PRIME PHYSICAL LAYER API SPECIFICATION

Document Revision: 2.0

Issue Date: 5 May, 2011

Revision History

Revision	Draft	Author	Date	Comment
0.0	Initial version	Susan Yim	04/20/2008	
0.1	Completed API sets	Susan Yim, Minghua Fu, Xiaolin Lu	05/05/2009	
0.2	0.0	Susan Yim, Minghua Fu, Xiaolin Lu	13/05/2009	Modified based upon feedback from customers.
0.8		Susan Yim Minghua Fu	11/12/2009	Updated API
0.9		Susan Yim	11/30/2009	-Updated API (Callbacks/Set/Get - data structure instead of UINT16 pointers) - Removed PHY internal APIs
1.0		Susan Yim, Minghua Fu	04/16/2010	- Added PHY get PIB API - Added HAL ECAP interrupt functions and HALT function - All changes from release 2.3 are highlighted
2.0		Susan Yim, Minghua Fu	05/05/2010	- Moved HAL APIs to separate spec

Sign Off List

Group	Title	Authority	Date	Comment

Table of Content

Table of Figures

Table of Tables

1.0 Introduction

This document describes the PRIME physical layer software interfaces.
Figure 1. shows the overall Prime SW Stack block diagrams. The PHY Layer includes the following modules:

- PHY Receive Manager (PRXM)
- PHY Transmit Manager (PTXM)
- Rx/Tx Synchronization Module
- MAC/PHY Interface Module
- Common Math Function Utility Module

In HAL Layer AFE driver module is used directly interfacing with the PHY Layer.

All APIs described in this document are C callable.

