SAM R34/R35 Microchip LoRaWAN™ Stack Software API Reference Manual

Introduction

Microchip LoRaWAN Stack (MLS) Software API provides an interface to the different software modules. This document describes how to configure and enable functionalities of the API software. A general description of each API is provided including the functionalities, syntax, responses, and an example. The API description defines the parameter with its type, range (valid /acceptable values), the default value (when available), and the factory-programmed value (when applicable).

Default value is set automatically if the parameter is omitted and at the software reset (if the command setting is not stored in NVM). The factory-programmed value is set at the software reset when the setting is not modified with respect to the manufacturer setting; it is valid for the commands that store the setting in Nonvolatile Memory (NVM).

MLS provides APIs for following software modules:

- LoRaWAN MAC Layer (MAC)
- LoRaWAN Radio Layer (TAL)
- Persistent Data Server (PDS)
- Power Manager Module (PMM)
- Hardware Abstraction Layer (HAL)

Table of Contents

1. Quick Reference Info

1.1 Reference Documentation

Following documents provide for further details:

- SAM R34 MLS Getting Started Guide (DS50002812)
- SAM R34/R35 Low Power LoRa® Sub-GHz SiP Data Sheet (DS70005356)
- SAM R34 Xplained Pro User Guide (DS50002803)
- WLR089 Xplained Pro User Guide (DSxxxxxxxx)

1.2 Acronyms and Abbreviations Used

Table 1-1. Acronyms and Abbreviations Used

Acronym	Abbreviation
ABP	Activation By Personalization
ADR	Adaptive Data Rate
APPEUI	Application End Unique Identifier
ASF	Advanced Software Framework
DEVEUI	End Device Unique Identifier
DR	Data Rate
EDBG	Embedded Debugger
FREQ	Frequency
LoRa	Long Range Modulation
LoRaWAN	Long Range Wide Area Network
LPWAN	Low Power Wide Area Network
MAC	Media Access Controller
MLS	Microchip LoRaWAN Stack
OTAA	Over-The-Air Activation
PDS	Persistent Data Storage
PMM	Power Management Module
TAL	Transceiver Abstraction Layer
UART	Universal Asynchronous Receiver/Transmitter

1.3 LoRaWAN Stack Directory Structure

The LoRaWAN stack code base is available in the directory present in the package (`src/ASF/thirdparty/wireless/lorawan`).

Table 1-2. LoRaWAN Stack Directory Structure

Directory	Description
hal	Contains implementation for radio's hardware interface, timers and so on.
inc	Contains common include file(s)
libgen	Contains the static library for LoRaWAN MAC and TAL
mac	Contains headers of LoRaWAN MAC layer specification independent of regional parameters
pmm	Contains Power Management Module (PMM)
regparams	Contains implementation of MAC layer functionality specific to the regional bands.
services	Contains modules such as software timer, PDS, and AES
sys	Contains system modules such as task manager, power management, and initialization
tal	Contains transceiver related headers, drivers for supported transceivers

2. LoRaWAN API

2.1 MAC API

Note: All LoRaWAN MAC APIs and structures are present in the included file `lorawan.h`.

2.1.1 LORAWAN_Init

Definition: This function initializes the LoRaWAN MAC stack and radio software layers. During this initialization procedure:

- Software timers required for MAC operations are created
- Application callback routine function pointers are stored in data base (DB)
- Radio is initialized
- MAC-related PDS files are registered

Syntax

```
void LORAWAN_Init(AppDataCb_t appdata, JoinResponseCb_t joindata);
```

Input Parameters

Table 2-1. Input Parameters

Parameter Name	Parameter Type	Description
`appdata`	`AppDataCb_t`	Pointer to function that is called when a down-link is received
`joindata`	`JoinResponseCb_t`	Pointer to function that is called when join response is received

Return Type and Values

<void>

API Type – Synchronous

2.1.2 LORAWAN_Reset

Definition: This function automatically resets the LoRaWAN stack software and initializes the stacks with the parameters for the selected ISM band. During this Reset routine:

- MAC DB is initialized with default parameters
- LoRaWAN regional parameters module is initialized
- Radio layer default DB initialization is triggered

The Reset routine must be called after every software reset of the stack. To change the regional band dynamically, Reset routine calls the new ISM band, which un-initializes an old regional parameter and re-initiates the new ISM band. If the ISM band is same as the one stored in DB, the regional parameter initializes the same default ISM band.

