MANUAL

LabVIEW Communications 802.11 Application Framework 2.5

This document provides detailed, technical information about how to use LabVIEW Communications 802.11 Application Framework.

Contents

1 Introduction

802.11 Application Framework provides a ready-to-run, easily modifiable real-time physical layer (PHY), and lower and middle medium access control (MAC)-layer reference design based on the IEEE 802.11 wireless standard. The application framework is available with LabVIEW Communications System Design Suite (LabVIEW Communications).

This application framework provides a substantial starting point for researchers looking for ways to improve the IEEE 802.11 standard by exploring brand-new algorithms and architectures that can support the tremendous increase of the number of terminals, inventing new waveforms by which to modulate and demodulate the signals, or finding new multi-antenna architectures that fully exploit the degrees of freedom in the wireless medium.

The application framework is comprised of modular PHY and MAC blocks implemented using LabVIEW Communications. It is designed to run on the powerful Xilinx Kintex-7 FPGA and an Intel x64 general-purpose processor, which are tightly integrated with the RF and analog front ends of the NI software defined radio (SDR) hardware.

The application framework is designed from the ground up for easy modifiability, while adhering to the main specifications of the IEEE 802.11 standard. This design allows wireless researchers to quickly set up and run their real-time prototyping laboratory based on the IEEE 802.11 standard. They can then primarily focus on selected aspects of the protocol that they wish to improve, and easily modify the design and compare their innovations with the existing standards.

2 Scope

The application framework provides functional elements of the physical (PHY) layer as well as the medium access control (MAC) layer of a single station (STA). The implementation includes a transmitter (TX) and a receiver (RX).

The following subsections describe which PHY and MAC functionalities from the IEEE 802.11 standard [1] are supported by the application framework.

2.1 System Features

The PHY and MAC functionalities implemented in the application framework allow communication between multiple nodes. A station can send data to either a single station or to a group of stations.

The application framework provides an external message-based interface to transmit and receive packets and to configure the transmission parameters. It can be used to connect an external higher MAC implementation to the framework. The interface is called *Middle MAC SAP* and described in Chapter 5; SAP refers to Service Access Point. Alternatively, when no external MAC implementation is used, the host application also offers a sample data source and sink for simple data exchange.

The implemented MAC functions include all functionalities essential for shared medium access using the distributed coordination function (DCF) according to Section 10.3 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. DCF uses carrier sensing multiple access with collision avoidance (CSMA/CA). The basic access procedure includes channel sensing, which waits for the channel to be free followed by a random backoff time. This basic access procedure is supported with and without request-to-send (RTS)/clear-to-send (CTS) exchange, which is useful for protecting long packets. This exchange reserves the channel for the duration of the sequence. The receiver acknowledges a successful packet reception by replying with an acknowledgement (ACK) packet. The sender ensures a robust communication by sending retransmissions in case no ACK was received. More information about the implemented MAC features is given in Section 2.2. Implementation details are described in Chapter 6.

The PHY implements the orthogonal frequency-division multiplexing (OFDM) PHY (also called 802.11a, Legacy or non-high throughput (non-HT)) described in Chapter 17 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1] as well as the Very High Throughput (VHT) PHY (also called 802.11ac) described in Chapter 21 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. Only single input, single output (SISO) and single-user transmission and reception are supported. The Legacy or non-HT subcarrier format supports a bandwidth of 20 MHz. The VHT subcarrier format supports bandwidths of 20 MHz, 40 MHz and 80 MHz.

Besides PHY packet transmission and reception, the PHY features include power measurement, clear channel assessment (CCA) and automatic gain control (AGC). More information about the implemented PHY features is given in Section 2.3. Implementation details are described in Chapter 7.

2.2 Medium Access Control Layer Overview

The implemented MAC functions include all essential functionalities for a shared medium access using the DCF according to Section 10.3 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. It involves functions for channel monitoring and determining opportunities when the channel may be accessed (TX opportunity), sending automatic responses (CTS and ACK), access procedure handling, MAC frame encoding at the transmitter, and MAC frame decoding at the receiver.

The following features are related to channel monitoring and TX opportunity detection which are described in Section 6.5.of DCF control block:

- Carrier sensing (CS) as described in Section 10.3.2.1 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. This includes physical CS where the channel state is detected based on signal detection and energy detection performed by the PHY, as well as virtual CS with the help of a network allocation vector (NAV).
- NAV handling as described in Section 10.3.2.4 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. The NAV is used to maintain a prediction of future traffic on the medium, based on duration information announced in RTC/CTS frames.
- Interframe space handling (IFS) as described in Section 10.3.2.3 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. The following IFSs are supported:
    - Short interframe space (SIFS)—Used for transmission of ACK and CTS packets.
    - DCF interframe space (DIFS)—Used if a packet was successfully received.
    - Extended interframe space (EIFS)—Used if a packet was not successfully received, either because of an error on PHY level (PhyRxEnd indication indicates error) or because frame check sequence (FCS) check failed on MAC level.

The following automatic responses are supported and implemented in the Frame Sequence TX control module (part of MAC TX) which is described in Section 6.3.5:

- MAC-level acknowledgements as defined in Section 10.3.2.2 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]—Sending an ACK packet as a response to a successfully received data or management packets.
- RTS/CTS procedure as defined in Section 10.3.2.2 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]—Sending a CTS packet as response to a successfully received RTS packet.

The following features are related to access procedure handling:

- Backoff handling as described in Section 10.3.4.3 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. This includes choosing a random backoff value within the contention window. The contention window is increased and reset as required by the recovery procedure.
- RTS/CTS procedure as described in Section 10.3.2.5 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].
- Recovery procedures as described in Section 10.3.4.4 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. This

includes sending retransmissions in case of errors and maintaining several retransmission counters. The counters are compared to the retry limits dot11ShortRetryLimit and dot11LongRetry-Limit to determine when the recovery procedure needs to be aborted. Depending on the success of a transmission, requests for increasing or resetting the contention window are generated.

- Duplicate detection and recovery procedure as described in Section 10.3.2.11 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. Duplicates can occur if multiple retransmissions are successfully received. The duplicate detection filters the duplicates based on a sequence number. At the transmitter, only the baseline sequence number space (SNS1) is supported. At the receiver, only the "not QoS Data" cache (receiver cache identifier (RC1)) is supported.
- Backoff handling is a module inside the DCF control, which is described in Section 6.5.2.1. The RTS/CTS procedure is handled inside the Frame Sequence TX control module as part of the MAC TX (described in Section 6.3.5). The recovery procedures are handled in the MPDU Retransmission Control module as part of MAC TX (described in Section 6.3.4). Duplicate detection is a module in the MAC RX (described in Section 6.4.5). The corresponding TX function, the sequence number assignment, is a module in the MAC TX (described in Section 6.3.1).

Overall, the following sequence types are supported:

- RTS | CTS | DATA | ACK
- DATA | ACK
- DATA

The following are MAC transmitter functions for packing the MAC service data unit (MSDU) and assembling the PHY service data unit (PSDU):

- MAC protocol data unit (MPDU) generation to assemble the MAC frame composed of MAC header, frame body and FCS according to Section 9.2 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. The supported frame structures are described in Section 2.2.2.
- Aggregated MPDU (A-MPDU) aggregation as defined in Section 9.7 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1] with the limitation that only single-MPDU (one A-MPDU subframe) is supported.

The corresponding modules are implemented as part of the MAC TX and described in Sections 6.3.6 and 6.3.7, respectively.

The following corresponding functional elements are provided for the MAC receiver:

- A-MPDU De-Aggregation with the limitation that only single-MPDU (one A-MPDU subframe) is supported.
- MPDU disassembly and frame check sequence (FCS) check.
- MPDU Filtering that evaluates the received MPDU configuration and decides if the packet was addressed to this station.

The following features are not supported:

- All features related to coordination functions other than DCF (that is, point coordination function (PCF), hybrid coordination function (HCF) and mesh coordination function (MCF)).
- All transmitter and receiver functions that are optional or that require special capabilities (for example, quality of service (QoS)).
- MSDU fragmentation as described in Section 10.2.7 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].
- A-MPDUs as defined in 9.7 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1] which contain multiple A-MPDU subframes.
- Block acknowledgement as described in Section 10.24 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].

Most of the listed MAC functions (referred to as *lower MAC*) are implemented on the FPGA because they need to react within the tight timing requirements mandated by the DCF. Only some MAC functions (referred to as *middle MAC*) are implemented on the host. The architecture is explained in Chapter 3. Implementation details are given in Chapter 6.

Higher MAC functionality is not implemented. Instead, an interface called MAC Middle Service Access Point (SAP) is provided which can be used to connect an external higher MAC implementation. The interface is described in Chapter 5.

2.2.1 MAC Frame Formats

The application framework follows the frame structure for the MPDU as defined in Section 9.2 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. A MAC frame consists of the MAC header, the frame body and the FCS field. The existence of the MAC header fields depends on the MPDU configuration, for example, on frame type and subtype.

The frame structures for data and management frames are shown in Figure 2-1 and Figure 2-2. The frame structure for control frames with subtypes RTS, CTS and ACK are shown in Figure 2-3, Figure 2-4 and Figure 2-5 respectively.

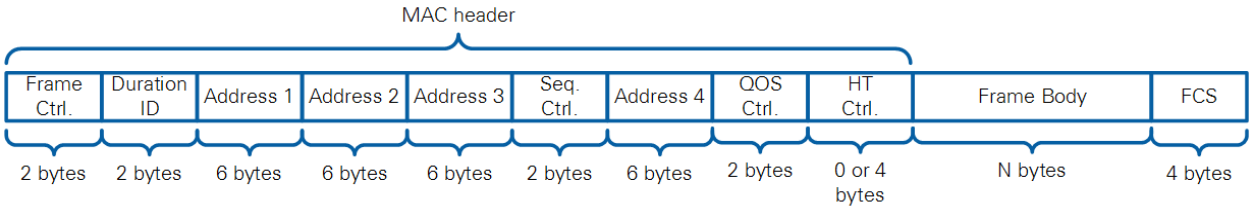

Figure 2-1: Structure of a Data Frame As in Section 9.2.3 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* **[1]**

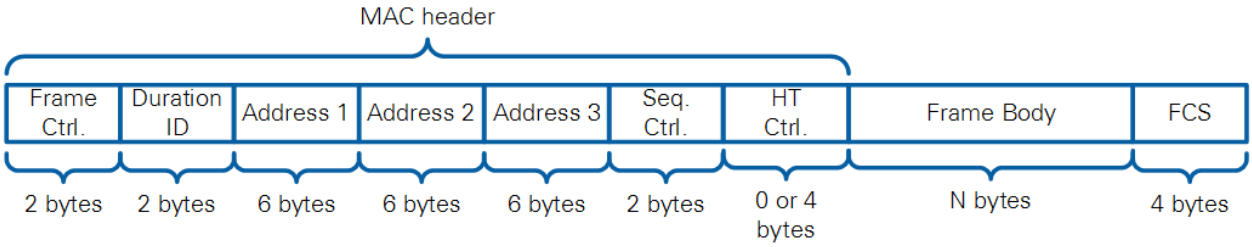

Figure 2-2: Management Frame As in Section 9.3.3.2 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* **[1]**

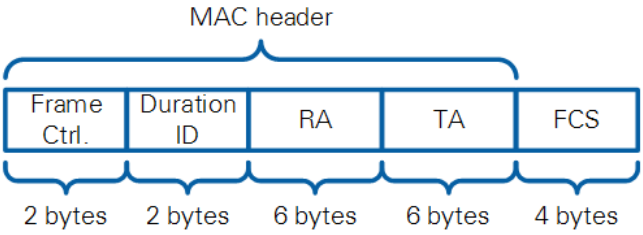

Figure 2-3: Control Frame: RTS Frame As in Section 9.3.1.2 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* **[1]**

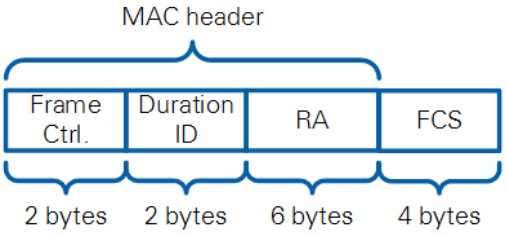

Figure 2-4: Control frame: CTS Frame As in Section 9.3.1.3 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* **[1]**

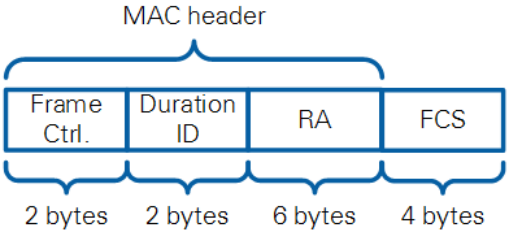

Figure 2-5: Control Frame: ACK Frame As in Section 9.3.1.4 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* **[1]**

Please note that Figure 2-1 and Figure 2-2 show the general frame structure. Not all of the fields are present in all cases. In the chosen implementation, the presence and the contents of the fields are defined as follows:

- **Frame control**—This field contains a bit mask as defined in Figure 9-2 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. The following subfields are filled: type, subtype, retry. Frame types and subtypes which are supported by the application framework are shown in Table 2-1.
- **Address fields**—The mapping of the address fields is shown in Table 2-2. In case no address is mapped to the address field, then the field is removed.
- **QoS control/HT control**—In the MAC transmitter implementation, these fields are not present. In the MAC receiver, these fields are extracted and reported as part of the RX indication but the fields are not interpreted in any way.
- **Frame body**—This field is only present in data and management frames. It contains the MPDU (in case of Legacy mode) or the A-MPDU (in case of VHT mode).
- **FCS**—Consists of a 32-bit cyclic redundancy check (CRC) as defined in Section 9.2.4.8 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].

Table 2-1: Frame Types Handled by the Application Framework

Frame Type	Frame Subtype	Initiator/Receiver of the Frame
Data	Data	Higher layers (Middle MAC SAP)
Management	Any	Higher layers (Middle MAC SAP)
Control	RTS, CTS, ACK	Automatically generated/evaluated inside lower MAC

Table 2-2. Presence and Mapping of Address Fields

Frame Type		Address 1	Address 2	Address 3	Address 4
Type == Control, to/from DS not relevant					
subtype==RTS		recipient address (RA)	transmitter address (TA)	—	—
subtype==CTS		recipient address (RA)	—	—	—
subtype==ACK		recipient address (RA)	—	—	—
Type == Data, subtype not relevant					
to DS	from DS				—

0	0	recipient address (RA)	transmitter address (TA)	basic service set identifier (BSSID)	—
0	1	recipient address (RA)	transmitter address (TA)	source address (SA)	—
1	0	recipient address (RA)	transmitter address (TA)	destination address (DA)	—
1	1	recipient address (RA)	transmitter address (TA)	destination address (DA)	source address (SA)
type == Management, from / to DS not relevant					
		recipient address (RA)	transmitter address (TA)	basic service set identifier (BSSID)	—

2.2.2 Aggregate MPDU

Figure 2-6 shows the A-MPDU format and also indicates that the application framework supports A-MPDU with one single MPDU as defined in Section 9.7.1 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].

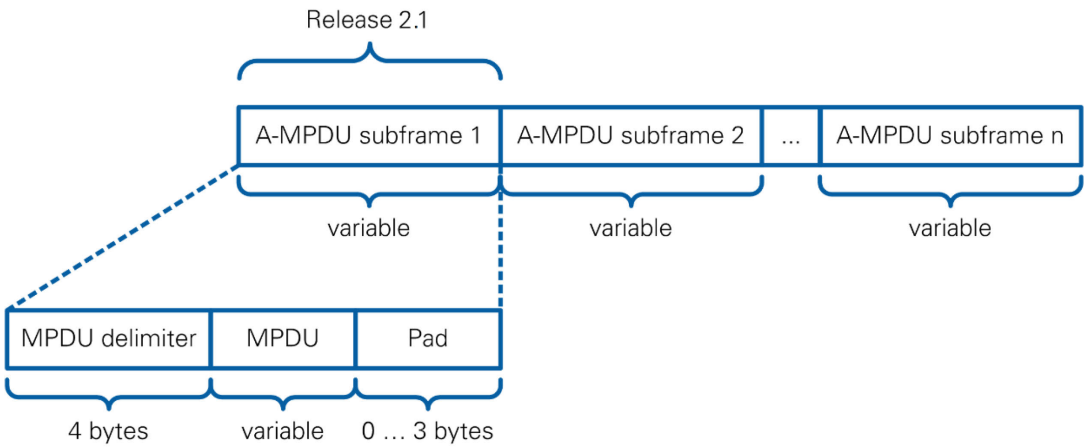

Figure 2-6: Format of A-MPDU

2.3 PHY Layer Overview

2.3.1 PHY Supported Features

The PHY in the application framework implements features of the OFDM PHY as defined in Chapter 17 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1] (also called 802.11a, non-HT or Legacy) and the

VHT PHY (also called 802.11ac) defined in Chapter 21 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].

The following limitations exist:

- Long guard interval only
- SISO architecture, ready for multiple-input, multiple-output (MIMO)
- Support of single user only for 802.11ac applications

The application framework provides the following PHY transmitter functionalities:

- Bit scrambling
- Convolutional encoding and bit interleaving
- Binary Phase-shift keying (BPSK)/quadrature amplitude modulation (QAM) constellation mapper up to 256-QAM
- Pilot sequence generation
- Signal fields generation
- OFDM modulation
- Guard interval (GI)[1] insertion
- Addition of training fields

The PHY TX is real-time configurable, that is, the configuration can be changed from packet to packet.

The following corresponding functionalities are provided for the receiver side:

- CCA based on energy detection as well as on signal detection
- Packet detection
- AGC
- Time and frequency synchronization
- GI removal
- OFDM symbol demodulation based on fast Fourier transform (FFT)
- Channel estimation and zero-forcing equalization based on the legacy long training field (L-LTF) and very high throughput legacy training field (VHT-LTF)
- Phase correction based on the pilot subcarriers
- BPSK/QAM demodulation
- Deinterleaving
- Convolutional decoding based on Viterbi decoding
- Descrambling

[1] Notice that the terms *guard interval* and *cyclic prefix* are used synonymously in this document.

Similar to the PHY TX, the receiver chain is real-time configurable. The receiver configuration (packet length, modulation and coding scheme (MCS), and so on) is chosen automatically based on the decoded signal fields (legacy signal field (L-SIG), VHT signal field A (VHT-SIG-A).

The subcarrier format 802.11a supports the bandwidth of 20 MHz. The subcarrier format support 802.11ac supports bandwidths of 20, 40 MHz and 80 MHz. The following MCSs are supported:

- 802.11a, 20 MHz—all defined MCS, up to MCS 7 (64-QAM, code rate 3/4)
- 802.11ac, 20 MHz—all defined MCS, up to MCS 8 (256-QAM, code rate 3/4)
- 802.11ac, 40 MHz—all defined MCS, up to MCS 9 (256-QAM, code rate 5/6)
- 802.11ac, 80 MHz—reduced MCS range, up to MCS 4 (16-QAM, code rate 3/4)

2.3.2 PHY Frame Formats

Figure 2-7 and Figure 2-8 show PHY frame formats of 802.11a and 802.11ac subcarrier formats, respectively. The application framework follows the PHY frame format as specified in Section 18.3. Fields specifically needed for the 802.11ac format have the name extension prefix VHT, and fields specifically needed for the 802.11a format are referred to as Legacy (L).

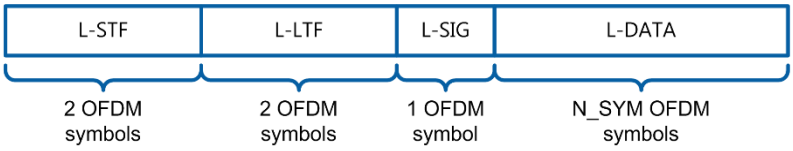

Figure 2-7: PHY Frame Format 802.11a

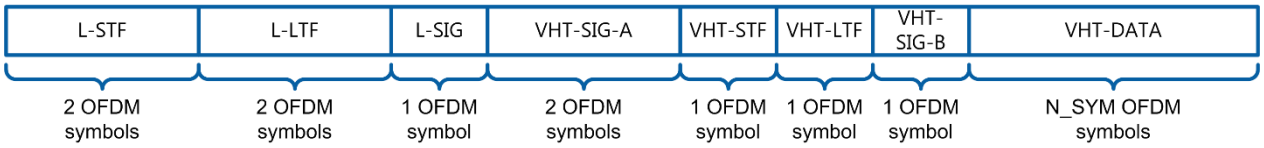

Figure 2-8: PHY Frame Format 802.11ac

Depending on the subcarrier format, the PHY frame consists of the following fields:

- Legacy short training field (L-STF), a static field that is used for packet detection and AGC at the receiver.
- L-LTF, a static field that is used for time and frequency synchronization as well as channel estimation.
- L-SIG, a dynamic field that contains information about the applied MCS and the frame length.
- VHT-SIG-A, a dynamic field that contains information about the applied channel bandwidth, MCS, and, if configured on MIMO, multi-user settings.
- VHT short training field (VHT-STF), a static field to be used for improving AGC estimation when performing MIMO transmission.

- VHT-LTF, a static field to be used for MIMO channel estimation purposes.
- VHT signal field B (VHT-SIG-B), a dynamic field containing information of the frame length and the MCS for single-user or multi-user modes.
- Legacy data (L-DATA) or VHT data (VHT-DATA) is a dynamic field, which is uniquely defined by the MAC message. The number of OFDM symbols N_{SYM} used for the data depends on the chosen MCS and the payload length.

The number of OFDM symbols necessary for the demodulation of the physical layer convergence procedure (PLCP) protocol data unit (PPDU) data or for channel estimation during a null data packet (NDP) [1] in the VHT-LTF field per frame can be either 1, 2, 4, 6, or 8. The number of OFDM symbols is determined by the total number of space-time streams across all users being transmitted in the VHT PPDU [1].

2.3.3 AGC

The framework implements an automatic gain control. The RF input power is permanently measured, and the RX gain is chosen so that the RX chain has an optimal operation point during reception of a packet. Because the AGC in implemented on the FPGA, it allows per-packet AGC. More information about the AGC implementation is given in Section 7.2.1.

2.3.4 CCA

The framework implements CCA including energy detection and signal detection. The CCA indications are evaluated by the DCF module to decide when the channel may be accessed.

Energy detection is performed according to Section 21.3.18.5 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1] with deviations listed below. The CCA energy detection module permanently measures the power on the filtered signals for the primary channel (20 MHz), the secondary channel (20 MHz) and the secondary40 channel (40 MHz). If the measured power exceeds a certain threshold, the channel is reported busy.

The following are deviations to the standard regarding energy detection:

- The following are conditions for CCA indication (busy, primary), Section 21.3.18.5.3 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]:
    - Signal detection (detection of a PPDU) shall be triggered only if the primary channel power exceeds -82 dBm. This check is not implemented.
    - The threshold used in combination with the time slot aCCATime is defined as a fixed value of 20 dBm above the minimum modulation and coding rate sensitivity (-82 dBm + 20 dB = -62 dBm). In the implementation, the value is configurable from the host using the parameter **CCA energy detection threshold**.

- The following are conditions for CCA indication (busy, secondary or secondary40), Section 21.3.18.5.4 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]:
    - The threshold used in combination with the time slot aCCATime is defined as a fixed value of -62 dBm. In the implementation, the host parameter CCA energy detection threshold is used as well.
    - A second condition using the power threshold -72 dBm in combination with the time slot aCCAMidTime is defined. This condition is not implemented.
- The following is a condition for CCA indication (busy, secondary80), Section 21.3.18.5.4 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]: Monitoring of the secondary80 channel is described. This functionality is not implemented.

More information about the CCA is given in Section 7.2.2.

2.3.5 Channelization

The supported channelization is illustrated in Figure 2-9. It shows where the selected subband is located in the baseband signal.

The 256-point FFT of the PHY covers 80 MHz bandwidth. Based on the 20 MHz bandwidth of the non-HT signal, the 80 MHz bandwidth can be divided into four subbands of 20 MHz, which are the trapezoids in top plot of Figure 2-9. The numeric control **primary channel selector** determines which subband is used as the primary channel. During transmission only the selected subband is used by the PHY in case of 20 MHz transmissions. The remaining subcarriers are filled with zeros. For wider bandwidths, the subbands get combined as shown in Figure 2-9. For example, if the primary channel selector is set to 3, it will be used in combination with the lower 20 MHz subband 2 in case of a 40 MHz transmission. It shows *PrimaryChannelUpperBehavior* as described in Annex E of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. For an 80 MHz transmission, the full FFT range will be used independently of the selected subband.

For the VHT PPDU format, the PHY RX must dynamically switch based on the channel bandwidth information in VHT-SIG-A. This switching is performed by selecting the correct range in the baseband frequency domain without changing the RF center frequency.

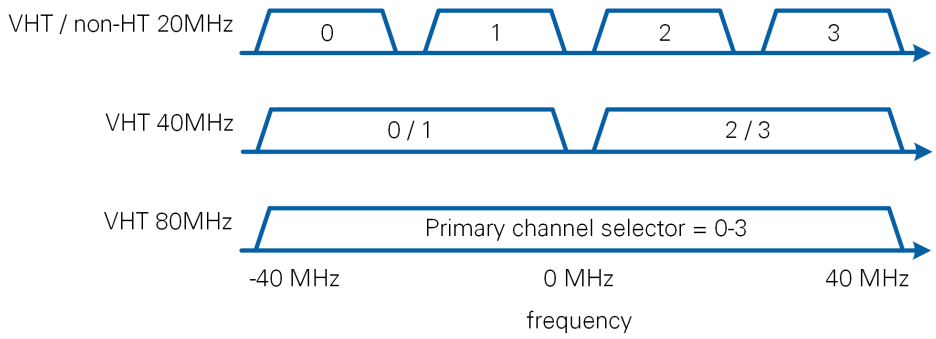

Figure 2-9: Channelization Used in the Application Framework

The RF center frequency depends on the primary channel center frequency and the primary channel selector, which are set on the host. It is chosen so that the center of the 20 MHz primary channel is at the primary channel center frequency as shown in Figure 2-10.

Valid frequencies when communicating with commercial equipment depend on the channel sets defined in Annex E of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. Otherwise, the wrong set of 20 MHz subbands could be concatenated for higher bandwidth transmissions.

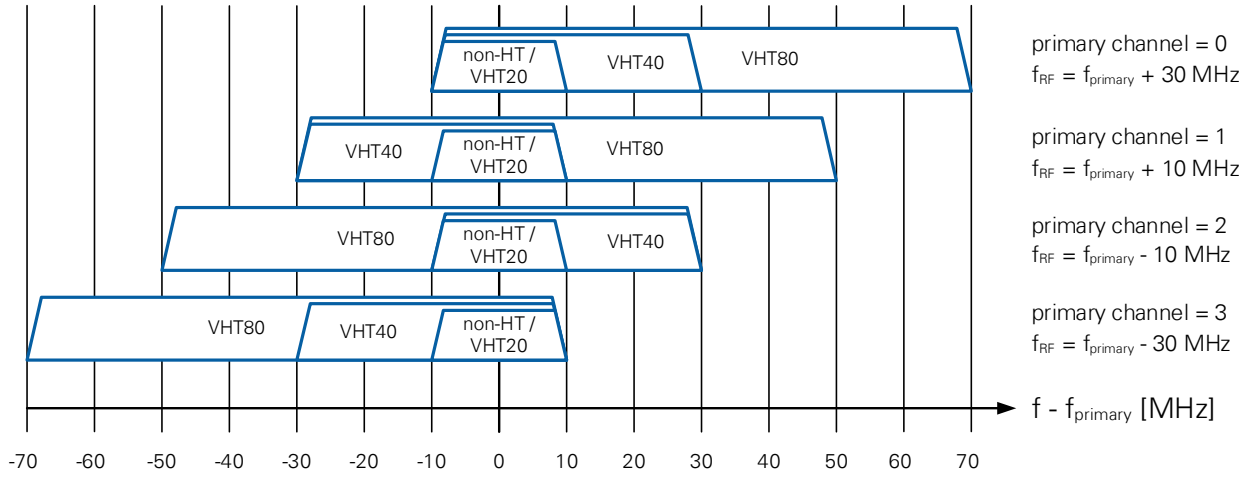

Figure 2-10: Frequency Mapping/Channelization

3 Architecture

A high-level system overview is shown in Figure 3-1. It shows the separation between higher MAC, Middle MAC, lower MAC, PHY and DCF control. Furthermore, it shows the partitioning between host and FPGA.

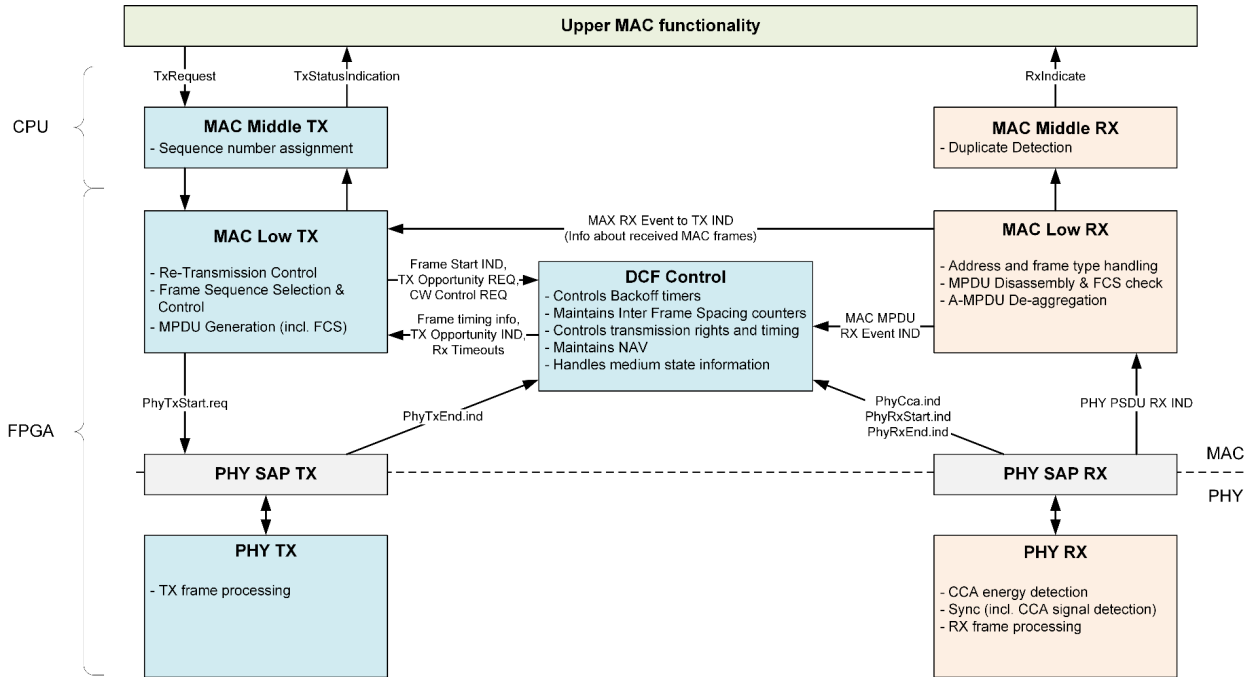

Figure 3-1: High-Level System Overview

3.1 Partitioning between Host and FPGA

All PHY and MAC functions requiring fast reaction within the SIFS timings defined by the DCF are implemented on the FPGA. Examples are sending CTS after successful RTS reception, sending ACK after successful data reception, and sending a pending TX request as soon as the channel is idle and backoff counting finished. The affected MAC functions are summarized as lower MAC in Figure 3-1. All DCF-related functions are summarized as DCF control. This includes backoff counting, interframe spacing counting for SIFS, DIFS and EIFS, maintaining the NAV, and handling the medium state information.

All MAC functions that are not time-critical, or that implement buffers (duplicate detection), are implemented on the host. This includes the middle MAC features and the interface to the higher MAC (Middle MAC SAP). The host also implements a stub for simple higher MAC functionality which read from a data source for packet generation and writes to a data sink for packet reception. The user interface also implements controls, indicators, graphs and plots for configuration and status display. More details on the host features are given in Chapter 4.

3.2 Communication between Modules

As shown in Figure 3-1, the high-level blocks are (lower and middle) MAC TX, PHY TX, (lower and middle) MAC RX, PHY RX and DCF control. They communicate with each other by sending requests and indications. A request is a command to perform a certain action like transmitting a packet. An indication is an information that an event happened

like a packet was received. The modules react up reception of a request/indication by changing their inner state, eventually waiting for a request/indication from another module to arrive and ultimately by sending a request/indication by themselves.

Requests and indications are also used for communication between modules within the high-level blocks. Typically, one module triggers the downstream module without expecting feedback. An example is the MAC RX block where a module sends an indication to the next module when it finished processing. In some cases, a module expects feedback from the downstream module. An example is the MAC TX block where the MPDU (Re-)Transmission Control module sends a request to the Frame Sequence TX Control and waits for an indication that the frame sequence was completed.

The requests and indications are implemented as clusters. Each cluster contains the elements which correspond to the parameters of the message and additionally a **valid** flag. On the FPGA, the **valid** flag is set to True in the clock cycle when the message is ready for processing.

Within the MAC TX and the MAC RX, the same cluster is used by multiple modules for collecting all information related to the current transmission or reception. The fields contain default values at first and they are filled by the submodules step by step. An example is the MSDU fragmentation which reads the MSDU length and sets the MPDU frame body length.

More information about the MAC TX and MAC RX modules is given in Sections 6.3 and 6.4. The DCF control is described in Section 6.5. More information about the PHY is presented in Chapter 7.

3.3 Architecture Interfaces

3.3.1 Interface between PHY and lower MAC: PHY SAP

The PHY SAP provides the interface to the PHY which is used by the lower MAC. Both entities are implemented on the FPGA but in different clock domains. The PHY SAP is used for transferring the requests, indications and the associated payload data to the other clock domain.

Interface technology

Because the interface is implemented on the FPGA, the following interface technologies are used:

- Requests and indications (default)—Handshake of type cluster (native type of the corresponding request or indication).
- Requests and indications (if the request or indication may arrive faster than it is processed)—Target-scoped FIFO of type cluster (native type of the corresponding request or indication).

- Payload data—Target-scoped FIFO of type unsigned byte.

Data formatting

No data conversion is necessary because the native data types can be used in all cases.

Messages

The messages used in the PHY SAP are listed in Section 7.1.

3.3.2 Interface between Lower MAC and Middle MAC: Interprocess Communication Protocol Interface

The application framework uses a protocol to transfer data from host to FPGA and vice versa which is called interprocess communication protocol (ICP). In the current implementation, it is used for communication between lower MAC (implemented on the FPGA) and the middle MAC (implemented on the host). The idea of ICP is to send messages with fixed structure and varying length over a channel that can transmit a stream of bytes such as the direct memory access (DMA) first-in-first-out memory buffers (FIFOs).

Interface Technology

The interface uses DMA FIFOs (target-to-host or host-to-target FIFOs) of type unsigned byte.

Data Format

The following ICP packet format is shown in Figure 3-2:

- The first 8 bytes represent the ICP header. The first 4 bytes contain the message length and the second 4 bytes contain the payload length.
- The second field is the message. The content of this field depends on the message. Each message must define encoding and decoding functions which translate the request or indication cluster to an array of bytes.
- The third field is the payload which is associated with the message, for example the MAC service data unit (MSDU). If there is no payload associated with the message, the field can also be omitted.

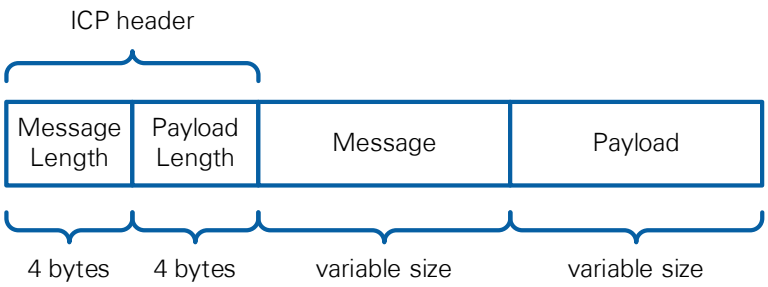

Figure 3-2: Format of ICP Packet

Messages

The messages defined for the interface between lower MAC and middle MAC are listed in Section 6.1.

3.3.3 Interface between Middle MAC and Higher MAC Implementation: Middle MAC SAP

The middle MAC offers an interface which can be used either with the implemented higher MAC stub or with an external MAC implementation. It is called *Middle MAC SAP*. It is described in detail in Chapter 4.4.

Interface technology

The interface uses user datagram protocol (UDP). Four UDP ports are used as described in Section 5.2.

Data format

The format of a MAC Middle SAP message is described in Section 5.3. It consists of a general message header, a SAP sub-header and multiple sub-messages. The general message header contains the message type ID and the reference ID. The SAP sub-header contains a timestamp and the number of sub-messages. Each sub-message contains one parameter set associated with the message.

Messages

The messages defined by the MAC Middle SAP are described in Section 5.3.

3.4 FPGA Design Considerations

3.4.1 Clocking

There are three clock domains within the FPGA design. The first clock domain is for the RF loop. It depends on the target and is referred to as the *data clock*. For the USRP RIO with 40 MHz bandwidth, the data clock is set to 120 MHz. For the USRP RIO with 120 MHz or 160 MHz bandwidth, it is set to 200 MHz. For the FlexRIO design, it is set to 130 MHz.

The second clock domain is used for PHY baseband processing. This clock rate must fulfill the requirements for 80 MHz bandwidth support. For this purpose, a 256-point FFT must run for each OFDM symbol. The Xilinx FFT is set to Radix-4, Burst I/O architecture to produce a continuous output of this core with minimum latency. With these settings, the FFT requires 871 cycles for loading data, executing, and unloading data, which are executed sequentially. That process leads to a minimum clock rate of 241.94 MHz, assuming an OFDM symbol duration of 3.6 µs with a short guard interval.

Note: A short guard interval is not implemented in the application framework, but the design is prepared for the future use of a short guard interval. Thus, the baseband clock is set to 250 MHz for all targets.

Based on this clock rate, the computation of each OFDM symbol of 4 µs duration (with long guard interval) can take up to 1,000 clock cycles.

The third clock domain is used for MAC processing. Thus, the MAC and PHY are running in different clock domains. Whereas the PHY runs at 250 MHz clock speed to be able to meet the SIFS timing, the MAC runs at 100 MHz, which is sufficient for MAC data processing in byte[2]. This relaxes the timing constraints for the MAC and eases its extensions. Data between PHY and MAC are exchanged through handshakes for events like start requests and end indications and FIFOs for MPDUs.

The clock domains are asynchronous because they do not use the same reference clock. The application framework uses FIFOs to transfer data between the RF and the PHY baseband loops. This transfer is straightforward for the RX chain since the samples are taken from RF to the higher rate baseband loop as soon as they are available. The TX chain produces a large number of samples per packet. This large number of samples could lead to overflows when transferring data to the RF loop, which has a fixed sample rate of 80 MS/s. To avoid overflows, the application framework generates the TX samples OFDM symbol-wise, and a trigger is generated in the RF clock domain. This architecture means that the FIFO fills at the same rate as it is read from.

3.4.2 Timing Constraints

To ensure efficient use of the shared unlicensed spectrum, the IEEE 802.11 standard defines challenging requirements for the interframe timing, particularly for the frame transmissions after a frame reception and for the frame transmissions after channel sensing. To meet those requirements, tight integration of PHY and lower MAC functionalities is needed. In this subsection, we describe the requirements of the 802.11 specifications for systems with OFDM-based PHY layer and compare the values assumed in the 802.11 specifications to actual achieved values of the application framework.

3.4.2.1 Timing Budget for Transmission after Frame Reception

One example of *transmission after frame reception* is receiving a data frame and transmitting an ACK frame with SIFS. Figure 3-3 shows such a scenario and the definitions of processing delays used in the IEEE 802.11 standard (refer to Figure 10-19 and Section 10.3.7 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]).

[2] The minimal clock rate for MAC could be 31.5 MHz (250 MHz/8 bits). It reflects the byte processing in MAC versus the bit processing in PHY.

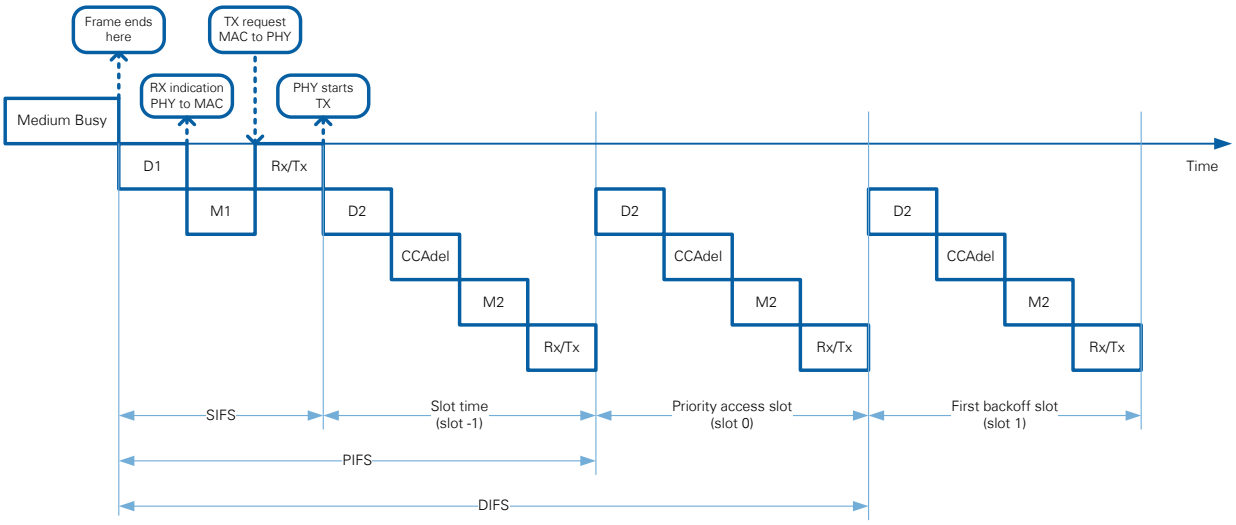

Figure 3-3: Timing Relationships for Transmission after Frame Reception

Table 3-1 summarizes the requirements of the standard and compares it to the values of the application framework. The RX and TX PHY delays, aRxPLCPDelay and aTxPLCPDelay respectively, are comprised of processing delays of the I/Q Processing modules and Bit Processing Modules (refer to Sections 7.3.3 and 7.3.2 for more information). It is assumed that separate RF channels are used for RX and TX, and no RX/TX switch is used. Therefore, aRxTxSwitchTime and aTxRampOnTime are assumed to be zero. The assumption for D1 being equal to 12.0 µs is calculated from the relation SIFS = D1 + M1 + Rx/Tx.

Table 3-1: Timing Budget for Transmission after Frame Reception

Name	Content	Assumption of IEEE 802.11 Standard (Section 17.4.4 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1])	Value for 802.11 Application Framework
D1	aRxRFDelay + aRxPLCPDelay IQ proc. + aRxPLCPDelay bit proc.	**~12.0 µs**	0.68 µs + 2.87 µs + 8.36 µs **= 11.91 µs**
M1	**aMACProcDelay 1**	**<2.0 µs**	**<2.0 µs**
Rx/Tx	aTxPLCPDelay bit proc. + aTxPLCPDelay IQ proc. + aRxTxSwitchTime + aTxRampOnTime + aTxRFDelay	**<2.0 µs**	0.0 µs + 0.0 µs + 0.0 µs + 0.0 µs + 1.29 µs **= 1.29 µs**

	= aRxTxTurnaroundTime		
	Sum	**16 µs**	**15.2 µs**[3]

3.4.2.2 Timing Budget for Transmission after Channel Sensing

An example of *transmission after channel sensing* is transmitting a data frame after a backoff procedure. Figure 3-4 visualizes such a scenario and the definitions of processing delays used in the IEEE 802.11 standard (refer to [1] Figure 10-19 and Section 10.3.7).

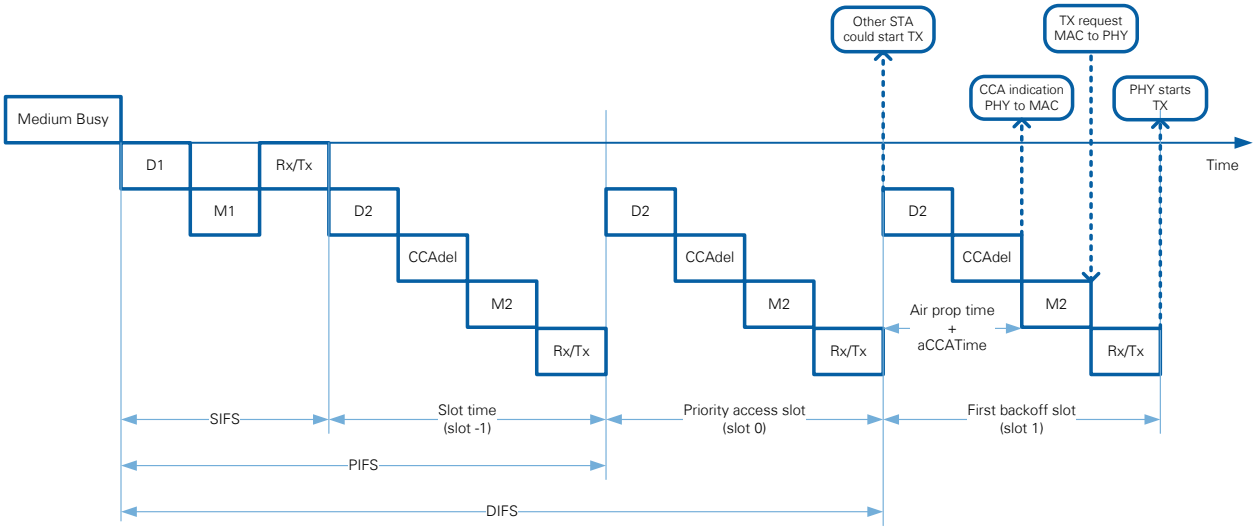

Figure 3-4: Timing Relationships for Transmission after Channel Sensing

Table 3-2 summarizes the requirements of the standard and compares it to the values of the application framework. These requirements assume that D2 + CCAdel = Air Propagation Time + aCCATime, which can be derived from the definitions given in Section 10.3.7 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. aCCATime refers to the time needed for performing the CCA operation.