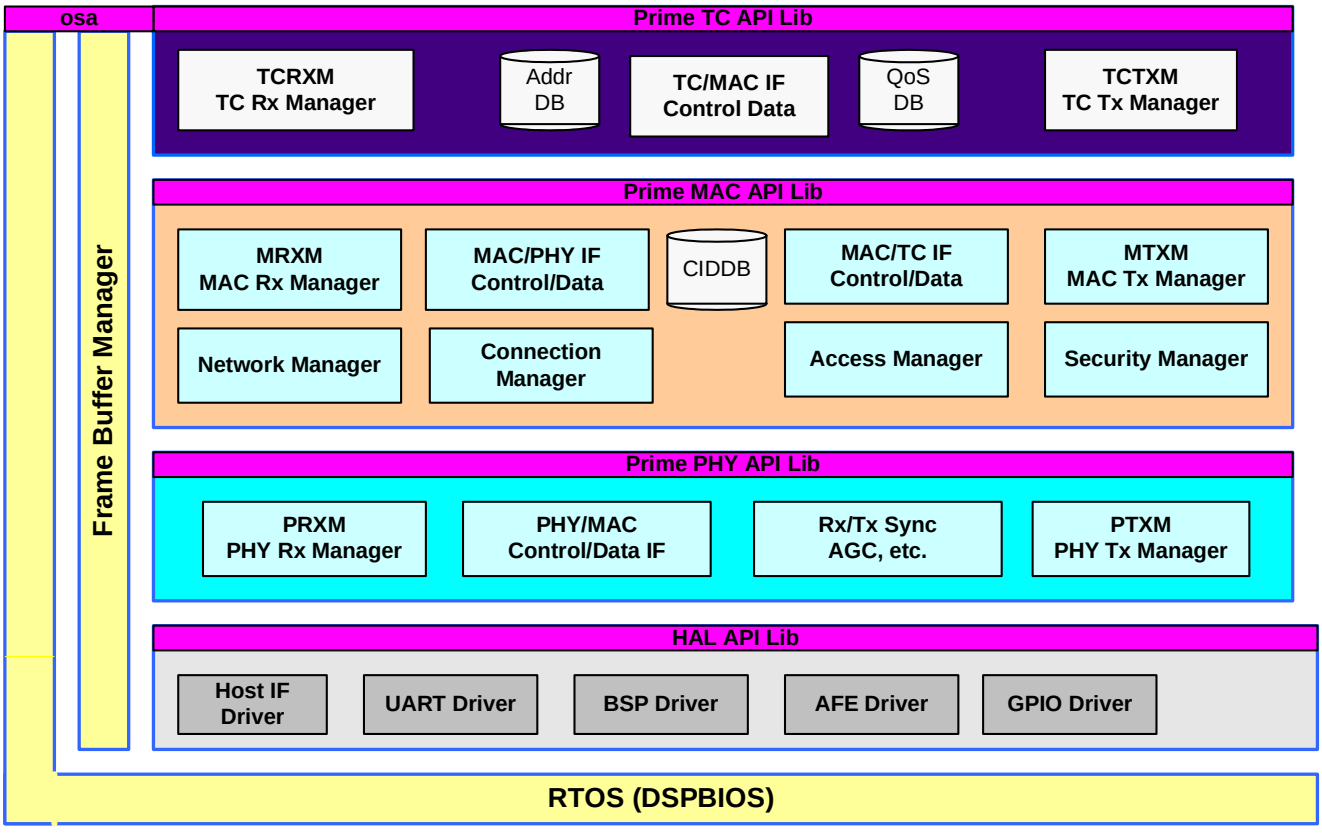

Figure 1 Overall PRIME Software Architecture

2.0 Prime PHY Layer Public APIs

Table 1 lists all the PHY layer library public API definitions.

API	Sync/Async	Descriptions
PHY_txInit	Sync	Initialize PHY Tx chain.
PHY_rxInit	Sync	Initialize PHY Rx chain
PHY_rxStart	Async	Starts initial preamble detection and synchronization
PHY_txPpdu	Async	Transmit PHY PPDU to the power line
PHY_rxPpduStart	Async	Starts PPDU reception process
PHY_rxPpduStop	Sync	Stops PPDU reception process
PHY_rxPpduRelease	Sync	Release RX PPDU buffer
PHY_txSuspend	Async	Suspend PHY TX activities
PHY_rxSuspend	Async	Suspend PHY RX operations
PHY_txResume	Sync	Resume suspended PHY TX activities
PHY_rxResume	Sync	Resume PHY suspended RX operations
PHY_txSmRun	Sync	Runs PHY TX processing
PHY_rxSmRun	Sync	Runs PHY Rx processing
PHY_txTestmode	Async	Put PHY TX in test mode
PHY_txSet	Sync	Set PHY Tx parameters
PHY_rxSet	Sync	Set PHY Rx parameters
PHY_txGet	Sync	Get PHY Tx parameters
PHY_rxGet	Sync	Get PHY Rx parameters
PHY_txGetStatistic	Sync	Read PHY TX statistical parameters
PHY_rxGetStatistic	Sync	Read PHY RX statistical parameters

Table 1 PHY Layer Public APIs

NOTE: All asynchronous APIs require callback function as parameter for caller to provide.

The following data structure is defined as common PHY layer public API return code:

```
typedef enum
{
    PHY_STAT_SUCCESS                    = 0,
    PHY_STAT_FAILURE                    = 1,
    PHY_STAT_PREAMBLE_NOT_DETECTED      = 2,
    PHY_STAT_HEADER_CRC_FAILED          = 3,
    PHY_STAT_PAYLOAD_CRC_FAILED         = 4,
    PHY_STAT_ILLEGAL_PARAMETERS         = 5,
    PHY_STAT_ILLEGAL_OPERATIONS         = 6,
```

```
    PHY_STAT_UNKOWN_ID                  = 7,
    PHY_STAT_TX_LATE                    = 8,
    PHY_STAT_INVALID_LEN                = 9,
    PHY_STAT_INVALID_SCH                = 10,
    PHY_STAT_INVALID_LEV                = 11,
    PHY_STAT_BUF_OVRUN                  = 12,
    PHY_STAT_BUSY                       = 13,
    PHY_STAT_CMD_IN_PLACE               = 14,
    PHY_STAT_NOT_IN_SYNC                = 15
    PHY_STAT_RX_BUF_OVERRUN             = 16

} PHY_status_t;
```

The callback function used for asynchronous APIs for notification is defined as:

```
typedef void (*PHY_cbFunc_t)(PHY_ev_t eventID, PHY_cbData_t *cbData_p);

typedef enum
{
        PHY_EV_RX_START_DONE            = 0,
        PHY_EV_TX_PPDU_DONE             = 1
        PHY_EV_RX_PPDU_DONE             = 2,
        PHY_EV_TX_SUSPEND_DONE          = 3,
        PHY_EV_RX_SUSPEND_DONE          = 4,
        PHY_EV_TX_TESTMODE_DONE         = 5,

} PHY_ev_t;


typedef struct
{
 PHY_status_t       status;    // callback status
 union
 {
   PHY_cbTxPpdu_t    txPpdu;    // PHY Tx ppdu done callback
   PHY_cbRxSync_t    rxSync;    // PHY Rx channel acq callback
   PHY_cbRxPpdu_t    rxPpdu;    // PHY Rx ppdu callback

 }cbParms;

}PHY_cbData_t;

/* Tx PPDU done callback data structure */
typedef struct
{
 UINT32      ppduAddr;         // Tx PPDU address passed by caller

}PHY_cbTxPpdu_t;

/* Rx channel acquired callback data structure */
typedef struct
{
 SINT16      rssi;             // rssi in dBm
 UINT16      nSymbols;         // number of symbols for ppdu

}PHY_cbRxSync_t;

/* Rx PPDU callback data structure */
typedef struct
{
 UINT32      ppduInfoAddr;      // Rx PPDU info address (see PHY_rxPpdu_t in phy_rx.h)

}PHY_cbRxPpdu_t;
```

