RW-WLAN-nX MAC HW STA 20MHz

Functional Specifications

RW-WLAN-nX-MAC-HW-STA20-FS/4.02

Version 4.02

2019-01-04

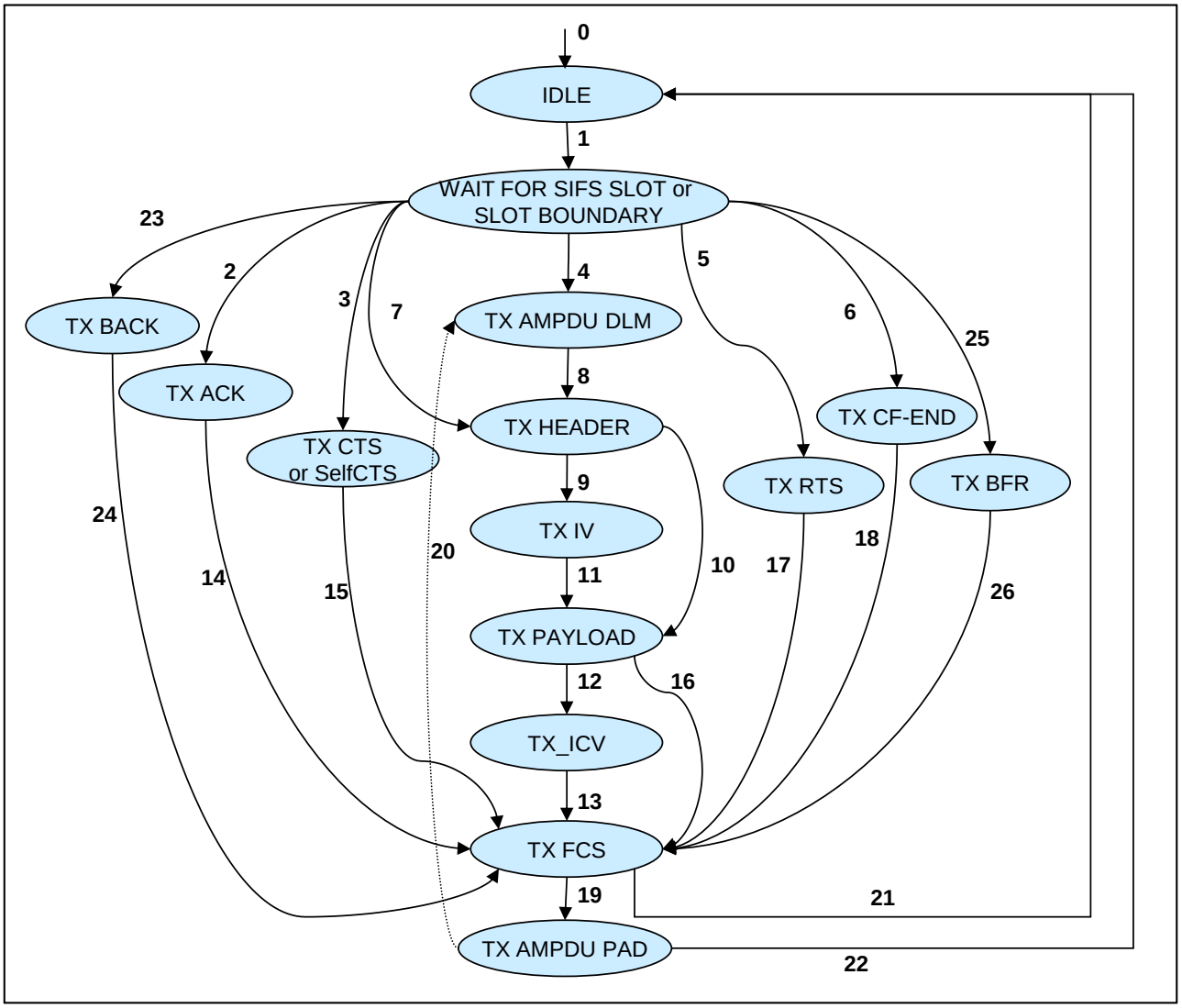
RivieraWaves
http://www.rivierawaves.com/

Revision History

Version	Date	Revision Description	Author
1.00	2011-06-07	Initial version	Jerome Vanthournout
1.01	2012-04-05	Updated RX FIFO with discard tag	Jerome Vanthournout
2.00	2012-11-21	Updated for 11ac	Jerome Vanthournout
2.01	2013-01-31	Updated with bandwidth management	Jerome Vanthournout
2.02	2014-11-01	Update with new clocking scheme and WAPI	Jerome Vanthournout
2.03	2015-02-05	Updated with WPI Engine	Romaric Blanc
2.04	2015-02-25	Updated with Beamforming Support	Jerome Vanthournout
3.00	2015-10-30	Added MU-MIMO TX Support (#5000)	Jerome Vanthournout
3.01	2016-05-15	Cleaning	Jerome Vanthournout
3.02	2016-09-10	Cleaning	Jerome Vanthournout
4.00	2017-09-10	Added Trigger Based Transmission (#) Updated THD & Policy Table for 11ax-20MHz Support (#) Removed MU-MIMO Tx support Removed support of start TXOP from register (#8085) Cleaning on RX FIFO tag Removed ATIM Wake-up support (#8090) Removed CFP support (#8093) Remove transmit RIFS provision (#8100) Remove quiet interval 2 (#8133) Remove provision for JIT (#8120)	Jerome Vanthournout
4.01	2018-05-16	Renamed file	Jerome Vanthournout
4.02	2018-06-28	Removed PS Bitmap RAM (#8858)	Jerome Vanthournout

Table of Contents

List of Figures



List of Tables

1 Overview

1.1 Document overview

1.1.1 Purpose

This document gives information on the high level design of RivieraWaves's RW-WLAN-nX MAC HW core along with timing diagrams and high level state machine descriptions. The intention of this document is to give sufficient information for hardware developers to come up with low level detailed design implementation and for customers to understand MAC HW implementation at a high level.

The hardware portion of the RW-WLAN-nX MAC (henceforth referred to as MAC-HW, or the MAC core) is implemented as a synthesizable core in Verilog.

1.1.2 Scope

The document does not address the internal signals and logical implementation of HW blocks.

1.1.3 Abbreviations and Acronyms

Acronym	Expansion
ACK	Acknowledgment
AC	Access Category
AIFS	Arbitration Inter frame Space
AP	Access Point
BFR	BeamForming Report
CBC-MAC	Cipher Block Chaining – Message Authentication Code
CF	Contention Free
CTS	Clear To Send
DCF	Distributed Coordination Function
DIFS	DCF Inter frame Space
DMA	Direct Memory Access
DTIM	Delivery Traffic Indication Message
EDCA	Enhanced Distributed Channel Access
HCCA	HCF Controlled Channel Access
HT	High Throughput
IV	Initialization Vector
L-SIG	Legacy (Non-HT) Signal Field
MAC	Medium Access Control
MIB	Management Information Base
MIC	Message Integrity Code
MIMO	Multiple Input Multiple Output
MPDU	MAC Protocol Data Unit

Acronym	Expansion
MSDU	MAC service data unit
NAV	Network Allocation Vector
OFB	Output Feedback
OS	Operating System
PC	Point Coordinator
PIFS	PCF Inter frame Space
PHY	Physical layer
PSMP	Power Save Multi-Poll
QoS	Quality of Service
RD	Reverse Direction
RIFS	Reduced Inter frame Space
RSNA	Robust Security Network Association
SIFS	Short Inter frame Space
STA	Station
STBC	Space-Time Block Coding
TBTT	Target Beacon Transmit Time
TID	Traffic Identifier
TU	Time Unit which is equal to 1024 µs
WAPI	WLAN Authentication and Privacy Infrastructure
WEP	Wired Equivalent Privacy
WLAN	Wireless LAN

1.2 Architecture overview

1.2.1 Block diagram

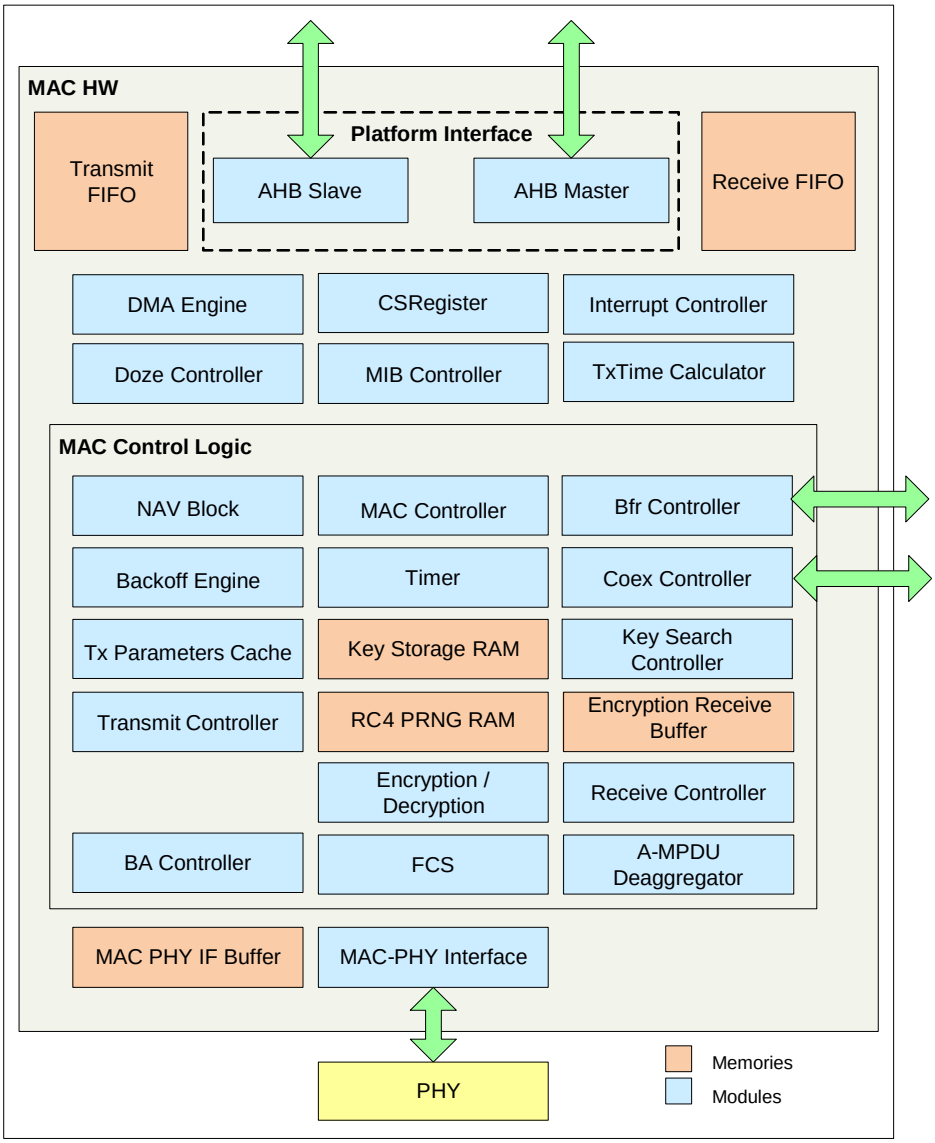


Figure 1: Block diagram of the MAC Core (SU-MIMO Configuration)

1.2.2 System description

The MAC HW block consists of the blocks as described below.

1.2.2.1 Platform Interface

1.2.2.1.1 AHB master

This provides a master interface on the AHB. The DMA Engine uses the master for DMA transfers from the system memory to/from the MAC FIFO.

1.2.2.1.2 AHB slave

This provides a slave interface to the MAC HW. The processor can access the registers and some memories present in the MAC hardware using the AHB slave.

1.2.2.2 Control & Status Register (CSRegister)

This block contains registers to enable software to configure and control the hardware. This block also contains status registers through which different hardware status is provided to software. This block provides window registers through which software can access various memories like the Key Storage RAM, etc.

1.2.2.3 Interrupt controller

This block is responsible for generation and multiplexing of interrupts to SW.

1.2.2.4 DMA engine

The DMA engine provides the DMA service and along with corresponding list processors, enables frame transmission and reception. The DMA engine is a linked-list, scatter gather, multi channel DMA engine.

1.2.2.5 Transmit FIFO

Each frame prepared by software (both MAC header and payload), when ready to be transmitted, is stored temporarily here before transmission. This helps in compensating for the latencies in reading the data from the system memory and handling the data speeds on the wireless medium.

It is implemented using a two-port HW RAM with one read port and one write port.

1.2.2.6 Receive FIFO

Each received frame (both MAC header and frame payload) is stored temporarily here before being moved into the system memory. This helps in compensating for the latencies in writing the data into system memory and handling the data speeds on the wireless medium.

It is implemented using a two-port HW RAM with one read port and one write port.

1.2.2.7 Doze Controller

This block is responsible for enabling or disabling clock gating for MAC HW. This ensures minimum power consumption.

1.2.2.8 MIB Controller

This block is responsible for storing in a Memory some statistics about the transmitted and received frames.

1.2.2.9 TXTime Calculator

In order to compute the duration field or the Legacy length in HT/VHT mode, in order to check if the next frame exchange fits in the TXOP or to calculate the exact TSF after a Beacon or Probe Response reception, it is required to calculate the frame duration based on its length, format (NON-HT, HT_MM, HT_GF or VHT), shortGI, bandwidth, …

This module, which is controlled by the MAC Controller, performs these computations and returns the frame duration in microsecond. It has also a register interface which allows the software to use this module for Time On Air computation.

1.2.2.10 MAC Control Logic

The MAC Control Logic block consists of the following blocks.

1.2.2.10.1 MAC Controller

The MAC Controller is the low level controller of the MAC HW. It controls the *Transmit Controller* and the *Receive Controller*.

1.2.2.10.2 NAV Block

This module maintains the state of the two NAV counters. One for intra-BSSS and one for inter-BSSS.

1.2.2.10.3 Timer

This logic is responsible for maintaining various core timing logic like TSF, beacon Interval Counters, Medium Occupancy Timer, TXOP Limit, IFS etc.

1.2.2.10.4 Backoff & AC Selection Engine

This block performs all the backoff related functions. It handles also the AC selection and internal collision if any.

1.2.2.10.5 TX Parameters Cache

The TX Parameters Cache is responsible for storing the Transmit Header Descriptor and the Policy Table fetched by the DMA Engine.

1.2.2.10.6 Transmit Controller

This block controls the transmit data path from the Transmit FIFO to the PHY Interface. When triggered from the MAC Controller block, it reads the frame pending in the Transmit FIFO and passes it through the encryption block (if required) and the FCS block. It controls the encryption and FCS blocks.

It creates and transmits the following frames when signaled by the MAC Controller block: ACK/CTS/RTS/B-ACK/CF-End.

1.2.2.10.7 FCS block

In the transmit path, the data bytes are passed through the FCS block on the fly as they being sent to the PHY Interface. At the end of the payload, the FCS generated by the FCS block is extracted and appended.

All received frames are also passed through the FCS block for FCS checking as the bytes are received from the A-MPDU De-aggregator block and written into the Receive FIFO. At the end of the reception, the received FCS is compared with the calculated FCS.

1.2.2.10.8 A-MPDU De-aggregator

This block handles all frames received from the PHY Interface. It de-aggregates the received A-MPDU and passes the individual MPDUs to the Receive Controller.

1.2.2.10.9 Receive Controller

The Receive Controller block handles all frames received from the *A-MPDU De-aggregator* block. It controls the receive data path between the PHY Interface and the Receive FIFO.

1.2.2.10.10 BA Controller

This block updates the BA Partial State Bitmap when frames are received in order to generate the BA frame. This block passes Block ACK bitmap to Transmit Controller to transmit Block ACK Response frames.

1.2.2.10.11 Encryption/ Decryption Engine

The encryption logic consists of four parallel inline engines which are integrated into the data path: WEP, TKIP and CCMP. A fourth encryption engine (WPI based on SMS4) is also available for WAPI support.

1.2.2.11 MAC-PHY Interface

The MAC-PHY interface is responsible for creating the TX Vector, receiving the Rx Vector and handshaking with the PHY as described in *[2]*.

1.2.2.12 Coex Controller

The Coex Controller is responsible of the Bluetooth Coexistence by providing information to an external BT/WLAN arbiter and also to abort the on-going transmission or reception in case the BT won the priority.

1.2.2.13 BFR Controller

The BFR Controller (i.e. Beamforming Controller) is in charge of managing the 802.11ac Beamforming procedure as a Beamformee.

1.3 Clocking strategy

The clock domains supplied to the MAC HW are as explained in *[1]*. The various functional parts of the MAC HW run on different clock domains as described in this section:

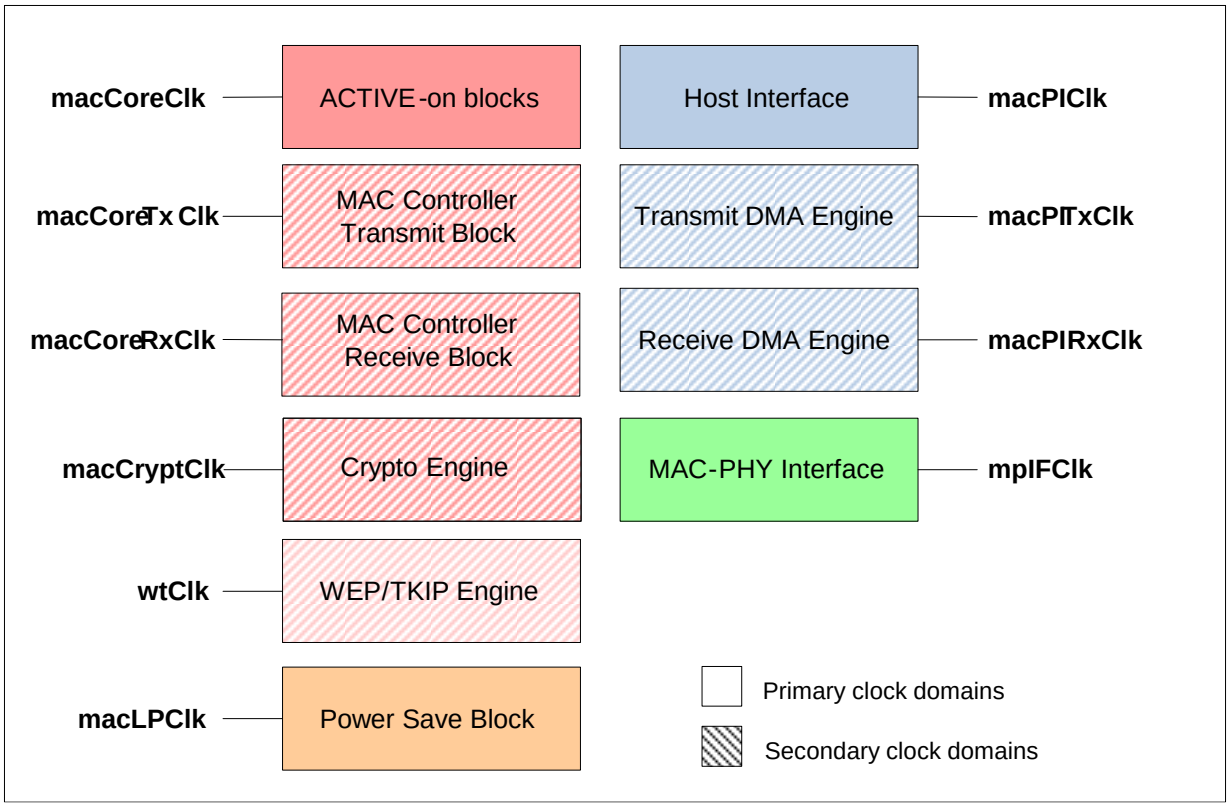


Figure 2: Clock inputs to MAC HW modules

The MAC HW implements extensive clock gating schemes for DOZE state power saving as well as ACTIVE state power saving.

1.3.1 DOZE state power save support

The following procedure is followed by the MAC HW when it is programmed into the DOZE state. That covers programmed transitions controlled by HW and SW.:

1. It first turns off the five secondary clock domains (*macCoreTxClk*, *macCoreRxClk*, *macPITxClk* and *macPIRxClk, macCryptClk* and *wtClk*) using the individual level sensitive enable signals.
2. Next the level sensitive *switchMACLPClk* signal is asserted to switch the *macLPClk* from *macCoreClk* frequency to 32 KHz.
3. Finally it de-asserts the macPriClkEn signal to the platform clock controller to turn off the three primary clock domains: *macPIClk*, *macCoreClk*, and *mpIFClk*.

The *macLPClk* feeds some minimum logic blocks that should continue running: synchronization and wake up block. The HW can wake up by itself or under explicit control from SW. In both conditions, it follows the following procedure:

1. It first deasserts the *switchMACLPClk* signal to turn the 32 KHz clock back to the *macCoreClk* frequency.
2. Next it asserts the macPriClkEn signal to the platform clock controller to turn on the three primary clock domains: *macPIClk*, *macCoreClk* and *mpIFClk*.
3. It turns on the five secondary clock domains (*macCoretxClk*, *macCoreRxClk*, *macPITxClk*, *macPIRxClk, macCryptClk* and *wtClk*) using the individual level sensitive enable signals as required.

1.3.2 ACTIVE state power save support

In the ACTIVE state, the MAC HW can turn off portions of the logic aggressively (i.e. whenever possible) to minimize active state power consumption for battery powered devices.

The primary clock domains are not turned off in ACTIVE state. Hence the blocks which are fed by the secondary clocks are selectively turned off in the ACTIVE state as explained below:

1. The *macCoreClk* feeds the CSR block and other blocks like NAV, MAC Controller and Backoff which can't be turned off except in DOZE state.
2. The *macPIClk* feeds the Platform Interface blocks like the AHB Master and the AHB Slave and these are not turned off except in DOZE state.
3. The *mpIFClk* feeds the MAC-PHY Interface block and this is turned off in IDLE and DOZE state.
4. The *macLPClk* feed a few modules like Doze controller and Timer which are active in the DOZE state; hence they cannot be turned off in the ACTIVE state.
5. The *macPITxClk* feeds the Transmit DMA Engine and hence this is turned off whenever possible: for instance, when it is in the HALTED or PASSIVE state..
6. The *macPIRxClk* feeds the Receive DMA Engine and hence this is turned off whenever possible: for instance, when it is in the HALTED or PASSIVE state or it is waiting for data to be available in the Receive FIFO, etc.
7. The *macCoreTxClk* feeds specific transmission-only blocks like the Transmit Controller, which can be turned off when the MAC is receiving a frame from air.
8. The *macCoreRxClk* feeds specific reception-only blocks like the Receive Controller, A-MPDU De-aggregator block, which can be turned off when the MAC is transmitting a frame on air.
9. The *wtClk* feeds the WEP/TKIP encryption/decryption block, which is turned off when it is not in use.
10. The *macCryptClk* feeds the encryption/decryption block, which is turned off when it is not in use.

HW Embedded RAMs are enabled only when required.

1.4 Reset strategy

There are two kinds of resets to the system.

1. **Hardware Reset**: This reset is an external asynchronous hard reset. The same hard reset is synchronized to the various clock domains by the platform reset module and supplied to the MAC as primary inputs. Assertion of this signal brings all the state machines to their initial condition.
2. **Software Reset**: This reset is a synchronous reset issued by the software. The software sets the "softReset" bit of the csReg. When the synchronous reset is performed in all the clock domains, the "softReset" bit is updated with "0" by hardware to indicate to the software that reset has been performed. This reset brings all the state machines to idle state, and resets all internal flags and counters.

2 Data flow

This section details the Transmit and Receive Data flow in MAC HW.

2.1 Transmit Data flow

Transmission of frames happens when the MAC Core is in the ACTIVE state. Transmission of a frame begins with the MAC Core trying to gain access to the medium. The medium contention procedure defined by the protocol to gain access to the medium is implemented by MAC Core. The virtual carrier sense from the NAV block and the physical carrier sense (CCA) from PHY together indicate to the MAC Core if the medium is busy or idle. The backoff procedure begins when the backoff block in the MAC Core finds the medium idle. The backoff block implements four backoff engines for the four access categories. The AIFSn and the slot interval timings are maintained by the Timer block.

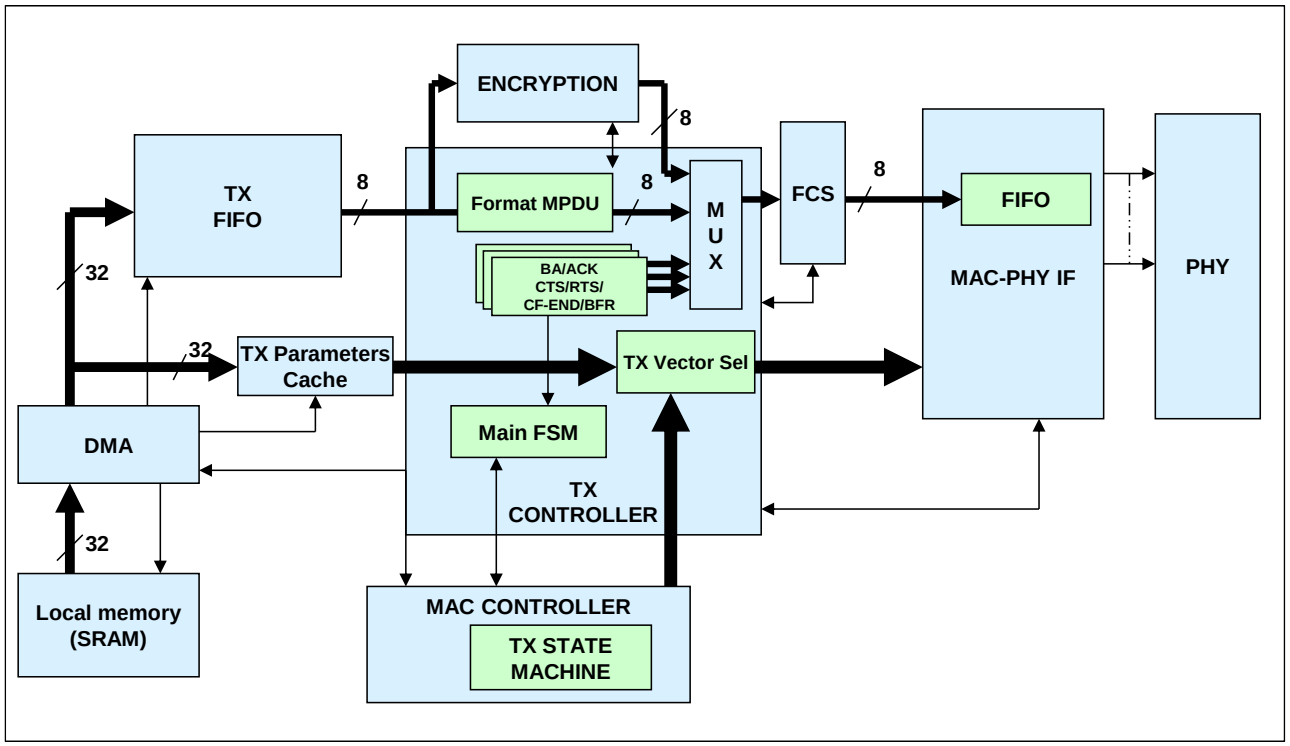


Figure 3: Block Level TX Flow Diagram

2.1.1 TXOP acquisition

The SW can indicate through register settings or through the Transmit Header Descriptor if the frame has to be transmitted in 20MHz, 40MHz, 80 MHz or 160MHz mode. 20MHz TXOP is considered to be obtained when the contention to the primary channel has been won. 40MHz TXOP is considered to be obtained when contention to the primary channel has been won and the secondary 20MHz channel (*CCASecondary20*) has been idle for PIFS duration. Same behavior with 80MHZ TXOP with *CCASecondary20 and CCASecondary40.* When *backoffDone_p* signal from the backoff block occurs the *sec20ChannelIdleFlag*flag should be asserted high to obtain a 40MHz TXOP. (*sec20ChannelIdleFlag* and *sec40ChannelIdleFlag* for a 80MHz TXOP; *sec20ChannelIdleFlag*, *sec40ChannelIdleFlag* and *sec80ChannelIdleFlag* for a 160MHz TXOP)

When trying to obtain a TXOP the MAC Controller checks the *txBWCntrlReg.defaultBWTXOP* and *txBWCntrlReg.defaultBWTXOPV* fields to see if a 40/80/160 MHz TXOP has to be obtained. If the indication is present, the MAC Core attempts to obtain a 40/80/160MHz TXOP irrespective of whether the queued data is to be transmitted. In case of a successful attempt the queued frames are transmitted. In case of an unsuccessful attempt, the decision to obtain a 40/80/160MHz TXOP next time would depend on the value in the *txBWCntrlReg.numTry40Acquisition* register. If the value is zero the MAC Core continually tries to obtain a 40/80/160MHz TXOP only. If the value is greater than zero

the MAC Core attempts that many times to obtain a 40/80/160MHz TXOP. If the MAC Core is unable to obtain a 40/80/160MHz TXOP within that many attempts, it obtains a lower bandwidth TXOP and transmits the frame. The MAC Core always does a TXOP check to see if the frame to be transmitted together with its required protection frames and acknowledgements fit within the remaining TXOP. If it doesn't fit, the MAC Core releases the TXOP and tries to the send the frame in the next TXOP. In case of continued failure to transmit successfully, the frame finally gets discarded due to expiry of lifetime.

If the *txBWCntrlReg* register setting is invalid then MAC Core checks if there is data queued and if the SW has indicated through the descriptor fields to transmit the frame in 20/40/80/160MHz mode. If the frame to be transmitted is in 40/80/160MHzmode then MAC Core attempts to obtain a 40/80/160MHz TXOP. The behavior of the MAC Core in case of a failure to obtain the 40/80/160MHz TXOP is governed by the *txBWCntrlReg.numTry40Acquisition* field as explained in the above paragraph. The SW can use the Start Transmission Registers to indicate to the MAC Core whether to obtain a 40/80/160MHz or 20MHZ TXOP when data has not yet been queued to the relevant DMA channel.

If the MAC Core acquires a TXOP and subsequently encounters a frame with a bandwidth higher than the TXOP then it checks the *txBWCntrlReg.dropToLowerBW* value to check if this frame can be transmitted in a lower mode. If the setting indicates that the frame exchange sequence can be transmitted in the lower bandwidth mode and the duration on air does not exceed value *txBWCntrlReg.aPPDUMaxTime* then the transmission continues else the TXOP is released. If this bit is not set and a 40/80/160MHz TXOP cannot be acquired, the TXOP is released.

In case of bandwidth signaling, the MAC Core transmits the RTS frame with a bandwidth signaling TA and indicates the type of bandwidth selection (static or dynamic). Upon CTS reception, the MAC Core checks if the requested bandwidth is available or not. If yes, then the TXOP is acquired with the requested bandwidth. If this is not the case, either the TXOP is not acquired and the frame will be retried later (in static mode) or the TXOP is acquired with a lower bandwidth (in dynamic mode).

2.1.2 Tx Frame descriptor preparation

When the MAC Core contends for the channel, the SW is indicated by the Protocol Trigger interrupt to keep the data ready for transmission. This interrupt is given when the MAC wins the medium. The Protocol Trigger is only given once before obtaining the TXOP and is given per Access Category. Subsequently to ask the SW to queue frames during the TXOP the Transmission Trigger is provided. On receiving the Protocol Trigger or a Transmission Trigger for a particular Access Category, the SW starts queuing frames for this Access Category if it has data to be sent from that Access Category. It is possible that the MAC Core will generate Protocol Triggers from two Access Categories simultaneously. Similarly it is also possible that the MAC Core will give Protocol Trigger from a different Access Category subsequently after it gives for a particular Access Category.

SW queues the frames and prepares descriptors. There are two types of descriptors: Header descriptor and Payload Descriptor. Each frame will have a single Transmit Header Descriptor and zero, one or more Payload Descriptors. Apart from the location of the buffers the Transmit Header Descriptor will also contain information to form the TX-Vector, certain MAC Core specific control information and an index to the location of the Policy Table (which also contains information necessary to form the TX-Vector). The structure of the DMA descriptor is detailed in *[1]*.

2.1.3 DMA transfer

The movement of data from the local memory to the TX FIFO is done by an embedded DMA Engine. The DMA engine is a scatter-gather DMA and with the help of descriptors moves data from the Local Memory to the TX FIFO.

The DMA is triggered by the MAC Controller in the last backoff slot when contending for the medium. The trigger for the DMA is per channel. On receiving the trigger (*trigTxAC0*, *trigTxAC1*, *trigTxAC2*, *trigTxAC3* or *trigTxBcn*, the DMA engine first fetches the Transmit Header Descriptor and forwards the control information from the Transmit Header Descriptor to the TX Parameters Cache in the MAC Controller.

At the same time it compares to check if the frame queued has expired its *frameLifetime*.

If the *frameLifetime* is expired then the DMA engine updates the status of this MPDU or A-MPDU in its Transmit Header Descriptor and starts operation on the next frame pointed by the *Next Atomic Frame Exchange Sequence Pointer*. If this pointer is not valid the DMA moves to HALTED.

If the *frameLifetime* is not expired, then the Tx process continues.

The Transmit Header Descriptor holds the Policy Table Entry address. After fetching the Transmit Header Descriptor the DMA engine fetches the Policy Table values and stores them in the TX Parameters Cache. Fetching of the Policy Table is not done for every MPDU. Populating the TX Parameters Cache with the values from the Transmit Header Descriptor and the Policy Table Entry is done before the MAC-PHY IF starts accessing these details to create the TX Vector. After these steps are completed the DMA engine starts fetching the frame to be transmitted from the local memory and stores them in the TX FIFO. The frame is present in the local memory according to the Transmit MPDU Template detailed in *[1].* The DMA engine is configured to move data in chunks to avoid repeated accesses to the memory. DMA engine completes this process when it has transferred the entire frame to the TX FIFO successfully. In case DMA engine encounters an error during these operations it raises an interrupt to the SW, stops this transfer and flushes the FIFO. It has to be reprogrammed by SW to do further operations. The DMA engine continues fetching the next MPDU as long as the number of frames in the TX FIFO is less than 2.

2.1.4 Transmission trigger

On completion of the backoff, the MAC Controller checks if the TX FIFO has data in it. It is possible that the SW did not queue any data for this Access Category and hence the DMA has not moved any data to the TX FIFO and not populated the TX Parameters Cache. In this case the MAC Controller restarts the backoff procedure for the other Access Categories.

The other possibility is that the MAC Controller finds that SW has queued the frames for transmission and the DMA has populated the TX Parameters Cache. The MAC Controller checks the TX Parameters Cache to check if the SW has requested transmission of protection frames. The SW can indicate this through the Transmit Header Descriptor.

If protection frame transmission is requested, the MAC Controller block triggers the TX Controller block to prepare and transmit the protection frames.

Otherwise the MAC Controller triggers the TX Controller to transmit the queued frame.

If a transmission error or a transmission abort is received (Underrun reported by the PHY or DMA error), the MAC Controller issues a stop indication to the TX Controller to stop the transmission.

2.1.5 Verification done before Tx

A series of checks are done by the MAC Core before transmitting the frame queued by the SW. Frame lifetime check is done by the DMA before moving the MPDU from the local memory to the TX FIFO. If the frame lifetime has expired, the DMA engine updates the status in the Transmit Header Descriptor for this frame and starts fetching the next frame pointed by the Atomic Frame Exchange Sequence Pointer.

Before transmitting the frame the MAC Controller checks if the frame can fit within the remaining TXOP. Only if it fits will the TX Controller be triggered to start its transmission operations. If it does not fit, the MAC Controller evaluates if it can transmit one or more CF-End frames as explained in *[1]* and in that case, will trigger the TX Controller to transmit the required CF-End frame(s).

As the TX Controller creates the MAC Header of the frame being transmitted, it will update certain fields described section *12.6.2.4*, *MAC Header generation*. The SW can prevent the MAC Core from doing this. This option is provided by setting various bits in the MAC Control Information 2 fields in the Transmit Header Descriptor fields. The TX Controller will take these settings into account when updating the fields in the MAC Header. The NAV block maintains the NAV timer which is updated with the NAV coverage if the protection frame transmission is successful. This information helps the MAC Controller in deciding to bypass the transmission of protection frames created and queued for transmission by SW or protection frame transmission requested through registers or DMA descriptors if NAV coverage already exists.

2.1.6 PHY Start procedure

Once the TX Controller receives a trigger for transmission, it waits for the SIFS/SLOT boundary. The SIFS/SLOT boundary indication is given by the Timer block. At SIFS/SLOT boundary it indicates to the MAC-PHY IF to assert the *txReq* and start its operations. The TX-Vector needs to be put out when the *txReq* is asserted. This is taken care by the MAC-PHY IF block. After the TX-Vector is given to the PHY the frame stored in the MAC-PHY IF FIFO is sent to the PHY. The MAC-PHY IF ensures that the TX-Vector is prepared and written in the MAC-PHY IF FIFO ready to be sent to PHY at the assertion of the *txReq*. Once the MAC-PHY IF prepares the TX Vector and has written it to the FIFO it asserts a signal *txVectorDone_p* to the TX Controller. The information to prepare the TX-Vector is obtained from the TX Parameters Cache. The TX Controller controls the movement of frame to be transmitted from the TX FIFO to the MAC-PHY IF FIFO.

2.1.7 TX frames processing

Once it is triggered, the TX Controller checks what kind of frame has to be transmitted.

If the TX Controller is triggered for a HW formed frame like RTS or self-CTS or CF-End and not in response to any frame it waits for the SIFS/SLOT boundary. At the SIFS/SLOT boundary it triggers a key search to get the destination address of the frame using the *RAIndex* field. Once the key search is complete the rest of the fields of the frame like the Duration and the MAC Header field are formed in the TX Controller and passed through the FCS block and written to the MAC-PHY IF FIFO.

If the TX Controller is triggered to transmit a SW formed frame which is part of an A-MPDU, the TX Controller first forms the delimiter and writes it to the MAC-PHY IF FIFO. Then the TX Controller waits for the SIFS/SLOT boundary.

In case the frame is not an A-MPDU then the TX Controller waits for the SIFS/SLOT boundary.

The TX Parameters Cache gives details on the type of frame. When trigged, TX Controller asserts read signal to TX FIFO to start fetching the frame. The TX Controller reads and decodes the TAG bits from the TAG FIFO to identify if the bytes being read are part of header of the frame or payload or which byte is valid in the quadlet which is read. When the first byte of the header is fetched from the TX FIFO, the FCS block is triggered and the first byte is passed through the encryption block and to the FCS block.

Then, the TX Controller fetches the second byte from the TX FIFO. These two first bytes are used by the Tx Controller in order to build the MAC header and update the fields accordingly. Based on the *Protected Frame* bit in the Frame Control field of the frame header and the *dontEncrypt* field in the Transmit Header Descriptor, the TX Controller trigs the Key Search Engine and get back the Encryption information (type of encryption/Key/...) from the KeyStorageRAM.

When the key search is completed, the type of encryption and the other information necessary for this encryption is got from the KeyStorageRAM. TX Controller triggers the respective encryption block based on encryption type got from the KeyStorageRAM.

After triggering Encryption engine, TX Controller waits till initialization of Encryption engine is completed in case of WEP or TKIP. In parallel, it processes the remaining part of the MAC Header.

1. Fetch the remaining part of the MAC header for the TX-FIFO
2. Generate / Update / Discard some of the MAC header fields
3. Pass the required MAC header fields through the FCS block
4. Push into the MAC-PHY IF FIFO the MAC header
5. In case of CCMP encryption, the MAC Header is sent to the encryption Engine for AAD computation

Once the encryption engine initialization is completed, the TX Controller starts reading from the TX FIFO the data, passes the data through the encryption engine. Even if the frame does not need to be encrypted, the payload data are passed through the encryption engine which does not change them.

Then, the output of the encryption engine is passed through the FCS block for FCS calculation. The output of the FCS block is written to the MAC-PHY IF FIFO.

After the frame payload is written, the MIC (in case of CCMP and TKIP) or ICV (for WEP and TKIP) from the encryption engine is written to the MAC-PHY IF FIFO after being passed through the FCS block.

At the end of the transmission, the TX Controller indicates to the FCS block to shift out the computed FCS to the MAC-PHY IF FIFO.

After writing the last byte of the FCS to the MAC-PHY IF FIFO the TX Controller indicates this to the MAC Controller block and moves to IDLE state.

In parallel with all this processing and at the SIFS/SLOT boundary, the TX Controller trigs the MAC-PHY to assert the *txReq* and to launch the transmission to the PHY.

2.1.8 Transmission finalization

The MAC Controller depending on the frame transmitted will take the following decisions:

If the frame transmitted was part of an A-MPDU frame and the next frame to be transmitted is also part of the same A-MPDU frame then it triggers the TX Controller to start its transmission operations again.

The SW always queues a BAR frame at the end of the A-MPDU. This is to be transmitted if the Compressed Block ACK frame is not received after an A-MPDU transmission. The MAC Controller waits after the transmission of an A-MPDU frame for the compressed Block ACK frame depending on the ACK policy. If the frame is the last frame of an A-MPDU then the MAC Controller triggers the DMA to update the status of the A-MPDU Transmit Header Descriptor to A-MPDU success. *[1]* explains the different fields to be updated for an A-MPDU success.

- If it is received, then the explicit BAR frame is discarded. The MAC Core does not parse the compressed Block ACK frame it received and does not perform retransmission of the MPDUs carried in the A-MPDU. The SW will parse this compressed Block Ack to find which MPDUs need to be retransmitted and will reprogram a retransmission if required.
- If the compressed Block ACK frame is not received, the BAR frame will be.

After a transmission of a singleton MPDU (data, management, control frames) prepared by the SW, the MAC Controller checks if transmitted frame requires a response frame. If response is required then the MAC Controller block triggers the RX Controller to receive the response and waits for the indication from the RX Controller that the required response frame type is received successfully.

- If the response frame is received successfully then MAC Controller triggers the DMA to update the status field of the Transmit Header Descriptor to indicate MPDU success/RTS success.
- If the response frame is not received successfully the MAC Controller triggers the DMA to update the status field of the Transmit Header Descriptor to indicate either a Partial Failure or Retry Limit Reached status based on the number of times the frame has been retried. Partial Failure for different frame types is defined in *[1]*. If a protection frame does not received response, the TXOP is released. If the singleton MPDU which is not a protection frame did not receive its response and if the NAV coverage exists, it is retried in the same TXOP else it is retried in the next TXOP if it has not reached its Retry Limit. In case the frame has to be retried in the same TXOP, the MAC Controller triggers the DMA again. The DMA on receiving this trigger stops the current operation, flushes the TX FIFO and fetches the frame pointed by the status pointer for that channel. If the retry limit is reached, the DMA Engine will update the status and jumps to the next atomic frame exchange if any.

For HW formed control frames requested for transmission through registers there is no Transmit Status update. If the transmission fails, the MAC HW will try to transmit again in the next TXOP.

When the transmission of a protection frame requested through the descriptors is unsuccessful the descriptor is updated with the Partial Failure or Retry Limit Reached status depending on the number of retries.

The MAC Controller after transmission of frame (includes checking for a response) will compute based on the frame length and the transmission speed if the next frame exchange sequence can fit within the remaining TXOP. If it can

then it will transmit the frame within SIFS duration. If it cannot transmit the frame within remaining TXOP the MAC Controller evaluates if it can transmit the required CF-End frame(s) as explained in *[1]*.

2.2 Receive Data Flow

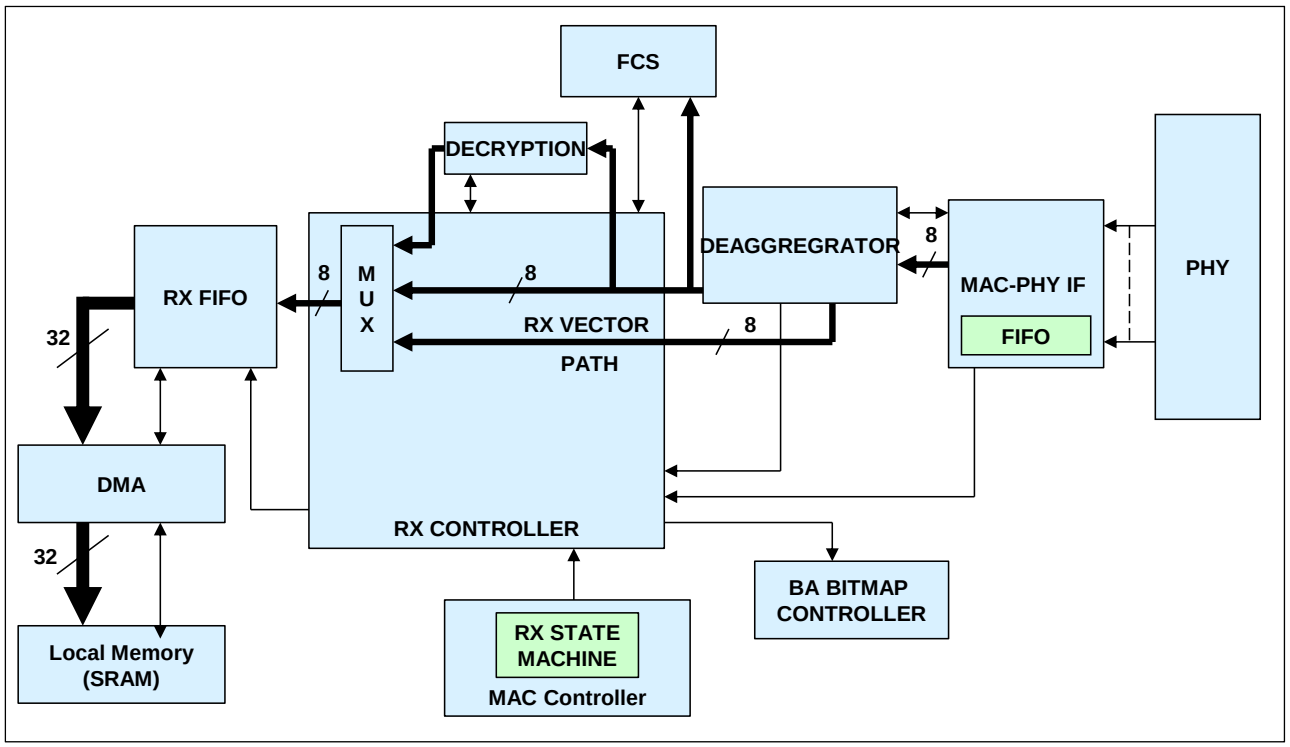


Figure 4: Block Level RX Flow Diagram

The movement of data during reception from the MAC-PHY IF FIFO to the RX FIFO and then to the SRAM along with the MAC Core functions is detailed in this section.

2.2.1 MAC PHY interface

The MAC Core is in the receive mode by default. Only when there is data to transmit then the MAC Core is put into transmit mode. In the receive mode, *rxReq* is asserted to the PHY to start reception.

When a frame is received the PHY fills the MAC-PHY FIFO first with part of the RX Vector followed by the received frame. At the end of reception the PHY writes the remaining RX Vector. The PHY gives two signals to indicate end of frame. One of them is *rxEnd_p* and the other is *rxEndForTiming_p*. The *rxEndForTiming_p* is given when the last data sample has been received from RF. This signal is primary used to generate SIFS but it is also used to sample TSF value. The *rxEnd_p* is given when the PHY has performed all the processing meaning after the end of RX Vector 2 transferred by the PHY to the MAC.

The MAC-PHY IF triggers the A-MPDU De-aggregator block once the RX Vector is written to the MAC-PHY IF FIFO.

2.2.2 A-MPDU De-aggregator

The A-MPDU De-aggregator block reads MAC-PHY FIFO and parses the RX Vector to find if the received data is an A-MPDU or singleton MPDU and then stores the RX Vector in internal registers.

If the received MPDU is a singleton MPDU then the de-aggregator block triggers the RX Controller and forwards RX data to the RX Controller when RX Controller is ready to accept data.

If the received MPDU is an A-MPDU then the de-aggregator block checks for the A-MPDU Delimiter and after the CRC passes, the RX Controller is triggered. The RX Controller uses the Rx Vector parameters stored in registers. However, the frame length indicated to the RX Controller is the individual length of an MPDU within the A-MPDU, obtained from an A-MPDU delimiter.