[3] MAC TX timing control ensures that interframe-spacing and slot timing requirements from IEEE specifications are met.

Table 3-2: Timing Budget for Transmission after Channel Sensing

Name	Content	Assumption of IEEE 802.11 standard (Section 17.4.4 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1])	Value for 802.11 Application Framework
Air Prop. Time		**<<1.0 µs**	Assume for instance **0.2 µs** for 60 m distance
aCCATime	**CCA detection time**	**<4.0 µs**	**<2.0 µs**
M2	**aMACProcDelay 2**	**<2.0 µs**	**<2.0 µs**
Rx/Tx	aTxPLCPDelay bit proc. + aTxPLCPDelay IQ proc. + aRxTxSwitchTime + aTxRampOnTime + aTxRFDelay **= aRxTxTurnaroundTime**	**<2.0 µs**	0.0 µs + 0.0 µs + 0.0 µs + 0.0 µs + 1.29 µs **= 1.29 µs**
	Sum	**9 µs**	**5.5 µs**[4]

3.4.3 Level Plan and Baseband Operating Points

3.4.3.1 Digital-to-Analog Headroom, Analog-to-Digital headroom, Signal Power

The digital-to-analog converter (DAC) and analog-to-digital converter (ADC) should operate in a manner that avoids clipping and saturation of the outgoing and incoming signal, respectively. For proper adjustment of the DAC and ADC operating points, consider the following factors:

- OFDM has a high peak-to-average power ratio approximately between 9 dB to 12 dB. This range implies that a DAC headroom of 15 dB would avoid clipping.
- On the receiver side, signals on adjacent channels can occur that are 10 dB higher than the desired signal. Having an ADC headroom of 25 dB would also handle such signal constellations without running the ADC into saturation. For example, with an ADC resolution of 14 bits (as in the case of the USRP RIO devices) and a noise figure of approximately 9 dB, approximately 8 effective bits available are in the baseband processing, which is sufficient for up to 256-QAM.

[4] MAC TX timing control ensures that interframe-spacing and slot-timing requirements from IEEE specifications are met.

3.4.3.2 Reference Signals according to IEEE 802.11a/ac

TX I/Q and RX I/Q and bit processing and the related submodules of the application framework follow the power normalization of the test vector generation tool provided by IEEE [2]. The test vector generation tool follows Equation (21-11) of Section 21.3.7.4 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1] and, hence, the power of the complex-valued baseband signal is normalized to be equal to 1. TX and RX IQ and bit processing and the related submodules of the application framework are tested against vectors generated by this tool. For any extension towards standard-compliant PHY features, NI strongly recommends that you test against vectors generated with this tool.

3.4.3.3 Complex Fixed Point (Format, Precision)

The mixed-signal processing output data type is a <1.15> fixed-point value. The peak-to-average power ratio of OFDM (refer to Section 3.4.3.1) means that the output must be extended by two bits in the integer part. Because each bit of a complex data type corresponds to 6 dB in signal power, this fact adds a headroom of 12 dB for the numeric representation. The data path interface between the baseband and mixed-signal processing consists of a <3.13> fixed-point value on both the TX and RX path. The conversion is done by reinterpretation of the 16 bits. The <3.13> fixed-point format allows you to compare against the reference signal of the user guide for 802.11ac waveform generator, IEEE 802.11-11/0517r6 [2] (refer to the previous Section 3.4.3.2). Furthermore, changing between simulation and operation mode does not require a change in scaling. In general, the precision of the fixed-point logic follows the fixed-point requirements of the implemented algorithms, and it is not optimized regarding to resource usage.

More information about fixed-point formats and related precisions to avoid clipping and saturation can be found in the following sections:

- Table 7-2 of Section 7.2.4 referring to RX IQ processing
- Table 7-9 of Section 7.3.3 referring to TX IQ processing

3.4.4 Global Timestamp

The system generates a global timestamp derived from the Data Clock (refer to Section 3.4.1**Error! Reference source not found.**). Its granularity is 0.1 µs, and it is used as a time base of MAC modules and the event tracing.

4 Host Functionality

The Host is a sample application that covers all important features of the application framework. This covers configuration of the FPGA target and exchanging payload data. It furthermore sets parameters during runtime and queries and displays the system status. It also acts as the connection between MAC middle on the host and MAC low on the FPGA where it supplies configurations and payload data to the TX chain and

collects the payload and extracted configurations from the RX chain for further processing on the host.

Each station consists of a host application with associated FPGA code. It is recommended to run only one station on a host processor. When running multiple stations on one host processor, there may be limitation regarding the achievable throughput.

In the following subsections, an overview of the host architecture is provided. In addition, the MAC High Stub and MAC High Abstraction layer are presented. The integration of them to the system is also presented. Although the MAC Middle SAP is a host functionality, it will be presented in Chapter 5.

4.1 Front Panel Layout and System Configuration

The front panel is laid out as shown in Figure 4-1. The bar on the top contains the title, some minimal information and instruction, station activation control, and some basic health indication, as well as a stop button on the right.

The left-hand side contains the most important controls such as the primary channel center frequency, TX and RX RF ports, subcarrier format, MCS, AGC, and the MAC addresses.

The main part of the front panel (center and right) provides access to detailed configuration settings and monitoring parameters distributed over multiple tabs. Those tabs are as follows: MAC, RF & PHY, Advanced, Events, and Status.

The parameters for system configuration are split into three groups, those are:

Group 1—Parameters can only be changed at system start. Those are the **RIO device** and **reference clock** which are in the bar on the top.

Group 2—Parameters can only be changed while the station is off. Those parameters are **station number**, **primary channel center frequency**, **power level**, **TX and RX port numbers**, **device MAC Address**, the **dot11RTSThreshold** and the retry limits **dot11ShortRetryLimit** and **dot11LongRetryLimit**.

Group 3—Parameters can be changed at any time. Such parameters are **subcarrier format**, **MCS**, **AGC mode**, **destination MAC address**, **data source** and **data sink** as well as the **backoff** parameter.

Refer to the *LabVIEW Communications 802.11 Application Framework Getting Started Guide* for a detailed description of all controls and indicators [3].

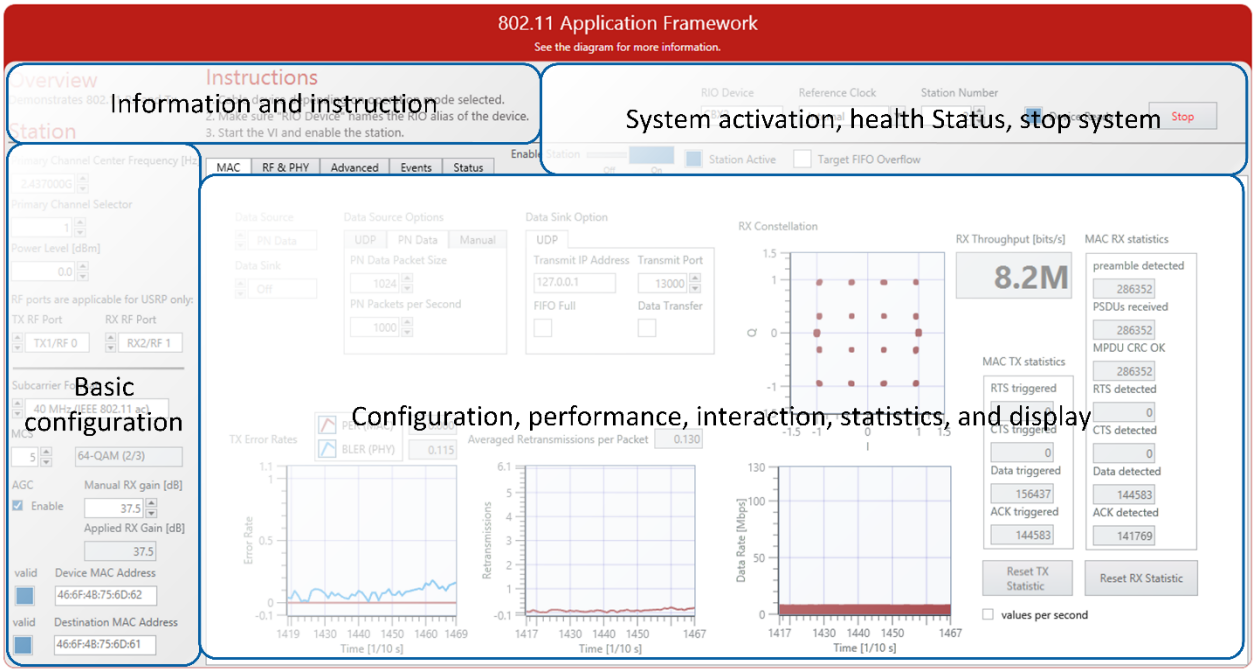

Figure 4-1: Front Panel Layout

4.2 Block Diagram Architecture

The host block diagram follows a basic layout scheme shown in Figure 4-2. On the left, there is the initialization code which loads the bitfile to the FPGA, starts the DMA FIFOs, and prepares local resources such as queues. In the center of the VI there are six loops which are executed continuously on runtime. Those loops are covering the jobs of configuration, data exchange, and status display. On the right-hand side, there is the cleanup code which is executed on system shutdown. The host executes sequentially as indicated by the yellow arrows (Figure 4-2) in the following order:

1. Initialization
2. Main Program
3. Cleanup

Handles that are persistent at runtime (such as FPGA interfaces, UDP port handles, or FIFOs) are grouped in a session cluster to minimize the number of wires on the block diagram. The different loops are further explained in the next sections.

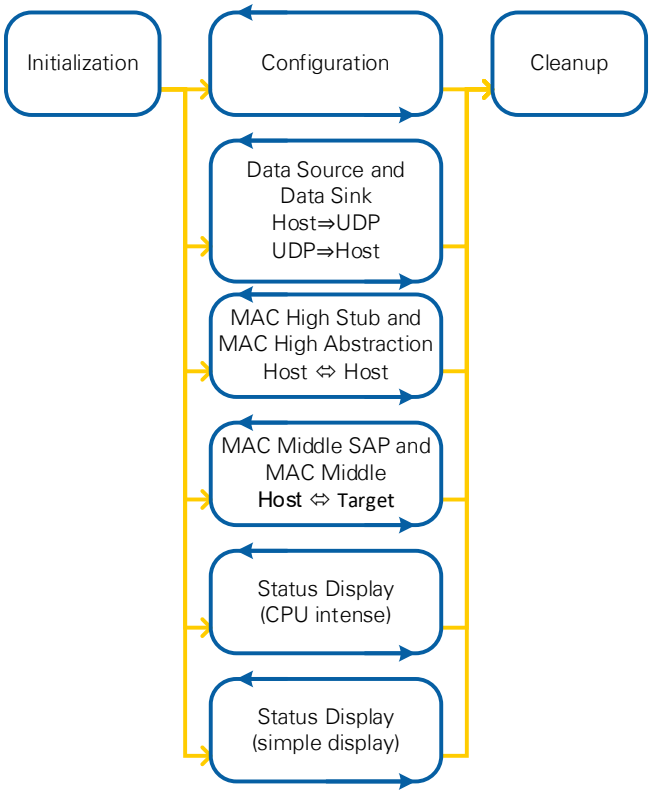

Figure 4-2: Host Schematic Block Diagram

4.2.1 Initialization

Initialization starts by resetting all controls and indicators in a sequence structure to initialize all controls and indicators to a known state. This also ensures proper execution after a previous run. The startup procedure is as follows:

1) Reset all controls and indicators.
2) Create a stop notifier that will be used in the main program.
3) Initialize the system.
   a) Identify USRP bandwidth.
   b) Load bitfile.
   c) Open UDP port.
   d) Setup FPGA target based on the selected reference clock.
   e) Start DMA FIFOs.
   f) Initialize the required queues.
4) Create a notifier for the logging entries using the initial value of max log level, which has different states.

4.2.1.1 Data Exchange

A session cluster is used on the host to store local resources such as the FPGA interface, queues, UDP ports, and so on. It is also used to share information between the different loops of the main program.

Table 4-1 lists the queues used to buffer and pass payload and status information within the system.

Table 4-1: Message Queues Used in Host Application

Queue	Purpose
MAC High Requests	Stores the payload data given from the data source (UDP or pseudo-noise (PN)) for the target transmitter. The MAC High Abstraction reads the packets from the queue and includes each packet in an MSDU TX payload sub-message.
Triggered TX Requests	Stores the reference ID and MSDU length of each TX request. The reference ID is resorted in the Generated MAC SAP TX status indication node when reading the MAC TX END indication from the target and generating the MAC SAP TX Status indication.
receive queue	Buffer payload received from the target.
receive throughput queue	Stores information about received payload size and timestamps (used for throughput graph display).

4.2.2 Main Program

The main program executes in multiple parallel while loops. The following functionality is implemented on the host.

- **System Configuration:** Configure and change the parameters that can be static or dynamic as it is described in Section 4.1. Refer to the *LabVIEW Communications 802.11 Application Framework 2.2 Getting Started Guide* for more details about the system configuration [3].
- **Data source and data sink**: The data source reads data either from a UDP port or from a random data (pseudo-noise, PN data) source. The data sink sends the received data to a UDP port, if enabled.
- **MAC High Stub**: Reads the queue that is filled from the data source. If data is available, it requests a packet transmission by creating corresponding requests that are handled by the MAC High Abstraction Layer. If the MAC High Abstraction Layer indicates a received packet, it extracts the payload data and writes it to the queue which is read by the data sink.
- **MAC High Abstraction Layer:** It is provided to create the required interfaces with the corresponding checks between the MAC High Stub and the MAC Middle SAP. Furthermore, it provides an example to the user to create own interface between an external upper MAC application and the MAC Middle SAP.
- **MAC Middle SAP**: It provides the required interfaces between the middle and lower MAC functionalities provided by the application framework and the MAC High Abstraction or external higher MAC applications. The message-based interface is described in Chapter 5.
- **MAC Middle:** Implements the following middle MAC functions: Sequence Number Assignment described in Section 6.3.1 and Duplicate Detection

described in Section 6.4.5. They are implemented within the MAC Middle SAP nodes.

- **AGC:** The AGC mechanism on the host allows the user to switch between a manual gain configured on the front panel and the AGC implemented on the FPGA as described in Section 7.2.1. The AGC target signal power that enables the gain control can be configured from the host.
- **Performance and Statistics Display:**
  - Display MAC TX and MAC RX Statistics.
  - Monitor transmission and reception parameters.
    - Display channel transfer function.
    - Display RX constellation.
    - Display TX and RX power spectrums.
    - Display RF input power and baseband RX power.
  - Evaluate and display throughput, TX block error rate and TX packet error rate.
  - Monitor FIFO overflows.
  - Evaluate and display the counters at different layers of the following messages: TX requests, TX Confirmations, TX Indications, and RX Indications.
- **Event Logging:** Create log file for error messages; each log message has the following information: the module name, the error message, the log level, and the time stamp.

4.2.2.1 Timing Considerations

The loops for data transmission between target and host as well as between host and UDP ports run at a fast rate to achieve the maximal possible throughput. They are running at 10 ms and 5 ms, respectively. The loops for configuration and status display run every 100 ms to enable a responsive system. The loop to display events, constellation, channel estimation, and spectral plots runs every 250 ms.

4.2.2.2 Data Exchange

Data exchange of data which requires high throughput such as the TX and RX payload is realized with queues. The queues available on the host are listed in Section 4.2.1.1.

Data exchange of status parameters that are not changed often, like the **Station Active** flag, is realized with duplicated terminals. This means that the same terminal is read at multiple places, typically from within different loops. This approach was chosen to minimize the number of wires on the block diagram. The session cluster which contains the local resources is also realized as duplicated terminal.

4.2.3 Stop Procedure and Cleanup

The host must ensure a proper shutdown. All handles to hardware and software resources which were allocated during initialization are released during cleanup.

4.2.3.1 Stop Procedure

All loops need to be stopped before the cleanup code is executed. The following conditions can stop a loop:

- An error occurred in the loop.
- The stop button has been pressed. Note that the stop button is monitored in one loop only.
- Another loop has stopped for one of the previous conditions.

The first two conditions only stop a single loop. The other loops must be notified to stop as well. The **stop notifier** is used for this purpose. The loop which stopped first creates a notifier using Stop on Error. This notifier is received by all other loops using Check Stop. Upon receiving the notifier, the other loops will stop as well. See Figure 4-3 for a typical use of both nodes.

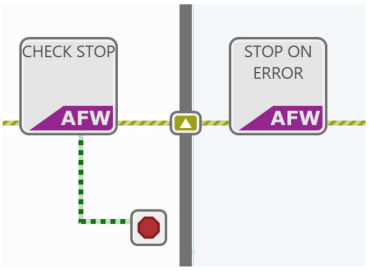

Figure 4-3. Use of the Stop Notifier in Host Loops

Users need to ensure to follow this design pattern when adding new loops to the hosts.

4.2.3.2 Cleanup Procedure

The cleanup procedure is executed in two steps. First, hardware related handles, such as FPGA reference, is closed. Second, data handling such as UDP handle is closed, as shown in Figure 4-4.

The host uses the information from the session cluster to identify resources to be deallocated.

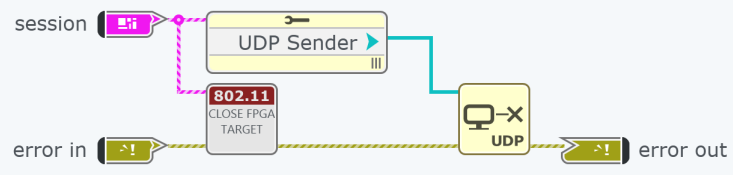

Figure 4-4. Closing Data Handling Related Resources on the Host

4.3 MAC High Stub

The 802.11 MAC High Stub function provided by the application framework is just a placeholder for a third party MAC connected to the MAC High Abstraction Layer. It does not contain actual 802.11 MAC High functionality. The functionality is limited to the following:

- Reading of the MSDU TX payload data (with potentially varying size) from a queue which is filled by a data source.
- Generation of the corresponding TX request in the MAC High Abstraction Layer specific format which includes the generation of corresponding configuration parameters needed by the MAC Middle SAP.
- Reception of RX indications from the MAC High Abstraction Layer. Extraction of reception parameters, and/or payload.
- Writing of the MSDU RX payload data into the queue of data sink.

4.4 MAC High Abstraction Layer

An 802.11 MAC High Abstraction Layer is needed to connect the application framework to a third party 802.11 higher MAC application, whose interfaces typically aren't (fully) compliant to the message based MAC Middle SAP (API) provided by the application framework. The implementation of MAC High Abstraction layer provides an example to the user to create own interface between an external application and the MAC Middle SAP. The following are general functionalities of the 802.11 MAC High Abstraction Layer:

- Translate MAC high specific requests into the appropriate request messages and message sequences defined for the MAC Middle SAP.
- Receive, decode, evaluate, and handle the confirmation and indication messages received from the application framework MAC Middle SAP and provide the information extracted from these messages to the MAC high in the format required by the MAC high.

The block diagram shown in Figure 4-5 gives an overview of the MAC High Abstraction module structure and interfaces to the MAC High Stub presented in Section 4.3 and MAC Middle SAP presented in Chapter 5. The definitions of the presented messages between different layers shown in Figure 4-5 are described in Section 5.3. The MAC High Abstraction has the following three top-modules:

- **MAC TX Request Confirm Abstraction:** It converts the configuration and payload inputs into MAC SAP TX request messages. Then, it writes them to the corresponding UDP socket: first the MAC SAP TX Configuration Request (MAC SAP TX CONFIG REQ), then the MAC SAP TX Payload Request (MAC SAP TX PAYLOAD REQ).
- **Receive MAC SAP TX Status Indication:** It dequeues two MAC SAP TX Confirmation messages, one message per each TX request and checks if the received header IDs match the ones of the created MAC SAP TX Configuration Request and MAC SAP TX Payload Request. Finally, it forwards the messages' confirmation states.
- **Receive MAC SAP RX Indications:** It dequeues the following MAC SAP RX indication message pairs from MAC Middle SAP: MAC SAP RX Configuration Indication (MAC SAP RX CONFIG IND) and MAC SAP RX Payload Indication (MAC

SAP RX PAYLOAD IND). It decodes the messages and checks if the sequence ID matches. The MAC SAP RX Configuration Indication should be received before MAC SAP RX Payload Indication. In addition, it checks configuration and payload messages for consistency (for example, matching MSDU indices and MSDU length values). Finally, it calls the MAC High RX Stub function and provides the extracted configuration and status parameters and payload data.

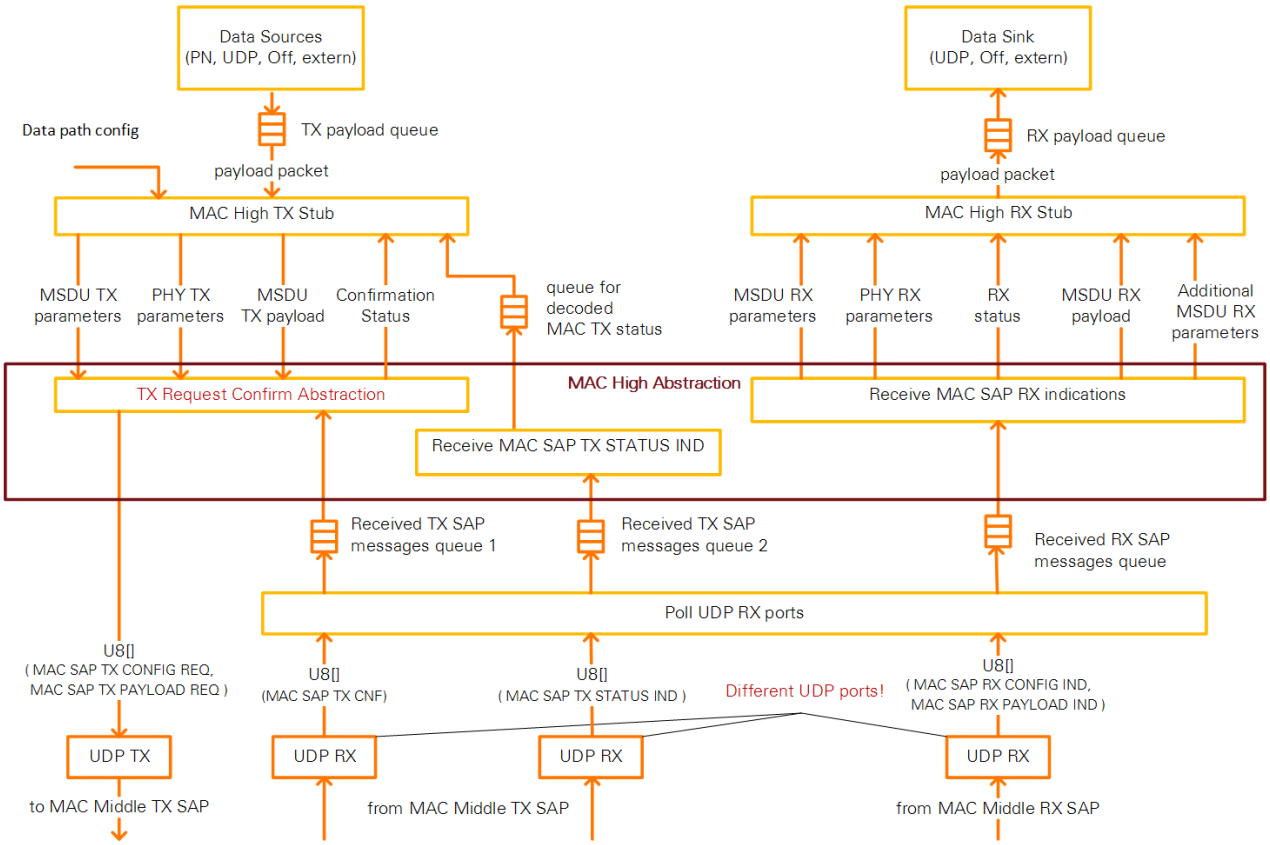

Figure 4-5: MAC High Stub and MAC High Abstraction Modules and Interfaces

4.4.1 MAC High Abstraction Interfaces

The outer interface to/from MAC high Abstraction and MAC TX and RX Stub and the inner interface to/from MAC high Abstraction and MAC Middle SAP are summarized in the following, see Figure 4-5. The description of every message is presented in Section 5.3.

4.4.1.1 Outer Interface

The following are inputs from MAC High TX Stub to TX Request Confirm Abstraction:

- MSDU TX parameters
- PHY TX parameters
- MSDU TX payload

The following are outputs to MAC High Stub:

- From TX Request Confirm Abstraction:
  - Confirmation Status
- From Receive MAC SAP TX Status Indication:
  - MAC TX Status
- From Receive MAC SAP RX indications:
  - MSDU RX parameters
  - PHY RX parameters
  - RX Status
  - MSDU RX Payload
  - Additional MSDU RX Parameters

4.4.1.2 Inner Interface

Internally, the MAC High Abstraction Layer uses the interface which is provided by the MAC Middle SAP. It is described in Section 5.1.1.

4.5 Integration into Overall System

Figure 4-6 provides a simplified block diagram of the overall system partitioning of the application framework. It shows where the MAC High Stub and MAC High Abstraction are integrated in the 802.11 host application. Furthermore, it shows how and where the MAC Middle SAP, presented in Chapter 5, is integrated into the overall system.

The MAC Middle SAP runs within the 802.11 host application. The message-based interface connects to the MAC High Abstraction using UDP ports. Internally it is directly connected to the MAC Middle functions **Sequence Number Assignment** and **Duplicate Detection**, which run on the host.

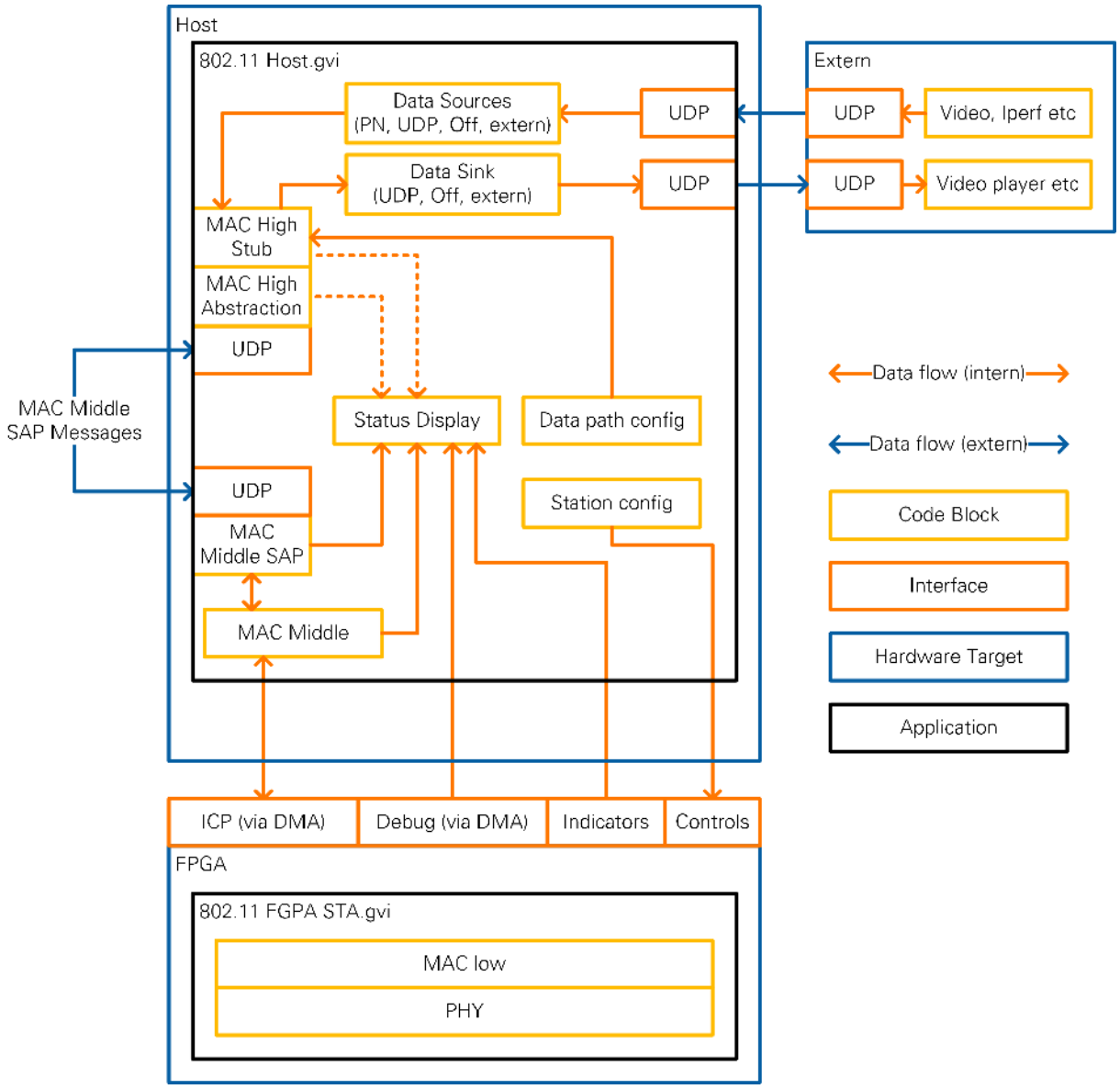

Figure 4-6: Overall System Diagram

4.5.1 Operating with External MAC High Application

The implementation of MAC High Abstraction layer and MAC Stub provides an example to the user to create own interface between an external application and the MAC Middle SAP. The MAC Middle SAP enables the connection of the application framework to an external MAC high application. For this, the 802.11 host application supports disabling of MAC High Abstraction and MAC Stub with its data and control interfaces. This operation mode is illustrated in Figure 4-7.

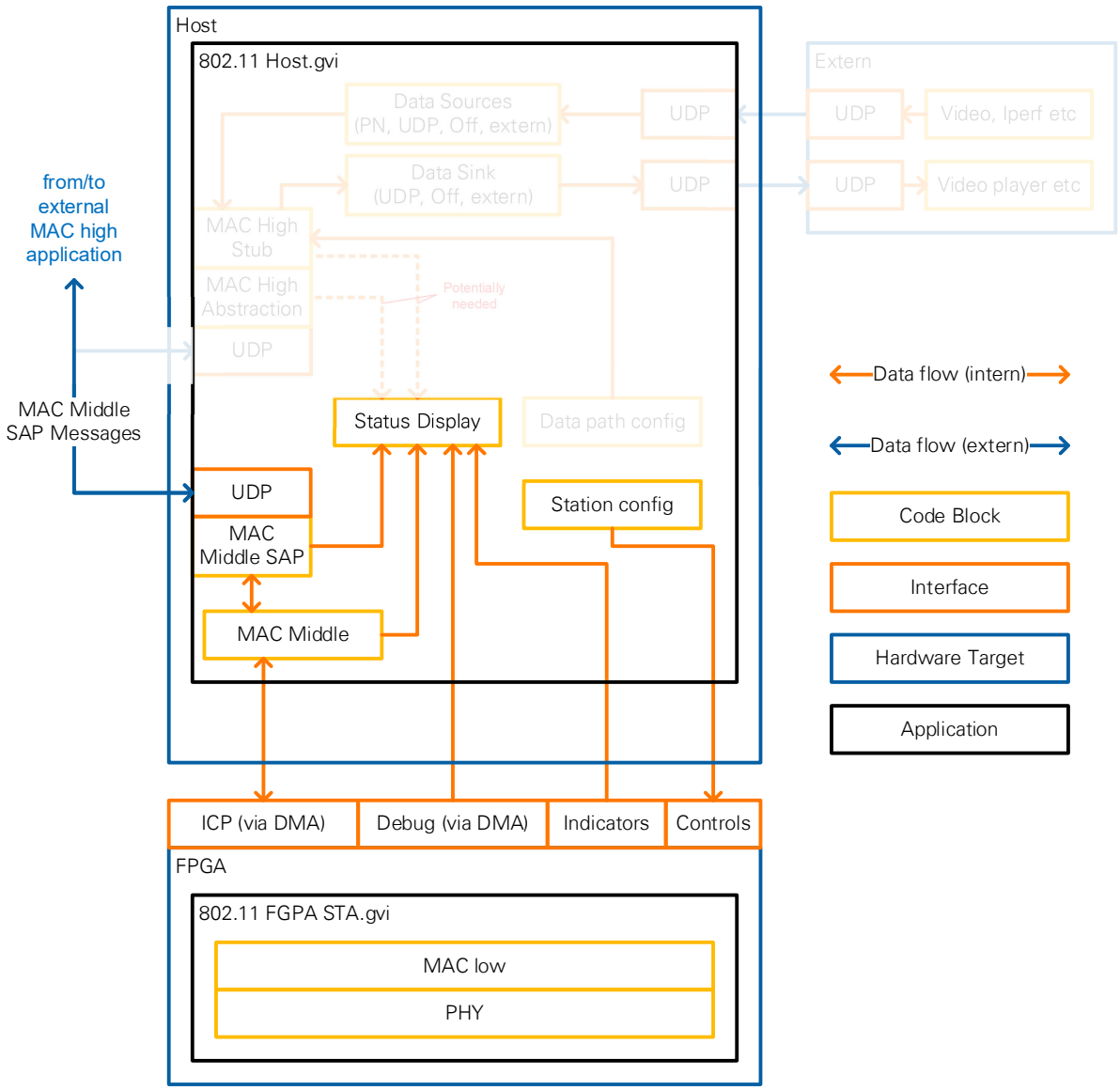

Figure 4-7: Operating with External MAC High Application

5 Middle MAC SAP

The MAC Middle SAP provides the interface between the MAC middle functionality provided by the application framework and a potential external upper MAC realization or other external applications. The API of the application framework allows external applications to use the MAC & PHY functionalities provided by the application framework. It is provided on top of MSDU Sequence Number Assignment in the MAC TX path as in Section 5.3 and on top of the Duplicate Detection in the MAC RX path as in Section 5.4. A message-based interface to/from the *upper MAC* application is provided. The provided message-based interface follows the general API concept and rules. The interface towards the MAC Middle implementation uses an event driven request & indication approach. The interface primitive implementation follows the requirements of the MAC Middle implementation.

The block diagram shown in Figure 5-1 gives an overview of the MAC Middle SAP-module structure as follows:

- It shows the outer interface between MAC middle SAP and MAC high and the corresponding messages in this interface.
- It shows the following top nodes of MAC Middle SAP:
  - MAC SAP TX Request Confirmation Handler
  - Generate MAC SAP TX Status Indication
  - Generate MAC SAP RX Indication
- It shows the inner interface between MAC middle SAP and MAC middle and the corresponding messages in this interface. In addition, it shows for each top node in MAC Middle SAP which functions are called from the MAC middle.

In the following subsections, the interfaces and the functionalities of each node will be described.

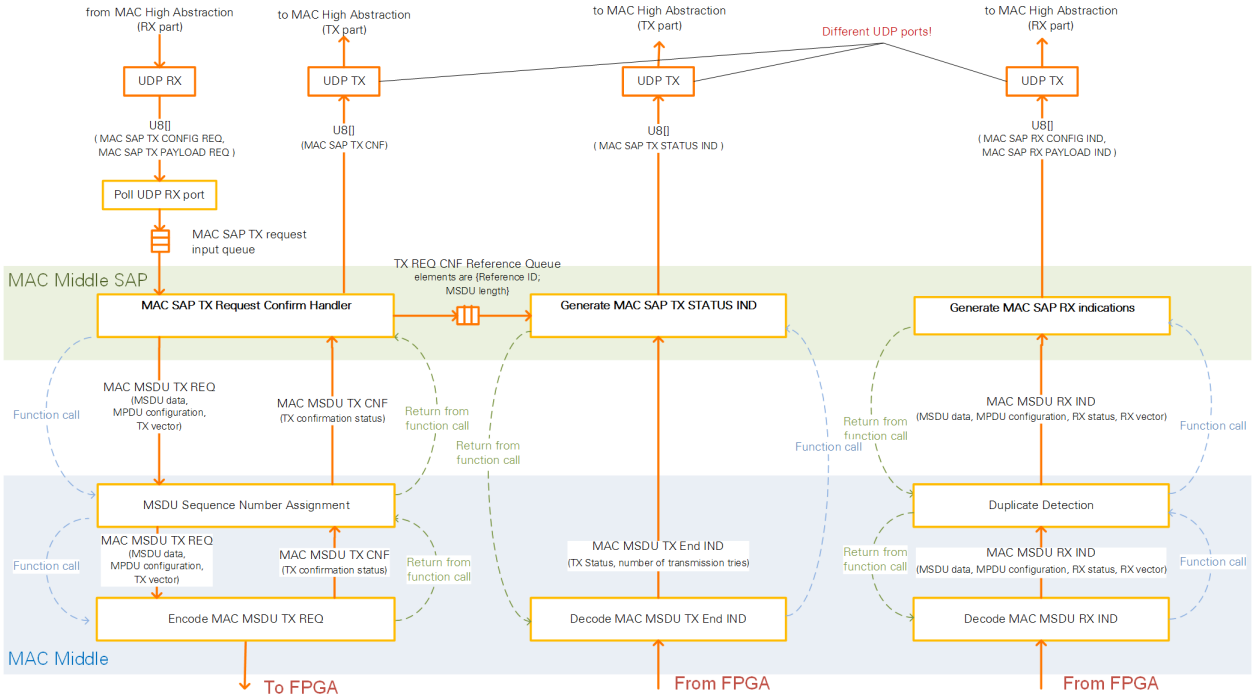

Figure 5-1: MAC Middle SAP Sub-Module Structure and Interfaces

5.1 Middle MAC SAP Interfaces

The interfaces of MAC Middle SAP are classified as either outer interface to/from upper MAC or inner interface to/from middle MAC. The messages list of each interface are presented in the following subsections.

5.1.1 Outer Interface to/from Upper MAC

From Figure 5-1, the inputs of the outer interface to upper MAC are as follows. The description of every message is presented in Section 5.3.

- MAC SAP TX CONFIG REQ (MAC SAP TX Configuration Request)
    - Parameter sets:
        - MSDU TX parameters
        - PHY TX parameters
- MAC SAP TX PAYLOAD REQ (MAC SAP TX Payload Request)
    - Parameter sets:
        - MSDU TX payload

From Figure 5-1, the outputs of the outer interface from upper MAC are as follows. The description of every message is presented in Section 5.3.

- MAC SAP TX CNF (MAC SAP TX Confirmation)
    - Parameter sets:
        - MAC SAP TX Confirmation
- MAC SAP TX STATUS IND (MAC SAP TX Status Indication)
    - Parameter sets:
        - MAC TX Status
- MAC SAP RX CONFIG IND (MAC SAP RX Configuration Indication)
    - Parameter sets:
        - MSDU RX parameters
        - PHY RX parameters
        - MAC RX status
        - (Additional MSDU RX Parameters)
- MAC SAP RX PAYLOAD IND (MAC SAP RX Payload Indication)
    - Parameter sets:
        - MSDU RX payload

5.1.2 Inner Interface to/from Middle MAC

From Figure 5-1, the inputs of the inner interface to middle MAC are as follows:

- MAC MSDU TX CNF (MAC MSDU TX Confirmation)
    - Parameter sets:
        - TX confirmation status
- MAC MSDU TX End IND (MAC MSDU TX End Indication)
    - Parameter sets:
        - TX Status
        - Number of transmission tries
- MAC MSDU RX IND (MAC MSDU RX Indication)
    - Parameter sets:
        - MSDU data
        - MPDU configuration
        - RX Status
        - RX Vector

From Figure 5-1, the outputs of the inner interface from middle MAC are as follows:

- MAC MSDU TX REQ (MAC MSDU TX Request)
    - Parameter sets:
        - MSDU data
        - MPDU configuration
        - TX Vector

5.2 UDP Socket Assignment

In the application framework, only UDP is supported. As it can be seen in Figure 5-1, the MAC Middle SAP is connected through a UDP transport layer to the MAC High Abstraction, on the other side it is directly connected to the MAC Middle, which is the upper layer of the application framework MAC and is running on the host, too.

To simplify the design, a different UDP port has been assigned for each message needs to be transferred between MAC high and MAC Middle SAP or vice versa. The UDP ports are quasi statically and have been set on dependence on station number (instance ID). As a result, different stations should use different instance IDs to assign different UDP ports for each station. It can help to mitigate wrong UDP port assignments in multi-station setups so that there is no conflict with existing port assignment.

For example, if you assume the station number is n. Then the assigned UDP ports for station number n are as follows:

- UDP port of MAC SAP TX REQ (TX CONFIG and Payload): $12100 + n$
- UDP port of MAC SAP TX CNF: $12200 + n$
- UDP port of MAC SAP TX Status IND: $12300 + n$
- UDP port of MAC SAP RX IND (RX CONFIG and Payload): $12400 + n$

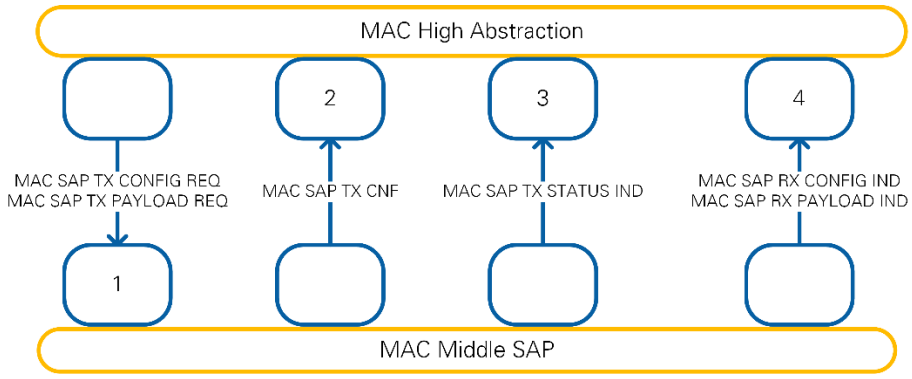

Figure 5-2: UDP Socket Assignment between MAC Middle SAP and MAC High Abstraction

5.3 Outer Interface Message Definitions

The general message structure of each message to/from upper MAC is shown in Figure 5-3. Each message has the following fields:

- **General message body:** The header of the message.
- **Message body:** It can have a number of sub-messages. Each sub-message has two fields:

- SAP sub-header: The header of the sub-message
- Sub-message body

From Figure 5-3, each message has a specific type ID. Table 5-1 lists the specific message type IDs supported in the application framework.

In the following subsections, the contents of each message are described.

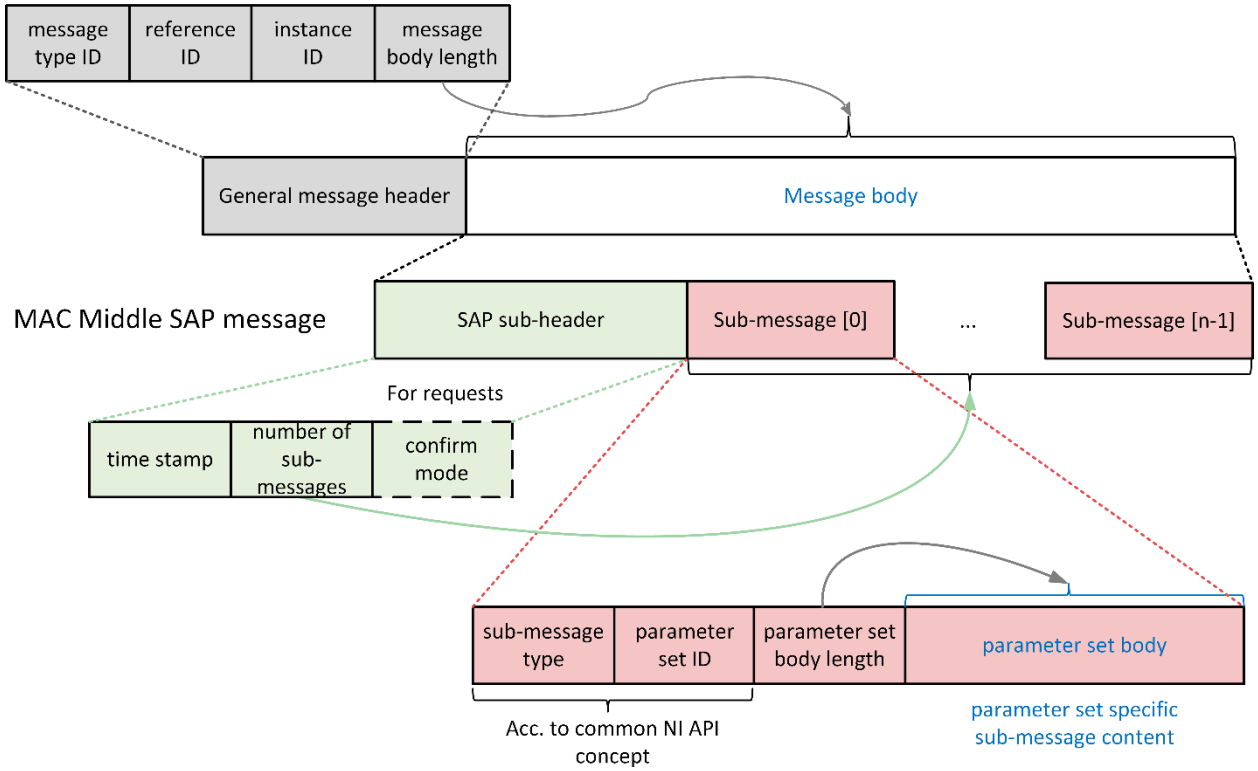

Figure 5-3: MAC Middle SAP—General Message Structure

Table 5-1. Supported Specific Message Type IDs

Message name	Message ID (binary) b15 … b0	Message ID (hex)
MAC SAP TX CONFIG REQ	0101 0001 0000 0001	0x5101
MAC SAP TX PAYLOAD REQ	0101 0001 0000 0010	0x5102
MAC SAP TX CNF	0101 0010 0000 0001	0x5201
MAC SAP TX STATUS IND	0101 0000 0000 0001	0x5001
MAC SAP RX CONFIG IND	0101 0000 1000 0001	0x5081
MAC SAP RX PAYLOAD IND	0101 0000 1000 0010	0x5082

5.3.1 General Message Header

Table 5-2 presents the contents of the general message header.