Syntax

```
StackRetStatus_t LORAWAN_Reset (IsmBand_t ismBand);
```

Input Parameters

Table 2-2. Input Parameters

Parameter Name	Parameter Type	Description
ismBand	IsmBand_t	ISM band types. Refer to Table 2-37 for a list of defined ISM band types.

Return Type and Values

Table 2-3. Return Type

Parameter Name	Parameter Type	Description
StackRetStatus_t	ENUM	Enumerated values containing all return types from LoRaWAN layers

Table 2-4. Return Values

Return Value	Reason
LORAWAN_SUCCESS	LoRaWAN stack is successfully being reset and default values are restored
LORAWAN_INVALID_PARAMETER	Given ISM band is invalid

API Type – Synchronous

2.1.3 LORAWAN_Join

Definition: This API initiates the LoRaWAN join procedure and activates the end device to successfully connect to the LoRaWAN network.

Syntax

```
StackRetStatus_t LORAWAN_Join(ActivationType_t activationTypeNew);
```

Input Parameters

Table 2-5. Input Parameters

Parameter Name	Parameter Type	Description
activationTypeNew	ActivationType_t	Activation type: • LORAWAN_OTAA = 0 • LORAWAN_ABP = 1.

Return Type and Values

Table 2-6. Return Type

Parameter Name	Parameter Type	Description
StackRetStatus_t	ENUM	Enumerated values containing all return types from LoRaWAN layers

Table 2-7. Return Values

Return Value	Reason
LORAWAN_SUCCESS	LoRaWAN join procedure is successfully initiated

...........continued

Return Value	Reason
LORAWAN_MAC_PAUSED	LoRaWAN MAC layer is paused. Join procedure will only happen in Active state
LORAWAN_SILENT_IMMEDIATELY_ACTIVE	The server decided that any further up-link transmission is not possible from this end device
LORAWAN_NWK_JOIN_IN_PROGRESS	Already one join procedure is in progress. MAC cannot initiate join procedure until previous request is completed.
LORAWAN_BUSY	MAC layer is not IDLE. Until other transaction is completed, MAC cannot initiate join procedure
LORAWAN_KEYS_NOT_INITIALIZED	For initiating join procedure, keys need to be available by MAC Layer. If keys are not set, MAC layer will not initiate join procedure.

API Type – Asynchronous

2.1.4 LORAWAN_Send

Definition: This API starts a bidirectional communication process and initiates the data packet send procedure. This API returns immediately after copying the data payload to MAC buffer and posting a task to MAC scheduler. MAC transmits the data packet and wait for down-link packet(s). After down-link procedure is completed, MAC layer calls the application callback function and that ends this API procedure.

Syntax

```
StackRetStatus_t LORAWAN_Send (LorawanSendReq_t *lorasendreq);
```

Input Parameters

Table 2-8. Input Parameter

Parameter Name	Parameter Type	Description
lorasendreq	LorawanSendReq_t	LoRaWAN send request – 1. Transmission type 2. Port value 3. Pointer to application payload buffer 4. Length of application payload

Return Type and Values

Table 2-9. Return Type

Parameter Name	Parameter Type	Description
StackRetStatus_t	ENUM	Enumerated values containing all return types from LoRaWAN layers

Table 2-10. Return Values

Return Value	Reason
LORAWAN_SUCCESS	LoRaWAN send request is successfully initiated

...........continued

Return Value	Reason
LORAWAN_MAC_PAUSED	LoRaWAN MAC layer is paused. Join procedure will only happen in Active state
LORAWAN_SILENT_IMMEDIATELY_ACTIVE	The server decided that any further up-link transmission is not possible from this end device
LORAWAN_INVALID_PARAMETER	Port number is wrong
LORAWAN_BUSY	MAC layer is not IDLE. Until other transaction is completed, MAC cannot initiate data packet send procedure
LORAWAN_NWK_NOT_JOINED	LoRaWAN end device is not joined to the network
LORAWAN_INVALID_BUFFER_LENGTH	Buffer length exceeds maximum payload size
LORAWAN_FCNTR_ERROR_REJOIN_NEEDED	Re-joining is required.