This procedure is followed for each MPDU within the A-MPDU. Once the RX Controller is triggered the data path is the same for both the singleton MPDU and the constituent MPDUs within an A-MPDU.

The entire length of the A-MPDU is known only to the de-aggregator block. The de-aggregator block ensures that it handles the entire A-MPDU based on the length specified in the RX Vector.

In case of A-MPDU with blank delimiters or incorrect delimiter (bad CRC), the De-aggregator automatically filters these delimiters.

2.2.3 RX controller

On receiving the trigger from the A-MPDU De-aggregator block the RX Controller traverses through its states to transfer the frame from the MAC-PHY IF FIFO to RX FIFO. The frame is passed through the FCS block for the FCS check. The RX Controller parses the MAC Header and checks the *Protected Frame* field and decides if the received frame is encrypted and triggers the decryption engine and provides the IV, Extended IV and other required fields. The decrypted frame is then written to the RX FIFO. If the FCS does not pass for the received frame then the FCS is written to the RX FIFO and the discard pattern is written to the RX TAG FIFO. No further information related to this frame is written to the RX FIFO. If the FCS passes then the RX Vector followed by the TSF and the MPDU Status Information are written to the RX FIFO. As soon as the FCS passes, the RX Controller also triggers the Block ACK Controller to update the bitmap irrespective of whether the ACK policy is set to Block ACK or Normal ACK. The RX Controller while receiving the frame triggers the NAV block to update the NAV counter or reset the NAV according to the type of frame received.

The RX Controller also writes the TAG bits in the RX TAG FIFO. The TAG bits carry information about the data written to the RX FIFO, and indicate MAC Header, frame payload, TSF, RX Vector and MPDU Status. Also the TAG bits indicate whether a frame has to be discarded or saved (passed to the SW). All the received frames are written to the RX FIFO and if the frame has not to be passed to SW, the TAG bits are written to the RX FIFO and coded with the discard pattern.

2.2.3.1 RX frame filtering

The HW discards the following received frames by default:

- Frames which are received with some error like FCS, PHY, Protocol etc
- All unicast frames that are destined for other stations. Only those frames with the destination address as the local MAC address (or range) are delivered to the MAC SW.
- All multicast addressed frames.
- All broadcast frames that do not contain the BSSID of this BSS.
- RTS, CTS, ACK and CF-End Control frames.
- CF-ACK, CF-Poll, CF-ACK + CF-Poll, QoS CF-Poll, QoS CF-ACK + CF-Poll frames.
- Expected BA

The MAC HW can be enabled to accept only certain kinds of frames through register setting. The frame filtering rules are explained in *[1]*.

2.2.4 BA Bitmap Controller

In case of QoS frame under BA Agreement, BA Bitmap Controller is trigged by the RX Controller and it updates its local memory based on the received sequence number, the keyIndex reported by the Key Search Engine, the TID and the status of the reception (correct or incorrect). It automatically moves the current bitmap window.

2.2.5 MAC controller

After the reception of the frame the RX Controller transfers the control to the MAC Controller. If a response is necessary the MAC Controller will then trigger the TX Controller to transmit the response frame (ACK / CTS / Block ACK).

2.2.6 DMA engine

The DMA engine starts movement of the frame from the RX FIFO into local memory once the RX FIFO has reached a watermark level. The DMA engine uses the RX DMA descriptors to move the frame between RX FIFO and local memory.

There are two types of RX DMA descriptors, one for MAC Headers and another for MPDU Payloads. The DMA engine uses one Receive Header Descriptor for the MAC Header and zero, one or more Receive Buffer Descriptors for each MPDU payload.

The MPDU payload can span multiple buffers, but multiple frames cannot reside in the same buffer. Frames which have zero payload (like PS-Poll, or QoS Null) will not use a Buffer Descriptor and will only have a Header Descriptor. The structure of the Receive Header Descriptor and the Receive Buffer Descriptor is explained in *[1]*. At any point of time if the DMA finds the discard pattern in the RX TAG FIFO it will not update the *descriptorDoneRx* bit and reuse the same descriptors for the next frame.

The DMA first moves the MAC Header from the RX FIFO and stores them in the buffer pointed by the Receive Header Descriptor. Then it moves the frame payload from the RX FIFO and stores them in the buffers pointed by the Receive Buffer Descriptor. After this the RX Vector, TSF and the MPDU Status Information are written to the Receive Header descriptor. The address of the first receive payload buffer in which the frame payload has been written is updated in the header descriptor for quick access from SW. The receive queue halts if the HW runs out of header or buffer descriptors, and newly received frames may be discarded if the RX FIFO runs out of space. A first interrupt is generated indicating the linked list is empty and if RX FIFO overflows a second interrupt is raised to SW indicating *Receive Overflow*. The almost full flag of the RX FIFO is set to one position below the maximum size of the FIFO. The data is written to the RX FIFO by the RX Controller until the almost full flag is asserted. If there is more than one quadlet to be written in the FIFO after this then the discard pattern is written. This helps the DMA to be in synchronization during reception.

2.3 Transmission of Immediate Response

2.3.1 Transmission of Immediate Response or Protection frame (CTS/ACK/CF-END/RTS)

The MAC HW can automatically handle the immediate response generation as well as the transmission of protection frame or TXOP truncation. All these tasked are managed by the MAC Controller based on information coming from the Policy table (for RTS or CTS) or at this end of a TXOP (CF-End) or from the RX Controller in case of immediate response (ACK, CTS, BA, BFR). The BA and BFR immediate response transmissions are detailed in *2.3.2 Transmission of BlockAck* and *2.3.3 Transmission of Beamforming Report*.

In case of RTS, CTS, ACK or CF-End, the MAC Controller trigs the TX Controller using a flag (*sendAck*_p, *sendRTS*_p, *sendCTS*_p, *sendCFEND*_p) indicating the type of frame to be transmitted as well as all the information required for the frame creation (such as *addr1*, *addr2* and *duration*) and its transmission (*formatMod*, *MCSIndex*, *legLenght*, *legRate*, *htLenght*, *BW*, …). It also indicates to the TX Controller if the frame shall be transmitted at SLOT of SIFS boundary.

Upon reception of the trigger, the TX Controller formats the requested frame, push them through the FCS into the MAC-PHY TX FIFO and waits for the SIFS or SLOT indication before triggering the MAC-PHY interface.

The following figure summarizes the data flow in case of Control Frame HW transmission.

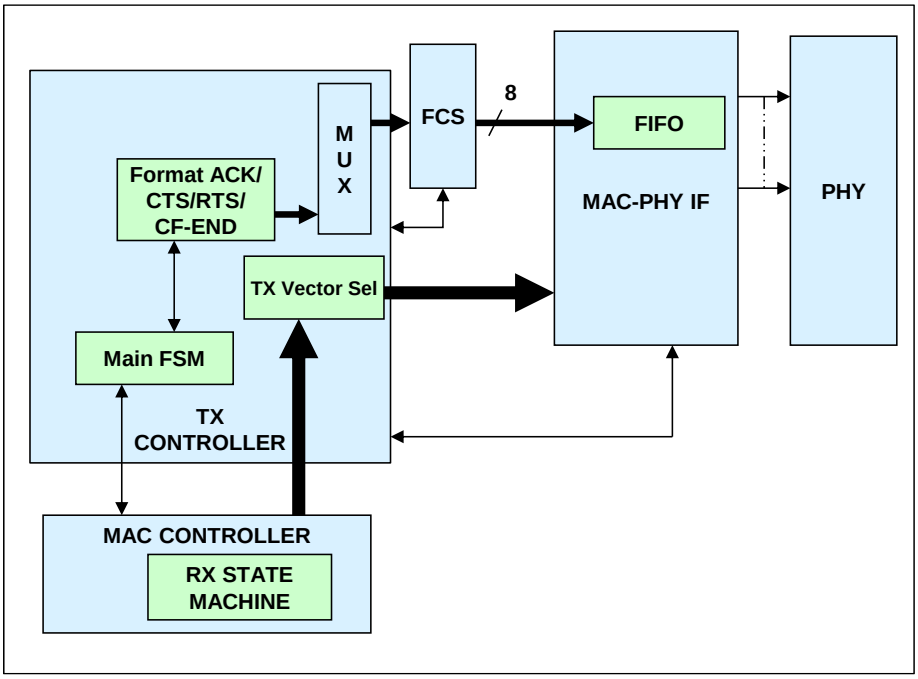


Figure 5: Block Level HW Tx Flow Diagram

2.3.2 Transmission of BlockAck

Upon reception of a A-MPDU or BAR to our device which requires the transmission of a BACK frame as immediate response, the BlockAck controller is triggered to update and prepare the BA bitmap.

At the end of the A-MPDU or BAR reception, the MAC Controller Rx performs the standard checks done before starting an immediate response and trigs the TX Controller to transmit a BA after SIFS.

Then, the TX Controller prepares the MAC Header and launches the transmission. When the MAC Header has been pushed into the TXFIFO, the TX Controller trigs the BA Controller to provide in the correct format the Bitmap (using *psBitmapReady*, *psBitmapValid* and *psBitmap* interface). Then, the bitmap goes through the FCS module and is pushed into the MAC-PHY TX FIFO.

The following figure summarizes the data flow in case of BA transmission.

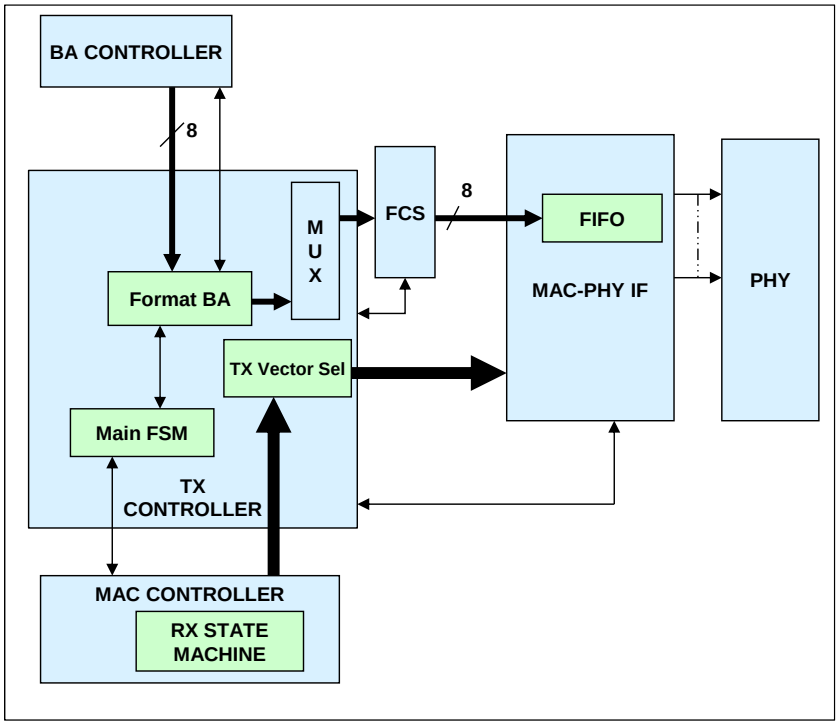


Figure 6: Block Level BA Tx Flow Diagram

2.3.3 Transmission of Beamforming Report

Upon reception of NDPA+NDP frame sequence addressed to our device which requires the transmission of a Beamforming Report frame as immediate response, the Beamforming Controller performed all the checks and trigs the Beamforming HW Accelerator (located on the PHY side) to compute the SVD and prepare the beamforming reports based on the parameters extracted from the NDPA and NPD.

At the end of the NDP reception, the MAC Controller Rx performs the standard checks done before starting an immediate response and trigs the TX Controller to transmit an Action frame containing a Beamforming Report after SIFS. Then, the TX Controller prepares the MAC Header and launches the transmission. When the MAC Header and the Action Frame header have been pushed into the TXFIFO, the TX Controller sets the *bfrDataReady* to indicate the Beamforming HW Accelerator to push the report. Once the report is ready inside the Beamforming HW Accelerator, it is provided in the correct format using *bfrDataValid* and *bfrData* signals. Then, it goes through the FCS module and is pushed into the MAC-PHY TX FIFO. Note that the SVD computation is performed during the preamble transmission. The system shall be correctly sized to guaranty the availability of the beamforming report before the end of the preamble. Otherwise, an underrun will be detected.

The following figure summarizes the data flow in case of Beamforming Report transmission.

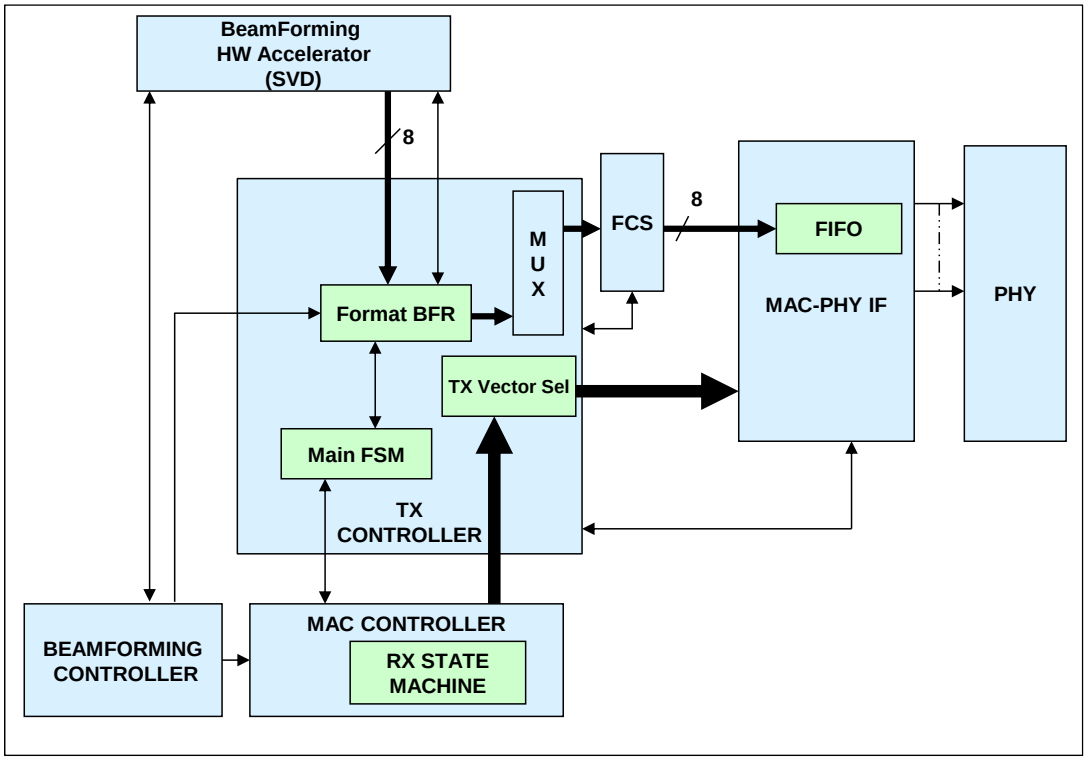


Figure 7: Block Level BFR Tx Flow Diagram

2.3.4 Transmission of Trigger Based Frame

Upon reception of a trigger frame addressed to the device, all the parameters included in the trigger frame are extracted by the rxController and are provided macControllerRx.

The trigger based frames can be split in two groups.

The ones which are fully generated by HW such as MU-BA, MU-CTS, Beamforming Report, NDP Feedback Report and Bandwidth Query Report.

The ones which are created by software such as Basic Trigger Based frame and Buffer Status Report.

2.3.4.1 Trigger Based transmission without software involvement

2.3.4.2 Trigger Based transmission with software involvement

In case of reception of Basic Trigger or Buffer Status Report Poll, the trigger based response is generated by the software. Thus, the rxController indicates using the Rx FiFO tag (XXX) to the DMA Engine rxList Proc that the current frame is a Trigger Frame which required SW processing. When the frame is successfully received, the DMA engine generates the XXX interrupt and the pointer to the trigger frame Rx Header Descriptor pointer is provided to the SW in XXX register.

Upon reception of this interrupt, the SW parses the Trigger Frame and constructs the expected response following the parameter of the trigger frame. Once built, the response frame is linked by the SW on the TB DMA channel. Note that such frame does not have policy table linked to the THD.

At the end of the Trigger frame reception, the MAC Controller Rx performs the standard checks done before starting an immediate response and trigs the TX Controller to transmit the SW programmed frame. All the parameters used to create the TX Vector come from the trigger frame and not from the THD/Policy table.

0 Basic Trigger

1 Beamforming Report Poll (BRP)

2 MU-BAR

3 MU-RTS

4 Buffer Status Report Poll (BSRP)

5 GCR MU-BAR

6 Bandwidth Query Report Poll (BQRP)

7 NDP Feedback Report Poll

3 Platform Interface

3.1 Overview

The Platform interface (also called Host Interface) provides AHB Slave and AHB Master interfaces to the MAC HW. The platform interface has following blocks:

- AHB Slave
- AHB Master

The AHB Slave interface is used by software to program the Control registers and monitor the Status registers of the core. The DMA engine uses the AHB Master to read and write data from system memory. The following block diagrams show how the platform interface is connected with the rest of the MAC Core in case of Single User transmission only or in case of MU-MIMO TX support.

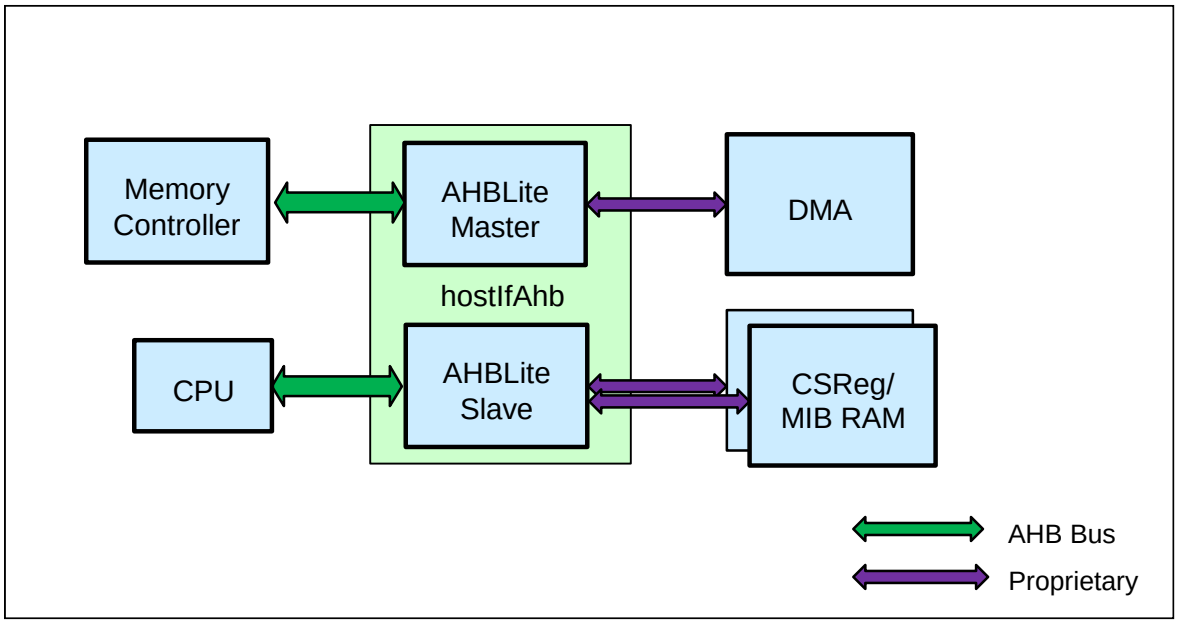


Figure 8: AHB Platform Interface block diagram (without MU-MIMO Tx support)

3.2 AHB Slave

3.2.1 Functional Description

The SW accesses all the registers present in MAC HW using the AHB Slave interface. Only 32-bit read or write transactions are supported on the slave interface. The slave state machine gets trigger when *hSSel* for this slave is asserted by the AHB bus decoder. The slave keeps *hSReadyOut* asserted always so that the write transaction is completed with zero wait states. Depending on the clock domain access, wait states, might be asserted

The state machine latches the first address and de-asserts *hSReadyOut* till it receives read or write transaction-completed indication from the MAC core. For write transactions, the slave passes the *hSWData* bus to the MAC core. For read transactions, the slave passes the data read from MAC core to the *hSRData* bus.

For all error responses slave generates a two-clock cycle error response.

3.2.2 Interconnection

The AHB slave is interfaced with both the MIB controller, and the CSR block as described in *Figure 9: AHB slave interconnection diagram*. The address decoding is done in the AHB slave interface.

The AHB slave data out are always sampled on the *macPIClk*. The connected blocks must resynchronize the data in case they need to be made available to others clock domain. Taking into account the connected block cannot always provide the data in 1 clock cycle, the internal interfaces of the AHB slave has to insert wait states depending on MAC internal blocks requests. As example, the read accesses into the DMA/Platform registers bank is done without latency because this register bank is clocked on the *macPIClk* as the AHB Slave interface. However, wait states are inserted for read access into the MAC register bank due to the clock domain crossing mechanism as the MAC register bank is running on the *macCoreClk* whereas the AHB slave is running on the *macPIClk*

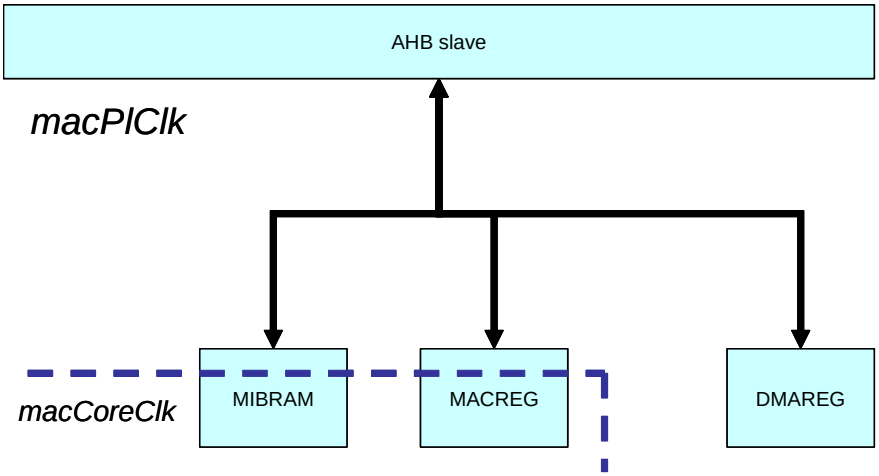


Figure 9: AHB slave interconnection diagram

3.3 AHB Master

3.3.1 Functional Description

The AHB master is used by the DMA engine to initiate read and write transfers to system memory. The AHB master supports 8-16-32-bit read or write transactions. The master supports increment bursts of unspecified length and single transfers. The AHB master handles 1KByte boundary crossing address burst transfers.

In case of reception of ERROR response from any slave, its notifies the DMA engine which will stop the transaction.

4 Control and Status Registers

4.1 Overview

The Control and Status Register block (henceforth referred to as CSR) implements the Control registers, Status Registers and Debug registers as per *[1]*. The registers are implemented in individual modules based on their operational clock and functionality. This block interfaces with most of the other modules in the MAC Core. The CSR block is organized into sub modules based on functionality but also based on clock domain as explained below.

All DMA registers operated in macPlClk are implemented in one module

All MAC Control and Status registers operated in macCoreClk are implemented in one module

Supports macPlClk and macCoreClk being fully asynchronous.

Implement a post-write mechanism to not halt the platform bus during write access due to the cross domain resynchronization.

Both macPlClk and macCoreClk clock domains are considered asynchronous. In order to support this, the CSR module performs a pre-decoding of the address between DMA registers and the core registers. Then, according to the pre-decoding, the Platform Slave access is resynchronized to the right clock domain and the access is performed.

Each register sub-module has its own address sub-decoder.

In case of write accesses, a post-write mechanism is implemented where the Platform Bus is released when the write access is crossing the clock domains. If a subsequent write or read access is requested when a post-write is on-going, the hSReadyOut is kept low until the completion of the post-write access and then the second access is performed. In the case where the clock domain is not crossed, the bus is accessed without wait states.

In the case of read access, there is no clock domain crossing. As a consequence, the bus is accessed without wait state. In the case of clock domain crossing, the number of wait states needed to properly resynchronize the read data is inserted.

The figure below gives the block diagram of CSR block:

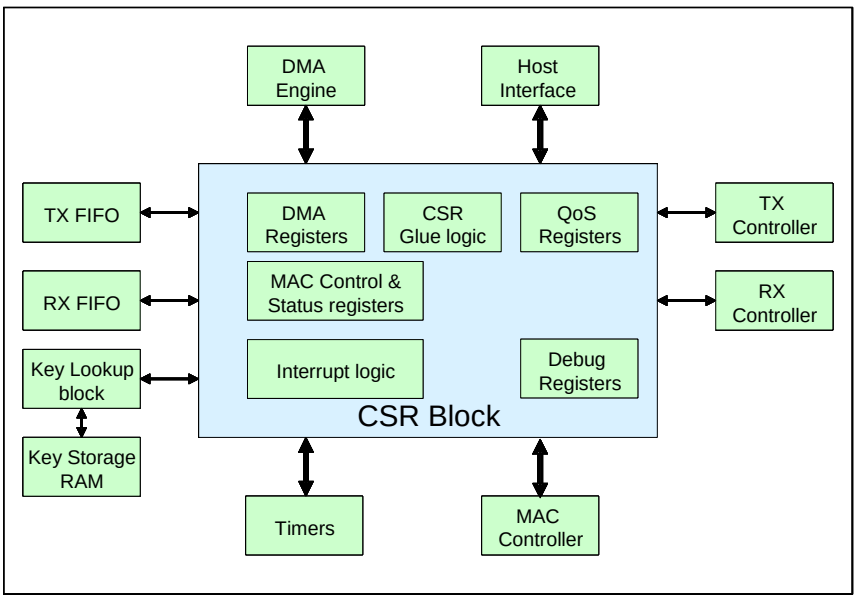


Figure 10: Control and Status Register block diagram

4.2 Registers clocking scheme

As previously described the register block takes care of the clock domain crossing information. It is responsible for synchronization of all registers to the required clock domain all the registers.

The following figure shows a block simplified diagram with clock domains representation:

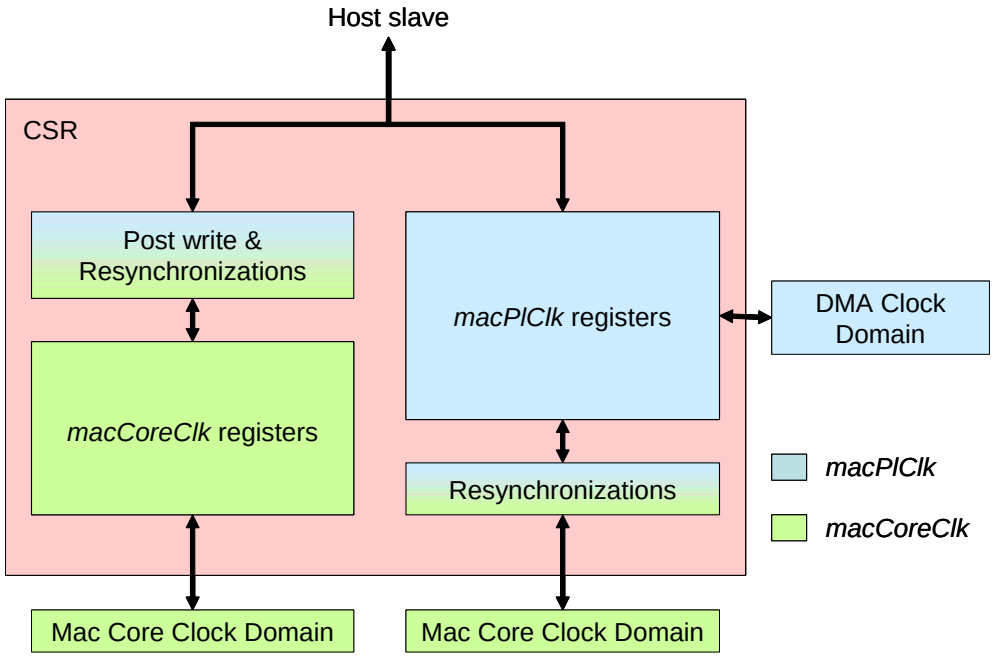


Figure 11: Clock domain handling

5 Interrupt Controller

5.1 Overview

The Interrupt Controller is in charge of interrupts generation coming from various sources like DMA Engine, MAC Controller, MAC-PHY Interface, etc and multiplexes them together to two interrupts outputs (*interruptGen_n* and *interruptTxRx_n*). Each interrupt source can be individually masked, cleared and even generated by software (for debug purposes). It implements also the interrupt moderation scheme described in the chapter *5.4*, *Interrupt Moderation Scheme*.

The interrupts generated by this block are classified into two types:

- General Purpose interrupts (refer section *5.2*, *General Purpose interrupts*)
- Transmit & Receive interrupts (refer section *5.3*, *Transmit & Receive Interrupts*)

The MAC HW outputs two interrupts lines (one for each interrupt type) on the pins name *interruptGen_n* and *interruptTxRx_n*. Both interrupt lines are active low signals and are level triggered (i.e. the interrupt line is forced low when any bit of the corresponding interrupt register is set and stays low until none of the bits are set). These two interrupt lines are generated on the *macPIClk* clock.

5.2 General Purpose interrupts

An interrupt is issued when an event trigger is received from an interrupt source and the interrupt bit has been unmasked by software. The interrupt unmask pattern is set in *genIntUnmaskReg* register. If software writes a one to any interrupt bit in the *genIntEventSetReg* address for debug purposes, an interrupt is generated. If software writes a one to any interrupt bit in *genIntEventClearReg* address, the pending interrupt is cleared. Reading the *genIntEventSetReg* or *genIntEventClearReg* register returns the current state of the *genIntEventSetReg* register.

When any bit in this register is set, the *interruptGen_n* line is asserted. When all the bits in this register are cleared, the *interruptGen_n* line is de-asserted.

5.3 Transmit & Receive Interrupts

The interrupt is issued when the event trigger is received from a module and the interrupt has been unmasked by software. The interrupt unmask pattern is set in *txRxIntUnMaskReg* register. If software writes a one to any interrupt bit in the *txRxIntEventSetReg* address for debug purposes, interrupt is generated. If software writes a one to any interrupt bit in *txRxIntEventClearReg* address, the pending interrupt is cleared. Reading the *txRxIntEventSetReg* or *txRxIntEventClearReg* register returns the current state of the *txRxIntEventReg* register.

When any bit in this register is set, the *interruptTxRx_n* line is asserted. When all the bits in this register are cleared, the *interruptTxRx_n* line is de-asserted.

5.4 Interrupt Moderation Scheme

To reduce the overhead of multiple interrupts which might degrade the overall system performance, the Interrupt Controller includes an interrupt moderation mechanism. This mechanism allows filtering the number of interrupts generated by collecting several events and by generating only one interrupt to SW. This mechanism is based on a set of two different timers (listed below) which generate an interrupt when they expire.

1. Absolute Timers defined in section *12.3.2.7*, *Absolute Timers*.
2. Packet Timers defined in section *12.3.2.8*, *Packet Timers*.

6 DMA Engine

6.1 Overview

There are 6 transmit DMA logical channels and 2 receive DMA logical channels. Of the 6 transmit channels 4 are used for the 4 ACs of EDCA, 1 is used for Beacon frame transmission and 1 is used for Trigger Based transmission. One of the 2 receive DMA logical channels is for the Header of the received frame and the other is for the payload of the received frame.

The block diagram shows the blocks of the DMA engine.

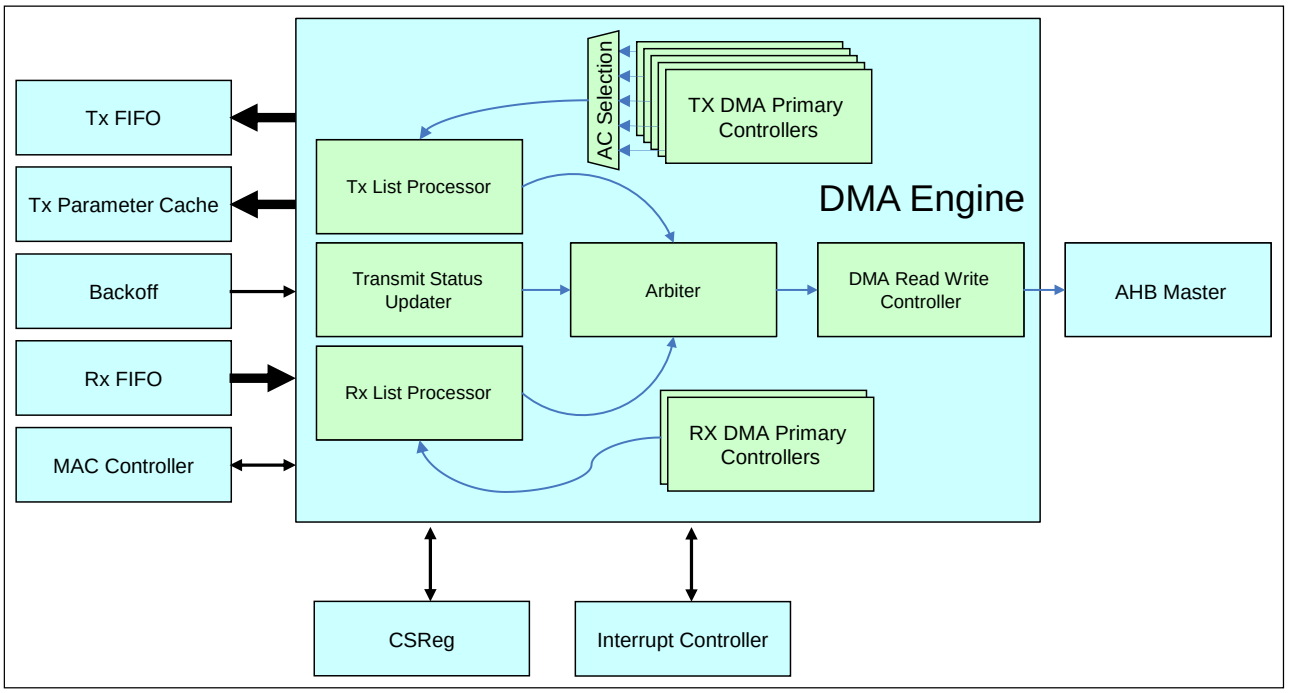


Figure 12: DMA Engine block diagram

6.2 DMA Primary Controller

6.2.1 Functional Description

The DMA is composed of six Primary Controllers :

- AC0 Primary Controller which manages the DMA chain list for Access Category AC_BK.
- AC1 Primary Controller which manages the DMA chain list for Access Category AC_BE.
- AC2 Primary Controller which manages the DMA chain list for Access Category AC_VI.
- AC3 Primary Controller which manages the DMA chain list for Access Category AC_VO.
- BCN Primary Controller which manages the DMA chain list for Beacon transmission.
- TB Primary Controller which manages the DMA chain list for Trigger Based transmission.

Each DMA Primary Controller maintains the overall state of each of the DMA channels, i.e. whether the DMA channel is in HALTED, PASSIVE, ACTIVE or DEAD state and controls the Transmit List Processor and the Receive List Processor. There are as many DMA Primary Controllers as there are DMA logical channels, since they operate in parallel.

Each DMA Primary Controller starts up in the HALTED state and moves into PASSIVE when triggered from SW. Multiple DMA channels can be in PASSIVE state at the same time, however, only one channel can be in ACTIVE at any time. The required channel is selected by the MAC core depending on the protocol on air. All the transmit DMA channels feed the single Transmit FIFO.

Once a DMA channel has moved to PASSIVE state, it waits for a trigger from the MAC core to start fetching frames from its linked list into the Transmit FIFO, or moving frames from the Receive FIFO to the linked list.

Transmit operation: When a protocol event occurs on air, say AC_VO winning contention, the MAC core signals AC_VO DMA channel to start fetching frames from System memory into the Transmit FIFO. Upon reception of this signal from the MAC core, the DMA Primary Controller checks the state of the relevant DMA channel. If the DMA channel is currently HALTED, it does not respond to this signal. If the DMA channel is currently PASSIVE, it moves to ACTIVE and triggers the Transmit List Processor.

A transmit trigger to the DMA engine (A) is given 3µs before the start of the *txReq* signal on air. The DMA engine has 2µs (1µs for MAC Processing) to fetch at least the first Header Descriptor and Policy Table Entry and store the parameters in registers in the MAC HW. The MAC core requires these parameters at the transmit trigger to the MAC core (B).

This operation is shown in the diagram below:

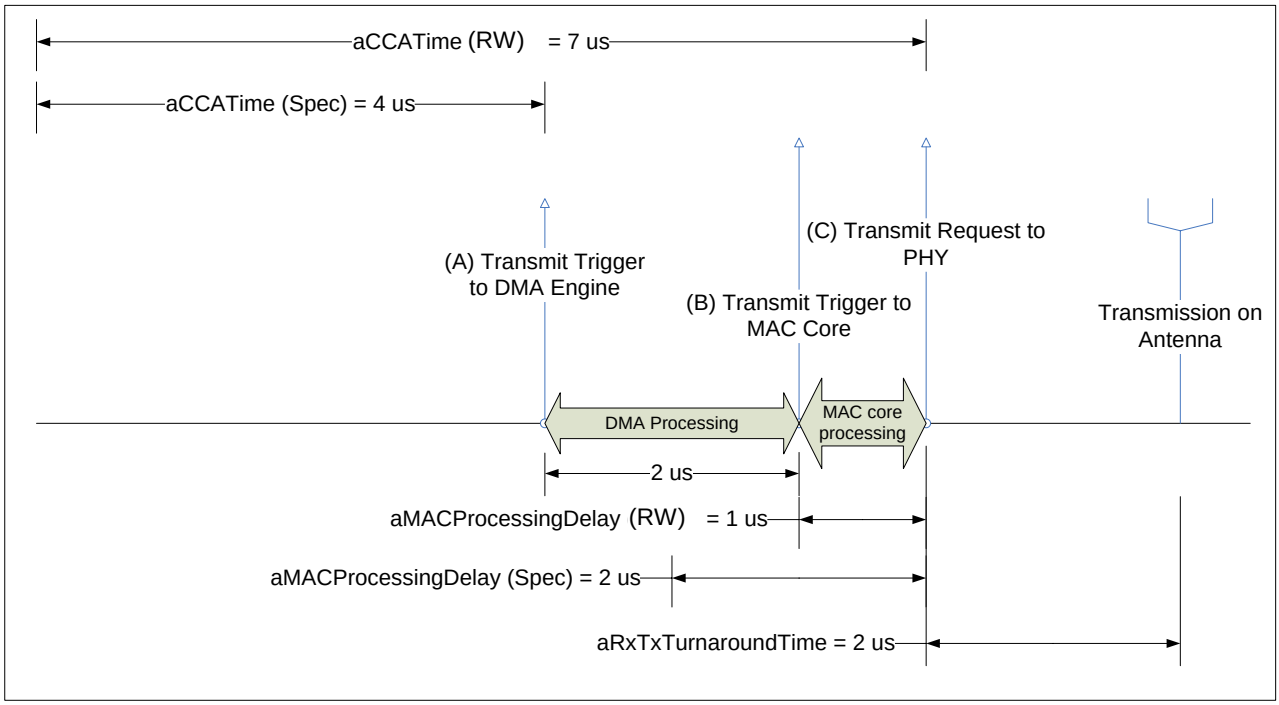


Figure 13: Timing relationship between DMA operations and protocol event

Note that all these timings are strongly linked with the macPIClk frequency.

The Transmit List Processor once triggered initiates the movement of the data from the System Memory to the Transmit FIFO.

The DMA gets the indication of end of the TXOP from the MAC Core. When this indication occurs the Transmit List Processor stops its operation and flushes the Transmit FIFO. If the *haltAfterTXOP* bit is set, the Transmit DMA Primary Controller state changes from ACTIVE to HALTED. If the *haltAfterTXOP* bit is not set, the Transmit DMA Primary Controller state changes from ACTIVE to PASSIVE.

After the transmission of every frame the status of transmission is indicated by the MAC Core. If the status is successful the DMA fetches the next MPDU for transmission. In case of retry the DMA flushes the Transmit FIFO and fetches the same MPDU for retransmission.

Receive Operation: The Primary Controllers for the two receive channels are triggered from the Receive List Processor as explained in section *6.7*, *Receive List Processor*.

6.2.2 DMA Primary Controller state machine

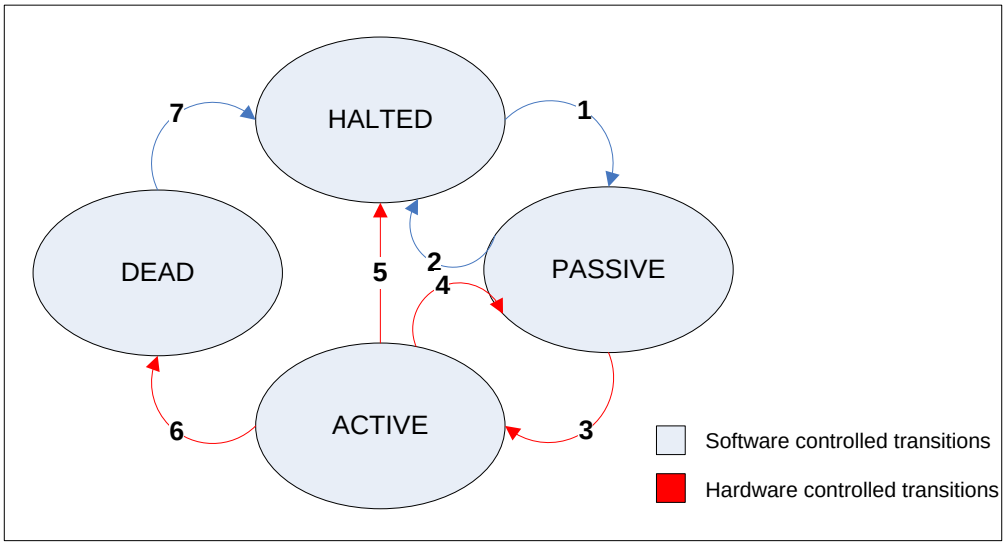


Figure 14: DMA Primary Controller states

State Name	State Description
HALTED	The state machine starts up in this state after reset. It may also transition to this state under other conditions. In this state, the state machine waits for the *newTail* or *newHead* signals to be asserted. More details about the SW control of a DMA channel can be found in [2].
PASSIVE	For transmit channels the Primary Controller waits for a trigger from the MAC core to start the DMA operations. For receive channels the Primary Controller waits for a trigger from the Receive List Processor to move to ACTIVE.
ACTIVE	The state machine triggers the Transmit List Processor to start operations, and waits for the Transmit List Processor to stop (if it does not find a valid next pointer), or for the MAC core to abort operations.
DEAD	In this state, the state machine waits for error recovery from SW.

Table 1: DMA Primary Controller FSM State description

Transition No.	Transition Description
1	This transition occurs if the *newHead* bit is set and the *queueHeadPointer* for that channel has a valid address. This transition also occurs if the *newTail* bit is set.
2	This transition occurs when the SW sets the *newHead* bit.
3	When triggered by the MAC core to read frames, it moves to the ACTIVE state. When the Receive List Processor indicates, this state machine moves to the ACTIVE state.
4	This transition is made when the MAC core aborts the current transmission operation and the *haltAfterTXOP* bit for that channel is not set. When the Receive List Processor indicates, this state machine moves to the PASSIVE state.

Transition No.	Transition Description
5	This transition is made on three conditions during transmission: The Transmit List Processor comes to a halt (because the *nextMADPTx* and *nextMDPTx* are invalid) OR The MAC core aborts the current transmission operation and the *haltAfterTXOP* for that channel is set OR The SW sets the *newHead* bit. When the Receive List Processor indicates, this state machine moves to the HALTED state.
6	When the Transmit or Receive List Processor indicates, this state machine moves to the DEAD state.
7	This transition occurs when the state machine is reset, as part of the error recovery from SW.

Table 2: DMA Primary Controller FSM State transition conditions

6.3 DMA Read Write Controller

6.3.1 Functional Description

The Read Write Controller interacts with the AHB Master Interface module to move data from and to the System Memory based on the request it receives from the Status Updater, Transmit List Processor or Receive List Processor.

It entertains a single request at a time. After the completion of the present request the next request is serviced.

The two kinds of request it can handle is a read from System RAM or a write to System RAM.

6.4 DMA Arbiter

6.4.1 Functional Description

The DMA Arbiter arbitrates between the Transmit List Processor, the Receive List Processor and the Transmit Status Updater. These three blocks may make contending requests for the AHB Master, hence the Arbiter block decides which of the three blocks gets access to the AHB Master.

The Arbiter follows this priority:

1. Receive List Processor request
2. Transmit Status Updater request
3. Transmit List Processor request

No request is ever aborted or pre-empted by another.

6.5 Transmit List Processor

6.5.1 Functional Description

The Tx List Processor is triggered by the DMA Primary Controller and is active when the DMA Primary Controller is in the ACTIVE state. It controls the AHB Master through the DMA Arbiter block and DMA Read Write Controller block. This block is responsible for following transmit linked list, reading descriptors and moving the data from memory buffers to the Transmit FIFO. This block also sets the Transmit Tag FIFO bits indicating which bytes in each quadlet have valid data. This saves the Transmit List Processor from having to perform a packing operation into the Transmit FIFO. It also sets the bits indicating the start and end of the MPDU in the Transmit FIFO as follows:

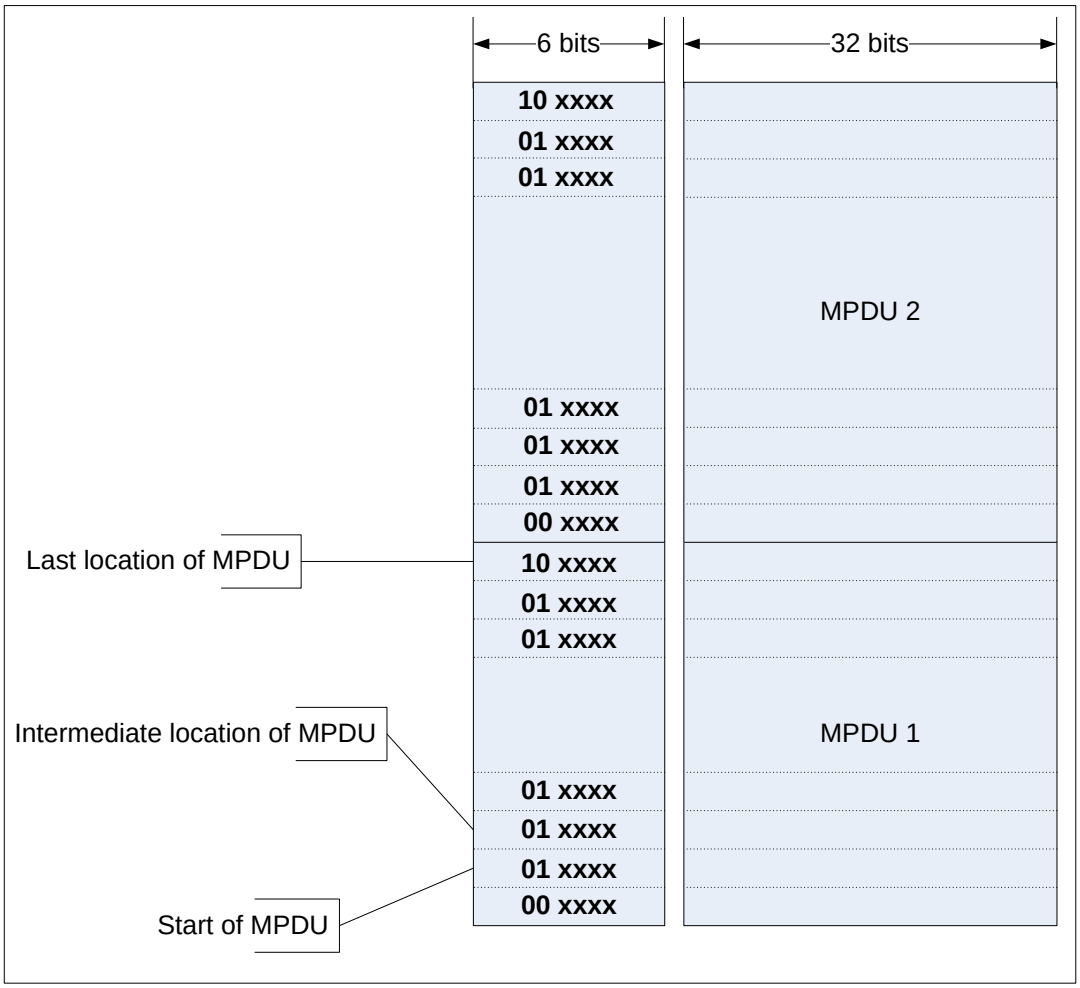


Figure 15: Transmit FIFO with the Transmit Tag FIFO

This block manipulates the *currentPointer*: Only one current pointer exists for all the logical channels since only one DMA channel is active at any time. The current pointer points to the descriptor that is currently being handled by this block.