Table 5-2. General Message Header Contents

Field	Type/ Length	Description	32-bit Words
message type ID	U16	Message type identifier	2xU32

reference ID	U16	Reference identifier used for the following: • Mapping requests to the corresponding confirmation messages and special response indications. • Detection of lost indications or requests. For requests and common indications, the content must be generated by a message type specific counter which starts at zero and is incremented with every newly generated request or indication of the specific type. For confirmation messages, this field is filled with the same reference ID as used in the handled request. For the MAC SAP TX STATUS IND sent in response to a MAC SAP TX request, this field is filled with the reference ID of the corresponding MAC SAP TX CONFIG REQ.	
instance ID	U8	Instance identifier used to support parallel API connections of the same type. Different stations should use different instance IDs to debug potentially wrong UDP port assignments in multi-station setups. The instance ID is the station number.	
message body length	U24	Message body length in bytes.	

5.3.2 SAP Sub-Header

Table 5-3 presents the contents of the SAP sub-header.

Table 5-3. SAP Sub-header Contents

Field	Type/ Length	(Sup-ported) Value range	Description	32-bit Words
reserved	U16	0	Reserved field to be filled with zero.	2xU32
timestamp	U32		Timestamp: 0.1 µs tick count (wrapping counter) **Note:** The nominal resolution is aligned to the FPGA tick count implementation. The LabVIEW host SAP realization will use the	

			LabVIEW tick count which only provides an actual resolution of 1 µs (with some jitter uncertainty), even if the values are provided with the nominal 0.1 µs resolution at the interface.	
number of sub-messages	U8		Number of sub-messages contained in the hierarchical message.	
confirm mode [or **reserved**]	U8	{[0], 1}	Confirm mode (only used in requests) • 0: No confirmation required • 1: Standard confirmation required The application framework only supports `1`. Requests with `0` will be rejected. **Note:** In non-request messages, the field will be reserved and won't be evaluated or checked. It should be filled with `0`.	

5.3.3 Byte Order and Bit Order

The MAC Middle SAP message parameters are transmitted in network byte order, which is equal to big-endian byte order (the highest byte first while the lowest byte last). The network byte order has been selected to be neutral (platform independent) byte order, since the host byte order depends on hardware platform (CPU architecture) and/or operating system, which can be different between the hardware system running the application framework and the system running an external MAC high application. The operating system typically provides functions to convert from platform dependent host byte order to network byte order and vice versa. The bit order within bytes is based on the concept of most significant bit (MSB) first and least significant bit (LSB) last.

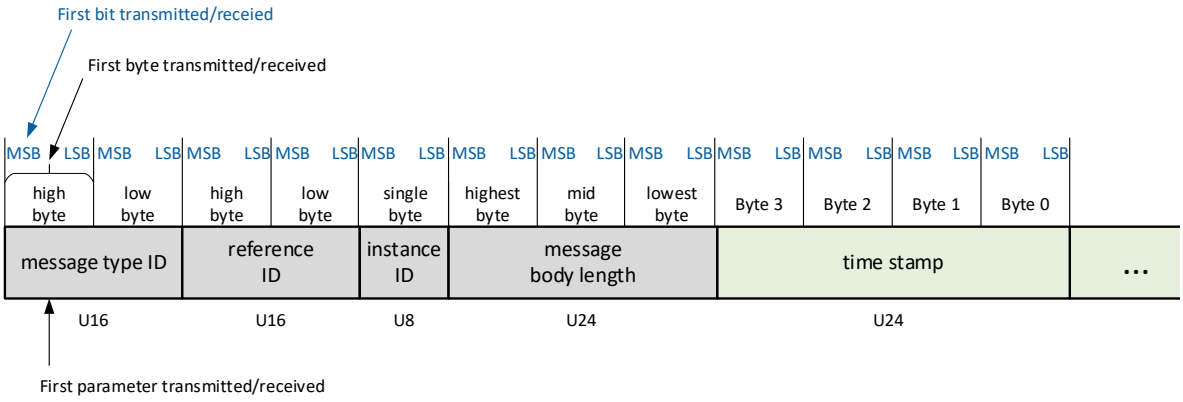

Figure 5-4: Illustration of Byte Order and Bit Order Defined for the Transmission of MAC Middle SAP Messages

5.3.4 MAC SAP TX CONFIG REQ

5.3.4.1 Overall Message

The MAC SAP TX CONFIG REQ requests a packet transmission. Table 5-4 lists the contents of the overall message. Two sub-messages MSDU TX parameters and PHY TX parameters are defined.

Table 5-4. Overall Message of MAC SAP TX CONFIG REQ

Field	Type/ Length	(sup-ported) Value range	Description	32-bit words
message type ID	U16	0x5101	Message type identifier	2xU32
reference ID	U16		See Section 5.3.1	
instance ID	U8		See Section 5.3.1	
message body length	U24		**message body length** in bytes = MSDU TX parameters (parameter set body length) + 4 + PHY TX parameters (parameter set body length) + 4 + 8 + 8	
reserved	U16		See Section 5.3.2	2xU32
timestamp	U32		See Section 5.3.2	
number of sub-messages	U8	2	For a single MSDU, the MAC SAP TX CONFIG REQ must always contain the following mandatory parameter sets: • MSDU TX parameters • PHY TX parameters	
confirm mode	U8	1	See Section 5.3.2	
For number of sub-messages (in MAC SAP TX CONFIG REQ)				
sub-message type	U8	{0, 1}	Sub-message type of MAC SAP TX CONFIG REQ 0: MSDU TX parameters (mandatory) 1: PHY TX parameters (mandatory)	U32
parameter set ID	U8	0	0	
parameter set body length	U16		Parameter set body length in bytes without the following fields: • sub-message type • parameter set ID • parameter set body length	
parameter set body	<parameter set body length>		Contents the specific sub-message according to the selected parameter set as it is shown in the following two subsections.	

5.3.4.2 Parameter Sets

5.3.4.2.1 MSDU TX Parameters

Table 5-5 presents the contents of the MSDU TX parameters sub-message.

Table 5-5. MSDU TX Parameters

Field		Type/ Length	(Sup-ported) Value range	Description	32-bit words
Sub-message/parameter set of MAC SAP TX CONFIG REQ					
	sub-message type	U8	0	Sub-message type of MAC SAP TX CONFIG REQ. 0: MSDU TX parameters 1: PHY TX parameters	U32
	parameter set ID	U8	0	0	
	parameter set body length	U16	41	Parameter set body length in bytes.	
	MSDU index	U8	0…255	Index to associate this configuration sub-message with the corresponding MSDU TX payload sub-message in the MAC SAP TX PAYLOAD REQ. It is incremented with every (new) MSDU TX parameters sub-message.	
	frame type	U8	{0,2}	Frame type as defined for Type field in frame control field of MAC header [1]. 0: management frames 1: control frames 2: data frames 3: extension frames **Note:** Only management frames and data frames are supported at the MAC middle SAP (TX side).	
	subtype	U8	Depends on frame type	Subtype as defined for **subtype** field in **frame control** field of MAC header [1]. Supported value range: 0…15: for all management frames (frame type = 0) 0: for only data frames of subtype data (frame type = 2)	
	to DS	U8	{0,1}	As defined for the corresponding **control** field in the MAC header [1].	

	from DS	U8	{0,1}	As defined for the corresponding **control** field in the MAC header [1].	
	power management	U8	0	See definition of related field in **frame control** field of MAC header [1]: 0: Indicates to active state mode 1: Indicates to power save (PS) mode In the application framework, only the active state is supported.	
	more data	U8	0	See definition of related field in **frame control** field of MAC header [1]. It is used by AP to indicate (more data ==1) to e.g. a station in PS mode that more MSDUs or MAC management protocol data unit (MMPDU) are buffered for that station at the AP. In the application framework, only the value of `0` is supported.	
	protected frame	U8	0	See definition of related field in frame control field of MAC header [1]. The value of `1` indicates that frame body is encrypted. In the application framework, only the value of `1` is supported, that is non-encrypted transmissions.	
	HTC / order	U8	0	See definition of related field in frame control field of MAC header [1]. It has the following meanings in dependence on the frame type, subtype, and TX vector (FORMAT): 1. For non QoS data frames: 0: Non-strictly-ordered service class 1: Strictly-ordered service class	

				2. For QoS data or management frames in HT/VHT format: 0: Frame does not contain HT control field 1: Frame contains HT control field In the application framework, only the value of `0` is supported.	
	source address (SA)	U48		Source station MAC address (individual/group bit must be `0` because it is checked by SAP)	
	destination address (DA)	U48		Destination station MAC address	
	basic service set identifier (BSSID)	U48		Basic service set identifier • Individual/group bit must be `0` • Universal/local bit set to `1` The application framework does not support wildcard BSSI, since this is represented by all binary `1`s.	
	recipient address (RA)	U48		Recipient station MAC address, only required or used if **from DS** is `1` and **to DS** is `1`	
	transmitter address (TA)	U48		Transmitter station MAC address, only required or used if **from DS** is `1` and **to DS** is `1`. Individual/group bit must be `0` since it is checked by SAP.	
	MSDU length	U16	0…4065	MSDU length in bytes	

5.3.4.2.2 PHY TX Parameters

Table 5-6 presents the contents of the PHY TX parameters sub-message. In addition, Table 5-7 shows the supported combinations of TX PHY parameters.

Table 5-6. PHY TX Parameters

Field	Type/ Length	(Sup-ported) Value range	Description	32-bit words
Sub-message/parameter set of MAC SAP TX CONFIG REQ				

	sub-message type (parameters set)	U8	1	Sub-message type of MAC SAP TX CONFIG REQ 0: MSDU TX parameters 1: PHY TX parameters	U32
	parameter set ID (parameter set version)	U8	0	0: For 802.11 Application Framework	
	parameter set body length	16	4	Parameter set body length in bytes	
	MSDU index	U8	0…255	Index to associate this configuration sub-message with the corresponding MSDU TX payload sub-message in the MAC SAP TX PAYLOAD REQ. It is incremented with every PHY TX parameters sub-message	

	format	U8	0…2	Target PHY format. It combines the TX/RX vector parameters **FORMAT** and **NON_HT_MODULATION** as defined in *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1], Table 21-1, as follows:
				• 0: NON_HT_OFDM
				  ○ **FORMAT—**NON_HT
				  ○ **NON_HT_MODULATION—**OFDM). Supported for 20 MHz bandwidth only
				• 1: NON_HT_DUP_OFDM
				  ○ **FORMAT**—NON_HT
				  ○ **NON_HT_MODULATION**—NON_HT_DUP_OFDM
				• 2: VHT
				  ○ **FORMAT**—VHT)	

	bandwidth	U8	0…2	Channel bandwidth. The supported bandwidths are as follows: 0: 20 MHz 1: 40 MHz 2: 80 MHz 3: 160 MHz (not supported) 4: 80 MHz + 80 MHz (not supported) Supported bandwidth depends on the format selection, see Table 5-7.	
	MCS	U8	0…9	Interpretation depends on format. If **format** is 2 (VHT), the MCS can be between 0 and 9, see Tables 21-30, 21-38, and 21-48 in *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY)*	

				Specifications [1]. If format is not 2, the MCS is mapped to L_DATARATE as follows: 0: 6 Mbit/s 1: 9 Mbit/s 2: 12 Mbit/s 3: 18 Mbit/s 4: 24 Mbit/s 5: 26 Mbit/s 6: 48 Mbit/s 7: 54 Mbit/s Supported range depends on format and bandwidth selection.	

-Table 5-7: Supported Combinations of TX PHY Parameters

Bandwidth / **Format**	20 MHz: 0	40 MHz: 1	80 MHz: 2	160 MHz: 3	80+80MHz: 4
NON_HT_OFDM: 1	MCS 0…7	Not supported	Not supported	Not supported	Not supported
NON_HT_DUP_OFDM: 2	MCS 0…7	MCS 0…7	MCS 0…7	Not supported	Not supported
VHT: 2	MCS 0…8	MCS 0…9	MCS 0…4	Not supported	Not supported

5.3.5 MAC SAP TX PAYLOAD REQ

5.3.5.1 Overall Message

The MAC SAP TX PAYLOAD REQ provides the payload data (MSDU) associated with the MAC SAP TX CONFIG REQ. Table 5-8 lists the contents of the overall message. The payload is contained in the MSDU TX payload parameter set.

Table 5-8. Overall message of MAC SAP TX Payload request

Field	**Type/ Length**	**Value range**	**Description**	**32-bit words**
message type ID	U16	**0x5102**	message type identifier	2xU32
reference ID	U16		See Section 5.3.1	
instance ID	U8		See Section 5.3.1	
message body length	U24		Message body length in bytes = MSDU TX payload (parameter set body length) + 4 + 8 + 8	
reserved	U16	0	See Section 5.3.2	2xU32
timestamp	U32		See Section 5.3.2	
number of sub-messages	U8	**1**	See Section 5.3.2 In the application framework, one MAC SAP TX PAYLOAD REQ mandatorily contains	

			exactly one MSDU TX payload sub-message providing the payload for a single MSDU only.	
confirm mode	U8	**1**	See Section 5.3.2	
For number of sub-messages (in MAC SAP TX PAYLOAD REQ)				
sub-message type (parameters set)	U8	**0**	Sub-message type defined for MAC SAP TX PAYLOAD REQ 0: MSDU TX payload **(mandatory)**	U32
parameter set ID (parameter set version)	U8	**0**	0: It contents type "direct data".	
parameter set body length	U16		Parameter set body length in bytes without the following fields: • sub-message type • parameter set ID • parameter set body length	
Parameter set body	<parameter set body length>		It contents the specific sub-message according to the selected parameter set as it is shown in Section 5.3.5.2.	

5.3.5.2 Parameter Sets

5.3.5.2.1 MSDU TX Payload

Table 5-9 presents the contents of the MSDU TX payload sub-message.

Table 5-9. MSDU TX Payload

Field	Type/ Length	(Sup-ported) Value range	Description	32bit words
Sub-message/parameter set of MAC SAP TX PAYLOAD REQ				
sub-message type (parameters set)	U8	0	Sub-message type defined for MAC SAP TX PAYLOAD REQ 0: MSDU TX payload	U32
parameter set ID (parameter set version)	U8	0	0: It contents type direct data	
Parameter set body length	U16		**parameter set body length** in bytes = MSDU length + 4	
MSDU index	U8	0…255	Index to associate the MSDU data contained in this sub-message to the corresponding configuration sub-messages contained in the SAP TX CONFIG REQ	U32

	reserved	U8	0	Reserved field to ensure 32-bit alignment	
	MSDU length	U16	0…4065	MSDU length in byte without padding bytes	
	MSDU data[5]	<MSDU length> x U8		MSDU data	

5.3.6 MAC SAP TX CNF

5.3.6.1 Message Definition

The MAC SAP TX Confirmation is the response to a MAC SAP TX CONFIG REQ and the associated MAC SAP TX PAYLOAD REQ. It contains the confirmation status which indicates if the request was handled successfully or otherwise which kind of error occurred. Table 5-10 lists the contents of the overall message.

Table 5-10. Overall Message of MAC SAP TX CNF

Field	Type/ Length	(sup-ported) Value range	Description	32bit words
message type ID	U16	0x5201	Message type identifier	2xU32
reference ID	U16		Filled with reference ID extracted from the incoming request for which the confirmation is generated	
instance ID	U8		See Section 5.3.1	
message body length	U24		Message body length in bytes = MAC confirmation (**parameter set body length**) + 4 + 8 + 8	
reserved	U16		See Section 5.3.2	2xU32
timestamp	U32		See Section 5.3.2	
number of sub-messages	U8	1	See Section 5.3.2. In the application framework, only one parameter set is defined for this message, which is mandatory.	
reserved	U8	0	Filled with value 0, see Section 5.3.2	
For number of sub-messages in MAC SAP TX CNF				
sub-message type (parameters set)	U8	0	Sub-message type of MAC SAP TX CNF 0: MAC TX confirmation (mandatory)	U32

[5] If MSDU data alignment to words wider than U8 (1 byte) is needed for the physical transport medium, this has to be ensured by the modules (VIs) supporting the access to the specific physical transport medium by **adding zero padding bytes** at the end of the message byte stream.

	parameter set ID (parameter set version)	U8	0	0: For 802.11 Application Framework	
	parameter set body length	U16	1	Parameter set body length in bytes	
	confirmation status	U8		(Common) confirmation status see Section 5.3.6.2	

5.3.6.2 Confirmation status

For every MAC TX request message, a parameter check is done to validate if the request and its parameters are supported, and if the requests are received in the right consequence. Furthermore, a check for any mismatch is executed. Table 5-11 shows the description of the possible cases that the reported confirmation status to MAC high can have.

Table 5-11. Confirmation Status Description

Value	Name	Description
0	SUCCESS	Request accepted and sent to the subsequent module; no error occurred during handling of the request.
1	UNKNOWN MESSAGE	The primitive (**message type ID**) is unknown or undefined.
2	MESSAGE NOT SUPPORTED	The primitive (**message type**) is defined but not supported at the specific interface.
3	UNKNOWN PARAMETER (SET)	One or more of the received parameter sets or parameters are not part of the message.
4	MISSING PARAMETER (SET)	One or more of the (mandatory) parameter sets or parameters of the received message are missing.
5	PARAMETER (SET) REPETITION	One or more of the received parameter sets or parameters have been set more than once.
6	RANGE VIOLATION	One or more parameters are out of the supported range, that is, <confirm mode> is in allowed range, only requests with confirm mode == 1 are supported.
7	STATE VIOLATION	The received request is not supported in the current state of operation. This indicates, for example, any violation of the allowed message sequence protocols, or specific configurations which are not allowed due to previously configured settings or configurations.
8	TIMEOUT	An expected message has not been received within the specific period after a previous event (**message**).
9	CONFIG PAYLOAD MISMATCH	The configuration part of a request does not fit to the payload part, for example, a mismatch of MSDU indices or MSDU lengths.
10	LENGTH MISMATCH	The content of any **message body length** or **parameter set body length** indicator field does not match to the actual message body length or parameter set body length.

11	INPUT BUFFER FULL	Request rejected since the input buffer for incoming requests is full. The subsequent module needs some time to execute the previously stored requests.
12	INTERNAL ERROR	Internal error.
13	Instance ID Mismatch	The local instance ID is not like the decoded instance ID from MAC high.

5.3.7 MAC SAP TX STATUS IND

5.3.7.1 Message definition

The MAC SAP TX Status Indication informs about the status of a previously requested transmission. Table 5-12 lists the contents of the overall message.

Table 5-12. Overall Message of MAC SAP TX STATUS IND

Field	Type/ Length	Value range	Description	32-bit words
message type ID	U16	0x5001	Message type identifier	2xU32
reference ID	U16		See Section 5.3.1. It is filled with the reference ID of the corresponding MAC SAP TX CONFIG REQ.	
instance ID	U8		See Section 5.3.1.	
message body length	U24		**message body length** in bytes = MAC TX status (**parameter set body length**) + 4 + 8 + 8	
reserved	U16		See Section 5.3.2	2xU32
timestamp	U32		See Section 5.3.2	
number of sub-messages	U8	1	See Section 5.3.2. For the application framework, only one parameter set is defined for this message, which is mandatory.	
reserved	U8		Filled with value 0	
For number of sub-messages (in MAC SAP TX STATUS IND)				
sub-message type (parameters set)	U8	0	Sub-message type of MAC SAP TX STATUS IND 0: MAC TX status (mandatory)	U32
parameter set ID (parameter set version)	U8	0	0: For 802.11 Application Framework	
parameter set body length	U16	1	**parameter set body length** in bytes	
MSDU transmission status	U8	{0,1}	0: MSDU successfully transmitted 1: MSDU discarded (retry limit reached)	

5.3.8 MAC SAP RX CONFIG IND

5.3.8.1 Overall Message

MAC SAP RX Config Indication informs about a received packet. Table 5-13 lists the contents of the overall message. Four parameter sets are defined which specify the MSDU and MHY parameters, the status of the retry flag and the status of all MAC RX modules.

Table 5-13. Overall message of MAC SAP RX CONFIG IND

Field	Type/ Length	Value range	Description	32-bit words
message type ID	U16	0x5181	Message type identifier	2xU32
reference ID	U16		See Section 5.3.1	
instance ID	U8		See Section 5.3.1	
message body length	U24		**message body length** in bytes = MSDU RX parameters (**parameter set body length**) + 4 + PHY RX parameters (**parameter set body length**) + 4 + RX Status (**parameter set body length**) + 4 + 8 + 8	
reserved	U16		See Section 5.3.2	2xU32
timestamp	U32		See Section 5.3.2	
number of sub-messages	U8	4	See Section 5.3.2. In the application framework, a MAC SAP RX CONFIG REQ contains the following parameter sets (mandatory) for a single MSDU: • MSDU RX parameters • PHY RX parameters • RX status • Additional MSDU RX parameters	
reserved	U8		Filled with value 0	
For number of sub-messages (in MAC SAP RX CONFIG IND)				
sub-message type (parameters set)	U8	{0…3}	Sub-message type of MAC SAP RX CONFIG IND: 0: MSDU RX parameters 1: PHY RX parameters 2: RX Status 3: Additional MSDU RX parameters	U32
parameter set ID (parameter set version)	U8	0	0: In 802.11 Application Framework	
parameter set body length	U16		**parameter set body length** in bytes (without fields of **sub-**	

				message type, **parameter set ID**, or **parameter set body length**)	
	parameter set body	<parameter set body length>		Contents the specific sub-message according to the selected parameter set.	

5.3.8.2 Parameter Sets

5.3.8.2.1 MSDU RX Parameters

Table 5-14 presents the contents of the MSDU RX parameters sub-message.

Table 5-14. Sub-message contents of MSDU RX parameters

Field		Type/ Length	(Sup-ported) Value range	Description	32-bit words
Sub-message/parameter set of MAC SAP RX CONFIG IND					
	sub-message type (parameters set)	U8	0	Sub-message type of MAC SAP RX CONFIG IND 0: MSDU RX parameters 1: PHY RX parameters 2: RX Status 3: Additional MSDU RX parameters	U32
	parameter set ID (parameter set version)	U8	0	0: For 802.11 Application Framework	
	parameter set body length	U16	41	Parameter set body length in bytes	
	parameter set body			Identical parameter set body as defined for MSDU TX parameters, see Section 5.3.4.2.1. Supported parameter value ranges might be extended, since RX is potentially not as restrictive as TX (sniffer mode, and so on). **Note:** No range value check is applied at MAC SAP RX side.	

5.3.8.2.2 PHY RX Parameters

Table 5-15 presents the contents of the PHY RX parameters sub-message.

Table 5-15. Sub-Message Contents of PHY RX Parameters

Field	Type/ Length	(Sup-ported) Value range	Description	32-bit words
Sub-message/parameter set of MAC SAP RX CONFIG IND				

	sub-message type (parameters set)	U8	1	Sub-message type of MAC SAP RX CONFIG IND 0: MSDU RX parameters 1: PHY RX parameters 2: RX Status 3: Additional MSDU RX parameters	U32
	parameter set ID (parameter set version)	U8	0	0: In 802.11 Application Framework	
	parameter set body length	U16	4	Parameter set body length in bytes	
	parameter set body			Identical parameter set body as defined for PHY TX parameters, see Section 5.3.4.2.2 **Note:** No range value check applied at MAC SAP RX side.	

5.3.8.2.3 Additional MSDU RX Parameters

This parameter set is used to provide additional MSDU RX configuration parameters at the MAC middle SAP RX side, which are not included in the (common) MSDU TX/RX parameters, but might be useful later. It serves also as an example for you to know how to provide additional parameters at the interface by means of an initial parameter set. Table 5-16 presents the contents of the additional MSDU RX parameters sub-message.

Table 5-16. Sub-Message Contents of Additional MSDU RX Parameters

Field	Type/ Length	(Sup-ported) Value range	Description	32bit words
Sub-message/parameter set of MAC SAP RX CONFIG IND				
sub-message type (parameters set)	U8	3	Sub-message type of MAC SAP RX CONFIG IND 0: MSDU RX parameters 1: PHY RX parameters 2: RX Status 3: Additional MSDU RX parameters	U32
parameter set ID (parameter set version)	U8	0	0: In 802.11 Application Framework	
parameter set body length	U16	1	Parameter set body length in bytes	
Retry flag	U8	{0,1}	Retransmission flag 0: MSDU received after initial transmission	

				1: MSDU received after re-transmission	

5.3.8.2.4 MAC RX Status

Table 5-17 presents the contents of the MAC RX status sub-message.

Table 5-17. Sub-Message Contents of MAC RX Status

Field	Type/ Length	(Supported) Value range	Description	32bit words
Sub-message/parameter set of MAC SAP RX CONFIG IND				
sub-message type (parameters set)	U8	2	Sub-message type of MAC SAP RX CONFIG IND 0: MSDU RX parameters 1: PHY RX parameters 2: MAC RX Status 3: Additional MSDU RX parameters	U32
parameter set ID (parameter set version)	U8	0	0: In 802.11 Application Framework	
parameter set body length	U16	7	**Parameter set body length** in bytes	
MSDU duplicate detection status	U8	0,…,2	0: Duplicate detection not executed 1: Duplicate detection successfully passed 2: Duplicate detected	
MSDU defragmentation status	U8	0,…,2	0: Defragmentation not executed 1: Unfragmented MSDU successfully received 2: Unsupported MSDU fragment received	
MPDU Filtering status	U8	0,…,2	0: MPDU Filtering not executed 1: MPDU valid for further processing 2: MPDU pass through for sniffer mode, which leads to extension for sniffer mode	
MPDU disassembly status	U8	0,…,3	0: Disassembly not executed 1: Disassembly successful 2: Disassembly done partially 3: Disassembly not successful	
MPDU length check status	U8	0,…,3	0: MPDU length check not executed 1: MPDU length check pass	

				2: MPDU length exceeds implemented buffer 3: MPDU length not standard compliant	
	MPDU FCS check status	U8	0,…,2	0: FCS check not executed 1: FCS check pass 2: FCS check fail	
	A-MPDU De-Aggregation status	U8	0,…,5	0: A-MPDU de-aggregation not executed 1: A-MPDU de-aggregation successful 2: MPDU length mismatch 3: End of file (EOF) field check fail 4: VHT delimiter signature comparison fail 5: VHT delimiter CRC fail	

5.3.9 MAC SAP RX PAYLOAD IND

5.3.9.1 Overall Message

The MAC SAP RX Payload Indication provides the payload (MSDU) data associated with the MAC SAP RX Config Indication. Table 5-18 lists the contents of the overall message. The payload is contained in the MSDU RX payload parameter set.

Table 5-18. Overall Message of MAC SAP RX Payload IND

Field	Type/ Length	Value range	Description	32bit words
message type ID	U16	0x5082	Message type identifier	2xU32
reference ID	U16		See Section 5.3.1	
instance ID	U8		See Section 5.3.1	
message body length	U24		**message body length** in bytes = MSDU RX payload (parameter set body length) + 4 + 8 + 8	
reserved	U16		See Section 5.3.2	2xU32
timestamp	U32		See Section 5.3.2	
number of sub-messages	U8	1	See Section 5.3.2. In the application framework, one MAC SAP RX PAYLOAD IND mandatorily contains one MSDU RX payload sub-message providing the payload for a single MSDU only.	
reserved	U8		Filled with value zero, see Section 5.3.2	
For number of sub-messages (in MAC SAP RX PAYLOAD IND)				
sub-message type (parameters set)	U8	0	Sub-message type defined for MAC SAP RX PAYLOAD IND 0: MSDU RX payload (mandatory)	U32

	parameter set ID (parameter set version)	U8	0	0: In 802.11 Application Framework, the content type direct data	
	parameter set body length	U16		Parameter set body length in bytes (without the fields **sub-message type**, **parameter set ID**, and **parameter set body length**)	
	parameter set body	<parameter set body length>		Contents the specific sub-message according to the selected parameter set.	

5.3.9.2 Parameter Sets

5.3.9.2.1 MSDU RX Payload

The MSDU RX payload has the same definition of MSDU TX payload presented in Table 5-9 of Section 5.3.5.2.1.

5.4 MAC Middle SAP Modules

5.4.1 MAC SAP TX Request Confirmation Handler

As shown in Figure 5-1, the MAC SAP TX Request Confirmation Handler polls input queue for MAC SAP TX request message pairs and relies on and checks the specified message sequence (MAC SAP TX CONFIG REQ before MAC SAP TX PAYLOAD REQ). In addition, it decodes the received TX request messages and runs parameter and consistency checks. If checks fail, it generates a negative MAC SAP TX CNF messages. Otherwise, it prepares the MAC MSDU TX REQ and calls the MAC middle TX. Then, it waits for the MAC MSDU TX CNF from MAC Middle and generates the (final) MAC SAP TX CNF messages. Figure 5-5 shows the logical state machine for TX request and confirmation handler. The validation of incoming MAC SAP TX requests and the generation of parameter sets for MAC MSDU TX REQ are described in Section 5.4.1.1 and Section 5.4.1.2, respectively.

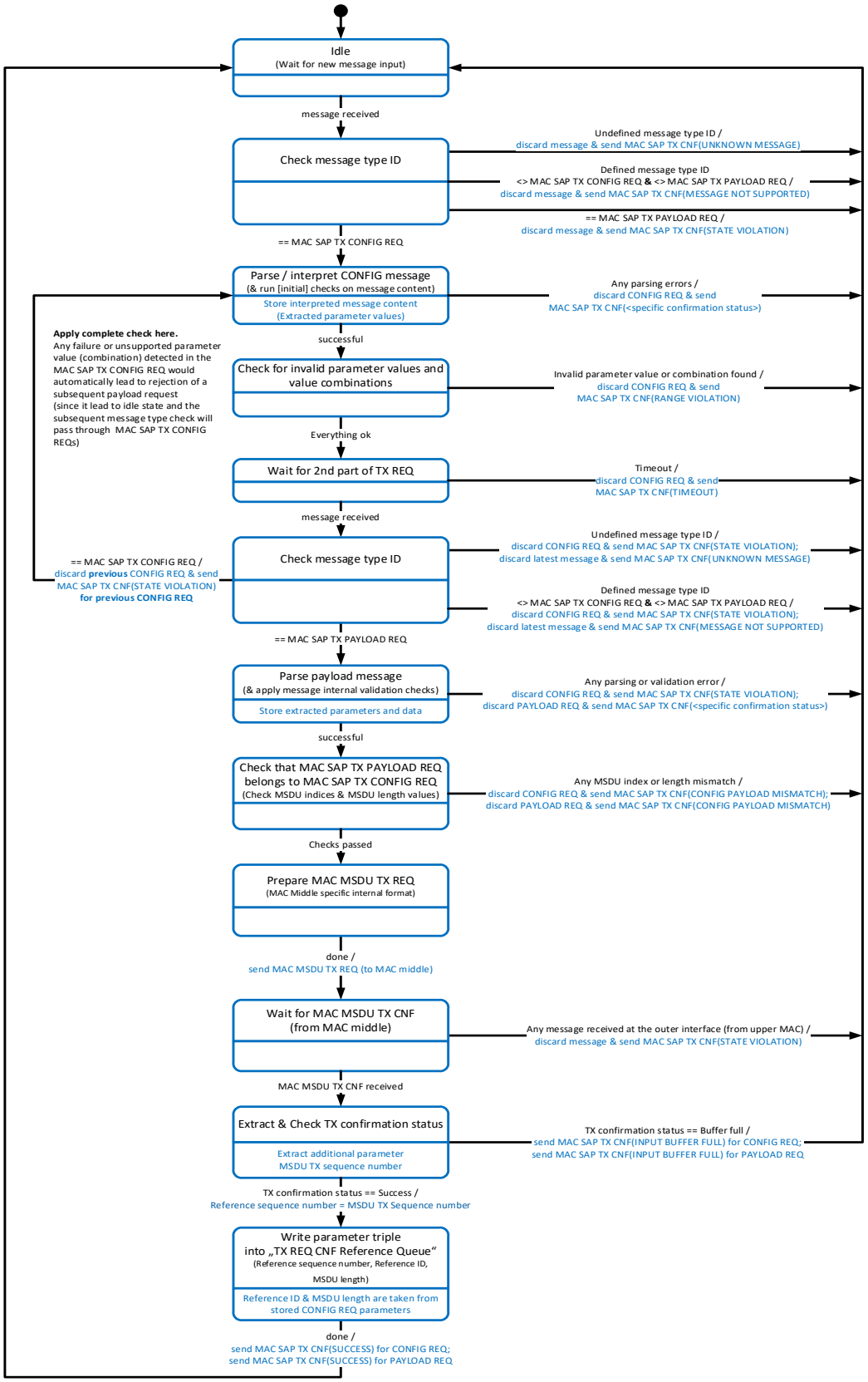

Figure 5-5: MAC Middle SAP—Logical State Machine for TX Request and Confirmation Handling

5.4.1.1 Decode and Validation of Incoming MAC SAP TX Requests

A decode and validation check are applied for every incoming MAC SAP TX request, that is, every incoming message pair MAC SAP TX CONFIG REQ and MAC SAP TX PAYLAOD REQ. The received 1D array in bytes of MAC SAP TX Configuration Request message is decoded to the following required subsets: General Message Header, SAP Sub Header, MSDU TX parameters and PHY TX parameters.

In addition, the received 1D array in bytes of MAC SAP TX Payload Request message is decoded to the following required subsets: General Message Header, SAP Sub Header, and MSDU TX payload. After the decoding process of each sub-message, a validation check for all parameters is executed if they are supported or there is a mismatch. Whenever a validation check fails, the following actions are applied:

1. The related message (or message pair) is discarded/rejected, see the state machine in Figure 5-5.
2. MAC SAP TX CNF message is generated per discarded message with a related negative confirmation status as it is described in Table 5-11.
3. A more detailed logging message is generated.

Besides the common message validation checks presented in Table 5-11, additional checks are executed on MAC SAP TX CONFIG REQ on MSDU TX and PHY TX parameters as shown in Table 5-19.

Table 5-19. Additional Checks on MAC SAP TX CONFIG REQ

Validation Condition (Pass Criteria)		**(Negative) Confirmation Status if Check is Not Passed.**
Checks applied on specific parameters contained in **MSDU TX parameters**		
	source address (SA) is an individual address (that is, not a group address; the individual/group bit has to be zero), see section 9.2.4.3 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].	RANGE VIOLATION
	basic service set identifier (BSSID) is an individual address and a locally administered address (that is, the universal/local bit is set to 1), see section 9.2.4.3.4 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].	RANGE VIOLATION
	transmitter address (TA) is an individual address if **to DS** is `1` and **from DS** is `1`. **Note:** This check is only applied when **to DS** = **from DS** = 1, since this is the only case where the transmitter address will be evaluated at the MAC Middle SAP.	RANGE VIOLATION
Checks applied on specific parameters contained in **PHY TX parameters**		
	bandwidth is within the supported range for the requested format, see Table 5-6.	RANGE VIOLATION
	MCS is within the range supported for the requested format and bandwidth combination, see Table 5-7.	RANGE VIOLATION

5.4.1.2 Generation of Parameter Sets for MAC MSDU TX REQ

5.4.1.2.1 Functional Description

It gets the validated parameter values decoded from the received MAC SAP TX REQ pair (MAC SAP TX CONFIG REQ and MAC SAP TX Payload REQ) and sets the corresponding parameters in the MAC MSDU TX REQ to the extracted values. It gets the MSDU data extracted from MAC SAP TX PAYLOAD REQ → MSDU TX Payload → MSDU data. Then, it provides the payload data in the MAC internal format as it is required for the MAC MSDU TX REQ, see Figure 5-1.

In addition, it determines receiver address (RA) and transmitter address (TA) from the destination address (DA), source address (SA), and BSSID based on the **to DS** and **from DS** settings. The mapping for data frames is shown in Section 9.3.2.1 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1] (Table 9-26). The mapping for management frames is defined in Section 9.3.3.2 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1] as follows: RA = DA and TA = SA (independent on **to DS** and **from DS**).

The inputs of parameter sets generation of MAC MSDU TX request module are as follows:

- MAC SAP TX configuration request including MSDU TX Parameters and PHY TX Parameters.
- MAC SAP TX payload request

The outputs of parameter sets generation of MAC MSDU TX request module are as follows:

- MAC TX request including MPDU configuration and TX vector.
- MSDU Data: Payload to be transferred to the target MAC TX.

5.4.1.3 Generation of MAC MSDU TX Confirmation

A MAC SAP TX CNF message is generated for each message received at the outer MAC Middle SAP TX interface. The confirmation message can be one of the following:

- **Positive confirmation** in normal operation case: When both parts of a MAC SAP TX request, that is, the MAC SAP TX CONFIG REQ and the corresponding MAC SAP TX PAYLOAD REQ, have been validated based on the following criteria:
  - Successfully received in the right order
  - Successfully decoded
  - Checked to be valid[6]

[6] This includes the check of all received parameter values to be in the supported range, parameter value combinations to be supported, and that MAC SAP TX CONFIG REQ and MAC SAP TX PAYLOAD REQ matches.

    - Successfully transferred from the MAC Middle to MAC low through the Host-to FPGA interface. In this case **2 positive confirm messages** will be generated, one for the MAC SAP TX CONFIG REQ and one for the MAC SAP TX PAYLOAD REQ.
- **Negative confirmation** in any of the following error cases:
    - The reception of any message at the outer (TX) SAP interface during the time period between a successful reception of a complete (and valid) MAC SAP TX request and the generation (transmission) of the corresponding MAC SAP TX CNF.
    - When an error occurs during parsing/interpreting the received message, for example, unknown message type ID, unsupported parameter ranges, and so on.
    - When a MAC SAP TX PAYLOAD REQ is received, but no corresponding MAC SAP TX CONFIG REQ has been received directly before.
    - When a MAC SAP TX CONFIG REQ is received directly after another MAC SAP TX CONFIG REQ instead of the expected MAC SAP TX PAYLOAD REQ. In this case, a negative confirm message is generated for the previous MAC SAP TX CONFIG REQ. The new MAC SAP TX CONFIG REQ is accepted and the module continues waiting for the MAC SAP TX PAYLOAD REQ.
    - When any of the received parameter values or value combinations are not supported by the application framework.
    - When the related MAC internal TX request could not be transferred to the MAC low on the FPGA due to full DMA buffer state.

5.4.1.4 Write TX Request Reference Parameters to Queue

If the MAC MSDU TX request derived from both parts of a MAC SAP TX request has been successfully transferred to FPGA, the corresponding reference ID and MSDU length are stored in a reference queue as shown in Figure 5-1. They will be used in the Generate MAC SAP TX status indication module.

5.4.2 Generate MAC SAP TX Status Indication

5.4.2.1 Functional Description

It receives the MAC MSDU TX End IND from MAC Middle TX and reads out the oldest element from TX REQ CNF Reference Queue to determine the reference ID for the MAC SAP TX STATUS IND. It generates the MAC SAP STATUS IND. Then, it calls the transport layer function(s) to send the generated indication message to MAC High Abstraction. The logical state machine of MAC SAP TX Status IND is presented in Figure 5-6.

It is worth mentioning that if a valid MAC MSDU TX end indication has been read from MAC middle, the reference queue of TX request confirmation should already have elements. Based on that, an element is read from the reference queue of TX request

confirmation to get the reference ID. A MAC SAP TX STATUS IND will be issued when a MAC SAP TX request is completed, whether successfully or not, at MAC level. The message includes the instance ID, given from the station configuration, reference ID, and MSDU transmission status. After the encoding process of MAC SAP TX STATUS IND sub-messages, it is written to the corresponding UDP port.

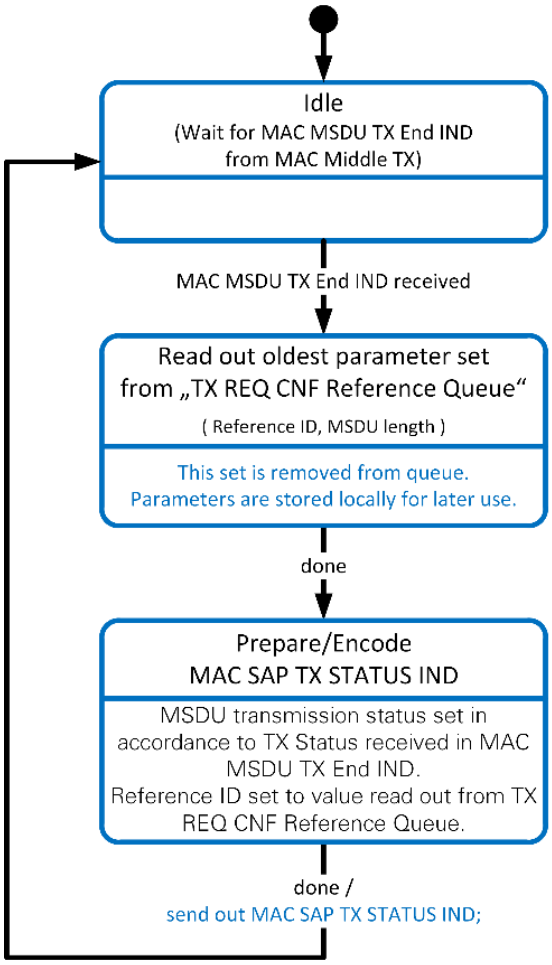

Figure 5-6: MAC Middle SAP—Logical State Machine for Generation of MAC SAP TX STATUS IND

5.4.2.1.1 Inputs

The input interface of this module has the following items:

- FPGA interface: Interface to the device's FPGA handle.
- TX request confirmation reference queue: Queue reference of TX request confirmation that stores clusters of two elements (reference ID and MSDU length).
- Instance ID of station.
- UDP connection: Reference to the UDP socket used to send the TX Status Indication message.

- UDP port: Reference to the UDP socket used to send the MAC SAP TX Status Indication.

5.4.3 802.11 Generate MAC SAP RX Indication

5.4.3.1 Functional Description

It receives MAC MSDU RX IND from MAC Middle RX; then it generates the corresponding MAC Middle SAP RX indication pair (MAC SAP RX CONFIG IND and MAC SAP RX PAYLOAD IND). The generation process is a kind of mapping, where the MSDU RX indication message is mapped to the sub-messages of MAC SAP RX indication that has the following sub-messages: MSDU RX parameters, PHY RX parameters, Additional MSDU RX parameters, and MAC RX status. The payload is mapped to MAC SAP RX Payload indication message. In addition, the MSDU index is created. After the encoding process of MAC SAP RX STATUS IND sub-messages, it calls the transport layer function(s) to send the generated indication messages to MAC High RX Abstraction, where the messages are sent in the following order: first the MAC SAP RX CONFIG IND, and then the MAC SAP RX PAYLOAD IND.

A MAC SAP RX indication is issued whenever an MSDU or MMPDU has been received and successfully processed at MAC level; no special checks have been applied at the MAC middle SAP. In normal mode, only MSDUs/MMPDUs successfully and completely processed will be indicated by the MAC RX to the MAC middle SAP. Figure 5-7 shows the module structure and program flow.

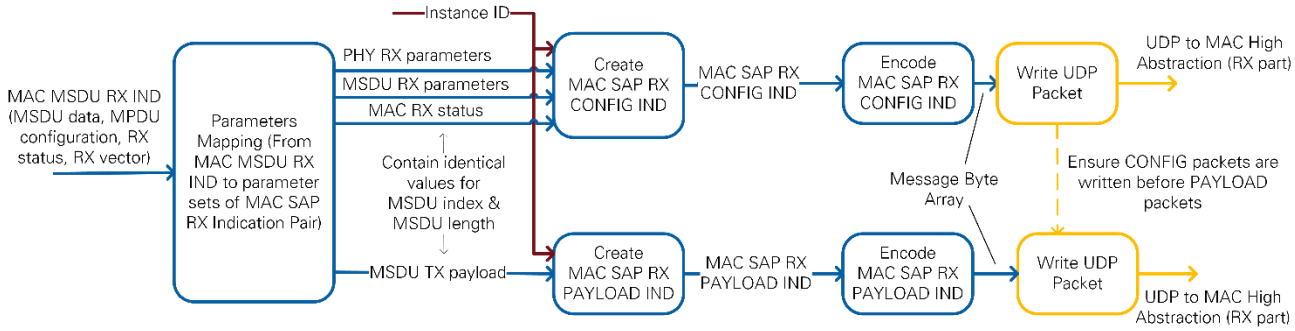

Figure 5-7: Generate MAC SAP RX Indications—Module Structure and Program Flow

5.4.3.2 Inputs

The input interface of this module has the following items:

- FPGA interface: Interface to the device's FPGA handle.
- Queue reference to store statistical information about the payload (time stamp and size).
- Queue reference to the UDP socket used to send the RX Indication messages (MAC SAP RX configuration indication and MAC SAP RX payload indication).
- Instance ID of the corresponding station.

5.4.4 Example of Decode and Encode Messages

The encoding and decoding process of messages between MAC high and MAC middle SAP has been described previously. The concept is the same. To show the process, the MAC SAP TX Configuration Request decoding and MAC SAP RX Configuration Indication encoding will be used as an example.

5.4.4.1 Decode MAC SAP TX Configuration Request

It decodes the given 1D array (in bytes) of MAC SAP TX Configuration Request message. An error message is reported if the value of one decoded parameter is not expected. Besides the decoding of the required subsets (General Message Header, SAP Sub Header, MSDU TX parameters and PHY TX parameters), the module checks the parameters of all sub-messages if they are supported or if there is mismatch as explained in Section 5.4.1.1.

The process is summarized as follows:

1. Decode the general message header, see Figure 5-3 and Section 5.3.4.
    a. Check the General Message Header.
        i. Report error if there is a mismatch in the body length (Length Mismatch).
        ii. Report error if the message ID is not expected (State Violation).
2. Decode the SAP Sub Header, see Figure 5-3 and Section 5.3.4.
    a. Check the SAP Sub Header.
        i. Report error if the number of sub-messages is not like the expected value (Range Violation).
        ii. Report error if the confirmation mode derived from the message ID is not similar to the expected value of confirmation mode (Range Violation).
3. Check the header of every sub-message.
    a. Report error if the number of sub-messages is not like the expected value (Range Violation).
    b. Report error if the body length is not similar to the expected value (Length mismatch).
4. Decode the MSDU TX Parameters.
    a. Check every parameter and report an error if it is not supported or there is a mismatch.
5. Decode the PHY TX Parameters.
    a. Check every parameter and report an error if it is not supported or there is a mismatch.
6. Check if any sub-message is duplicated or missing and report the appropriate error.
    - Duplicated: Report error (Parameter (Set) Repetition)
    - Missing: Report error (Missing Parameter (Set))

5.4.4.2 Encode MAC SAP RX Configuration Indication

It encodes the MAC SAP RX Configuration Indication message to 1D array of bytes. From Section 5.3.8, the message has the following sub-messages (General Message Header, SAP Sub Header, MSDU RX Parameters, PHY RX Parameter, Additional MSDU RX Parameters, and MAC RX Status). The process is presented in Figure 5-8, where there are six nodes to encode each sub-message.