2.1 PHY Tx Path Initialization

This API initializes or resets the PHY TX Manager (PHY_TX) module. It's only called once after the unit powered up or system reset.

Syntax `PHY_status_t PHY_txInit(void)`

Parameter	Description
Return Value	PHY_status_t: PHY_STAT_SUCCESS PHY_STAT_FAILURE

2.2 PHY Transmit PPDU

This API starts a transmission of a PPDU (maximum 2268 bytes) to the power line IF. It is called by MAC TX Manager (MTXM) every time when there is a PPDU to be transmitted.

Syntax
```
PHY_status_t PHY_txPpdu(PHY_tx_ppdu_t *ppdu_p,
                        PHY_cbFunc_t   cb_p)
```

Parameter	Description

Ppdu_p	Pointer to the PPDU primitive data structure configured by caller which contains data buffer for PPDU to be transmitted, modulation, coding scheme and transmit power level for PPDU transmission.
	typedef struct
	{
	UINT16 *ppduMacH_p; // pointer to MAC_H (16-bit word aligned)
	UINT16 *ppduPld_p; // pointer to PPDU payload (16-bit word aligned)
	UINT16 length; // MPDU buffer length in bytes
	UINT16 level; // TX level signal level encoding:
	// 0: Maximum output level (MOL)
	// 1: MOL – 3dB
	// 2: MOL – 6dB
	// ….
	// 7: MOL – 7dB
	UINT16 mcs; // Tx modulation coding scheme:
	// 0: DBPSK
	// 1: DQPSK
	// 2: D8PSK
	// 3; Not used
	// 4: DBPSK + Convolution Code (FEC)
	// 5; DQPSK + FEC
	// 6: D8PSK + FEC
	// 7: Not used
	// 8: ROBO ½ (DBPSK + FEC + ¼ repetition)
	// 9-11: Not used

	// 12 : ROBO ¼ (DBPSK + FEC + 1/8 repitition) UINT16 txTime ; // PPDU transmit time schedule. } PHY_tx_ppdu_t;

cb_p	Callback function when PPDU transmit finishes. It contains the following parameters:
	eventID = PHY_EV_TX_PPDU_DONE;
	cbData.status = PHY_STAT_SUCCESS // successful transmission
	= PHY_STAT_FAILURE // Failure transmission
	cbData.cbParms:
	typedef struct
	{
	UINT32 ppduAddr; // Tx PPDU address passed by caller
	
	}PHY_cbTxPpdu_t;

Return Values	PHY_status_t: PHY_STAT_SUCCESS PHY_STAT_FAILURE PHY_STAT_TX_LATE PHY_STAT_INVALID_LEN PHY_STAT_INVALID_SCH PHY_STAT_INVALID_LEV PHY_STAT_BUF_OVRUN PHY_STAT_BUSY

2.3 PHY TX Suspend

This API suspends PHY TX activities. PHY should complete all pending transmissions before entering the sleep mode.

Syntax `PHY_status_t PHY_txSuspend(PHY_cbFunct_t cb_p)`

Parameter	Description
cb_p	Callback function when suspend finishes with following parameters: eventID = PHY_EV_TX_SUSPEND_DONE
Return Values	PHY_status_t: PHY_STAT_SUCCESS // Suspend successful, immediately suspended PHY_STAT_BUSY // TX is busy; will suspend and callback after finishing pending transmission PHY_STAT_CMD_IN_PLACE // TX already suspended

2.4 PHY TX Resume

This API resumes suspended PHY activities and starts normal transmission.

Syntax `PHY_status_t PHY_txResume(void)`

Parameter	Description
Return Values	PHY_status_t: PHY_STAT_SUCCESS PHY_STAT_CMD_IN_PLACE

2.5 PHY TX Process

This API runs PHY Tx process during active packet transmission. It is to be called at PRIME PHY symbol rate (per DMA interrupt).

Syntax `PHY_status_t PHY_txSmRun(void)`

Parameter	Description
Return Values	PHY_status_t: PHY_STAT_SUCCESS PHY_STAT_FAILURE

2.6 PHY TX Testmode

This API enables/disables the PHY TX test mode operation.