When this block is triggered for a particular DMA channel, it starts at the address pointed to by the *statusPointer* for that channel. The Header Descriptor pointed to by the *statusPointer* has to be fetched. An AHB read request is asserted along with the starting address and the number of quadlets that should be read. The DMA Arbiter block checks if there is any other higher priority request and then makes a decision to provide the grant for Transmit List Processor. The Read Write Controller processes this request. When the DMA Read Write Controller block signals the completion of the command, this block checks if HW has already handled this descriptor by checking the *descriptorDoneHWTx* and the *whichDescriptor* bits.

If the bit is set, the HW checks if at least one of the next descriptor pointers is valid. If the *nextAFrmExSeqP* or *nextMDPTx* contains a valid address, the block continues operation; else it moves to IDLE and signals the DMA Primary Controller to change the state of this channel to HALTED.

The lifetime expiry check is done by the Transmit List Processor block itself. If the lifetime has expired, the indication is given to the status updater block to update the same in the status information of the descriptor. Once frame Lifetime has expired for a particular atomic frame exchange sequence, the Transmit List Processor starts fetching the next Atomic Frame Exchange sequence if it is valid; else it moves to IDLE and signals the DMA Primary Controller to change the state of this channel to HALTED.

Once the lifetime check passes and it is determined if the current descriptor needs to be processed, the Policy Table Entry associated with this MPDU may need to be fetched. The Policy Table is fetched for only the starting of every Atomic Frame Exchange sequence. If the Policy Table has to be fetched this block asserts the AHB Read request with

the Policy Table address as the starting address and the depth of the Policy Table as the number of quadlets. In case of TB Channel, the transmit header descriptor does not include policy table. In this specific case, the Policy Table is not fetch.

Next, the data from the buffer associated with the current Header Descriptor is DMAed into the Transmit FIFO.

When the buffer associated with the Header Descriptor is empty, the state machine checks whether another buffer has been chained to this descriptor, i.e. is the First Payload Buffer Descriptor Pointer valid. If the First Payload Buffer Descriptor Pointer is invalid, then there are no Buffer Descriptors attached. If the First Payload Buffer Descriptor Pointer is valid, the first Buffer Descriptor is fetched.

Next the data from the buffer associated with the current Buffer Descriptor is DMAed into the Transmit FIFO. When the buffer associated with the Buffer Descriptor is empty, the state machine checks whether another buffer has been chained to this descriptor, i.e. is the Next Buffer Descriptor pointer valid.

An MPDU is thus read from the memory and moved into the Transmit FIFO, until the Next Buffer Descriptor is invalid. When the First/Next Buffer Descriptor is invalid, the state machine checks the Next MPDU Descriptor Pointer (*nextMDPTx*). If the Next MPDU Descriptor Pointer is valid, then the entire process of fetching the MPDU is restarted.

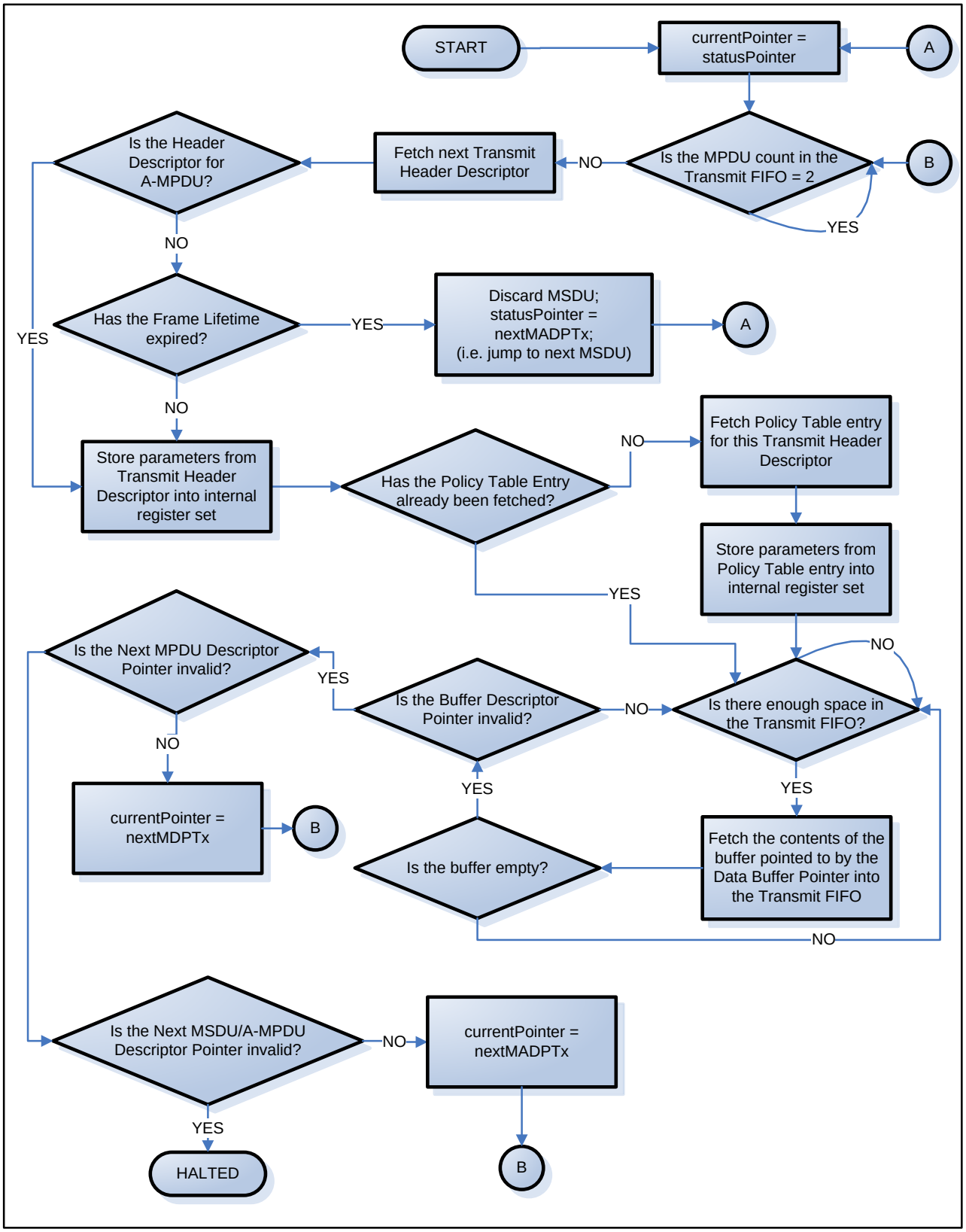


Figure 16: DMA channel operation in HW

If the Next MPDU Descriptor Pointer is invalid, the HW checks the Next Atomic Frame Exchange Sequence Pointer (*nextAFrmExSeqP*). If this pointer is valid, the block continues operation; else it moves to IDLE and signals the DMA Primary Controller to change the state of this channel to HALTED.

At any point, if the MAC Core indicates that the last fetched frame has to be retried the Transmit List Processor flushes the Transmit FIFO and re-fetches the frame pointed by the *statusPointer.*

The DMA Transmit Controller always fetches the immediate next MPDU in the linked list into the Transmit FIFO ahead of time in order to handle a MPDU burst or the formation of an A-MPDU. The Transmit FIFO is a window into the frame and may hold part of a MPDU (if the frame is larger than the FIFO), or may hold (at most) two MPDUs if the MPDUs are small enough. Even if there is space for a third MPDU to be fetched into the Transmit FIFO, the state machine does not do so.

If any error is detected while performing any of the DMA operations like reading a descriptor or writing data into a buffer in system memory, the state machine asks the Primary Controller of the relevant DMA channel to change the state to DEAD and this state machine halts.

6.6 Transmit Status Updater

6.6.1 Functional description

The Transmit Status Updater block gets the relevant status from the MAC core, and updates the status in the descriptor. This may occur even if the current state of the DMA channel is halted. It manipulates the *statusPointer* for each of logical DMA channels. There is one status pointer per logical channel. The status pointer points to the first Header Descriptor in the list whose *descriptorDoneHWTx* bit is not set, OR it points to the last Header Descriptor in the list when there is no next descriptor chained after this descriptor.

If the frame is an MPDU, then the status of the transmission is updated in the *Status Information* field of the Header Descriptor for the MPDU. If the frame is an A-MPDU, then the status of the transmission is updated in the *Status Information* field of the A-MPDU Header Descriptor for the entire A-MPDU. The Header Descriptor for each constituent MPDU within the A-MPDU is not updated.

Hence the *statusPointer* points to an individual MPDU Header Descriptor when the MPDU is not part of an A-MPDU and points to the A-MPDU Header Descriptor when the MPDU is part of an A-MPDU. The *statusPointer* is updated to the next frame only when the entire A-MPDU is transmitted. Note that the *statusPointer* can never point to the MPDU Header Descriptor of a constituent MPDU of an A-MPDU.

When an active DMA channel receives a transmit status update, it updates the *statusPointer* value for that channel with the address of the next Header Descriptor.

When HW sets the *descriptorDoneHWTx* bit in the Header Descriptor of any frame, it checks the *interruptEnTx* bit. If the *interruptEnTx* bit is set, the HW raises a *transmission trigger* interrupt to SW.

After an unaggregated MPDU or A-MPDU transmission is complete, six conditions can occur: RTS success, Normal success, A-MPDU failure, Partial failure, RT Limit Reached and LT Expired.

The table summarizes the various events that can occur in HW and the status fields that are updated in each case.

Event	Defined as	Status fields updated
RTS formed by HW is successful	RTS Success	The *numRTSRetries* field of the Header Descriptor of the RTS is updated.
RTS formed by SW is successful	RTS Success	The *numRTSRetries* field of the Header Descriptor of the RTS is updated. The *frmSuccessfulTx* bit is set. The *descriptorDoneHWTx* bit is NOT set in this case.
MPDU is successful	Normal Success	The *numMPDURetries* field of the Header Descriptor of the MPDU is updated. The *frmSuccessfulTx* bit is set. The *descriptorDoneHWTx* bit is set.
A-MPDU is successful	Normal Success	The *frmSuccessfulTx* bit is set. The *descriptorDoneHWTx* bit is set.
A-MPDU is unsuccessful	A-MPDU Failure	The *descriptorDoneHWTx* bit is set.
RTS formed by HW is unsuccessful	Partial Failure	The *numRTSRetries* field of the Header Descriptor of the MPDU or A-MPDU is updated.
RTS formed by SW is unsuccessful	Partial Failure	The *numRTSRetries* field of the Header Descriptor of the RTS is updated.
MPDU is unsuccessful	Partial Failure	The *numMPDURetries* field of the Header Descriptor of the MPDU is updated.
RTS formed by HW fails	RT Limit Reached	The *numRTSRetries* field of the Header Descriptor of the MPDU or A-MPDU is updated. The *retryLimitReached* bit is set. The *descriptorDoneHWTx* bit in set.
RTS formed by SW fails	RT Limit Reached	The *numRTSRetries* field of the Header Descriptor of the RTS is updated. The *retryLimitReached* bit is set. The *descriptorDoneHWTx* bit is set.
MPDU fails	RT Limit Reached	The *numMPDURetries* field of the Header Descriptor of the MPDU is updated. The *retryLimitReached* bit is set. The *descriptorDoneHWTx* bit in set.
RTS formed by SW expires	LT Expired	The *lifetimeExpired* bit of the Header Descriptor of the RTS is set. The *descriptorDoneHWTx* bit is set.
MPDU or A-MPDU expires	LT Expired	The *lifetimeExpired* bit of the Header Descriptor of the MPDU is set. The *descriptorDoneHWTx* bit is set.

Table 3: Air events and DMA status

6.7 Receive List Processor

6.7.1 Functional description

Unlike the Transmit DMA, the DMA Primary Controller does not control the operation of the Receive List Processor. The Receive List Processor is always active. It controls the AHB Master through the DMA Arbiter block and DMA Read Write Controller block. This block is responsible for following receive linked list, reading descriptors and moving the data from the Receive FIFO to the memory buffers.

This block manipulates the *statusPointers* for the two receive DMA channels and the *currentPointer* for the Receive Payload DMA channel. The Receive Header DMA channel does not have a *currentPointer*. The *statusPointer* is enough.

When the Receive FIFO threshold (*rxFIFOThreshold*) level is reached, this module checks the state of the Receive Header DMA channel. If this DMA channel is currently HALTED, the state machine stops and waits for the Header DMA channel to move to PASSIVE. In this condition, the current and the subsequent MPDUs may be discarded if the Receive FIFO is full.

If the DMA channel is currently PASSIVE, the DMA Primary Controller is asked to change the state of the channel to ACTIVE, and this state machine proceeds. It starts at the address pointed to by the *currentPointer* for the Receive Header DMA channel. The Header Descriptor pointed to by the *currentPointer* is fetched.

The MAC Header is read from the Receive FIFO and moved into the buffer associated with this Header Descriptor.

Next, the state of the Receive Payload DMA channel is checked. If this DMA channel is currently HALTED, the state machine stops and waits for the Header DMA channel to move to PASSIVE. In this condition, the current and the subsequent MPDUs may be discarded if the Receive FIFO is full.

If the DMA channel is currently PASSIVE, the DMA Primary Controller is asked to change the state of the channel to ACTIVE, and this state machine proceeds. It starts at the address pointed to by the *statusPointer* for the Payload DMA channel. The Payload Descriptor pointed to by the *statusPointer* is fetched.

Next, the data from the Receive FIFO is written into the buffer associated with this descriptor. When the buffer associated with the Payload Descriptor is full and there is data in the Receive FIFO for this MPDU, the state machine checks whether another buffer has been chained to this descriptor, i.e. is the Next Payload Buffer Descriptor Pointer valid. The next Payload Buffer Descriptor is fetched and the data is moved from the Receive FIFO into the buffer associated with this descriptor. The *currentPointer* of the Payload DMA channel is updated with the address of the current descriptor being handled, but the *statusPointer* is updated only if the Header Descriptor of the Header DMA channel is successfully updated. This process continues until the MPDU has been completely moved from the Receive FIFO to system memory.

While reading the data from the Receive FIFO, the Rx Tag FIFO is examined to determine the start and end of the MPDU in the Receive FIFO, as well as the internal status of the MPDU. The MAC core writes the tag bits to simplify the operation of the Receive List Processor.

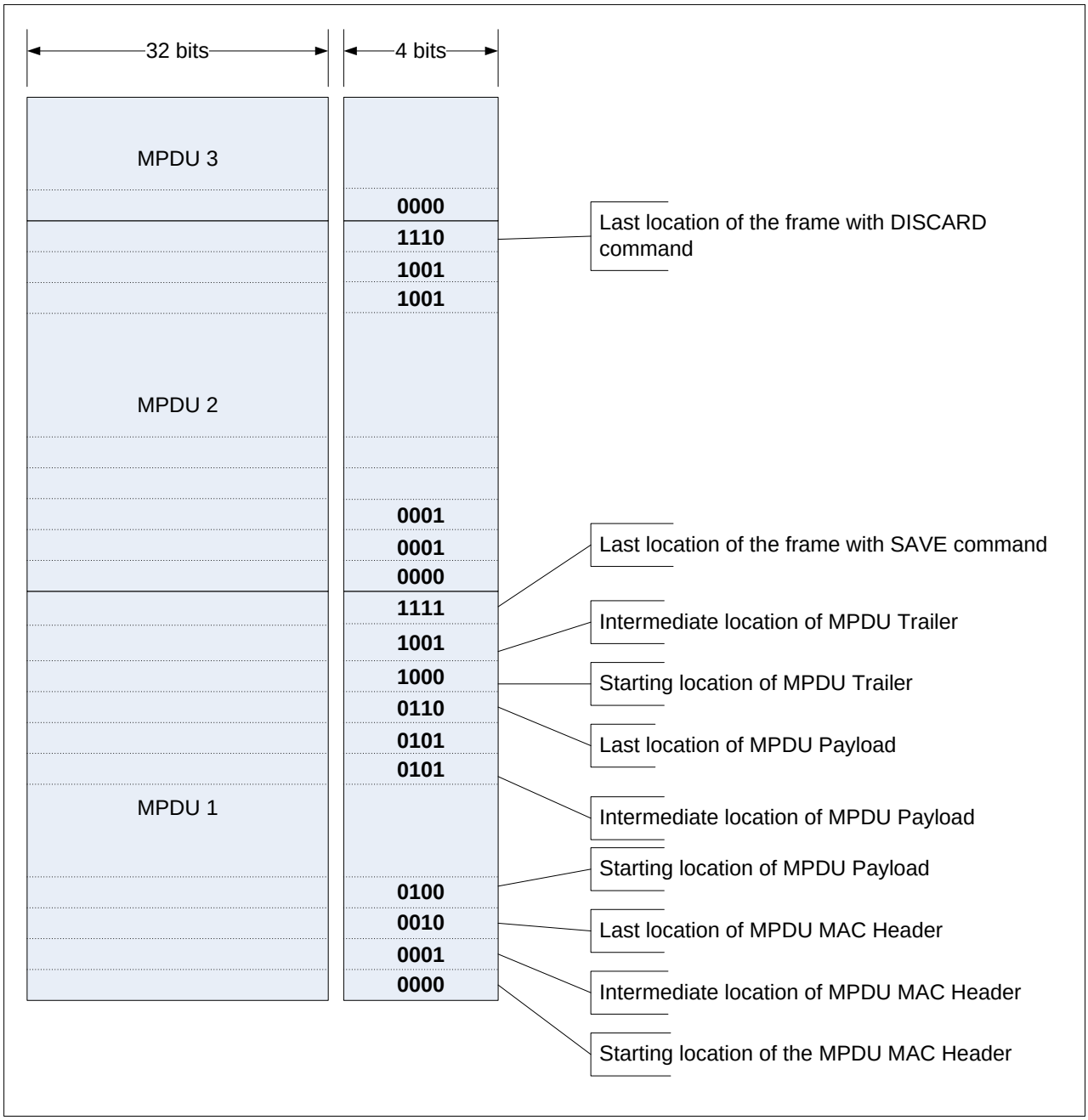


Figure 17: Receive FIFO with the Receive Tag FIFO

The Receive List Processor moves an MPDU starting from the first location, indicated by "0001" in the Receive Tag FIFO, until it finds either "1111" or "0000" at a particular location of the Tag FIFO. A "0000" indicates that this is the last location (or quadlet) of this MPDU and the MPDU should be **saved**. A "1111" indicates that the MPDU should be **discarded**[1].

When this block reads "0101" the *lastBuffer* in the Receive DMA Payload Descriptor is updated to indicate this forms the end of the link of Payload Descriptors.

[1] The MAC core can discard a frame for different reasons: the packet had some error and should not be passed to SW, the packet did not have an error but should not be passed to SW anyway, this is the last location of the receive FIFO and the MPDU has not ended i.e. Receive FIFO overflow.

When this block reads “1111”, it sets the *currentPointer* of the Payload DMA channel back to the *statusPointer* so that the descriptors can be reused. It asks the Primary Controllers of both channels to change the state to PASSIVE, and the state machine waits for the next trigger.

When this block reads “0000”, the address of the first receive Payload Buffer in which the MPDU payload has been written is updated in the Header Descriptor field *firstBufferDPRx* for quick access from SW. The *statusPointer* register of the Payload DMA channel is updated with the contents of the *currentPointer*.

It asks the Primary Controllers of both channels to change the state to PASSIVE, and the state machine waits for the next trigger.

The MPDU payload can span multiple Payload Buffers, but multiple MPDUs cannot reside in the same buffer. Note that certain frames which have zero payload (like PS-Poll, or QoS Null) will not use a Payload Descriptor and will only have a Header Descriptor.

Note that at any point if the *Next Buffer Descriptor Pointer* or *Next Header Descriptor Pointer* is found to be invalid, a *Receive DMA Empty* interrupt is raised to SW.

If any error is detected while performing any of the DMA operations like reading a descriptor or writing data into a buffer in system memory, the state machine asks the Primary Controller of the relevant DMA channel to change the state to DEAD and this state machine halts.

In case of Trigger frame indicated by the Rx Tag 1110, the rxListProc will generate the *tbProtTrigger* interrupt, will store the Rx Header address in *rxHeaderTFPtrReg* and will mark it valid by setting *rxHeaderTFPtrValid* when the MPDU save tag is received. In case of MPDU discard, none of the previous actions are done.

7 TX FIFO

7.1 Functional Description

Transmit FIFO (henceforth referred to as TX FIFO) is used to buffer the Tx data fetched by the DMA, to cross the *macPl1Clk* and *mac1Clk* clock domains. It supports also 32bits write accesses from the DMA Engine and 8-bits read accesses from the Transmit Controller thanks to the tags carried by the TX FIFO TAG. Frames chained from SW are stored temporarily here before transmission. This helps in compensating for the latencies in reading the data from the system memory and handling the data speeds on the wireless medium.

Even if the DMA Engine supports five channels, only one channel is transmitted simultaneously. This is why only a single TX FIFO is required.

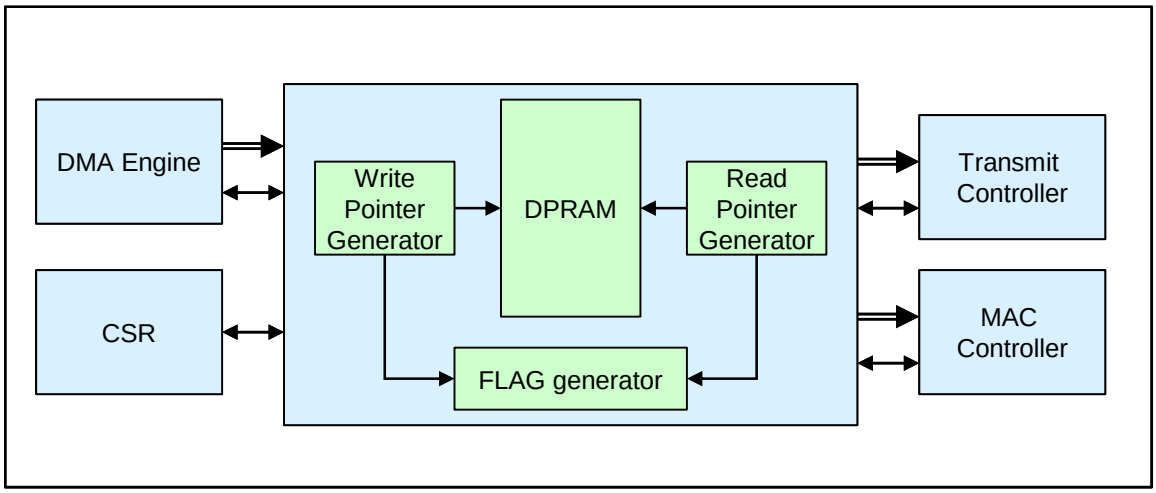


Figure 18: TX FIFO Block Diagram

7.2 Two Port RAM

It is a two-port RAM of 64 words with one read port and one write port. Each word has a 38bits width to carry 32-bits data and the 6 additional flag bits. The write happens in *macPl1Clk* domain while read happens in *macCore1Clk* domain.

The Two-Port hard macro connections are available at the Top level to ease the integration.

7.3 Write Module

Write Module generates write pointer for TX FIFO in macPl1Clk domain and the different flags for the DMA Engine. The data coming from DMA Engine is written to the location pointed by write pointer. The write pointer is incremented for every write and points to next location. When TX FIFO is full, write pointer is not incremented further. The write interface to the DMA Engine is 32-bits + 6-bits for the Tags. (refer section *7.5, TX FIFO Tags*)

The TX FIFO provides information about its content to the DMA Engine using *txFifoAlmostFull* and *txFifoAlmostEmpty* flags.

txFifoAlmostFull indicates to the DMA Engine that the TX FIFO is almost full and can accept only one additional write access. When this signal goes high, the DMA must finish the on-going access. When the on-going access is finished, the DMA Engine pushes the data into the TX FIFO (one words remain when *txFifoAlmostFull* is high) and does not restart new access.

txFifoAlmostEmpty indicates to the DMA Engine that number of quadlets contained in the TX FIFO is less than the value defined in *dmaThresholdReg.txFIFOThreshold* register. Based on this signal, the DMA Engine can start an AHB transaction to fetch a part of data from memory.

7.4 Read Module

Read Module generates read pointer for TX FIFO in *macCore1Clk* domain. The read pointer is incremented with every read request from MAC Controller or TX Controller. When the TX FIFO is empty, the read pointer is not incremented further. TX Controller stops reading data from TX FIFO when the TX FIFO status shows empty.

As the read interface of the TX FIFO is 8-bits, the Read Module manages the read pointer and the byte selection based on the TX FIFO Tags described in *7.5 TX FIFO Tags.*

Along with the 8-bits data, the Read Module provides also two MPDU delimiters flags (*txDataMPDUStart* and *txDataMPDUEnd*) based on the TX FIFO Tag to the TX and MAC Controller. Note that the *txDataMPDUStart* signal is high only when the TX or MAC Controller reads the first byte of an MPDU and the *txDataMPDUEnd* signal is high only when the last byte of the MPDU is read.

The TX FIFO provides information about its content to the TX and MAC Controller using *txFifoEmpty* flag.

> *txFifoEmpty* indicates to the TX and MAC Controllers that the TX FIFO is empty. If this signal is low, they can pop a new word from this FIFO else they are not allow to request a new data. If this signal is high, the MAC Controller or TX Controller are not allowed to read data from the TX FIFO and should generate an underrun error.

7.5 TX FIFO Tags

Along with the 32bits data, the TX FIFO carries 6 additional bits named flag which indicate the valid bytes and delimits the MPDUs.

The bits 3-0 are data bytes valid and the bits 5-4 are the MPDU delimiters.

Flag description	b5-b4	b3-b0
data (b31-b24) is valid	xx	1xxx
data (b23-b16) is valid	xx	x1xx
data (b15-b8) is valid	xx	xx1x
data (b7-b0) is valid	xx	xxx1
First quadlet of the MPDU	00	xxxx
Intermediate MPDU quadlet	01	xxxx
Last quadlet of the MPDU	10	xxxx
Reserved	11	xxxx

Table 4: TX FIFO Tags

8 RX FIFO

8.1 Functional Description

Receive FIFO (henceforth referred to as RX FIFO) buffers received data after RX Controller and if required Encryption engine completes processing. This Module interfaces to RX Controller, DMA Engine, CSR and Encryption Engine. Receive DMA Engine moves data from RX FIFO to system memory. The Receive DMA Engine takes care of discarding the data according to Tags and starting DMA operations at next frame boundary.

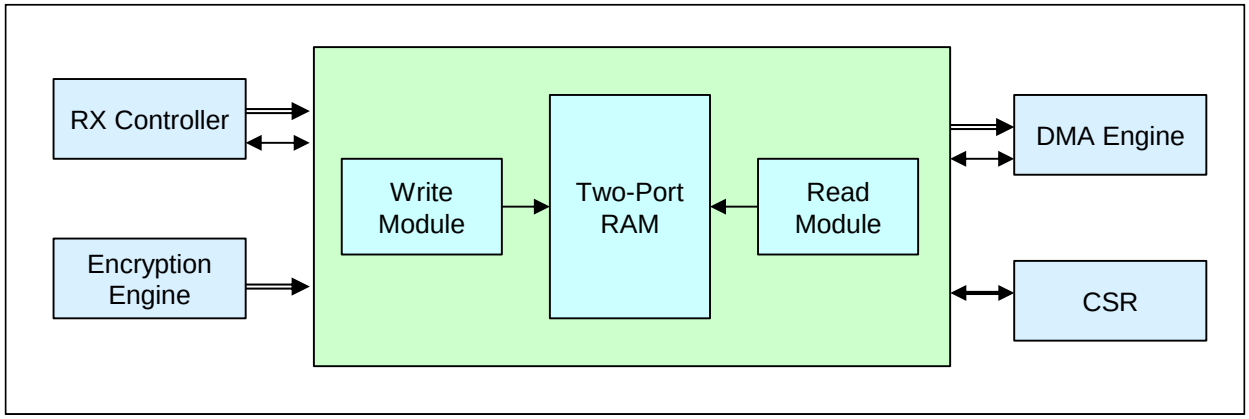


Figure 19: RX FIFO Block Diagram

RX FIFO as well as write and read pointers can be reset by *macErrRecCntrlReg.rxFIFOReset* in case of error recovery.

8.2 Two-Port RAM

It is a two-port RAM of 256 bytes with one read port and one write port. The write happens in *macPl2Clk* domain while read happens in *macCore2Clk* domain.

The Two-Port hard macro connections are available at the Top level to ease the integration.

8.3 Write Module

Write Module generates write pointer for RX FIFO in mac2Clk domain and the different flags for the RX Controller. The data coming from RX Controller is written to the location pointed by write pointer. The write pointer is incremented for every write and points to next location. When RX FIFO is full, write pointer is not incremented further. The write interface to the RX Controller is 32-bits + 4-bits for the Tags. (refer section *8.5, RX FIFO Tags*).

The RX FIFO provides information about its content using following flags:

rxFifoAlmostFull is generated to the RX Controller when the RX FIFO is almost full and can accept only one new write access. The RX Controller is allowed to write a last word with Tags update when this signal is high.

rxFifoOverrun is generated to Interrupt controller for interrupt generation when a write request is received and FIFO is full.

8.4 Read Module

Read Pointer Module generates read pointer for RX FIFO in *macH2Clk* domain and the different flags for the DMA Engine. The read pointer is incremented with every read request from Receive DMA channel. When the RX FIFO is empty, the read pointer is not incremented further. The Receive DMA channel starts reading data from RX FIFO when *dmaThresholdReg.rxFIFOThreshold* number of quadlets are present in RX FIFO. The threshold is ignored if the number

of quadlets remaining to be moved into the current buffer of DMA is smaller than the threshold. The Receive DMA channel stops reading data from RX FIFO when the RX FIFO status shows empty.

The RX FIFO provides information about its content using following flags:

rxFifoAboveThreshold is generated to the DMA Engine when there is more data present in RX FIFO than the threshold defined in the register dmaThresholdReg.rxFIFOThreshold. This allows the DMA Engine to start to initiate burst access.

rxFifoEmpty is generated to the DMA Engine when there is no data present in RX FIFO. If this signal is low, DMA Engine can pop a new word from the RX FIFO else it is not allowed to request a new data.

rxFifoUnderrun is generated when the DMA Engine requests a read access when the RX FIFO is empty.

8.5 RX FIFO Tags

Along with the 32bits data, the RX FIFO carries 4 additional flag bits defined in *Table 5: RX FIFO Tags* which delimits the MPDUs and indicates if frame have to be saved or discarded.

Flag description	b3-b0
Save the received frame	0000
Start of Header	0001
Reserved	0010-0100
End of Payload	0101
Intermediate header or payload	1100
MPDU Length	1010
Reserved	1011-1101
Indicate a Trigger Frame	1110
Discard the received frame	1111

Table 5: RX FIFO Tags

9 Doze Controller

9.1 Functional Description

The Doze module (henceforth referred to as Doze) is activated when *stateCntlrReg.currentState* is set to DOZE. It is responsible for enabling or disabling clock gating for MAC HW. This ensures minimum power consumption. It operates in *macLPClk* and interfaces to MAC Controller, RX Controller, CSReg and Timer Block as shown in *Figure 20: Doze Module Block Diagram*.

Two power-save modes are supported:

Auto wake up mode: In auto wake up mode, the MAC HW is turned on at Listen Intervals or before DTIM beacons for Legacy Power Save.

Non auto wake up mode: In non auto wake up mode, SW has to explicitly move HW out of DOZE via register*stateCntrlReg.currentState*.

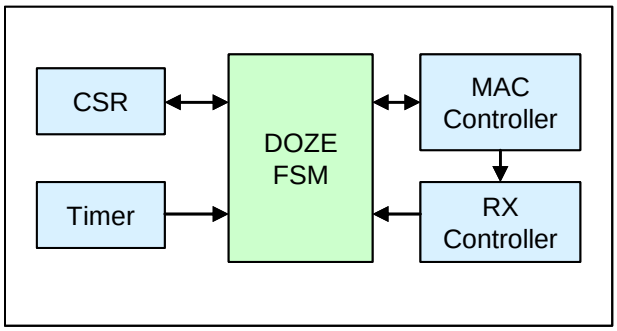


Figure 20: Doze Module Block Diagram

9.2 Doze Module State machine

Figure 21: Doze Module State Machine below gives high level overview of Doze state machine. *Table 10: MAC Core state machine states* and *Table 11: MAC core state machine state transition conditions* below explain different activities in each state and state transition conditions.

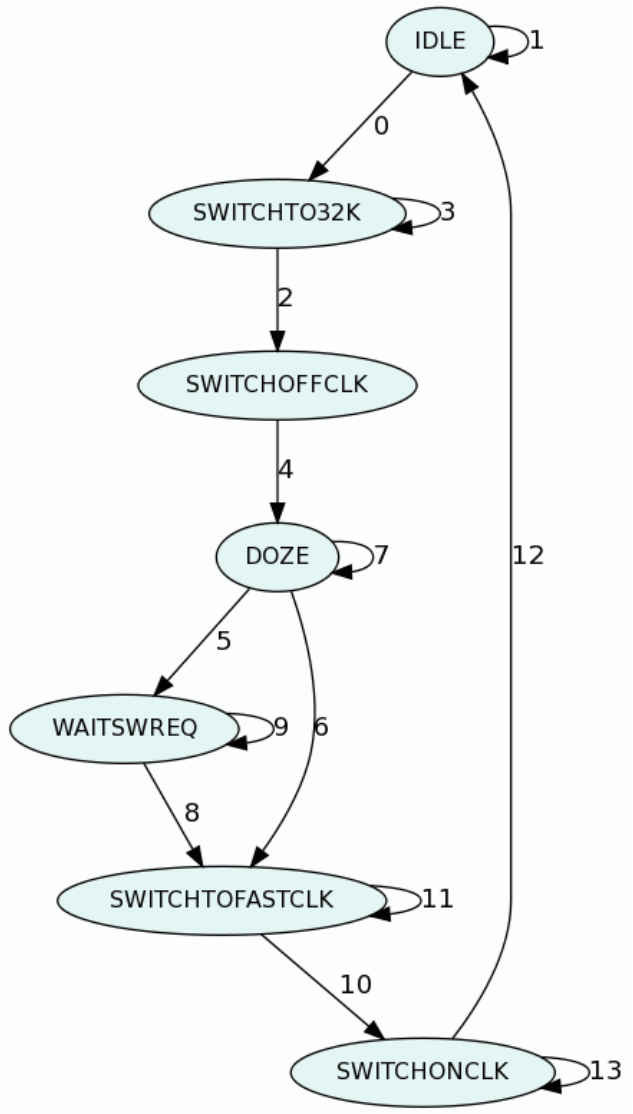


Figure 21: Doze Module State Machine

State Name	State Description
IDLE	In IDLE state, the state machine waits for a moveToDoze request from the SW
SWITCHTO32K	In SWITCHTO32K state, the state machine waits for counterDone_p indicating that the time required to switch to 32kHz is completed (`SWITCHTO32K_DURATION in fast clock cycle)
SWITCHOFFCLK	In SWITCHOFFCLK state, the state machine switches the Primar Clocks Off
DOZE	In DOZE state, the state machine waits for a Wake Up indication either from the Timer Unit or from the SW
WAITSWREQ	In WAITSWREQ state, the state machine waits until the SW moves the state to IDLE.
SWITCHTOFASTCLK	In SWITCHTOFASTCLK state, the state machine waits for the counterDone_p indicating that the time required to switch to fast clocks is completed (`SWITCHTOFASTCLK_DURATION in LP clock cycle)
SWITCHONCLK	In SWITCHONCLK state, the state machine switches the Primary Clocks On

Table 6: DOZE Module FSM States

Transition No.	State Description
0	When moveToDoze_p is received, the state machine goes to SWITCHOFFCLK state.
1	While moveToDoze_p is not received, the state machine stays in IDLE state.
2	When counterDone_p is received, the state machine goes to SWITCHOFFCLK state.
3	While counterDone_p is not received, the state machine stays in SWITCHTO32K state.
4	One clock cycle after, the state machine stays in SWITCHOFFCLK state.
5	When wakeUp is received and wakeUpSW is set, the state machine goes to WAITSWREQ state.
6	When wakeUp is received and wakeUpSW is not set, the state machine goes to SWITCHTOFASTCLK state.
7	While wakeUp is not received and SW changed nextState to IDLE, the state machine stays in DOZE state.
8	When SW changes nextState to IDLE, the state machine goes to SWITCHTOFASTCLK state.
9	While wakeUp is not received and SW changed nextState to IDLE, the state machine stays in WAITSWREQ state.
10	When counterDone_p is received, the state machine goes to SWITCHONCLK state.
11	While counterDone_p is not received, the state machine stays in SWITCHTOFASTCLK state.
12	When fastClkReady is received, the state machine moves to IDLE state.
13	While fastClkReady is not set, the state machine stays in SWITCHONCLK state.

Table 7: DOZE Module FSM State transition conditions

10 MIB Controller

10.1 Overview

The MIB Controller is in charge to collect; both from HW and SW; all the management information. The data coming from the HW is exposed on the block interface. The MIB Controller drives a RAM block. The memory store is triggered by the MAC Controller.

The following block diagram describes MIB controller:

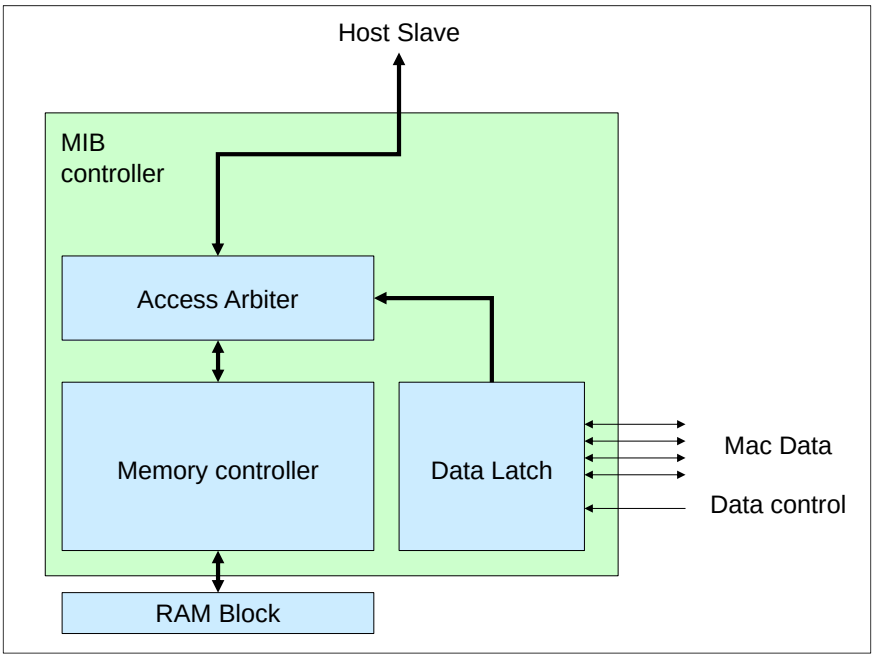


Figure 22: MIB Controller block diagram

10.2 Hardware operations

The MIB Controller is triggered by the MAC Controller. All the MIB fields are exposed on the MIB controller interface. When the trigger happens the MIB Controller will latch all the fields. It will allow the MAC controller to release the fields immediately after the trigger. Then the MIB controller will start to update the MIB information held in the RAM.

In order to avoid the full memory parsing and optimize the processing time, the MIB Controller implements a mechanism looking for the fields to update and will provide the address to the memory controller. This is described in *Figure 23: Event detection diagram*.

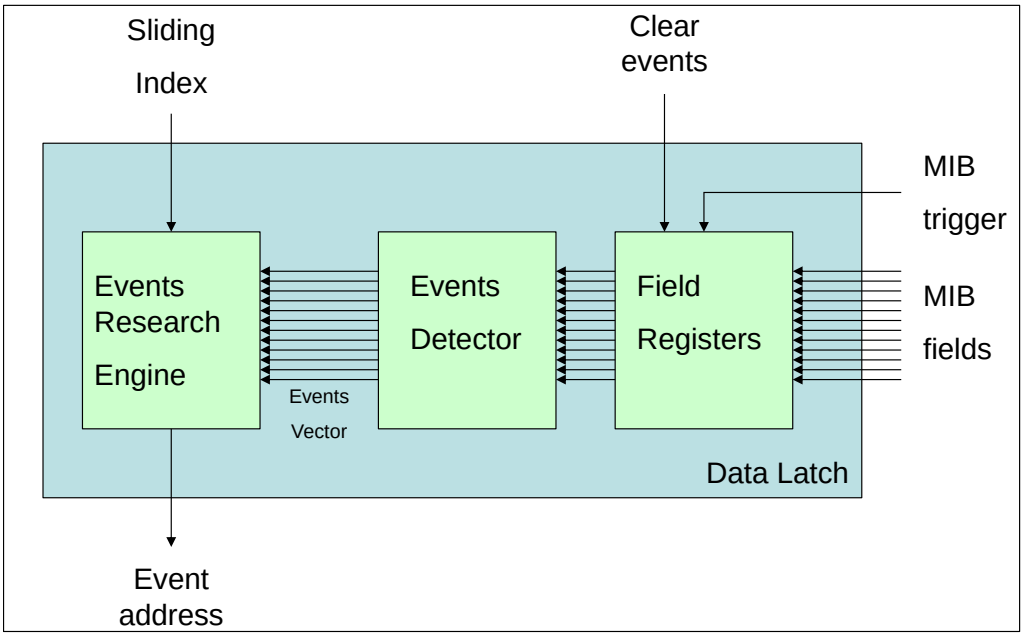


Figure 23: Event detection diagram

The event research engine will return the address of the next field to be updated. When the address is updated, the memory controller will increase the sliding index parameter to the last found address plus one. This operation will be repeated until all the fields have been parsed. The event search engine has a sizable event vector inputs. The size will be defined to match both synthesis constraints and performance needs.

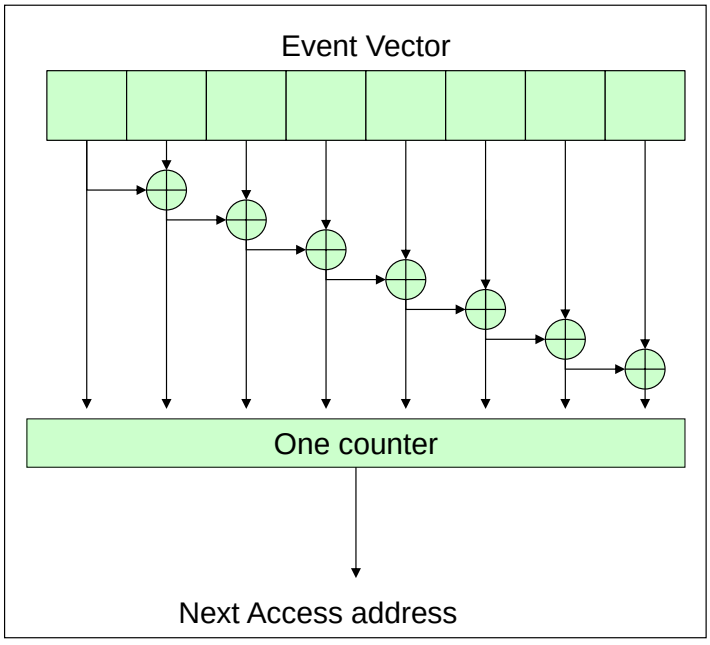


Figure 24: Address search mechanism

The MIB's fields indexed by the TID can be updated only one by one at each trigger. The event detection will be done on the field value. The address will be calculated depending on the TID index also exposed on the interface.

10.3 Software operations

The software has a full access to the MIB's RAM block through the MAC's Platform slave interface. The MIB controller implements a dedicated interface to the Platform slave block for this purpose.

The software can set a field or reset the complete MIB table using the MAC Controller 1 register. In the case a MIB table request is issued will and update is ongoing all the latched fields are cleared and the update is canceled.

The software has to wait the end reset table procedure before accessing the MIB RAM. This is ensured by check the *mibTableReset* bit (used to reset the MIB table) is in its initial state.

10.4 State Machines

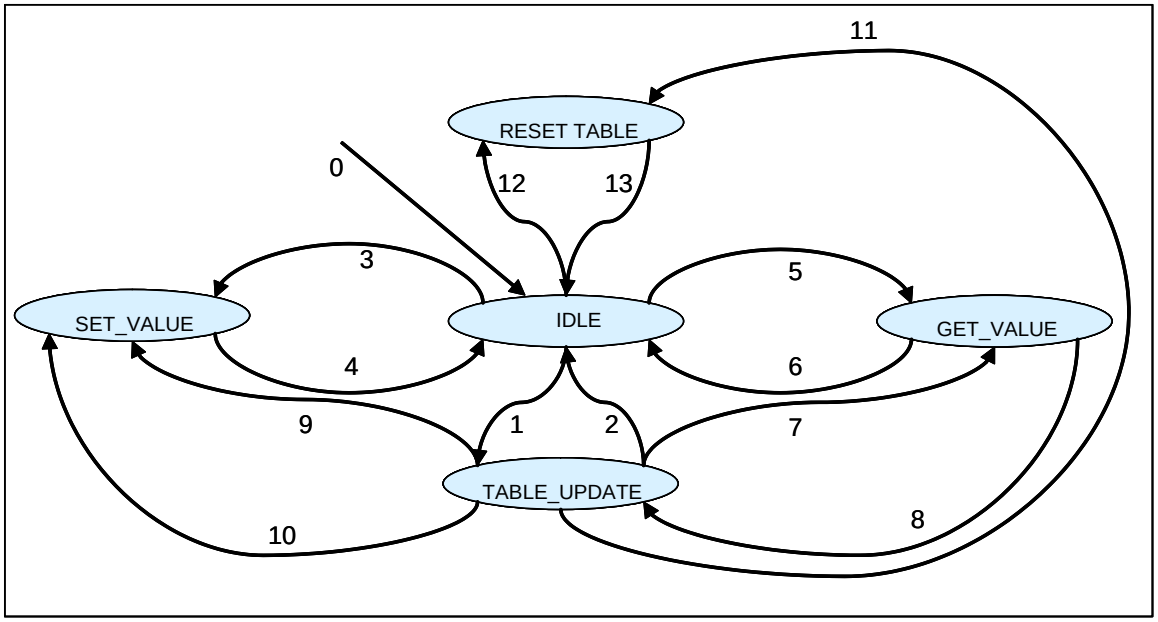


Figure 25: MIB controller State Machine

State Name	State Description
IDLE	After reset, FSM enters this state. In this state, all locations are programmed with default values. This is all 1's for the MAC Address field, all 0's for the Key field.
TABLE_UPDATE	In this state FSM updates the table fields for which an event has been detected.
GET_VALUE	FSM reads the value stored in the RAM and will return it to the platform slave block
SET_VALUE	FSM will take the value provided by platform slave and will write it into the RAM
RESET_TABLE	FSM will write the value zero into all the table fields

Table 8: Key Search Engine FSM States

Transition No.	State Description
0	Power reset or Soft reset.
1	Trigger from the MAC controller.
2	Update operation is done

Transition No.	State Description
3	Read trigger from the Platform slave
4	Read done
5	Write trigger from the Platform slave
6	Write done
7	Read trigger from the Platform slave.
8	Read done
9	Write trigger from the Platform slave
10	Write done
11	Trigger from the CSR MAC Control 1 register
12	Trigger from the CSR MAC Control 1 register
13	Table reset done

Table 9: MIB Controller State machine transitions conditions

11 TxTime Calculator

11.1 Overview

Both SW and HW need to compute the duration of a frame based on the length and rate selected and the TxTime Calculator provides this service. This unit can be triggered by both HW (MAC Controller) or SW via the *timeOnAir* registers.