API Type – Asynchronous

2.1.5 LORAWAN_SetAttr

Definition: This API is used to set various LoRaWAN MAC attributes that are stored in the MAC data base (DB).

Syntax

```
StackRetStatus_t LORAWAN_SetAttr(LorawanAttributes_t attrType, void *attrValue);
```

Input Parameters

Table 2-11. Input Parameters

Parameter Name	Parameter Type	Description
attrType	LorawanAttributes_t	List of LoRaWAN attributes. Refer to Table 2-38 for a list of defined attrType names.
attrValue	Void pointer	Value of attribute type.

Return Type and Values

Table 2-12. Return Type

Parameter Name	Parameter Type	Description
StackRetStatus_t	ENUM	Enumerated values containing all return types from LoRaWAN layers

Table 2-13. Return Values

Return Value	Reason
LORAWAN_SUCCESS	LoRaWAN join procedure is successfully initiated
LORAWAN_INVALID_PARAMETER	Set attribute type is invalid
LORAWAN_BUSY	MAC layer is not IDLE. Set attribute function cannot be performed.

API Type – Synchronous

2.1.6 LORAWAN_GetAttr

Definition: This API is used to get various LoRaWAN MAC attributes that are stored in the MAC data base.

Syntax

```
StackRetStatus_t LORAWAN_GetAttr(LorawanAttributes_t attrType, void *attrInput, void
*attrOutput);
```

Input Parameters

Table 2-14. Input Parameters

Parameter Name	Parameter Type	Description
attrType	LorawanAttributes_t	List of LoRaWAN attributes. Refer to Table 2-38 for a list of defined attrType names.
attrInput	Void pointer	Pointer to attribute input value
attrOutput	Void pointer	Pointer to output value

Return Type and Values

Table 2-15. Return Type

Parameter Name	Parameter Type	Description
StackRetStatus_t	ENUM	Enumerated values containing all return types from LoRaWAN layers

Table 2-16. Return Values

Return Value	Reason
LORAWAN_SUCCESS	LoRaWAN join procedure is successfully initiated
LORAWAN_INVALID_PARAMETER	Set attribute type is invalid
LORAWAN_BUSY	MAC layer is not IDLE. Set attribute function cannot be performed

API Type – Synchronous

2.1.7 LORAWAN_Pause

Definition: This function pauses the LoRaWAN stack functionality to allow transceiver (radio) configuration and functionality to be performed. Using "`mac pause`", radio commands can be generated between a LoRaWAN protocol up-link application and the LoRaWAN protocol receive windows. This function will reply with the time interval in milliseconds that the transceiver can be used without affecting the LoRaWAN functionality.

Syntax

```
uint32_t LORAWAN_Pause (void);
```

Input Parameters

<None>

Return Type and Values

Table 2-17. Return Type

Parameter Name	Parameter Type	Description
`Uint32_t`	Integer	• Returns the number in milliseconds representing how much it can be paused without affecting the functionality • Returns ‘0’ if it cannot be paused, maximum value when in Idle mode

API Type – Synchronous

2.1.8 LORAWAN_Resume

Definition: This function resumes the LoRaWAN stack functionality, in order to continue normal functionality after being paused.

Syntax

```
void LORAWAN_Resume (void);
```

Input Parameters

<None>

Return Type and Values

<None>

API Type – Synchronous

2.1.9 LORAWAN_ForceEnable

Definition: The network can issue certain commands that would require the end device to go the Silent Immediately state. This mechanism disables any further communication of the module, effectively isolating it from the network. Using this function after this network command has been received restores the module’s connectivity by allowing it to send data.

Syntax

```
void LORAWAN_ForceEnable (void);
```

Input Parameters

<None>

Return Type and Values

<None>

API Type – Synchronous

2.1.10 LORAWAN_ReadyToSleep

Definition: This function is used for querying the stack’s readiness for sleep. This function has a dependency on radio for the corresponding readiness check function in TAL.