Syntax
```
PHY_status_t PHY_txTestmode(UINT16 enable, UINT16 mode,
UINT16 mcs, UINT16 pwrLevel, PHY_cbFunct_t cb_p)
```

Parameters	Description
enable	Starts or stops test mode: 0 = stop test mode, 1 = enable test mode
mode	PHY Tx Test mode settings: 0 = continuous transmit, 1 = transmit with 50% duty cycle
mcs	MCS settings under test mode. It contains one of the following 8 values: 0: DBPSK 1: DQPSK 2: D8PSK 3; Not used 4: DBPSK + Convolution Code 5; DQPSK + Convolutional Code 6: D8PSK + Convolutional Code 7 : Not used
pwrLevel	relative level the test signal is transmitted. It contains one of the following values: 0: Maximum Output Level (MOL) 1: MOL – 3 dB 1: MOL – 6 dB ..... 7: MOL – 21 dB

cb_p	Callback function when PHY Tx path has been switched into the specified mode. It has the following parameters: eventID = PHY_EV_TESTMODE_DONE;
Return Values	PHY_status_t: PHY_STAT_SUCCESS PHY_STAT_FAILURE PHY_STAT_CMD_IN_PLACE: PHY already in test mode

2.7 PHY TX Parameter Set

This API sets PHY TX parameters.

Syntax `PHY_status_t PHY_txSet(UINT16 setType, PHY_txSetData_t *setData_p)`

Parameter	Description
setType	Parameter type to be set: 0x0000 - PHY_TX_CLEAR_STAT: clear PHY TX statistics (set to defaults) 0x0001 – PHY_TX_SETMODE: 1-ROBO mode; 0-PRIME mode 0x0002 – 0xFFFF: reserved

setData_p	Pointer to data to be set
	Typedef union
	{
	UINT16 roboMode; // 0-PRIME; 1-ROBO
	}
	PHY_TX_CLEAR_STAT: do not need to pass setData_p (put 0)
	PHY_TX_SETMODE: Set ROBO or PRIME mode
	UINT16 roboMode; // 0-PRIME; 1-ROBO

Return Values	PHY_status_t: PHY_STAT_SUCCESS PHY_STAT_FAILURE

2.8 PHY TX Get

This API gets PHY TX parameters.

Syntax `PHY_status_t PHY_txGet(UINT16 getType, PHY_txGetData_t *getData_p)`

Parameter	Description
getType	Parameter type to get: 0x0000 – PHY_TIMER: get the current timer (in 10us) 0x0001 – PHY_TXTIME: get Next Transmission Start Time 0x0002 – PHY_ZCT: zero crossing time main 0x0003:-- PHY_TX_GETMODE: get TX mode (PRIME or ROBO)

	0x0004 – 0xFFFF: reserved

getData_p	stores the results corresponding getType:
	typedef union
	{
	UINT32 currTime; // Current time in units of 10 us
	UINT32 nextTxTime; // Next transmission start time in 10us.
	PHY_zcTime_t zcTime; // Last ZC time
	}PHY_txGetData_t;
	
	PHY_TIMER:
	getData_p->currTime = current time in 10s of microseconds.
	PHY_TXTIME:
	getData_p->nextTxTime = next transmit start time in 10s of microseconds. It returns 0xFFFFFFFF when there is no pending transmission.
	
	PHY_ZCT:
	typedef struct
	{
	UINT32 lastZcaTime; // Phase 1 last zero crossing time in 10us (20-bits)
	UINT32 lastZcbTime; // Phase 2 last zero crossing time in 10us (20-bits)
	}PHY_zcTime_t;
	
	PHY_TX_GETMODE:
	getData_p->roboMode: 0-PRIME; 1-ROBO

Return Values	PHY_status_t: PHY_STAT_SUCCESS PHY_STAT_FAILURE

2.9 PHY TX Get Statistics

This API reads PHY TX path statistics.

Syntax `PHY_status_t PHY_txGetStat(PHY_tx_stat_t *getData_p)`

Parameter	Description

getData_p	Pointer to the buffer which can be used to store the returned PHY Tx statistics:
	Typedef struct {
	UINT16 phyStatsTxDropCount; // # of times Tx PHY drops data owing to overflow
	UINT16 phyTxQLen; // # of MPDUs Tx PHY can hold
	UINT32 phyTxProcDelay; // time in 10's us between TxPpdu() to PPDU goes to line IF
	} PHY_tx_stat_t;

Return Values	PHY_status_t: PHY_STAT_SUCCESS PHY_STAT_FAILURE

2.10 PHY Layer Rx Path Initialization

This function initializes or resets the PHY Rx Manager (PHY_RX) module. It initializes the PHY receiver global data structure, internal states and reset buffers. It is called once at power up or reset.