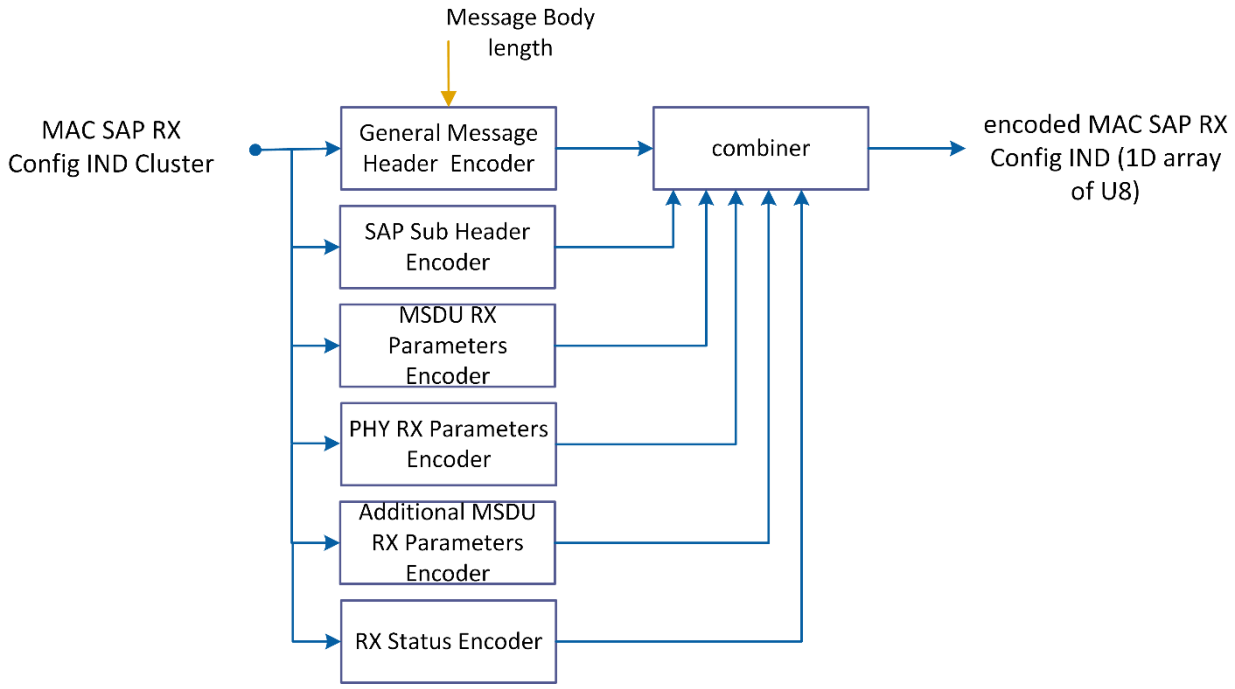

Figure 5-8: Code Structure of MAC SAP RX Configuration Indication Message Encoder

6 MAC Layer

A high-level overview of the MAC is shown in Figure 6-1. It shows the top-level modules of MAC TX and MAC RX chains that are provided to get the supported MAC features presented in Section 2.2. In addition, it shows the MAC part that has been implemented in the host and the other part that has been implemented in the FPGA. The MAC components are classified as follows:

- **Upper MAC**: The higher MAC functionalities are not implemented. However, several interfaces are provided by the Middle MAC to allow for the transmission and reception of packets. As it is described in Section 2.1, the user can use the High MAC stub provided by the host application or an external MAC implementation, where the external MAC implementation interfaces are conducted with the Middle MAC SAP directly. The interface is described in Chapter 5.
- **MAC Middle SAP:** Provides the interface between the supported MAC middle functionalities and external upper MAC applications as described in Chapter 5.
- **MAC TX:** The top modules are partitioned based on the target into two sets:

    - **Host Module:** The MSDU Sequence Number assignment is implemented in the host to assign a sequence number for each incoming MSDU/MMPDU.
    - **FPGA Modules:** The MAC TX chain includes the following main functions: MSDU Fragmentation, Frame Sequence Selection, MPDU Retransmission Control, Frame Sequence TX Control, MPDU Generation, and A-MPDU Aggregation. The functionalities of each node are presented in Section 6.3.
- **MAC RX:** The top modules are partitioned based on the target into two sets:
    - **Host Module:** The Duplicate Detection module is implemented in the host to check whether the received MSDU/MMPDUU (Data or Management frame) is duplicated.
    - **FPGA Modules:** The MAC RX chain contains the following MAC RX functions: A-MPDU De-Aggregation, MPDU Disassembly and FCS Check, MPDU Filtering, and MSDU De-Fragmentation. The functionalities of each node are presented in Section 6.4.
- **DCF Control:** This top-level module realizes the DCF described in Section 10.3 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. The DCF functionalities are presented in Section 6.5.1.
- **Host / FPGA Communication:** The ICP is used to transfer data from host to FPGA and vice versa.

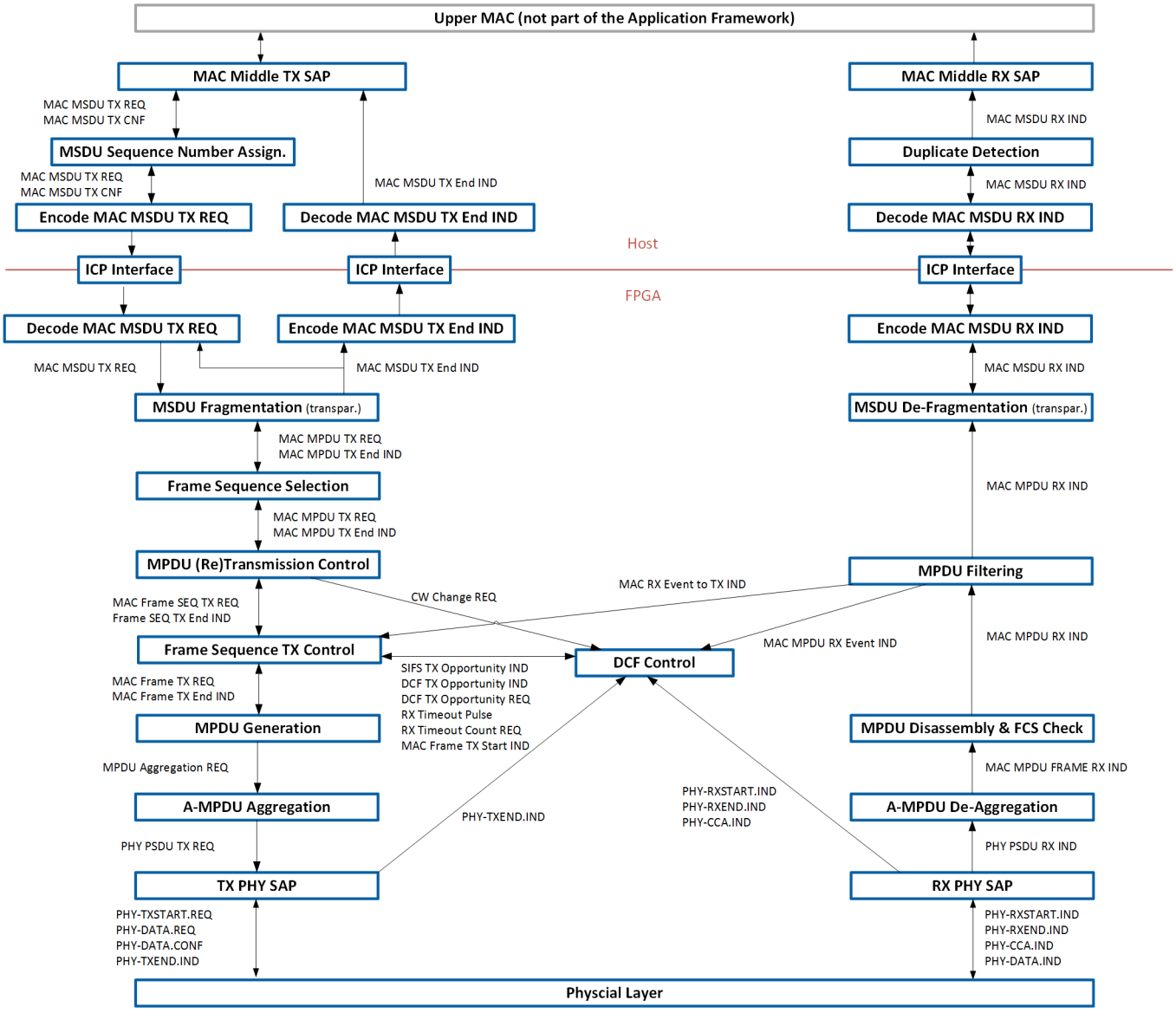

Figure 6-1: MAC High Level Block Diagram

6.1 MAC Interfaces

The interface exposed by the MAC is described in Section 5.1.2. It is used by the Middle MAC SAP, which is described in Chapter 5.

6.2 ICP Interface

As discussed in Section **Error! Reference source not found.**, the lower part of the MAC is implemented on the FPGA, and the middle MAC is implemented on the host. The ICP serves as the interface between the two. The general format of the protocol is described in Section 3.2.

6.2.1 FIFOs

The following H2T and T2H FIFOs are used to transfer the requests and indications from host to FPGA or from FPGA to host:

- H2T TX Data: Transfers MAC MSDU TX REQ from host to FPGA
- T2H TX Feedback: Transfers MAC MSDU TX End IND from FPGA to host
- T2H RX Data: Transfers MAC MSDU RX IND from FPGA to host

6.2.2 Encoding and Decoding Functions

The following messages are transferred through the ICP interface between the MAC middle implemented in the host and MAC low implemented in the FPGA:

- MAC MSDU TX Request: Transferred from host to FPGA
- MAC MSDU TX End Indication: Transferred from FPGA to host
- MAC MSDU RX Indication: Transferred from FPGA to host

For each message, an encoder and decoder have been implemented. The encoder converts the corresponding message from a cluster to 1D array of U8 while the decoder converts the given message in 1D array of U8 to the corresponding cluster.

To show the concept, the encoders and decoders of MAC MSDU TX Request message and MAC MSDU RX Indication will be presented as an example.

6.2.2.1 MAC MSDU TX Request Encoder and Decoder

From Figure 6-1, the MAC MSDU TX Request message is encoded to 1D array of U8 in the host after MSDU sequence number assignment. The encoding process is as follows:

1. Get the following message sub-clusters: MPDU configuration and TX vector.
2. For MPDU configuration cluster, get the following three sub-clusters: frame control, header contents, and parameters.
3. Convert each sub-message to 1D array of U8. There is a module for each sub-message. The following considerations have been considered:
    - If the parameter is wider than one byte, it is transmitted in Little Endian Byte order (the lowest byte first while the highest byte last).
    - If the parameter is fixed point (FXP), it is converted to binary, and then to the proper number of bytes.
    - If only one binary parameter is available per sub-message, it is converted to one byte.
    - If the parameter is binary, it is collected with the other binary parameters in the same sub-message into one byte.
    - If the parameter is an enum value, it is mapped to U8.

Figure 6-2 shows the module components of the MAC MSDU TX request message encoder in the host. The decoder in the FPGA reverses the operation of the encoder in the host to get the MAC MSDU TX Request cluster.

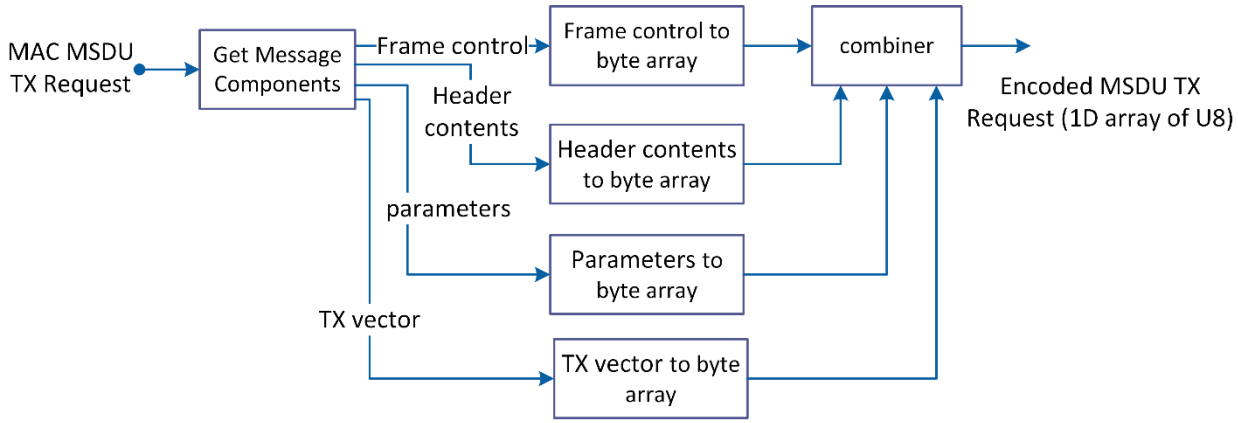

Figure 6-2: MAC MSDU TX Request Message Encoder in Host

6.2.2.2 MAC MSDU RX Indication Encoder and Decoder

From Figure 6-1, the MAC MSDU RX Indication message is encoded to 1D array of U8 in the FPGA. The encoding process is as follows:

1. Get the message sub-clusters: RX vector, RX status, and MPDU configuration.
2. For MPDU configuration cluster, get the three sub-clusters: frame control, header contents, and parameters.
3. Convert each sub-message to 1D array of U8 considering the described considerations presented above. Figure 6-3 shows the module components of the MAC MSDU RX Indication message encoder in the FPGA. The decoder in the host reverses the operation of the encoder in the FPGA to get the MAC MSDU RX Indication cluster.

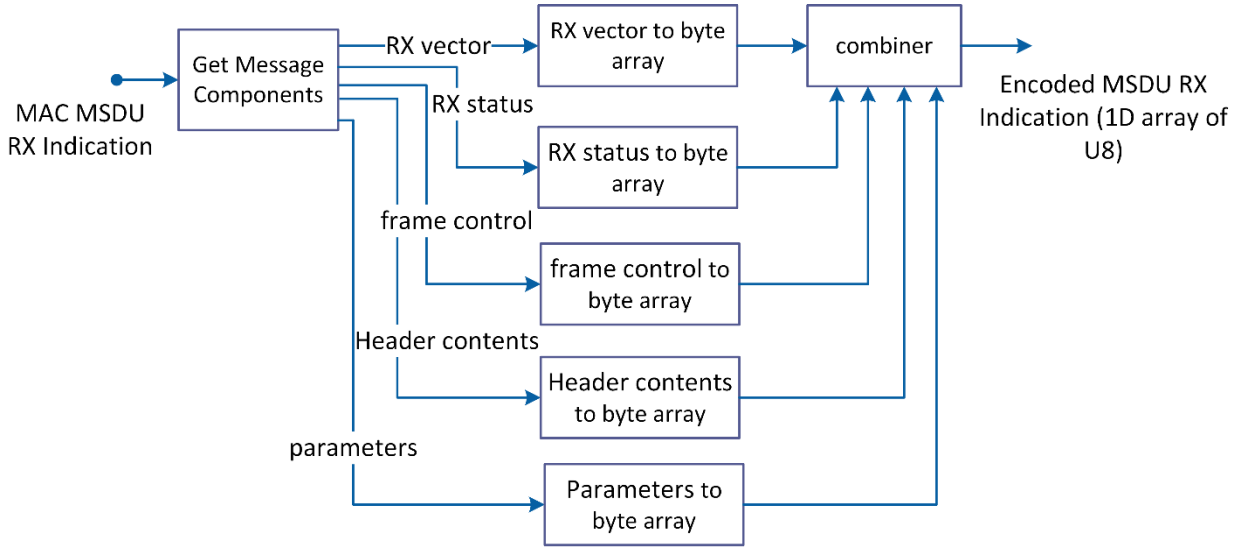

Figure 6-3: MAC MSDU RX Indication Message Encoder in FPGA.

6.3 MAC TX Modules

6.3.1 MSDU Sequence Number Assignment

This module assigns the sequence number as needed for the duplicate detection and recovery which is described in Section 10.3.2.11.2 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. The incoming MSDU/MMPDU may be re-transmitted multiple times by the MPDU (Re-)Transmission Control function using the exact same sequence number. This module maintains a sequence number cache to ensure that successive MSDUs/MMPDUs for the same RA are not assigned the same sequence number. The following state machine shown in Figure 6-4 gives an overview of the MSDE sequence number assignment.

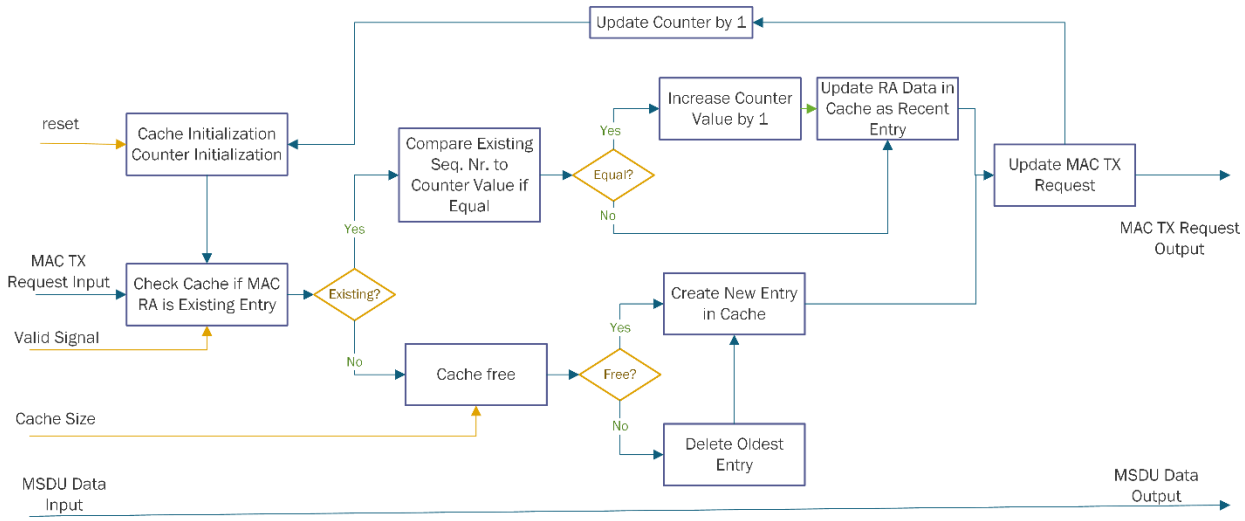

Figure 6-4: State Machine of MSDU Sequence Number Assignment

Inputs are as follows:

- MAC MSDU TX REQ (MSDU data, MPDU configuration, TX vector)
  - Interpreted parameters:
    - MPDU configuration: RA
- MAC MSDU TX CNF (TX confirmation status)
  - Not used for processing

Outputs are as follows:

- MAC MSDU TX REQ (MSDU data, MPDU configuration, TX vector)
  - Modified parameters:
    - MPDU configuration: sequence number
- MAC MSDU TX CNF (TX confirmation status)
  - Forwarded upon reception without change

6.3.2 MSDU Fragmentation

MSDU fragmentation as described in Section 10.2.7 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1] is not implemented.

Instead, the MSDU is passed unfragmented, that is, without modification. Hence, the following fixed parameters are set:

- **fragment number** = `0`
- **more fragments** = `false (0)`

The parameter **MPDU frame body length** is copied from **MSDU length**.

Inputs are as follows:

- MAC MSDU TX REQ (MSDU data, MPDU configuration, TX vector)
  - Interpreted parameters:
    - MSDU length
- MAC MPDU TX End IND (TX status, number of transmission tries)
  - Not used for processing

Outputs are as follows:

- MAC MPDU TX REQ (MPDU data, MPDU configuration, TX vector)
  - Modified parameters:
    - MPDU frame body length
    - Fragment number
    - More fragments
- MAC MSDU TX End IND (TX status, number of transmission tries)
  - Forwarded upon reception without change

6.3.3 Frame Sequence Selection

Frame Sequence selection is described in Section 10.3.6 and 10.3.5 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. This module defines which frame sequence from the following frame sequences is used for the requested transmission:

- {RTS | CTS | DATA | ACK}
- {DATA | ACK}
- {DATA}

Furthermore, the module sets the TX Vector parameters **PSDU length** and **(A-)MPDU length** dependent on the MPDU frame body length and the frame type and subtype.

The {RTS | CTS | DATA | ACK} sequence is used for packets which are longer than a certain threshold which is defined by the parameter dot11RTSThreshold. In this case, RTS and CTS control frames are sent before the actual data transmission.

The {DATA | ACK} sequence is used if the dot11RTSThreshold is not reached.

The {DATA} sequence is used instead of the sequences listed above if one of the following conditions is met:

- **Recipient address** is a group address and **to DS** is `false (0)`

- **type** is `Management` and **subtype** is `Action No Ack`

Retransmissions are not possible for this sequence because of the missing ACK transmission.

Inputs are as follows:

- MAC MPDU TX REQ (MPDU data, MPDU configuration, TX vector)
  - Specific parameters required for actual processing
    - MPDU configuration: Recipient address, MPDU frame body length, frame type, from DS, to DS
  - TX vector
    - format
- MAC MPDU TX End IND (TX status, number of transmission tries)
  - Not used for processing

Outputs are as follows:

- MAC MPDU TX REQ (MPDU data, MPDU configuration, TX vector)
  - Modified parameters:
    - MPDU Configuration: frame sequence type, MPDU length
    - TX vector: PSDU length, (A-)MPDU length
- MAC MPDU TX End IND (TX status, number of transmission tries)
  - Forwarded upon reception w/o change

6.3.4 MPDU (Re-)Transmission Control

This module handles recovery procedures as described in Section 10.3.4.4 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. It triggers an initial transmission and, if necessary, retransmissions until one of the retry limits is reached. It handles the retry counters (SRC, LRC, SSRC, SLRC) and sets the retry flag accordingly. In detail, the following tasks are performed:

- It receives the MAC MPDU TX request from the Frame Sequence Selection module (upstream module) and stores a single incoming MPDU data and the associated MPDU configuration and TX vector in a memory (data is forwarded to replay buffer).
- It generates the MAC MPDU TX end indication to signal (un)successful transmission of the MPDU to the Frame Sequence module (upstream module).
- It generates the (re)transmission request MAC frame sequence TX request for the current MPDU to the Frame Sequence TX Control module (downstream module).
- It receives and evaluates the Frame sequence TX end indication for the requested frame sequence transmission received from the Frame Sequence TX Control module (downstream module).

- It generates signals to increase or reset the contention window parameter, depending on the (un)successful transmission of an MPDU (contention window change request). This signal is consumed by the DCF control module.
- It discards MPDU data if one of the retry limits is reached (dot11ShortRetryLimit, dot11LongRetryLimit).

The inner state machine is shown in Figure 6-5. It maintains four retransmission counters with according increment and reset flags:

- SRC: short (frame sequence) retransmission counter
- LRC: long (frame sequence) retransmission counter
- SSRC: (global) STA short retransmission counter (for contention window handling)
- SLRC: (global) STA long retransmission counter (for contention window handling)

It contains the following states:

- Idle: wait for incoming indications.
- Handle MPDU TX request: generate MAC MPDU TX request (retry: False) for initial transmission.
- Handle short frame: increment short frame retransmission counters (increment SRC, increment SSRC for data frames).
- Handle long frame: increment long frame retransmission counters (increment LRC, increment SLRC for data frames, reset SSRC).
- Set contention window: reset contention window if retry limit was reached; otherwise increase contention window in case of data frames.
- Retry or discard: generate MAC MPDU TX request (retry: True) if retry limit was not yet reached; otherwise generate MAC MPDU TX end indication (TX status: MPDU discarded), reset SRC, reset LRC and clear MPDU.
- Frame sequence completed: generate MAC MPDU TX end indication (TX status: MPDU transmitted), reset counters, reset contention window, clear MPDU.
- Error: this state is entered upon detecting an error state (for example, new MAC MPDU TX request was received before the previous one was completely handled), do not accept/generate further requests/indications or data.

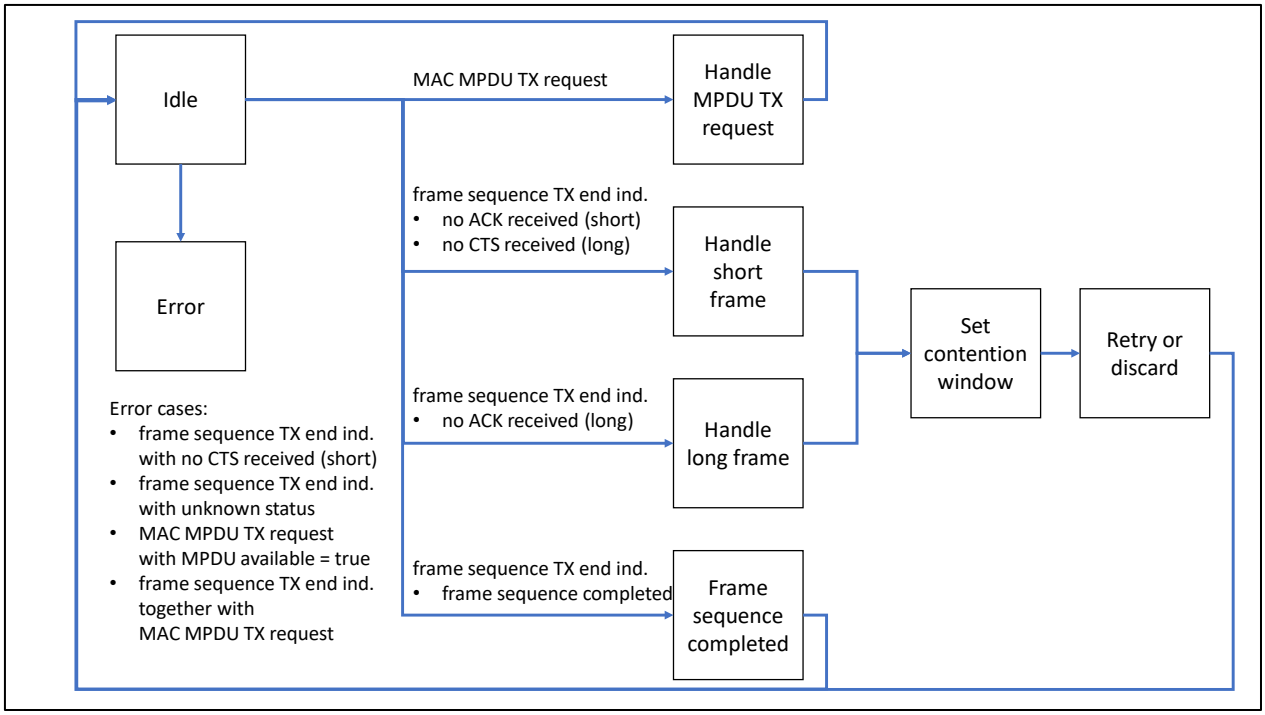

Figure 6-5: State Machine of MPDU (Re-)Transmission

Inputs are as follows:

- MAC MPDU TX REQ (MPDU data, MPDU configuration, TX vector)
  - Interpreted parameters:
    - MPDU data
    - MPDU configuration: MPDU length, frame sequence type, frame type
    - TX vector
- Frame SEQ TX End IND (Frame SEQ TX status)
  - Interpreted parameters:
    - Frame SEQ TX status (frame sequence completed, no CTS received, no ACK received)

Outputs are as follows:

- MAC Frame SEQ TX REQ (MPDU data, reset replay buffer, MPDU configuration, TX vector)
  - Modified parameters:
    - MPDU data
    - Reset replay buffer
    - MPDU Configuration: retry flag
    - TX vector
- MAC MPDU TX End IND (TX status, number of transmission tries)
  - Whole primitive is generated here:
    - TX status (MPDU transmitted, MPDU discarded)

        - number of transmission tries
- CW Change REQ (Continuous wave (CW) action)
    - Modified parameters (all parameters are generated here):
        - CW action: reset, increase CW

6.3.5 Frame Sequence TX Control

This module controls and requests the transmission of MAC frames. Packet transmissions are caused by one of the following scenarios shown in Figure 6-6:

- As part of the requested TX frame sequence
- As response to received MAC frames indicated by the MAC RX Event to TX IND

In the first scenario, the sequence depends on the requested sequence type in the following ways:

- In case of the {RTS | CTS | DATA | ACK} sequence, a RTS packet is generated first. The DATA (or Management) packet is only generated if a CTS packet was successfully received. The sequence is considered successful if an ACK packet was received successfully.
- In case of the {DATA | ACK} sequence, a DATA (or Management) packet is generated. The sequence is considered successful if an ACK packet was received successfully.
- In case of the {DATA} sequence, only a DATA (or Management) packet is generated. The sequence is always considered successful. However, the module waits for the MPDU generation to finish (indicated by a MAC frame TX end indication) before it generates a frame sequence TX end indication.

In the second scenario, the module needs to react to MAC frames that are received either in idle mode or while waiting for a DCF opportunity. An RTS packet is acknowledged by sending a CTS packet. A DATA (or Management) packet is acknowledged by sending an ACK packet. Sending a response has priority over handling a requested TX frame sequence.

The module sets the duration field according to the requested TX frame sequence. Also, it sets the TX vector parameters in one of the following ways:

- Copies the TX vector from the incoming MAC Frame SEQ TX REQ for DATA (or Management) frame types.
- Sets it to predefined static configurations for RTS, CTS and ACK frames. This configuration determines the subcarrier format and the MCS. The default configuration is 20 MHz, non-HT subcarrier format, and MCS 0.

For determining the right time for starting transmission, it communicates with the DCF module. It generates a DCF TX opportunity request upon reception of a MAC frame sequence TX request from the MPDU (Re-)Transmission Control. The DCF module

produces indications for a DCF TX opportunity and a SIFS TX opportunity indication. Figure 6-7 shows the state machine of frame sequence TX control module.

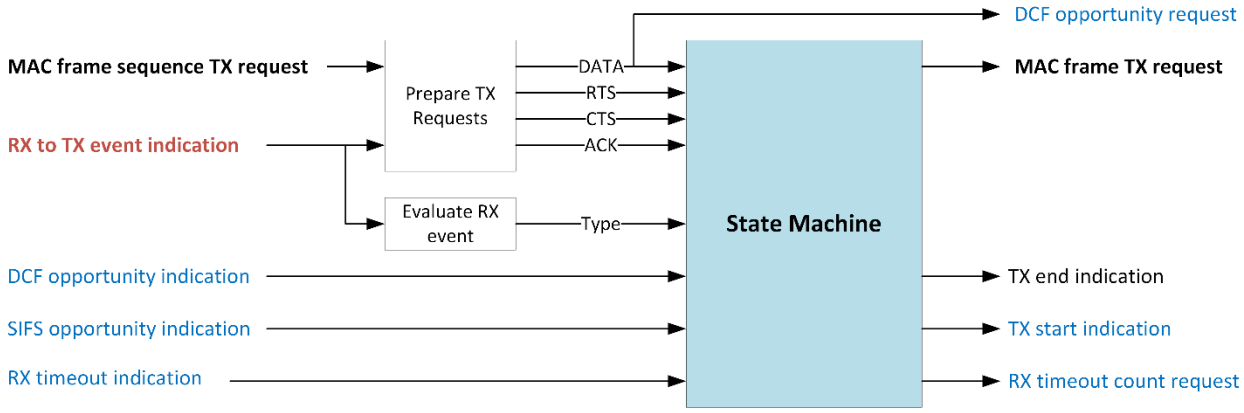

Figure 6-6: Overview on Frame Sequence TX Control

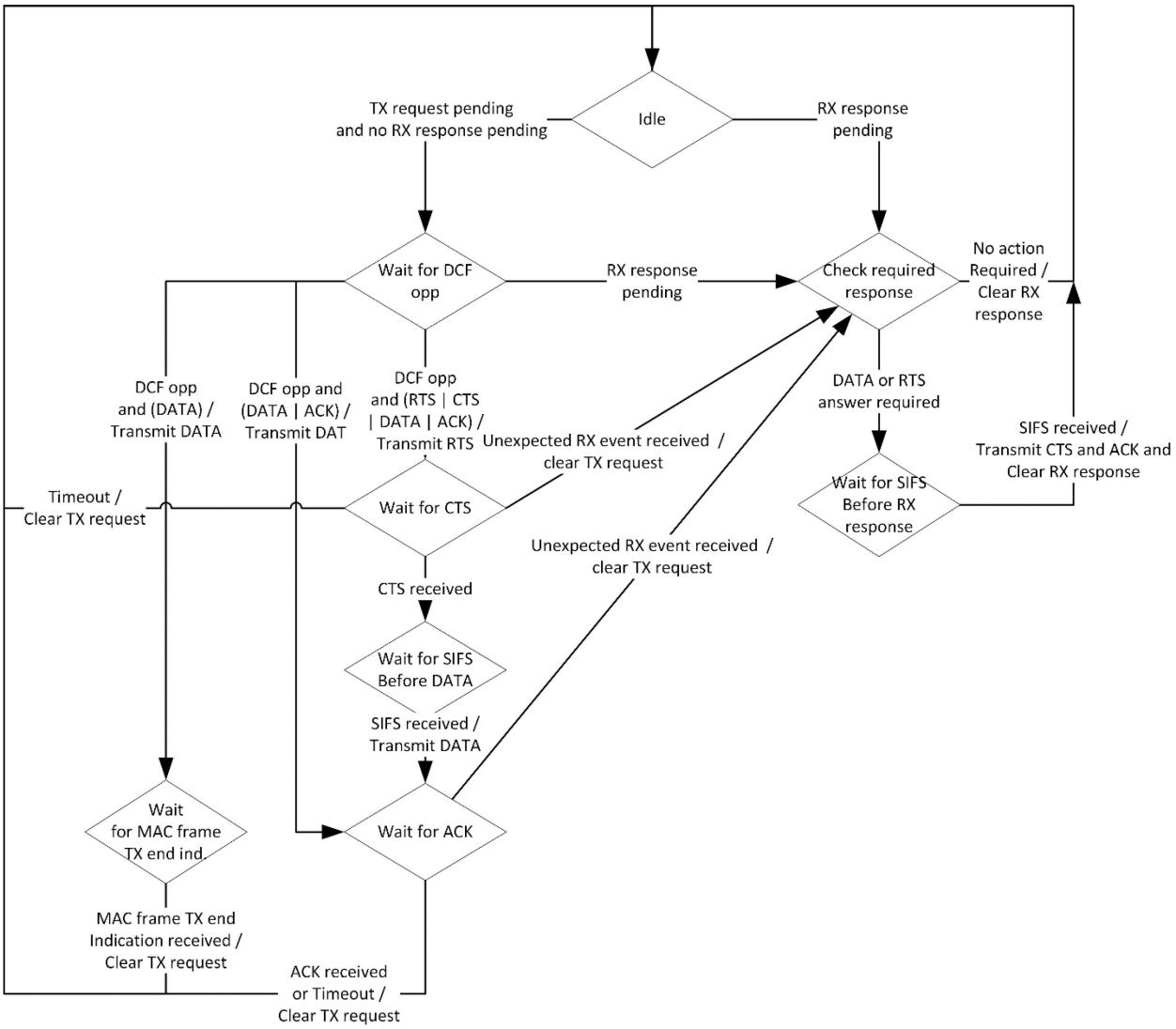

Figure 6-7: State Machine of Frame Sequence TX Control

The inputs are as follows:

- MAC Frame SEQ TX REQ (MPDU configuration, TX vector)
  - Interpreted parameters: all
- MAC Frame TX End IND (no parameters)
- MAC RX Event to TX IND (received MPDU parameters)
  - Interpreted parameters: all
- DCF TX Opportunity IND (no parameters)
- SIFS TX Opportunity IND (NAV busy, MAX TX bandwidth)
  - Interpreted parameters: all
- RX Timeout Pulse (no parameters)

The outputs are as follows:

- MAC Frame TX REQ (MPDU configuration, TX vector)
  - Modified parameters:
    - MPDU configuration
    - TX vector
- Frame SEQ TX End IND (Frame SEQ TX status)
  - Whole primitive is generated here
- DCF TX Opportunity REQ (target TX bandwidth)
  - Whole primitive is generated here
- RX Timeout Count REQ (timeout type)
  - Whole primitive is generated here
- MAC Frame TX Start IND (no parameters)
  - Whole primitive is generated here

6.3.6 MPDU Generation

This function assembles the MAC frame composed of MAC header, frame body and FCS field. The existence of fields depends on the MPDU configuration, for example, on frame type and subtype. The MPDU structure is discussed in Section 2.2.2.

Only the following frame types are handled:

- Control Frames: RTS, CTS, ACK (other Control frames are handled similar to ACK)
- Data Frames: no restrictions
- Management Frames: the address 3 field for management frames does always use the BSSID. Special cases described in 9.3.3.2 *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1] are not handled.
  Note: BSSID is valid in most cases

The block diagram is shown in Figure 6-8. The MAC Frame Header Generator generates the MAC header based on the parameters provided in the MAC Frame TX REQ. When the header was completely generated, the state machine creates a Frame Body request which reads the frame body from the replay buffer. The FCS Add submodule completes

the MPDU by adding the FCS field after MAC header and MPDU frame body. The MPDU Aggregation request is generated for the A-MPDU Aggregation module (downstream module) and a MAC Frame TX End indication is generated for the Frame Sequence TX Control module (upstream module).

The block diagram of the MAC Frame Header Generator is shown in Figure 6-9. It contains a state machine which sends trigger signals to the Field Generator submodules. They generate the content of the different fields and send a done flag as soon as the field was completely generated. Upon receiving this flag, the state machine transitions to the next state. When all fields are generated, it generates a done flag which is consumed by the state machine of the MPDU Generation module.

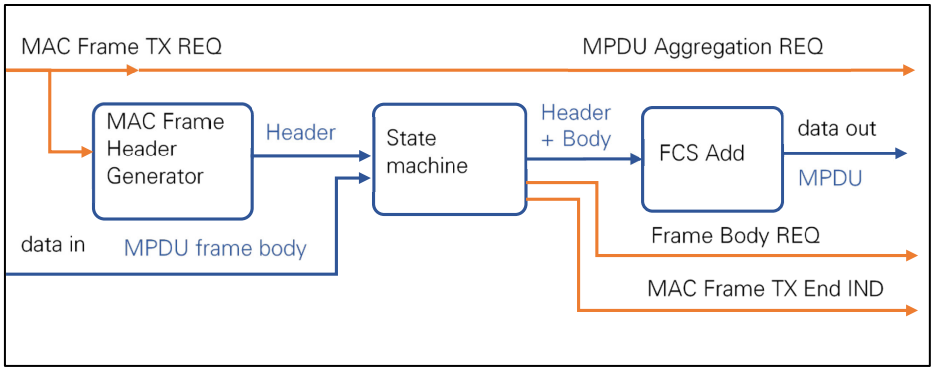

Figure 6-8: Block Diagram of MPDU Generation

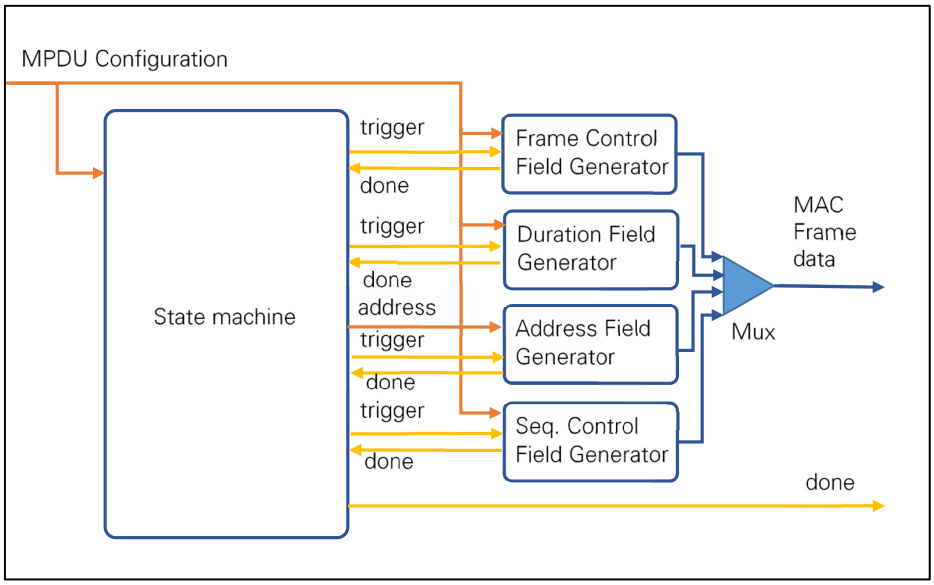

Figure 6-9: Block Diagram of MAC Frame Header Generator

The inputs are as follows:

- MAC Frame TX REQ (MPDU data, MPDU configuration, TX vector)
  - Interpreted parameters:
    - MPDU data

        - MPDU configuration: all elements

The outputs are as follows:

- MPDU Aggregation REQ (MPDU, MPDU configuration, TX vector)
    - Modified parameters:
        - MPDU
- MAC Frame TX End IND (no parameters)

6.3.7 A-MPDU Aggregation

For VHT frame formats, this module performs A-MPDU aggregation as described in Section 9.7 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. The general A-MPDU format is shown in Figure 6-10. The A-MPDU subframe format is presented in Figure 6-11. Note that the implementation only supports a single MPDU (n=1).

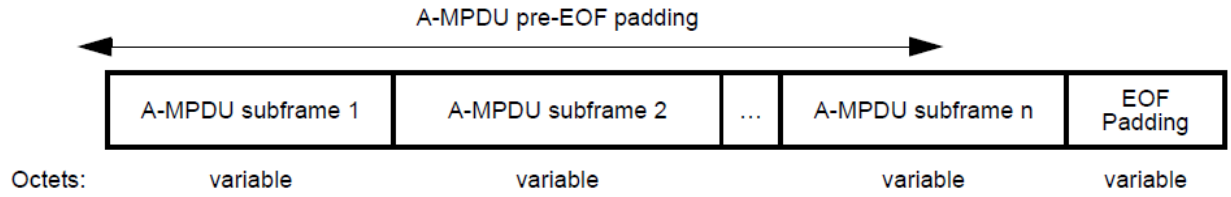

Figure 6-10: A-MPDU Format (Figure 9-741 in [1])

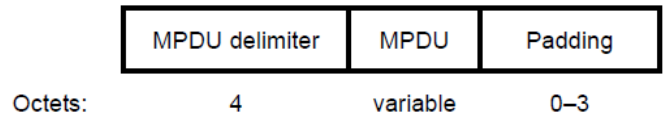

Figure 6-11: A-MPDU Subframe Format (Figure 9-743 in *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1])

Overall, the following steps are executed when aggregating an A-MPDU:

1. MPDU delimiter generation
    a. Field computation
    b. CRC computation
    c. Delimiter assembly
2. Insertion of the MPDU
3. Padding generation (for the single MPDU subframe)
4. EOF padding generation (EOF Padding Subframes and EOF Padding Octets)

For non-HT frame format, the MPDU is passed without modification. Figure 6-12 gives an overview of the A-MPDU aggregation state machine.

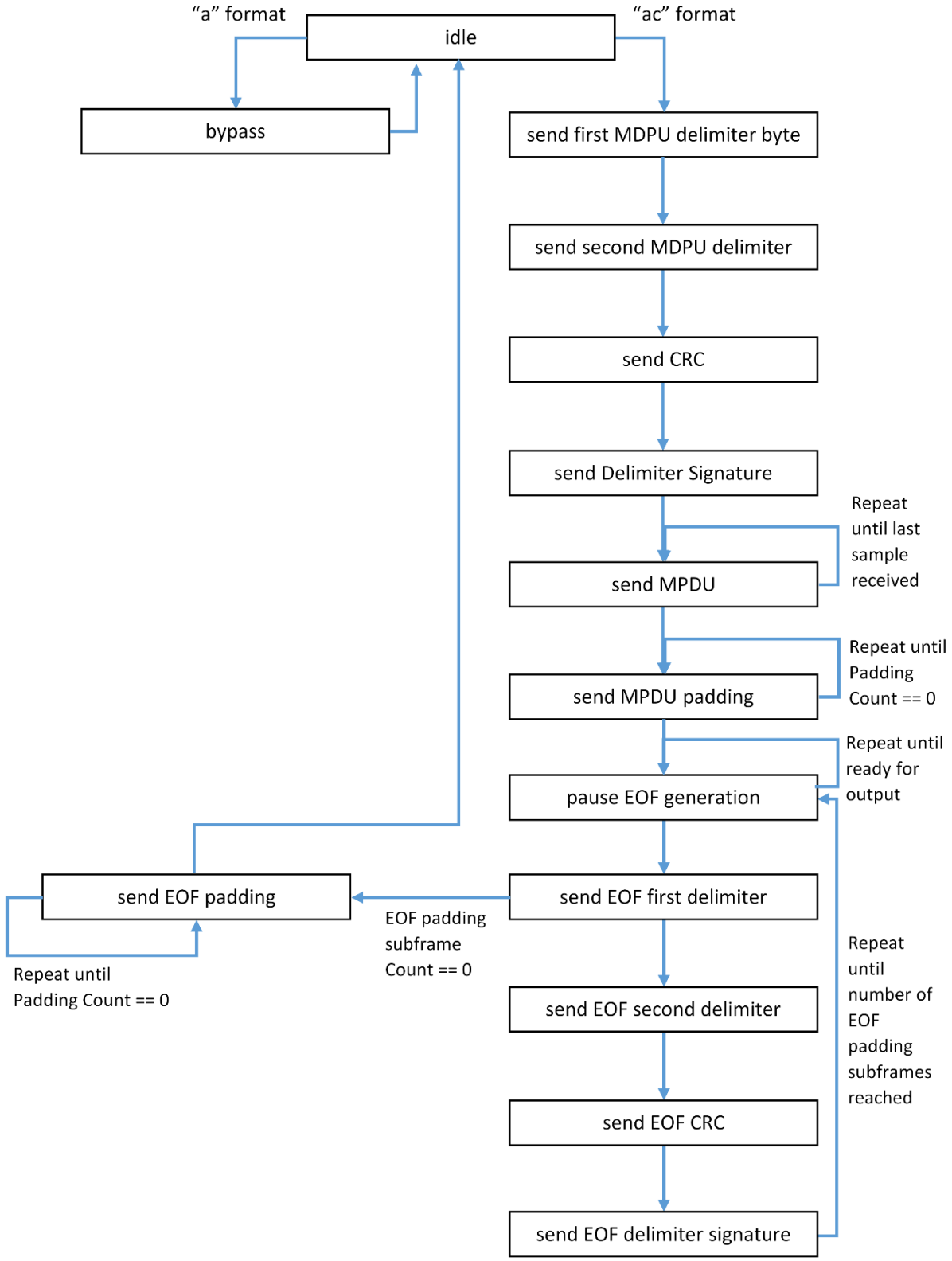

Figure 6-12: A-MPDU Aggregation State Machine

The inputs are as follows:

- MPDU Aggregation REQ (MPDU, MPDU configuration, TX vector)
    - Interpreted parameters:
        - TX vector: format, (A-)MPDU length, PSDU length
        - MPDU configuration: MPDU length

The outputs are as follows:

- PHY PSDU TX REQ (PSDU, TX vector)
    - Modified parameters: PSDU

6.4 MAC RX Modules

6.4.1 A-MPDU De-Aggregation

This module reverses the A-MPDU Aggregation which is described in Section 9.7 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. Hence, it is the counterpart of the A-MPDU Aggregation module described in Section 6.3.7. In the given implementation, only single MPDU (one MPDU per A-MPDU) are supported.