It is also in charge of computing the Short GI NSYM Disambiguation bit of VHT frame.

11.2 Functional Description

11.2.1 Triggered by HW

The MAC Controller needs to compute the duration of a frame. In this case, it provides to the TxTime Calculator, some parameters like the length, the rate, the modulation type (non-HT, non-HT-DUP-OFDM, HT-MF,HT-GF or VHT) and the preamble type. When triggered by the HW, the TxTime Calculator launches the processing.

If a SW triggered computing was on-going, the HW request is put on hold until the completion of the SW computing and then, the HW request will be processed

Once the HW requested processing is completed, a flag is provided to indicate that the *hwDuration* output is correct. Note that this output will stay until the next HW requested computation. The SW requested computations do not impact this value.

11.2.2 Triggered by SW

The software can request the TxTime Calculator to calculate the time on air of a frame. When SW sets the *timeOnAirParamReg.startComputation* bit, the duration is computed based on the *timeOnAir* parameters using the following formulas. When the computing is completed, the TxTime Calculator updates the *timeOnAirValueReg* with the computed duration and the *timeOnAirValid* indication.

Note that the software can request a *timeOnAir* computing while a duration requested by HW is on-going. In this case, the software request is postponed until the end of the on-going computation.

12 MAC Control Logic

12.1 MAC Controller

12.1.1 Overview

The MAC Controller is the core block of RW-WLAN-nX MAC HW. The MAC Controller block trigs and gets triggered from all major blocks in MAC HW. All major protocol decisions are taken by MAC Controller block. The MAC Controller operates in *macCoreClk* and interfaces to TX Controller, RX Controller, DMA Engine, TX Parameters Cache (Header descriptor and Policy Table), CSR, Timers, NAV, Backoff, Block ACK Controller and MAC-PHY Interface blocks as shown in *Figure 26: MAC Controller block diagram*.

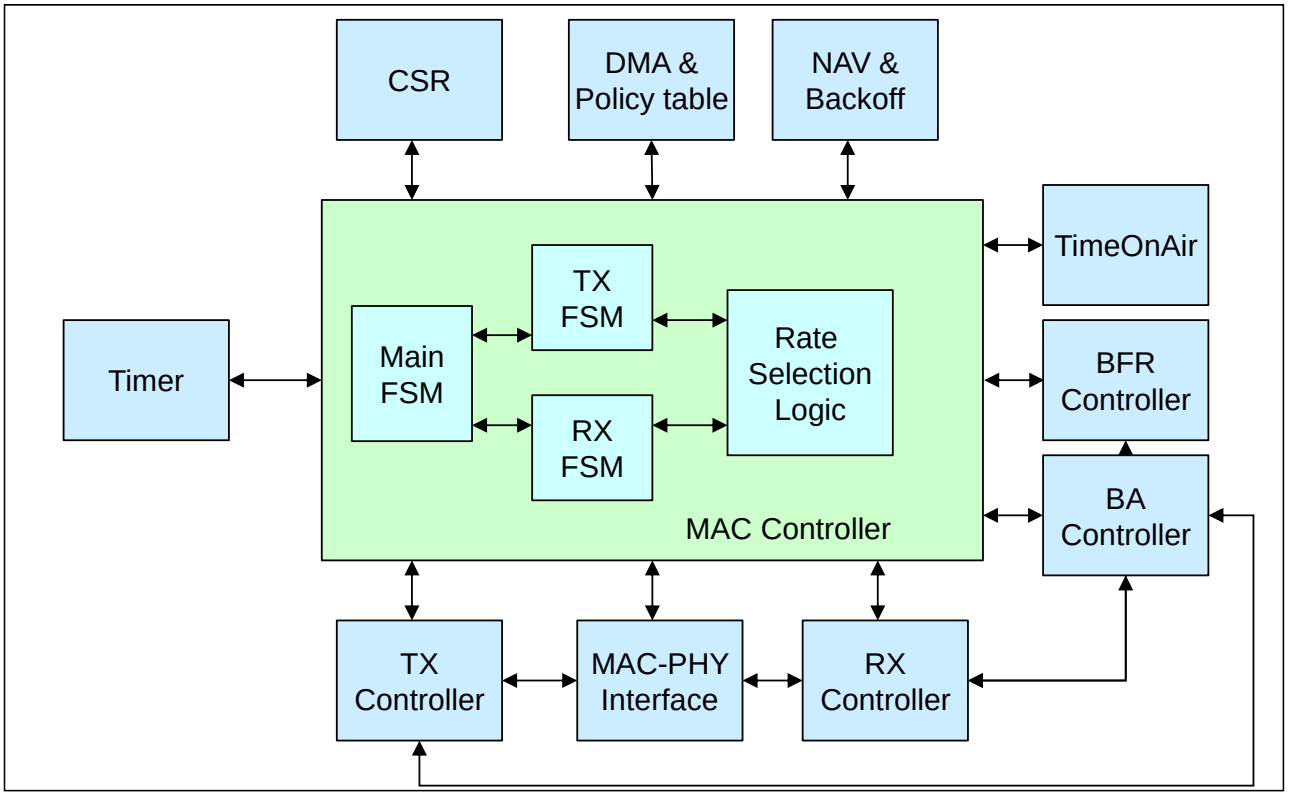


Figure 26: MAC Controller block diagram

12.1.2 Functional Description

Following are main functions of MAC Controller.

- Triggering of Transmit controller, Receive controller, MAC-PHY interface
- Triggering Policy Table lookup, Block ACK response preparation
- Decisions on aggregation, Beacon and Control frame transmission as per protocol
- TXOP acquisition
- Handling errors in frame transmission and reception
- Updating MIB

All the above functions are performed by the MAC Controller state machines and associated control logic. Depending on contents of CSR, triggers from RX Controller, TX Controller, Timer, NAV block, Backoff engine and DMA Engine, various transmit and receive decisions are taken. The sections below explain all these functions in detail.

12.1.2.1 ACTIVE state operations

MAC core enters ACTIVE state in three conditions.

1. Directly from IDLE
2. From DOZE state

In second scenario, software programs MAC core to ACTIVE by setting *stateCntlrReg.currentState* register to ACTIVE. Note that this transition is possible only from IDLE and the software has to move first the state to IDLE if it decides to move from DOZE to ACTIVE.

The DOZE Controller can also request the MAC Controller to move from DOZE to ACTIVE state.

In ACTIVE state, MAC core transmits and receives frames as explained in sections below.

12.1.2.2 Triggering Transmit state machine

Transmit controller is triggered under two conditions.

- For transmission of response frames
- For transmission of frames after winning contention.

12.1.2.2.1 Response frame transmission

All response frame transmit decisions are taken by MAC Receive state machine. If any response frames like ACK, CTS, Dual-CTS or Compressed Block ACK response frame to be transmitted, the MAC Receive state machine triggers TX Controller to prepare the frame and transmit.

In case of A-MPDU reception with Implicit Block ACK policy, even if a single MPDU is received successfully while remaining MPDU's fail the FCS check, the MAC Receive state machine triggers Block ACK response transmission at the end of A-MPDU reception. The end of A-MPDU reception is indicated by A-MPDU Deaggregator block.

12.1.2.2.2 Transmission of frames by MAC

In WAIT_TXRX_TRIG state, if MAC Core wins contention, the MAC Core state machine moves to ACTIVE_TX state triggering DMA and MAC Transmit state machine. The MAC Transmit state machine is responsible for all transmit decisions, triggering of TX Controller, triggering of DMA engine to fetch new frames and status update of transmitted frames.

The MAC Transmit state machine receives frame header information and Policy table information from DMA controller and takes transmit decisions based on frame type, frame length, rate of transmission, retry limit, TXOP remaining etc. Following are transmitting decisions taken by MAC Transmit state machine.

- If *navProtFrmEx* set in MAC Control information indicates self-CTS to be transmitted followed by Data, MAC Transmit state machine first transmits self-CTS followed by Data frame exchange.
- If *navProtFrmEx* set in MAC Control information indicates RTS-CTS frame exchange to intended Data receiver, MAC Transmit state machine initiates RTS-CTS frame sequence followed by Data frame exchange.
- If DMA indicates the frame to be transmitted is CF-End and it is start of TXOP then MAC Transmit state machine discards this CF-End.
- If DMA indicates the frame to be transmitted is CF-End during continuation of TXOP and DMA indicates next frame is available for transmission then MAC Transmit state machine discards the CF-End.
- If the remaining TXOP which was not obtained using Dual-CTS protection is not sufficient to transmit next frame in the queue, MAC HW curtails the TXOP by transmitting CF-End frame.

- As an AP, if *edcaCntrlReg.sendCFEnd* flag bit is set, then MAC HW checks if the remaining TXOP is sufficient to transmit CF-End at the highest BSS basic rate defined in the register *ratesReg.bssBasicRateSet*. If the CF-End duration is less than remaining TXOP, CF-End frame is transmitted.
- As a STA, if *edcaCntrlReg.sendCFEnd* flag bit is set, then MAC HW checks if the remaining TXOP is sufficient to transmit two CF-End frames at the highest BSS basic rate defined in the register *ratesReg.bssBasicRateSet* + SIFS. If the total duration is less than remaining TXOP, CF-End frame is transmitted.
- In both the above cases if remaining TXOP is more than the CF-End transmission duration, CF-End is not transmitted.
- As an AP/STA, if *edcaCntrlReg.sendCFEnd* flag bit is not set, then MAC HW waits for SW to set *sendCFEndNow* bit or chain a CF-End frame in DMA queue. Once this flag is set then the duration of CF-End is compared with remaining TXOP and transmit decision is taken as previously described.

- If the remaining TXOP which was obtained using Dual-CTS protection is not sufficient to transmit next frame in the queue, MAC HW curtails the TXOP by transmitting CF-End frame.
  - As an AP, if *edcaCntrlReg.sendCFEnd* flag bit is set, then MAC HW checks if the remaining TXOP is sufficient to transmit CF-End. If the *stbcCntrlReg.cfEndSTBCDur* value is less than remaining TXOP, CF-End frame is transmitted.
  - As a STA, if *edcaCntrlReg.sendCFEnd* flag bit is set, then MAC HW checks if the remaining TXOP is sufficient to transmit a CF-End frame at the highest BSS basic rate defined in the register *ratesReg.bssBasicRateSet* + SIFS + *stbcCntrlReg.cfEndSTBCDur.* If the total duration is less than remaining TXOP, CF-End frame is transmitted.
  - In both the above cases if remaining TXOP is more than the total duration, CF-End is not transmitted.
  - As an AP/STA, if *edcaCntrlReg.sendCFEnd* flag bit is not set, then MAC HW waits for SW to set *sendCFEndNow* bit or chain a CF-End frame in DMA queue. Once this flag is set then the duration of CF-End is compared with remaining TXOP and transmit decision is taken.

  If DMA indicates no more frame are queued for this Access category and medium is protected with NAV and *edcaCntrlReg.sendCFEnd* bit is set then MAC Transmit state machine triggers CF-End transmission to release the medium.

  If DMA indicates no more frame are queued for this Access category and medium is protected with NAV and *edcaCntrlReg.sendCFEnd* bit is not set then MAC Transmit state machine waits for SW to set *edcaCntrlReg.sendCFEndNow* or a CF-End is queued. Once the *edcaCntrlReg.sendCFEndNow* bit is set, MAC Transmit state machine triggers CF-End frame transmission if TXOP is sufficient. As a STA, Transmit state machine checks if TXOP is sufficient for retransmission of CF-End by AP also.

  If a frame is retried at lower rate and remaining TXOP is not sufficient to fit the frame and ACK, MAC Transmit state machine continues transmission of the frame.

  If a frame transmission fails, the MAC Transmit state machine updates the descriptor and continues with retransmission till retry limit reaches.

  If implicit Block ACK is used by software, software additionally queues a Block ACK request at the end of AMPDU. MAC HW will transmit this Block ACK request frame only if the recipient device doesn't respond with Block ACK response at the end of AMPDU transmission. If the recipient device responds with Block ACK, the MAC Transmit state machine ignores the Block ACK request and proceeds for next frame transmission.

After winning contention and if the DMA channel is not ACTIVE and the SW has not requested for control frame transmission, the HW does nothing and waits for either condition to be true before proceeding. Subsequent transmission in either condition occurs at the next Slot boundary. This is post-backoff and is not part of the standard. However, the PHY is not put out of receive mode until either condition is true, Hence, if a receive operation starts in the meanwhile, the MAC is ready to accept the frame. In this case, the post-backoff is cancelled and the Backoff block will load a new random number and perform backoff again.

The inter frame spacing is controlled by MAC Transmit state machine. The MAC Controller TX state machine trigs the Tx Controller as soon it decides that a transmission is required. However, based on the SIFS or SLOT boundaries information coming from Timers Unit, the state machine waits for the right boundary and indicates to the TX Controller to start the transmission at the MAC-PHY IF. So, the TX Controller can launch some processing before the beginning of the transmission at the right slot boundary.

If TXOP programmed for any AC that wins contention is 0, then MAC Transmit state machine ensures only one MSDU or one A-MPDU is transmitted.

12.1.2.3 TXOP Decision

Before requesting a transition, the MAC Controller checks if the frame exchange fits into the TXOP.

In case of 40MHz,80MHz or 160MHz TXOP, refer section *12.1.2.16, 40MHz-80MHz-160MHz PPDU Transmission*.

12.1.2.3.1 If TXOP limit = 0

When TXOP Limit is equal to 0, only the atomic frame exchange is covered by the Long NAV. In this case, the TXOP duration has to be computed (refer section *12.1.2.11, Duration update & Long NAV*).

When starting a frame exchange, the MAC Controller checks if the frame exchange duration does not cross a quiet interval. For that, it checks the remaining time before the next Quiet Interval (*impQI* signal coming from the Timers Unit). If the duration is bigger than *impQI*, then frame exchange is postponed after the Quiet interval.

12.1.2.3.2 If TXOP limit != 0

When win contention on an AC which has non null TXOP Limit, the TXOP duration is set with the TXOP Limit defined in *edcaACXReg* registers. In this case, the duration is the remaining TXOP at the end of the transmitted frame (refer section *12.1.2.11, Duration update & Long NAV*).

When acquiring a TXOP, the MAC Controller checks if the TXOP does not cross a quiet interval. For that, it checks the remaining time before the next Quiet Interval (*impQI* signal coming from the Timers Unit). If the TXOP is bigger than *impQI*, then the TXOP is limited to *impQI*.

12.1.2.3.3 TXOP termination

When leaving the UPDATE_STATUS State, the MAC Controller TX state machine checks if the remaining TXOP is bigger than zero and the DMA Channel is HALTED indicated no more MPDUs are queued. In this case, the MAC Controller TX state machine computes the CF_END transmission duration

If the CF_END transmission duration does not fit into the remaining TXOP, nothing happen.

If the CF_END transmission duration does not fit into the remaining TXOP, the CF-END is transmitted as described below.

12.1.2.3.3.1 CF-END Duration Computing

If the TXOP was not acquired using Dual-CTS protection mode, only one CF-END is transmitted following the rate selection defined in *12.1.4.4 Control frame (CF-END) prepared by hardware.*

If the TXOP was not acquired using Dual-CTS protection mode, then

If the STA is an AP (*macControl1Reg.ap* = 1'b1), two CF-END are transmitted (one STBC and one non-STBC) and the duration comes from *stbcCntlReg.cfEndSTBCDur*[2].

If the STA is not an AP (*macControl1Reg.ap* = 1'b0), three CF-END frames are transmitted (one from the STA and two by the AP) and the duration is the duration of one CF-END frame at the rate defined in *12.1.4.4 Control frame (CF-END) prepared by hardware* + SIFS + *stbcCntlReg.cfEndSTBCDur*.

12.1.2.3.3.2 CF-END transmission

If the TXOP was not acquired using Dual-CTS protection mode, the MAC Controller trigs the TX Controller to transmit a CF-END at the rate defined in *12.1.4.4 Control frame (CF-END) prepared by hardware.*

If the TXOP was acquired using Dual-CTS protection mode, then

If the STA is not an AP (macControl1Reg.ap = 1'b1), the MAC Controller trigs the TX Controller to transmit a CF-END at the rate defined in *12.1.4.4 Control frame (CF-END) prepared by hardware*.

If the STA in an AP, two CF-END frames are transmitted (one in STBC rate and one in non-STBC rate) following the rates defined in *12.1.4.4 Control frame (CF-END) prepared by hardware* and sequence defined in *12.1.2.14 STBC support*

12.1.2.4 Triggering Receive state machine

In WAIT_TXRX_TRIG state, MAC Core switches on Receiver and waits for start of frame reception. At the start of the frame reception, indicated by the signal *cca* coming from the MAC-PHY IF, MAC Core state machine moves to ACTIVE_RX state and waits till frame reception and response handling is completed. The MAC Receive state machine takes all receive and response decisions.

Irrespective of the frame is passed to upper layer or discarded, the MAC Receive state machine triggers response frame transmission as per 802.11 protocol.

- If any Broadcast or Multi cast frame is received, no response frame is transmitted.
- If any unicast frame for which RA doesn't match MAC address, no response is transmitted.
- If any RTS frame is received with RA matching MAC address, MAC Receive state machine triggers TX Controller to transmit CTS frame if channel is free.
- If any unicast Data or Management frame is received with RA matching MAC address, MAC Receive state machine triggers TX Controller to transmit ACK frame.
- If any unexpected CTS or ACK frame is received with RA matching MAC address, MAC Receive state machine discards these frames.
- If any PS-Poll frame is received with RA matching MAC address, MAC Receive state machine triggers TX Controller to transmit ACK frame.
- If any 11e Block ACK Request or Response frame is received with RA matching MAC address, MAC Receive state machine triggers transmit controller to transmit ACK frame.
- If any compressed Block ACK frame is received with RA matching MAC address, MAC Receive state machine accepts the frame and doesn't transmit any response.
- If any compressed Block ACK Request frame is received with RA matching MAC address, MAC Receive state machine triggers TX Controller and Block ACK Controller state machines for transmission of Compressed Block ACK frame.

[2] Note that *cfEndSTBCDur* contains the duration of the two CF-END frames + SIFS.

- If any Control frame is received with RA not matching MAC address, MAC Receive state machine discards the frame.

For all response frames, the rate of transmission is decided based on received frame rate and basic rate set in the BSS.

12.1.2.5 Triggering MAC-PHY interface

The MAC-PHY interface is triggered for reception by default. Only for transmission of response frames or any other frame transmission is scheduled from software MAC-PHY interface is switched to Transmit mode. For any frame transmission even the response frames, the MAC Transmit state machine trigger MAC-PHY interface.

12.1.2.6 Triggering DMA

The MAC Core state machine triggers the DMA with Access category that won contention for transmission. The MAC Transmit state machine takes control of frame transmission and status update. At end of every frame transmission status is updated to the Header descriptor. If frame is transmitted successfully, the MAC Transmit state machine indicates success to the DMA engine. If the frame transmission fails, the MAC Transmit state machine triggers DMA to get same frame for transmission.

In receive direction, at the end of frame reception MAC Receive state machine triggers DMA to update status.

12.1.2.7 AMPDU transmission

The DMA engine fetches A-MPDU Transmit Header descriptor and passes A-MPDU transmission parameters to TX Parameters Cache module. These parameters are valid for entire A-MPDU. Based on these parameters, the MAC Transmit state machine takes decisions for protection and initiates frame exchange sequence. After each MPDU, the TX Controller creates as many Blank delimiters as indicated by the SW in the *Transmit Header descriptor*. After the transmission of the A-MPDU is complete, the MAC Transmit state machine waits for the BA response frame if the A-MPDU indicated that it was expected (*expectedAck* bit of the TX Header Descriptor equals to 2'b10 or 2'b11).

If the BA is received successfully, the MAC Transmit state machine updates the status for the entire aggregate in the A-MPDU transmit header descriptor as successful, and discards the explicit BAR frame that is attached next.

If the BA is not received successfully, the MAC Transmit state machine transmits the BAR frame which has been linked by the software to request for a BA.

After completion of A-MPDU transmission, if remaining TXOP is sufficient for transmission of next frame, the MAC Transmit state machine continues with transmission.

12.1.2.8 Beacon transmission

As an Access point or STA in IBSS at every TBTT, MAC Core state machine triggers MAC Transmit state machine to transmit a Beacon frame and all the frames linked on the Beacon after SIFS.

As an STA in IBSS at every TBTT, MAC Core state machine waits for Backoff and triggers MAC Transmit state machine to transmit a Beacon frame and all the frames linked on the Beacon after SIFS.

As an STA if a Beacon is received then Beacon transmission is cancelled. Once *backoffDone_p* trigger is received and Beacon transmission is pending, the MAC Core state machine triggers DMA Beacon channel to fetch Beacon frame.

As an AP, at every TBTT, MAC Core state machine triggers MAC Transmit state machine to transmit a Beacon frame.

12.1.2.9 Buffered BC/MC frame transmission

Before the DTIM, an interrupt is generated to the software. Software queues up BC/MC frames for transmission. Hardware transmits frames on the Beacon Queue. Thus, when the Beacon is transmitted, all the buffered BC/MC frames are transmitted after SIFS

12.1.2.10 Error handling in transmission

If any transmit (MAC or PHY) underrun is detected, the MAC Transmit state machine indicates underrun error to DMA and generates *txDMAError* Interrupt. Then, it moved to WAIT_TXRX_TRIG State.

If DMA reports any AHB bus error while fetching data from System memory, the MAC Transmit state machine aborts current frame transmission and generates *txDMAError* Interrupt. Then, it moved to IDLE State.

12.1.2.11 Duration update & Long NAV

If software passes Duration to be transmitted in the frame by setting *dontTouchDur* to value 1'b1 in the MAC Control information, MAC Transmit Controller doesn't update the Duration field transmitted.

If the *dontTouchDur* is set to value 1'b0, the HW has to compute and set the TXOP duration based on the following rules:

If TXOP Limit != 0, the TXOP duration computing steps are:

1. Get the duration from TXOP Limit
2. Check if the duration does not cross the Quiet Interval (*impQI* information from Timers Unit)
3. Set the TXOP Counter based on the small value between the TXOP Limit and Quiet Interval.

If TXOP Limit = 0, the TXOP duration computing steps are:

1. Check if protection is requested
2. Compute the duration of the RTS if required (*navProtFrmEx* = 3'b01X)
3. Compute the duration of the CTS if required (*navProtFrmEx* = 3'b01X or 3'b001)
4. Compute the duration of the DATA frame to be transmitted
5. Compute the duration of the ACK or BA if required (*expectedAck* field)
6. Sum all the computed duration and add also as many SIFS as required (see following tables)
7. Check if the computed duration does not cross the Quiet Interval (*impQI* information from Timers Unit)
8. Set the TXOP Counter based on computed duration.

If expectedAck=2'b00

navProtFrmEx	NAV duration
000	DATA
001	CTS + DATA + SIFS
01X	RTS + CTS + DATA + 2*SIFS
100	RTS + CTS + ctsSTBCDur + DATA + 3*SIFS

If expectedAck=2'b01

navProtFrmEx	NAV duration
000	DATA + ACK + SIFS
001	CTS + DATA + ACK + 2*SIFS
01X	RTS + CTS + DATA + ACK + 3*SIFS
100	RTS + CTS + ctsSTBCDur + DATA + ACK + 4*SIFS

If expectedAck=2'b1X

navProtFrmEx	NAV duration
000	DATA + BA + SIFS
001	CTS + DATA + BA + 2*SIFS
01X	RTS + CTS + DATA + BA + 3*SIFS
100	RTS + CTS + ctsSTBCDur + DATA + BA + 4*SIFS

Once the TXOP duration is set and if *dontTouchDur* is reset, the MAC Controller has to update the duration field (Long NAV). For that, when a frame is transmitted (protection or data), the duration provided to the TX Controller is the remaining TXOP minus the duration of the frame. Note that even if TXOP Limit is 0, the TXOP Counter has been set with the frame duration.

If Long NAV protection mechanism is used to protect entire TXOP and no frames are available to transmit, the TXOP protection is truncated using CF-End frame transmission.

12.1.2.12 Legacy Rate and Length generation

The frames transmitted by MAC can be Non-HT frames, HT frames in Mixed mode and HT frames in Green Field mode or VHT frames.

When MAC is transmitting Non-HT frames (*formatModTx* and *formatModProtTx* equal to 2'b00 or 2'b01), MAC Transmit controller passes Length from the *frameLengthTx* field in Transmit DMA Header descriptor. The rate of the frame is provided by the Rate Management Unit based on the *mcsInde0x* of Transmit DMA Header descriptor for initial transmission. For retransmissions the rate is taken from *mcsIndex1*, *mcsIndex2* and *mcsIndex3* fields from Policy Table following the rates selection mechanism described in *12.1.4 Rate Management Unit*. When transmitting the Non-HT frames, the HT parameters are not valid and any value driven on these fields is ignored.

When MAC is transmitting HT frames in Green field mode (*formatModTx* and *formatModProtTx* set to 2'b11), the Legacy parameters are not valid and any value driven on these fields is ignored.

When MAC is transmitting HT frames in Mixed mode (*formatModTx* and *formatModProtTx* set to 2'b10), both Legacy and HT parameters are valid. Software passes the HT parameters and actual length of the frame in Transmit DMA Header descriptor. MAC Transmit state machine calculates the Legacy length based on the actual length and time taken for frame transmission at HT rate and equivalent legacy length. This length covers only the current frame not associated responses (refer section *11,* TxTime Calculator).

When MAC is transmitting VHT frames (*formatModTx* and *formatModProtTx* set to 3'b100), both Legacy and HT parameters are valid. Software passes the HT parameters and actual length of the frame in Transmit DMA Header descriptor. MAC Transmit state machine calculates the Legacy length based on the actual length and time taken for frame transmission at VHT rate and equivalent legacy length. This length covers only the current frame not associated responses (refer section *11,* TxTime Calculator).

12.1.2.13 L-SIG TXOP protection

Software indicates to Hardware to use L-SIG TXOP protection by setting *lstp* bit in MAC Control Information field of Transmit DMA Header.

When this bit is set, the MAC Transmit state machine calculates the L-Length from Duration of frame and any associated response at HT or VHT Rate with 6Mbps rate. The MAC Transmit state machine passes this information to MAC-PHY Interface.

In this case, the L-Length (*legLength* output of the MAC-PHY IF) is computed as following:

For all frames except RTS,

$$legLength = \frac{MACDuration + frameduration - 20}{6*8}$$

For the RTS which starts the TXOP, the MAC Duration covers the entire TXOP but the L-SIG duration covers only the RTS, SIFS and CTS. In this case, the *legLength* is

$$legLength = \frac{RTSDuration - 20 + SIFS + CTSDuration}{6*8}$$

The MAC Transmit state machine avoids transmission of CF-End to release TXOP when L-SIG protection is used.

MAC Receive state machine identifies L-SIG frames based on duration of the frame calculated from Legacy fields and Duration of frame from HT fields.

If MAC HW receives a frame with L-SIG protection addressed to it, the ACK is transmitted with L-SIG protection.

If MAC HW receives a frame with L-SIG protection and not addressed to it, the NAV is updated by the RX Controller with the duration computed based on legacy length.

12.1.2.14 STBC support

If MAC is part of BSS that supports STBC, software sets *stbcCntrlReg.dualCTSProt* bit

The following functions are performed in HW to support STBC operation:

As a STA, transmission of RTS at the start of TXOP. The type of transmission whether STBC or non-STBC is set by SW in the Transmit Header descriptor fields.

As an AP, responding with Dual CTS at STBC and non-STBC rates to an RTS. For this the Transmit State machine triggers the TX Controller twice to transmit 2 CTS frames. The indication whether the CTS is with STBC or non-STBC is passed through the TX Vector.

- If the RTS received was at STBC rate, the first CTS is transmitted at STBC rate and the second at non-STBC rate.
- If the RTS received was at non-STBC rate, the first CTS is transmitted at non-STBC rate and the second at STBC rate.

Similarly if there are no frames available with HW to transmit and the TXOP is protected with Long-NAV, then HW transmits the CF-End frame to curtail the TXOP.

- When configured as a STA the CF-End is transmitted at the STBC rate taking into account the time taken for the transmission of 2 CF-End frames and 3 SIFS durations by the AP.
- When it is an AP then to curtail the TXOP the AP transmits 2 CF-End frames separated by SIFS. One of them at the lowest STBC basic rate and the other a non-STBC frame at the lowest basic rate. The first CF-End frame will use the same encoding as the transmission used in the TXOP (as for CTS).

When configured as an AP, when CF-End is received, it transmits 2 CF-End frames separated by SIFS duration. One of them at the lowest STBC basic rate and the other a non-STBC frame at the lowest basic rate. The first CF-End frame will use the same encoding as the CF-End received.

For secondary Beacon transmission the HW maintains the timer for TBTT. Before the TBTT the HW indicates to the SW early enough to prepare the Beacon frame for transmission. At TBTT the HW transmits the Beacon after winning contention.

12.1.2.15 20-40-80-160 Coexistence

Functions for control and management of 20-40 Coexistence are handled by the SW. The minimal functions for this feature performed in HW are:

- *40MHz-80MHz-160MHz PPDU Transmission*
- *NAV Update*

12.1.2.16 40MHz-80MHz-160MHz PPDU Transmission

The identification of the data to be a 160MHz, 80MHz, 40MHz or 20MHz PPDU is known by reading the Phy Control Information field of the Transmit Header descriptor (*bwtx* & *bwProtTx* bits) or by reading the *txBWCntrlReg* register.

The right to transmit a 40MHz, 80MHz or 160MHz PPDU after establishment/association to a 20/40/80/160 MHz BSS is when the following condition exists: Contention for the primary channel is successful and in the secondary channel the medium should be idle for duration of PIFS.

To implement this feature, secondary channels CCA from the PHY is available to the Timer block. The Timer block generates flags once the CCA is idle for PIFS duration. When backoff is successful in the primary channel, the assertion of this flag indicates that a 40MHz, 80MHz or 160MHz TXOP is available.

Possible situations which might be encountered when sensing the channels to transmit a 40/80/160MHz PPDU along with the behavior of the HW is detailed below:

When the contention in the primary channel is successful and at the same time the flags indicating PIFS duration idle on the corresponding secondary channels are also set, then 40/80/160MHz TXOP is successfully obtained. In this case all the PPDU with a requested bandwidth equal or lower than the acquired TXOP can be transmitted successfully.

Unsuccessful at obtaining a 40/80/160MHz TXOP because the one of the corresponding secondary channels is not free when the contention for the medium on the primary channel is successful. In this case if it is a 40MHz frame then the number of times the frame is attempted to transmit at 40MHz is indicated by the *txBWCntrlReg.numTry40Acquisition* set by SW. (Same mechanism for 80 or 160MHz)

- If this value is zero, the HW will continue trying to obtain a 40/80/160MHz TXOP until the lifetime of the frame expires.
- If this value has a number, then the frame is tried to send at 40/80/160MHz as defined. Else it is sent at a lower bandwidth ensuring that the frame when transmitted will still fit in the TXOP.

It is possible that a 20MHz TXOP is obtained and a 40MHz PPDU is queued for transmission. In this case the *txBWCntrlReg.dropToLowerBW* register is checked.

- If this bit is set then the frame will be transmitted in 20MHz if its fits within the TXOP.
- If the bit is reset, the HW will release the TXOP by transmitting a CF-End frame. This is the case only if the frame is queued between 20MHz frames. If this is the first frame then CF-End is not transmitted. This allows other AC's to continue the backoff.

12.1.2.17 NAV Update

In case of 20/40/80 operation, NAV update happens only when the received frame is either a 20MHz MPDU in the primary channel or a 40/80MHz MPDU, and both the following conditions are satisfied: the FCS for the frame passes and the received frame is not destined for this STA.

12.1.2.18 Beamformee calibration procedure

During the beamforming calibration, the MAC Controller receives information from the beamforming Controller and from the register in order to prepare and trigs the Beamforming Report transmission.

The decision to transmit or not the beamforming report upon reception of NDPA/NDP or Beamforming Report Poll is taken by the Beamforming Controller which indicates to the MAC Controller the needs of BFR response using *needBfr_p*. The beamforming report frame length is computed and provided by the beamforming Controller and used for the duration computation. All the PHY parameters needs for the duration computation and transmission come from either the Beamforming Controller or from the beamformee register.

For more information about the calibration procedure, see *17 Beamforming Controller*.

12.1.2.19 MU-MIMO RX procedure

Upon reception of a correct MU-MIMO frame (indicated by *groupID* other than 0 or 63) addressed to the device, the MAC Controller checks the user position before triggering the transmission of the immediate response BA. The user position is given by the PHY using the *firstUser* bit of the rxVector1. If this bit is set indicating that the user position of the received frame was 0, the MAC Controller trigs the transmission of BA. Otherwise, the MAC Controller closes the reception event. In this last case, the BA will be transmitted after a correct reception of a BAR.

12.1.3 State machines

The MAC Controller is implemented in three state machines.

1. MAC Controller Master state machine
2. MAC Controller Transmit state machine
3. MAC Controller Receive state machine

The MAC Controller Master state machine is the core state machine that implements ACTIVE state functionalities of MAC. This state machine triggers MAC Transmit and MAC Receive state machines.

The MAC Transmit state machine is responsible for all transmit decisions at run time. This state machine takes care of A-MPDU aggregation, Control frame generation to support different protection mechanisms. This state machine also takes care of response frame reception.

The MAC Receive state machine is responsible for all receive decisions and transmission of responses.

12.1.3.1 MAC Controller Master state machine

The MAC Controller Master state machine is also called as Active state machine. The name is derived from the two major functions of MAC performed by this module. ACTIVE state operations

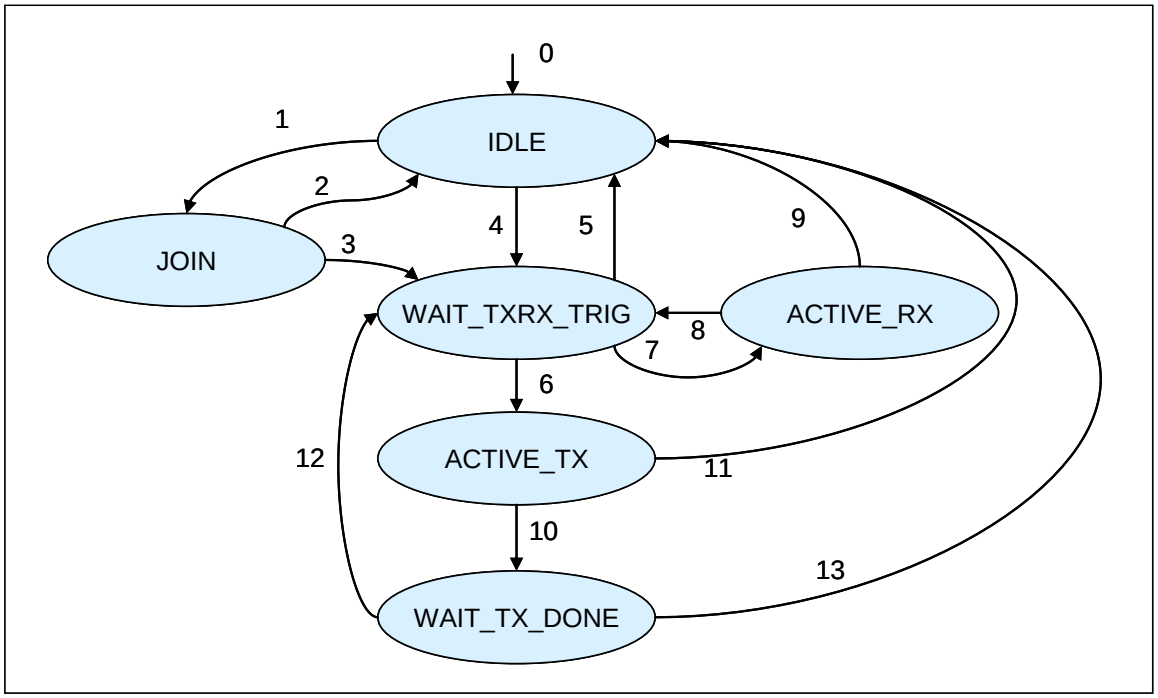


Figure 27: MAC Controller Core state machine

State Name	State Description
IDLE	The state machine starts up in this state after reset. It may also transition to this state under other conditions. In this state, the state machine waits for Software to program the *stateCntlrReg.currentState* register to ACTIVE states or till trigger is received from DOZE state machine to move out of IDLE. When entering to this state from DOZE or JOIN State, the *idleInterrupt* is generated.
WAIT_TXRX_TRIG	In this state, MAC Core state machine waits for *backOffDone_p* indication from Backoff block, a *tickEarlyTbtt_p* from the Timers Unit or CCA indication from MAC-PHY Interface.
ACTIVE_RX	In this state MAC Core waits till current reception and applicable response frame transmission is completed.
ACTIVE_TX	In this state, MAC Core state machine checks if a beacon transmission has been request and if it will not cross the TBTT using the *noBcnTxFlag*. It triggers MAC Transmit state machine and DMA engine and moves to WAIT_TX_DONE state.
WAIT_TX_DONE	In this state MAC Core waits till current transmission is completed which includes reception of appropriate response frames for the transmission like ACK, CTS, Block ACK, etc.

Table 10: MAC Core state machine states

Transition No.	State Description
0	Hard reset or Soft reset.
4	This transition occurs when *stateCntlrReg.currentState* register is set to ACTIVE.

Transition No.	State Description
5	This transition occurs when *stateCntlrReg.currentState* register is set to IDLE or DOZE state machine triggered MAC core state machine to move to IDLE state.
6	This transition occurs when *backoffDone_p* indication is received or if the remaining TXOP is != 0. As an AP or in IBSS, this transition can also occur when *tickEarlyTbtt_p* is received indicating the beacon transmission.
7	This transition occurs when CCA indication is received.
8	This transition occurs when all receive processing and response frame transmission is completed.
9	This transition occurs when software programs *stateCntlrReg.currentState* register to IDLE and current reception is completed.
10	This transition occurs when the received frame is QCF-Poll. In this case, the state machine moves to ACTIVE_TX state triggering frame transmission for granted TXOP.
11	This transition happens at next clock edge.
12	This transition occurs when all transmit processing and response frame reception is completed.
13	This transition occurs when software programs *stateCntlrReg.currentState* register to IDLE or if a beacon transmission will cross the TBTT.
14	This transition occurs when software programs *stateCntlrReg.currentState* register to IDLE and current transmission is completed.

Table 11: MAC core state machine state transition conditions

12.1.3.1.1 Timing diagrams

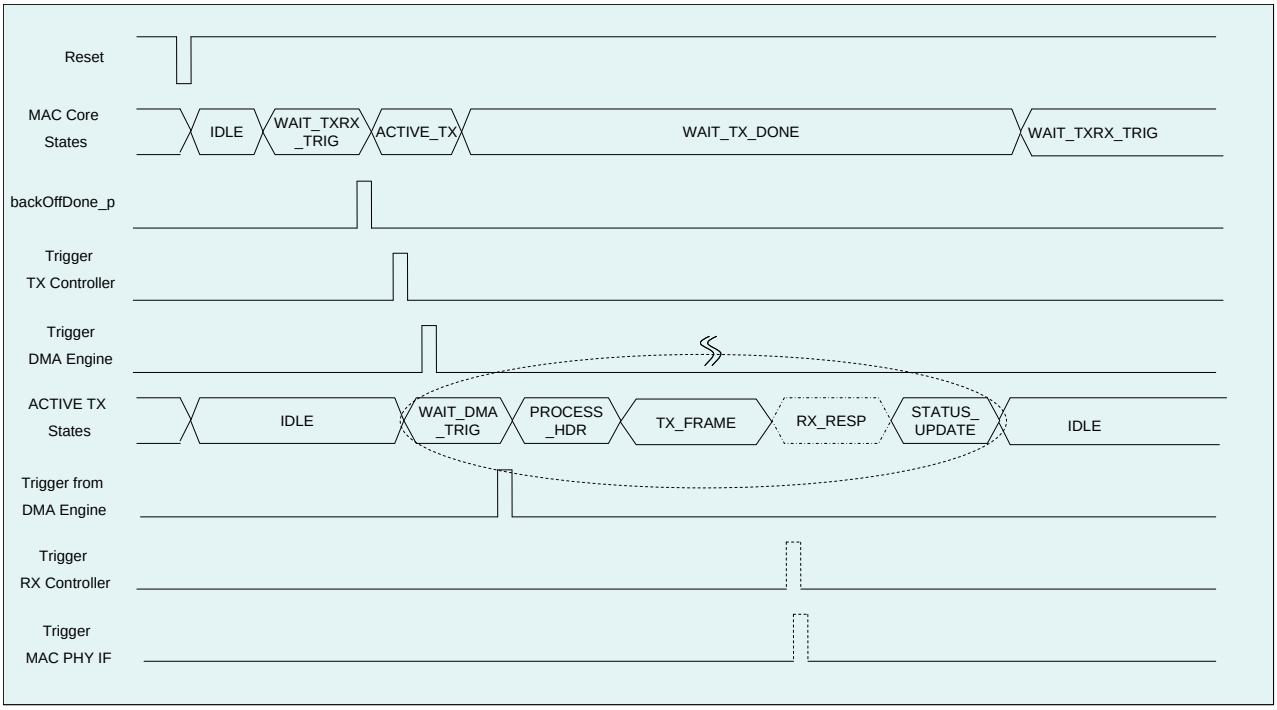


Figure 28: MAC Core state machine transmit timing diagram

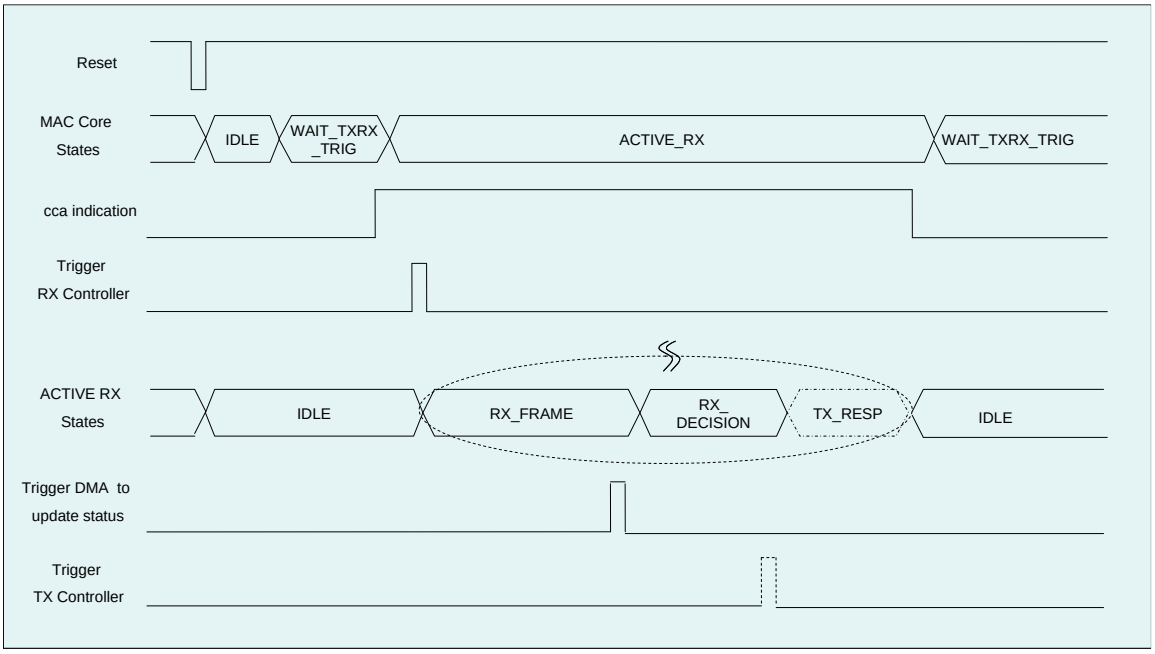


Figure 29: MAC Core state machine transmit timing diagram

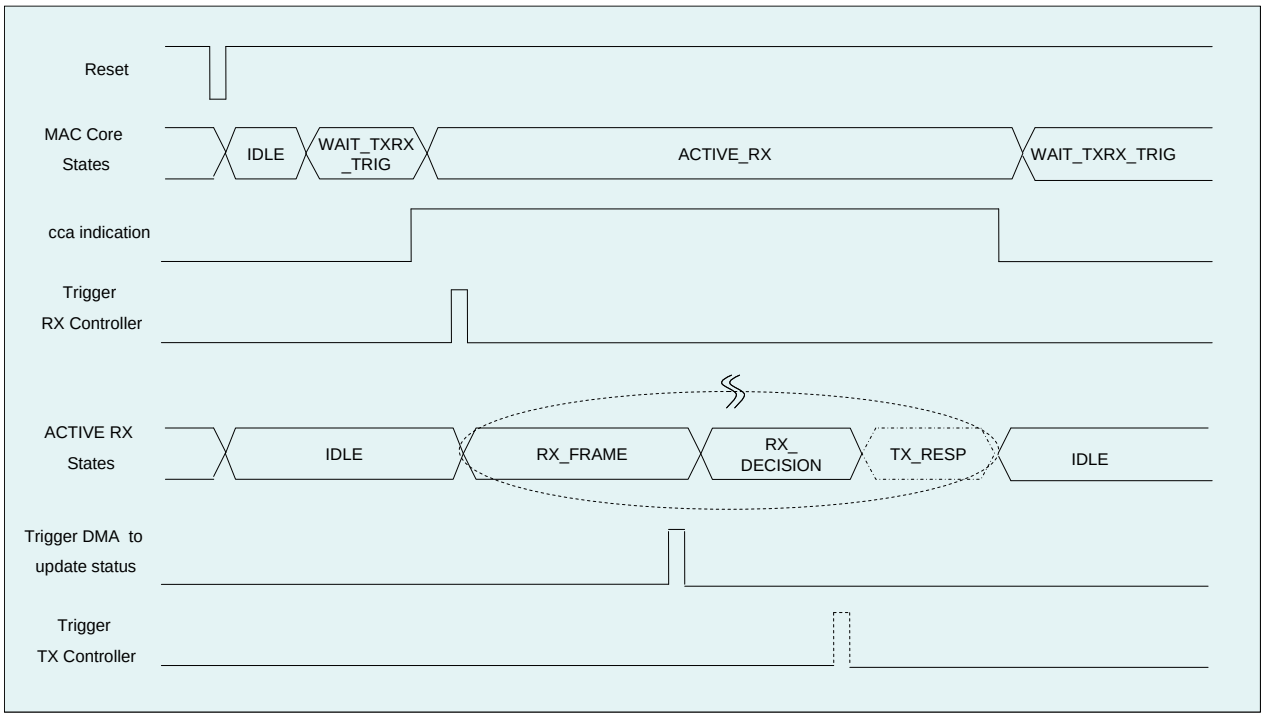


Figure 30: MAC Core state machine receive timing diagram

12.1.3.2 MAC Controller Transmit state machine

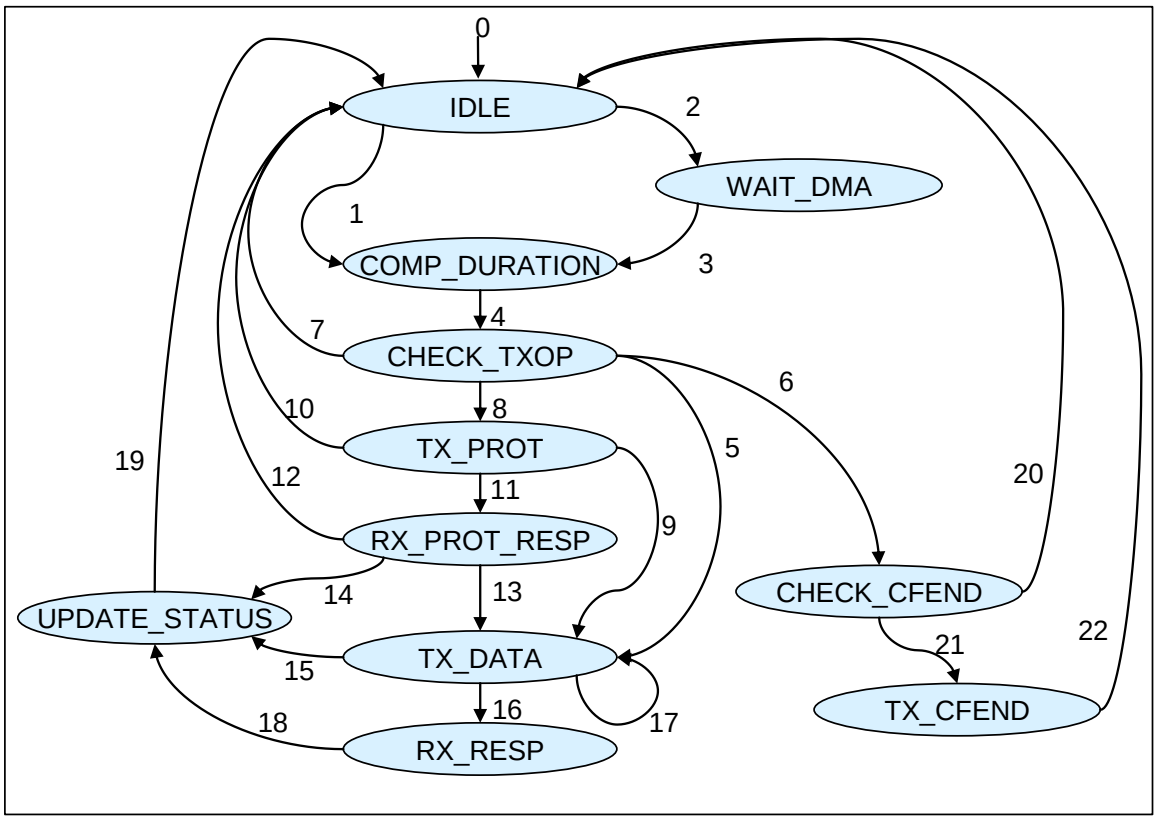


Figure 31: MAC Transmit state machine

State Name	State Description
IDLE	The state machine starts up in this state after reset or at the end of current transmit operation. In this state, the state machine waits for a trigger from MAC Core state machine.
WAIT_DMA	In this state, the MAC Transmit state machine trigs the DMA and waits till the Header Descriptor and Policy table are available on the TX Parameter Cache.
COMP_DURATION	In this state, the state machine launch the duration computing based on selected protection, rate, length, format and modulation.
CHECK_TXOP	In this state, the state machine checks if a NAV coverage already exist and if the frame sequence to be transmitted fit into the remaining TXOP. The *motXReg.acXMOT* register is updated according to the *TXOPLimit* and the estimated remaining TXOP after the transmission of the frame sequence.
TX_PROT	In this state, the Tx Controller is triggered to transmit the selected protection (CTS, Self-CTS, Dual-CTS …) and waits until the completion of the transmission. All the TX parameters like rate, MAC duration, legRate & legLenght, HT Info, MCS are provided to the TX Controller and MAC-PHY IF.
RX_PROT_RESP	In this state, the state machine trigs the RX Controller and waits till protection response is received or timeout happens.
TX_DATA	In this state, the state machine trigs the TX Controller to transmit the frame from the FIFO and waits till current MPDU transmission is completed.