Syntax `PHY_status_t PHY_rxInit(void)`

Parameter	Description
Return Values	PHY_status_t: PHY_STAT_SUCCESS PHY_STAT_FAILURE

2.11 PHY Receiver Start Channel Acquisition

This function requests PHY to start the initial synchronization or resynchronization to the power line channel. This includes preamble detection and valid header parsing. This should be called at least once after power up.

Syntax

```
PHY_status_t PHY_rxStart(UINT16 timeOut,
                         PHY_cbFunct_t cb_p)
```

Parameter	Description
timeOut	initial synchronization time out values (in number of symbols). 0xFFFF means never time-out. Caller has to use PHY_rxSuspend() to stop the receiver synchronization process.

cb_p	Callback function when preamble detection and header is completed. It contains the following parameters:
	eventID: PHY_EV_RX_START_DONE;
	cbData.status : channel synchronization status with following possible values:
	PHY_STAT_SUCCESS
	PHY_STAT_FAILURE
	PHY_STAT_PREAMBLE_NOT_DETECTED
	PHY_STAT_ILLEGAL_PARAMETERS
	cbData.cbParms
	typedef struct
	{
	SINT16 rssi; // RSSI value in dBm
	UINT16 nSymbols; // PPDU duration in number of symbols
	
	}PHY_cbRxSync_t;

Return Values	PHY_status_t: PHY_STAT_SUCCESS PHY_STAT_FAILURE PHY_STAT_CMD_IN_PLACE // PHY Rx is already in sync

2.12 PHY Receiver Suspend

This function requests to suspend all PHY receiver's present activities including all reception functions.

Syntax `PHY_status_t PHY_rxSuspend(PHY_cbFunct_t cb_p)`

Parameter	Description
cb_p	Callback function when PHY_rxSuspend() is completed. It has the following parameters: eventID: PHY_EV_RX_SUSPEND_DONE;
Return Values	PHY_status_t PHY_STAT_SUCCESS PHY_STAT_FAILURE PHY_STAT_CMD_IN_PLACE

2.13 PHY Receiver Resume

This function requests to resume PHY receiver's suspended activities. PHY shall start its normal reception. More details TBD.

Syntax `PHY_status_t PHY_rxResume(void)`

Parameter	Description
Return Value	PHY_status_t: PHY_STAT_SUCCESS, PHY_STAT_FAILURE, PHY_STAT_CMD_IN_PLACE

2.14 PHY PPDU Reception Start

This function can only be called after PHY completed initial channel acquisition successfully. By calling this function, PHY will start passing newly received PPDUs to the caller through the callback function. The PPDU reception process is continuing until it is canceled through PHY_rxPpduStop().

Syntax `PHY_status_t PHY_rxPpduStart(PHY_cbFunct_t cb_p)`

Parameter	Description
cb_p	Callback function when a complete PPDU data unit has been received from power line. It contains the following parameters: eventID: PHY_EV_RX_PPDU_ DONE cbData.status: PHY PPDU receive status code which contains PHY_STAT_SUCCESS PHY_STAT_RX_BUF_OVERRUN

	cbData.cbParms.
	typedef struct
	{
	UINT32 ppduInfoAddr; // Rx PPDU info address (see PHY_rxPpdu_t in phy_rx.h)
	}PHY_cbRxPpdu_t
	
	ppduInfoAddr: received PPDU data structure start address (buffer owned by PHY). The PPDU format is as following:
	typedef struct {
	UINT16 id; // ppdu ID
	UINT16 length; // number of (MAC_H + data) bytes
	UINT16 level; // received level:
	// 0: <= 70dBuV
	// 1: <= 72dBuV
	// 2: <= 74dBuV
	// …..
	// 15: > 98dBuV
	UINT16 mcs; // modulation coding scheme:
	// 0: DBPSK
	// 1: DQPSK
	// 2: D8PSK
	// 3; Not used
	// 4: DBPSK + Convolution Code
	// 5; DQPSK + Convolutional Code
	// 6: D8PSK + Convolutional Code
	// 7: Not used
	// 8: ROBO ½ (DBPSK + FEC + ¼ repetition)
	// 9-11: Not used
	// 12 : ROBO ¼ (DBPSK + FEC + 1/8 repitition)
	UINT16 *ppdu_data_p; // pointer to data (MAC_H + payload)
	UINT32 time; // time when preamble is received
	} PHY_rxPpdu_t
	A NULL pointer means current PPDU reception contains errors (reference statusCode).