If the format is VHT, the following steps are performed:

1. Extract and decode the VHT single MPDU delimiter from the PSDU as defined in Figure 9-744 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].
2. Error Check: The A-MPDU is not been correctly received if:
    a. CRC check fails (for the CRC contained in the MPDU delimiter):
    b. EOF = 0 (indicates this is not a VHT single MPDU)
    Note: A-MDPUs containing more than 1 MPDU subframe are discarded.
    c. Delimiter signature is incorrect (Delimiter Field Value != 0x4E).
    d. MPDU length checks fail.
3. Remove delimiter from A-MDPU.
4. Remove padding from the A-MPDU.

Non-HT MPDUs are passed without modification.

The inputs are as follows:

- PHY PSDU RX IND (PSDU, RX vector)
    - Interpreted parameters:
        - PSDU
        - RX vector: PSDU length, format

The outputs are as follows:

- MAC MPDU FRAME RX IND (RX vector, RX status, MPDU, MPDU configuration)
    - Modified parameters:
        - MPDU

- RX status: A-MPDU De-Aggregation status
- MPDU configuration: MPDU length

The **A-MPDU De-Aggregation status** enumeration has the following elements:

- A-MPDU De-Aggregation successful
- VHT delimiter CRC fail
- VHT delimiter signature comparison fail
- EOF field check fail
- MPDU length mismatch
- PSDU length mismatch
- A-MPDU De-Aggregation not executed (Default value)

6.4.2 MPDU Disassembly & FCS Check

This module extracts the MAC header fields and the MAC frame body as described in Section 9.2 and Section 9.3 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. Hence, it is the counterpart of the MPDU Generation module as described in Section 6.3.6. The block diagram is shown in Figure 6-13.

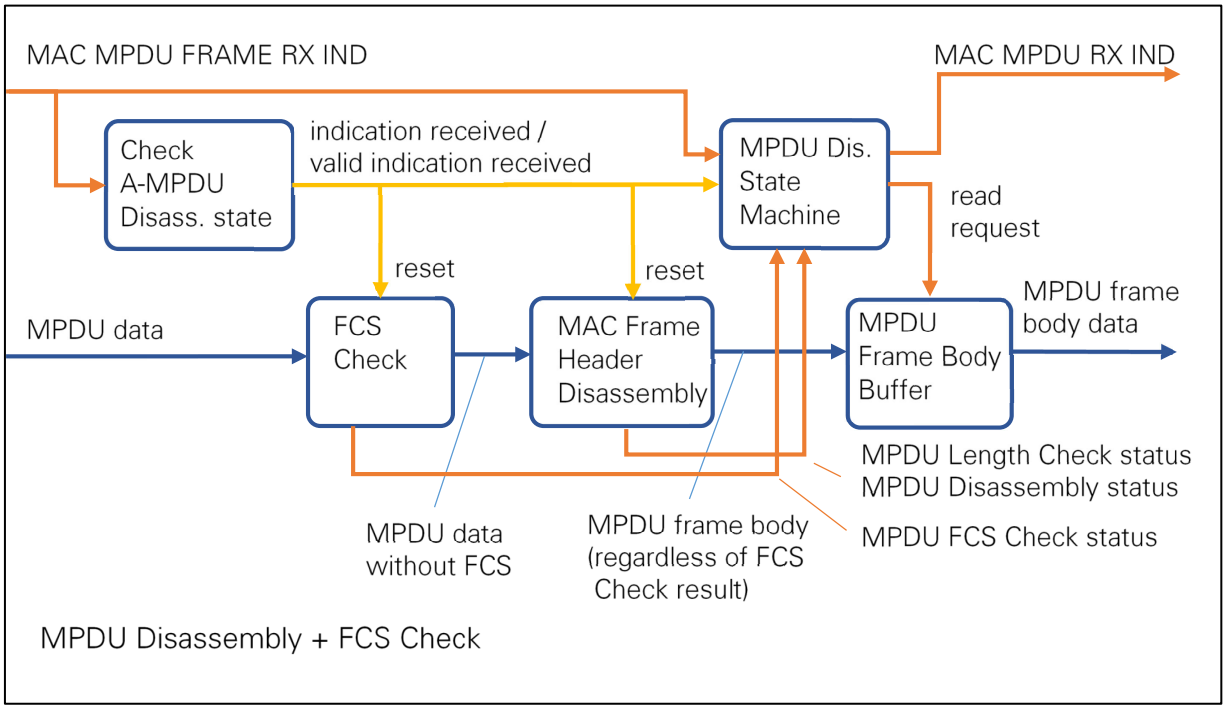

Figure 6-13: Block Diagram of MPDU Disassembly and FCS Check

If a valid MAC MPDU FRAME RX indication was received and if the A-MPDU Disassembly was successful, the submodule performs the following tasks:

- FCS Check
  - Extract the frame control field

    - Calculate the FCS value and compare it to the extracted FCS field
    - Set MPDU FCS Check Status accordingly
    - Forward all other MPDU fields to the MAC Frame Header Disassembly
- MAC Frame Header Disassembly
    - Extract the Frame Control field
    - Dynamically detect the MAC header field allocation dependent on the Frame Control field (frame type and subtype; **to DS**, **from DS** and **+HTC/Order** fields)
    - Extract the other MAC Header fields; map the address fields (for example, Address 1 maps to the Transmitter address)
    - Extract the MAC frame body and forward it to the MPDU Frame Body buffer
    - Perform length checks:
        - Length check while extracting each field: Check if the currently extracted field is fully contained in the MPDU.
        - Maximum length check: Check if the extracted MPDU frame body does not exceed a maximum length which is mandated by the chosen implementation (MPDU frame body buffer = 4,096 bytes).
    - Set MPDU Length Check Status and MPDU Disassembly Status depending on the length checks
- MPDU Disassembly + FCS Check State Machine
    - Store an incoming valid MAC MPDU FRAME RX request
    - Wait for the FCS Check submodule to provide its status
    - Wait for the MPDU Disassembly submodule to provide its status
    - Generate the MAC MPDU RX indication (modified parameters are listed below)
    - Create a read request to the MPDU Frame body buffer, if the FCS check passed and the MPDU frame body could be extracted completely
- MPDU Frame Body Buffer
    - Store the extracted frame body and write it to the output on request

The module extracts the MPDU frame body, sets the corresponding length in **parameters:MPDU frame body length** and generates the following MPDU configuration parameters:

- **Frame control**—information extracted from the frame control field is as follows: Protocol version, type, subtype, to DS, from DS, more fragments, retry, power management, more data, protected frame, HTC / order
- **Header contents**—information extracted from the other MAC header fields is as follows: Duration, source address, destination address, basic service set identifier, recipient address, transmitter address, sequence number, fragment number, QoS control, HT control

If the header content could not be extracted, the corresponding parameter is set to a default value.

The module generates the following status codes:

- RX status:MPDU FPGA check status determines if the FCS check was successful. Only if this is the case, de-aggregation and length check are executed.
- RX status:MPDU length check status informs about the result of the length check.
- RX status:MPDU de-aggregation status determines if the header could be de-aggregated completely, partially, or not at all. If the status is **`Disassembly successful`**, all header fields were extracted completely. If the status is **`Disassembly done partially`**, only the fields **duration** and **recipient address** were extracted. These fields are required for the DCF control. If the status is **`Disassembly not successful`** or **`Disassembly not executed`**, no fields were extracted at all.

The module does not filter (valid) MPDUs based on their MAC header content, except for unsupported control frame types which are only partially disassembled. Filtering is implemented in the MPDU Filtering module.

This module forwards all data and management frame subtypes even though the respective subtype may not be supported in the transmitter.

The interpreted parameters of the MAC MPDU FRAME RX IND (RX vector, RX status, MPDU, MPDU configuration) input are as follows:

- MPDU
- MPDU configuration: MPDU length
- RX vector: format
- RX status: A-MPDU De-Aggregation status

The module performs Disassembly and FCS check only if **A-MPDU De-Aggregation status** is **`A-MPDU De-Aggregation successful`**.

The outputs are as follows:

- MAC MPDU RX IND (RX vector, RX status, MPDU data, MPDU configuration)
    - Modified parameters:
        - MPDU data
        - MPDU configuration
            - MPDU frame body length
            - All parameters contained in the MAC header (frame control, header contents)
        - RX status:

- MPDU FCS check status, MPDU length check status, MPDU disassembly status

The **MPDU FCS check status** enumeration has the following values:

- FCS check fail
- FCS check pass
- FCS check not executed (default value)

The **MPDU length check status** enumeration has the following values:

- MPDU length not standard compliant
- MPDU length exceeds implemented buffer
- MPDU length check pass
- MPDU length check not executed (default value)

The **MPDU disassembly status** enumeration has the following values:

- Disassembly successful
- Disassembly done partially
- Disassembly not successful
- Disassembly not executed (default value)

6.4.3 MPDU Filtering

This module evaluates the received MPDU configuration using three sets of criteria and generates the following corresponding three indications for three downstream functions: DCF control, Frame Sequence TX control and MAC middle SAP (through MSDU De-Fragmentation). Details about forwarding conditions and the contained information are given in Table 6-1. The block diagram of the module is shown in Figure 6-14.

Table 6-1: Indications Generated by MPDU Filtering Module

Indication	Downstream Module	Forwarding Condition	Contained Information
MPDU Filtering for MAC MPDU RX Event IND	DCF control (updates the NAV or counts EIFS)	All received MPDUs, regardless of their type (also with FCS check status == fail).	• FCS check passed (disassembly at least partially successful) • Update NAV (recipient address matches STA address) • Duration
MPDU Filtering for generation of MAC RX Event to TX IND	Frame Sequence TX control	All received MPDUs, regardless of their type (also with FCS check status == fail)	• Address match (RA address matches STA address and response required) • MPDU broken (disassembly failed or not executed)

			• Parameters: duration, frame type, frame subtype, transmitted address
MPDU Filtering for MAC MPDU RX IND Generation	MAC middle SAP (through MSDU Defragmentation)	Only successfully received MPDUs of type Data (subtype Data) or Type Management	Same as MAC MPDU RX IND with MPDU Filtering status updated

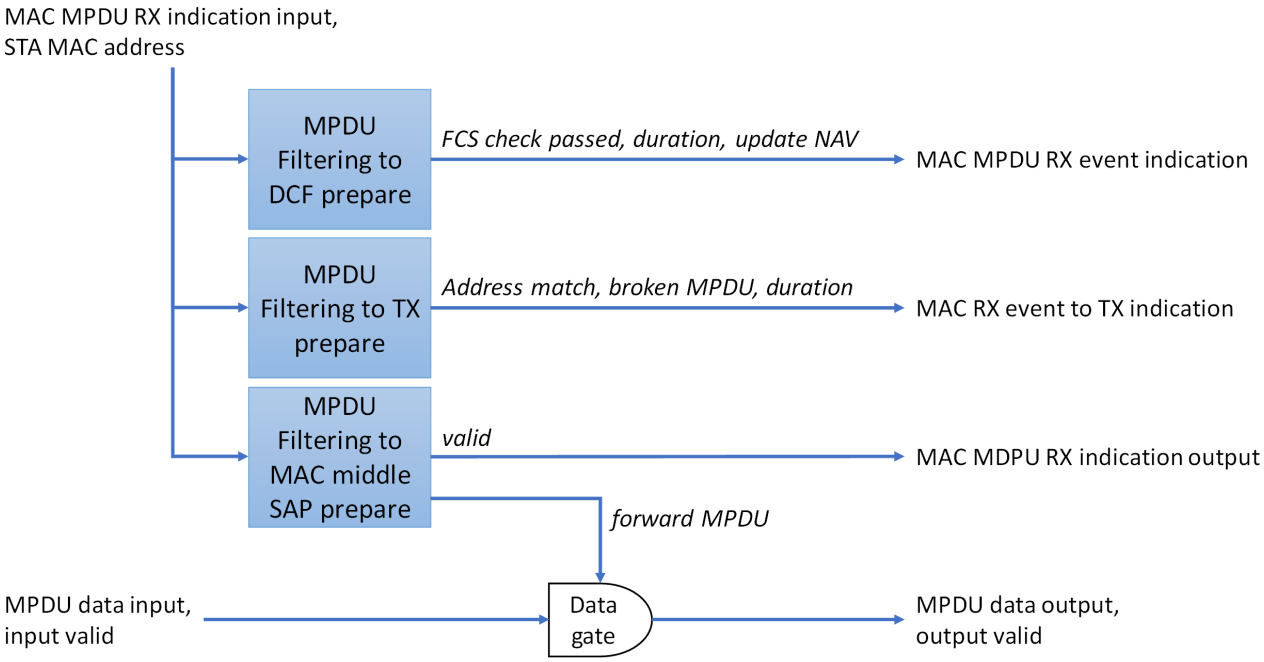

Figure 6-14: Block Diagram of MPDU Filtering

The interpreted parameters of the MAC MPDU RX IND (MPDU data, MPDU configuration, RX status, RX vector) are as follows:

- MPDU data
- MPDU configuration: MPDU frame body length, recipient address, duration
- RX status: MPDU disassembly status

The outputs are as follows:

- MAC MPDU RX Event IND (received MPDU event parameters)
  - Modified parameters:
    - received MPDU event parameters: all elements
- MAC RX Event to TX IND (received MPDU parameters)
  - Modified parameters:
    - received MPDU parameters: all elements
- MAC MDPU RX IND (MPDU data, MPDU configuration, RX status, RX vector)
  - Modified parameters:

    - MPDU data
    - MPDU configuration: MPDU frame body length
    - RX status: MPDU Filtering status

The **MPDU Filtering status** enumeration has the following values:

- MPDU valid for further processing
- MPDU pass through for sniffer mode
- MPDU Filtering not executed (default value)

6.4.4 MSDU De-Fragmentation

Because real MSDU fragmentation and defragmentation is not supported by the application framework, the functionality of this module is limited to check for the following conditions:

- **fragment number** is `0`
- **more fragments** is `0 (false)`

If these conditions are met, the received MPDU data is copied to the MSDU data. Otherwise the MSDU data remains empty. The module dependent status is set accordingly.

The interpreted parameters of the MAC MPDU RX IND (MPDU data, MPDU configuration, RX status, RX vector) input are as follows:

- MPDU data
- MPDU configuration: fragment number, more fragments, MPDU frame body length
- RX status: MPDU Filtering status

The module performs the MSDU De-Fragmentation check only if **MPDU Filtering status** is **`MPDU valid for further processing`**.

The modified parameters of the MAC MSDU RX IND (MSDU data, MPDU configuration, RX status, RX vector) output are as follows:

- MSDU data
- MPDU configuration: MSDU length
- RX status: MSDU defragmentation status

The **MSDU defragmentation detection status** enumeration has the following values:

- Unfragmented MSDU successfully received
- Unsupported MSDU fragment received
- Defragmentation not executed (default value)

6.4.5 Duplicate Detection

This module detects and removes duplicates by evaluating the **sequence number**[7] and the **retry** flag[8] based on Section 10.3.2.11.3 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. This applies for Data and Management type frames. The check is initiated if the **retry** flag is true which indicates the reception of a retransmitted MPDU. The check is implemented by comparing the received sequence number to the sequence number of previously received frames with the same TA[9], that is, received from the same transmitter station.

To enable these kind of checks, different receiver caches are defined in the IEEE 802.11 standard, see *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1], table 10-4. The application framework implements the RC1 cache (Not QoS Data) only. The other caches do not apply because the required features (for example, QoS traffic) are not supported by the framework.

The cache stores the following tuple: <Address2, sequence number, fragment number>, where the fragment number is always 0. It keeps only the most recent entry per TA. The cache size can be configured by an input parameter. It is set to 10 by default.

If the tuple exists in the cache already, a duplicate is detected and the MPDU data is cleared. Otherwise the tuple is added to the cache and the duplicate detection is assumed to be passed. In this case, the MSDU data is copied to the output. The module dependent status is set accordingly.

The following frames are forwarded without duplicate detection and cache update:

- Group addressed frames, that is, frames with an RA that is a group address.
- ATIM management frames (frame type=00, subtype 1,001).

The state machine of duplicate detection is shown in Figure 6-15.

[7] The sequence number is extracted from the sequence control field contained in the MAC header of the received frame. The extraction is done by MPDU Disassembly & FCS Check module.
[8] The retry flag is the Boolean representation of the retry bit extracted from the frame control field contained in the MAC header of the received frame. The extraction is done by MPDU Disassembly & FCS Check module.
[9] The transmitter address (TA) is extracted from the address 2 field contained in the MAC header of the received frame. The extraction is done by MPDU Disassembly & FCS Check module.

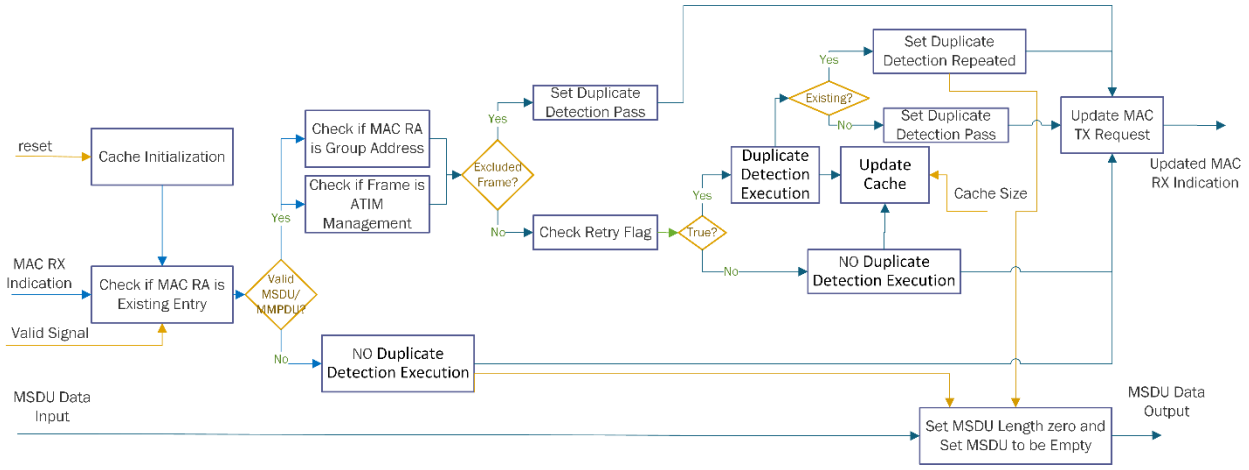

Figure 6-15: State Machine of Duplicate Detection

The interpreted parameters of the MAC MSDU RX IND (MSDU data, MPDU configuration, RX status, RX vector) input are as follows:

- MSDU data
- MPDU configuration: retry flag, sequence number, fragment number, transmitter address
- RX status: MSDU defragmentation status

The module performs duplicate detection only if **MSDU defragmentation** is **`Unfragmented MSDU successfully received`**.

The modified parameters of the MAC MSDU RX IND (MSDU data, MPDU configuration, RX status, RX vector) output are as follows:

- MSDU data
- MPDU configuration: MSDU length
- RX status: MSDU duplicate detection status

The **MSDU duplicate detection status** enumeration has the following values:

- Duplicate detection successfully passed
- Duplicate detected
- Duplicate detection not executed (default value)

6.5 DCF

6.5.1 Functional Description

The DCF mandated by the IEEE 802.11 standard [1] is carrier sense multiple access with collision avoidance (CSMA/CA). It requires that a STA sense the medium to determine if another STA is transmitting. Only if the medium is free for a certain time, it may start transmission, otherwise it must defer and wait until the end of the current transmission. After deferral, or prior to attempting to transmit again immediately after a

successful transmission, the STA shall select a random backoff interval and shall decrement the backoff interval counter while the medium is idle.

The necessary procedures are handled by the DCF. It maintains the necessary counters and generates TX opportunities for the MAC TX. To determine if the channel is busy, it communicates with the PHY CCA, PHY RX and MAC RX module.

Overall, it performs the following functions:

- Handles IFS
- Handles NAV
- Handles RX Timeouts after transmissions (CTS, ACK)
- Grants DCF opportunity on request
- Monitors PHY channels for DCF grants with variable bandwidth (20 MHz, 40 MHz, 80 MHz)
- Grants SIFS opportunity after each successful received packet

The basic operation includes the following steps. Steps 4 through 7 cover the backoff procedure:

1. Wait for CCA idle or transmission end (TX active or RX primary channel busy).
2. On successful frame reception, update NAV if required and report SIFS opportunity.
3. Wait for NAV to expire.
4. Wait for IFS to expire (IFS time based on RX / TX result from 1).
5. Generate random backoff counter if current counter value = 0.
6. Decrement backoff counter each slot until counter = 0.
7. Primary channel idle, wait for DCF request .
8. Check available bandwidth of request. If it is available, report for DCF TX opportunity. Otherwise, return to step 4 to trigger DIFS.

The backoff procedure is invoked every time, even when no transmission is scheduled.

The state machine is shown in Figure 6-16. The following list gives more information about the states and the corresponding state transitions:

- Primary channel busy:
  - PHY RX has priority over all other steps; the state machine can jump from any state (like a reset condition) to the **`primary channel busy`** state.
  - In this state, the reason (energy of signal detection) is not known.
- Frame Reception:
  - A PhyRxStart.ind clearly indicates the reason is signal detection. The Frame is processed in MAC and the timing relies on the received PhyRxEnd.ind and the MAC result as described in the receive procedure in chapter 21.3.20 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].

    - Regardless of the result, the same mechanism as for failed receptions is used to start IFS counting and eventually NAV updates.
- SIFS waiting: In case of a successful frame, the IFS to be applied is DIFS. Additionally, SIFS is started.
- NAV waiting:
    - All receive paths result in the NAV waiting state. Reason for this is that reception of frames can happen during a blocked medium by NAV. There could be remaining NAV after the reception of the frame.
    - Transition from Primary Channel busy: By default, if the primary channel is reported as idle again the current timestamp is used to trigger EIFS because of an unsuccessful reception. This applies for glitches of energy detection, L-SIG check failing or VHT-SIG-A check failing (but RXTIME known). The PHY is responsible for issuing CCA idle on primary channel at RXTIME or based on energy detection.
    - NAV waiting is followed by IFS waiting. DIFS or EIFS is chosen based on the previous events.
- Backoff waiting:
    - Backoff counting is applied afterwards. This includes generation of a random backoff time and the decrement in each slot, where the medium is not busy.
    - The backoff counter CW is controlled from outside (MPDU (Re-)Transmission control in MAC TX). It can be increased or reset. The DCF just controls the generation of the count of the slots and the initial value.
- Primary channel free:
    - After backoff counting the channel is free to use.
    - Transition to Wait for PhyTXStart.req: a DCF TX request must be issued to get a DCF opportunity. It includes the desired bandwidth. It can be issued anytime. Just one TX request can be issued at a time.
    - Transition to IFS waiting: In parallel to the state machine, which monitors the primary channel, the secondary channels are monitored. If they are idle for point interframe space (PIFS), a DCF is granted. If one of the required channels is blocked, a new backoff procedure with DIFS is triggered.
- Wait for PhyTXStart.req:
    - Once a DCF opportunity is granted, the DCF waits for the TX to get active. This is indicated by a valid PhyTXStart.req.
- TX active:
    - If TX is active, the request gets cleared and the state machine waits for PhyTxEnd.ind, which corresponds to the completion of the frame on PHY level.
    - If a PhyTXEnd.ind is received and no timeout is pending, the state changes to NAV waiting.
- Timeout waiting:

- There is the option to request an RX timeout (for CTS or ACK). This is applied after TX has finished the frame transmission.
- RX timeout can be cancelled by RX frame reception. As the RX part of the state machine can be reached from any state.
- If there is no reception during timeout, the timeout expires and causes a backoff procedure using EIFS.

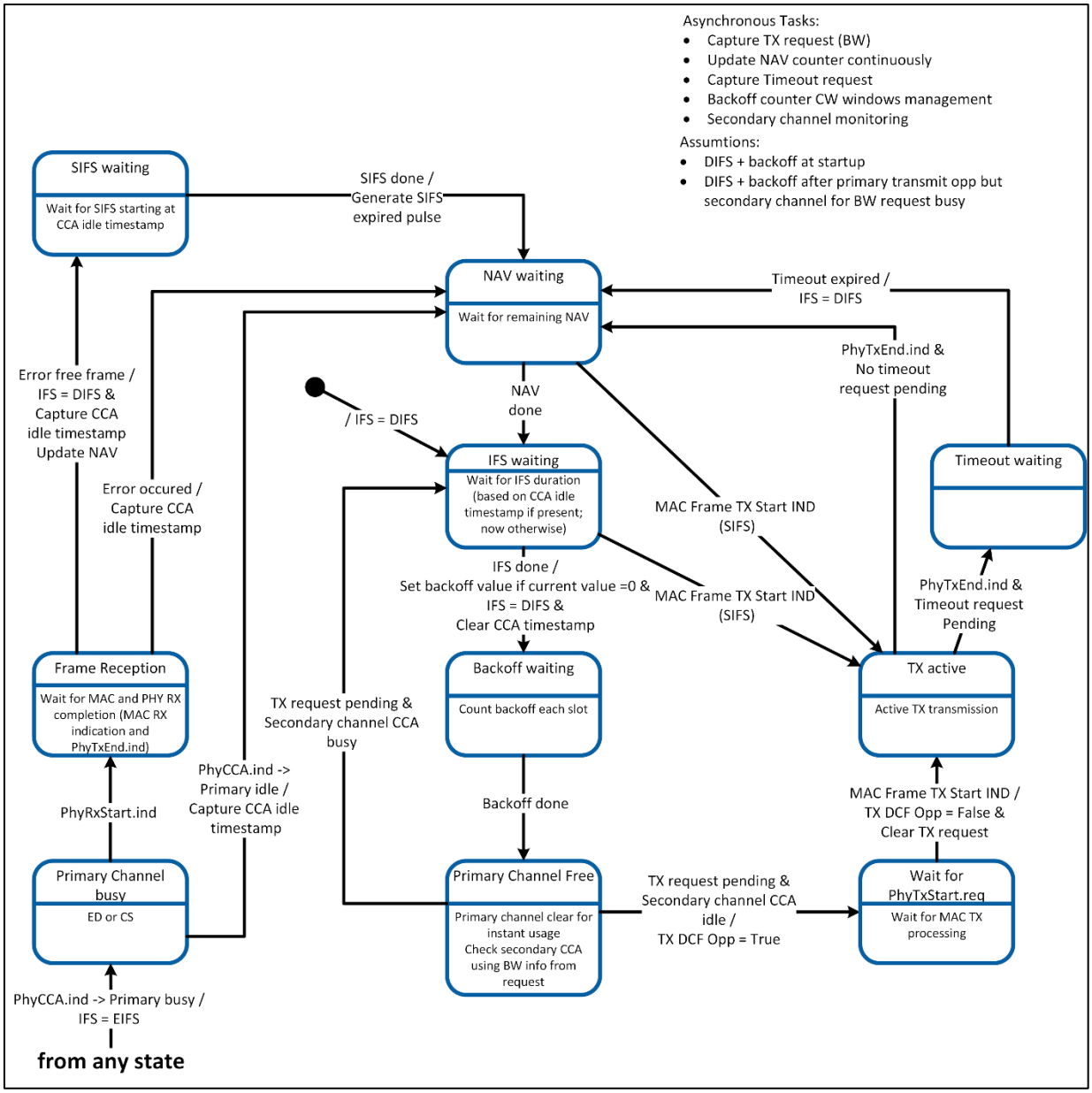

Figure 6-16: State Machine of DCF Control Module

6.5.2 DCF Implementation

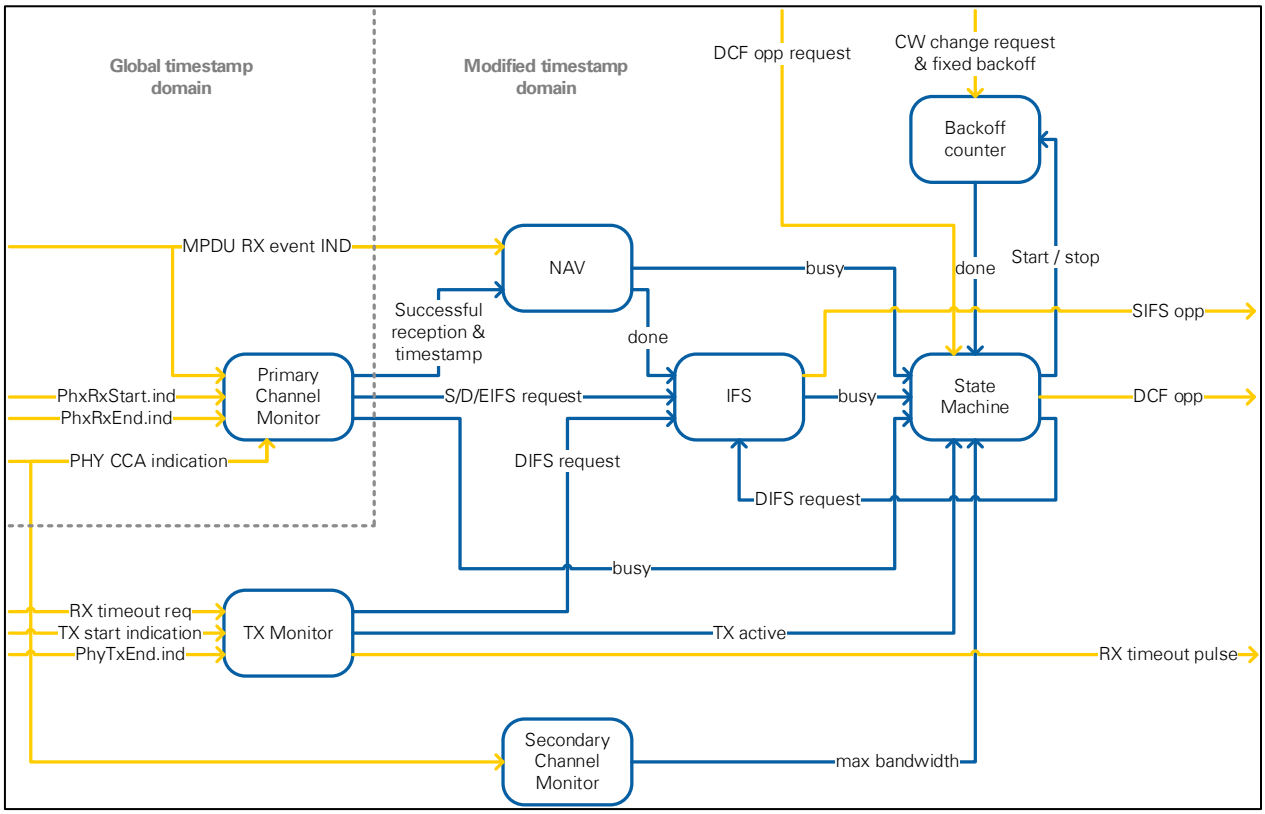

Figure 6-17: Block Diagram of DCF Implementation

The block diagram of the DCF implementation is shown in Figure 6-17. The implementation places all counters in submodules. The state machine just collects busy or done signals and cares about the logical order of states as defined in the functional description above.

All submodules containing timeout counters get an U32 timestamp, which is the runtime of the FPGA in 0.1 µs (10 MHz) unit. Two timestamp domains are used which are marked in Figure 6-17:

- The original **global timestamp** aligns with the PHY timing.
- The **modified timestamp** is calculated by **global timestamp** + **TX advance**. The shifting into the future allows the triggering of transmissions ahead of time. This ensures that the packet is provided to the RF at the correct point in time.

6.5.2.1 Backoff Counter

This module contains a random backoff value generator. It uses a linear feedback shift register (LFSR) of 11 bits. This results in non-zero random numbers in the range of 1 to 2,047. The shift register is updated each clock cycle.

The generator also takes care of the CW. It starts a aCWMin = 15 and increases to aCWMax = 1023 controlled by the CW change request. The CW is realized by marking a

part of the LFSR bits. Masking 4 bits results in a value range of 0 to 15. Masking 10 bits results in a value range of 0 to 1,023.

The backoff counter will be decreased until reaching zero or being stopped. Once restarted (from stopped state), the backoff counter continues with the previous value. Upon reaching zero, a done pulse is generated. Start/stop commands are generated by the state machine depending on the CCA state.

6.5.2.2 NAV

A NAV update is triggered upon successful reception of a frame if the "update NAV" field was set in the captured MPDU RX event indication. The trigger signal is generated by the Primary Channel Monitor module.

The update uses the duration value from the RX event indication and the timestamp given by the TX monitor. The elapsed time from TX monitor to current timestamp is deducted from the duration value. The resulting duration is loaded to the counter if it is bigger than the current value.

The counter is decremented at each 10 MHz tick. A "busy" flag indicates that NAV is currently active. A "done" flag indicates that NAV waiting finished (the counter reached zero).

6.5.2.3 IFS

The IFS module contains multiple timeout counters for SIFS, DIFS and EIFS. IFS requests can have the following sources:

- RX PHY (IFS request is provided by the primary channel monitor): Needs SIFS, DIFS and EIFS counter.
- TX PHY (IFS request is provided by the TX monitor): Needs DIFS counter only.
- DCF state machine: Needs DIFS counter only.

DIFS counting for RX PHY and TX PHY uses the same counter because the events cannot occur at the same time. A start timestamp which is provided as part of the IFS request is used for initializing the counter.

The DCF triggered DIFS uses a separate counter. Reason is a timing dependency at the end of NAV; if IFS finished before NAV+DIFS, this would disturb the state machine operation. The IFS request is generated either by the NAV module or by the DCF state machine. The modified timestamp is used for initializing the counter.

The module generates the following flags: "SIFS expired", "DIFS busy", "EIFS busy", any IFS busy".

6.5.2.4 Secondary Channel Monitor

This module determines one of the following maximum available bandwidths: 20 MHz, 40 MHz or 80 MHz. The basis is the PHY CCA indication which reports the current maximum channel bandwidth accessible. Secondary channels must not be busy for

PIFS to be accessible. There is a timer for both the secondary channel (20 MHz) and the secondary40 channel (40 MHz) which counts PIFS. It is set whenever the channel is reported busy by the PHY. Depending on the state of both timers, the maximum available bandwidth is derived.

Another output "free bandwidth at last primary busy" captures the free bandwidth at the CCA(idle) to CCA(primary) transition. This information is needed when sending CTS at a higher bandwidth. The transmission of the CTS packet may be triggered only if the corresponding channel was free at the beginning of the RTS frame.

6.5.2.5 TX Monitor

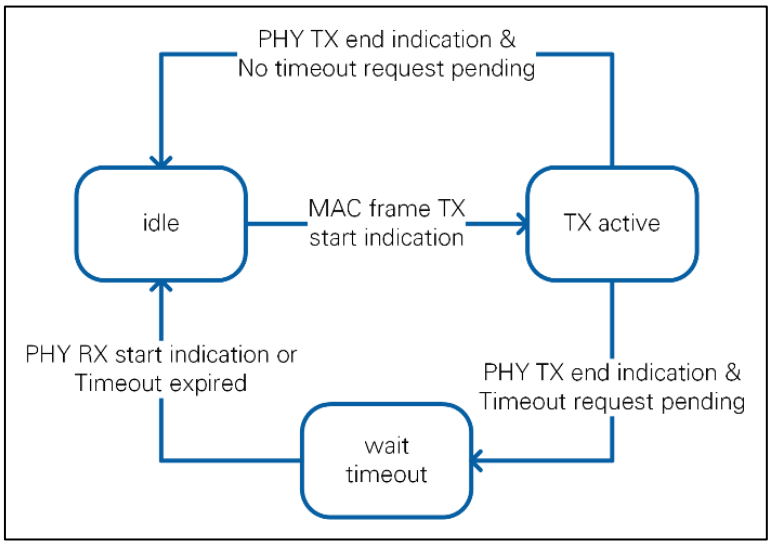

Figure 6-18: State Machine of TX Monitor

The TX monitor reports the status of the PHY TX and sets the "TX active" flag accordingly. It also handles RX timeout requests which are generated by the MAC TX. They can happen at any time but only one RX timeout can be handled at once.

The state machine is shown in Figure 6-18:

- Idle: the state machine waits for a MAC frame TX start indication. If it is received, the state changes to "TX active".
- TX active: the state machine waits for a PHY TX end indication. If no RX timeout request is pending, the state changes back to "Idle" and an IFS request (DIFS) is generated. If a RX timeout request is pending, the timeout counter is started and the state changes to "wait timeout".
- Wait timeout: the state machine waits for either RX timeout or a PHY RX start indication. In both cases the timeout counter is stopped and the state changes back to "idle". Only if RX timeout occurred, an IFS request (DIFS) is generated.

The module generates the following output signals:

- TX active (true if state machine is in "TX active" or "wait timeout" state)

- Timeout expired
- IFS request

6.5.2.6 Primary Channel Monitor

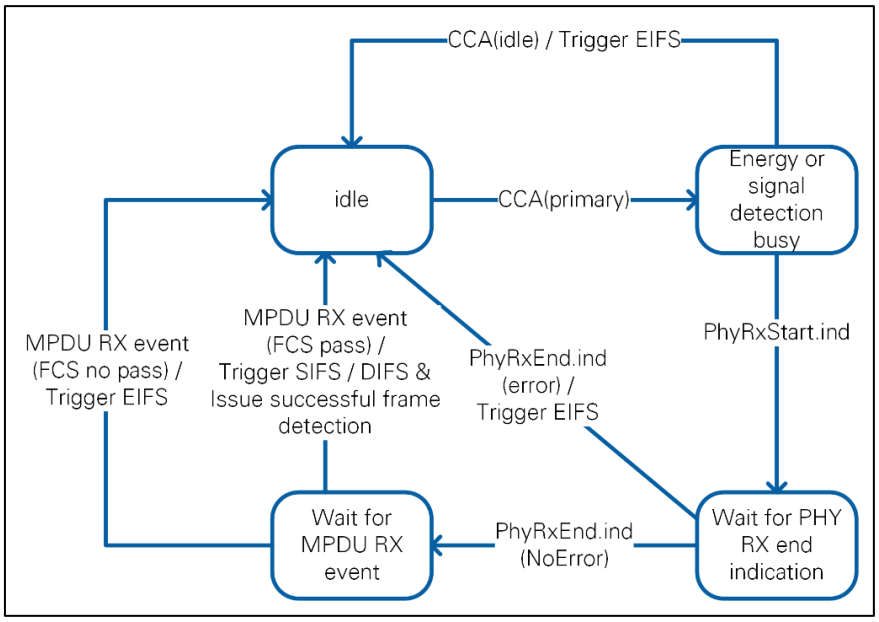

Figure 6-19: State Machine of Primary Channel Monitor

This module monitors the state of the RX primary channel. The state machine is shown in Figure 6-19.

- "Idle": The state machine waits for a PHY CCA indication indicating the primary channel is busy.
- "Energy or signal detection busy": At this time, it is not known if the busy results from energy or signal detection. If the primary channel is indicated free again (unsuccessful frame reception or only energy detected) the state machine goes back to "Idle". If a PhyRxStart indication is received (successful frame reception), the state changes to the next state.
- "Wait for PHY RX end indication": The state machine waits for a PhyRxEnd indication. If an error is indicated, the state machine changes back to "idle" and triggers EIFS. If no error is indicated, the state machine changes to the next state.
- "Wait for MPDU RX event": The state machine waits for a MPDU RX event. If it indicates a FCS error, the state machine triggers EIFS and changes back to idle. If no FCS error was indicated, the state machine changes back to idle and triggers the "successful frame detection" flag.

The module generates the following output signals:

- Primary channel busy (true in all states except for idle state)
- Successful frame reception

- IFS request cluster, which has the following parameters: SIFS, DIFS, EIFS, and start time stamp.

6.5.2.7 State Machine

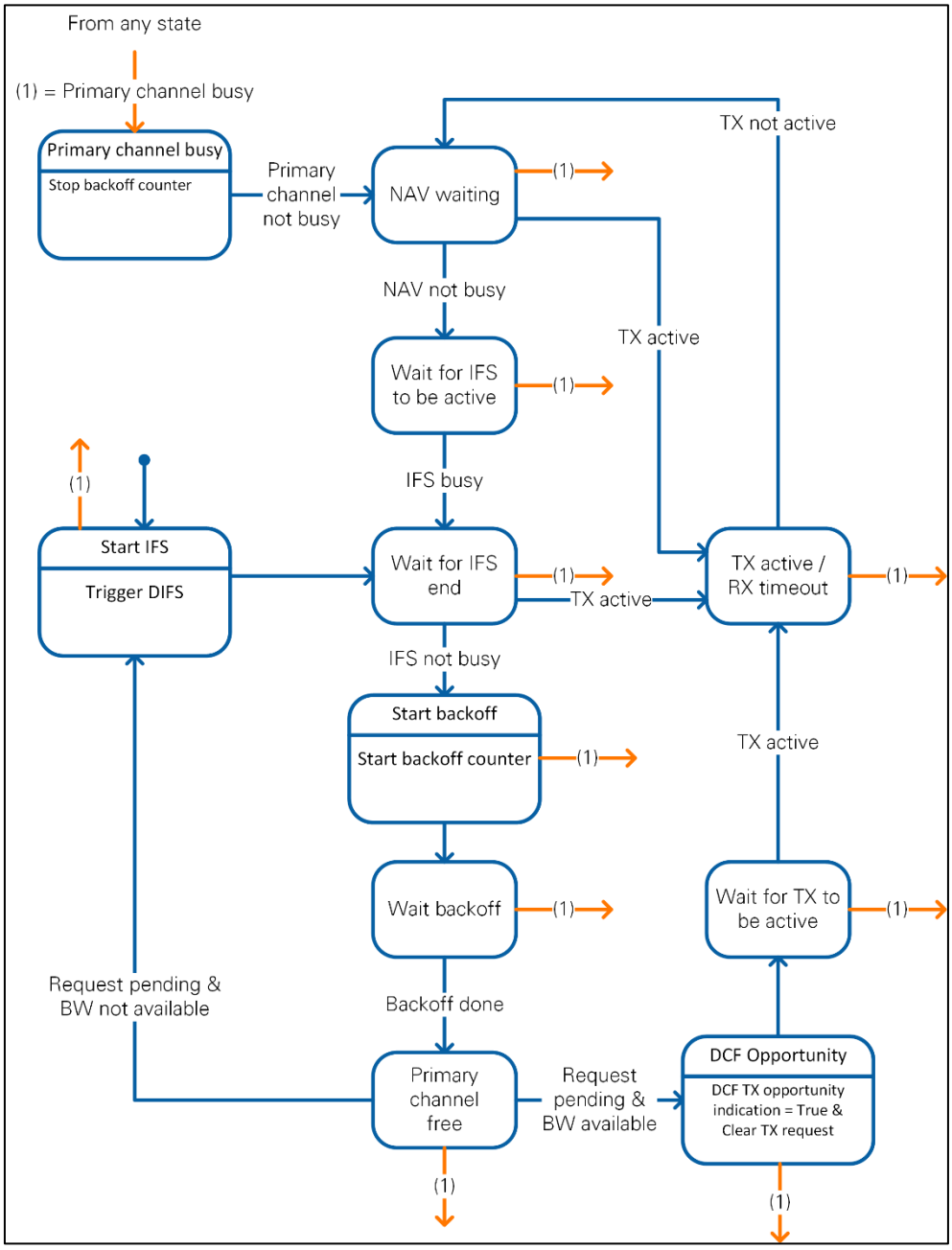

Figure 6-20: State Machine of DCF

The state machine of the DCF is shown in Figure 6-20:

- "Primary channel busy": The state machine changes to this state as soon as primary channel busy is reported. The transition can happen from any other state (orange arrows). In this state, the backoff counter is stopped continuously as the previous state may have started the backoff counting.

- "NAV waiting": As soon as the primary channel got idle, the state machine changes to this state. This is the beginning of the backoff procedure (NAV to IFS to Backoff).
- The other states follow the functional description presented at the beginning of this section. The states rely on the busy signals of the modules described above.

6.5.3 DCF Interfaces

The inputs are as follows:

- DCF TX opportunity request
- MAV MPDU RX event indication
- PhyRxStart.ind
    - Interpreted parameters: valid flag
- PhyRxEnd.ind
    - Interpreted parameters: valid flag and Rx Error
- contention window change request
- PHY CCA indication
    - Interpreted parameters: valid flag, busy flag, channel list
- RX timeout request
- PHY TX end indication
    - Interpreted parameters: valid flag
- MAC frame TX start indication

The outputs are as follows:

- SIFS TX opportunity indication
- DCF TX opportunity indication
- RX timeout pulse

7 PHY Layer

The PHY layer consists of a transmitter and a receiver. It is operating in the PHY clock domain of 250 MHz. The interfaces to MAC and RF are shown in Figure 7-1. The PHY SAP takes care of clock domain crossing to the MAC which is running in the MAC clock domain of 100 MHz. Target-scoped FIFOs are used to transfer the digital baseband signal to and from the RF which is running in the "Data Clock" clock domain which is depending on the hardware model.