State Name	State Description
RX_RESP	In this state, the state machine trigs the RX Controller and waits till response is received or timeout happens.
STATUS_UPDATE	In this state, the state machine triggers DMA to update status and wait till status update is completed. In case of unsuccessful protection, the backoff procedure is restarted.
CHECK_CFEND	In this state, the state machine checks if a CF-END fit into the remaining TXOP and L-SIG protection is not enabled. For that, the CF-END duration has to be computed.
TX_CFEND	In this state, the state machine trigs the TX Controller to transmit the CF-END and waits till the transmission is completed.

Table 12: MAC Transmit state machine state description

Transition No.	State Description
0	Hard reset or Soft reset.
1	This transition happens when MAC Core state machine triggers Transmit state machine for a transmission requested by SW.
2	This transition happens when MAC Core state machine triggers Transmit state machine and DMA Channel is PASSIVE.
3	This transition happens once DMA indicates completion of Header and Policy table reading from System memory.
4	This transition happens if the frame doesn't fit remaining TXOP or if the 40MHz TXOP acquisition did not succeed and the transmission cannot be switched to 20MHz (refer section *12.1.2.16 40MHz-80MHz-160MHz PPDU Transmission*).This transition happens once the computation of the different durations (MAC Duration, L-SIG Duration) is completed.
5	This transition happens if the frame exchanges sequence fits remaining TXOP and either no protection is required or a NAV coverage already exist.
6	This transition happens if the frame exchanges sequence doesn't fit remaining TXOP
7	This transition happens if the 40MHz TXOP acquisition did not succeed and the transmission cannot be switched to 20MHz (refer section *12.1.2.16 40MHz-80MHz-160MHz PPDU Transmission*). In this case the medium is released and the backoff procedure is restarted.
8	This transition happens if the frame exchanges sequence fits remaining TXOP and a protection is required (Self-CTS, RTS/CTS, RTS/CTS with QAP, STBC).
9	This transition happens if a response of the protection frame is not expected (Self-CTS) and the DMA Channel is ACTIVE
10	This transition happens if a response of the protection frame is not expected (Self-CTS) and if the SW has requested for frame control transmission.
11	This transition happens if a response of the protection frame is expected (RTS/CTS, RTS/CTS with QAP, STBC)
12	This transition happens if the SW has requested for frame control transmission. If the response frame is successfully received, the TXOP is granted.

Transition No.	State Description
13	This transition happens if the expected protection response has been received successfully in this case, the TXOP is granted.
14	This transition happens if the expected response is not received. In this case, the TXOP is not granted, the medium is released and the backoff procedure is restarted.
15	This transition happens when the transmission is completed and if the transmitted frame is not expecting any response.
16	This transition happens when the transmission is completed and if the transmitted frame is expecting response.
17	This transition happens when the transmission is completed and if the transmitted frame is part of A-MPDU.
18	This transition happens at the end of response frame reception or timeout event is received.
19	This transition happens if DMA status updated indication is received.
20	This transition happens if the CF-END if both *edcaCntrlReg.sendCFEnd* and *edcaCntrlReg.sendCFEndNow* are reset. In this case the medium is not released and the backoff procedure is not restarted. This transition happens if the CF-END does not fit into the remaining TXOP. In this case the medium is released and the backoff procedure is restarted.
21	This transition happens if the CF-END fit into the remaining TXOP and if one of the *edcaCntrlReg.sendCFEnd* or *edcaCntrlReg.sendCFEndNow* is set.
22	This transition happens when the CF_END transmission is completed. In this case the medium is released and the backoff procedure is restarted.

Table 13: MAC Transmit state machine state transition conditions

Note that this diagram represents the Functional Behavior of the MAC transmit FSM and might slightly differ from the implementation due to design constraints.

12.1.3.3 MAC Controller Receive state machine

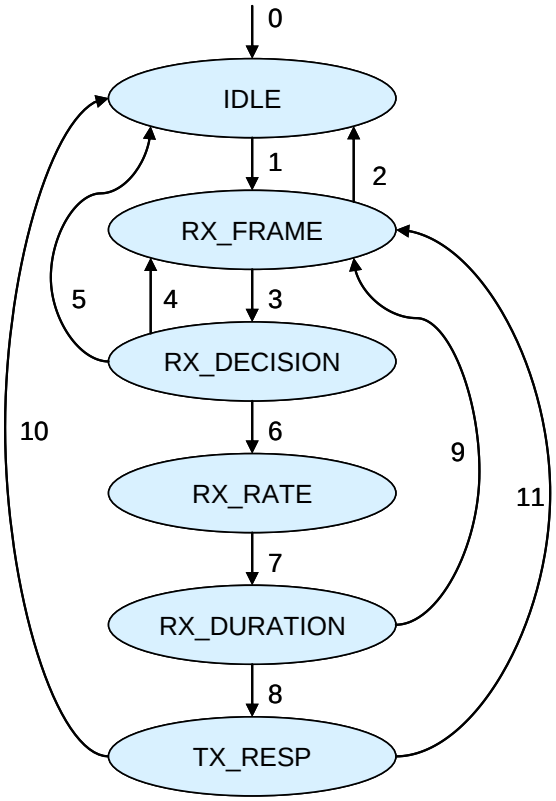


Figure 32: MAC Receive state machine

State Name	State Description
IDLE	The state machine starts up in this state after reset or at the end of current receive operation. In this state, the state machine waits for trigger from MAC Core state machine.
RX_FRAME	In this state, the state machine trigs the RX Controller and waits till reception of MAC Header is completed. Based on the frame type, BSSID, MAC Address, ACK Policy information are passed from RX Controller.
RX_DECISION	In this state, the state machine decides whether the frame requires a response transmission or not (ACK, CTS, CF-END, BA).
RX_RATE	In this state, the state machine trigs the rate management unit defined in section *12.1.4 Rate Management Unit* for rate calculation and waits till end of calculation.
RX_DURATION	In this state, the state machine trigs the TxTime Calculator defined in section *11,* TxTime Calculator for duration calculation and waits till end of frame with FCS status.
TX_RESP	In this state, the state machine trigs the TX Controller via *txControlFrame_req* signal with frame type (*txControlFrameType*), duration and RA to transmit the response frame and waits till TX Controller has completed the transmission.

Table 14: MAC Receive state machine state description

Transition No.	State Description
0	Hard reset or Soft reset.
1	This transition happens when MAC Core state machine triggers MAC Receive state machine.
2	This transition happens when MAC Core state machine moves MAC Receive state machine to IDLE state.
3	This transition happens when RX Controller indicates MAC Header reception is completed.
4	This transition happens when the frame reception is completed and the received frame is not expecting response and Duration field is bigger than zero.
5	This transition happens when MAC Core state machine moves MAC Receive state machine to IDLE state or if the frame reception is completed and the received frame is a Beacon frame or if the Duration field is set to 0 and no response transmission is required.
6	This transition happens when the received frame is expecting response.
7	This transition happens when the rate calculation is completed.
8	This transition happens when the duration calculation is completed and FCS is correct.
9	This transition happens when the FCS is failing.
10	This transition happens when the response transmission is completed and the Duration field in received frame is covering more then ACK.
11	This transition happens when the response transmission is completed and the Duration field in received frame is covering only ACK or zero.

Table 15: MAC Receive state machine state transition conditions

Note that this diagram represents the Functional Behavior of the MAC receive FSM and might slightly differ from the implementation due to design constraints.

12.1.4 Rate Management Unit

When the MAC Controller requests a transmission to the TX Controller, the TX Rate Management Unit computes immediately at which rate the transmission must be done. This rate selection depends on the type of transmission requested.

For the non-protection frame; the rate depends on the Rate Control Information defined in the Policy Table.

12.1.4.1 RTS prepared by software

If the frame is a protection frame generated by software and linked to the DMA, the rate selection follows the same scheme than for a non-protection frame.

12.1.4.2 RTS or Self-CTS prepared by hardware via *navProtfrmEx*

If the frame is a protection frame created by hardware and requested by software via *navProtfrmEx*, the rate depends on the *mcsIndexProtTx* fields defined in the Policy Table, the *ratesReg* and *MCSReg* registers and the number of retries already done.

12.1.4.3 Response frame (CTS, ACK, BACK) prepared by hardware

The response frame rate depends on the *bssBasicRateSet* or *MCSReg* registers and on the rate of the received frame. The selected rate is the highest rate defined in *bssBasicRateSet* or *MCSReg* registers which is lower or equal to the rate of the frame received.

The beamforming response frame preparation is described in *17, Beamforming Controller*

12.1.4.4 Control frame (CF-END) prepared by hardware

In case of CF-END transmission, the CF-END rate depends on how the TXOP has been acquired. The rules are:

If the TXOP has been acquired using RTC/CTS or Self-CTS, the rate selected is the highest rate defined in bssBasicRateSet

If the TXOP is 20MHz and has been acquired using Dual-CTS, the rate selected is the same than the RTS which started the TXOP.

If the TXOP is 40MHz and has been acquired using Dual-CTS, the rate selected is the same than the RTS which started the TXOP.

12.2 NAV

12.2.1 Overview

The NAV block is essentially a timer that keeps track of Network Allocation Vector which is used to maintain Virtual Carrier Sense (VCS). It interfaces to RX Controller, MAC Controller, CSR, Timer and Backoff Engine blocks. *Figure 33: NAV block diagram* shows the interactions of the NAV block with the various blocks in the MAC HW.

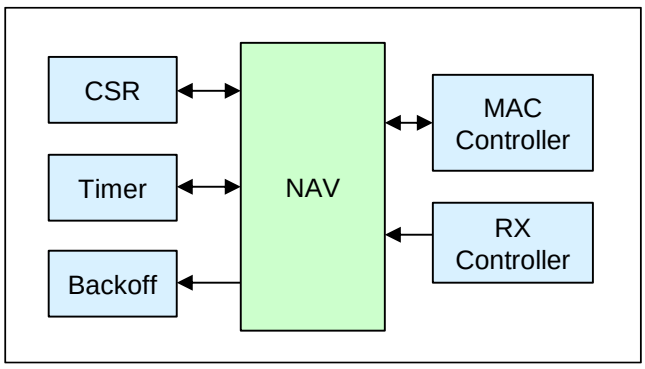


Figure 33: NAV block diagram

12.2.2 Functional Description

The NAV counter is decremented every 1 microsecond by the *tick1us_p* signal. The NAV block gives the counter value as output and is used by other MAC blocks. The *channelBusy* and *channelIdle* outputs are generated when the NAV counter is non-zero and zero respectively. These outputs are used in addition to *cca* signal from PHY by Backoff Engine and Timer to know if the medium is free or busy.

The NAV counter is reset based on following conditions:

1. If a RTS has been received which sets NAV, but the channel remains idle for (2 × aSIFSTime) + aPHY-RX-START-Delay + (2 × aSlotTime) + (CTS_Time) (at rate of previous RTS).
2. If a CF-END frame type is transmitted or received.

3. If a frame with L-SIG protection has been received where MAC duration is longer than L-SIG duration, but no PHY-RXSTART indication is received from PHY after aSIFSTime + aPHY-RX-START-Delay + (2 × aSlotTime) starting at the end of L-SIG duration.

The NAV counter is updated based on following conditions.

1. When core goes from IDLE to ACTIVE, NAV is updated with Probe Delay duration.
2. If a PS-Poll frame type is received, NAV is updated with time, in microseconds, required to transmit one ACK frame plus one SIFS interval if this is value is greater than current value in NAV.
3. For every correctly received frames not meant for this device, *durationID* is received from RX Controller and if the *durationID* is >= 32768, (check if 16th bit is set to 1), do not update the NAV, else update if the *durationID* is greater than current value in NAV.
4. If MAC HW receives an incorrect frame (FCS error) with L-SIG protection, NAV is updated with following value: L-SIG duration – (TXTIME – (aPreambleLength + aPLCPHeaderLength)) where:
    a. L-SIG duration is the time required from protection w.r.t. L-SIG parameters (L-SIG length and L-SIG rate),
    b. TXTIME is the time required to send the entire PPDU w.r.t. HT parameters (HT length and HT rate),
    c. aPreambleLength + aPLCPHeaderLength equal to 20 us.
    d. Note that L-SIG and HT durations are calculated by *1.2.2.9, TXTime Calculator*.
5. If quiet period 1 is reached, then update NAV with quiet duration defined in *quietElement1aReg.quietDuration1*.

12.3 Timer

12.3.1 Overview

The Timer block generates all timing related signals for MAC. It operates in *macCoreClk* and *macLPClk* for timers which must continue ticking in Doze state. It interfaces to RX Controller, MAC Controller, TX Controller, NAV, Backoff Engine and CSR blocks. Figure 34: Timer block diagram shows the interactions of the Timer block with the various blocks in the MAC HW.

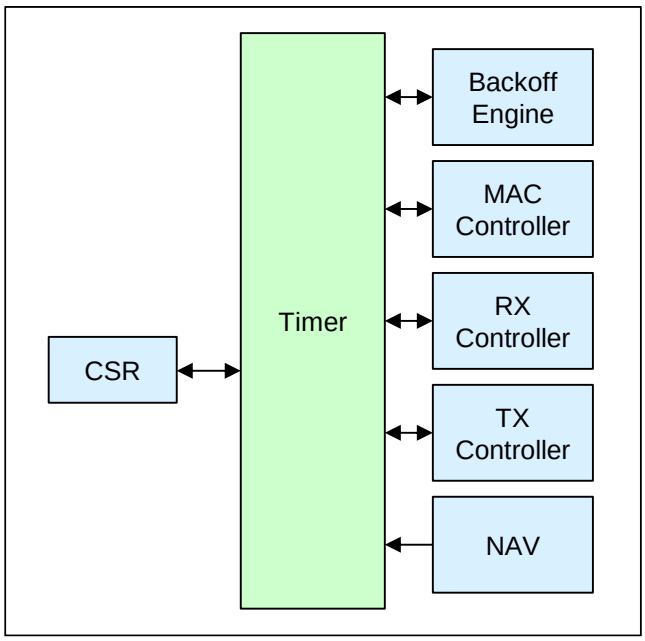


Figure 34: Timer block diagram

12.3.2 Functional Description

The Timer block generates all timing signals for MAC. It controls the timing reference of microseconds, TU (1024 µs), TBTT, TSF, SIFS, EIFS, PIFS, AIFS, RIFS, Slot time etc. Some of the parameters like AIFS and EIFS are generated per each Access Category separately. For generation of timing signals, the corresponding delays like MAC Processing delay, PHY Transmit Chain Delay, PHY Receive Chain Delay, Air propagation time etc are considered. All the timing values are programmable through timing registers (*timings1Reg* to *timings9Reg*). This block is also responsible for tracking DTIM count, beacon interval, etc.

12.3.2.1 TSF

The TSF is a free running 64-bit counter and is incremented every 1 microsecond by the *tick1us_p* signal. It can be initialized by SW through *tsfHiReg.tsfTimer[63:32]* and *tsfLoReg.tsfTimer[31:0]*. TSF is read for TX and maintained by RX as described below.

As an AP in an Infrastructure BSS, TSF is read by TX Controller in order to update timestamp field of Beacon or Probe Response frame transmission.

As a STA in an Infrastructure BSS, TSF is updated by RX Controller when a Beacon or Probe Response frame belonging to this BSS is correctly received.

As a STA in an IBSS, TSF is updated by RX Controller when a Beacon or Probe Response frame belonging to this BSS is correctly received only if timestamp is later than the local TSF. In IBSS mode, Beacon generation is distributed between each STA, this allows to synchronize TSF timer on the fastest STA clock.

TSF update from RX Controller is detailed in *Figure 35: TSF update from Beacon/Probe response reception* below. Beacon or Probe response reception at MAC is delayed from air by RX RF + RX PHY delays which is not fixed and depends on the length and rate, and thus cannot be estimated via a simple register value.

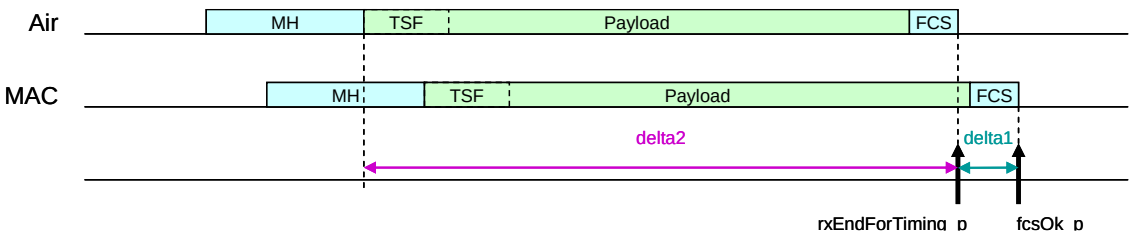


Figure 35: TSF update from Beacon/Probe response reception

If frame has been successfully received and when FCS pass, the TSF is updated with following formula:

TSF_{Update} = TSFreceived + delta1 + delta2 with:

delta1 = time between rxEndForTiming_p signal coming from PHY offset by RF delay and fcsOk_p signal.

delta2 = frame duration – MAC Header duration with duration calculated from length and rate of the frame. MAC Header length is 24-byte for management frame.

12.3.2.2 Beacon Interval and TBTT

The Beacon Interval counter is a 16-bit counter. It is loaded with the Beacon period set in register *bcnCntrl1Reg.beaconInt* and is decremented every TU.

When TSF is updated from RX Controller, Beacon Interval counter is also updated in order to guarantee that TBTT is generated when TSF modulo Beacon Interval = 0. TBTT have to be always aligned with TSF.

TBTT generated by *tickTBTT_p* signal tells HW to transmit beacon when the Beacon Interval counter reaches 0. In order to let time to SW to prepare Beacon transmission, an Impending TBTT interrupt is generated to SW via *tickImpTbtt_p* signal. It is configured in register *bcnCntrl1Reg.impTBTTPeriod*.

As an AP in an Infrastructure BSS, a *tickEarlyTbtt_p* signal is generated to BackOff Engine for Beacon transmission in order to compensate TX delays of MAC, PHY and RF defined in *timings3Reg.macProcDelayInMACClkI*, *timings1Reg.txChainDelayInMACClk* and *timings1Reg.txRFDelayInMACClk* respectively.

In order to avoid beacon transmission overlap with short beacon interval, a flag *noBcnTxFlag* is generated to MAC controller when the Beacon Interval counter reaches *bcnCntrl1Reg.noBcnTxTime* register value.

12.3.2.3 Listen interval

The Listen Interval counter is a 16-bit counter. It is loaded with the Listen Interval duration from register *dozeCntrl1Reg.listenInterval* when the MAC Core enters DOZE state and is decremented every Beacon Interval. This counter is used to wake up the MAC Core from DOZE to ACTIVE state if register *dozeCntrl1Reg.listenInterval* is not null.

12.3.2.4 DTIM

The DTIM is an 8-bit counter and is used only in STA mode. It is decremented every Beacon Interval When the counter reaches 0, it is re-loaded with DTIM period. DTIM counter informs about the next delivery of the DTIM Beacon where the TIM indication will be delivered. The MAC Core in DOZE state is woken-up when DTIM counter is null if *register dozeCntrl1Reg.wakeupDTIM* is set.

dozeCntrl

12.3.2.5 Slot time

The slot time is an 8-bit counter. It is loaded with the slot time value set in register *timings2Reg.slotTime* when the channel is busy. It is triggered when SIFS time expires by the *tickSifs_p* signal and decremented every 1 microsecond by the *tick1us_p* signal. When the counter reaches 0, it is reloaded with initial register value and it generates the *tickSlot_p* signal used by TX Controller to transmit frame as shown in *Figure 36: IFS timing diagram.*

In order to compensate MAC delay defined in *timings3Reg.macProcDelayInMACClk*, a *tickEarlySlot_p* signal is also generated to Backoff Engine. It is generated some MAC clock-cycle before tickSlot_p.

In order to reduce gate count, this counter is shared with SIFS counters defined in **Error! Reference source not found.** and 12.3.2.6.1 respectively.

12.3.2.6 IFS

The timer block is responsible of Interframe Space calculation as shown in *Figure 36: IFS timing diagram* and *Figure 37: EIFS timing diagram*. Six different IFSs are defined in the following sections:

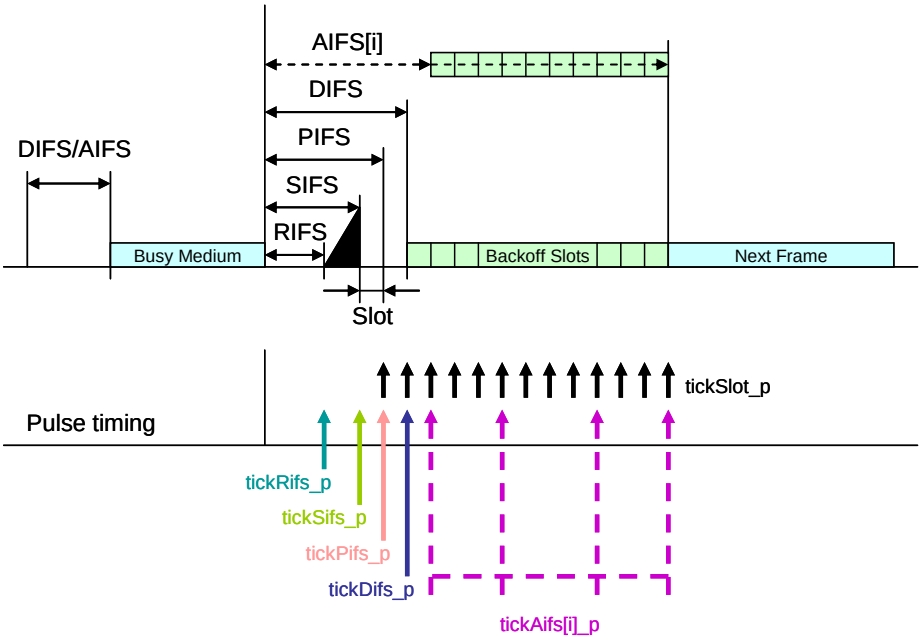


Figure 36: IFS timing diagram

12.3.2.6.1 SIFS

The SIFS is an 8-bit counter and is decremented every clock cycles. It is loaded with the SIFS values set in register *timings5Reg.sifsB* for DSSS-CCK frames or register *timings6Reg.sifsA* for OFDM and MIMO-OFDM frames. It is loaded when channel goes to idle (NAV or physical CS), at the end of the transmission or when *rxEndForTiming_p* is active. When the counter reaches [*timings1Reg.txRFDelayInMACClk* + *timings1Reg.txChainDelayInMACClk*] in case of the end of the transmission or [*timings1Reg.rxRFDelayInMACClk* + *timings1Reg.txChainDelayInMACClk*] in case of *rxEndForTiming_p* is active., it generates the *tickSifs_p* signal as shown in *Figure 36: IFS timing diagram*.

In order to compensate MAC delay defined in *timings3Reg.macProcDelayInMACClk*, a *tickEarlySifs_p* signal is also generated to Backoff Engine.

12.3.2.6.2 PIFS

The PIFS duration is evaluated on the secondary channel by three 8-bit counters decremented every 1 microsecond by the *tick1us_p* signal (one for secondary20, one for secondary40 and one for secondary80). Each of them is loaded with PIFS time when the secondary channel is free and re-initialized when the secondary channel is busy based on CCA. When the primary channel wins contention, the value of each counter is checked. If it has decremented to 0, then it means that the secondary has been free for PIFS.

12.3.2.6.3 AIFS

The AIFS is a 4-bit counter and is decremented every Slot duration by the *tickEarlySlot_p* signal. Four counters are needed, one for each AC. .Counters are loaded with the AIFS values set in registers *edcaAC0Reg.aifsn0*, *edcaAC1Reg.aifsn1*, *edcaAC2Reg.aifsn2* and *edcaAC3Reg.aifsn3* when the channel is busy. Each counter is loaded after SIFS time expiration via *tickSifs_p* signal. When the counters reach 0, they generate the *tickAifs0_p*, *tickAifs1_p*, *tickAifs2_p* and *tickAifs3_p* signals for the 4 ACs respectively. These pulses are used mainly by the Backoff Engine for Backoff decrement as shown in *Figure 36: IFS timing diagram*.

In order to compensate MAC delay defined in *timings3Reg.macProcDelayInMACClk*, *tickEarlyAifs[i]_p* signals are also generated to Backoff Engine.

12.3.2.6.4 EIFS

The EIFS is a 9-bit counter and is decremented every 1 microsecond by the *tick1us_p* signal. Four counters are needed, one per AC. These counters are used after reception of a frame with error from PHY or with an FCS error. They are loaded with the time in microsecond to transmit an ACK at the lowest rate of the previous erroneous frame mode (6Mb/s for OFDM or 1Mb/s DSSS-CCK) + one corresponding SIFS duration (OFDM or DSSS-CCK). Each counter is triggered after AIFS time expiration via *tickAifs0_p*, *tickAifs1_p*, *tickAifs2_p* and *tickAifs3_p* signals for the 4 ACs respectively. When the counters reach 0, they generate the *tickEifs0_p*, *tickEifs1_p*, *tickEifs2_p* and *tickEifs3_p* signals. These pulses are used mainly by the Backoff Engine for Backoff decrement as shown in *Figure 37: EIFS timing diagram*.

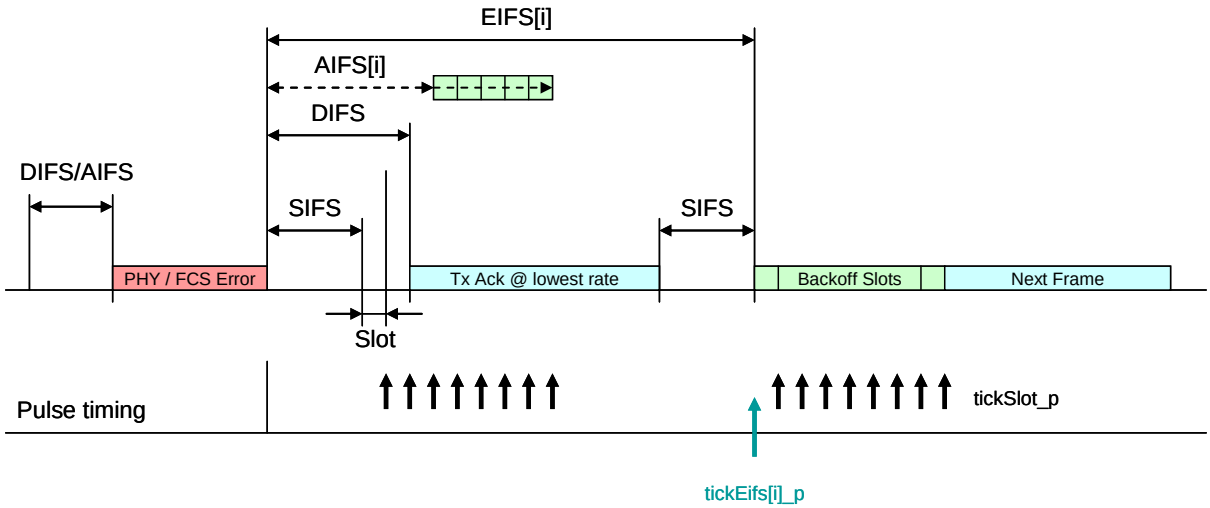


Figure 37: EIFS timing diagram

In order to compensate MAC delay defined in *timings3Reg.macProcDelayInMACClk*, *tickEarlyEifs[i]_p* signals are also generated to Backoff Engine.

EIFS is not triggered if L-SIG protection was requested on previous frame and if at least one frame of an A-MPDU has been received correctly.

An error free frame reception during an EIFS provokes EIFS counter reset.

12.3.2.7 Absolute Timers

The Absolute Timers delay the assertion of an interrupt to enable the MAC HW to collect additional interrupt events before delivering them to software as described in *Figure 38: Absolute Timer*. The Absolute Timers are particularly useful in high traffic environments. The Timer contains two Absolute Timers (one for RX and another one for TX)

The RX Absolute Timer starts to count down from *rxTriggerTimerReg.rxAbsoluteTimeout* upon receipt of the first frame (after software has enabled interrupts). Subsequent frames, if any, do not alter the countdown. Once the RX Absolute Timer reaches zero, the interrupt *timerRxTrigger* is generated. Then, the TX Absolute Timer is reloaded with *rxTriggerTimerReg.rxAbsoluteTimeout* and will restart upon the next reception,

The TX Absolute Timer starts to count down from *txTriggerTimerReg.txAbsoluteTimeout* upon receipt of the first frame (after software has enabled interrupts). Subsequent frames, if any, do not alter the countdown. Once the TX Absolute Timer reaches zero, the interrupt *timerTxTrigger* is generated. Then, the TX Absolute Timer will restart upon the next reception.

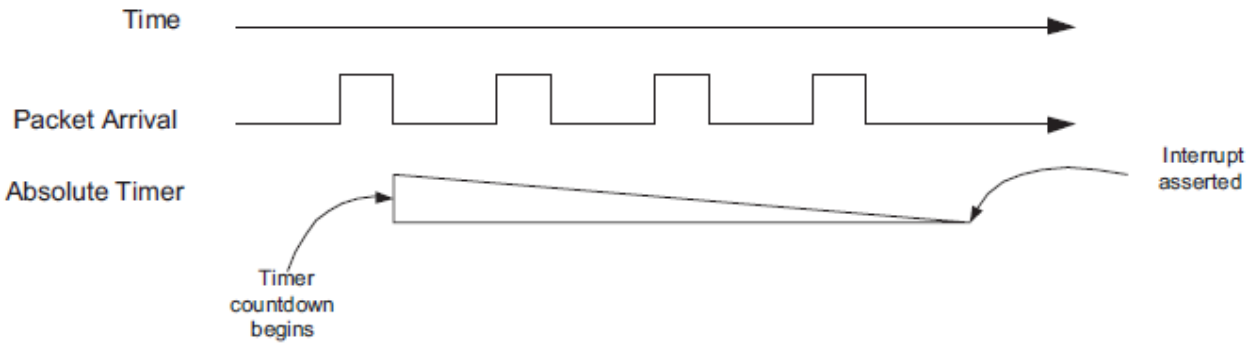


Figure 38: Absolute Timer

12.3.2.8 Packet Timers

The Packet timers are inactivity timers, triggering interrupts when the link has been idle for a long interval as described in *Figure 39: Packet Timer*. Software can use these timers to minimize frame latency in low traffic environments. The Timer contains two Packet Timers (one for RX and another one for TX)

The RX Packet Timer begins to count down from *rxTriggerTimerReg.rxPacketTimeout* upon receipt of a frame. If a new frame is received before the Packet Timer expires, it is reset and restarts to count down. Once the RX Packet Timer reaches zero, the interrupt *timerRxTrigger* is generated and the RX Packet Timer will restart upon the next reception.

The TX Packet Timer begins to count down from *txTriggerTimerReg. txPacketTimeout* upon transmission. If a new frame is transmitted before the Packet Timer expires, it is reset and restarts to count down. Once the TX Packet Timer reaches zero, the interrupt *timerTxTrigger* is generated and the TX Packet Timer will restart upon the next reception.

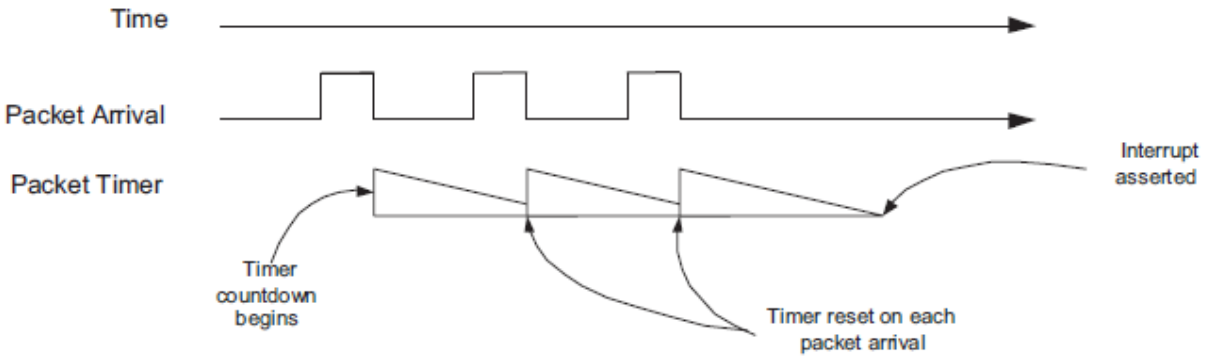


Figure 39: Packet Timer

12.3.2.8.1 EDCA CCA busy duration

The EDCA CCA busy duration is evaluated by a 32-bit counter. It is used only in QoS AP mode. Counter is incremented every 1 microsecond by the *tick1us_p* signal when the medium is busy (from NAV or physical CS signals). This counter is available in register *edcaCCABusyReg.ccaBusyDur* and cleared after SW reading.

Following the same mechanism, this counter is duplicated to also compute the CCA busy duration on secondary Chanel (Secondary20, secondary40 and secondary80). These values are available in register *ccaBusyDurSec20*, *caBusyDurSec40* and *ccaBusyDurSec80*.

12.3.2.8.2 Quiet intervals

The Quiet Intervals counters is a 16-bit counters which is decremented every 1 microsecond by the *tick1us_p* signal and loaded at each TBTT with time defined by following formula:

Impending quiet interval counter = (Quiet count x Beacon interval) + Quiet offset with:

Quiet count defined in register quietElement1aReg.quietCount1,

Beacon interval defined in register bcnCntrl1Reg.beaconInt and converted in microsecond,

Quiet offset defined in register quietElement1bReg.quietOffset1 and converted in microsecond.

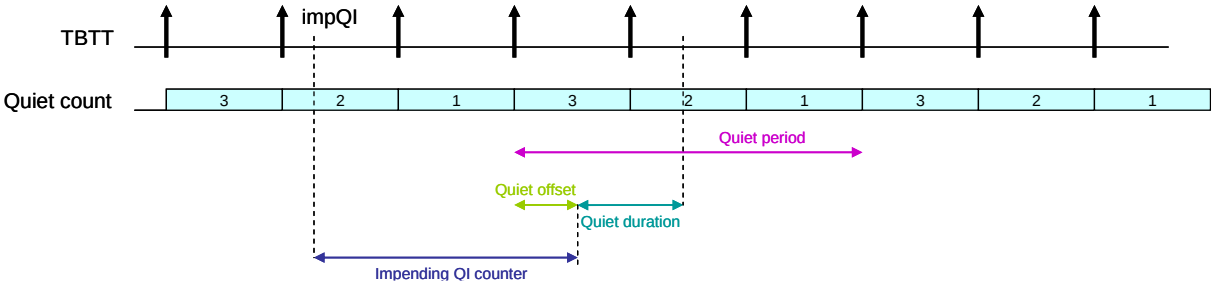


Figure 40: Quiet interval timing diagram

The quiet counter called *impQI* is sent to MAC Controller for TXOP calculation.

When quiet interval is reached (*impQI* = 0), NAV is triggered to load quiet duration defined in register *quietElement1aReg.quietDuration1* as described in section *12.2 NAV.*

12.3.2.9 Overlapping Legacy BSS Condition

RX Controller updates OLBC counters if an OLBC detection period has been requested by SW. The OLBC detection period is set in register olbcReg.olbcTimer. OLBC detection period is a 16-bit counter and is decremented every 1 microsecond by the *tick1us_p* signal. This counter is triggered when the detection period is not null and the counter runs continuously.

If the number of OFDM frames received not belonging to this BSS during the programmed detection period reaches olbcReg.ofdmCount register value, a trigger is sent to the interrupt controller. The same principle is applied to DSSS-CCK frames with olbcReg.dsssCount register. These two 8-bit frame counters are re-initialized when the detection period expires.

12.3.2.10 Monotonic Timers

Two monotonic counters are available in this module and the second can generate an interrupt when the counter reaches a programmed value.

The first monotonic counter, named monotonicCounter1, is 32bits and counts in macCoreClk clock cycle and is free running. It does not generate interrupt and is switched off when the macCore is running on the LPClk.

The second monotonic counter, named monotonicCounter2, is 48bits and counts in us. It generates interrupts when is value reaches one of the 10 programmed value (registers *absTimerValue0-9*). It still count when the macCore is running on the LPClk.

12.4 Backoff and AC Selection engine

12.4.1 Functional description

The Backoff Engine implements random backoff number generation, and performs the contention procedure based on current medium status. The backoff block has these following sub blocks.

Backoff random number generator that generates a random number.

Backoff Counter which counts the backoff of an AC based on the slot boundary. This block also keeps track of the current Contention Window.

Backoff Controller instantiates a backoff counter for each AC and maintains the Backoff counter.

The AC Selection Unit performs the contention procedure and resolves internal collision by giving *backoffDone_p* indication for high priority AC while indicating internal collision for low priority AC It generates also the early protocol trigger.

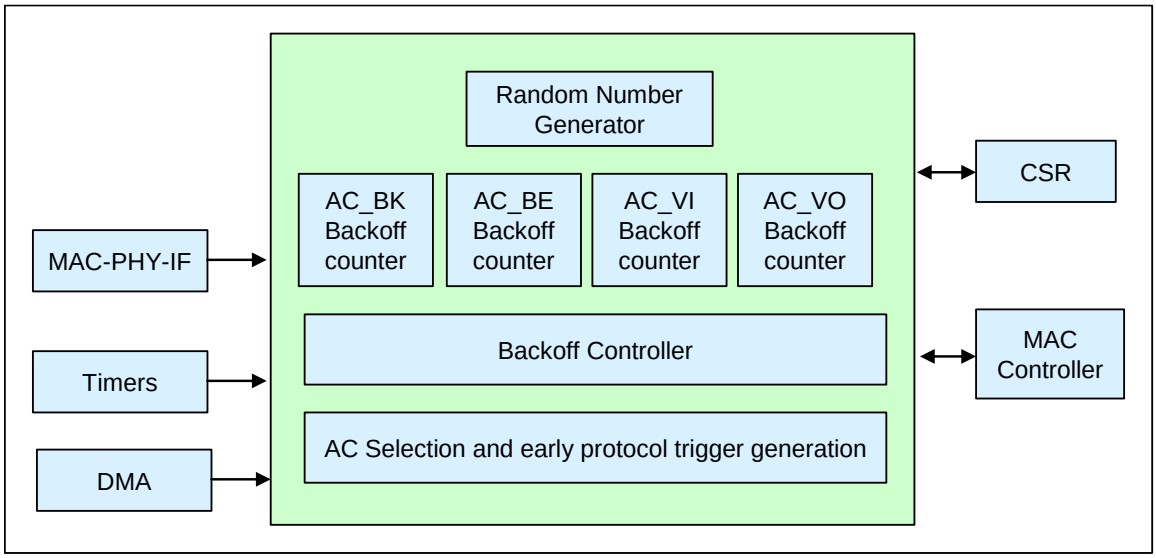


Figure 41: Backoff and AC selection Engine Block diagram

12.4.2 Random number generator

This block generates every *macCoreClk* clock cycle, a 14bits pseudorandom sequence named *randomValue*, based on a LFSR. The sixteen bit pseudo random number generator is implemented as a LFSR and it uses a simple polynomial, x[0] = x[13] ^ x[4] ^ x[3] ^ x[1]. Based on this 14bits pseudorandom number, each backoff counter is initialized using a different portion.

12.4.3 Backoff Counter

Each AC queue has its own backoff counter which counts down based on the slot (*tickSlot_p* pulse) information generated by the Timer Unit. Each counter is initialized using a part of the *randomValue* number coming from the Random number generator. This part depends on the current AC Contention Window and the AC number itself.

The counter is initialized based on the following formula:

initValue = *randomValue[CWx+nAC:nAC*]

where: *CWx* is the current Contention Window for this AC

nAC is the AC number (AC_BK = 0, AC_BE = 1, AC_VI = 2 and AC_VO = 3),

When reset, a backoff counter is reloaded using the current random number based on the CW value given by the backoff Controller.

When enabled, a backoff counter is decremented at each slot. When it reaches zero, it generates a pulse named *backoffTriggerACX*_p where X corresponds to one of the ACs and waits for a restart. When it is waiting, it still generates the *backoffTriggerACX*_p pulse at each slot. When it gets a restart trigger, it is reloaded with a new random number and restart counting down.

When disabled, a backoff counters is not reset but does not count down and does not generate pulse.

12.4.4 Backoff Controller

The Backoff Controller is the unit which controls the different Backoff Counters. There is one backoff counter per Access Category.

When the MAC state moves from IDLE to WAIT_TXRX_TRIG state, the Backoff Controller resets the different Backoff Counters with *cwMinX* (where X corresponds to one of the ACs) coming from the register *edcaACXReg.cwMinX*.

When the medium becomes idle (CCA low or end of NAV), the timer Unit starts to count SIFS, DIFS, AIFS or EIFS if FCS error occurred and then generates pulses at the end of each slot. Note that the Timers Unit takes into account the TX PHY delays.

When the medium becomes busy, all the backoff counters which are equal to zero are reset and loaded with a new random number. In this case, the contention window is not changed.

In IBSS mode, the beacon is transmitted after a backoff. As the QoS is not supported in IBSS Mode, the AC_BE backoff counter is reused to count the beacon backoff.

The AC_VI and AC_VO backoff counter are enabled upon a *tickAifsX_p* pulse.

The AC_BE backoff counter is enabled upon a *tickAifs1_p* pulse in QoS mode or *tickTbtt_p* pulse in IBSS mode.

The AC_BK backoff counter is enabled upon a *tickAifs0_p* pulse in QoS mode or DIFS in non-QoS mode.

All the backoff counters are disabled when the CCA or NAV are set or when a transmission is triggered.

The backoff Controller manages also the CW of the backoff counters.

In case of unsuccessful transmission, it reloads the backoff counter of the AC which won the contention and doubles the CW until it reaches *cwMaxX*.

In case of successful transmission, it reloads the backoff counter which won the contention and set the CW to *cwMinX*

In case of internal collision (refer to *12.4.5.1.2 Internal collision case*), it reloads the backoff counters of the AC which are equal to 0. For those ACs, it doubles also their CW.

12.4.5 AC Selection Unit

The AC Selection Unit generates the DMA TX triggers and manages the collision between the different queues. The AC Selection is done on the early slot boundary (*tickEarlySlot_p*) in order to take into account the TX delay. Only the backoff counters which are equal to zero on the early slot boundary take part of the AC selection procedure described below.

12.4.5.1 AC Selection procedure

The AC selection procedure depends on the number of backoff counter which are equal to zero at the same time. If several backoff counters are equal to zero at the same slot boundary, we have an internal collision.

12.4.5.1.1 No internal collision case

If only one backoff counter is equal to zero at the early slot boundary, the AC Selection Units checks if a frame has to be transmitted on this AC before to trig the MAC Controller with *backoffDone_p* pulse. This pulse is generated if:

1. The corresponding DMA channel is in PASSIVE state.

If none of the previous conditions happen, the *backoffDone_p* pulse is not generated and the backoff counter which won contention stays equal to zero until the next early slot boundary.

12.4.5.1.2 Internal collision case

If two or more backoff counters are equal to zero at the early slot boundary, we have an internal collision. In this case, the AC selection decides which AC win contention using the following rules.

1. The highest priority channel which has its DMA state in PASSIVE state wins contention.

If none of the previous conditions happen, the *backoffDone_p* pulse is not generated and all the backoff counters which won contention stay equal to zero until the next early slot boundary.

12.4.5.2 Early protocol triggers generation

This module is also in charge of the early protocol triggers generation. This early protocol triggers (*ac0ProtTrigger*, *ac1ProtTrigger*, *ac2ProtTrigger* or *ac3ProtTrigger*).

When one or more of the ACs will win contention, the corresponding early protocol trigger is generated.