Return Values	PHY_status_t : PHY_STAT_SUCCESS PHY_STAT_FAILURE PHY_STAT_CMD_IN_PLACE // Rx receive process already started PHY_STAT_NOT_IN_SYNC // Receiver synchronization process is not started.

2.15 PHY PPDU Reception Stop

This function can only be called after a PHY PPDU reception process has been started through PHY_rxPpduStart().By calling this function, PHY will stop the delivering of the received streams to the caller. This is a synchronous call.

Syntax `PHY_status_t PHY_rxPpduStop(void)`

Parameter	Description
Return Values	PHY_status_t : PHY_STAT_SUCCESS PHY_STAT_FAILURE

	PHY_STAT_CMD_IN_PLACE // no active PPDU reception in progress

2.16 PHY PPDU Buffer Release

This function is called every time after PHY delivers a received PPDU from the power line IF to the caller (through the callback function registered) to indicate to the PHY that the buffer used to store the PPDU is available to PHY again. If the buffer is not released in time, the PHY will issue a callback function with NULL PPDU buffer and signal PHY Rx buffer overrun error. This is a synchronous API.

Syntax `PHY_status_t PHY_rxPpduRelease(PHY_rxPpdu_t *ppdu_p)`

Parameter	Description

Ppdu_p	PPDU buffer pointer to be released. PPDU buffer format:
	typedef struct {
	UINT16 id; // ppdu ID
	UINT16 *data_p; // pointer to data (MAC_H + payload)
	UINT16 length; // number of (MAC_H + data) bytes
	UINT16 level; // received level:
	// 0: <= 70dBuV
	// 1: <= 72dBuV
	// 2: <= 74dBuV
	// …..
	// 15: > 98dBuV
	UINT16 mcs; // modulation coding scheme:
	// 0: DBPSK
	// 1: DQPSK
	// 2: D8PSK
	// 3; Not used
	// 4: DBPSK + Convolution Code
	// 5; DQPSK + Convolutional Code
	// 6: D8PSK + Convolutional Code
	// 7: Not used
	// 8: ROBO ½ (DBPSK + FEC + ¼ repetition)
	// 9-11: Not used
	// 12 : ROBO ¼ (DBPSK + FEC + 1/8 repitition)
	} PHY_rxPpdu_t

Return Values	PHY_status_t : PHY_STAT_SUCCESS PHY_STAT_FAILURE PHY_STAT_CMD_IN_PLACE // buffer already released

2.17PHY RX Process

This API runs PHY Rx process during active packet transmission. It is to be called at PRIME PHY symbol rate (per DMA interrupt).

Syntax `PHY_status_t PHY_rxSmRun(void)`

Parameter	Description
Return Values	PHY_status_t: PHY_STAT_SUCCESS PHY_STAT_FAILURE

2.18 PHY Receiver Parameters Set

This function sets the values for PHY parameters.

Syntax

```
PHY_status_t PHY_rxSet(UINT16 setType,
                       PHY_rxSetData_t *setData_p)
```

Parameter	Description
setType	Parameter type to be set: 0x0000: PHY_RX_SET_AGC – set AGC parameters 0x0001: PHY_RX_SET_PPDU_INFO – set PPDU parameters 0x0002: PHY_RX_SET_ROBOMODE – 1: ROBO mode; 0: PRIME mode 0x0003-0xFFFF: reserved

seData_p	Pointer to data to be set:
	typedef union
	{
	PHY_rxAagcSet_t aagc;
	PHY_rxPpduSet_t ppdu;
	UINT16 roboMode;
	}PHY_rxSetData_t;
	PHY_SET_AGC: set AGC parameters
	typedef struct
	{
	UINT16 mode; // AGC mode = 0 (auto), 1(manual)
	UINT16 step; // AGC gain step = 0 to N-1 where N is maximum number of AGC gain steps (in increment of AGC gain_step_dB)
	
	}PHY_rxAagcSet_t;
	
	PHY_SET_PPDU_INFO: set PPDU info of previous PPDU received
	typedef struct
	{
	UINT16 ppduId; // indicates information on the PPDU that was transferred from PHY previously.
	UINT16 nodeId; // node id from which the previous PPDU is received
	// 0 – base node beacon
	// 1 - switch node beacon
	// 2 to 15 data from other nodes
	
	}PHY_rxPpduSet_t;

	PHY_RX_SET_ROBOMODE: Set ROBO or PRIME mode UINT16 roboMode; //0_PRIME; 1-ROBO
Return Values	PHY_status_t: PHY_STAT_SUCCES PHY_STAT_FAILURE

2.19 PHY Receiver Parameters Get

This function gets the PHY receiver parameter values.