The PHY transmitter receives the PhyTxStart and the associated PSDU data from the MAC and generates the PHY frame as digital baseband signal which is passed to the RF. A PhyTxEnd indication is sent back to the MAC when the processing finished.

The PHY receiver takes the digital baseband signal from the RF. If a packet is detected by the synchronization module, a PhyRxStart indication is sent to the MAC and the packet is decoded. If decoding was successful, the PSDU data and a PhyRxEnd

indication is sent to the MAC. If decoding was not successful, only a PhyRxEnd indication is sent to the MAC indicating unsuccessful reception.

The PHY receiver also includes modules for power measurement, CCA, and AGC:

- The power measurements module measures the input power on the unfiltered baseband signal. 64 input samples are averaged for one measurement. The measurements are input to the AGC module and are used for status display on the host.
- The CCA module generates PHY CCA indications which indicate that the primary or the secondary channels are busy. The reason for primary channel busy can be either energy detected or signal detected. The energy detection information is generated by the CCA energy detection module, which performs measurements on the three filtered signals (primary, secondary, secondary40). The signal detected status is derived from the "packet detected" flag from the synchronization module and the "timing valid" flag from the RX PHY state machine. The indications are evaluated by the DCF module in the MAC.
- The AGC module evaluates the power measurements and determines the RX gain needed for an optimal operation point of the PHY RF modules. If a gain change is required, it sends RX gain change commands to the register bus of the RF. Because the AGC is implemented completely on the FPGA, the AGC works on a per-packet basis.

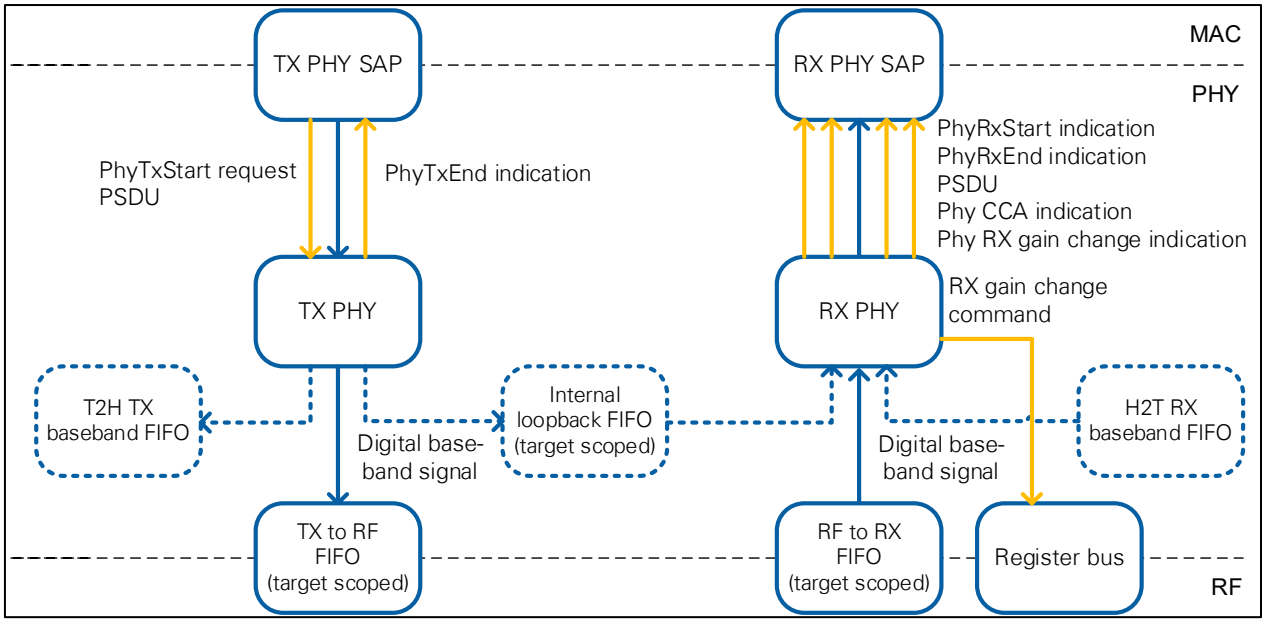

Figure 7-1: PHY Architecture

Throughout this chapter, graphic elements are formatted as described in Table 7-1**Error! Reference source not found.**.

Element	Usage

Blue rectangle (rounded edges)	Code block or VI
Blue rectangle	Code entity
Blue arrow	Data path
Yellow arrow	Control path
Red arrow	Reference to element

Table 7-1: Formatting Used for Graphics

7.1 PHY SAP

7.1.1 TX PHY SAP

The following services related to transmission are provided by the PHY to the MAC.

- PhyTxStart.req
  - This primitive is a request by the MAC sublayer to the PHY entity to start the transmission of a PSDU
  - Parameters: TXVECTOR (represents a list of parameters that the MAC sublayer provides to the local PHY entity to transmit a PSDU)
  - Corresponds to the PHY-TXSTART.request described in Section 8.3.5.5 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].
- PhyTxEnd.ind
  - This primitive is an indication by the PHY entity to the MAC layer indicating the end of a transmission.
  - Parameters: "unsupported mode" (Boolean), "scrambler seed"
  - This indication is used instead of the request/response PHY-TXEND.request/confirm described in Section 8.3.5.7 and 8.3.5.8 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].

7.1.2 RX PHY SAP

The following services related to RX are provided by the PHY to the MAC.

- PhyRxStart.ind / PHY PSDU RX IND
  - This primitive is an indication by the PHY to the local MAC entity that the PHY has received a valid start of a PPDU, including a valid PHY header
  - Parameters: RX vector (format, bandwidth, PSDU length, MCS, non-HT bandwidth (=20 MHz), dynamic bandwidth support (=False))
  - Corresponds to the PHY-RXSTART.indication described in Section 8.3.5.13 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].
- PhyRxEnd.ind
  - This primitive is an indication by the PHY to the local MAC entity that the PPDU currently being received is complete.
  - Parameters: RX vector (see above), RX error (Enum: "NoError", "CarrierLost", "FormatViolation", "UnsupportedRate", "Filtered")

    - Corresponds to the PHY-RXEND.indication described in Section 8.3.5.14 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].
- Phy CCA indication
    - This primitive is an indication by the PHY to the local MAC entity of the current state of the medium.
    - Parameters: busy (Boolean), channel list (Enum: "none", "primary", "secondary", "secondary40")
    - Corresponds to the PHY-CCA.indication described in Section 8.3.5.12 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].
- Phy RX gain change indication
    - This primitive is an indication by the PHY to the local MAC indicating the current status of the automatic gain control (AGC) module.
    - Parameters: gain, state (Enum: "Idle", "maximum gain", "wait for measurement", "gain set", "gain locked", "wait for channel clearance")
    - Proprietary; does not have a correspondent in *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].

7.2 PHY RX

The main function of the PHY receiver (PHY RX) is the decoding of the PHY frame. It also includes modules for power measurement, CCA and AGC. The PHY RX block diagram is shown in Figure 7-2:

- The **data source** module selects the source for the receiver. Data can be taken from RF, from the TX baseband using internal loopback, or from the host or by using a H2T FIFO. The stream always has a sample rate of 80 MS/s for all sources.
- The **signal filter** extracts the primary, secondary and secondary 40 MHz channel. The **synchronization** detects the packet start within the primary channel and compensates an estimated carrier frequency offset on the full bandwidth signal. In parallel, the **CCA energy detection** module calculates the received signal power and compares it against the configured CCA threshold.
- The baseband samples are given to the **RX I/Q Processing** module, where the samples are transferred to the frequency domain. Then channel estimation, equalization, and phase tracking are done. The I/Q processing also contains the format detection module (non-HT, HT, VHT).
- The constellation with field assignment information is provided to the **RX Bit Processing** module. Inside this block the modulation is reversed, the bits are deinterleaved, decoded using a Viterbi decoder and descrambled.
- This bit stream is given to the **RX PHY state machine** which generates control information for the in-phase/quadrature (I/Q) processing and bit processing

modules. It also generates the PhyRxStart and PhyRxEnd indications. The state machine interprets the signal fields (L-SIG and VHT-SIG-A) to determine the number of OFDM symbols, the bandwidth, the MCS and the PSDU length. The PSDU, which can be MPDU or A-MPDU, is extracted from the bit stream and delivered to the MAC as unsigned bytes.

Every module is designed to keep up with the data rate from the upstream module, so there is no need for throttle control inside the modules. The timing of the transfers is described in the following sections.

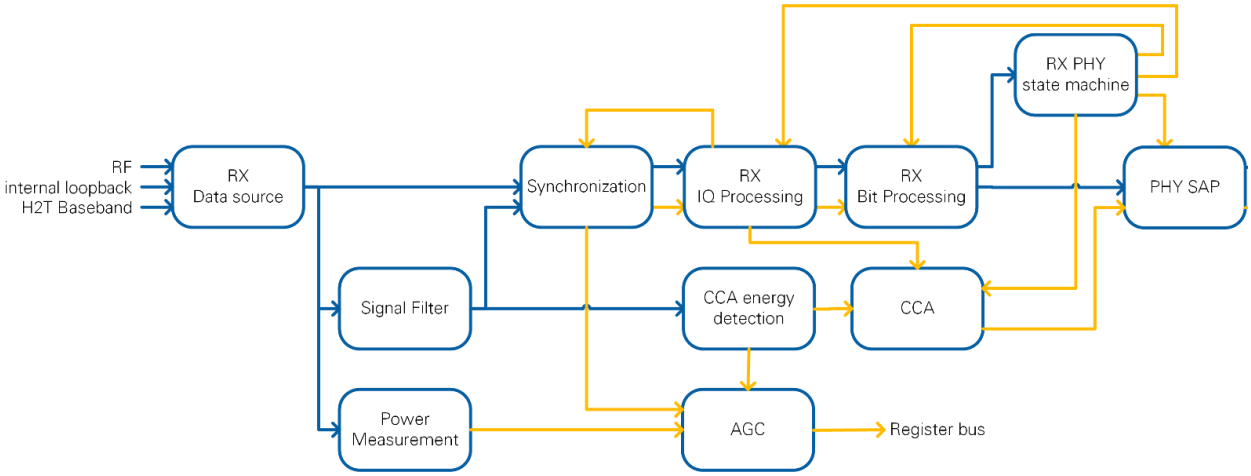

Figure 7-2: RX PHY Block Diagram

7.2.1 PHY RX AGC

The AGC ensures that the operating point of the system keeps in an optimum range. Therefore, it can apply RX RF input gain values from 0 dB to 31.5 dB. The module is implemented as a simple state machine as shown in Figure 7-3.

Note that the USRP and the FlexRIO targets have a wider gain range that is divided onto multiple gain stages. Both targets contain at least one stage whose gain range matches the AGC range. Timing wise the AGC is only capable of changing this one stage before the channel estimation requires a constant amplitude. The other gain stages are set to their maximum.

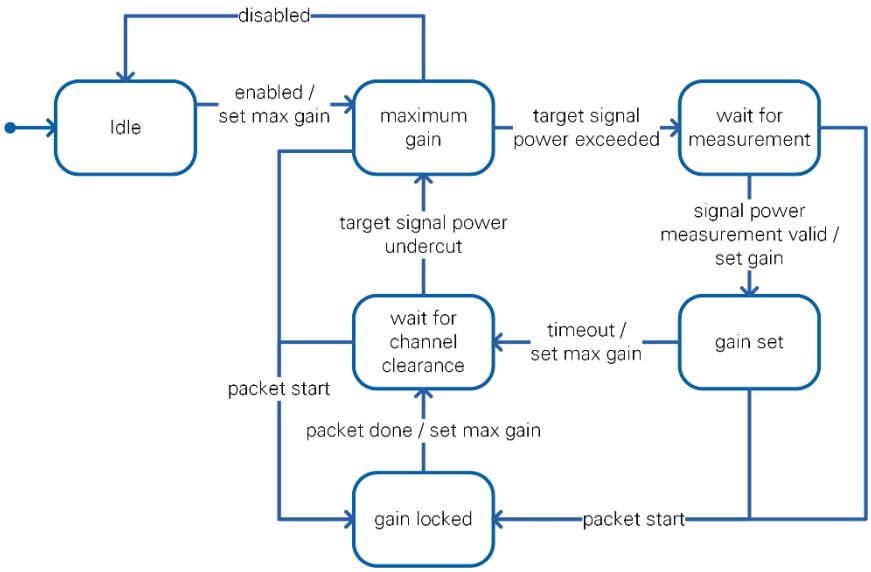

Figure 7-3: AGC State Machine

In state *Idle* the AGC waits until it gets activated. It switches to state *maximum gain* and set the maximum possible gain value (31.5 dB) to make sure that very low powered incoming packets won't be missed. The total gain is higher due to the setting of the other gain stages. When the measured baseband signal power on the primary channel exceeds a target level that can be configured from the host, the AGC waits for a second power measurement from the overall baseband signal in the *wait for measurement* state. This second value will be used to determine the applied gain based the configured target level. The finite-state machine (FSM) changes into the *gain set* state. It may be that some other signal than a PPDU triggered a gain change. If there is no packet detected after setting a gain value a timeout occurs and the FSM changes to *wait for channel clearance* state. The timeout is set to 16 µs, which is the preamble length of L-STF and L-LTF. All states except for the idle state transition to the *gain locked* state if a packet is detected by the synchronization to avoid any packet corruption by gain changes. This state is left when the packet processing is done. Due to the signal filtering and the power measurement the measured energy might still exceed the target level. The *wait for channel clearance* state therefore waits until the measured energy undercuts the level before setting the maximum gain value.

The expected timing compared to the over-the-air (OTA) signal is shown in Figure 7-4. The gain for the arriving packet must be set within a few microseconds since the gain setting using the register bus is taking around 9 µs. The design provides 12 µs after the first sample of the packet until the second part of L-LTF which is used in RX IQ processing. The maximum gain is set at the earliest point in time to settle the gain within SIFS for the next packet.

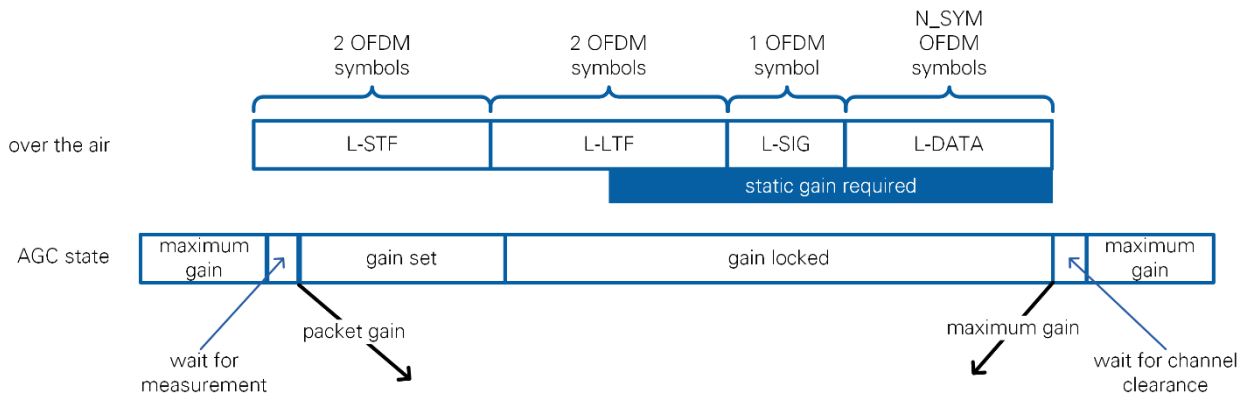

Figure 7-4 AGC Timing

7.2.2 PHY RX CCA

The RX PHY is responsible for generating PHY CCA indications to the MAC based on signal detection and energy detection as defined in Section 21.3.18.5 in *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. The only missing feature is the detection of PPDU start in the secondary channels within aCCAMidTime. The overall block diagram of the CCA mechanism is shown in Figure 7-5. Details of the signal filter and the energy detection module are illustrated in Figure 7-6.

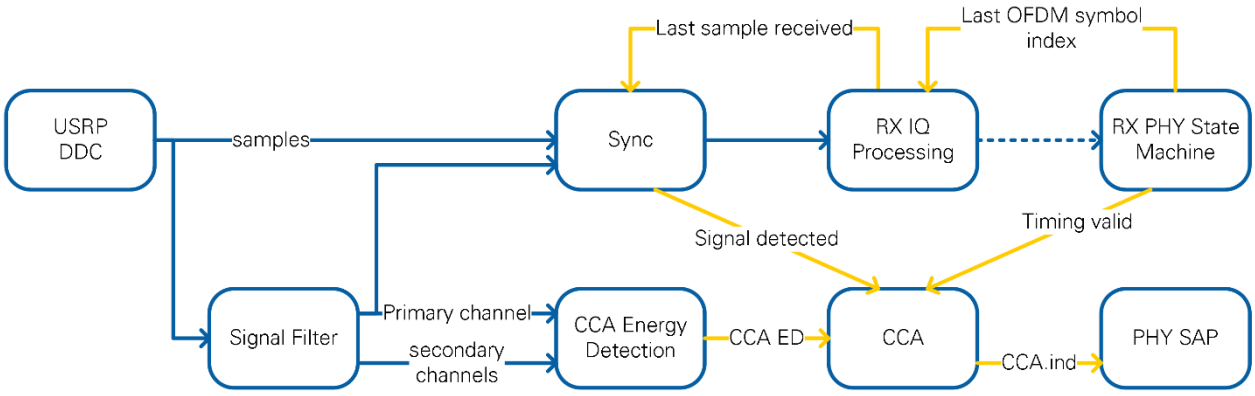

Figure 7-5: CCA Block Diagram

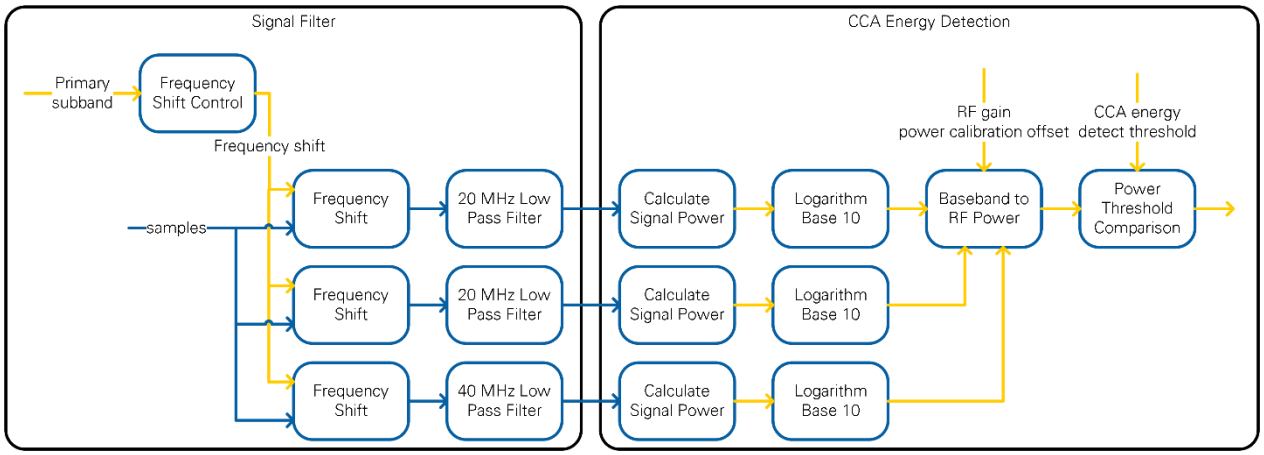

Figure 7-6: CCA Energy Detection Block Diagram

The sample from the RF are fed into a signal filter. The synchronization is responsible for the signal detection. This module separates the primary, the 20 MHz secondary, and the 40 MHz secondary channel. Each one is shifted to DC and low-pass filtered. The low-pass filtered primary channel is provided to the synchronization which is responsible for the CCA signal detection part. It reports the existence of a preamble within aCCATime. The signal detection gets deasserted when the synchronization is rearmed. This is triggered by the last sample received in RX IQ processing which is based on the last OFDM symbol index decoded from the received PPDU.

The CCA Energy detection module detects any signal above a given CCA energy detection threshold. Based on the incoming samples of each channel x the signal power s is calculated over a window of 64 samples as described in Equation 7-1. The output of this calculation is updated after 64 samples arrive. The next step is the iterative calculation of the logarithm to the base of 10. The value of s shifts n times to the left until the most significant bit (MSB) contains a one. The number of shifts, n, and a look-up table (LUT) of the six MSBs of the shifted value s' are used to calculate the signal power p in logarithmic scale. This value represents the baseband signal power in dBFS.

$$s = \sum_{j=0}^{64-1} x_{i+j} x_{i+j}^{*}$$
$$s' = 2^{n} * s$$
$$p = \log_{10}(s) = \log_{10}(s') + \log_{10}(2^{-n}) \approx \log_{10}(s') - 3n$$

Equation 7-1: Signal Power Calculation

Based on p, the RF input power r is calculated using the power calibration offset (configured from the host) and the RF gain (see Equation 7-2) provided by the AGC. Both values are given from the host. The analog gain value is subtracted from p because applying gain before ADC means that the RF input power is lower than the measured signal power. The power calibration offset is based on the calibration data of the device. It maps the baseband signal power at minimum gain to the corresponding

reference power level[10] at the RF input port. This mapping is assumed to be linear at all gain levels.

$$r = p - \text{gain} + \text{offset}$$

Equation 7-2: RF input Power Calculation

The value of r is compared against the given CCA energy threshold. If this threshold is exceeded, a CCA indication is generated containing the affected channel.

The information of signal and energy detection is combined in the CCA module. The energy detection information is forwarded by default. If necessary, this information is overwritten by signal detection (reported as primary channel busy) when activated. If signal detection is deasserted, there are two possible options as follows. The RX PHY state machine provides the information if the duration of the packet is known.

- If the duration of the packet is known, the CCA indication reports idle to the MAC. This state is kept for 3 energy detection measurement to ensure that the filtering and power measurement modules are clear of remaining samples from the PPDU.
- If the timing is not known, the CCA energy detection values are directly forwarded to the MAC.

7.2.3 Synchronization

The purpose of the synchronization module is to find the packet start in the continuous sample stream. The ideal position of the packet start for the implemented algorithm is in the center of the L-LTF field. This second half is treated as L-LTF-2 where the remaining 64 samples from L-LTF-1 are like a cyclic prefix in all the other following OFDM symbols.

The block diagram of the synchronization unit is shown in Figure 7-7.

The synchronization is fed from the data source with a sample rate of 80 MS/s. In the baseband clock domain of 250 MHz, approximately every third sample is valid. Each VI must use the enable chain to update only on valid samples.

[10] This reference power level corresponds to the power level of a continuous wave (CW) signal having an amplitude of -3 dBfs at the ADC input.

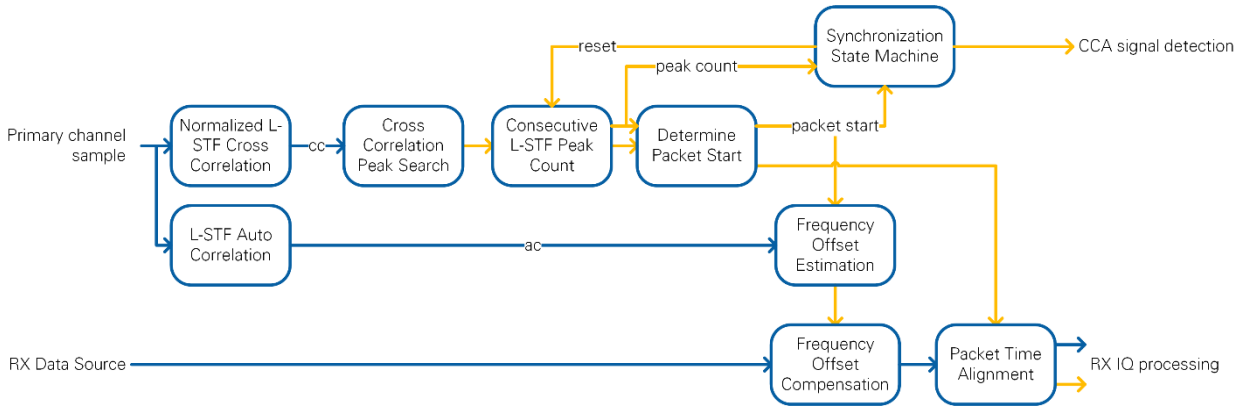

Figure 7-7: Synchronization Block Diagram

The synchronization block is implemented in two parallel paths (refer to Figure 7-7) to minimize the latency on the data path. The upper path finds the packet start sample index and estimates the frequency offset based on the Schmidl and Cox algorithm [4] on the low pass filtered primary channel. These estimates are used by the lower path, which is the main unfiltered data path, to compensate frequency offset and generate the packet start pulse for downstream modules.

For testing purposes, there is a bypass for the synchronization block where the packet start index can be given from the host. This path is not included in Figure 7-7. Use this bypass in combination with RX samples from the host or internal loopback to characterize the RX baseband without the impact of synchronization algorithms.

As shown in Figure 7-7, the upper path of the synchronization block starts to calculate the cross correlation of the received signal x and one period of the L-STF time signal p (see Equation 7-3). Additionally, the result is divided by the received signal power. The signal strength of p is left out for simplifying the calculation on the FPGA. As a result, the magnitude of the cross-correlation results in 10 peaks during the 640 samples of L-STF as shown in Figure 7-8 (X-Corr).

$$cc(n) = \frac{\sum_{i=0}^{63} x(n-i) \cdot p^*(64-i)}{\sum_{i=0}^{63} |x(n-i)|}$$

Equation 7-3: Synchronization Cross-Correlation

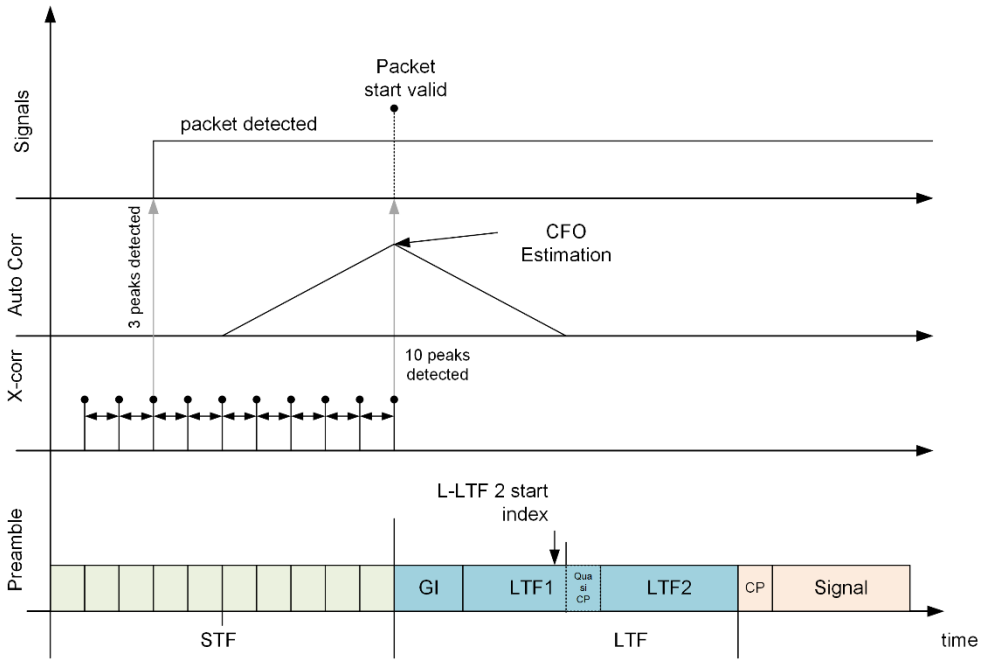

Figure 7-8: Simplified Signal Charts of Synchronization

Based on the magnitude of the cross correlation a peak search is performed. It compares the magnitude against a given threshold and reports a peak if a configured number of sample exceeds this threshold. A wrapping U16 counter, incremented for each valid sample, is used to represent the position within the continuous sample stream. The numeric range of 16 bits is sufficient to express a unique position within the preamble of the packet. The next module reports the number of consecutive peaks. The peaks caused by the L-STF appear each 64 samples. Given an allowed derivation the peak search identifies peaks which match the expected distance. It accounts for a possible wrap of the index counter. The output is the number of consecutive peaks and the last peak index from the module input. The *Determine Packet Start* module reports the packet start if the number of consecutive peaks exceeds 9.

Note that it may be due to random noise processes that a peak is detected before the start of the packet leading to a total number of 11 peaks (reported as 10 consecutive peaks). In this case the packet start reported after 9 peaks is overwritten with the next peak as the last peak belongs to the L-STF. Further peak detections within the L-LTF field are very unlikely.

There is a state machine which controls the generation of the CCA signal detection (see Figure 7-9). The MAC must be informed about a preamble within aCCATime (see Table 21-27 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]). This requires an indication ahead of the packet start. The existence of a preamble is reported to the MAC in case there are two consecutive peaks found. As this is not a fully safe decision there is a timeout started which count expire in the *preamble candidate detected* state and leads back to the initial *search preamble* state. Any additional consecutive peak resets this timeout. If a packet start was found the

state is switched to *packet start detected.* Because of the optimal additional peak mentioned above the same timeout mechanism is used to lock the sync afterwards. In the *sync locked* state the consecutive peak count is forced to zero to disable detection of further preambles. It requires the rearm trigger to enable the peak search again and to disable the signal detection signal. The rearm trigger is generated by the RX IQ processing on reception of the last sample of the packet.

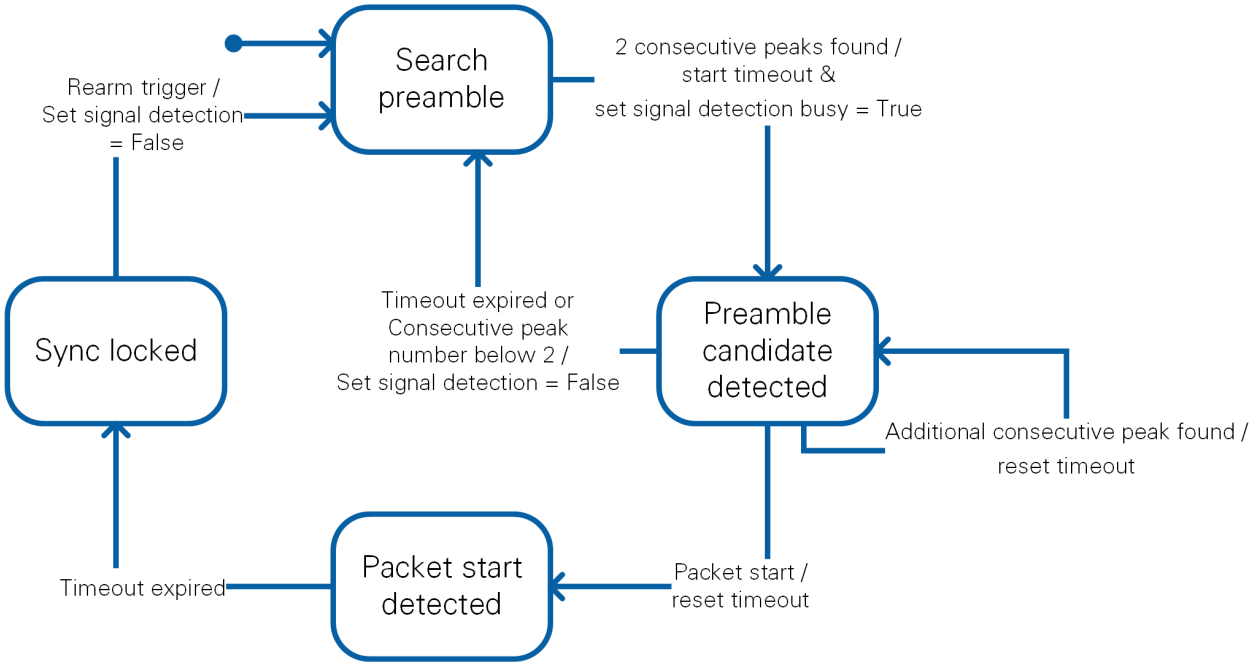

Figure 7-9: Synchronization State Machine

In parallel to the computation of the packet start index there is the frequency offset estimation. It is based on autocorrelation as shown in Equation 7-4. The signal *x* is folded onto itself using a window up half L-STF length (320 samples). The ideal result is shown in Figure 7-8 as Auto Corr.

$$a(n) = \sum_{i=0}^{319} x(n-i) \cdot x^*(n-i-320)$$

Equation 7-4: Synchronization Autocorrelation

The frequency offset estimator module uses the autocorrelation result and the packet start index to pick the peak of the autocorrelation. The phase of the auto correlation value at the packet start index position directly translates the frequency offset as shown in Equation 7-5, where *n* is 320 samples from the autocorrelation.

$$\boldsymbol{\varphi(a(packet\ start\ index)) = 2\pi n \frac{\Delta f}{f_s}}$$

Equation 7-5: Synchronization frequency offset estimation

The frequency offset and the packet start index are captured in the lower chain upon a detection of a packet start. The estimated frequency offset is compensated by applying a digital frequency shift. The frequency estimate is used for all OFDM symbols of the entire packet. The Frame Alignment module generates the packet start trigger pulse at the captured start sample index.

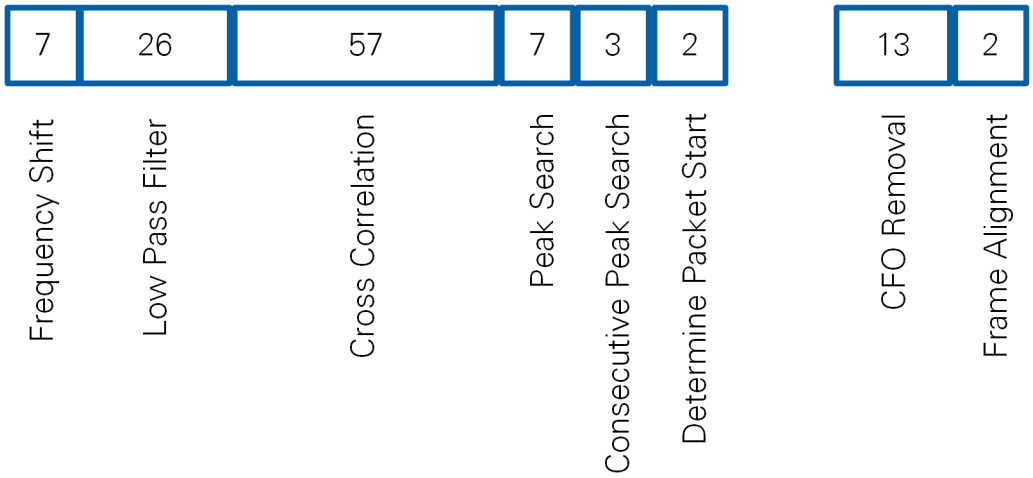

Figure 7-10: Synchronization Latency

7.2.3.1 Synchronization Latency

The latencies for the different modules in the Synchronization block are illustrated in Figure 7-10. The left part of the figure contains the modules of the sample index computation path. The latency of those modules totals 102 clock cycles. Given the sample rate of 80 MS/s at 250 MHz clock rate, this time is equivalent to about 32 samples. Since the packet start index is located 160 samples after the last sample of L-STF, the packet start index is calculated before the packet start signal must be asserted, and there is no effective delay.

The latency of the main data path is shown in the right part of Figure 7-10. This latency increases the length of the RX processing path by 15 clock cycles.

7.2.4 RX IQ Processing

The receiver RX IQ Processing block purpose is to restore the transmitted I/Q constellation. The block diagram is shown in Figure 7-11. Details of data types, control information, and identifiers used in equations are presented in Table 7-2.

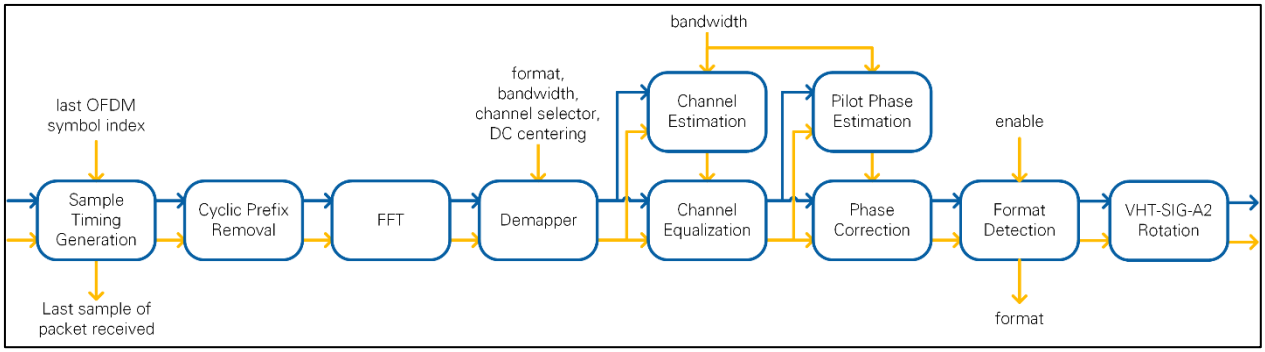

Figure 7-11: RX IQ Processing Block Diagram

Module	Identifier	Output Data Type	Output control information
Synchronization	—	CFX 3.13	Packet Start
Sample Timing Generation	—		Sample Timing
Cyclic Prefix Removal	—		
FFT	R	CFX 4.21	OFDM symbol index
Demapper	—		Field Map Subcarrier Timing
Channel Estimation	H_{est}		Subcarrier index
Channel Equalization	Y_{est}	CFX 2.14	Field Map Subcarrier Timing
Pilot Phase Estimation	B	FXP 1.14	—
Phase Correction	X_{est}	CFX 2.14	Field Map Subcarrier Timing
Format Detection	—		
VHT-SIG-A2 Rotation	—		

Table 7-2: RX IQ Processing Data Types and Control Information

The Sample Timing Generation module gets samples from the Synchronization module along with the packet start index. It starts passing samples to downstream modules as soon as the packet start signal is asserted. It stops passing samples as soon as the last OFDM symbol is finished whose index is given by the RX PHY state machine. The control information is carried by the sample timing cluster, which contains the following elements:

- OFDM symbol index
- Sample index (within the OFDM symbol in the range of 0 to 319)
- Packet start flag
- OFDM symbol start flag
- Valid flag

The sample index is used by the Cyclic Prefix Removal module to invalidate the first 64 samples of each OFDM symbol.

The next downstream module is the FFT, which is a wrapper for the Xilinx FFT core. It contains a 256-point FFT operation using a Radix 4, Burst I/O architecture. A toggling negation realizes the FFT shift to have the DC at the 128th output value. The FFT starts execution as soon as 256 samples are provided. During the execution, no samples are taken on the input. A FIFO is placed before the input to capture the samples that arrive in the meantime. On finishing execution, the 256 subcarriers are provided at the output consecutively. The OFDM symbol index from the incoming sample timing cluster is passed through this module, parallel to the data stream. The maximum gain of the FFT is 256 if the energy is limited to only one subcarrier. Therefore, the fixed point data type is extended by nine bits to capture this output dynamic range of the FFT module. The output of the FFT is divided by 256 to have the same scaling as on the input of the IFFT in the transmitter chain. The resulting fixed-point format is <4.21>.

The Demapper block aligns two control information clusters with the data stream. The first cluster is the subcarrier timing cluster, which contains the following elements:

- OFDM symbol index
- Subcarrier index (0 to 255)
- Frequency offset index (named *k* in equation 17-23 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]; -128 to 128)
- OFDM symbol start flag
- Valid flag

The frequency offset index is generated based on the control information from the RX PHY state machine (refer to Section 7.2.7). The second control information cluster is the field map. This cluster is made up of Booleans, and each Boolean represents one field of the IEEE 802.11 packet structure, such as L-SIG, L-LTF, VHT-SIG-A, pilot subcarrier, or data subcarrier. Similar to a one-hot-code, only one of these Booleans is asserted for each sample. The packet structure is known to the Demapper module. Downstream modules can take this field map to filter for specific fields, such as the pilot subcarriers.

The channel estimation is computed using the second L-LTF OFDM symbol for 802.11a and VHT-LTF for 802.11ac. The inverse channel transfer function is calculated for each subcarrier *R* individually using the L-LTF definitions *L* from Section 21.3.8.2.3 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1] as shown in Equation 7-6. The signal names are included in Figure 7-11. The frequency offset index *k* from the subcarrier timing is used. The channel estimation block is implemented in a parallel path to minimize latency to the data path. The values of H_{est} are given to the channel equalization module where they are stored in memory. They have the same data type as the incoming subcarriers. Beginning with the L-SIG, the channel equalization uses those values to apply zero forcing to get signal Y_{est}. The fixed-point format of <2.14> is sufficient to represent the values of Y_{est}. Larger values are saturated.

$$H_{est,k} = L_k^* R_k$$
$$Y_{est,k} = \frac{R_k}{|H_{est,k}|} \frac{H_{est,k}^*}{|H_{est,k}|}$$

Equation 7-6: Channel Estimation and Compensation

The signal Y_{est} is passed to the pilot phase modules that follow the same structure as the channel estimation and equalization. Removing the cyclic prefix leads to a phase jump between consecutive OFDM symbols due to the residual carrier frequency offset after the synchronization. The phase for the current OFDM symbol α_n is calculated based on the pilot sequences *P*. These sequences are taken from sections 17.3.5.10 and 21.3.10.10 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1] at the frequency offset index *k*. The phase offset between OFDM symbols is compensated by adding the difference to the last phase estimation from OFDM symbol *n* - 1. The estimated phase β of OFDM symbol n is applied to the OFDM symbol *n* + 1 by the Phase Correction module. This operation does not change the magnitude of the values, so the fixed-point format is kept.

$$\alpha_n = \measuredangle\left(\sum_k P_n^k Y_{est,k,n}\right)$$
$$\beta_n = \alpha_n + (\alpha_n - \alpha_{n-1})$$
$$X_{est,k,n+1} = Y_{est,k,n+1}(\cos(\beta_n) + i\sin(\beta_n))$$

Equation 7-7: Phase estimation and Compensation

The next downstream module is the format detection, whose functionality is described in section 7.2.5. The default operation mode of this module is just a bypass, which does not apply any changes to the data. As the last step of the RX I/Q processing, the clockwise rotation of VHT-SIG-A2 (refer to Section 21.3.4.5 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]) is reversed.

Module	Output timing
Sample Timing Generation	~1 sample/3 clock cycles (320 samples per OFDM symbol)
Cyclic Prefix Removal	~1 sample/3 clock cycles (256 samples per OFDM symbol)
FFT	256 subcarriers/OFDM symbol burstwise
Demapper	
Channel Estimation	
Channel Equalization	
Pilot Phase Estimation	1 phase estimate/OFDM symbol
Phase Correction	256 subcarriers/OFDM symbol burstwise
VHT-SIG-A2 Rotation	

Table 7-3: RX IQ Processing Transfer Timing

The timing of the data stream is changed inside the RX I/Q Processing module. A summary for all submodules is given in Table 7-3. The input is given by the digital downconversion at a sample rate of 80 MS/s. The Cyclic Prefix Removal module removes 64 samples from the stream. Because of the chosen FFT architecture configuration the output of the Xilinx core is given burstwise. This transfer timing is kept for all downstream modules. The only exception is the Pilot Phase Estimation module that computes one phase estimate per OFDM symbol.

7.2.4.1 RX I/Q Processing Latency

The overall latency of the RX I/Q Processing module for the last sample of the OFDM symbol is 665 clock cycles as shown in Figure 7-12. The FFT latency is smaller than reported by the Xilinx IP Generator, and this latency includes loading of all 256 samples. During the packet, the FFT executes and unloads samples in 617 clock cycles after the last sample arrived. The remaining clock cycles per OFDM symbol are used to transfer data from the input FIFO to the FFT core. By the time the last sample is available on the input, the FIFO is empty, and it is passed to the core as fast as possible. The delay of the FIFO is unknown, which is indicated in Figure 7-12. All other modules have a fixed latency.

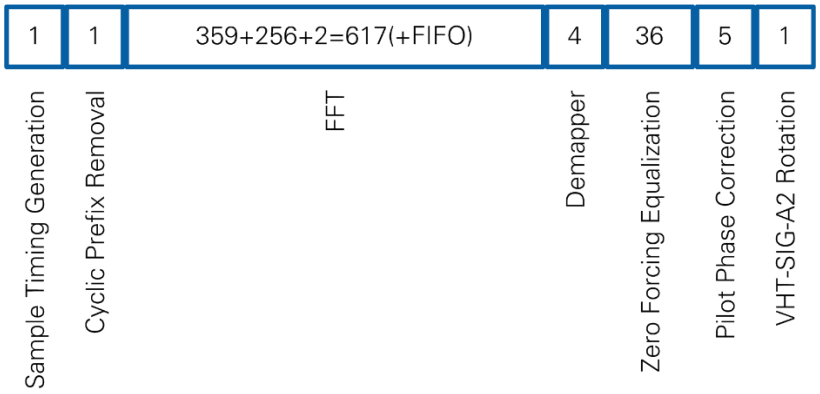

Figure 7-12: RX IQ Processing Latency

7.2.5 PPDU Format Detection

To distinguish the PPDU formats non-HT, HT and VHT the constellations of VHT-SIG-A and the HT-SIG are rotated compared to a non-HT frame as shown in Figure 21-20 and Figure 19-7 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. This rotation is determined by the format detection module within the RX IQ processing. It is controlled by the RX PHY state machine.