Syntax

```
bool LORAWAN_ReadyToSleep(bool deviceResetAfterSleep);
```

Input Parameters

Table 2-18. Input Parameter

Parameter Name	Parameter Type	Description
deviceResetAfterSleep	boolean	• 'true' means device is reset during the wake up • 'false' means device is not reset during wake up

Return Type and Values

Table 2-19. Return Type

Parameter Name	Parameter Type	Description
Bool	Boolean	• 'true' – stack is in Ready state to sleep or • 'false'

API Type – Synchronous

2.2 TAL API

Note: All LoRaWAN TAL APIs and structures are present in the included file `radio_interface.h`.

2.2.1 RADIO_Init

Definition: This API initializes the radio software module. During this initialization routine:

- Radio DB is updated with default parameters
- Call-back routines for transmit and receive are updated in radio layer
- DIO interrupt handlers are initialized
- SX1276 transceiver is initialized and put into sleep after successful initialization

Syntax

```
void RADIO_Init(void);
```

Input Parameters

<None>

Return Type and Values

<None>

API Type – Synchronous

2.2.2 RADIO_Receive

Definition: This function receives the data and stores it in the buffer pointer space by doing a task post to the `RADIO_RxHandler`.

Syntax

```
RadioError_t RADIO_Receive(RadioReceiveParam_t *param);
```

Input Parameters

Table 2-20. Input Parameter

Parameter Name	Parameter Type	Description
param	RadioReceiveParam_t	A structure for storing the receive parameters.

Using RadioReceiveParam_t, upper layers can control time window for receive operation, indefinite receive open or receive stop.

Return Type and Values

Table 2-21. Return Type

Parameter Name	Parameter Type	Description
RadioError_t	ENUM	Enumerated values containing all return types from radio layer

Table 2-22. Return Values

Return Value	Reason
ERR_RADIO_BUSY	Radio is not in IDLE state
ERR_NONE	Radio in IDLE state and configuring transceiver to Receive state is initiated
ERR_INVALID_REQ	Radio is already in Receive state

API Type – Asynchronous

2.2.3 RADIO_Transmit

Definition: This function transmits the data by doing a task post to the RADIO_TxHandler.

Syntax

```
RadioError_t RADIO_Transmit(RadioTransmitParam_t *param);
```

Input Parameter

Table 2-23. Input Parameter

Parameter Name	Parameter Type	Description
param	RadioTransmitParam_t	A structure for storing the transmit parameters

Return Type and Values

Table 2-24. Return Type

Parameter Name	Parameter Type	Description
RadioError_t	ENUM	Enumerated values containing all return types from radio layer

Table 2-25. Return Values

Return Value	Reason
ERR_RADIO_BUSY	Radio is not in IDLE state
ERR_NONE	Radio in IDLE state and configuring transceiver to Transmit state is initiated
ERR_DATA_SIZE	Data buffer in transmit request is greater than max size (64)

API Type – Asynchronous

2.2.4 RADIO_TransmitCW

Definition: This function transmits a continuous wave. This API uses radio parameters (such as Frequency, Modulation, Spreading factor and so on) stored in TAL data base to transmit the continuous wave. Radio parameters can be configured using `RADIO_SetAttr` API described in 2.2.6 RADIO_SetAttr. If user did not configure any parameters, this API will use default parameters stored in data base. All default values for the radio layer are given in the 2.3.3 Radio/TAL Attributes.

Syntax

```
RadioError_t RADIO_TransmitCW(void);
```

Input Parameters

<None>

Return Type and Values

Table 2-26. Return Type

Parameter Name	Parameter Type	Description
`RadioError_t`	ENUM	Enumerated values containing all return types from radio layer

Table 2-27. Return Values

Return Value	Reason
`ERR_RADIO_BUSY`	Radio is not in IDLE state
`ERR_NONE`	Radio in IDLE state and starts the continuous transmission

API Type – Synchronous

2.2.5 RADIO_StopCW

Definition: This function stops the transmission of continuous wave.