Syntax `PHY_status_t PHY_rxGet(UINT16 getType, UINT16 *getData_p)`

Parameter	Description
getType	Parameter type to be get: 0x0000: PHY_RX_GET_AGC 0x0001: PHY_RX_GET_CD 0x0002: PHY_RX_GET_NL 0x0003: PHY_RX_GET_SNR 0x0004: PHY_RX_GET_RQ 0x0005: PHY_RX_GET_ROBOMODE

getData_p	stores the results corresponding getType:
	typedef union
	{
	PHY_rxAagcGet_t aagc;
	PHY_rxCdGet_t cd;
	UINT16 nl; // noise level 0 to 15 (50 to 92 dBuV in 3 dB step)
	UINT16 snr; // snr 0 tp 7 (0 to 18 dB in 3 dB step)
	UINT16 rq; // receive quality (0 to 8 from bad to good)
	UINT16 roboMode; // 0-PRIME; 1-ROBO
	}PHY_rxGetData_t;
	PHY_RX_GET_AGC: get AGC parameters
	typedef struct
	{
	UINT16 mode; // AGC mode = 0 (auto), 1(manual)
	UINT16 step; // 0 to N-1 where N is maximum number of AGC gain
	}PHY_rxAagcGet_t;
	
	PHY_RX_GET_CD: get carrier detection status
	typedef struct
	{
	UINT32 busyTime; // current header or PPDU time in 10s of us (20-bits)
	UINT16 detected; // 0 = no carrier detected, 1 = carrier detected
	UINT16 rssi; //RSSI of preamble, it contains one of following numbers:
	0: <= 70dBuV
	1: <= 72dBuV
	2: <= 74dBuV
	…..
	15: > 98dBuV RSSI of preamble detected
	UINT16 status; // 1 = preamble detected, but len unknown from header decoding

	// 0 = otherwise }PHY_rxCdGet_t; **PHY_RX_GET_NL:** get noise floor level UINT16 nl: noise floor level, it contains one of following numbers: 0: <= 50dBuV 1: <= 53dBuV 2: <= 56dBuV ….. 15: > 92dBuV **PHY_RX_GET_SNR:** ratio of measured received signal level to noise level of last received PPDU: UINT16 snr: SNR of last received PPDU, ti contains one of following numbers: 0: <= 0 dB 1: <= 3 dB 2: <= 6 dB ….. 7: > 18 dB **PHY_RX_GET_RQ:** last received frame quality UINT16 rq: quality of the received signal, measured by the EVM parameter. It contains one of the following numbers: 0: bad ….. 8: good **PHY_RX_GET_ROBOMODE**: RX mode UINT16 roboMode: 0-PRIME; 1-ROBO
Return Values	PHY_status_t: PHY_STAT_SUCCES PHY_STAT_FAILURE

2.20 PHY RX Get Statistics

This API gets PHY RX path statistics.

Syntax `PHY_status_t PHY_rxGetStat(PHY_rx_stat_t *getData_p)`

Parameter	Description

getData_p	Pointer to the buffer which can be used to store the returned PHY Rx statistics:
	Typedef struct {
	UINT16 phyCrcIncorrectCount; // # of bursts received with CRC errors
	UINT16 phyCrcFailCount; // # of bursts received with correct CRC but invalid protocol header
	UINT16 phyRxDropCount; // # of received PPDU drop owing buffer overrun
	UINT16 phyRxQueueLen; // # of concurrent MPDUs Rx buffer can hold
	UINT32 phyRxProcDelay; // time in us from PPDU received from AFE to MPDU available to MAC
	SINT16 phyAgcMinGain; // minimum gain for AGC <0dB
	UINT16 phyAgcStepValue; // distance between steps in dB < 6dB
	UINT16 phyAgcStepNum; // number of steps so that phyAgcMinGain+((phAgcStepNum – 1)*phyAgcStepValue) >= 21dB
	UINT32 phyRxPpduCount; // number of packets received since switched on
	} PHY_tx_stat_t;

Return Values	PHY_status_t: PHY_STAT_SUCCESS PHY_STAT_FAILURE

2.21 PHY Get PRIME Information Base (PIB)

Two PIB APIs are implemented. One is to get the PIB value, and the other is to get the PIB size.