Whenever the L-SIG reports an MCS 0 transmission with more than 6 OFDM symbols the PPDU is treaded as an VHT candidate. An VHT frame would contain at least six OFDM symbols after the L-SIG with N_SYM = 1. In this case the format for the I/Q processing is set the VHT and enables the format detection module. The demapper then marks the next two OFDM symbols as VHT-SIG-A. The format detection module saves those subcarriers in an internal FIFO and analyses the rotation. Based on the

signal names given in Table 7-2 the operation is provided in Equation 7-8. After all subcarriers of the two OFDM symbols have been processed, there are two *d* values. *d*=1 represents the presence of a rotated BPSK modulation. The detected format is then reported based on Table 7-4. The state machine sets the configuration for I/Q and bit processing based on the given format and instructs the format detection module to flush the saves subcarriers. During this step the format detection module reassembles the field mapping of the subcarriers based on the detected format. After all subcarriers have been flushed out of the internal FIFO the module returns to its default bypass state.

$$I_n = \sum_{k=-26}^{26} \left| re(X_{est,k,n}) \right|$$

$$Q_n = \sum_{k=-26}^{26} \left| im(X_{est,k,n}) \right|$$

$$d_n = \begin{cases} 1 \; if I_n < Q_n \\ 0 \; otherwise \end{cases}$$

Equation 7-8: Format Detection Algorithm

d_0	d_1	format
0	0	Non-HT
0	1	VHT
1	0	undefined
1	1	HT

Table 7-4: Format Detection Translation

7.2.6 RX Bit Processing

The RX bit-processing chain deinterleaves, decodes, and descrambles the data. It provides the received bits to the RX PHY state machine and the PSDU bytes to the MAC. The block diagram is shown in Figure 7-13. Details about data types and control information is given in Table 7-5.

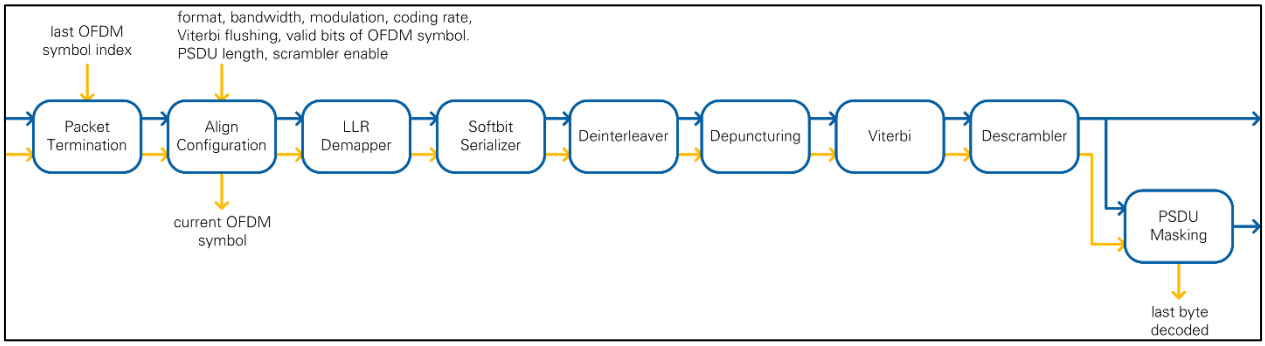

Figure 7-13: RX Bit Processing Block Diagram

Module	Output Data Type	Output control information
RX IQ Processing	CFX 2.14	Field Map Subcarrier Timing
Packet Termination		
Align Configuration		Bit Processing Configuration
LLR Demapper	FXP8.0 Array (8 elements)	
Softbit Serializer	FXP8.0	
Deinterleaver		
Depuncturing	FXP8.0, Boolean Array (2 elements)	
Viterbi	Boolean	
Descrambler		
PSDU Masking	U8	-

Table 7-5: RX Bit Processing Data Types and Control Information

The first module of the chain is the *Packet Termination* module. It passes all samples that have OFDM symbol indices in the subcarrier timing cluster below the value given from the RX PHY state machine. Passing only these samples ensures that the packet end is correctly processed. If you abort the current packet reception, this module terminates all I/Q data by setting the last OFDM symbol index to 0.

Next block is the *Align Configuration* module. It has two functions. First function is to align the bit-processing configuration cluster from the RX PHY state machine with the start of a new OFDM symbol. All other control information is terminated in this module. The bit-processing configuration is transferred parallel to the data stream, and it contains the following information:

- Packet format
- Bandwidth
- Modulation
- Coding rate
- PSDU length (in bytes)
- Valid bits in current OFDM symbol
- Descrambler enable flag
- Viterbi flush required flag

The second function is the filtering of all noncoded fields for downstream modules. It uses the field map provided by the RX I/Q processing chain.

The I/Q samples in the coded fields are processed by the *log-likelihood ratio (LLR) Demapper* block. Based on the given modulation scheme, an array of up to eight softbits is given at the output. The data type of each softbit is unsigned 8-bit integer.

The *Softbit Serializer* module takes this array of softbits and provides the serialized stream on the output. The number of valid softbits in the array is derived from the modulation. An internal FIFO is used to buffer softbits on the input.

The *Deinterleaver* module reverts the binary convolutionally encoded (BCC) interleaver operations defined in sections 17.3.5.7 and 21.3.10.8 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. The write operation into the memory is based on equations 21 through 82 of [1], which reverses the second permutation. The read operation is based on equation 21-77 of [1], which reverses the first permutation. Reading is started as soon as all softbits of the current OFDM symbol are saved to memory. A double page memory is used, which enables reading and writing at the same time.

Based Figures 17-9 and 19-11 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1], the *Depuncturer* module converts the incoming bit stolen data sequence to the bit inserted data sequence. Each bit gets a puncturing flag attached depending on whether it was transmitted or left out. One element of A and the corresponding element of B are combined into an array of two elements, where A and B are defined as in Figure 17-9 and 19-11 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].

The array is given to the Viterbi wrapper, which converts the softbits into the required format and inserts a softbit value of zero (the "maximum uncertainty value") for all punctured bits before feeding the sequence to the Viterbi decoder (see section 10.4 for implementation details). After the last softbit of the current code word, the Viterbi is flushed to get the remaining bits out of the core. For flushing, strong zeros are pushed to the input with data bit flag set to FALSE. The data bit flag has the same latency as the data path. Hence on the output of the core, the data bit information can be used to filter out the zeros from the flushing operation. The bits of the code word are provided at the output.

The *Descrambler* module processes the bits at the output of the decoder. If the scrambler is disabled, the input bits are bypassed to the output. On activation, detected by the rising edge of the enable signal, the Descrambler module assumes it is receiving a packet starting with the SERVICE field and uses the first seven bits to extract the scrambler seed. Those initial bits are overwritten by zeros. Afterward, all bits are descrambled with the recovered seed until deactivation.

The output is transmitted to the RX PHY state machine. Before sending to the MAC, the bit stream is filtered by the PSDU Masking module. The SERVICE, TAIL, and PAD fields are removed, and the bits are concatenated to bytes. The length of the PSDU is given by the configuration. Padding bits are removed. For the 802.11ac format, parts of the PAD field may be included in the PSDU data stream (refer to Section 7.2.7 for more information).

Module	Output Timing
Packet Termination	256 subcarriers burstwise/OFDM symbol
Align Configuration	N_{SD} data subcarriers/OFDM symbol (48-108) burstwise; gaps due to pilots
LLR Demapper	
Softbit Serializer	N_{CBPS} coded softbits/OFDM symbol (48-864) burstwise; gaps due to pilots
Deinterleaver	N_{CBPS} coded softbits/OFDM symbol (48-864) burstwise
Depuncturing	N_{DBPS} encoded stream values or data bits/OFDM symbol (24-720) Peak rate: 1 value/clock cycle
Viterbi	
Descrambler	
PSDU Masking	N_{DBPS}/8 data bytes/OFDM symbol (3-90) Peak rate: 1 byte/10 clock cycles

Table 7-6: RX Bit Processing Transfer Timing

The output timing of the submodules is given in Table 7-6. The number of values depends on the format, bandwidth, and MCS. The referred variables can be found in Tables 17-4, 17-5, 21-30 and 21-38 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. In brackets, the minimum and maximum values are given indicating the valid range. The minimum value is based on L-SIG, which uses non-HT mode with MCS 0. The maximum value is based on VHT 40 MHz transmissions using MCS 9.

The RX I/Q Processing module provides 256 subcarriers in one burst. The first module that changes this pattern is the Configuration Alignment. Only subcarriers belonging to coded fields remain on the output. Since there are multiple pilot tones, this stream contains gaps. The serialized stream on the output of the Softbit Serializer module can have much more valid items per OFDM symbol. Nevertheless, the pilot gaps remain if BPSK modulation is used, where each subcarrier is translated to one softbit by the LLR Demapper. The gaps are gone after the Deinterleaver module because the softbit stream is read burstwise from the internal memory. The Depuncturer adds gaps to this data stream when there are two valid bits of stream A and B available. Adding punctured bits does not produce gaps. The Viterbi generates data on the output as soon as traceback length input bits are provided to the input. As a result, the output is given burstwise where each burst has traceback length bits. The concatenation to byte data type of the PSDU reduces the data rate by factor 8. At a coding rate of 5/6, the peak rate is reached.

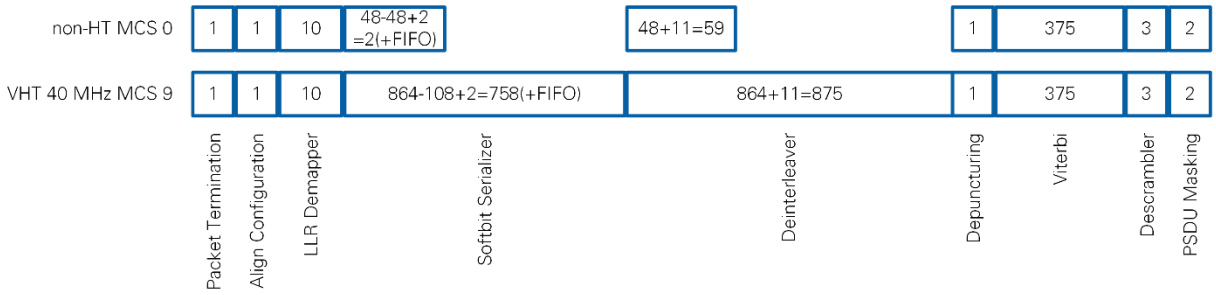

Figure 7-14: RX Bit Processing Latency

7.2.6.1 Rx Bit Processing Latency

The latency of the RX Bit Processing chain depends on the format, bandwidth, and MCS. Similar to Table 7-6, Figure 7-14 refers to the two corner cases L-SIG and highest MCS at highest bandwidth. The latency is given for the last subcarrier of the packet generated by the RX I/Q Processing module. Most of the modules have a fixed latency.

The delay of the Softbit Serializer depends on the modulation. For BPSK, each subcarrier is mapped to one softbit so the serialization does not add any delay. The internal FIFO is empty when the last value arrives. The FIFO delay is unknown. The latency is 2 because of internal registers. For 256-QAM, each softbit array has to be split into eight softbits on the output. When the last value arrives, 108 (N_{SD}) of 864 (N_{CBPS}) softbits are processed on the output. The delay for the last softbit added with the two register stages results in 758 clocks latency.

The Deinterleaver has to store one complete OFDM symbol of softbits. The read operation starts as soon as the last value arrives. N_{CBPS} softbits must be read before the last sample is available on the output of the Deinterleaver. An additional latency of 11 is incurred because of the pipeline stages.

The latency of the Viterbi decoder is determined by the chosen traceback length. An additional latency of 15 is incurred due to pipeline stages (see Section 10.4 for implementation details).

The latency for other configurations can be calculated using Equation 7-9 with values from tables 17-4, 17-5, 21-30 and 21-38 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].

$$L_{Bit\ Processing} = 406 + 2N_{CBPS} - N_{SD}(+FIFO)$$

Equation 7-9: RX Bit Processing Latency

7.2.7 RX PHY State Machine

The RX PHY state machine, which is based on Figure 21-37 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1], provides the configuration for RX IQ and RX Bit Processing modules and generates indications for the MAC. Notice also that the PHY is not capable of decoding VHT MU PPDUs, so the reception of VHT-SIG-B is skipped as described in Section 21.3.20 of *Part 11: Wireless*

LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications [1]. The state diagram is given in Figure 7-15. The word *timing* in this diagram refers to the known RXTIME due to the reception of the L-SIG.

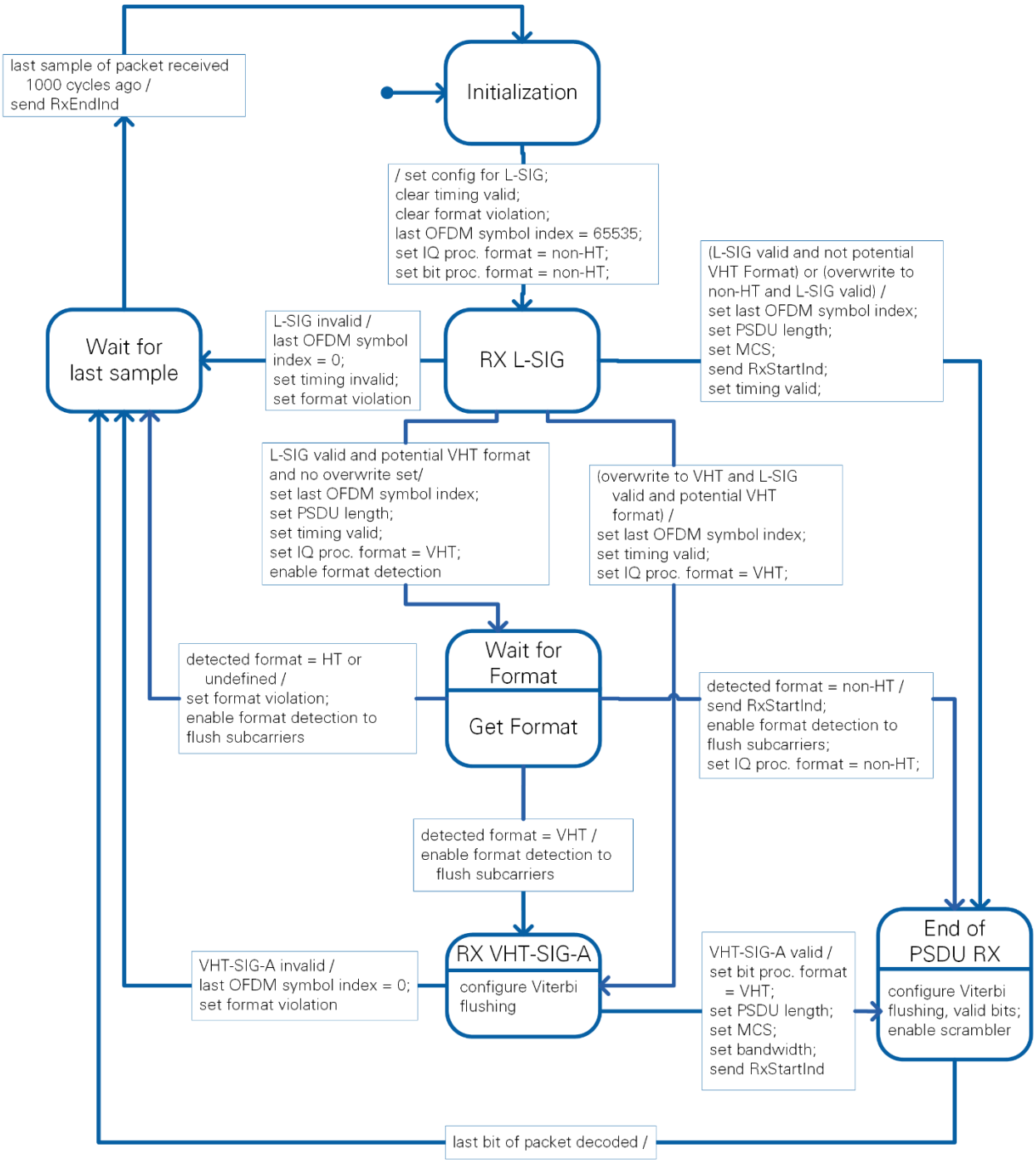

Figure 7-15: RX PHY State Machine States

Initialization: It is the startup state. In this state, the internal configuration is reset such that it can receive the first coded field in the packet (L-SIG in the primary subband). This setting consists of the non-HT format, 20 MHz bandwidth, disabled scrambler, and MCS

0. The unknown length of the packet means that the last OFDM symbol index is set to the maximum unsigned 16-bit integer value of 65,535.

RX L-SIG: As soon as the synchronization detects a packet, the processing chain uses the configuration from the Initialization state to provide the 24 bits of the SIGNAL field to the RX PHY state machine. The received bits are verified to be a valid L-SIG field based on section 17.3.4 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. The L-SIG check includes verifying the following conditions:

- R4 of RATE field is one
- Bit 4 is zero
- SIGNAL TAIL field is all zeros
- Parity bit is matching
- LENGTH>0

The result of the check is used as condition *L-SIG valid* in the state machine. As soon as this condition is evaluated the state machine leaves this state.

If L-SIG is invalid, the reception of the current packet is aborted. The last OFDM symbol index is set to 0. This forces the Sample Timing Generation module of the RX I/Q processing chain and the Packet Termination module of the RX bit processing chain to finish the current OFDM symbol and stop. Because there is no packet length information available at this point, the timing information is marked as invalid. Furthermore, the internal format violation error is reported using the PhyRxEnd indication.

If a valid L-SIG is received, the index of the last OFDM symbol is calculated based on equation 17-11 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. This index as well as MCS and PSDU length are provided to the processing chain. In addition, the packet frame timing is set to valid. There are multiple options for the following state. The format can be enforced by the host. In this case the state machine switches to *End of PSDU RX* state or *RX VHT-SIG-A* state based on the given information. If the host does not overwrite the format the next state depends whether the received PPDU could be VHT format. If so the next state is *wait for format*. Otherwise the frame is received as non-HT format frame in the *End of PSDU RX* state.

Wait for format: In this state, the format detection module in the RX IQ processing is enabled to detect the format based on the received modulation (see section 7.2.5 for details). Once the format is received non-HT PPDUs are received in the *End of PSDU RX* state. The PSDU length has already been set in L-SIG. VHT frames require the decoding of RX VHT-SIG-A which would be the next step. Unsupported formats lead to a termination of the packet reception.

RX VHT-SIG-A: Similar to the RX L-SIG state, the processing chain is configured to provide the bits of the VHT-SIG-A to the RX PHY state machine. The code word of VHT-SIG-A is provided in two OFDM symbols. The Viterbi decoder flush required flag in the bit processing configuration cluster is set for the second OFDM symbol. This bit-processing configuration cluster is aligned with the data stream in the RX Bit Processing block by the Configuration Align Configuration module (see section 7.2.6). Hence, accurate indication of the current OFDM symbol index is available from the RX Bit Processing module and can be used to set the Viterbi decoder flush required flag.

The 48 bits of the VHT-SIG-A are captured, and its validity is verified based on Section 21.3.8.3.3 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. The condition VHT-SIG-A valid is based on the following checks:

- Bandwidth is supported by PHY
- Group ID indicates VHT SU PPDU (0 or 63)
- Short GI is set to zero (disabled)
- B2 is set to zero (BCC encoding)
- CRC checksum is matching

If VHT-SIG-A is invalid, the reception is aborted, similar to an abortion out of the L-SIG state but here the timing information is known from a successful L-SIG reception.

If VHT-SIG-A is valid, you can configure the bandwidth, format, MCS and PSDU length in the processing chain. Since there is no specific length information given in VHT-SIG-A, the PSDU length is calculated using Equation 21-112 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].

Wait for last sample: The state machine waits until the Sample Timing Generation module of the RX I/Q Processing module indicates that the last sample of the current OFDM symbol has been processed. This signal is captured and indicated as done 1,000 clock cycles later outside of the state machine. It may be for correctly received frames that the 1,000 clock cycles already elapsed. The purpose of the 1,000 clock cycles is to finish processing in the RX IQ processing before configuring the RX path for a new packet reception. On leaving the state a PHY RX end indication is generated using the internal information of the RX error.

End of PSDU RX: This field is entered when the signaling information is correctly received and the data field is to be decoded. Similar to the RX VHT-SIG-A state, flushing the Viterbi decoder is enabled only for the last OFDM symbol of the packet, which is identified by the last OFDM symbol index computed in RX L-SIG state. Furthermore, in non-HT format, the number of valid data bits in the last OFDM symbol before tail and padding is known and configured to the Viterbi module so that the TAIL bits are the last to be decoded. For VHT format, the padding is inserted before tail bits. The valid bits

limitation is not used in this case. All bits of the last OFDM symbol are processed by the Viterbi decoder.

The state is left as soon as the PSDU Masking module in RX bit processing indicates that the last byte of PSDU has been decoded.

7.2.8 PHY RX Timing

The overall timing of the RX chain including Synchronization, I/Q and bit processing, and the RX PHY state machine is shown in Figure 7-16 for non-HT packets. Time is represented on the horizontal axis. On the vertical axis, several selected modules with important outputs or that change the transfer timing are displayed. The colored rectangles correspond to the data values of one OFDM symbol. The size and the placement among the time axis are related to the latencies and transfer timings of the modules. The black arrows show important control signals between processing chain and state machine and between PHY and MAC. The arrows are based on the timing information. Neither the start nor the end position must be related to the module that generates or consumes this control information.

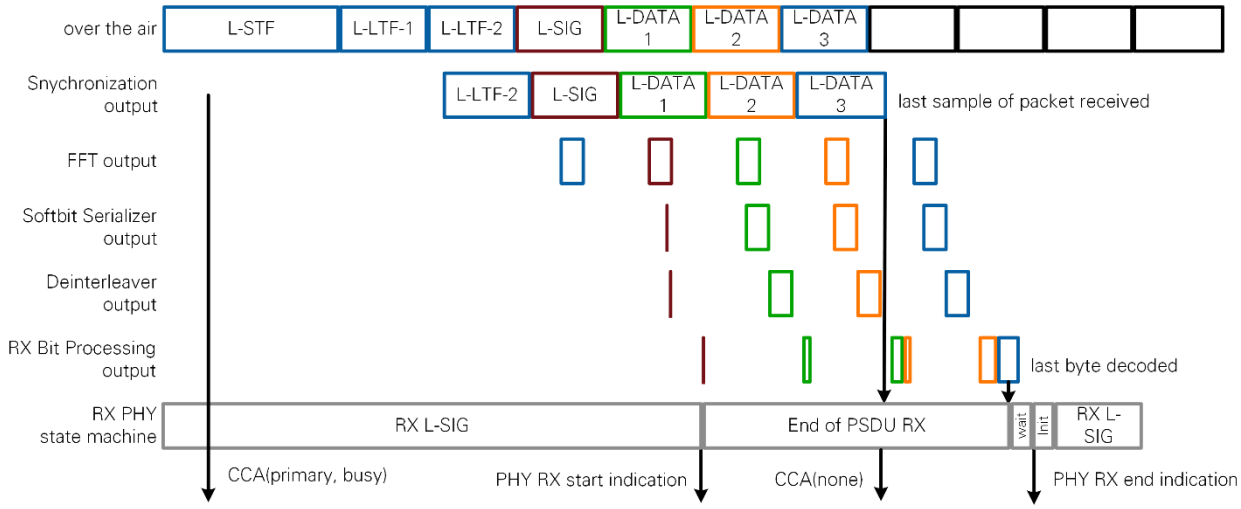

Figure 7-16: RX PHY Timing for Non-HT Packets

Figure 7-16 shows the timing of the receiver for the packet of non-HT with MCS 7 and N_{SYM}=3. RF and Synchronization add the latency between over-the-air transmission and the synchronization output. The first OFDM symbol after the packet start is L-LTF-2. The RX PHY state machine has configured the RX IQ and Bit Processing modules to receive L-SIG.

As soon as the last sample of the L-SIG field is available in the FFT the execution starts. The burstwise unloading of data is done in parallel to reception of the next OFDM symbol on the FFT module input. L-LTF-2 is terminated in the Align Configuration module of the RX Bit Processing block.

As the first dynamic field, the L-SIG is the first field handled by the RX Bit Processing. L-SIG uses MCS 0, so it has only 24 data bits, and the latency is much smaller than one

OFDM symbol duration. The decoding and flushing of the Viterbi decoder take most of the time. The RX PHY state machine can update the configuration cluster for the reception of the coded data symbols based on the L-SIG field contents long before the next OFDM symbol is unloaded by the FFT.

Starting with L-DATA-1, the RX Bit Processing chain uses MCS 7. This results in a larger number of bits on the Softbit Serializer module output. The Viterbi divides the bits into chunks of the traceback length. Because of this processing pattern, the output of the RX Bit Processing chain is not given an OFDM symbol.

The code word ends in the last OFDM symbol, and the Viterbi is flushed. Due to padding bits, the decoding can end before the last bit has been received. The PSDU Masking module notifies the RX PHY state machine to send out RX PHY end indication goes to wait for packet end state. As the last sample has entered the I/Q processing more than 1,000 clock cycles ago the state is immediately left toward the Initialization state. After the Initialization state, the RX chain is ready to process a new packet.

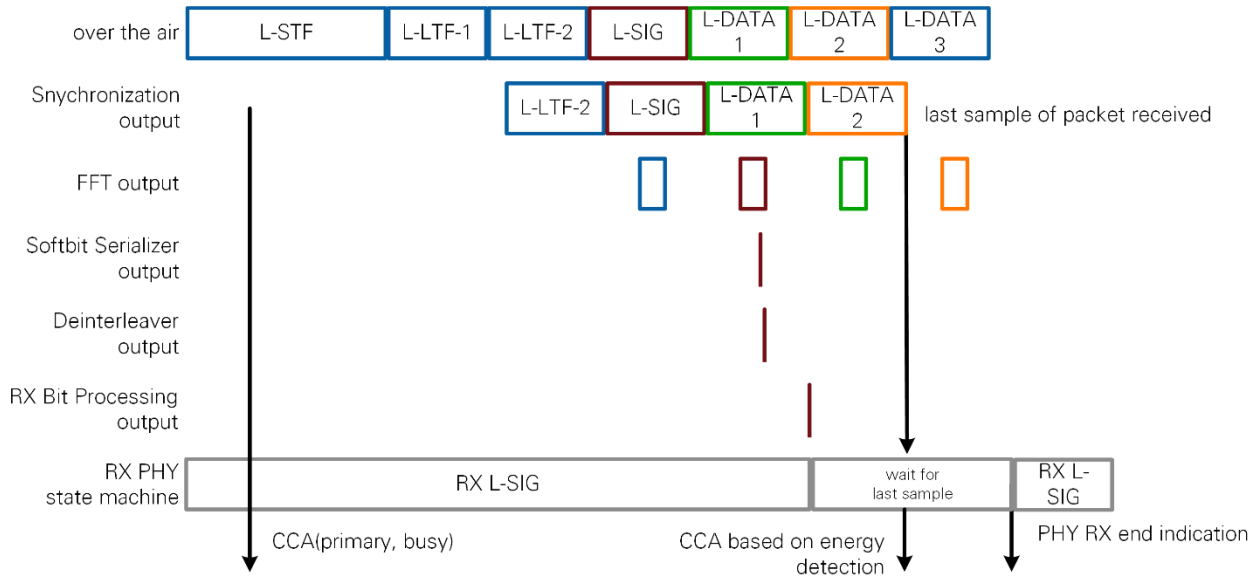

Figure 7-17: RX PHY Timing for Invalid Packets

Figure 7-17 illustrates the termination of the reception in case L-SIG was not valid. An invalid VHT-SIG-A is handled similarly. Like in Figure 7-16, L-SIG is provided to the RX PHY state machine. Once it is determined that L-SIG field contents are invalid, the last OFDM symbol index is set to zero, and the state machine goes to wait for last sample state.

At this point in time, the FFT may be filled with data from the next OFDM symbol and cannot be aborted immediately. The Sample Timing generation module in RX IQ Processing block completes the current OFDM symbol and notifies RX PHY state machine after the last sample. RX PHY state machine switches to wait for packet end state and waits for the duration of one OFDM symbol. During this time the FFT unloads

the remaining data. This data is terminated in the Packet Termination module of the RX Bit Processing chain.

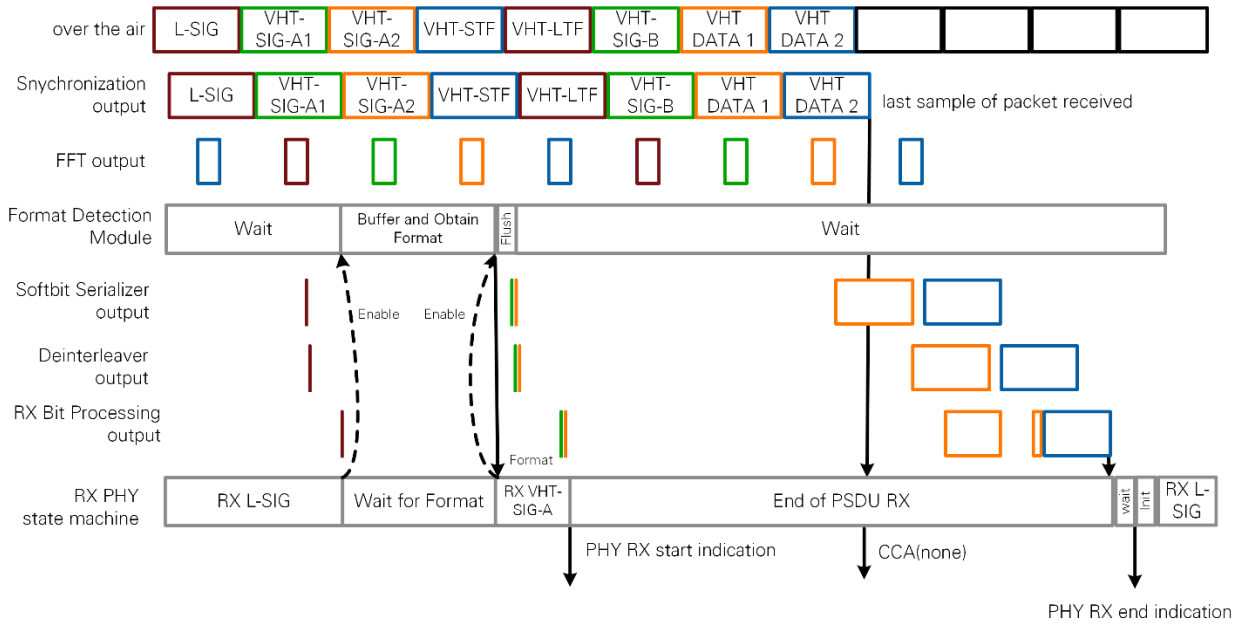

Figure 7-18: RX PHY Timing for VHT packets

The reception of a VHT packet with a bandwidth of 40 MHz at MCS 9 is shown in Figure 7-18. Since the process before L-SIG is equal to Figure 7-16, it is left out. After the L-SIG field, the RX PHY state machine switches to wait for format state and enables the format detection module. The module consumes the samples of VHT-SIG-A and reports the format back to the RX PHY state machine. Once it is determined that the format is VHT, the state machine transitions to RX VHT-SIG-A state. The format detection module is instructed to flush the consume samples. Those are given burstwise, which might cause the frequency offset index to leave out subcarriers from the FFT output, which are not data subcarriers used for VHT-SIG-A. The timing of VHT-SIG-A reception is similar to L-SIG in the Bit Processing chain. Because the VHT-SIG field only has a small number of bits and the Viterbi code is flushed at the end of VHT-SIG-A2, the 48 bits arrive in one burst at the RX PHY state machine.

If VHT-SIG-A is determined to be invalid, the reception would be aborted similar to the L-SIG invalid case illustrated in Figure 7-17. In this case, VHT-STF would be the last OFDM symbol getting out of the FFT.

If VHT-SIG-A is valid, the parameters bandwidth and MCS are obtained from the field and used to set the configuration clusters for the processing chain. PHY RX start indication is sent to MAC and the RX PHY state machine transitions to the End of PSDU RX state and waits for end of decoding.

The next OFDM symbols contain training sequences and VHT-SIG-B. This information is not handled in RX Bit processing chain.

In this example, the RX Bit Processing is for MCS 9, which consists of 256-QAM modulation. This scenario results in a large number of bits generated by the LLR Demapper, which are serialized by the Softbit Serializer. Reading and writing the Deinterleaver memory overlaps for this large number of bits is the reason for having a double page memory in this module. The Viterbi is flushed on the last OFDM symbol as in non-HT format. RX PHY end indication is sent by the state machine if the last byte has been provided to MAC.

7.3 PHY TX

The PHY transmitter receives the PhyTxStart request and the corresponding PSDU data from the MAC and generates the PHY frame. A high-level block diagram is shown in Figure 7-19.

- The **TX PHY SAP** is the interface to the MAC. It provides the PhyTxStart request and the PSDU data.
- The **PHY TX Request Handler** interprets the PhyTxStart request and creates configuration for the submodules of the PHY TX.
- The **PSDU Discard** module discards the PSDU data if the PhyTxStart request did not pass the consistency check (for example, invalid MCS, PSDU length=0, or bandwidth not supported by the USRP device model).
- The **TX Bit Processing** includes serialization, scrambling, convolutional encoding, puncturing and interleaving. It prepares the data for the signal field (L-SIG, VHT-SIG-A and VHT-SIG-B) and for the data field.
- The **TX IQ processing** generates the whole PHY frame which consists of multiple OFDM-symbols. This includes the training fields (L-STF, LTF, VHT-STF, VHT-LTF), the signal fields, and the data field. For each field, the following operations are performed: channel duplication, channel rotation, IFFT prescaling and FFT with cyclic prefix (CP) insertion. The training fields are read from a memory ("assembly" modules) whereas the subcarrier data for the signal fields and the data field is taken from the bit processing.
- The **TX Data Sink** module writes the PHY frame (time-domain signal) to the following FIFOs:
    - TX to RF (target scoped): The signal is passed to the RF for transmission over the air.
    - Internal loopback (target scoped): The signal is passed to the RX PHY of the same device. This way the RF is bypassed. This operation mode is useful for debug purposes and not used in normal operation.
    - T2H TX Baseband: The signal is passed to the host where it is used for displaying the TX power spectrum. A filter is used to select only a subset of the samples. The purpose of this filter is to reduce the data rate which is sent to the host.

All modules operate in the baseband clock domain of 250 MHz. Every module is designed to keep up with the data rate from the upstream module, so there is no need for throttle control inside the modules.

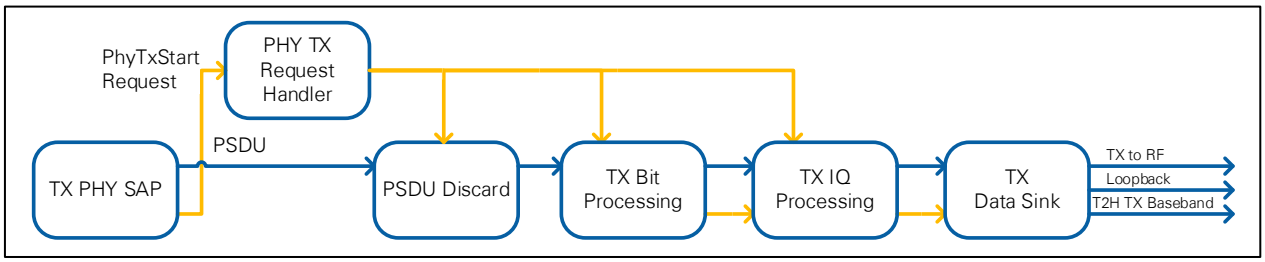

Figure 7-19: PHY TX Block Diagram

7.3.1 TX Request Handler

The TX Request Handler performs a consistency check on the incoming PhyTxStart request (for example, invalid MCS, PSDU length=0, or bandwidth not supported by the USRP device model) and generates various signals.

The following signals are needed by the other PHY TX modules:

- **TX vector**: The captured PhyTxStart request (valid only if it passed the consistency check)
- **PSDU discard configuration**: the configuration for the PSDU Discard module. If the PhyTxStart request was invalid and the PSDU data shall be discarded, the "discard PSDU" flag is set to True.
- **Start of packet**: Pulse signal which marks the start of a packet. It is generated immediately after a valid PhyTxStart request was received. It is used by the TX IQ processing module to send out the L-STF immediately.
- **Symbol trigger:** Pulse signal which marks the beginning of an OFDM symbol. The timing is aligned with the OFDM symbol pulse which is generated in the RF loop.
- **Symbol index**: Index of the OFDM symbol which shall be processed by the TX Bit processing and TX IQ processing upon receiving the trigger. The initial value is 2, that is, the L-STF field will be skipped because it is handled separately. The TX Bit processing internally adds 2, that is, it skips the L-LTF field and starts processing the L-SIG field.

The following signal is needed by the PHY RX:

- **TX active**: the Boolean flag is set to True while the TX modules are handling a valid request. The flag is used by the PHY RX to mute the RX signal which is received from the RF module. This ensures that a packet generated by the local PHY TX is not received by the local PHY RX.

The following signal is needed by the MAC: **PhyTxEnd indication**, which informs the MAC that the PhyTxStart request was processed. If the request was valid, the

indication is sent after the frame was generated completely. If the request was invalid, the indication is generated immediately (after the PSDU was discarded) and the "unsupported mode" flag is set to True.

7.3.2 TX Bit Processing

The purpose of the TX Bit Processing module is to generate the signal fields and enqueue the PSDU into the data stream. This stream is then serialized, scrambled, encoded, punctured and interleaved before it is passed to TX I/Q Processing. Its block diagram is shown in Figure 7-20. The types of the data path and the elements of the control path are listed in Table 7-7.

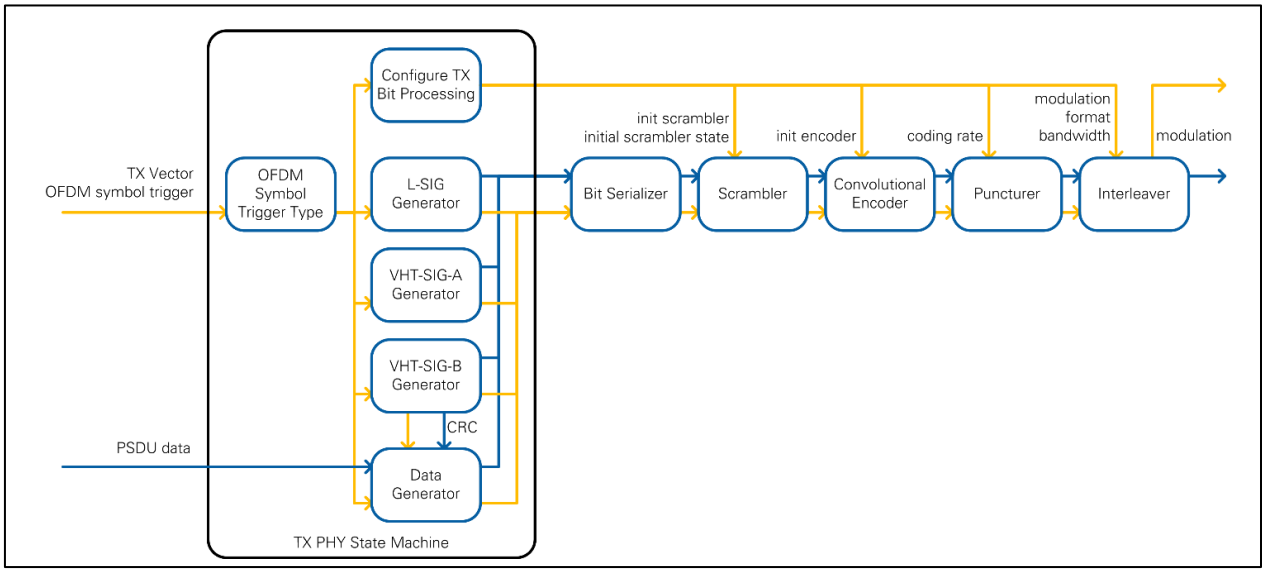

Figure 7-20: TX Bit Processing Block Diagram

Module	Output Data Type	Output control information
MAC TX	U8	
TX PHY State Machine	U32	TX bit processing parameter length[11] enable scrambler
Bit Serializer	Boolean	enable scrambler
Scrambler	Boolean	
Convolutional Encoder	Boolean array (2 elements)	
Puncturer	Boolean	
Interleaver	Boolean	packet configuration

Table 7-7: TX Bit Processing Data Types and Control Information

The first module in TX Bit Processing is the TX PHY State Machine which encodes the signal fields according to section 17.3.4 (L-SIG), 21.3.8.3.3 (VHT-SIG-A) and 21.3.8.3.6 (VHT-SIG-B) of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. The Data Generator VI furthermore turns PSDU into a

[11] Defines actual length of U32 output

sequence of SERVICE field, PSDU data, TAIL, and PADDING for the 802.11a format according to 17.3.5 in *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1] or SERVICE field, PSDU data, PADDING and TAIL for 802.11ac format according to 21.3.4.9 in *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1], respectively. The generator outputs are combined using the enable driven stream combiner (EDSC) pattern (see appendix, section 10.1.1). The TX Bit Processing has a head start of two OFDM symbols to have the first bits available when needed by the TX I/Q Processing. This is due to the pre-generation of L-STF (two OFDM symbols) in time domain; the I/Q data of L-STF is stored in a block RAM.

Each signal field is generated in one burst. The data bits are generated as one continuous burst per OFDM symbol in dependence on N_{DBPS}. A small FIFO with a four-wire handshake ensures that bytes for at least one OFDM symbol are available. Furthermore, TAIL and PADDING bits are also generated as part of the corresponding burst. In the worst case, the bit processing chain generates bits for up to two OFDM symbols in one burst, which is compensated in a FIFO of the TX IQ Processing Data Assembler.

After the downstream module is the Bit Serializer module, which converts data fields and PSDU data into one bit per cycle. At the start of the module, a FIFO is used to ensure that the module can process the incoming data rate. The maximum number of data bit per symbol N_{DBPS} is 720 (802.11ac, 40 MHz, MCS 9). Because PSDU data is given in bytes, the FIFO must store at least 90 samples.

After bit serialization, scrambling, convolutional encoding, puncturing, and interleaving are applied as described in 17.3.5.5 – 17.3.5.7 and 21.3.10.4 – 21.3.10.8 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. The scrambler and the encoder must be reset before the first bit of the data field is processed. The Scrambler module is bypassed for signal fields. The Puncturer module serializes the two streams of the convolutional encoder using the puncturing patterns of Figure 17-9 and 19-11 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. A FIFO is used on the input of the module since the data rate is higher on the input. The Interleaver applies the BCC interleaver operations, which are defined in Section 17.3.5.7 and 21.3.10.8 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. The write operation into the memory is based on equation 21-77 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1], which applies the first permutation. The read operation is based on equation 21-82 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1], which applies the second permutation. Reading is started as soon as all bits of the current OFDM symbol are saved to memory. A double page memory is used, which enables reading and writing at the same time. The output of the Interleaver and the

modulation scheme that is used are provided on the output of the TX Bit processing module.

Module	Output Timing
L-SIG Generator	U32 on start of L-SIG processing
VHT-SIG-A Generator	Two U24 on start of VHT-SIG-A processing burstwise
VHT-SIG-B Generator	U32 on start of VHT-SIG-B processing
Data Generator	N_{DBPS}/8 U32/OFDM symbol burstwise
Bit Serializer	N_{DBPS} bits or array of bits/OFDM symbol burstwise
Scrambler	
Convolutional Encoder	
Puncturer	N_{CBPS} bits/OFDM symbol burstwise
Interleaver	

Table 7-8: TX Bit Processing Transfer Timing

The output timing of the submodules is given in Table 7-8. All submodules of the TX PHY state machine generate data on the asserted enable signal from the OFDM symbol trigger type module. The modules need up to two U32 words. Starting with the OFDM symbol for the data field, the Data Generator provides the required number of bytes. After the Bit Serializer, N_{DBPS} clock cycles are needed to complete the transfer. The rate ½ convolutional encoder doubles the number of bits but due to the transfer of an array, the number of transfers is not changed. After puncturing, N_{CBPS} bits remain.

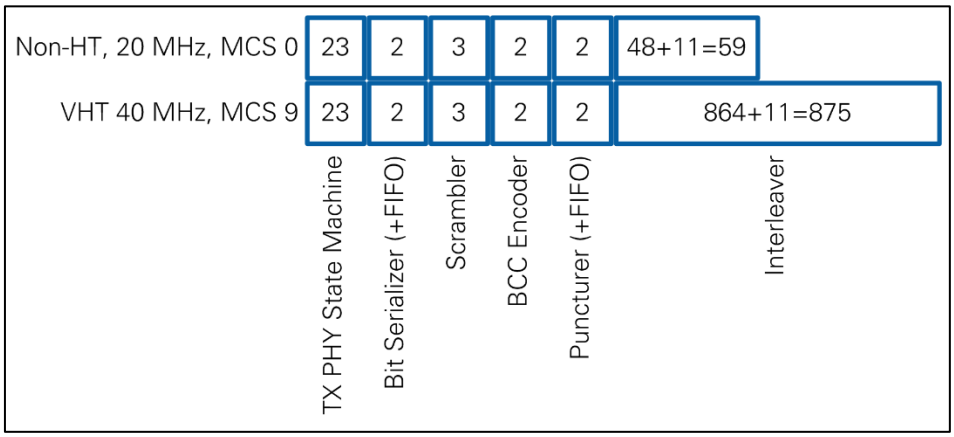

Figure 7-21: TX Bit Processing Latency

The latency of the bit processing chain depends on the format, bandwidth, and MCS. Similar to Table 7-8, Figure 7-21 refers to the two corner cases of non-HT mode with MCS 0 and highest MCS at highest bandwidth. The latency is given for the first subcarrier of the packet. Thus, this is when the TX bit processing chain needs from start trigger until the first valid bit is provided. Most of the modules have a fixed latency.

The Interleaver must store bits from one complete OFDM symbol. The read operation starts as soon as the last value arrives. N_{CBPS} bits must be read before the last sample is

available on the output of the Interleaver. An additional latency of 11 comes from the pipeline stages.

The latency of the FIFOs in Bitserializer and Puncturer are unknown.

7.3.3 TX IQ Processing

The purpose of the TX IQ Processing module is to add the training fields and to convert the bits from TX Bit Processing into baseband I/Q samples. The OFDM symbol trigger of the RF loop is used to clock the generation of the OFDM symbols. The block diagram is illustrated in Figure 7-22. The data types and control information are listed in Table 7-9.