12.4.6 Timing diagram

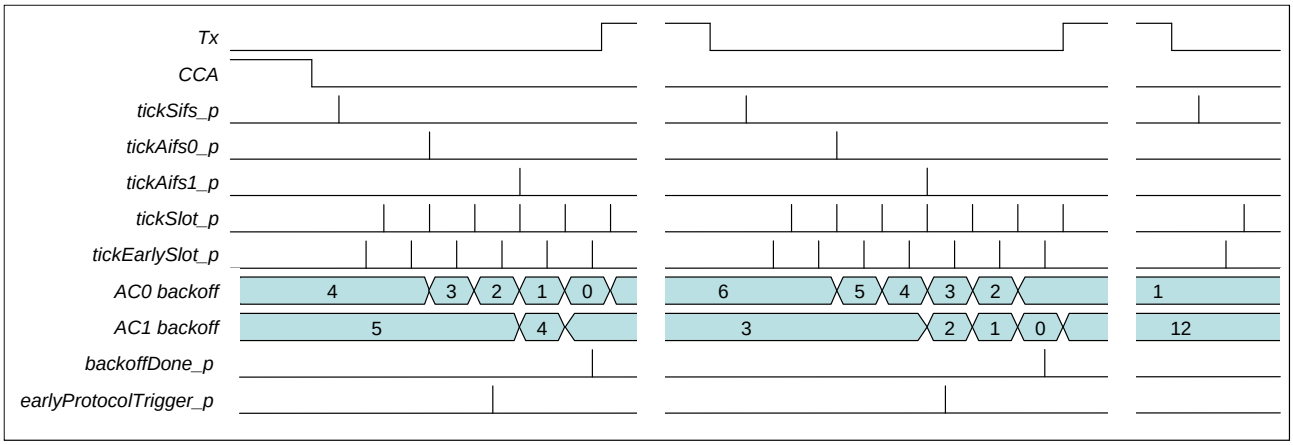


Figure 42: Backoff and AC selection timing diagram

12.5 TX Parameters Cache

12.5.1 Overview

The TX Parameters Cache block is responsible for storing the parameters of the Policy Table and Transmit Header descriptors read by the DMA Engine. It interfaces to the DMA Engine, the TX and MAC Controller, the MAC-PHY IF and the TX FIFO.

In order to not impact the DMA performance and to allow frame retransmission within SIFS, this block behaves as a cache for all the TX Parameters. It gets the Policy Table and Transmit Header Descriptor data from the DMA Engine via an interface named txCtrlReg.

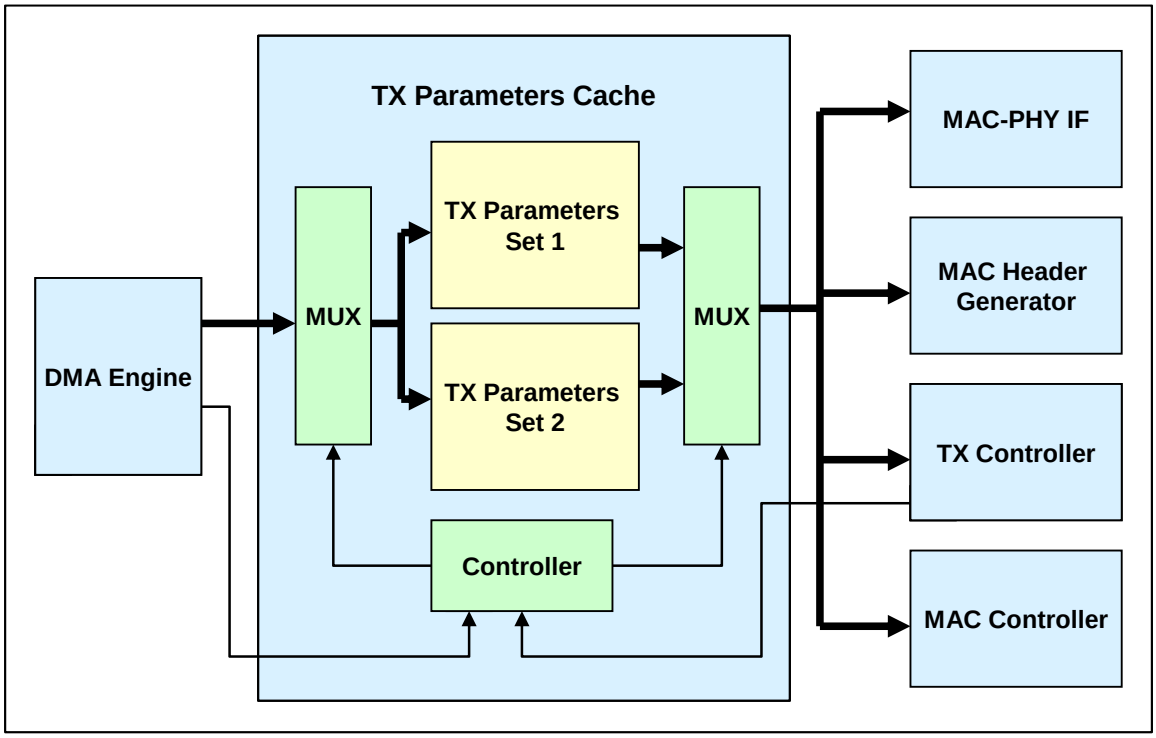


Figure 43: TX Parameters Cache Module Block Diagram

12.5.2 Functional Description

As the DMA can be fetching information for a second frame when the TX Controller is still transmitting the first one, the TX Parameters Cache block can store up to two different TX Parameters sets (Policy table and Header descriptor). If the TX Controller is still using the first set of TX Parameters and if the DMA Engine has already pushed the second set, the TX Parameters Cache block ties high the *txCtrlRegBusy* signal to stop the *txCtrlReg* write accesses from the DMA Engine.

This block runs on the *macPlClk1* clock.

12.5.2.1 Updating Tx Parameter Set

When the DMA is fetching a Transmit Header Descriptor, it sends some fields to the TX Parameters Cache module through the *txCtrlReg.* It indicates also using the *txCtrlRegHD* signal that it is sending the Transmit Header Descriptor. This signal is asserted during the Transmit Header Descriptor transfer and it is cleared when the last quadlet has been sent to the *txCtrlReg* interface. The DMA engine does not send all the Transmit Header Descriptor fields but only a subset of them as defined in *Figure 44: Fields of the Transmit Header Descriptor sent to the TX Parameters Cache*

.

Medium Time Used	
Status Information	
MAC Control Information 2	
MAC Control Information 1	
	Optional Frame Length (drop to 80MHz)
	Optional Frame Length (drop to 40MHz)
	Optional Frame Length (drop to 20MHz)
PHY Control Information	

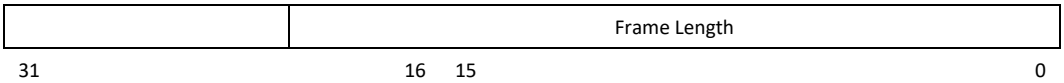


Figure 44: Fields of the Transmit Header Descriptor sent to the TX Parameters Cache

When the TX Parameters Cache receives a Transmit Header Descriptor, it checks the type of header descriptor defined in *whichDescriptor* in the *MAC Control Information 2* field to determine which fields are reserved. Note that the DMA Engine does not check this field and sends always all the Transmit Header Descriptor fields defined previously through the *txCtrlReg* interface even if some of them are reserved. Note also that the Buffer Length is provided by the DMA Engine but not used.

The TX Parameters Cache module updates the TX Parameters Set according to the *Figure 45: TX Parameters sets management scheme for singleton MPDU frames.*

In case of A-MPDU Header Descriptor, the TX Parameters Cache stored the *Frame Length* in a dedicated register named *txAMPDULength* to keep the A-MPDU frame length information even when the first MPDU of the A-MPDU frame Length will be received. This register will be cleared in case of singleton MPDU.

In case of an A-MPDU transfer, the DMA engine sends first the Transmit Header Descriptor which has {*aMPDU*, *whichDescriptor*} set to 3'b100 (A-MPDU Transmit Header Descriptor). So, the TX Parameters Cache updates the *txFrameLength* register, the *PHY Control Information*, the *MAC Control Information 1* and the *Status Information*. Then, the DMA Engine sends the Transmit Header Descriptor of the first MPDU which is part of this A-MPDU ({*aMPDU*, *whichDescriptor*} set to 3'b101) and the TX Parameters Cache updates the *txMPDULength* and the *MAC Control Information 2* but will not touch the others fields which will keep the A-MPDU value.

TX Parameters	***{aMPDU, whichDescriptor}* value**				
	3'b0XX	**3'b100**	**3'b101**	**3'b110**	**3'b111**
Frame Length	X	X	X	X	X
PHY Control Information	X	X			
MAC Control Information 1	X	X			
MAC Control Information 2	X	X	X	X	X
Status Information	X	X			

Table 16: Valid fields in the Transmit Header Descriptor

Then, the DMA fetches the Policy Table associated to this Transmit Header Descriptor and sends it also through the *txCtrlReg* interface. It indicates also using the *txCtrlRegPolicy* signal that it is sending the Transmit Header Descriptor. This signal stays high during the Policy Table transfer and it is cleared when the last quadlet has been sent through the *txCtrlReg* interface. The DMA Engine does not send the Unique Pattern through the *txCtrlReg* interface. Note that in case of constituent MPDUs of an A-MPDU, the Policy Table is not updated.

12.5.2.2 TX Parameters Sets management

12.5.2.2.1 Normal Operation

The TX Parameters Cache module has to manage the fact that the DMA Engine can provides a new TX Parameters set when the previous one is still in use. So, it keeps two set of TX Parameters which are alternatively used. Each time the MAC Controller updates the *Status Information* of a transmission, it trigs the TX Parameters Cache module to toggle or clear the TX Parameters Sets. In addition to that, the Tx Parameters Cache gives a status (*txCtrlRegBusy*) to the DMA

Engine to avoid overflow (i.e. to prevent the DMA Engine from sending through the *txCtrlReg* interface a third TX Parameters Set when the MAC Core is still using the first one).

The *Figure 45: TX Parameters sets management scheme for singleton MPDU frames* and the *Figure 46: TX Parameters sets management scheme for A-MPDU frames* show how the TX Parameters Cache module toggles between the two sets of TX Parameters.

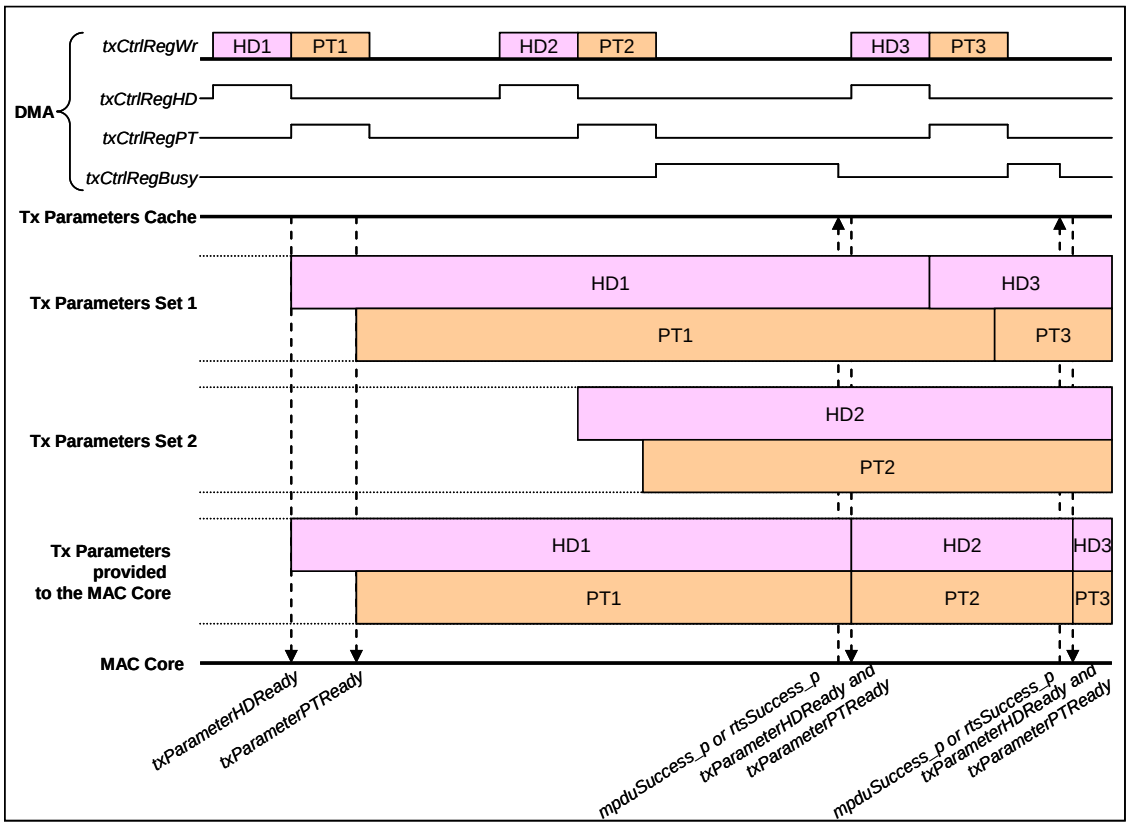


Figure 45: TX Parameters sets management scheme for singleton MPDU frames

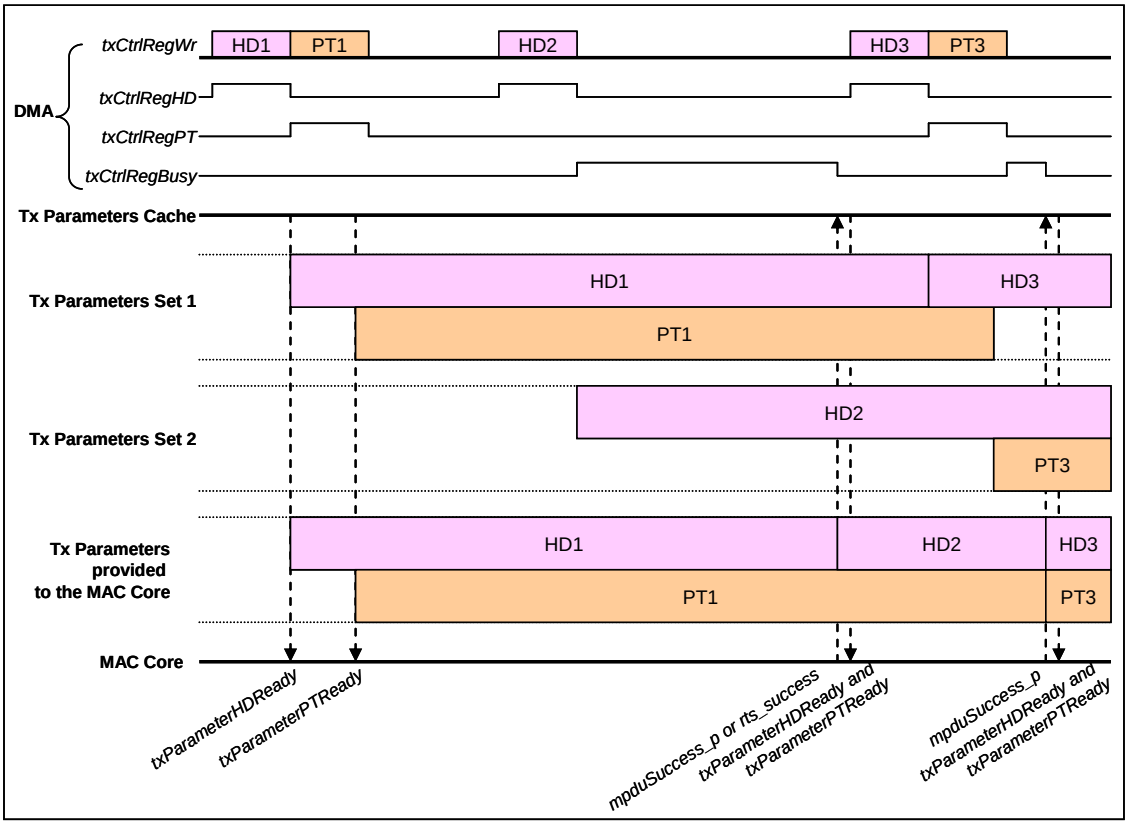


Figure 46: TX Parameters sets management scheme for A-MPDU frames

12.5.2.2.2 Discard Operation

After fetching a Transmit Header Descriptor and sending that over the *txCntlrReg* interface to the TX Parameters Cache block, the DMA Engine may discover that the frame should be discarded (due to *descriptorDone*HW*Tx* bit set or *frameLifetime* expiry). In this case, a signal *txCtrlRegDiscardHD_p* is generated to inform the TX Parameters Cache module to discard the Transmit Header Descriptor which was provided through the *txCtrlReg* interface. The *Figure 47: TX Parameters sets management scheme in case of Discard Operation* shows how the TX Parameters Sets are managed in case of Discard Operation.

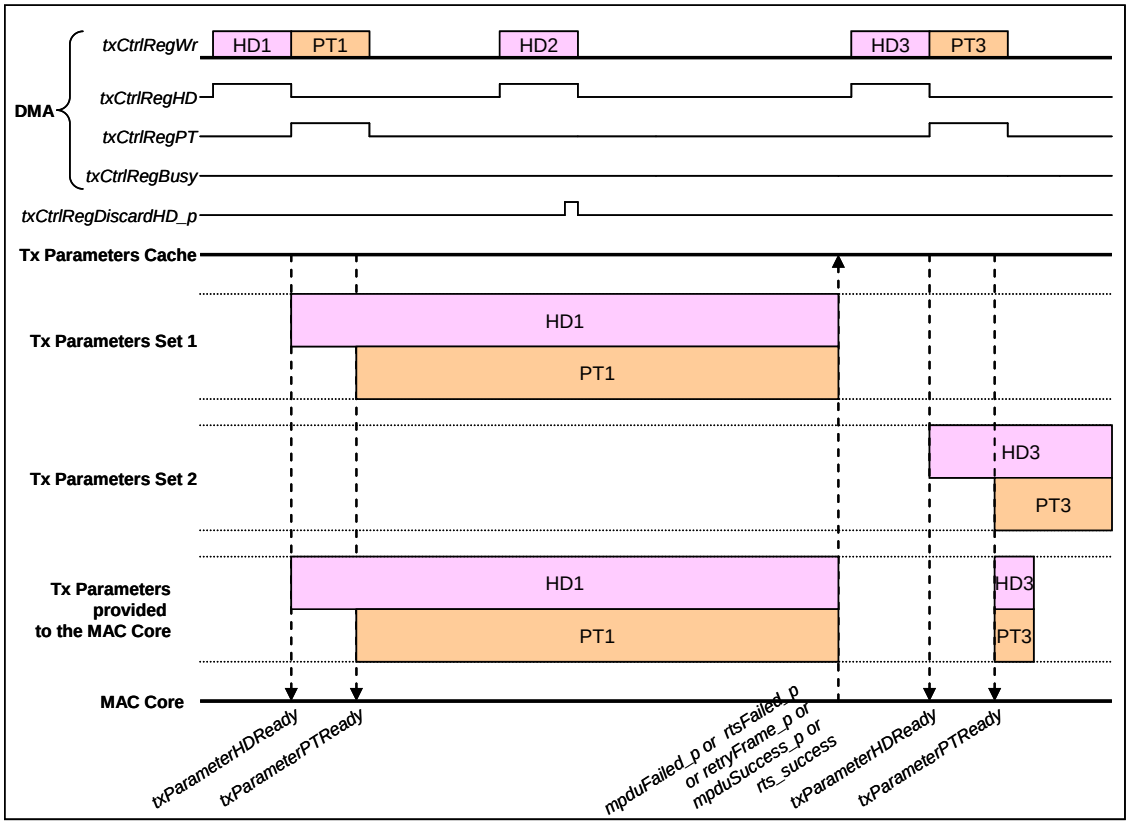


Figure 47: TX Parameters sets management scheme in case of Discard Operation

12.5.2.2.3 Retry Operation

In case of retry in a TXOP (*retryFrame_p*), the TX Parameters Cache module does not toggle the TX Parameters Sets. Thanks to that, the MAC Controller does not have to wait for the Transmit Header Descriptor and Policy table. As the DMA will provide anyway the THD and PT, the Tx Parameters Cache will drop them.

12.6 Transmit Controller

12.6.1 Overview

The Transmit controller (henceforth referred to as TX Controller or the transmitter) block controls the transmit data path from the Transmit FIFO to the PHY Interface. It reads the frame present in the Transmit FIFO and passes it through the encryption block (if required) and the FCS block and writes to MAC-PHY Interface FIFO. The transmitter interfaces to Transmit FIFO, MAC Controller, DMA Engine, CSReg, Timer block, BA Controller, FCS and Encryption blocks.

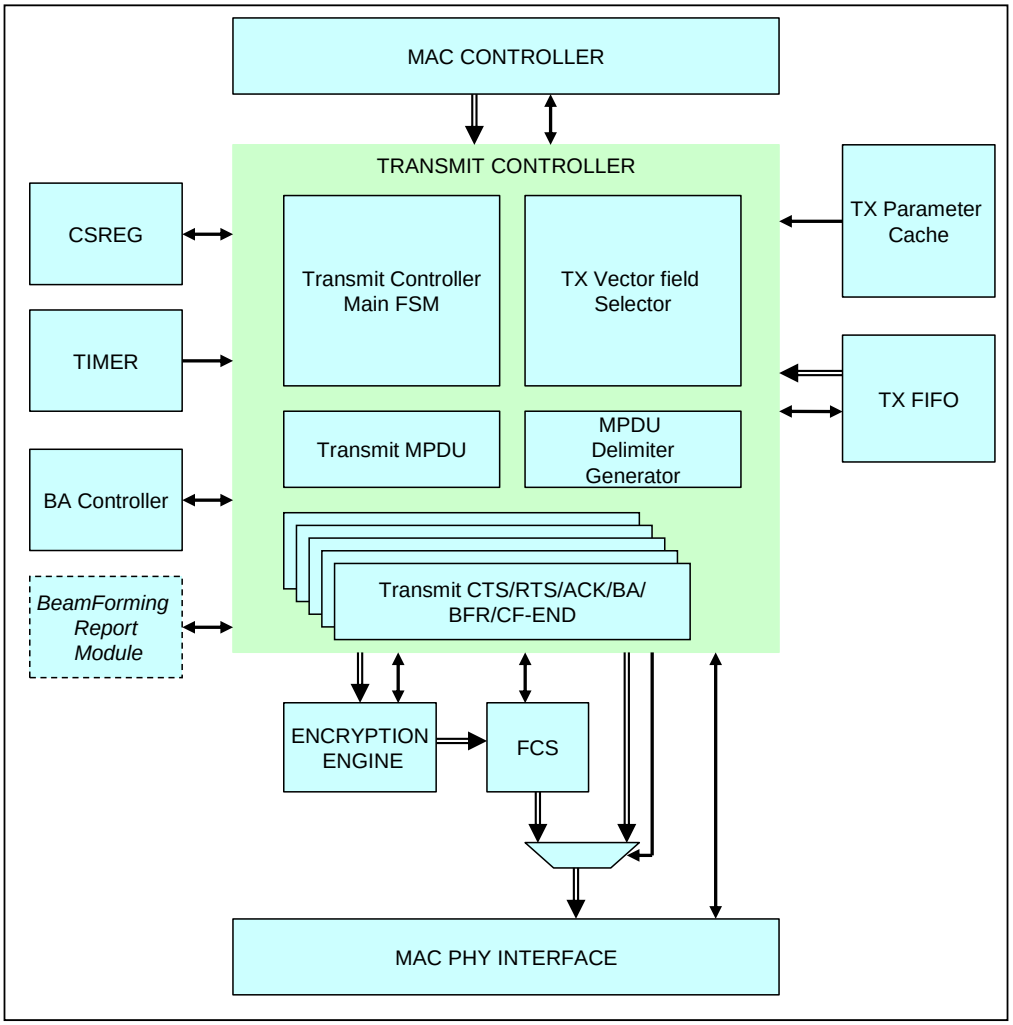


Figure 48: Transmit Controller Block Diagram

12.6.2 Functional Description

Following are the main functions of TX Controller.

- *Transmission of Data, Management and Control frames from Transmit FIFO*
- *Control and Special frames preparation and transmission*
- *A-MPDU/MPDU transmission*
- *Triggering MAC-PHY Interface for transmission of frames*
- *Triggering Crypto Engine*
- *Triggering FCS block*
- *Interacting with MAC Controller*
- *Interacting with DMA Engine/Tx Parameter Cache*
- *Getting triggers from Timer block*

All the above functions of TX Controller are performed by the transmitter state machine along with transmitter logic block. Depending on inputs from CSReg, MAC Controller, TX Parameters Cache and DMA Engine control signals for other blocks are generated.

This block runs on the *macPlClk1* clock.

12.6.2.1 Transmission preparation

There is three different type of transmission but all of them are triggered by the MAC Controller.

When the TX Controller receives a *sendData_p*, it starts the transmission of the frame contained into the TX FIFO and follows the process described in *12.6.2.1.1 Transmission of data, management and control frames from Transmit FIFO*.

The MAC Controller can also request, using the *sendRTS_p, sendCTS_p, sendACK_p, sendBA_p, sendCFEND_p and sendBFR_p,* the TX Controller to transmit a control frame which will be completely generated by the TX Controller itself. This second method is described in *12.6.2.1.2 Control and Special frames preparation and transmission.*

12.6.2.1.1 Transmission of data, management and control frames from Transmit FIFO

When receiving the *sendData_p* from the MAC Controller, the TX Controller starts the transmission of the frame contained into the TX FIFO.

It starts reading the first two bytes of the Tx FIFO to get the frame type and subtype but also the value of the protection bit.

During the Key Search process, the Tx Controller processes the MAC Header as described in *12.6.2.4 MAC Header generation* and pushes it to the FCS Engine. The output of FCS Engine is fed into the MAC-PHY Interface. TX Controller ignores any padding present in TX FIFO during transmission.

When it gets the *txKeySearchEnd* from the Key Search Engine indicating that the process is completed, it launches the initialization of the Encryption Engine by raising the signal *txInitCrypto*.

Once the Encryption Engine has completed its initialization, the Tx Controller is informed by the signal *txInitCrypto*.

Once the MAC Header has been fully pushed to the FCS block, and only when the Encryption Engine initialization is completed, the TX Controller starts to push the data to the Encryption Engine, which will be encrypted or not and then forwarded to the FCS Engine.

If the frame type and subtype indicate protection frame (CTS) when the TXOP is already acquired, the frame is not transmitted. In this case, the TX FIFO is popped until the end of MPDU flag is read. The transmission status reported to the MAC Controller is “transmission successful”.

12.6.2.1.2 Control and Special frames preparation and transmission

The TX Controller can create and transmit several Control and Special frames like (RTS, Self-CTS, ACK, CTS, CF-END, BeamForming Report, BA, Bandwidth Report or NDP Feedback). However, the decision to transmit such frames is not taken by the TX Controller but triggered by the MAC Controller using the *sendRTS_p, sendCTS_p, sendACK_p, sendBA_p or sendCFEND_p* indication.

Depending on *NAV Protection Frame Exchange* field of the *MAC Control Information* field, TX Controller can be requested by the MAC Controller to create and transmit RTS or Self CTS.

TX Controller prepares and transmits ACK, Block ACK or CTS based on trigger from MAC Controller in response to received frames.

CF-End frame is created in TX Controller when triggered by MAC Controller and is used to release the un-used TXOP under Long NAV protection.

The MAC Controller block tracks the successful FCS check of received frames per TID/RA and requests the transmission of an ACK or BA towards the TX Controller block either automatically or upon BAR reception. In case of a BA transmission, the TX Controller generates the BA using the bitmap field coming from the Block Ack Controller.

Receiving the *txControlFrame_req* request, the TX Controller raises the signal *txVectorGeneration_req* to request the MAC-PHY IF to start the transmission. As soon as the MAC-PHY FIFO is ready (not full),, it pushes the generated frame into the FCS Engine. The *txVectorGeneration_req* flag is generated on the slot boundary.

The TX rate comes from the MAC Controller and its Rate Management Unit which is in charge of the TX rate selection as described in *12.1.4 Rate Management Unit*.

12.6.2.2 Beacon/probe response Transmission

When reading the first two bytes of the TX FIFO, the TX Controller detects if the transmitting is a beacon or a probe response. In this case, it can update, depending of the *dontTouch* definition of the Header Descriptor, the value of the DTIM Count and the TSF fields. The TSF field is updated for both probe response and beacon. The TSF value comes from the TSF block

If the *donTouchDTIM* bit is not set, the TX Controller searches the TIM Element ID (i.e. 5) when shifting the data from the TX FIFO. Note that this field is present only in the beacon and does not exist in probe response frame.

12.6.2.3 A-MPDU Transmission

TX Controller prepares A-MPDU header, generates CRC and appends MPDU Delimiter. Once A-MPDU header transmission is passed to the MAC-PHY Interface, TX Controller transmits actual MPDU. Depending on the *nBlankMDelimiters* value in Transmit Header Descriptor, TX Controller inserts blank MPDU delimiters (MPDU delimiter with null length) to satisfy the MPDU Density requirement.

It automatically inserts the required number of padding bytes while transmitting the A-MPDU to make the length a multiple of 4. Note that this A-MPDU Delimiter does not go through the FCS block and is directly pushed to the MAC-PHY IF.

12.6.2.4 MAC Header generation

The TX Controller decides whether certain MAC Header fields will be transmitted or not depending on the *dontGenerateMH* field in the MPDU Header Descriptor.

If *dontGenerateMH* is set, TX Controller transmits the frame exactly as programmed from SW whatever the frame type/subtype is.

If *dontGenerateMH* is reset, TX Controller transmits the MAC Header fields like Frame control, Duration, Address 1-4, Sequence Control, QoS Control, HTC and encryption related fields depending of the type and subtype of the frame and some other parameters. The MAC Header field selection follows the rules defined in the *Table 17: MAC Header fields included during transmission*.

Field	Condition	Type	SubType	Others
Frame Control	Always	XX	XXXX	
Duration/ID	Always	XX	XXXX	
Address 1	Always	XX	XXXX	
Address 2	For all Data and Management frames.	X0	XXXX	
	For all control frames except Control Wrapper, ACK and CTS	01	All except 110X and 0111	

Field	Condition	Type	SubType	Others
Address 3	For all the Data and Management frames.	X0	XXXX	
Sequence Control	For all the Data and Management frames.	X0	XXXX	
Address 4	Only for Data frame with *ToDS* and *FromDS* fields set to 1	10	XXXX	If *ToDS* and *FromDS* bits (from *Frame Control* field) are set to 1
QoS Control	Only for QoS Data frame	10	1XXX	
Carried Frame Control	Only for Control Wrapper frame	01	0111	
HT Control	For Control Wrapper frame and frame with Order bit set	01	0111	
		XX	XXXX	If *Order* bit (from *Frame Control* field)is set to 1
IV / Key ID	All Data and Management authentication frames which have been encrypted	10	XXXX	If *Protected Frame* bit (from *Frame Control* field) is set to 1
		00	1011	If *Protected Frame* bit (from *Frame Control* field) is set to 1
Extended IV	All encrypted frames with extended IV information	X0	XXXX	If *Protected Frame* bit (from *Frame Control* field) is set to 1 And if *ExtID* bit (from *IV/Key ID* field) is set to 1
ICV	All Data and Management authentication frames which have been encrypted	XX	XXXX	If *Protected Frame* bit (from *Frame Control* field) is set to 1, and if the encryption algorithm is TKIP and the *dontEncrypt* bit is not set
CCMP MIC	All Data and Management authentication frames which have been encrypted	XX	XXXX	If *Protected Frame* bit (from *Frame Control* field) and *ExtID* bit (from *IV/Key ID* field) are set to 1, and if the encryption algorithm is CCMP and the *dontEncrypt* bit is not set
FCS	All frames	XX	XXXX	

Table 17: MAC Header fields included during transmission

Then the TX Controller decides whether certain MAC Header fields will be updated or not depending on the *dontTouchXX* fields in the MPDU Header Descriptor.

Retry bit is set by TX Controller if it is performing frame retransmissions if *dontTouchFC is equal to 0*.

PowerMgmt is set by the HW for each frame based on the value programmed in the *macCntrl1Reg.pwrMgt* field by the SW if *dontTouchFC is equal to 0*.

Duration is updated with the duration provided by the Duration Computing block (refer section *11,* TxTime Calculator) if *dontTouchDur is equal to 0*.

The TX Controller is also in charge of the modification of some fields in the beacon or probe response frames

TSF field is updated based on the current value of the TSF timer if dontTouchTSF is equal to 0

DTIM Count field is updated based on the TIM Counter value if dontTouchDTIM is equal to 0

Software can program any type of frame to TX FIFO and TX Controller transmits the frame based on length and frame type.

12.6.2.5 CTS generation

When requested by MAC Controller, the TX Controller creates and transmits a CTS or Self-CTS frame.

The Frame Control field is generated as following

type and sub type : 6'b011100

all the other bits are set to 1'b0

The duration and RA fields are provided by the MAC Controller.

The FCS is computed by the TX Controller using the FCS Engine.

Note that in case of Self-CTS the MAC Controller provides the MAC Address contained into *macAddrLowReg* and *macAddrHighReg* registers

12.6.2.6 RTS generation

When requested by MAC Controller, the TX Controller creates and transmits a RTS frame.

The Frame Control field is generated as following

- type and sub type : 6'b011011
- all the other bits are set to 1'b0

The duration, TA (srcAddr) and RA (destAddr) fields are provided by the MAC Controller.

The FCS is computed by the TX Controller using the FCS Engine.

12.6.2.7 ACK generation

When requested by MAC Controller, the TX Controller creates and transmits a ACK frame.

The Frame Control field is generated as following

type and sub type : 6'b011101

all the other bits are set to 1'b0

The duration and RA fields are provided by the MAC Controller.

The FCS is computed by the TX Controller using the FCS Engine.

12.6.2.8 BA generation

When requested by MAC Controller, the TX Controller creates and transmits a BA frame.

The Frame Control field is generated as following

type and sub type : 6'b011001

all the other bits are set to 1'b0

The duration, TA (srcAddr) and RA (destAddr) fields are provided by the MAC Controller.

The BA Control field is generated as following

BA Ack Policy

The Block Ack Starting Sequence Control field comes from the BA Controller.

The Block Ack Bitmap comes from the BA Controller.

The FCS is computed by the TX Controller using the FCS Engine.

12.6.2.9 CF-END generation

When requested by MAC Controller, the TX Controller creates and transmits a CF-END frame.

The Frame Control field is generated as following

- type and sub type : 6'b011110

-pwr bit is set based on *macCntrl1Reg.pwrMgt* bit

- all the other bits are set to 1'b0

The duration is forced to 0

The RA field is force to FFFFFF.

The TA (srcAddr) field is provided by the MAC Controller.

The FCS is computed by the TX Controller using the FCS Engine.

12.6.2.10 Compressed BeamForming Report generation

The Compressed Beamforming Report is carried in a Non-Ack Action Frame.

The Figure 49 represents a Non-Ack Action frame format. The Frame Body carries the VHT Compressed Beamforming report (Figure 50)

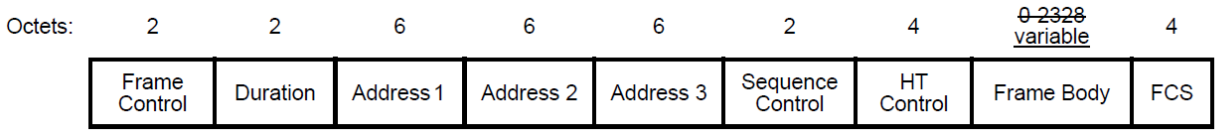


Figure 49: No-Ack Action frame Format

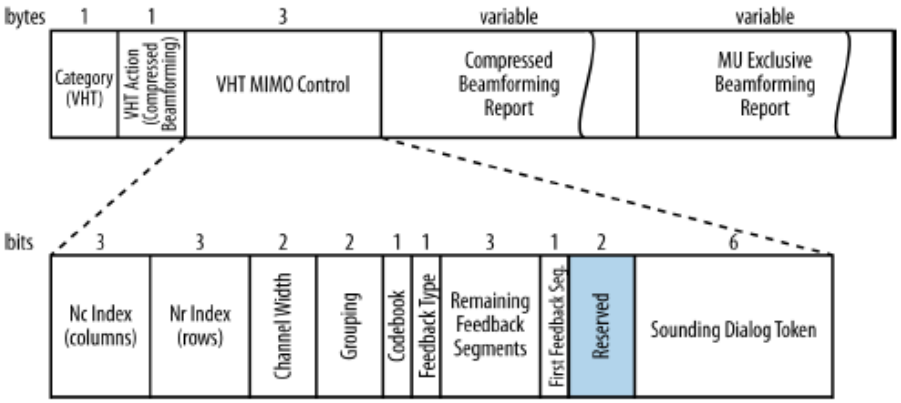


Figure 50: VHT Compresssed BeamForming report

When requested by MAC Controller using sendBFR_p, the TX Controller creates and transmits a Beamforming Report frame.

Prior to the frame generation, it shall create the AMPDU-Delimiter if the *formatMod* is VHT.

The Frame Control field is generated as following

- Type and SubType : 2'b00 4'b1110 (No-ACK Action frame)
- The pwr comes from a register.
- All the other bits are set to 1'b0

The duration, TA (srcAddr) and RA (destAddr) fields are provided by the MAC Controller.

The addr3 field is the BSS Address contained into bssID register

The Sequence Control: fragment number is set to 0. Sequencve number is incremented each time a BFR is transmitted.

The HT Control is not transmitted

The category is VHT Action frame set to 8'd21

VHT Action (Compressed Beamforming) is set to 8'd0

The VHT MIMO control is generated as following:

- Nc Index, Nr Index, Channel Width, Feedback Type and Sounding Dial Token are provided by the MAC Controller
- Grouping, Codebook come from registers
- Remaining Feedback Segment is set to 3'b0
- First Feedback Seq is set to 1'b1;

The Compressed Beamforming report and MU exclusive Beamforming report (if any) come from the BeamForming external module. These two fields are provided through the Beamforming Report interface correctly formed. No processing is expected on the received by from the TX Controller.

The FCS is computed by the TX Controller using the FCS Engine.

The length of this frame depends on the Beamforming report parameters. This length is computed in the MA Controller and provided to the TX Controller (*txHTLength* input).

The interface with the Beamforming report external module is the following:

Name	Direction	Size	Description
bfrData	Input	8	BeamForming Report Byte
bfrDataValid	Input	1	BeamForming Report Byte Valid Indication from the beamforming HW accelerator that the bfrData is valid and shall be captured.
bfrDataReady	Output	1	BeamForming Report Byte Ready Indicate to the beamforming HW accelerator that the TX Controller is ready to accept new byte.

Table 18: Interface with the Beamforming external HW accelerator

12.6.2.11 Triggering MAC-PHY Interface for transmission of frames

Depending on early transmit trigger from MAC Controller, TX Controller waits for slot or SIFS boundary coming from the Timers module and indicates start transmission indication to MAC-PHY IF. Based on this trigger, the MAC-PHY IF prepares and starts sending the TX Vector. Once the TX Vector transmitted, the TX Controller gets a confirmation from the MAC-PHY IF. TX Controller indicates also last byte transmission to MAC-PHY IF. Note that the Slot or SIFS boundary already includes the TX PHY delay.

In HT-MF and VHT mode, the Tx Controller forces the MAC-PHY-IF *legRate* to 6Mbps and the *legLength* comes from the MAC Controller.

12.6.2.12 Triggering Crypto Engine

If the *Protected Frame* bit in the *Frame Control* field of the MAC Header is set and the *dontEncrypt* field in the MPDU Header Descriptor is reset, the TX Controller triggers the Key Search Engine to read the required encryption parameters from Key Storage RAM and also gives IV to Crypto Engine. Once key search is completed, the TX Controller triggers other crypto blocks based on encryption type required.

After triggering Encryption engine, TX Controller waits till Crypto engine initialization is completed in case of WEP or TKIP. Once the Crypto engine is ready for encryption, TX Controller pushes data to Encryption engine. Even if the frame does not need to be encrypted, the transmitted data are pushed through the crypto engine which will be in “bypass” mode.

12.6.2.13 Triggering FCS block

TX Controller triggers FCS block at start of frame transmission with the signal *fcsStart_p* and passes every byte of data to be transmitted on air through it. Contrary to the data which come from the crypto engine, the MAC Header data are pushed into the FCS block by the TX Controller. Once all bytes are transmitted, TX Controller indicates FCS engine to append FCS calculated on data to be transmitted by raising the *fcsShift_p* signal.

The data outputted by the FCS block are pushed to the MAC-PHY IF. When the latest byte of the FCS signature has been pushed to the MAC-PHY IF, the FCS block indicate the end of the transmission to the TX Controller with the pulse *fcsEnd_p*..

12.6.2.14 Interacting with MAC Controller

MAC Controller triggers TX Controller to transmit

Frames from TX FIFO

Transmit Control frames for protection

Transmit Control frames for response

For all response frames, TX Controller starts transmission at SIFS boundary. For all non response frames, TX Controller starts transmission at early slot boundary. Once frame transmission is completed, TX Controller moves to IDLE state and indicates same to MAC Controller. In abort cases, TX Controller indicates same to MAC Controller and moves to IDLE state.

12.6.2.15 Interacting with DMA Engine/Tx Parameter Cache

The DMA engine is responsible for reading data from Policy table, MAC Header descriptor and moving data to TX FIFO based on availability of TX FIFO. TX Controller starts reading data from TX FIFO after getting trigger from MAC Controller.

12.6.2.16 Getting triggers from MAC Controller and Timer block

The TX Controller is triggered by the MAC Controller based on the early SIFS and Slot triggers coming from the Timer block. The TX Controller also uses SIFS and SLOT indications from Timer block for requesting frame transmission to the MAC-PHY Interface.

12.6.2.17 Transmit finalization

The TX Controller reads bytes from the TX FIFO and pushes them to the Encryption Engine until the number of transmitted bytes reaches the number of bytes defined in *frameLengthTx* field of the Header Descriptor minus four.

If the field *dontTouchFCS* of the Header Descriptor is reset, the TX Controller trigs the FCS block to shift out the signature and does not push the last four bytes of the TX FIFO to the Encryption Engine. In this case, these last four bytes are discarded.

If the field *dontTouchFCS* of the Header Descriptor is set, the TX Controller does not trigs the FCS block to shift out the signature and does not push the last four bytes of the TX FIFO to the Encryption Engine. In this case, these last four bytes are pushed to the MAC-PHY Interface through the FCS Engine (as for the MAC Header).

When the FCS block has shifted out the last byte of the FCS, the TX Controller gets the *endOfFCS_p* indication.

If the transmission came from a *txControlFrame_req*, then the TX Controller sends the confirmation (*txControlFrame_conf*) to the MAC Controller.

If the transmission came from a *txFromFifo_conf* and if the frame was a single MPDU, then the TX Controller sends the confirmation (*txFromFifo_conf*) to the MAC Controller.

If the transmission came from a *txFromFifo_conf* and if the frame is an A-MPDU, then the TX Controller checks if the MPDU sent was the last MPDU of the A-MPDU or not

- If it was the last one, it sends the confirmation (*txFromFifo_conf*) to the MAC Controller.
- If the MPDU sent was not the last one, it restarts processing the next MPDU and indicates to the DMA Engine that a MPDU has been transmitted using *txMpduDone* signal.

If the transmission came from a *txFromSw_req*, then the TX Controller sends the confirmation (*txFromSw_conf*) to the MAC Controller.

Along with the confirmation, the TX Controller provides the transmission status (*txStatus*) to the MAC Controller according to the *Table 19: txStatus provided by the Tx Controller to the MAC Controller*.

There are several errors which might occur during a transmission.

The first one is a PHY error indicated by the *phyErr_p* pulse from the MAC-PHY F. In this case, the transmission is stopped, the TX Controller sends the *txFromFifo_conf* or *txControlFrame_conf* to the MAC Controller with the right *txStatus* and goes back to IDLE.

If the TX Controller detects a length mismatch between the frame length defined in the Header descriptor and the amount of data coming from the TX FIFO, then the transmission is finalized as following. The TX Controller will push to the MAC-PHY IF, the number of bytes defined in the *Frame Length* field of the header descriptor. However, the TX-FIFO will be popped until the TX FIFO Flag indicated the end of the MPDU.

- If the number of bytes contained into the TX FIFO is smaller that the frame length defined, the TX Controller does not push dummy data into the MAC-PHY IF. In this case, a PHY underrrun will be detected.
- If the number of bytes contained into the TX FIFO is bigger that the frame length defined, the TX Controller will not request the shift out of the FCS signature into the MAC-PHY IF. In this case also, a PHY underrrun will be detected.

txStatus value	Description
3'b000	Transmission OK
3'b001	Reserved
3'b100	Transmit aborted due to PHY Error
3'b101	Transmit Length Mismatch
3'b110	Reserved

Table 19: txStatus provided by the Tx Controller to the MAC Controller

The TX Controller does not update the MIB but provides all the information to the MAC Controller which will handle this task.

12.6.2.18 Flow Control

The TX Controller fetches data from the TX FIFO and pushes them to the MAC-PHY IF. As the MAC-PHY IF is slower than the DMA Interface, a flow control is implemented. This flow control is based on *dataValid*/*dataReady* signals. When a consumer can receive a new byte, it raises the *dataReady* to indicate to the provider that a new byte can be transmitted. When the provider pushes a new byte, it raises the *dataValid* to indicate to the consumer that a new byte has been pushed. The provider is not allowed to push new data into a consumer if the *dataReady* of this consumer is low.

Following the mechanism, the blocks part of TX Chain are connected together and process a byte every clock cycle until the MAC-PHY IF is full (*macPhyIFdataReady* = 0) or the TXFIFO is empty (*txFIFOdataValid = 0).*

12.6.3 TX Controller Logic block

12.6.3.1 Overview

The TX Controller Logic block performs the following functions based on current state of TX Controller and control signals from MAC Controller. All these functions are explained in the above sections.

Preparation and Transmission of Control and Special frames

A-MPDU transmission

Triggering MAC Header Generator

Triggering MAC-PHY Interface for transmission of frames

Triggering Crypto Engine

Triggering FCS block

12.6.3.2 State machine

The state machine below gives high level FSM for TX Controller. The tables below the picture explain different activities in each state and state transition conditions.