Syntax `PHY_status_t PHY_PIB_get(PHY_PIB_ID_t id, void *pdata, int data_size);`

Parameters	Description

id	PHY_PIB_ID_t:
	typedef enum
	{
	phyStatsCRCIncorrectCount = 0xa0, // 16bit: Number of bursts received on the physical layer // for which the CRC was incorrect.
	phyStatsCRCFailCount = 0xa1, // 16bit: Number of bursts received on the physical layer // for which the CRC was correct, but the Protocol field of // PHY header had invalid value.
	phyStatsTxDropCount = 0xa2, // 16bit: Number of times when PHY layer received new // data to transmit and had to either overwrite on existing // data in its transmit queue or drop the data in new request // due to full queue
	phyStatsRxDropCount = 0xa3, // 16bit: Number of times when PHY layer received new // data on the channel and had to either overwrite on // existing data in its receive queue or drop the newly // received data due to full queue.
	phyTxQueueLen = 0xb0, // 10bit: Number of concurrent MPDUs that the PHY // transmit buffers can hold.
	phyRxQueueLen = 0xb1, // 10bit: Number of concurrent MPDUs that the PHY // receive buffers can hold.
	phyTxProcessingDelay = 0xb2, // 20bit: Time elapsed from the instance when data is // received on MAC-PHY communication interface to the // time when it is put on the physical channel. This shall not // include communication delay over the MAC-PHY // interface.In unit of microseconds
	phyRxProcessingDelay = 0xb3, // 20bit: Time elapsed from the instance when data is // received on physical channel to the time when it is made // available to MAC across the MAC-PHY communication // interface. This shall not include communication delay // over the MAC-PHY interface. In unit of microseconds
	phyAgcMinGain = 0xb4, // 8bit: Minimum gain for the AGC in dB
	phyAgcStepValue = 0xb5, // 3bit: Distance between steps in dB
	phyAgcStepNumber = 0xb6 // 8bit: Number of steps so that phyAgcMinGain + // ((phyAgcStepNumber – 1) * phyAgcStepValue) >= 21dB
	}

pdata	Pointer to the buffer to put the PIB value into

Return Values	PHY_status_t: PHY_STAT_SUCCESS PHY_STAT_FAILURE

Syntax `int PHY_PIB_get_size(PHY_PIB_ID_t id);`

Parameters	Description

id	PHY_PIB_ID_t:
	typedef enum
	{
	phyStatsCRCIncorrectCount = 0xa0, // 16bit: Number of bursts received on the physical layer
	// for which the CRC was incorrect.
	phyStatsCRCFailCount = 0xa1, // 16bit: Number of bursts received on the physical layer
	// for which the CRC was correct, but the Protocol field of
	// PHY header had invalid value.
	phyStatsTxDropCount = 0xa2, // 16bit: Number of times when PHY layer received new
	// data to transmit and had to either overwrite on existing
	// data in its transmit queue or drop the data in new request
	// due to full queue
	phyStatsRxDropCount = 0xa3, // 16bit: Number of times when PHY layer received new
	// data on the channel and had to either overwrite on
	// existing data in its receive queue or drop the newly
	// received data due to full queue.
	phyTxQueueLen = 0xb0, // 10bit: Number of concurrent MPDUs that the PHY
	// transmit buffers can hold.
	phyRxQueueLen = 0xb1, // 10bit: Number of concurrent MPDUs that the PHY
	// receive buffers can hold.
	phyTxProcessingDelay = 0xb2, // 20bit: Time elapsed from the instance when data is
	// received on MAC-PHY communication interface to the
	// time when it is put on the physical channel. This shall not
	// include communication delay over the MAC-PHY
	// interface.In unit of microseconds
	phyRxProcessingDelay = 0xb3, // 20bit: Time elapsed from the instance when data is
	// received on physical channel to the time when it is made
	// available to MAC across the MAC-PHY communication
	// interface. This shall not include communication delay
	// over the MAC-PHY interface. In unit of microseconds
	phyAgcMinGain = 0xb4, // 8bit: Minimum gain for the AGC in dB
	phyAgcStepValue = 0xb5, // 3bit: Distance between steps in dB
	phyAgcStepNumber = 0xb6 // 8bit: Number of steps so that phyAgcMinGain +
	// ((phyAgcStepNumber – 1) * phyAgcStepValue) >= 21dB
	}

pdata	Pointer to the buffer to put the PIB value into
Return Values	PIB size (in number of bytes)

3.0 References

[1] "PRIME Spec v1.1"
[2] "Prime Preamble Detection and FFT Implementation Algorithm Spec"
[3] "Prime AAGC Algorithm Spec"
[4] "Prime Noise Estimation Algorithm Spec"
[5] "FSM SDS"