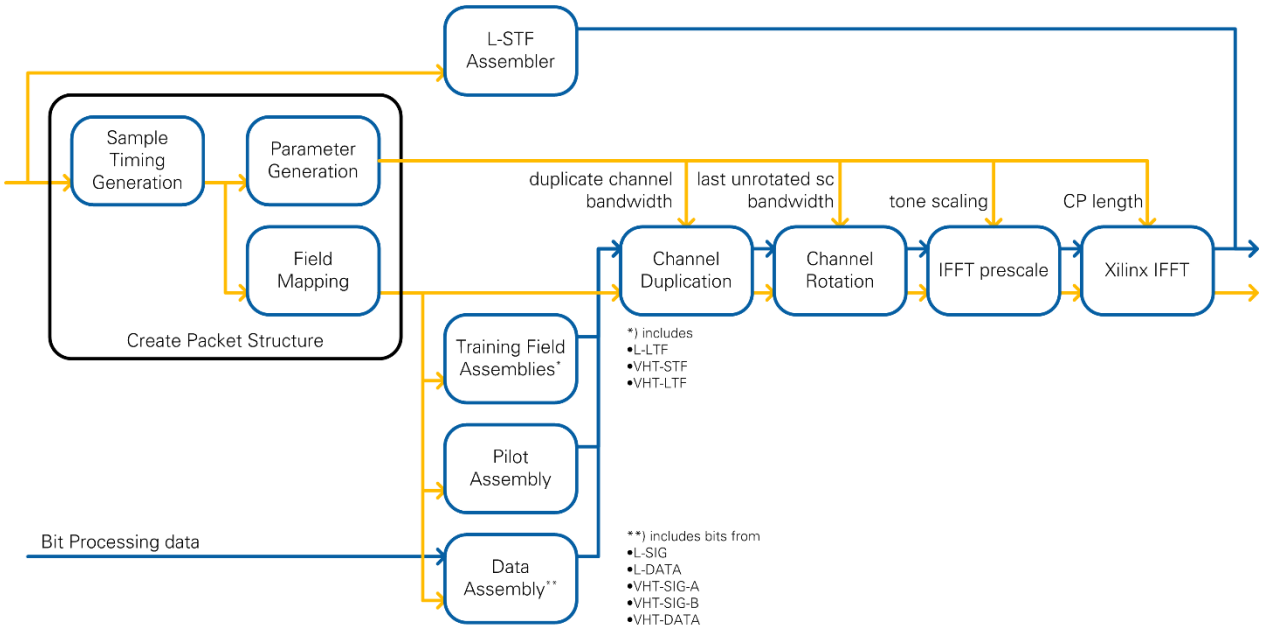

Figure 7-22: TX IQ Processing Block Diagram

Module	Output Data Type	Output control information
TX Bit Processing	Boolean	
Create Packet Structure		Field Map Subcarrier Timing TX IQ Processing Parameter
L-STF Assembler	CFX 3.13[12]	
Assembler modules	CFX 2.14	
Channel Duplication	CFX 2.14	
Channel Rotation	CFX 2.14	
IFFT Prescale	CFX 0.16	
Xilinx IFFT	CFX 3.13	

Table 7-9: TX IQ Processing Data Types and Control Information

[12] Bypasses all following modules

The Create Packet Structure modules creates a timing structure, a field map, and the processing parameters that stay constant along the OFDM symbol. These parameters can include bandwidth, CP length, channel duplication, channel rotation, and tone scaling factor.

The field map controls which field is generated for the current OFDM symbol. Using the EDSC pattern (see appendix, section 10.1.1) for field generation reduces the latency caused by parallel execution. There is one assembler for each training field, for the pilots, and one for the bit taken from TX Bit Processing module.

TX IQ Processing must start delivery of I/Q data as soon as possible after the TX start request is triggered. Because the IFFT takes about half an OFDM symbol (as described in section 7.3.4), the L-STF is pregenerated in the time domain, and its I/Q data is stored in a block RAM. For each combination of bandwidth and primary subband, a bank is reserved in the memory. Because L-STF is a Non-HT field, you do not need to distinguish between 802.11a and 802.11ac. For 802.11a, an additional bank for DC centered signal exists. Because L-STF is a repeating sequence in time domain with a period of 0.8 µs, you need to store only 64 samples.

The remaining training fields are generated according to Sections 21.3.4.3 (L-LTF), 21.3.4.6 (VHT-STF), and 21.3.4.7 (VHT-LTF) of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].

The L-DATA and VHT-DATA are built from bits generated by TX Bit Processing modules. The bit stream is buffered in a FIFO that is laid out to buffer up to three OFDM symbols, which are the bit processing head start, the current OFDM symbol, and the last OFDM symbol, if filled with padding. Besides applying the correct modulation, the module also rotates VHT-SIG-A2 according to 21.3.8.3.3 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1].

The pilot tones are inserted according to 21.3.10.10 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1] using the information from the field map.

After the assembler modules, the channels are duplicated and rotated according to Sections 21.3.4.x and 21.3.7.5 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]. Here, the Create Packet Structure modules ensure correct settings for handling channel duplication and rotation.

Next downstream module is the IFFT prescale. This module applies tone field scaling according to 21.3.7.4 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1] to ensure that the time domain power of VHT modulated fields does not exceed the time domain power of pre-VHT modulated fields (each summed over all transmit channel). The scaling factor is determined in the Create Packet Structure module and depends on bandwidth and field type (refer to table 21-8

of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]).

The last module is the IFFT, which is a wrapper around the Xilinx IFFT core. It contains a 256-point IFFT operation using "Radix 4, Burst I/O" architecture similar to the RX IQ processing. In addition, the configuration input is used to enable cyclic prefix insertion by the core. The core configuration settings, such as this input, are dynamic and are provided parallel to the first sample of each OFDM symbol. A small FIFO is placed before the IFFT input to compensate the longer execution time due to guard interval GI2 of L-LTF-1. The IFFT output is shifted in frequency using a toggling negation due to the implementation on FPGA. The fixed point format on the output is CFX 3.13 based on the requirements of section 3.4.3.3.

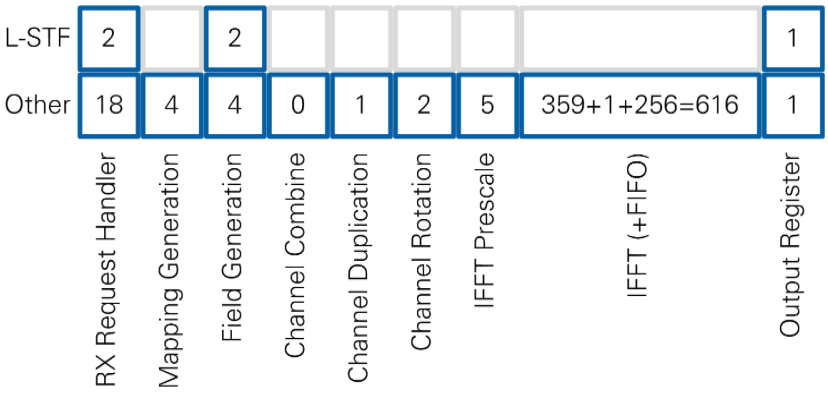

Figure 7-23: TX IQ Processing Latency

There are two paths for latency in TX I/Q processing. One goes for the pre-calculated samples of the L-STF in time domain. This path has only a latency of five cycles, which allows the PHY to ensure a packet starts at the interface to the RF when it is trigger inside the PHY. The second path goes for all the other symbols that are created using the path shown in Figure 7-23. The FFT latency is smaller than reported by the Xilinx core because the reported value includes loading of all 256 samples. During the packet, the FFT executes and unloads samples in 616 clock cycles after the last sample arrived. The remaining clock cycles per OFDM symbol are used to transfer data from the input FIFO to the FFT core. By the time the last sample is available on the input, the FIFO is empty and it is passed to the core as fast as possible. The delay of the FIFO is unknown. All other modules have a fixed latency.

7.3.4 PHY TX Timing

The overall timing of the TX chain including TX Bit processing and TX IQ Processing is shown in Figure 7-24 for packets in 802.11ac format. The timing works similarly for 802.11a packets. Time is represented on the horizontal axis. On the vertical axis only a couple of module outputs are chosen that are important or change the transfer timing. The colored rectangles correspond to the data values of one OFDM symbol. The size and the placement among the time axis are related to the latencies and transfer timings

of the modules. The black arrows show important control signals inside the processing chain and between PHY and MAC. The arrows are based on the timing information.

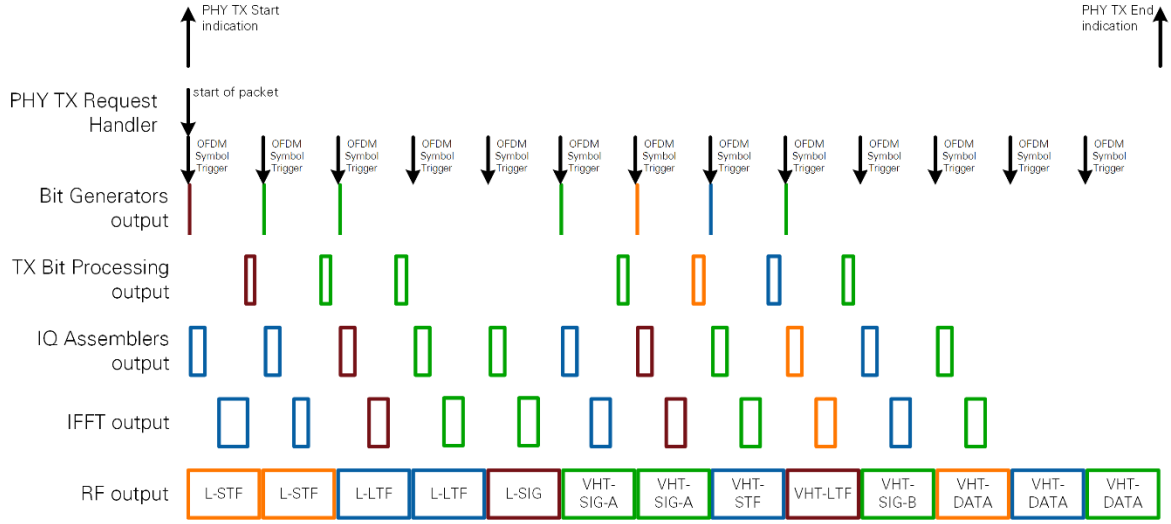

Figure 7-24: TX PHY Timing for 802.11ac Packets

Figure 7-24 shows the timing of the transmitter for an 802.11ac packet. The RF transmission starts with the start of packet trigger that is given with the start of the first OFDM symbol. This is possible because the L-STF is unloaded from memory in time domain. The latency is only a few cycles. In parallel, the bit processing starts with the encoding of bits for the L-SIG and the I/Q processing starts assembling the L-LTF. The head start of four symbols for the bit processing and two symbols for the I/Q processing is kept during the packet generation. This design ensures that all samples arrive in time for I/Q processing and RF.

Assembling the bits takes only a few cycles. Bit insertion when the code rate is applied in convolutional encoding and puncturing causes the chain length of valid samples to increase at the end of TX Bit processing. The samples leave the bit processing as a burst because the interleaver handles data in groups on N_{CBPS}.

The assemblers in RX IQ Processing module append training fields and add pilots to the fields generated by TX Bit Processing module. Each OFDM symbol will contain 256 I/Q samples. The IFFT transforms this I/Q data into time domain and adds guard interval. This transformation takes 360 cycles plus FIFO delay.

In case of an invalid TX Request, there is no packet generation at all and the PHY TX Request handler generates the TX end indication immediately.

7.4 RF Module

The digital downconversion (DDC) and digital upconversion (DUC) modules are based on the PXIe Streaming project templates of USRP devices and NI-579x modules. Their block diagrams are shown in Figure 7-25 and Figure 7-26. In the DDC path, a DC Offset Correction module is present to estimate and compensate the residual DC offset from

the RX LO. This module mitigates the impact to the autocorrelation computation within the Synchronization block. The DC Offset Correction module estimation uses an average over 512 samples. After each averaging window, the LSB of the correction value is increased or decreased. Over time, the correction value is approaching the DC offset iteratively.

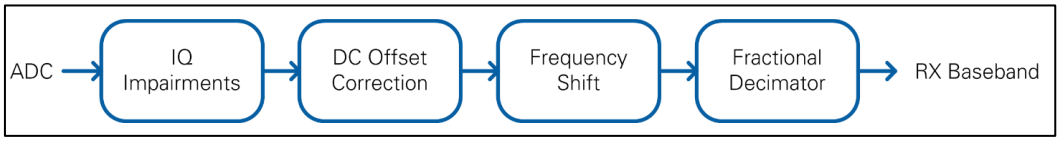

Figure 7-25: DDC Block Diagram

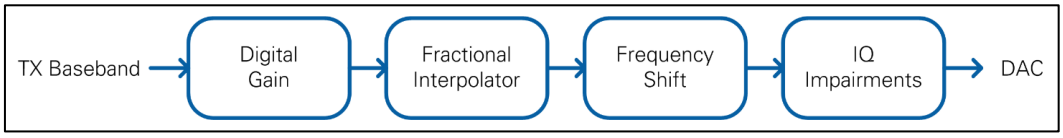

Figure 7-26: DUC Block Diagram

The latencies of DDC and DUC are given in Figure 7-27 and Figure 7-28. The Fractional Decimator and Interpolator latencies depend on the ratio of clock rate versus sample rate. Since the clock rate is different between the USRP and FlexRIO hardware, the DDC has a target-specific latency. The latency for the DUC remains the same for both target types.

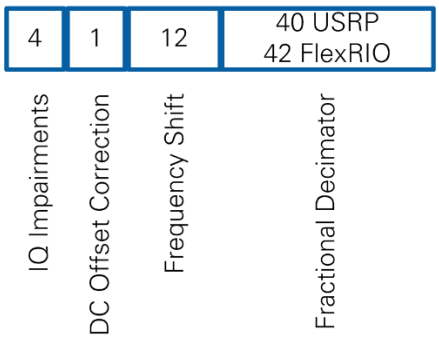

Figure 7-27: DDC Latency

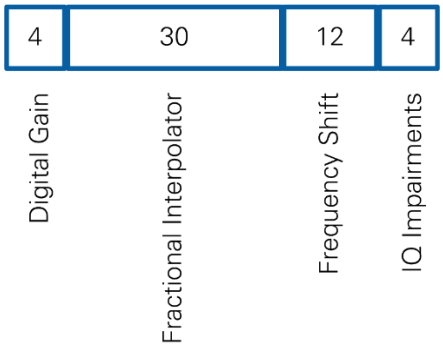

Figure 7-28: DUC Latency

The analog parts of the device and FPGA logic that are not presented on the block diagram add latency to the RF path. Those can be measured using RF loopback and the Streaming project templates for the specific target. The results are listed in Table 7-10.

	USRP RIO 40 MHz bandwidth (Data clock = 120 MHz)	**USRP RIO 120 or 160 MHz bandwidth (Data clock = 200 MHz)**	**FlexRIO or front-end adapter module (FAM) (Data clock = 130 MHz)**
DDC	57 clock cycles ≈ 0.48 µs	72 clock cycles ≈ 0.36 µs	59 clock cycles ≈ 0.45 µs
DUC	40 clock cycles ≈ 0.33 µs	50 clock cycles ≈ 0.25 µs	40 clock cycles ≈ 0.31 µs
Others (ADC, DAC, and so on)	100 clock cycles ≈ 0.83 µs	50 clock cycles ≈ 0.25 µs	115 clock cycles ≈ 0.88 µs
RF Round Trip Time	**197 clock cycles ≈ 1.64 µs**	**172 clock cycles ≈ 0.86 µs**	**214 clock cycles ≈ 1.65 µs**

Table 7-10: RF latency

8 Performance

8.1 TX EVM

The TX error vector magnitude (EVM) is -35 dB or better for the peak TX power level settings as given in Table 8-1. Measurements have been taken using NI WLAN Analysis Soft Front Panel Rel. 14.0 and the NI 5644R VST.

	NI USRP-2942		**NI USRP-2943**		**NI 5791**	
Frequency	**Min.**	**Max.**	**Min.**	**Max.**	**Min.**	**Max.**
2.45 GHz	-8 dBm	19 dBm	-6 dBm	21 dBm	-24 dBm	7 dBm
5.85 GHz	-	-	-18 dBm	10 dBm	-	-

Table 8-1: Minimum and maximum peak TX power level for EVM = -35 dB or better

8.2 Minimum RX Sensitivity

Minimum RX sensitivity for MCS 0 (BPSK, 1/2) and MCS 7 (64 QAM, 3/4) is presented in Table 8-2. The measurements follow the rule given in Section 17.3.10.2 of *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [1]:

1. The packet error ratio (PER) shall be 10% or less when the PSDU length is 1,000 octets.
2. The minimum input levels are measured at the antenna connector.

Measurements have been taken using WLAN Generation Toolkit 14.0 and the PXIe-5644 Vector Signal Transceiver.

	USRP-2953		NI-5791	
Frequency	**MCS 0**	**MCS 7**	**MCS 0**	**MCS 7**
2.437 GHz	-82.5 dBm	-68.5 dBm	-74 dBm	-62.5 dBm
5.85 GHz	-75 dBm	-66.4 dBm	—	—

Table 8-2: Minimum RX Sensitivity

8.3 Performance Throughput

The implementation of the middle/lower MAC in combination with the PHY reaches the full theoretical throughput. Table 8-3 shows the throughput values for different sequence types and MCS and the following for a certain configuration (802.11a 20 MHz, backoff=3, packet size=1,000 Bytes). The throughput values are reached with a reduced host application where a data source writes to the lower MAC on the FPGA directly and when using cables. Please note that the achievable throughput when interfacing with the Middle MAC SAP can be lower.

Table 8-3: Maximal Throughput [Mbit/s] for the Configuration: 802.11a 20 MHz, Backoff=3, Packet Size=1000 Bytes

Sequence type	**MCS**							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
DATA	5.49	7.99	10.4	14.78	18.82	25.55	31.6	33.73
DATA | ACK	5.28	7.54	9.65	13.31	16.5	21.45	25.56	26.94
RTS | CTS | DATA | ACK	4.87	6.73	8.36	10.98	13.05	15.97	18.14	18.83

9 Conclusion

LabVIEW Communications 802.11 Application Framework provides a real-time 802.11 implementation running on NI SDR hardware. This framework enables you to focus on a specific area of research by utilizing the existing link and only making changes or additions where desired.

Because of the flexibility of LabVIEW and the modularity of the framework, you can easily exchange portions of the design for prototyping new algorithms for future wireless systems. In addition, LabVIEW's native interface between the host and the FPGA means that the design can be partitioned to profit from the parallel execution on the FPGA as well as calculations on the host.

The application framework provides a comprehensive set of features. The overall architecture described in chapter 3 allows to extend the application framework towards features such as support for MIMO, 802.11p PHY extensions, QoS support, or 802.11p MAC extensions.

The application framework offers a variety of starting points for wireless research and prototyping. Start now by downloading an evaluation copy of LabVIEW Communications at www.ni.com/labview-communications.

Questions? Email us at labview.communications@ni.com.

10 APPENDIX

10.1 Design Pattern

10.1.1 Enable Driven Stream Combiner

The enable driven stream combiner (EDSC) is used to combine information from different sources into one stream. You can use the EDSC to map different fields into the frequency domain of one OFDM symbol. The idea is illustrated in Figure 10-1 for 3 computational VIs.

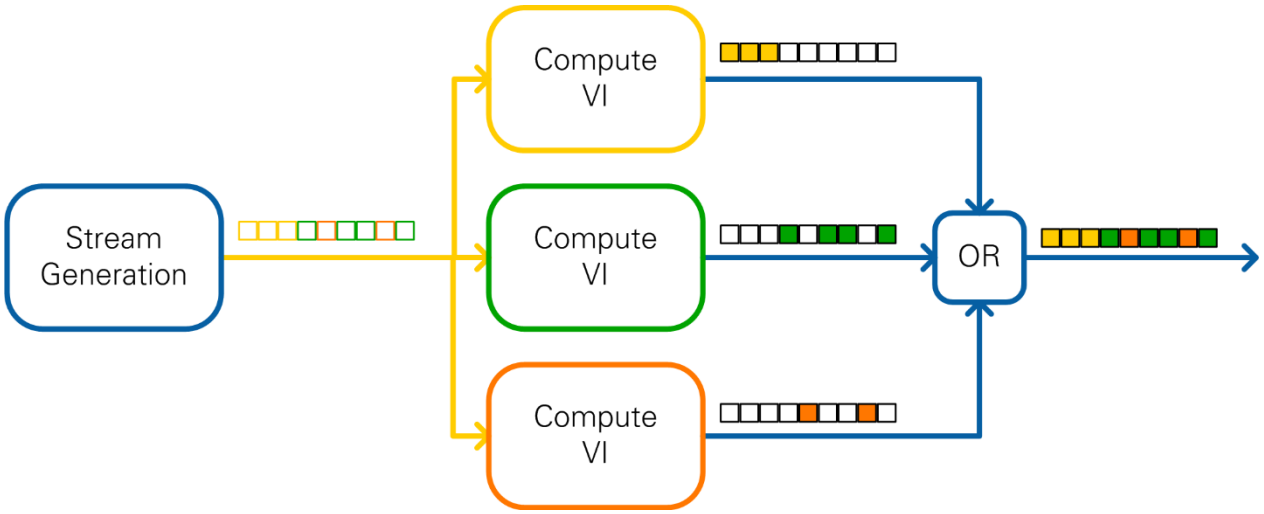

Figure 10-1: Enable Driven Stream Combiner

There is a single Stream Generation VI that generates control information for the computational VIs. This comprises an enable signal for each computational VI. Only one of these enable signals is asserted at a given time. The asserted enable signal defines the structure of the stream that should be generated. Afterwards, an unlimited number of computational VIs provide their data on the output whenever the corresponding enable signal is asserted. When the enable signal is not asserted, the output is zero. Because of this constraint, a simple OR gate can be used to combine the streams.

There is no throttle control mechanism in this design pattern. The assumption is that the computational module can always provide data when the enable signal is asserted. This must be ensured by the stream generation before start.

If the computation of one of the VIs require pipelining, the other paths between Stream Generation VI and the OR gate have to be delayed to equalize path latencies. Since the

output of the computation usually has a wider bit width than the enable signal, it is recommended to add the delay before the computational modules.

10.2 Protocols

10.2.1 Event FIFO

The application framework utilizes a target-to-host FIFO. It is reserved for event messages, that are passed between modules on the target. The message must be small in size. All information must fit into a 64-bit integer, where eight bits are reserved for the event ID. These IDs must be consecutive, unique numbers starting with zero.

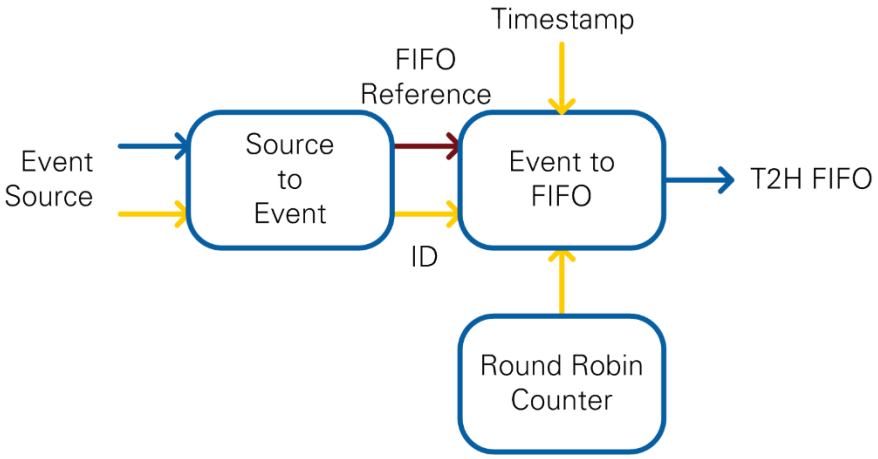

Figure 10-2: Event Arbitration on FPGA

The arbitration of the FIFO events in one clock domain on the FPGA is illustrated in Figure 10-2.

There is an event specific conversion VI that creates an U64 integer from the event information and contains the condition for writing an event. If the condition is met, the U64 value is written to a local FIFO. This FIFO ensures that there is no loss of events because of the arbitration for the target-to-host FIFO. The conversion VI also contains the unique ID for this event. It is available on the output of this VI and is part of the U64 word.

The local FIFO reference and the event ID are provided to a common Event to FIFO VI that must be instantiated once per event. It contains the arbitration logic for the target-to-host FIFO. The value of the round robin counter is checked against the given event ID. If a match is detected, two U64 words are written to the target-to-host FIFO. The first word contains the timestamp while the second word is the event data read from the local FIFO reference. Since the event IDs are unique, there is only one instance of the Event to FIFO VI writing to the FIFO.

Due to the round robin scheduling and two words per event, the event FIFO throughput is limited. The rate for each event must be lower than two times the total number of events. Events can appear at a higher rate if the local FIFO is able to store those values.

Since the timestamp is added in the Event to FIFO VI, the order of the events in the target to host FIFO is not related to the correct order of arrival on the FPGA. One event that occurs burstwise needs multiple arbitration rounds to get written to the FIFO. These multiple arbitration rounds increase the time difference between arrival and timestamp value.

10.3 Component and Namespace Layout

With the release of LabVIEW Communications 2.1, all project elements can be organized in components, and all VIs can be unequivocally identified by namespaces. A component can be assigned easily to one or more targets in SystemDesigner. A component must not include any element that is not suited for the target the component is assigned to.

10.3.1 Component Layout

Because each element of the component should be openable on the target the component is assigned to, NI recommends that you separate module implementations into one component for FPGA and one for Host code. If Host and FPGA share some resources like typedef, create a third component to be used by both the Host and FPGA components.

10.3.2 Namespacing

Each element of a component receives a namespace which consists of the root namespace of the component and an inner component namespace. Note that the folder hierarchy of the component maps 1:1 to the inner namespace as shown in Figure 80. The namespace layout is independent of the component layout; two components can share the same root namespace as shown in Figure 81.

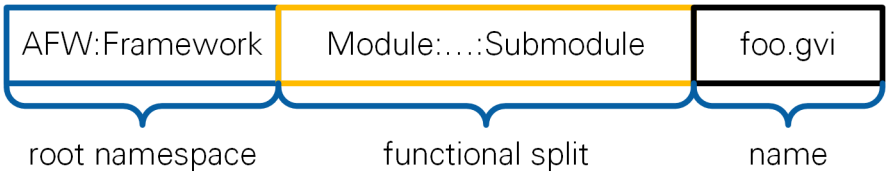

Figure 3 Namespace Hierarchy

All application frameworks components have `AFW` as the first level of their namespace. `AFW` is followed by the name of the framework as the second hierarchy level. Elements that are shared between several frameworks have `AFW:Common` as their root namespace. The component target, such as FPGA or Host, is never part of the namespace. The inner namespace corresponds to the functional split inside the component.

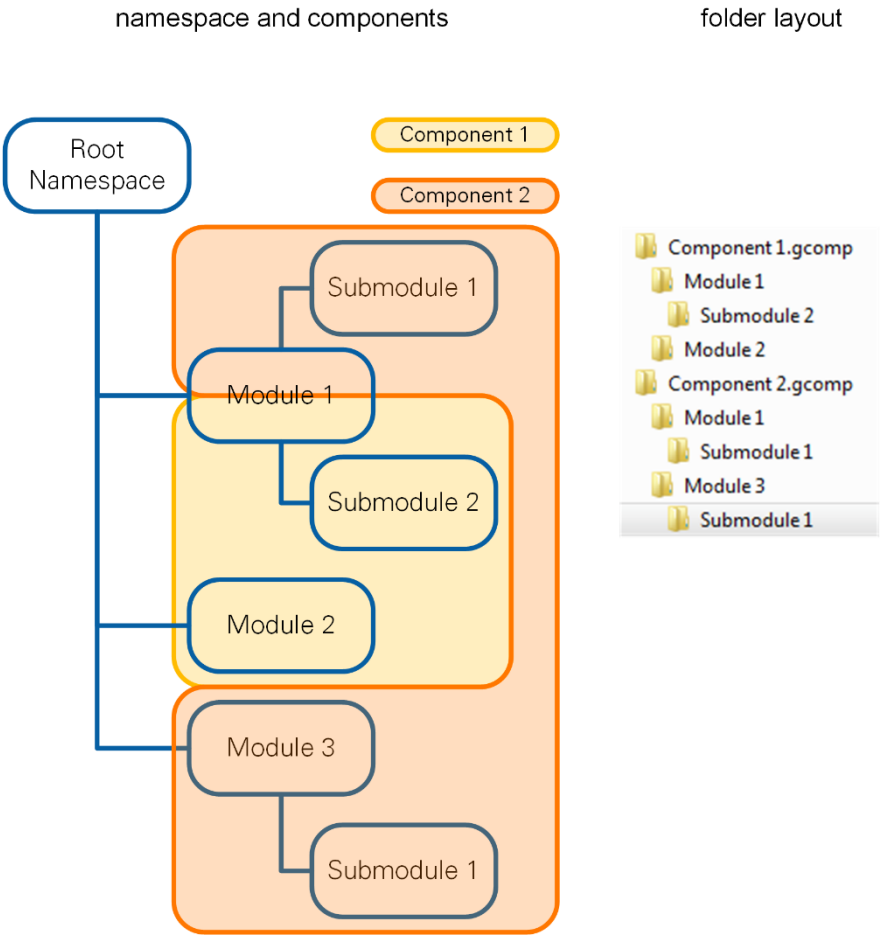

Figure 4 Connection between Namespaces, Components and Folder Layout on Disk

10.3.3 Extension of the Framework

To prevent conflicts with current and future versions of the framework, choose a root namespace that does not start with AFW.

To ease integration with future versions of the frameworks, keep the framework components untouched and put new VIs into separate components.

10.4 Viterbi Decoder

The Viterbi decoder core is capable of handling convolutional decoding for the IEEE 802.11 standard as well as for the control channel of the LTE standard. This can be configured through the operation mode interface.

The IEEE 802.11 standard defines convolutional encoding as forward-error correction coding in all transmission fields. The corresponding receiver in the application framework uses a Viterbi decoder implementing the maximum likelihood sequence estimation (MLSE) algorithm based on softbit input. A rate ½ convolutional code with a constraint length of 7 and the code polynomials [133, 171] is used. It is terminated with a sequence of zeros to reset the encoder to the all zero state after encoding.

10.4.1 Design Considerations

For the application framework, the Viterbi implementation must enable a throughput of 180 Mbit/s at a clock rate of 250 MHz. Additionally, the latency of the block must be less than 2 µs to meet the SIFS timing. In the IEEE 802.11 standard, the code blocks of signal fields are very short compared to the large data field code blocks.

10.4.2 Operation Principle

The Viterbi decoder consists of the three modules: branch metric computation, path metric accumulation & survivor selection, and traceback handling for actual decoding, as shown in Figure 10-3.

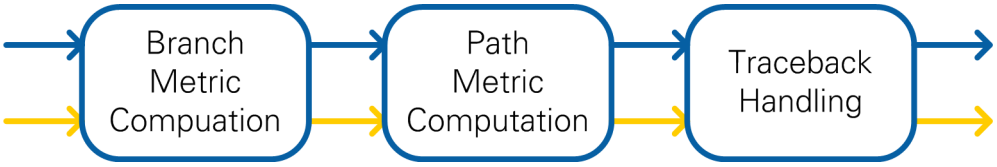

Figure 10-5: Viterbi Operation Principle

In the branch metric computation, the received softbits are multiplied by the hypotheses to form the state transition metric. This branch metric is used to update the path metric of all 64 states and calculate the surviving path. The corresponding Boolean bit value is stored in the traceback memory. After a certain amount of iterations—the traceback length—the maximum path metric is determined, and from its state the traceback memory is evaluated backward to decode bits in history along the most likely path in the Trellis.

The metric computation is running in a streaming mode and fills the traceback buffer continuously, but the actual decoding with evaluation of the traceback memory is initiated only every traceback length time instances. Hence, it is necessary to flush the metric computation with artificial softbits to enable traceback evaluation and decoding for the last bits of a code block, as well.

10.4.3 Viterbi Decoder Interface

The Viterbi core is capable of handling one bit per clock cycle. There is no handshaking implemented in the direction of upstream and downstream modules. The modules must be capable of handling continuous data streaming. The input valid and output valid signals are used to indicate valid samples.

The *data bit?* flag is aligned to the data. This Boolean is not used by the core but is delayed parallel to the processing. It can be used to distinguish data bits and flushing bits, which are required to decode the last bits of the code block.

The incoming data shall be given as Log-likelihood Ratios as defined in Equation 10-1. Based on the code rate, two code bit inputs (*code bit 0 and code bit 1*) must be used. The third code bit input (*code bit 2)* remains unused for rate 1/3 convolutional coding in the application framework. (However, it is used for LabVIEW Communications LTE

Application Framework.) The fixed-point format is FXP4.1. Based on the quotient a strong probability for a transmitted symbol $u_k = 0$ is mapped to 7.5. The strong probability towards a transmitted symbol $u_k = 1$ is mapped to -8. In case of puncturing, zero represents the maximum uncertainty.

$$LLR(y_k) \triangleq \ln\left(\frac{P(y_k \mid u_k = 0)}{P(y_k \mid u_k = 1)}\right)$$

Equation 10-1: Log-Likelihood Ratios

The output of decoded bits is given as a Boolean. The mapping from Booleans to symbols is done such that a false equals 1 and true equals -1 (see Equation 10-2).

$$x = 1 - 2u_k$$

Equation 10-2: BPSK Mapping

The operation mode must be set to constant `802.11` to work within the application framework. The traceback length defines the minimum number of states the Trellis is continued before decoding the current state. The valid range is 1 to 127. This value must be set to a constant as well.

10.4.4 Viterbi Decoder Implementation

The block diagram of the implementation is illustrated in Figure 10-4.

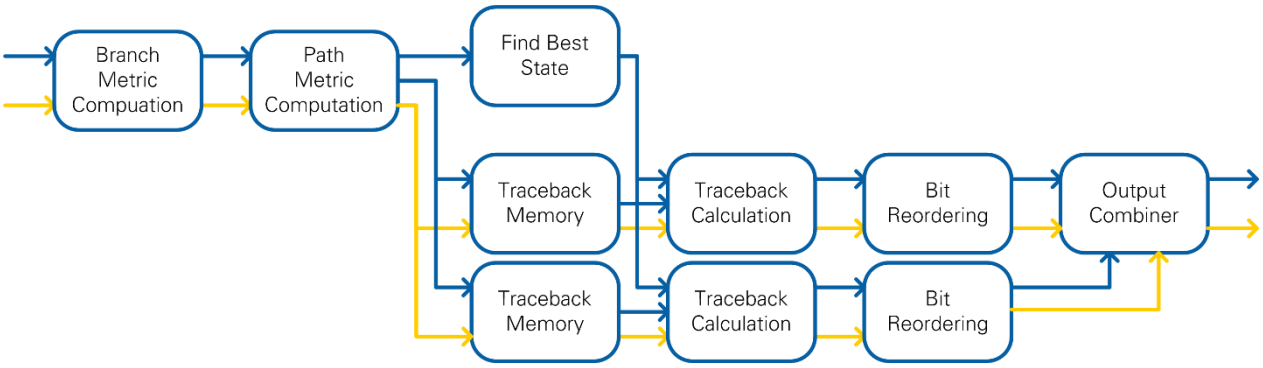

Figure 10-6: Viterbi Block Diagram

The branch metric computation is implemented with simple sign changes and additions. This submodule has an application framework-specific implementation corresponding to the code rate. For the application framework, only code bit 0 and 1 are used since the encoder has code rate 1/2. Code bit 2 is ignored. Hence, the output of the branch metric computation yields 4 valid values only. The subsequent modules are independent of the code rate.

In the path metric computation, often named *Add-Compare-Select* in literature, for every state from the 64 states, the path metric values of the two preceding states are updated with the corresponding branch metrics and the larger value of the resulting values is stored as the new path metric for this state. At the same time, the result of

the comparison is stored as a Boolean value to mark the more likely state transition of the surviving path. The outputs of the submodule are a new 64-element vector of path metrics and a 64-element Boolean vector of survivor paths for every bit vector input.

The operation mode selector is used to set the appropriate initial value of path metrics and the correct selection of the branch metric element for the state transition (one out of the valid four). Since the IEEE 802.11 standard defines Trellis termination with zeros, the start vector of path metrics gives a competitive edge for the zero state over all other states.

The path metric computation submodule does not have a reset. Thus, at the end of a code block, the path metric memory must be similar to the described start vector to allow continuity of code block handling. This is achieved by flushing appropriate softbits (see Section 10.4.2). For the application framework, Trellis termination softbits representing a high probability of transmitted zeros are flushed (also known as *strong zeros*). If the path metric vector does not already have the highest value for state 0, flushing will take care of it.

Normalization of the path metric values is used to avoid infinitively growing values and restrict the bitwidth. Since only the difference between path metrics is of interest and not their absolute value, normalization does not influence the decoding result. The process is stretched to two clock cycles. In the first clock cycle, all path metric values are checked versus a threshold before they are written to a memory. In the second clock cycle based on the threshold comparison, a constant value is subtracted from the branch metric prior to updating the path metrics.

The survivor path is written to two traceback memories. After traceback length samples, one of the two traceback paths is triggered. The most probable state at this point is the one with the largest path metric value. Its index is provided by the *Find Best State* module. Starting from this state, the *Traceback Calculation* module recursively calculates the previous state based on the survivor path vectors from the traceback memory. The decoded bit is derived from the LSB of this survivor state.

It is worth mentioning that the first decoded bits of the survivor path show lower reliability than later elements in the traceback. Thus, the first half of the bits is discarded. The order of the remaining decoded bits must be reversed because the traceback memory is evaluated backwards. Both operations are done in the *Bit Reordering* module.

In the last step, the outputs of the two traceback chains are combined to a final decoded sequence, which is available on the output.

10.4.5 Viterbi Decoder Timing

The timing of the Viterbi decoder is shown in Figure 10-5. It is independent of the chosen operation mode but it depends on the traceback length. As described in Section 10.4.4, there are two traceback chains which are illustrated in different colors. The

horizontal axis represents the time. A scale with multiples of traceback length clock cycles is visible on the top as a reference. The timing diagram assumes that there is a valid input in each clock cycle. The traceback memory is empty at the beginning.

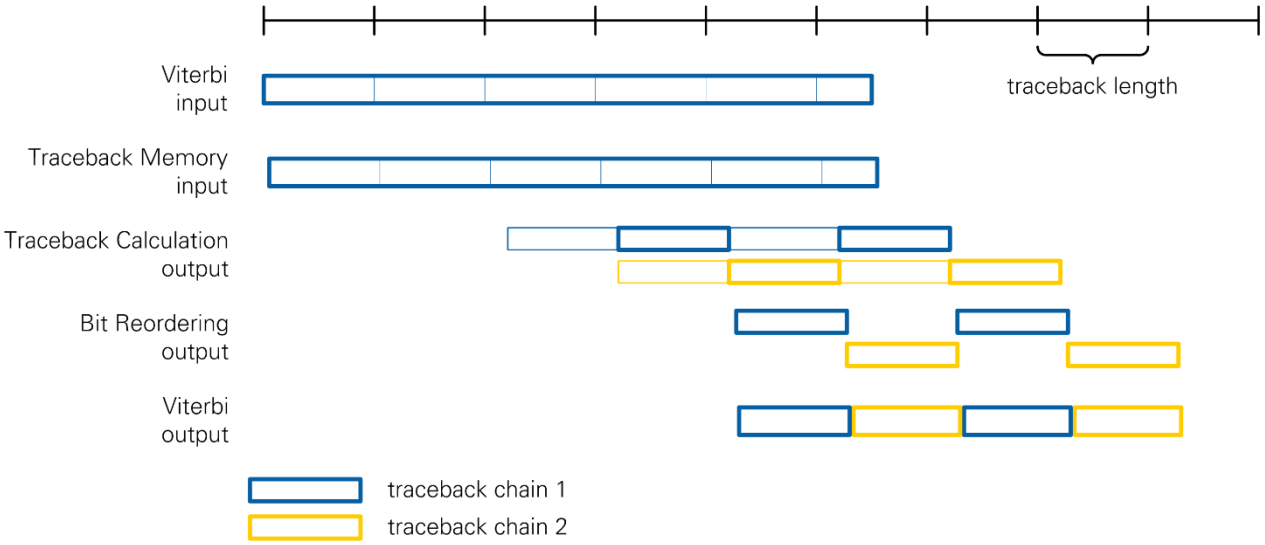

Figure 10-7: Viterbi Timing

All input data is processed in the branch and path metric calculation. This calculation adds two cycles of latency before storing the data to the traceback memory. The first traceback memory is read as soon as two times traceback samples are written. The second traceback chains starts another traceback samples delayed. The Traceback Calculation module just adds one cycle of latency. At the output of the Bit Reordering module, only the second half of the samples is declared as valid after two times traceback length elements have been written. The outputs of both traceback chains are combined to a continuous output stream.

If the input is not valid at each clock cycle, the input pattern is kept until the traceback memory input. Afterwards, the traceback decoding and bit reordering are performed burst wise. In this case, the latency of every wait cycle on the input will increase the latency for the first code block input by one cycle. NI recommends flushing the Viterbi core with a continuous stream to have the minimum latency for the end of the code block.

This concept results in a decoding latency of four times the traceback length (plus 13 clock cycles processing time). During two times the traceback length of the traceback buffer is written, and during another two traceback lengths the evaluation and decoding take place. The evaluation in chunks of two times the traceback length makes it necessary to flush the Viterbi decoder with the same number of input softbit triples. The latency of each module is summarized in Figure 10-6.

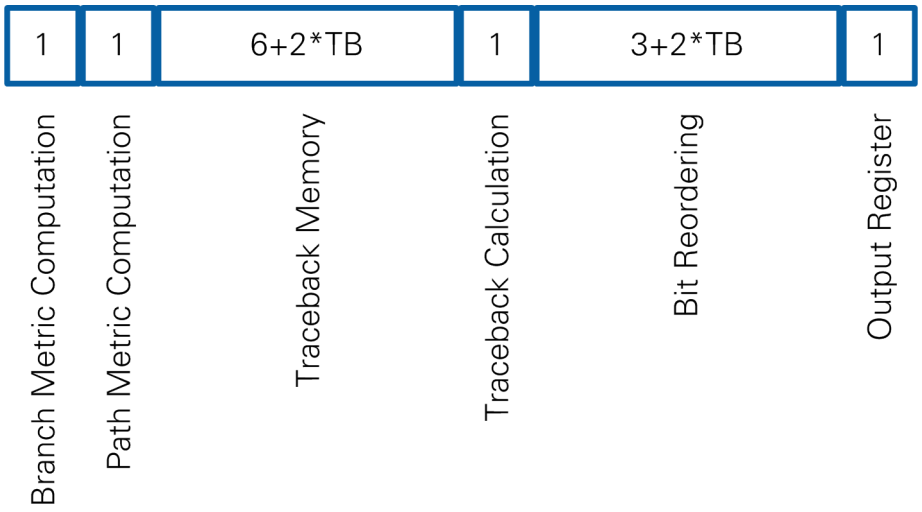

Figure 10-8: Viterbi Latency

10.4.6 Resource Usage

The Viterbi implementation occupies the FPGA resources listed in Table 10-1.

Operation mode	**802.11**
LUTs	4189
Registers	1990
Block Ram (36k)	2

Table 10-1: Viterbi Resource Usage

10.4.7 Viterbi Decoder Throughput

The throughput in MS/s is equal to the clock rate in MHz since the core is capable of handling one sample per each clock cycle. Synthesis of the core is successful up to a clock rate of 300 MHz.

11 Abbreviations

Abbreviation	Meaning
ACK	Acknowledgement
ADC	Analog digital converter
AGC	Automatic gain control
A-MPDU	Aggregated MPDU
BCC	Binary convolutionally encoded
BPSK	Binary phase-shift keying
BSSID	Basic service set identifier
CCA	Clear channel assessment
CP	Cyclic prefix
CS	Carrier sensing
CSMA/CA	Carrier sense multiple access with collision avoidance
CTS	Clear-to-send
CW	Continuous wave
DA	Destination address
DAC	Digital analog converter
DCF	Distributed coordination function
DDC	Digital downconversion

DIFS	Distributed (coordination function) interframe space
DUC	Digital upconversion
EDSC	Enable driven stream combiner
EIFS	Extended interframe space
EOF	End of file
EVM	Error vector magnitude
FAM	Frontend adapter module (RF module)
FCS	Frame check sequence
FFT	Fast Fourier transform
FSM	Finite state machine
FXP	Fixed point
GI	Guard interval
H2T	Host-to-target
HCF	Hybrid coordination function
I/Q	In-phase/quadrature
ICP	Interface communication protocol
IFS	Inter frame spacing
L-DATA	Legacy data
LFSR	Linear feedback shift register
LLR	Log-likelihood ratio
L-LTF	Legacy long training field
LRC	Long retransmission counter
L-SIG	Legacy signal field
LUT	Look-up table
MAC	Medium access control layer
MCF	Mesh coordination function
MCS	Modulation and coding scheme
MIMO	Multiple-input, multiple-output
MLSE	Maximum likelihood sequence estimation
MMPDU	MAC management protocol data unit
MPDU	MAC protocol data unit
MSB	Most significant bit
MSDU	MAC service data unit
NAV	Network allocation vector
NDP	Null data packet
FDM	Orthogonal frequency-division multiplexing
OTA	Over-the-air
PCF	Point coordination function
PHY	Physical layer
PIFS	Pointt (coordination function) interframe space
PLCP	PHY layer convergence protocol
PN	Pseudo noise
PPDU	PLCP protocol data unit
QAM	Quadrature amplitude modulation
RA	Receiver address
RC	Receiver cache
RTS	Request-to-send
RX	Receive
SA	Source address
SAP	Service access point
SDR	Software-defined radio
SIFS	Short interframe space
SISO	Single input, single output

SLRC	STA LRC
SNS	Sequence number space
SRC	Short retransmission counter
SSRC	STA SRC
STA	Station
T2H	Target-to-host
TA	Transmitter address
TX	Transmit
UDP	User datagram protocol
VHT	Very high throughput
VHT-LTF	VHT legacy long training field
VHT-SIG-A	VHT signal field A
VHT-SIG-B	VHT signal field B
VHT-STF	VHT signal training field

12 Bibliography

[1] IEEE, "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," 2016.

[2] IEEE, User guide for 802.11ac waveform generator, IEEE 802.11-11/0517r6, 2011.

[3] "LabVIEW Communications 802.11 Application Framework Getting Started Guide," National Instruments, 2018.

[4] T. M. Schmidl and D. C. Cox, "Robust Frequency and Timing Synchronization for OFDM," *IEEE Transactions on Communications,* vol. 45, no. 12, pp. 1613-1621, 1997.

377555C-01 Oct 2018