
6. Correct for the offset by using the fine estimate
7. Extract the VHT-LTF and demodulate
8. Estimate the channel by using the demodulated VHT-LTF

9 Extract the data field

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

The block diagram shows these steps, along with their corresponding commands.



Set the channel bandwidth and sample rate.

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

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

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

Generate a VHT waveform containing a random PSDU.

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

Create a 2x2 TGac channel.

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

Create a phase and frequency offset object.

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

Pass the transmitted waveform through the noisy TGac channel.

```
rxSigNoNoise = step(tgac,txSig);
rxSig = awgn(rxSigNoNoise,15);
```

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

```
rxSigFreqOffset = step(pfo,rxSig,500);
```

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

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

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

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

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

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

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

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

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

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

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

```
rxData = rxSig2(ind.VHTData(1):ind.VHTData(2),:);
```

```
rxPSDU = wlanVHTDataRecover(rxData,chEst,0.03,vht);
```

Calculate the number of bit errors in the received packet.

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

```
numErr =

     0
```