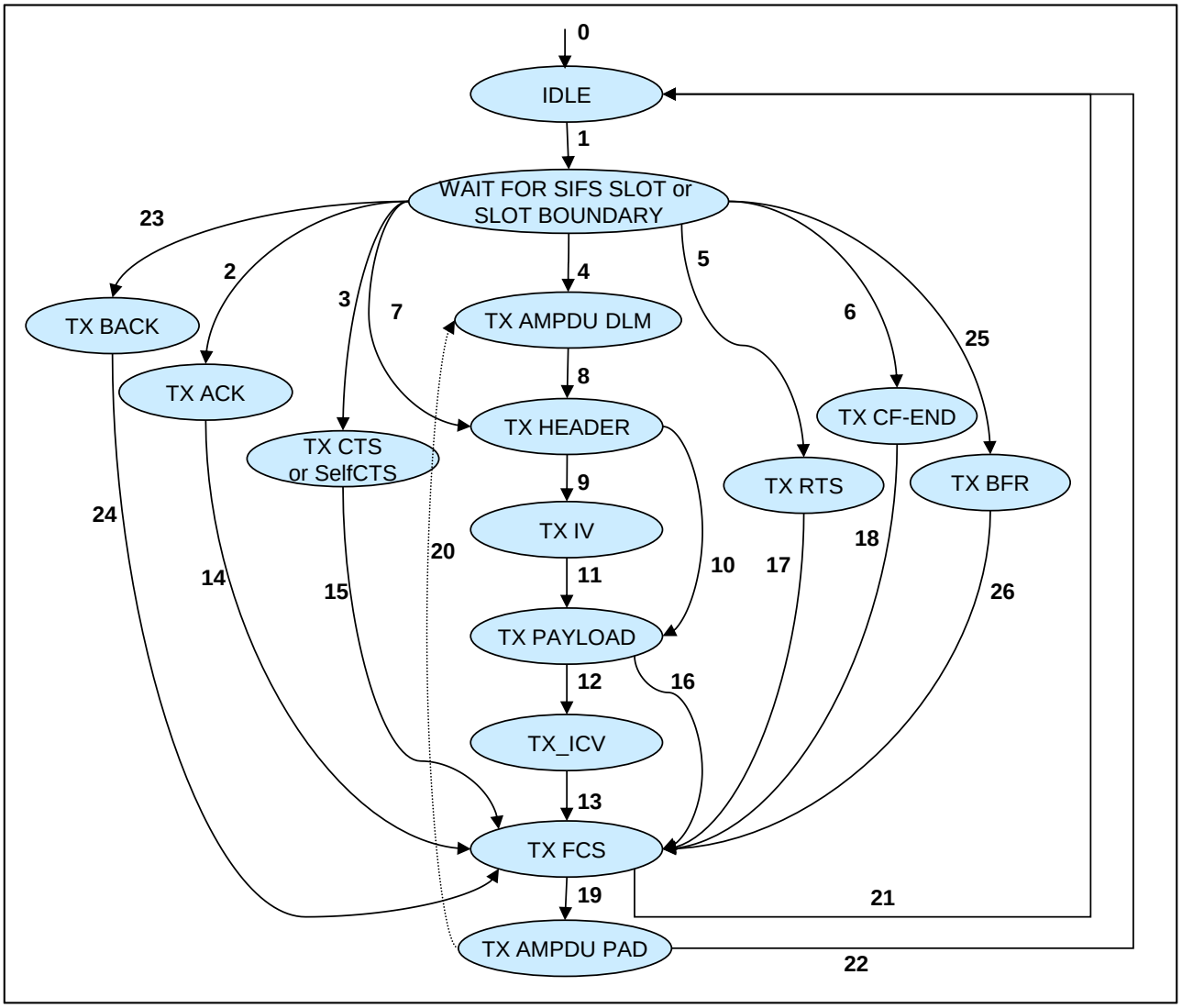


Figure 51: TX Controller State diagram

State Name	State Description
IDLE	In IDLE state, TX Controller is not transmitting any data to PHY IF. When the state machine enters IDLE state after any reset, all output signals are driven to default variables.
WAIT_FOR_SIFS_OR_SLOT	In this state, TX Controller waits for the respective slot times for transmission of frames. The possible transitions from this state are TX_AMPDU_DLM, TX_HEADER, TX_RTS, TX_CF-END, TX_CTS, TX_ACK, TX_BACK, TX_QoSNULL
TX_AMPDU_DLM	In this state, delimiter is transmitted if the frame is a part of Aggregated MPDU.
TX_HEADER	In this state, Header field bytes are transmitted, the KeySearch is running as well as the crypto engine initialization
TX_IV	Initialization vector, Extended Initialization vector are transmitted for an encrypted frame
TX_PAYLOAD	In this state, frame body bytes are transmitted.
TX_ICV	Integrity check value is transmitted for an encrypted frame

State Name	State Description
TX_FCS	In this state, FCS block is triggered and calculated FCS is transmitted. This state can be entered from TX_PAYLOAD, TX_ICV, TX_RTS,TX_CF-END, TX_CTS,TX_ACK, TX_BACK, TX_BFR
TX_AMPDU_PAD	In this state, padding bytes are transmitted if the frame is a part of an Aggregated A-MPDU, to make the length a multiple of 4
TX_CF-END	In this state, CF-End frame is prepared and transmitted.
TX_RTS	In this state, RTS frame is prepared and transmitted.
TX_CTS	In this state, CTS frame is prepared and transmitted.
TX_ACK	In this state, ACK frame is prepared and transmitted.
TX_BACK	In this state, Block ACK frame is prepared and transmitted.
TX_BFR	In this state, BeamForming Report frame is prepared and transmitted.

Table 20: TX Controller FSM States

Transition No.	State Description
0	Power reset or Soft reset
1	MAC Controller triggers the Transmission
2	Early SIFS boundary has occurred; Transmission of ACK is triggered by Protocol controller
3	Early SIFS boundary has occurred; Transmission of CTS is triggered by Protocol controller
4	Early SIFS or slot boundary has occurred; MAC Controller triggers the transmission of frames from Transmit FIFO; Frame is a part of an Aggregated MPDU.
5	Early Slot boundary has occurred; Transmission of RTS is triggered by Protocol controller
6	Early SIFS boundary has occurred; Transmission of CF-END is triggered by Protocol controller
7	Early SIFS boundary or slot boundary has occurred; MAC Controller triggers the transmission of frames from Transmit FIFO; Frame is not a part of an Aggregated MPDU.
8	Transmission of AMPDU Delimiter field bytes is completed
9	Transmission of Header field bytes is completed; Encryption enabled
10	Transmission of Header field bytes is completed; Encryption not enabled
11	Transmission of IV field bytes is completed and the Encryption Engine has been initialized
12	Transmission of Frame body field bytes is completed; Encryption enabled
13	Transmission of ICV field bytes is completed
14	Preparation and Transmission of ACK field bytes is completed
15	Preparation and Transmission of CTS field bytes is completed
16	Transmission of Frame body field bytes is completed; Encryption not enabled

Transition No.	State Description
17	Preparation and Transmission of RTS field bytes is completed
18	Preparation and Transmission of CF-END field bytes is completed
19	Appending and Transmission of FCS is completed; Frame is a part of an Aggregated MPDU
20	Padding completed; Frame is not the last frame in the Aggregate.
21	Appending and Transmission of FCS is completed; Frame is not a part of an Aggregated MPDU
22	Padding completed; Frame is the last frame in the Aggregate.
23	SIFS boundary has occurred; Transmission of Block ACK is triggered by Protocol controller
24	Preparation and Transmission of Block ACK field bytes is completed
25	SIFS boundary has occurred; Transmission of BeamForming Report is triggered by Protocol controller
26	Preparation and Transmission of BeamForming Report field bytes is completed

Table 21: TX Controller FSM State transition conditions

12.6.4 Timing diagrams

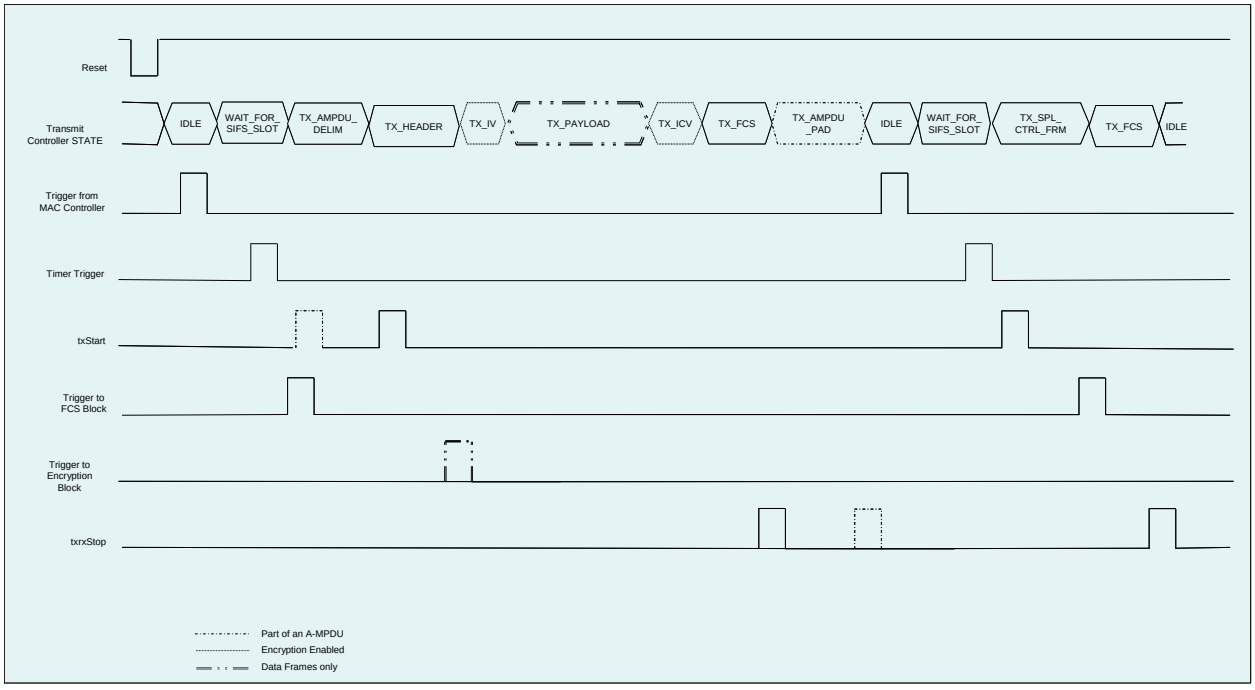


Figure 52: Transmit controller timing diagram

12.7 FCS

12.7.1 Overview

The FCS block is essentially a 32-bit CRC used for the FCS field calculation. This block is shared between TX and RX. It interfaces to RX Controller and TX Controller. *Figure 53: FCS block diagram* shows the interactions of the FCS block with the various blocks in the MAC HW.

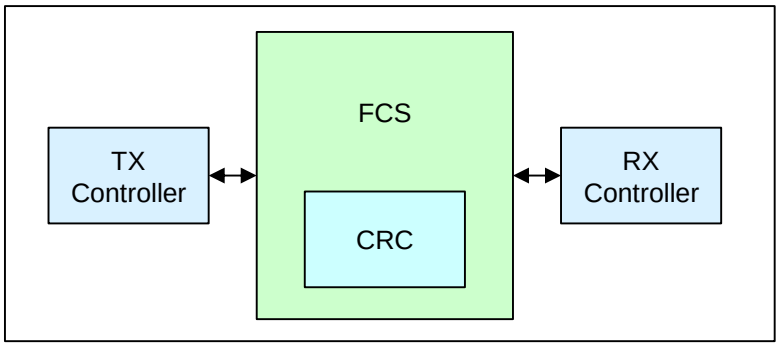


Figure 53: FCS block diagram

In case of MU-MIMO, this module is duplicated in the secondary paths.

12.7.2 Functional Description

The 32-bit CRC is performed using the following generator polynomial:

$$G(x) = x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1$$

The FCS calculation is triggered by *fcsStart_p* input signal.

For TX FCS generation, TX Controller triggers FCS block to generate FCS output via *fcsShift_p* input signal. When the 4-byte FCS has been delivered, the FCS block informs the TX Controller via its *fcsEnd_p* output signal.

For RX FCS checking, RX Controller enables FCS block. When the 4-byte received FCS has been checked with the internal calculated FCS, the FCS block informs the RX controller of the FCS correctness via its *fcsOk* output signals. If *fcsOk* is set to 1, it means that current frame has been correctly received else the frame is corrupted.

12.8 A-MPDU Deaggregator

12.8.1 Overview

The A-MPDU Deaggregator reads RX Vector from MAC-PHY Interface FIFO and triggers RX Controller to receive individual MPDU's. The Deaggregator block receives the individual A-MPDU delimiters and checks its CRC field. If the CRC passes, RX Controller is triggered. If the CRC fails, Deaggregator scans for the next A-MPDU delimiter, by reading data from the MAC-PHY IF FIFO. If received frame is a singleton MPDU, the A-MPDU Deaggregator passes directly the frame to the RX Controller. *Figure 54: A-MPDU Deaggregator block diagram* shows the interactions of the A-MPDU Deaggregator with the various blocks in the MAC HW. It operates in *macCore2Clk*.

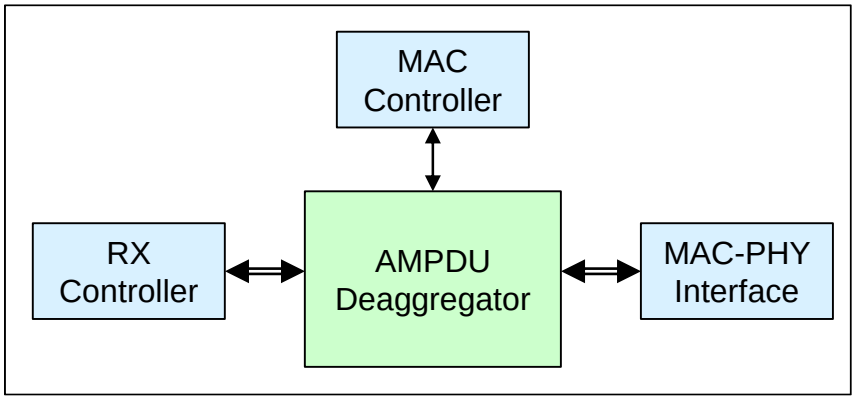


Figure 54: A-MPDU Deaggregator block diagram

12.8.2 State machine

The *Figure 55: A-MPDU Deaggregator State diagram* below gives high level FSM for A-MPDU Deaggregator. The *Table 22: A-MPDU Deaggregator FSM States* and *Table 23: A-MPDU Deaggregator State transition conditions* below explain different activities in each state and state transition conditions.

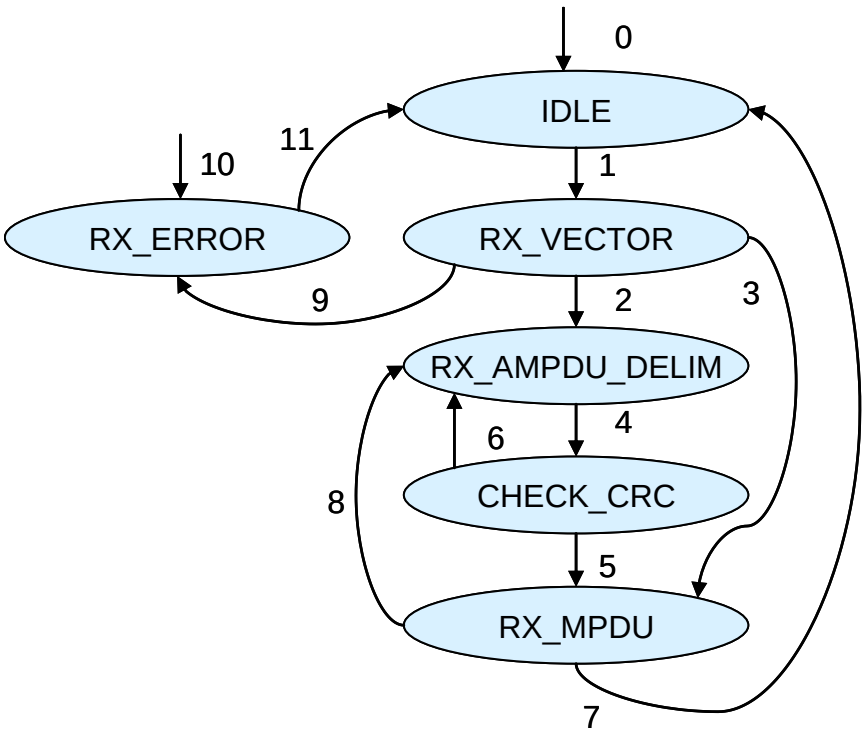


Figure 55: A-MPDU Deaggregator State diagram

State Name	State Description
IDLE	In this state, the state machine waits for trigger from MAC Controller.
RX_VECTOR	In this state, fsm reads Receive Vectors from MAC PHY IF FIFO and stores them in internal registers.
RX_AMPDU_DELIM	In this state, the state machine waits for reception of A-MPDU delimiter, computes the CRC and extracts the reserved, length and CRC fields.

State Name	State Description
CHECK_CRC	In this state, the state machine checks CRC of received A-MPDU Delimiter.
RX_MPDU	In this state, the state machine passes MPDU from MAC PHY IF FIFO to RX Controller.
RX_ERROR	In this state, the state machine waits *rxEnd_p* from PHY.

Table 22: A-MPDU Deaggregator FSM States

Transition No.	Transition Description
0	Power reset or Soft reset.
1	This transition occurs when MAC Controller triggers A-MPDU Deaggregator to start receive mode.
2	This transition occurs when the RX Vectors are read from MAC-PHY Interface FIFO and received frame is A-MPDU.
3	This transition occurs when the RX Vectors are read from MAC-PHY Interface FIFO and received frame is singleton MPDU.
4	This transition occurs once the A-MPDU Delimiter is received.
5	This transition occurs if the CRC check passes for received A-MPDU delimiter and this is not blank delimiter.
6	This transition occurs if the CRC check fails for received A-MPDU delimiter or this is a blank delimiter.
7	This transition occurs at the end of singleton MPDU or end of the entire A-MPDU.
8	This transition occurs at the end of MPDU reception and the frame is part of A-MPDU and this is not the end of the entire A-MPDU.
9	This transition occurs when length extracted from Receive vectors is null.
10	This transition occurs when MAC Controller aborts reception or when PHY sends a error on rxErr_p.
11	This transition occurs when PHY sends rxEnd_p.

Table 23: A-MPDU Deaggregator State transition conditions

12.9 Receive Controller

12.9.1 Overview

The Receive Controller (henceforth referred to as RX Controller) block handles the reception of the MPDUs, generates triggers for FCS and Decryption Engine (if necessary) before writing in the Receive FIFO. It triggers BA Controller for updating the Partial State Bitmap and NAV for updating virtual CS. On Beacon and probe response reception it updates the local TSF, DTIM Period and DTIM Count. It informs MAC controller of all frames received.

The RX Controller operates in *macCore2Clk* and interfaces to MAC-PHY interface, A-MPDU Deaggregator, CSR, BA Controller, Receive FIFO, MAC Controller, NAV, Timers for TSF update, FCS and Encryption engine blocks. *Figure 56: Receive Controller Block Diagram* shows the major sub blocks and the interactions of the RX Controller with the various blocks in the MAC HW.

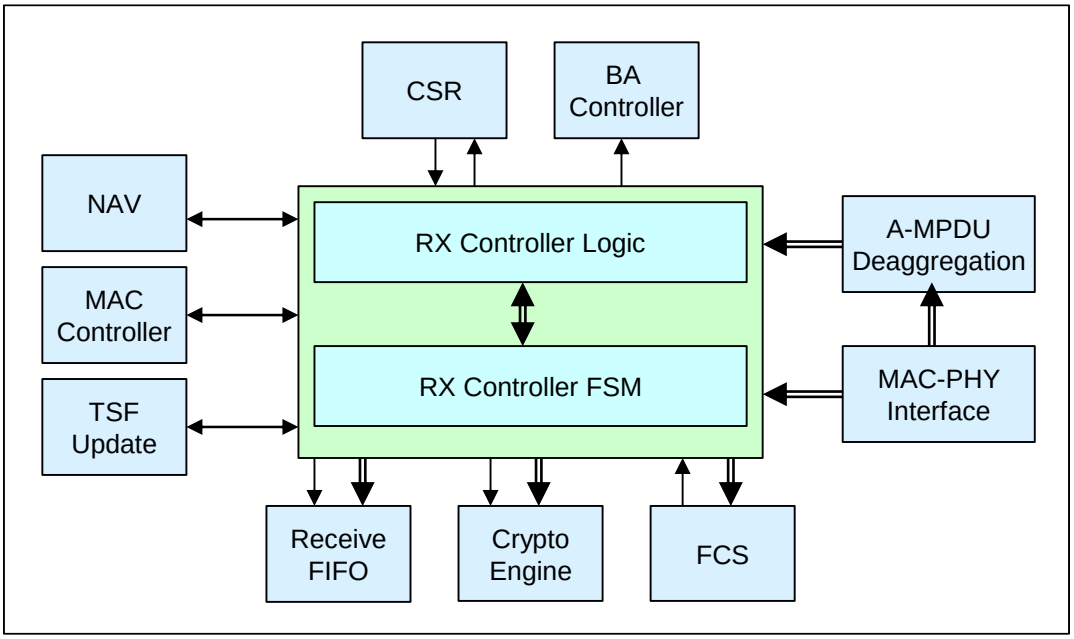


Figure 56: Receive Controller Block Diagram

12.9.2 Functional description

RX Controller is unaware of an A-MPDU. Hence the length of an MPDU carried in an A-MPDU is indicated directly from the A-MPDU Deaggregator block via Receive Vectors. All other PHY parameters, like rate, modulation are read from stored registers since they are common for each MPDU of an A-MPDU. Each received MPDU is parsed in RX Controller and written in the RX FIFO according to the Receive MPDU template defined in *[1]*. The RX controller follows the frame filtering rules defined in *[1]* and updates the RX TAG FIFO accordingly. Following are the main functions of RX Controller:

- *Writing received frame to Receive FIFO*
- *Triggering FCS block*
- *Triggering Encryption Engine*
- *Updating NAV*
- *Triggering TSF updates*
- *Updating spectrum management extension (11H)*
- *Triggering BA controller*
- *Indications to MAC Controller*
- *Extract information for BF Controller*

All the above functions of RX Controller are performed by the RX Controller State Machine along with the *Receive Controller Logic block.* Depending on current frame reception and validity of the frame, control signals for other blocks are generated.

12.9.2.1 Writing received frame to Receive FIFO

The Frame header is written to Receive FIFO directly irrespective of frame type. However, in case of Trigger frame, information is provided to the DMA using the specific Rx FIFO Tag 4'b1110. This specific tag indicates to the DMA that the frame which is being provided toward the RX FIFO is a Trigger frame.

At the end of the frame, RX-Vectors and trailing information are added to Receive FIFO to complete MPDU template for DMA.

For un-encrypted frames, frame is written directly to Receive FIFO. For encrypted frames, first the payload of the frame is written to Encryption Receive FIFO then Encryption engine writes the decrypted data to Receive FIFO. If *rxCntrlReg.dontDecrypt* bit is set, any received encrypted frames are directly written to Receive FIFO without decryption.

Depending on frame filtering rules defined in *[1]* (*rxCntrlReg*), if the frame has to be discarded, at the end of the frame, the Receive Tag FIFO is updated using the tag 4'b1111 to discard the frame. The RX Controller takes decision to discard or accept frames in the following conditions:

If *rxCntrlReg.acceptErrorFrames* bit is not set, any frame received with FCS Error or Invalid length or Invalid Protocol Version field etc is discarded.

If the *rxCntrlReg.excUnencrypted* bit is set, all un-encrypted MPDUs are discarded except Management frames.

If *rxCntrlReg.acceptMulticast* bit is not set, any Multicast address frame received is discarded.

If *rxCntrlReg.acceptBroadcast* bit is not set, any Broadcast address frame received is discarded.

If *rxCntrlReg.acceptOtherBSSID* bit is not set, any broadcast and multicast frame that contain a BSSID not matching the BSS's BSSID is discarded.

If *rxCntrlReg.acceptUnicast* bit is not set, any unicast frame not directed to this device is discarded.

If *rxCntrlReg.acceptMyUnicast* bit is not set, any unicast frames directed to this device is discarded.

If *rxCntrlReg.acceptProbeReq* bit is not set, any Probe request subtype frame is discarded.

If *rxCntrlReg.acceptProbeResp* bit is not set, any Probe response subtype frame is discarded.

If *rxCntrlReg.acceptBeacon* bit is not set, any Beacon subtype frame is discarded.

If *rxCntrlReg.acceptOtherMgmtFrames* bit is not set, any other Management subtype frame which is not explicitly covered (like *acceptProbeRes*, *acceptProbeResp*) is discarded.

If *rxCntrlReg.acceptBAR* bit is not set, any Block ACK Request subtype frame is discarded.

If *rxCntrlReg.acceptBA* bit is not set, any Block ACK Response subtype frame is discarded.

If *rxCntrlReg.acceptPSPoll* bit is not set, any PS Poll subtype frame is discarded.

If *rxCntrlReg.acceptRTS* bit is not set, any RTS subtype frame is discarded.

If *rxCntrlReg.acceptCTS* bit is not set, any CTS subtype frame is discarded.

If *rxCntrlReg.acceptACK* bit is not set, any ACK subtype frame is discarded.

If *rxCntrlReg.acceptCFEnd* bit is not set, any CF-End or CF-End + CF-ACK subtype frame is discarded.

If *rxCntrlReg.acceptOtherCntrlFrames* bit is not set, any other Control subtype frame which is not explicitly covered (like *acceptACK*, *acceptCTS*) is discarded.

If *rxCntrlReg.acceptData* bit is not set, any Data subtype frame like Data, Data + CF-ACK, Data + CF-Poll, Data + CF-ACK + CF-Poll is discarded.

If *rxCntrlReg.acceptCFWOData* bit is not set, any CF without data subtype frame like CF-ACK, CF-Poll, CF-ACK + CF-Poll is discarded.

If *rxCntrlReg.acceptQData* bit is not set, any QoS Data subtype frame like QoS Data, QoS Data + CF-ACK, QoS Data + CF-Poll, QoS Data + CF-ACK + CF-Poll is discarded.

If *rxCntrlReg.acceptQCFWOData* bit is not set, any QoS CF without Data subtype frame like QoS CF-Poll, QoS CF-ACK + CF-Poll is discarded.

If *rxCntrlReg.acceptQoSNull* bit is not set, any QoS Null subtype frame is discarded.

If *rxCntrlReg.acceptOtherDataFrames* bit is not set, any other Data subtype frame which is not explicitly covered (like *acceptQData*, *acceptData*) is discarded.

If *rxCntrlReg.acceptUnknown* bit is not set, any frame containing unknown Type is discarded.

If *rxCntrlReg.acceptBfmeeFrames* bit is not set, any NDPA, NDP and Beamforming Report Poll frame is discarded.

The RX Controller writes tag bits to RX TAG FIFO along with the received frame. The tag bit encoding is shown in *Table 5: RX FIFO Tags.*

12.9.2.2 Triggering FCS block

The RX Controller triggers FCS block at soon as A-MPDU deaggregator triggers RX Controller. Since FCS is calculated over both frame header and payload, at the end of payload reception RX Controller checks for FCS correctness. RX Controller generates *fcsStart_p* and *fcsEnable* control signals for the FCS block.

12.9.2.3 Triggering Encryption Engine

The RX Controller checks the *Protected Frame* bit in the *Frame Control* field in the MAC Header of the received MPDU. If this bit is set, receiver waits till Transmit Address and IV of the frame are received and triggers Key search engine to extract the decryption parameters from Key Storage RAM. Once key search is completed, the receiver triggers crypto blocks based on encryption type. The received data is written to Encryption RX Buffer continuously till the end of the frame. Once the Encryption engine is ready for decryption, it reads data from Encryption RX Buffer, performs decryption operation and writes output data to Receive FIFO. Once decryption is completed, decryption status is passed to RX Controller in order to update Receive FIFO with trailing information.

If *rxCntrlReg.dontDecrypt* bit is set, Encryption engine is not triggered.

12.9.2.4 Updating NAV

Please refer to section *12.2.2 Functional Description*.

12.9.2.5 Triggering TSF updates

As a station, RX Controller updates TSF from received Beacon frames and Probe response frames. The update happens only if the BSSID match happens and FCS check passes for received Beacon or Probe response frames.

12.9.2.6 Updating spectrum management extension (11H)

As a station, RX Controller updates Quiet IE (Quiet Count, Period, Duration and Offset) from received Beacon frames and Probe response frames. Update is needed only if the BSSID match happens and FCS check passes for received Beacon or Probe response frames.

12.9.2.7 Triggering BA controller

RX Controller triggers the BA controller with SN, TA-TID and reception status of the frame received for updating the Partial State Bitmap if the MPDU is directed to this device, if ACK policy is not set to NO-ACK and there is no error in reception.

12.9.2.8 Indications to MAC Controller

The RX Controller indicates the status of every MPDU reception to MAC Controller except in case of A-MPDU frame because MAC Controller could stop reception before the end of the entire A-MPDU. In case of A-MPDU, the reception status is indicated to MAC Controller only at the end of the entire A-MPDU. RX Controller signals whether a frame has been successfully or unsuccessfully received. If the frame is successfully received, it indicates the frame type and whether the frame was destined to this device. If the frame is not successfully received, it indicates the kind of error that occurred.

12.9.2.9 Extract information for BF Controller

When receiving a NDPA or Beamforming Report Poll frame, the RX Controller parses the different fields, check the RA or STA Index with AID register, indicates to the BF Controller and MAC Controller if the received frame is addressed to the device or not and extracts the following information required by the BF Controller:

RA : Received Address

TA : Transmit Address

Bme Sounding Token from NDPA

Nc Index and Feedback Type (from NDPA).

“STA Index is first” flag indicating that the RA matches the device MAC Address or if the device partial AID matches the first STA Info 1 field.

“Valid NDPA” flag indicating that the frame is a NDPA frame and either RA matches the device MAC Address or if the device partial AID matches one of the STA Info.

12.9.2.10 Extract information for Trigger Based transmission.

When receiving a Trigger frame, the RX Controller parses the different fields, check the RA or STA Index with AID register, indicates to MAC Controller if the received frame is addressed to the device or not and extracts the following information required by the MAC Controller:

Extracted field	Assigned signal name
Common field	
Trigger Type	rxTFTriggerType
Length	rxTFLength
Cascade Indication	rxTFCascadeInd
CS Required	rxTFCSReq
BW	rxTFBW
GI And LTF Type	rxTFGIAndLTFType
MU-MIMO LTF Mode	rxTFLTFMode
Number Of HE-LTF Symbols And Mid-amble Periodicity	rxTFNumLTF and rxTFMidamble
STBC	rxTFSTBC
LDPC Extra Symbol Segment	rxTFLDPCExtraSymb
AP TX Power	rxTFAPTxPower
Packet Extension	rxTFPE
Spatial Reuse	rxTFSR
Doppler	rxTFDoppler
HE-SIG-A	rxTFSIGA
User Info Field	
RU Allocation	rxTFRU
Coding Type	rxTFFecCoding
MCS	rxTFMCS
DCM	rxTFDCM
SS Allocation / Random Access RU Information	rxTFSSAlloc

Target RSSI	rxTFTRSSI

In case of MU-BAR variant, the rxController extracts also the BAR Control and BAR Information field from the Trigger Dependent User field and assigned to the rxBACtrl and rxBAInformation as for a standard BAR reception.

12.9.3 Receive Controller Logic block

The Receiver controller Logic block takes care of following functions based on current state of RX Controller and control signals from MAC Controller. All these functions are explained in the above sections.

- *Writing received frame to Receive FIFO*
- *Triggering FCS block*
- *Triggering Encryption Engine*
- *Updating NAV*
- *Triggering TSF updates*
- *Updating spectrum management extension (11H)*
- *Triggering BA controller*
- *Indications to MAC Controller*
- *Extract information for BF Controller*

The RX data flow control is shared by RX controller and Encryption Engine which interface MAC-PHY Interface FIFO and RX FIFO. Handshaking between each module is performed via *dataReady* and *dataValid* signals which indicates respectively that modules can accept new data and that data is valid for being sampled.

12.9.4 State machine

The *Figure 57: RX Controller State diagram* below shows the high level FSM for RX Controller. The *Table 24: RX Controller FSM States* and *Table 25: RX Controller State transition conditions* below explain different activities in each state and state transition conditions.

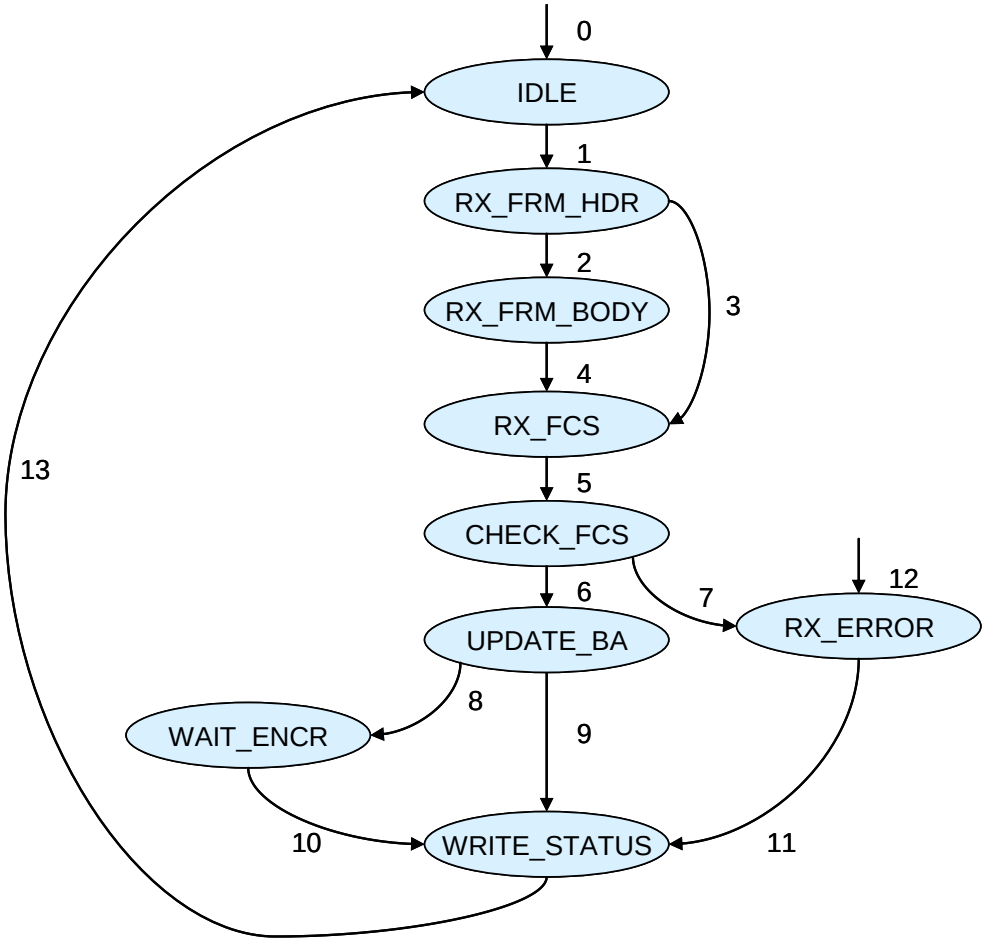


Figure 57: RX Controller State diagram

State Name	State Description
IDLE	In this state, RX controller is not receiving any data from A-MPDU Deaggregator. All output signals are driven to default values.
RX_FRM_HDR	In this state, RX controller receives MAC header and starts frame parsing. RX frame is written in the RX FIFO and TAG FIFO is updated accordingly.
RX_FRM_BODY	In this state, RX controller receives frame body of MPDU. If MPDU is encrypted, Encryption Engine is triggered and encrypted data are written in the Encryption Receive buffer.
RX_FCS	In this state, RX controller receives last 4 FCS bytes of the MPDU.
CHECK_FCS	In this state, RX controller checks for FCS correctness and informs MAC Controller about the frame received.
UPDATE_BA	In this state, RX controller triggers BA controller for updating the Partial State Bitmap.
WAIT_ENCR	In this state, RX controller waits end of decryption from Encryption Engine.

State Name	State Description
RX_ERROR	In this state, RX controller updates error status for coming trailing information writing.
WRITE_STATUS	In this state, RX Controller writes trailing information in the RX FIFO if the frame has to be passed to SW else it discards the frame via Tag RX FIFO writing.

Table 24: RX Controller FSM States

Transition No.	Transition Description
0	Power reset or Soft reset
1	A-MPDU Deaggregator triggers RX Controller to receive any frame. FCS block is triggered to start CRC calculation.
2	Frame header is received and frame type and length indicates frame body is present for the frame.
3	Frame header is received and frame type and length indicates frame body is not present for the frame.
4	Frame body reception is completed.
5	Frame FCS reception is completed.
6	FCS check is successful.
7	FCS error.
8	Frame is encrypted.
9	Frame is not encrypted.
10	End of decryption.
11	Error status has been updated.
12	Error from PHY or abort from MAC Controller.
13	RX FIFO writing completed with trailing information or discard pattern.

Table 25: RX Controller State transition conditions

12.9.5 Timing diagrams

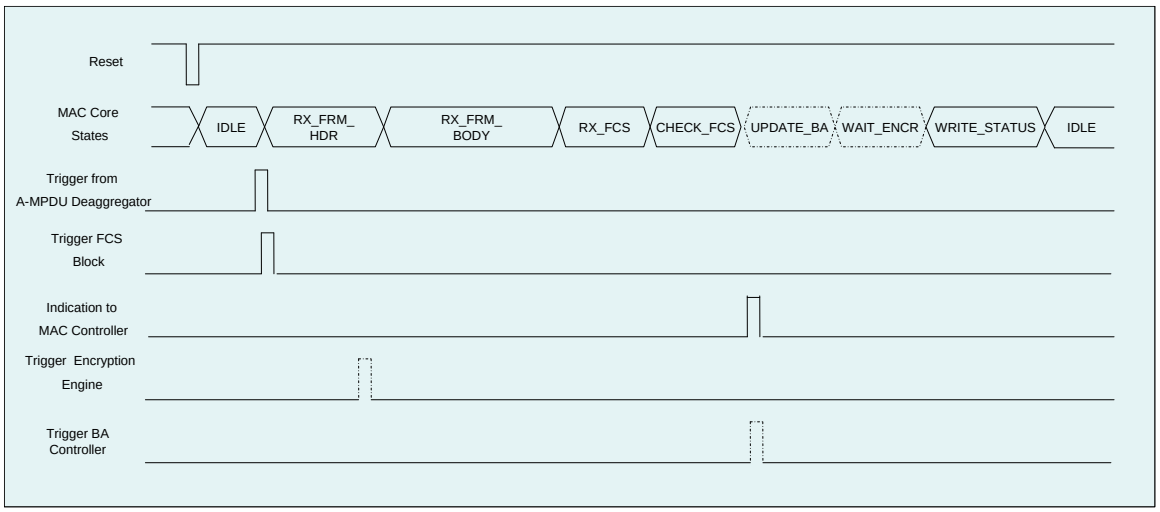


Figure 58: Receive controller timing diagram

12.10 Block ACK Controller

12.10.1 Functional Description

The Block ACK controller (henceforth referred to as BA Controller) block maintains Partial State Bitmap for the received MPDUs. Whenever RX Controller triggers this block with Sequence number, MAC Address of transmitter, the Block ACK bitmap is updated. When triggered by TX Controller, it provides the bitmap for the requested TA-TID pair. It then passes the formed bitmap when requested by the TX Controller for the transmission of BA. The BA Controller operates in *macCoreClk* and interfaces to MAC Controller, RX Controller, TX Controller and Key Search Engine. *Figure 59: Block ACK Controller Block Diagram* shows the major sub blocks and the interactions of the BA Controller with the various blocks in the MAC HW.

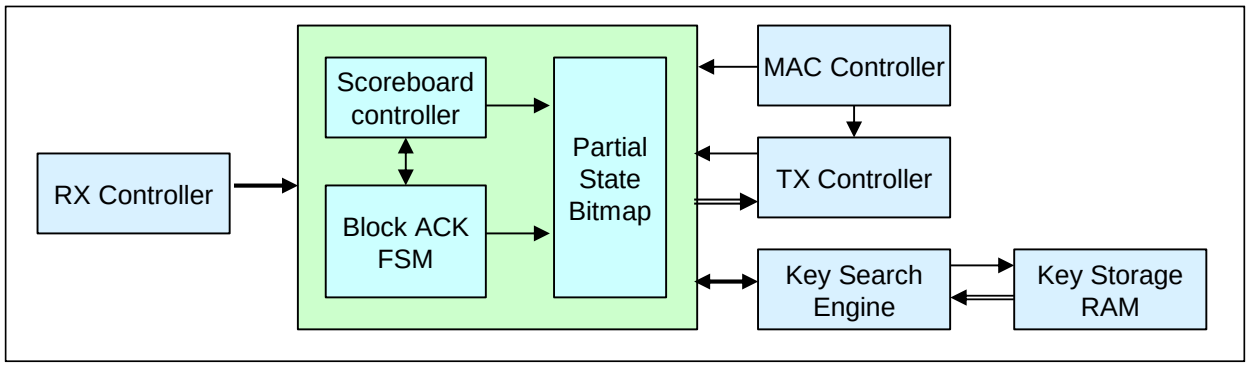


Figure 59: Block ACK Controller Block Diagram

12.10.2 Block ACK Controller FSM

The *Figure 60: Block ACK Controller State Machine* below gives high level FSM for BA Controller. The *Table 26: Block ACK Controller FSM States* and *Table 27: Block ACK Controller FSM State transition conditions* below explain different activities in each state and state transition conditions.

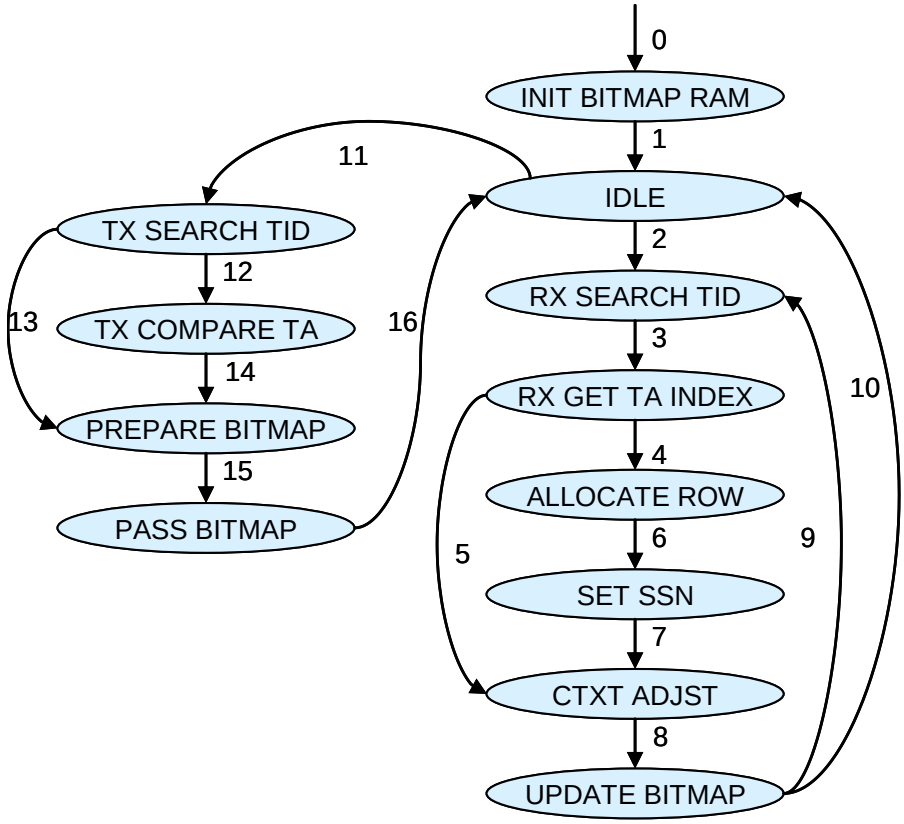


Figure 60: Block ACK Controller State Machine

State Name	State Description
INIT_BITMAP_RAM	In this state, BA controller initializes all the contents of the partial state bitmap RAM to 0.
IDLE	In this state, BA controller waits for trigger from either RX Controller or MAC Controller.
RX_SEARCH_TID	In this state, TID of the received MPDU is searched.
RX_GET_TA_INDEX	In this state, index of the Transmitter Address is found. If TID match is found, the received TA is first compared with the MAC Address pointed in the Partial state bitmap. If Address match is not found, Key Storage RAM is searched for the index. If no match for TID is found, index for the received TA is obtained by searching the Key Storage RAM.
ALLOCATE_ROW	In this state, a new row in the partial state bitmap is allocated if no match for TA-TID pair is found. If no row is empty, the oldest entry is taken.
SET_SSN	In this state, sequence number of the first received MPDU is set as SSN in the Score Board Controller.
CTXT_ADJUST	In this state, received SN is passed to Scoreboard Controller which compares and adjusts the context window for the bitmap.
UPDATE_BITMAP	In this state, partial state bitmap is updated as per the received SN.
TX_SEARCH_TID	In this state, TID given by the MAC Controller is searched.
TX_COMPARE_TA	In this state, TA given by the MAC Controller is compared with the MAC Address pointed by the index to KEY Storage RAM.
PREPARE_BITMAP	In this state, if TA-TID match is found, bitmap is prepared from the partial state bitmap. Otherwise bitmap is prepared with all bits set to zero.
PASS_BITMAP	In this state, prepared bitmap is passed to TX Controller.

Table 26: Block ACK Controller FSM States

Transition No.	State Description
0	Power reset or Soft reset.
1	Initialization of RAM completed.
2	Trigger is received from RX Controller.
3	TID Search is completed.
4	Transmitter Address index Search is completed and TA-TID match not found.
5	Transmitter Address index Search is completed and TA-TID match found.
6	Allocation of new row is completed for the TA-TID pair.
7	Starting Sequence Number (SSN) is set.
8	Adjusting score board context completed.
9	Bitmap is updated and next MPDU reception is indicated.

Transition No.	State Description
10	Bitmap is updated and last MPDU reception is indicated.
11	Trigger is received from MAC Controller.
12	TID Search is completed and match found.
13	TID search completed and match not found.
14	Transmitter Address index Search is completed.
15	Preparation of bitmap completed and Ready indication from TX Controller is received.
16	Bitmap pass to TX Controller is completed.

Table 27: Block ACK Controller FSM State transition conditions

12.10.3 Scoreboard Controller

This block handles the context window adjustment for the bitmap i.e., it sets the relative position in the bitmap based on the incoming MPDU SN (Sequence Number) and sets or resets bits accordingly. FSM passes the SN of each received MPDU. SSN is initially set to the SN of first received MPDU. Each received SN is compared with the existing SSN. The SSN may be shifted backwards when an MPDU contains a SN that is lower than SSN. This can happen due to out of order MPDUs or because a previous MPDU transmission with lower SN is not successful.

For each successfully received MPDU, the Scoreboard Controller checks the SN and performs one of the following:

If MPDU SN >= SSN, bitmap at (SN - SSN) is set to 1

If MPDU SN > SSN + 64, SSN is set to SN-32+1, the bitmap at [SN-SSN] is set to 1

If MPDU SN < SSN, SSN is set to SN, the bitmap at SSN is set to 1

12.10.4 Partial State Bitmap

This is a memorty of `RW_PSBITMAPSIZE x 88 bits. It can hold up to `RW_PSBITMAPSIZE TA-TID combinations. The structure of the RAM is as shown in the following *Figure 61: Partial State Bitmap RAM Structure*.

Location	Index (8 bit)	TID (4 bit)	SSN (12 bit)	Bitmap (64 bit)
0				
1				
2				
3				

Figure 61: Partial State Bitmap RAM Structure

64 bits are allocated for the bitmap, 12 bits for Starting Sequence Number, 4 bits for TID and 8 bits for the Index to Key Storage RAM.

When a MPDU is received, FSM looks into the Partial State bitmap for the TID match. If TID match is found, FSM triggers Key Search Engine with the index to match the Transmitter MAC Address. If TA match is found, bitmap is updated in the same row.

If the MAC address match is not found for a matched TID, FSM checks further locations.

If no TID match is found, then the TID is overwritten at which the current write pointer is present. Key storage ram is searched for TA to get the Index and then Score Board Controller updates the bitmap.

When trigger is received from MAC Controller, FSM first looks for TID match. If the match is found, then TA is compared with the MAC Address pointed by the index to KEY Storage RAM. If the match is found, bitmap is prepared from the partial state bitmap and sent to TX Controller. If TID-TA match is not found, bitmap is prepared with all bits set to zero.

12.10.5 Timing diagrams

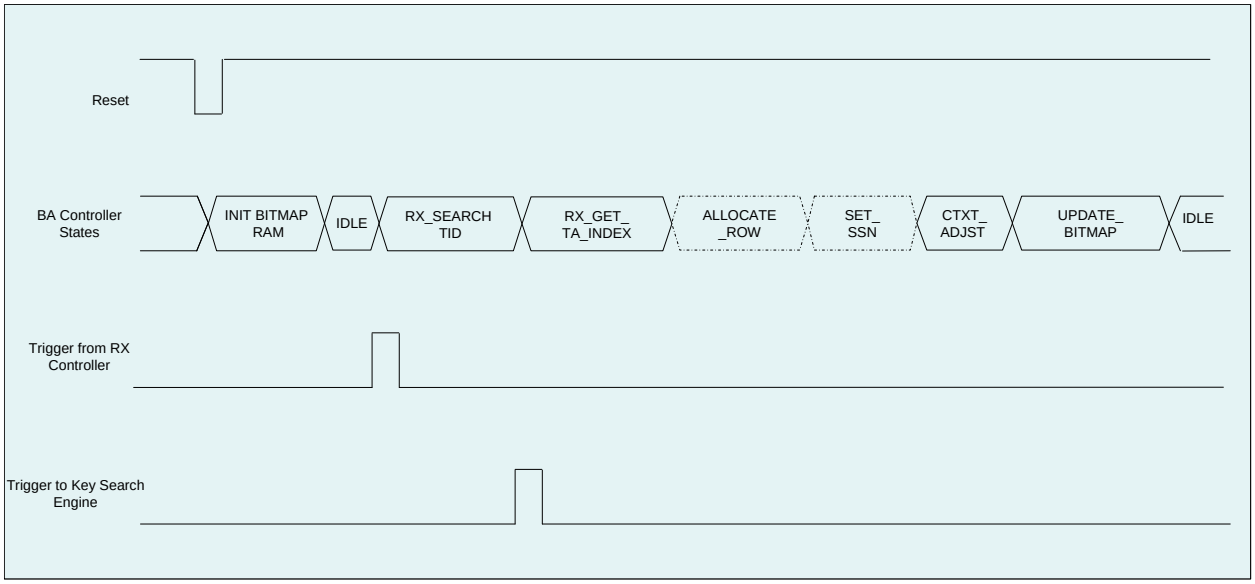


Figure 62: Block ACK Control bitmap update

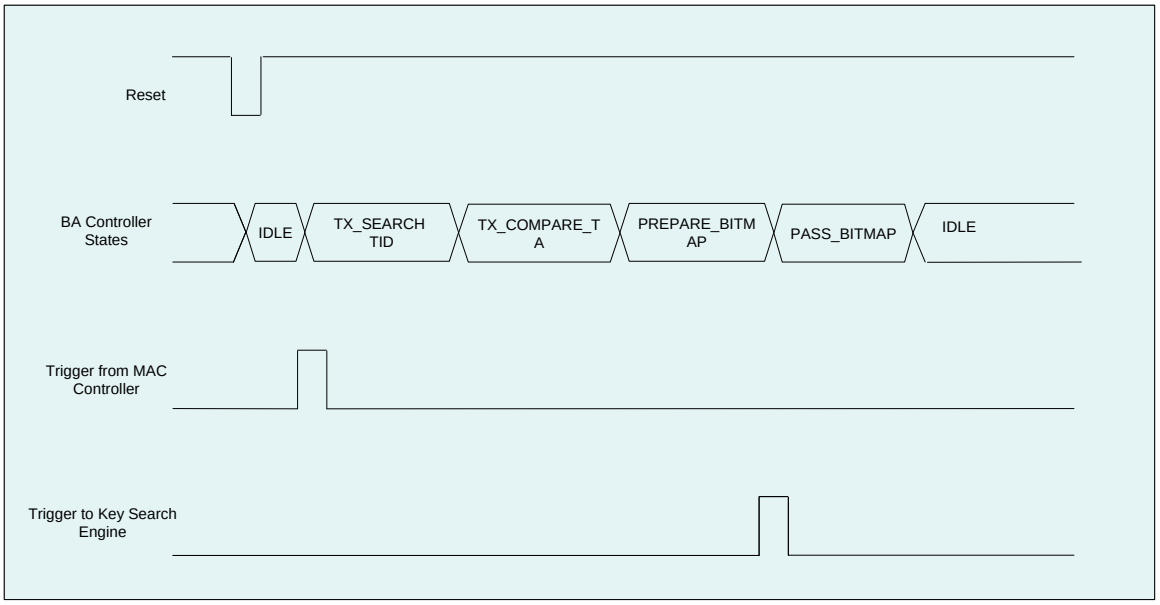


Figure 63: BA bitmap preparation

13 Encryption Engine

13.1 Functional Description

The Encryption Engine block handles the encryption of the data to be transmitted and performs the decryption on the received encrypted data. The Encryption engine interfaces to TX Controller, RX Controller, BA Controller, CSReg, Policy Table, Receive FIFO and FCS blocks.

When an MPDU is to be encrypted, TX controller will trigger the Encryption Engine. When the frame is ready to be transmitted by the TX controller, it passes the MPDU payload through Encryption Engine block. Encrypted frame is passed to the FCS block.

When an encrypted MPDU is to be decrypted, RX controller will trigger the Encryption Engine. It performs decryption and passes the decrypted MPDU to Receive FIFO.

The picture below shows major blocks in Encryption Engine and its interaction with other modules.

Note that WPI Engine is an optional module.

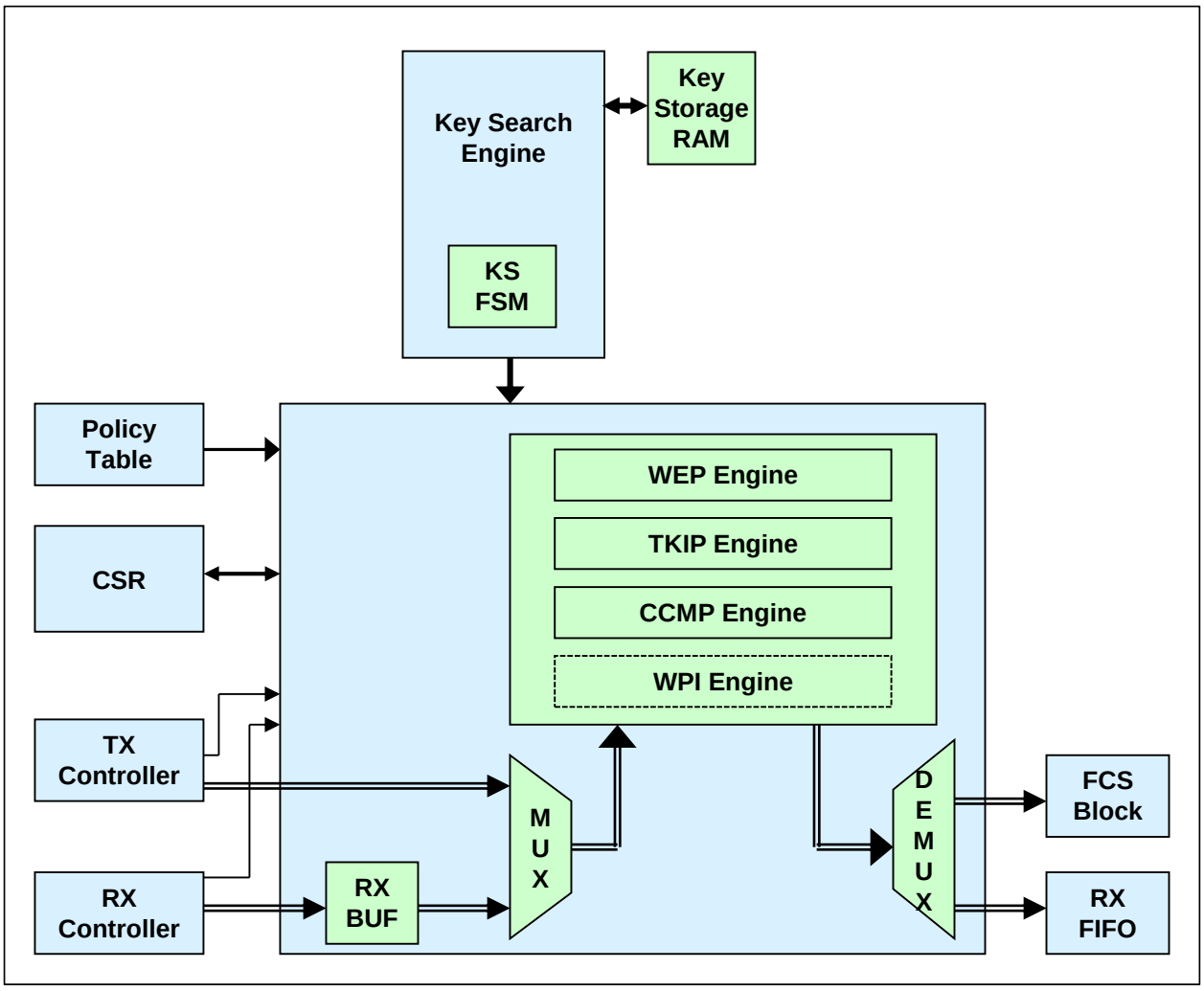


Figure 64: Encryption Engine Block Diagram

13.1.1 Encryption Flow

Key Search Engine is triggered by TX Controller to read the required encryption parameters from Key Storage RAM. The Tx controller gives the keyIndexRAM to the key search engine. Based on encryption type required, corresponding

Encryption block (WEP/TKIP/CCMP/WAPI) is selected. Payload from TX Controller and Key from the Key Search Engine are passed to Encryption Block. The encrypted data is passed to FCS Block after which it is written into MAC-PHY Interface FIFO.

13.1.2 Decryption Flow

Key search engine is triggered by RX Controller to extract Encryption Key from Key Storage RAM. The received data is written to Encryption RX Buffer till the Key search and encryption engine initialization is completed. Once the Encryption engine is ready for decryption, Key from the Key Search Engine and data from Encryption RX Buffer are fed to corresponding Encryption block. Decrypted data is written to Receive FIFO.

13.1.3 Debug Mode

For debugging purposes, the Key Storage RAM can be read back using the encryption registers. When *readKey* bit set by SW, Key Search Engine reads the *indexRAM* from the encryption registers. It then copies the contents of the Key Storage RAM pointed by *indexRAM* field into the corresponding fields of the encryption registers. When the read is complete, Key Search Engine resets the *readKey* bit. The previously described procedure is only available when *debugKSR* bit is set to one. When this bit is set to zero, the key search engine does not write the encryption key to the registers.

13.2 WEP Engine

The WEP Engine encrypts the data using the RC4 algorithm from RSA Corporation, thus ensuring security of the transmitted data over the wireless medium. It similarly decrypts received data. WEP is a symmetric algorithm.

13.2.1 WEP encryption procedure

WEP Engine applies transformations to the plaintext MPDU. It computes the ICV over the plaintext data and appends this after the MPDU data. WEP Engine encrypts the MPDU plaintext data and ICV with Key stream. Key stream is generated from pseudo-random number generator using the seed constructed from IV and Key. The ICV is computed and appended to the plaintext data prior to encryption.

13.2.2 WEP decryption procedure

WEP Engine applies transformations to the received WEP MPDU to decrypt its payload. It extracts IV and Key identifier from the received MPDU. If a Key-mapping Key is present for the <TA, RA> pair, then this shall be used as the WEP Key. Otherwise, the Key identifier is extracted from the Key ID subfield of the WEP IV field in the received MPDU, identifying the default Key to use. Key stream is constructed to decrypt the Data field of the WEP MPDU resulting in plaintext data and an ICV. Finally WEP Engine re-computes the ICV and bit-wise compares it with the decrypted ICV from the MPDU. If the two are bit-wise identical, then WEP Engine removes the IV and ICV from the MPDU. If they differ in any bit position, WEP Engine generates an error indication to RX Controller.

13.3 TKIP Engine

The TKIP algorithm enhances the WEP by using the Phase I and Phase II Key mixing functions to generate the seed for pseudo-random number generator used in WEP.

13.3.1 TKIP Encryption Procedure

The TX controller triggers the Key search for the destination address. Once the Key is found, the TKIP Engine is triggered. The TKIP Engine implements the phase I and phase II Key mixing functions to manipulate the TSC, the transmitter

address and the temporal Key to generate a 128-bit WEP seed to be used by the pseudo-random number generator module.

13.3.2 TKIP Decryption Procedure

The Key Search Engine is triggered by RX Controller as soon as the IV, EIV (if required) is received. After the Key search, TKIP Engine is triggered. The TKIP module follows the same procedure as transmission to generate the 128-bit WEP seed.

13.4 CCMP Engine

CCMP engine encrypts the transmitting data using AES (Advanced Encryption Standard). Similarly CCMP block decrypts the received data.

CCMP input register takes a stream of 8-bit data input (during receive as well as transmit), converts it into 128-bit and gives it to CCM block. AAD and nonce are constructed using the MAC Header and given to CCM block. CCM block encrypts or decrypts the data. The Key for CCM block is given by the Key Search Engine. The encrypted /decrypted data is fed to CCMP output register block. The output register block converts 128-bit data into the stream 8-bit data which is written into FCS block / receive FIFO. The CCM block is also responsible for comparing the calculated MIC with the received MIC.

13.5 WPI Engine

WPI Engine encrypts the transmitting data using SMS4 cipher suite. Similarly WPI block decrypts the received data. The WPI Engine uses two SMS4 blocks. One in OFB symmetric mode of operation to encrypt and decrypt data. The other in CBC-MAC mode of operation in order to perform the integrity check of the data by generating a 128-bit MIC.

A first 128-bit key named encryption key is used for OFB block. A second 128-bit key named integrity key is used for CBC block.

13.5.1 WPI Encryption Procedure

The TX controller triggers the Key search for the destination address. It also prepare all specific fields needed by WPI encryption procedure : 128-bit PN (Packet Number), 8-bit KeyIdx (Key Index) and some parts of the MAC-Header field. Once the Keys are found and all the parameters are ready, the WPI Engine is triggered. The WPI Engine encrypts plain data and resulting MIC and appends the encrypted MIC after encrypted data.

13.5.2 WPI Decryption Procedure

The Key Search Engine is triggered by RX Controller as soon as the transmitter address is received. Similarly to the encryption procedure, the Rx Controller prepares the PN, KeyIdx and MAC header fields. After the Key search and WPI Header reception, WPI Engine is triggered. The WPI Engine decrypts the received data and MIC. It finally checks the received MIC and indicates to Rx Controller if the integrity check comparison has passed or failed.

13.6 RX Buffer

This module interfaces with Rx Controller and Multiplexer block. This Two Port RAM has two independent ports and allows shared access to a single memory space. The size of RAM is 128 x 8, and is used for buffering of received encrypted data from RX Controller. It allows the buffurization of the payload during the key search procedure and crypto engine initialization without impacting the MAC-PHY.

13.7 Flow Control

The encryption engine implements flow control for both Rx and Tx.

For the Tx, the encryption engine receives a full signal from the FCS. This signal indicates to the encryption engine it can feed FCS with data. The FCS samples data as soon as the encryption engine provides a data enable. On the other side the encryption engine provides the same full signal to the TX Controller. The state of this signal is defined by the internal pipe's state of the encryption engine. The TX Controller cannot write data to the encryption engine and a full is asserted.

Concerning the Rx, The same mechanism is used on the FCS side. For the RX Controller side, a bigger flexibility is given to the RX Controller with the use of the Rx buffer. Flow control is based on the same signaling method.

13.8 Timing diagrams

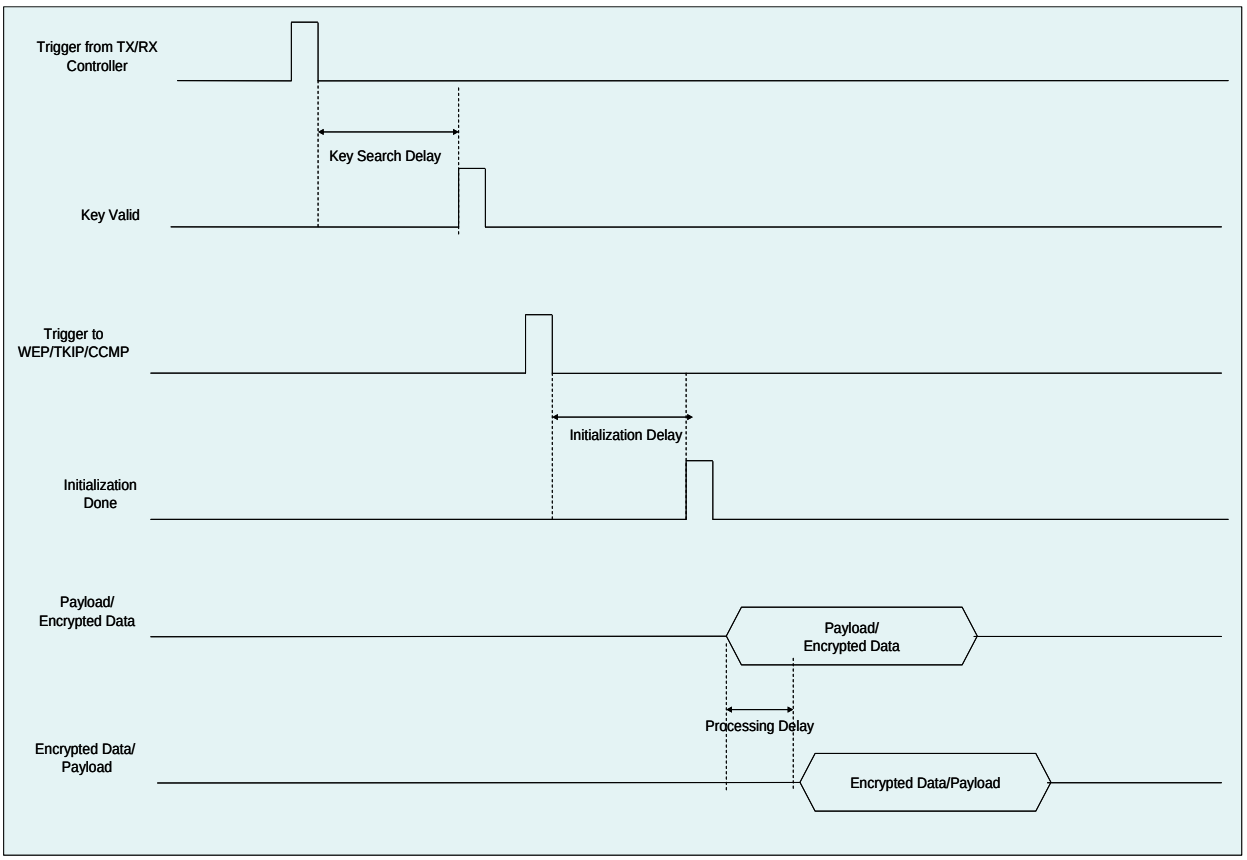


Figure 65: Encryption/Decryption process timing

13.9 MU-MIMO Support

In MU-MIMO, each user stream can be encrypted using CCMP. As MU-MIMO users are independent and may have different data-rate, the CCMP encryption is duplicated per secondary path as shown on the *Figure 66: Encryption Engine on secondary path for MU-MIMO support.*

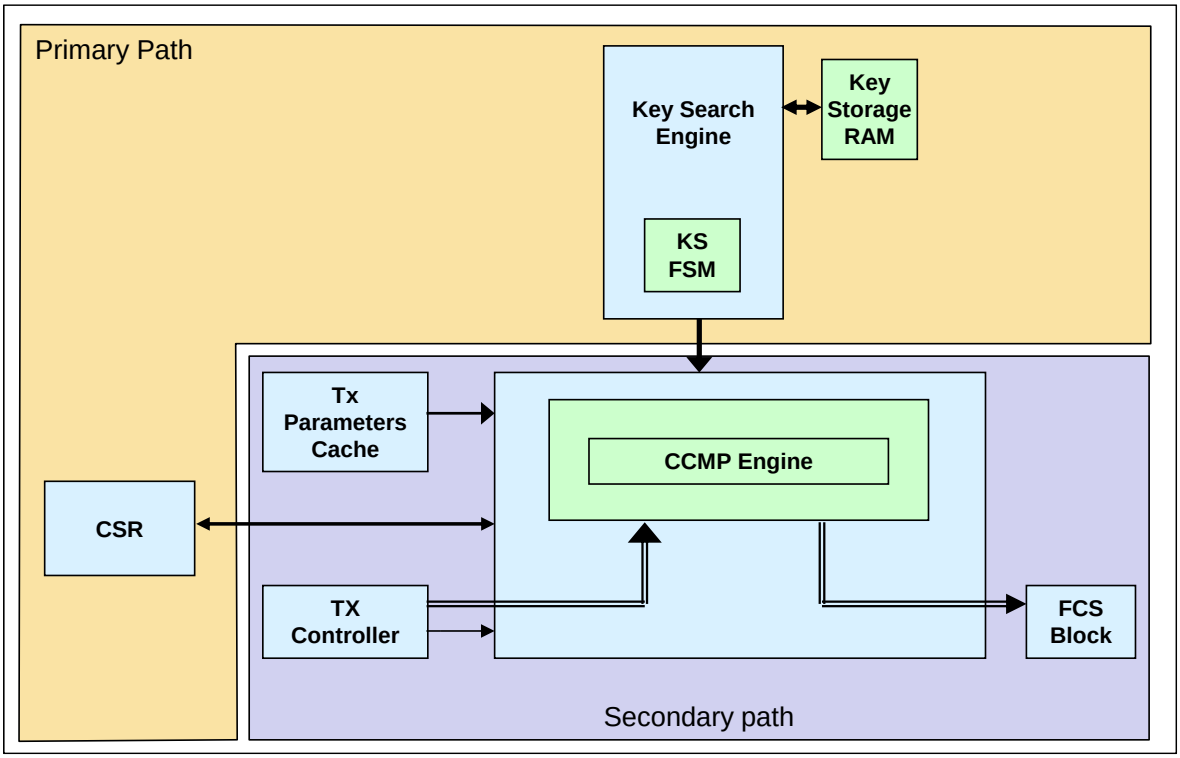


Figure 66: Encryption Engine on secondary path for MU-MIMO support

The Secondary Encryption Engine includes only the CCMP Engine. The WEP, TKIP and WPI encryption are forbidden on MU-MIMO. The CCMP Engine is exactly the same than in the primary encryption engine, only its interface has been changed to keep only the transmit path. The flow control, encryption and MIC computation is as described in *13.1.1 Encryption Flow, 13.4 CCMP Engine* and *13.7 Flow Control.*

14 Key Search Engine

Key Search Engine is responsible for Encryption Key management and MAC Address search for Immediate Block ACK support. MAC Address search for N-Immediate Block ACK is also made part of this module so that the memory usage is reduced. Other major functions of this block are Key search, Key update and IV generation for WEP.

For transmission of encrypted frames, Encryption Key is taken from the Key Storage RAM pointed by Key Storage RAM Index in Policy Table. For decryption of frames, the MAC address of received frame is searched for match in Key Storage RAM.

MAC software writes MAC Address and corresponding Keys to Key Storage RAM through CSR irrespective of whether the associated STA supports encryption or not. When a particular device is removed, its MAC address is overwritten by all 0's through CSReg interface. Priority is given for Key search over the Key update.

The state diagram below gives high level overview of Key Search Engine state machine. The tables below the picture explain different activities in each state and state transition conditions.

The key search engine starts the search at the index of the last found key. As the probability for two transmissions/receptions to be for the same peer device, it would reduce the research time. For the first research the key search engine starts at the first index.

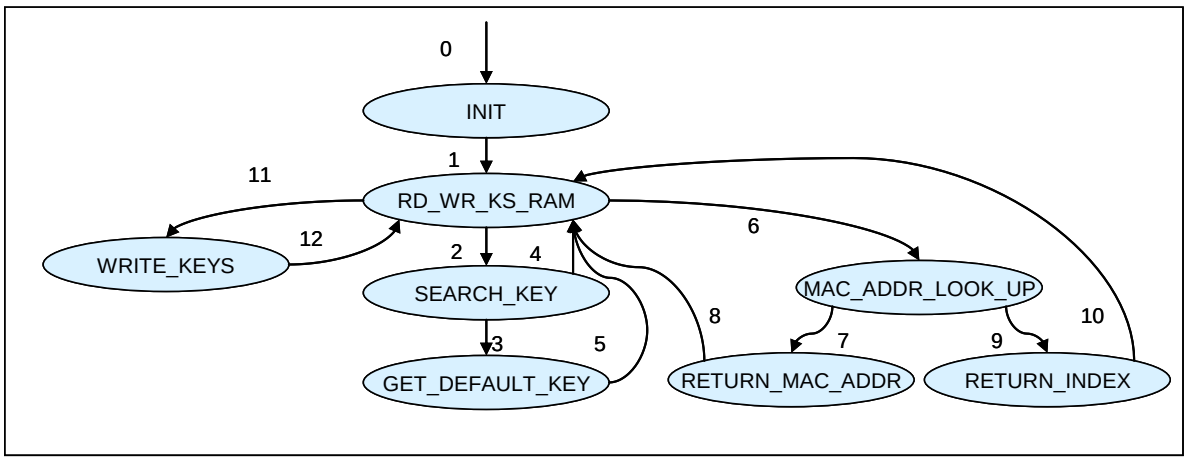


Figure 67: Key Search Engine State Machine

State Name	State Description
INIT	After reset, FSM enters this state. In this state, all locations are programmed with default values. This is all 1's for the MAC Address field, all 0's for the Key field.
RD_WR_KS_RAM	In this state FSM updates the KEY Storage RAM with the encryption registers when *writeKey* bit is set. When *readKey* bit is set, FSM reads the contents from the Key Storage RAM. It remains in this state until it gets a trigger from TX Controller or RX Controller or BA Controller.
SEARCH_KEY	FSM searches the Key storage RAM for the required Key using the 48-bit MAC Address as search index or Key is selected from the Key Storage RAM using Key Storage RAM Index in Policy Table.
GET_DEFAULT_KEY	A default Key is selected corresponding to Default Key ID.
MAC_ADDR_LOOK_UP	FSM enters this state when BA Controller triggers and gives Index requesting the MAC Address or the FSM enters this state when BA Controller triggers and gives MAC Address requesting its index.

State Name	State Description
RETURN_MAC_ADDR	MAC Address corresponding to the Index is returned to BA Controller in this state.
RETURN_INDEX	Index for the corresponding MAC Address is returned to BA Controller after search in this state.
WRITE_CSR	Contents of Key Storage RAM corresponding to the *indexRAM* are written in to CSReg.

Table 28: Key Search Engine FSM States

Transition No.	State Description
0	Power reset or Soft reset.
1	Initialization of Key Storage RAM is completed.
2	Trigger from TX Controller or RX Controller received
3	Searching of Key Storage RAM is completed and match not found.
4	Searching of Key Storage RAM is completed and match found or Key selection using index from policy table is done.
5	Selection of Default Key is completed.
6	Trigger from BA Controller is received requesting MAC Address corresponding to the given Index or Trigger from BA Controller is received requesting the Index corresponding to the given MAC Address.
7	MAC Address lookup in the Key Storage RAM is completed.
8	Returned MAC Address to BA Controller.
9	MAC Address search and finding its index is completed.
10	Returned Index to BA Controller.
11	*readKey* bit is set and reading the contents of Key storage RAM corresponding to the *indexRAM* is completed.
12	Writing into CSReg is completed.

Table 29: Key Search Engine FSM State transition conditions

14.1 Key Storage RAM

The per-MAC address and group address encryption keys are stored in an embedded HW RAM, called the Key Storage RAM. The address locations 0 - 3 of the Key Storage RAM are reserved for storing the default key IDs 0, 1, 2 and 3 respectively. The remaining 60 locations of the RAM holds address mapped encryption keys.

SW programs the Key, Cipher length, Cipher type and MAC Address into the Encryption Registers. Key Search Engine copies the contents of the encryption fields into the Key Storage RAM location as pointed by the Policy Table.

A SW API is provided through register interface which allows the SW to request a search inside the KeyRAM using the same mechanism than for a HW triggered search. As both could occur simultaneously, an arbitration mechanism has been implemented based on first request first response which gives the priority to the HW in case of request within the same clock cycle.

In case of MU-MIMO support, each secondary path also requests search in this keyRAM. All these requests are arbitrated together. The keySearchEngine keeps the results available per user which will remain valid until the next request.

The Key Storage RAM hard macro connections are available at the Top level to ease the integration.

14.2 Support for Block ACK Mechanism

BA Controller triggers the Key Search Engine requesting the Index. Key Search Engine returns the Index for the MAC Address passed by BA Controller

keyIndex RAM	cType RAM [2:0]	vlanID RAM[3:0]	sppRAM[1:0]	useDef Key RAM[0]	cLen RAM [0]	macAddrRAM [47:0]	encrKeyRam [127:0]	intWPIKeyRam[3] [127:0]
0		Don't care				Don't care	Default Key ID 0 for VLAN 0	
1		Don't care				Don't care	Default Key ID 1 for VLAN 0	
2		Don't care				Don't care	Default Key ID 2 for VLAN 0	
3		Don't care				Don't care	Default Key ID 3 for VLAN 0	
4		Don't care				Don't care	Default Key ID 0 for VLAN 1	
5		Don't care				Don't care	Default Key ID 1 for VLAN 1	
. . .		Don't care				Don't care		
22		Don't care				Don't care	Default Key ID 2 for VLAN 5	
23		Don't care				Don't care	Default Key ID 3 for VLAN 5	
24						Device 1 MAC Address	Device 1 Secret Key	Device 1 WPI Integrity Key
25						Device 2 MAC Address	Device 2 Secret Key	Device 2 WPI Integrity Key
26						Device 3 MAC Address	Device 3 Secret Key	Device 3 WPI Integrity Key
. . .						. . .	. . .	. . .
62						Device 39 MAC Address	Device 39 Secret Key	Device 39 WPI Integrity Key
63						Device 40 MAC Address	Device 40 Secret Key	Device 40 WPI Integrity Key

Table 30: Key RAM structure

[3] This field exists only if the MAC HW supports the WAPI.

15 MAC-PHY Interface

15.1 Overview

The MAC-PHY Interface (hence forth called as MAC-PHY IF in this document) block has an asynchronous FIFO to buffer data movement between the different clock domains of the MAC and the PHY. Also it provides the necessary signals for interfacing with the PHY. The MAC-PHY Interface is defined in *[2]*. In case of MU-MIMO, the Transmit path is duplicated in order to provide the TX Vector and TX Data for each secondary users.

This block receives two clocks: *mpIFClk* and *macCoreClk*. This block interacts with the TX Controller, RX Controller, A-MPDU Deaggregator, MAC Controller, the FCS and the Tx Parameters Cache modules and the PHY for transmission and reception of frames. MAC-PHY IF has the following sub blocks.

- Transmit path
- Receive path
- MAC-PHY Interface Tx and Rx FIFO

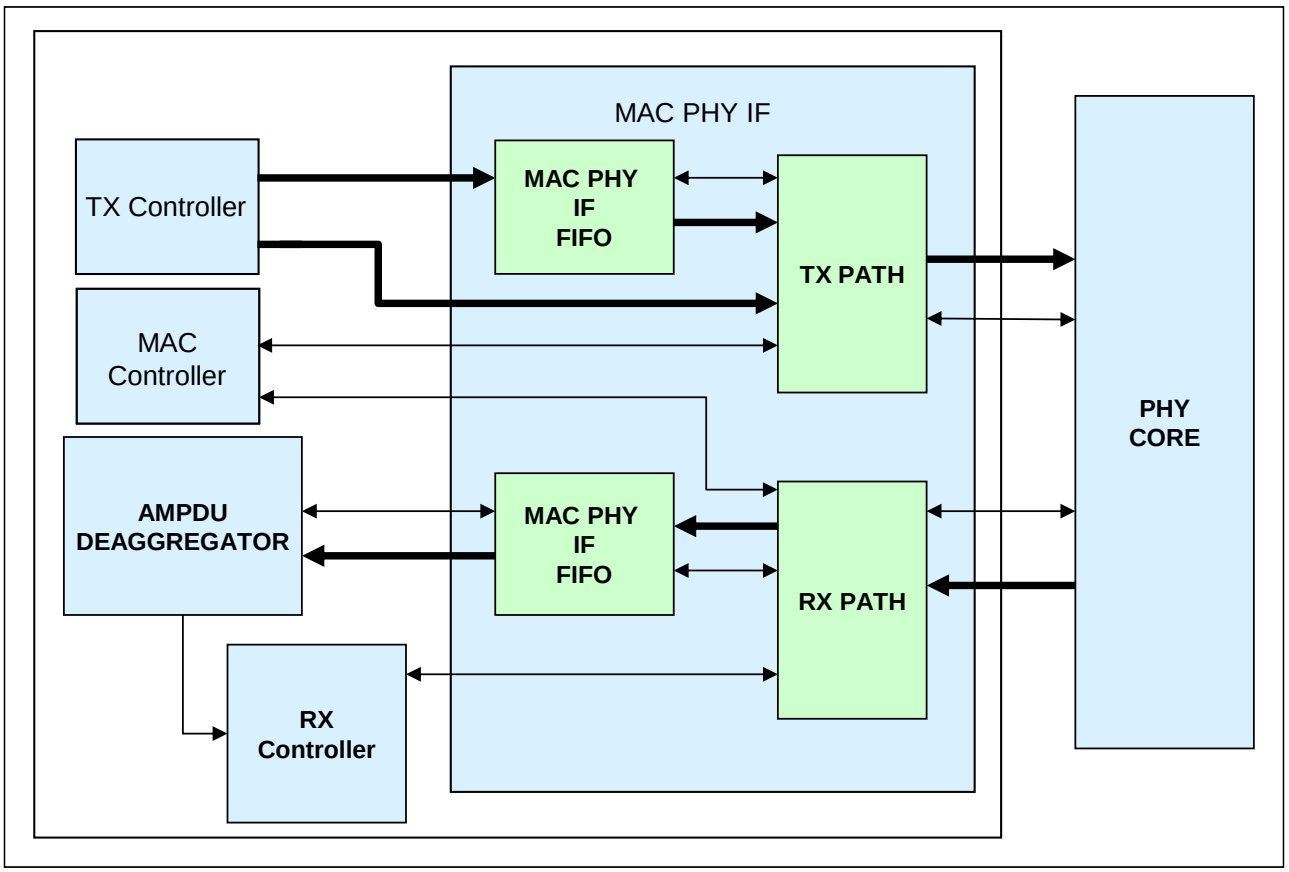


Figure 68: MAC-PHY Interface Block Diagram

The sections below cover each of the major blocks in the MAC-PHY IF.

15.2 Transmit path

15.2.1 Functional description

The transmit path prepares the TX Vector from the information provided by the MAC TxController and the TX Parameters Cache. The transmit path generates control signals for PHY and passes data read from MAC PHY Interface to PHY. The transmit path on getting a *startTx* signal from Transmit Controller, generates *txReq* and transmits TX-Vector to PHY. Once TX-Vector is transmitted data is read from the MAC PHY IF FIFO and passed to the PHY. The data transmission is based on *phyRdy* (provided by the PHY). In cases where an abort from the MAC Core, or the PHY error is generated, *txReq* is deasserted and the MAC-PHY IF FIFO is flushed.

15.2.2 State machine

The Transmit Path logic is controlled by simple state machine as explained below. The picture below shows the state diagram.

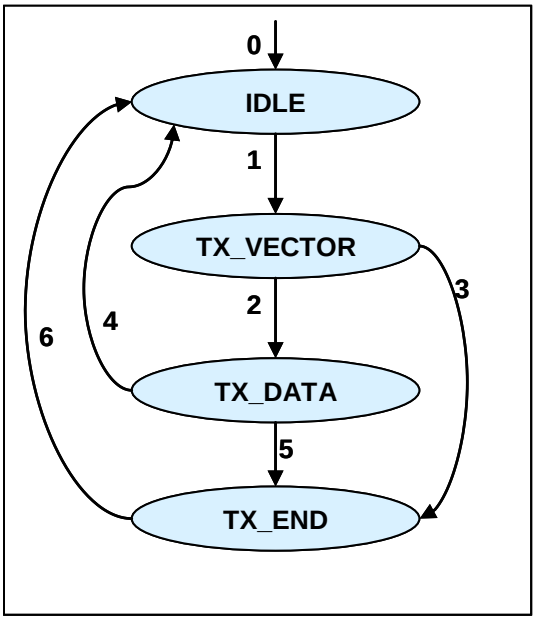


Figure 69: Transmit path state machine

The table below gives description of all states of Transmit path state machine.

State Name	State Description
IDLE	State machine is in this state when it is not active.
TX_VECTOR	In this state, *txReq* is asserted, and TX-Vector is passed to PHY by reading it from the MAC-PHY IF FIFO. If an abort or error indication from the MAC or PHY is received the *txReq* is deasserted. *keepRFOn* is deasserted.
TX_DATA	The *txReq* is kept asserted in this state. In this state, based on *phyRdy*, data is read from FIFO and passed to PHY. When the data is output the *macDataValid* is asserted indicating there is valid data on the data bus. When a *txEnd_p* is received the *txReq* is deasserted.
TX_END	In this state, the FSM waits for the *txEnd_p* signal from the PHY, removes *txReq* and asserts FIFO flush signal and moves to IDLE state.

Table 31: MAC-PHY IF Transmit path state description

The table below gives description of all state transition conditions for state machine.

Transition No.	Transition Description
0	On assertion of soft or hard reset the FSM enters IDLE state.
1	Transmission Start signal received from Transmit controller.
2	Transmission of TX-Vector completed.
3	Error indication received from PHY or abort indication from MAC controller.
4	*txEnd_p* occurs or *txEnd_p* occurs with phy_err or stopTx_p.
5	Error indication received from PHY or abort indication from MAC controller
6	*txEnd_p* occurs.

Table 32: MAC-PHY IF Transmit path state transition conditions

15.3 Receive path

15.3.1 Functional description

The receive path gets Receive Trigger from Receive Controller and generates *rxReq*. The data received from PHY is stored in the MAC-PHY Interface FIFO based on *phyRdy*. PHY sends the RX Vector1 followed by the payload and Rx Vector2. In case of a MAC request to abort reception the *rxReq* is de-asserted and the FSM waits for the *rxEnd_p* signal from the PHY. In case of a PHY error, the indication from the PHY is passed to the RX Controller and it continues waiting for the *rxEnd_p*. The *rxReq* is deasserted only after receiving the *rxEnd_p* from the PHY.

15.3.2 State machine

The Receive Path implements a simple state machine to complete PHY to MAC data transfer. The picture below shows the state diagram.

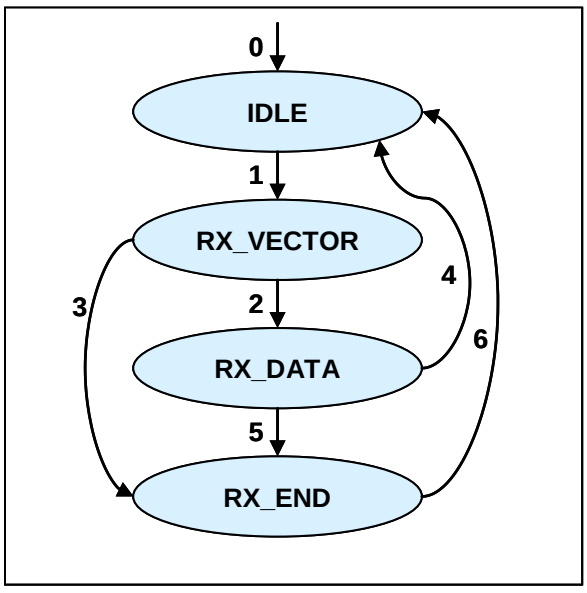


Figure 70: Receive path state machine

The table below gives description of all states of Receive Path state machine.

State Name	State Description
IDLE	The state machine remains in this state when there is no active reception.
RX_DATA	Data received from the PHY is written to the MAC-PHY IF FIFO. If there is a reception abort request from the MAC Controller the *rxReq* is deasserted else *rxReq* is kept asserted until the *rxEnd_p* occurs. *keepRFOn* is asserted in this state if the RX Controller indicates that the next frame will be received in RIFS time.
RX_END	This state is entered during an error condition and the FSM waits for the *rxEnd_p* in this state. Once it is received, if *rxReq* if asserted is deasserted. *keepRFOn* is deasserted.

Table 33: MAC-PHY IF Receive path state description

The table below gives description of all state transition conditions for state machine.

Transition No.	Transition Description
0	FSM enters this state on a hard or soft reset.
1	Receive Start Trigger received from Receive controller.
2	Reception RX Vector is completed.
3	When abort condition is indicated from Receive controller or PHY.
4	*rxEnd_p* occurs.
5	There is a PHY error while receiving a frame or the MAC indicates to stop reception.
6	*rxEnd_p* occurs.

Table 34: MAC-PHY IF Receive path state transition conditions

15.4 MAC-PHY Interface FIFO

This is a two port RAM. It is used by both Transmit path and Receive path. This is an asynchronous FIFO to take care of transfer of data between the two different clock domains of the PHY and MAC.

15.4.1 During Transmit operation

The *fifoFull* signal is used by the TX Controller to ensure proper write to the MAC-PHY IF FIFO. When the read signal to the MAC-PHY IF FIFO is asserted the data is available at the output bus one clock cycle later.

15.4.2 During Receive operation

Received data and RX Vector from PHY are written into the MAC-PHY Interface FIFO based on *phyRdy*. The A-MPDU Deaggregator will control the reading from the FIFO.

16 Coexistence Controller

16.1 Overview

The Coexistence Interface (hence forth called as Coex Interface in this document) block generates the different information requires by an external module in charge of the arbitration.

- Tx/Rx indication
- BW information
- PTI

This feature is enabled using `define RW_WLAN_COEX_EN

This block is running on *macCoreClk* and it interacts with the RX Controller, MAC Controller, TX Controller, De-aggregator, TX ParametersCache, MAC_PHY Interface and Registers.

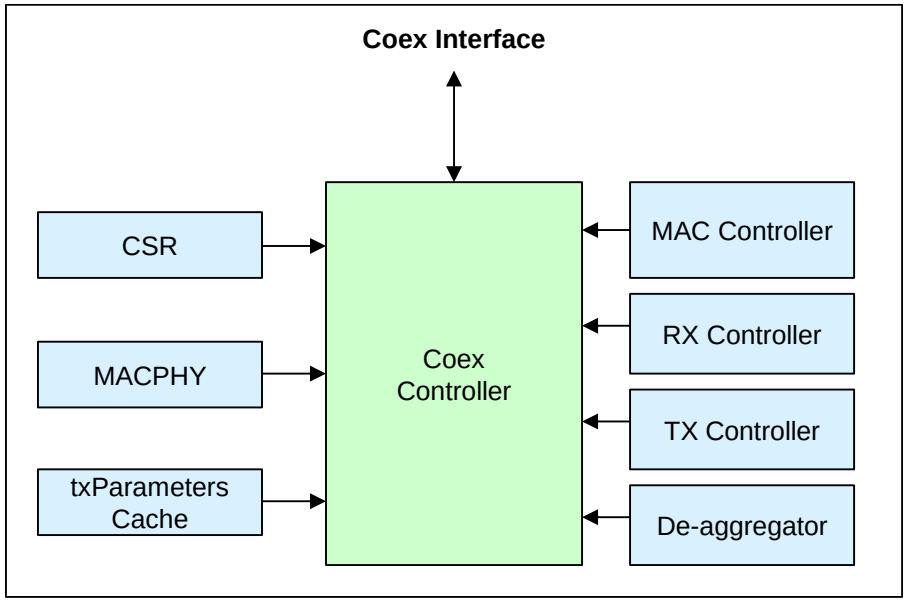


Figure 71: Coexistence Controller Block Diagram

16.2 Functional description

16.2.1 Basic

By default, the HW reports the on-going activities.
When a transmission is on-going (from *backoffDone* indication until *txEnd_p* reception from the PHY), the flag *coexWlanTx* is set.
When a reception is on-going (from the reception of the first byte of *rxVector1*, until the *rxEndFortiming_p* indication, the flag *coexWlanRx* is set. In parallel with these two flags, additionals information are provided to the external PTA, such as the transmission/reception BW (*coexWlanBW* output), the current channel frequency and channel offset used (*coexWlanFreq* and *coexWlanChanOffset* outputs) and the PTI (*coexWlanPTI* and *coexWlanPTIToggle* outputs) (see below). Note that the current channel frequency and offset outputs are directly connected to the Coex Control Register fields.

16.2.2 Transmission/Reception abort

In a WLAN – Bluetooth coexistence scenario, WLAN might be requested to abort transmission by means of *coexWlanTxAbort*, so as to prevent mutual interfering with a collocated BT device. This request is generated outside the MACHW.

When set and only when *coexEnable* bit is set, the *coexWlanTxAbort* is considered by the HW as a Medium Busy indication (similar to a Virtual CarrierSense). As a consequence, all the pending transmissions are postponed.

The *coexWlanTxAbort* also aborts the on-going transmission but this is done at the radio controller level. However, in case of Tx abort due to coexistence arbitration, the Singleton MPDU frames are retried but the retry numbers as well as the backoff CW are not incremented. In case of AMPDU, nothing special is done. In this specific case, the BACK is expected and the BAR is transmitted if the BA is not received.

It may also occur that the reception shall also be aborted due to a sharing of the PHY resources (RF for example) with collocated BT device. In this case, the *coexWlanRxAbort* is asserted and the *rxReq* is immediately de-asserted.

16.2.3 Packet Traffic Information

16.2.3.1 *Automatic PTI Adjustment*

16.2.4 *Coexistence Interface SW Control*

The full control of the Coexistence interface can be given to the software through the Coex Control register by setting the *coexForceEnable* bit. Then, the Coex interface output takes the values specified in *coexForceWlanXXX* fields.

An interrupt, named *coexEvent*, can also be generated in case of event detection on the *coexWlanTxAbort* and *coexWlanRxAbort* inputs. The detected events types are:

- *coexWlanTxAbort Rising Edge detected*
- *coexWlanTxAbort Falling Edge detected*
- *coexWlanRxAbort Rising Edge detected*
- *coexWlanRxAbort Falling Edge detected*

Each event detection can be enabled or disabled independently.

16.3 Timing Diagram

17 Beamforming Controller

The beamforming controller is in charge of managing the 802.11ac Beamforming procedure as a Beamformee.

- Capture and store the information extracted from the NDPA by the rxController
- Check the reception of a NDP after a NDPA based on an internal timeout counter.
- Control the external Beamforming HW accelerator
- Decide if a Beamforming Report frame shall be transmitted upon reception of NPD or Beamforming Report Poll frames.
- Compute the Beamforming Report frame length based on the information extracted from the NPDA and NPD.
- Provide information to the macControllerRx in order to compute the duration field and to trig the txController
- Provide the information to the txController to allow it to generate the Beamforming Report frame.

After a reception of a valid NDPA addressed to this device, the beamforming controller captures and stores the information provided by the rxController and de-aggregator which will be used later to control the external beamforming HW accelerator or the txController. Then, it starts a 1us downcounter which is initialized to SIFS + rxStartDelay + Slot).

If none *rxVectorStart* indication has been received before it reaches 0, all the stored information are discarded and the next NDP and Beamforming Report Poll will not be considered as valid.

If a *rxVectorStart* indication is received before the timeout counter elapses, the information extracted from the NDPA are stored and the next NDP will be considered as valid.

The macControllerRx do not generate response to a NDPA frame and moves to IDLE state. The macController moves back to WAIT_TX_RX state.

Upon reception of a NDP, the Beamforming Controller provides to the external Beamforming HW accelerator all the information required for the generation of the Beamforming report and starts it if the following conditions are fulfilled:

- bfmeeEnable = 1
- NDP has been received after a NPDA and the timeout did not expire.
- The NDPA was addressed to the device or the device AID matches on the STA Info AID fields.
- FeedbackType extracted from NDPA is 0 or bfmeeMUSupport = 1

The parameters provided to the Beamforming HW accelerator are decided as follow:

- The feedbackType is extracted from the NDPA
- The BW is equal to the BW received from the NDP.
- The Nr is equal to the NSTS received from the NDP.
- The Nc provided to the Beamforming HW accelerator is computed as follow:
    - In case of FeedbackType SU, Nc = min(bfmeeNc register value, Nr extracted from the received NDP)
    - In case of FeedbackType MU, Nc = min(bfmeeNc register value, Nc requested in NDPA)
- The other parameters such as Codebook and grouping come from the bfmeeControl register.

The *rxReq* shall be de-asserted during all the external Beamforming HW accelerator processing.

Then, the Beamforming Controller computes the length of the Beamforming Report frame and selects the PHY parameters used its transmission such as the MCS, transmission BW, formatMod. All these parameters remain valid until the next NDPA reception as they will be used by the macControllerRx in case of BeamForming Report Poll reception.

If the flag “STA Index is first” has been received, the transmission of Beamforming Report frame is requested to the txController by the macControllerRx. This transmission will be done after SIFS and starts even if the external beamforming HW accelerator has not finished the beamforming report computation.

If the flag “STA Index is first” has not been received, the macControllerRx waits for the completion of the beamforming report computation before moving back to IDLE state. This is indicated by the XXX pulse.

Upon reception of a Beamforming Report Poll addressed to our device, the macControllerRx computes the response duration based on the information provided by the Beamforming Controller and launches the generation of the beamforming Report frame.

References

[1] RW-WLAN-nX MAC HW STA 20MHz User Manual (RW-WLAN-nX-MAC-HW-STA20-UM)

[2] RW-WLAN-nX MAC PHY Interface Functional Specification (RW-WLAN-nX-MAC-PHY-IF-FS)