Syntax

```
RadioError_t RADIO_StopCW(void);
```

Input Parameters

<None>

Return Type and Values

Table 2-28. Return Type

Parameter Name	Parameter Type	Description
`RadioError_t`	ENUM	Enumerated values containing all return types from radio layer

Table 2-29. Return Values

Return Value	Reason
`ERR_RADIO_BUSY`	Radio is not in IDLE state
`ERR_NONE`	Radio in IDLE state and stopping the continuous transmission

API Type – Synchronous

2.2.6 RADIO_SetAttr

Definition: This function writes the given value to the specified attribute.

Syntax

```
RadioError_t RADIO_SetAttr(RadioAttribute_t attribute, void *value);
```

Input Parameter

Table 2-30. Input Parameter

Parameter Name	Parameter Type	Description
`attribute`	`RadioAttribute_t`	Structure for attribute list. Refer to Table 2-39 for a list of defined attribute names.

Return Type and Values

Table 2-31. Return Type

Parameter Name	Parameter Type	Description
`RadioError_t`	ENUM	Enumerated values containing all return types from radio layer

Table 2-32. Return Value

Return Value	Reason
`ERR_RADIO_BUSY`	Radio is not in IDLE state
`ERR_NONE`	Radio in IDLE state and stopping the continuous transmission

API Type – Synchronous

2.2.7 RADIO_GetAttr

Definition: This function gets the stored value of the specified attribute.

Syntax

```
RadioError_t RADIO_GetAttr(RadioAttribute_t attribute, void *value);
```

Input Parameter

Table 2-33. Input Parameter

Parameter Name	Parameter Type	Description
`attribute`	`RadioAttribute_t`	Structure for attribute list. Refer to Table 2-39 for a list of defined attribute names.

Return Type and Values

Table 2-34. Return Type

Parameter Name	Parameter Type	Description
`RadioError_t`	ENUM	Enumerated values containing all return types from radio layer

Table 2-35. Return Values

Return Value	Reason
`ERR_RADIO_BUSY`	Radio is not in IDLE state
`ERR_NONE`	Radio in IDLE state and stopping the continuous transmission

API Type – Synchronous

2.3 Stack Attributes

2.3.1 Regional Configuration Parameters

Note: All LoRaWAN Regional Configuration Parameters are present in the included file `conf/conf_regparams.h`.

Table 2-36. Regional Configuration Parameters

Macro Definition	Default Value	Description
`MAC_DEF_TX_POWER_<r>`	• `For AS: 1` • `For AU: 7` • `For EU: 1` • `For IN: 1` • `For JP: 1` • `For NA: 7`	Transmission power table index
`MAC_DEF_TX_CURRENT_DATARATE_<r>`	• `For AS: DR3` • `For AU: DR3` • `For EU: DR3` • `For IN: DR3` • `For JP: DR3` • `For NA: DR2`	Initial data rate to be used by application for up-link
`MAC_DATARATE_MIN<r>`	• `For AS: DR7` • `For AU: DR6` • `For EU: DR7` • `For IN: DR7` • `For JP: DR7` • `For NA: DR4`	Minimum data rate to be used by end device
`MAC_DATARATE_MAX_r`	`DR0` (all regions)	Maximum data rate to be used by end device

Note:

1. The `<r>` value is replaced by a 2-letter region identifier. The possible region identifiers are: NA, AS, AU, EU, IN, JP, and KR.

Table 2-37. ISM Band Types

ISM Band	Description
ISM_EU868	EU 863 - 870MHz ISM Band
ISM_EU433	EU 433MHz ISM Band

...........continued

ISM Band	Description
ISM_NA915	North America
ISM_AU915	Australia
ISM_KR920	South Korea
ISM_JPN923	Japan
ISM_BRN923	Brunei
ISM_CMB923	Cambodia
ISM_INS923	Indonesia
ISM_LAOS923	Laos
ISM_NZ923	New Zealand
ISM_SP923	Singapore
ISM_TWN923	Taiwan
ISM_THAI923	Thailand
ISM_VTM923	Vietnam
ISM_IND865	India

2.3.2 LoRaWAN/MAC Attributes

Note: All LoRaWAN MAC Attributes are set or read using the LORAWAN_SetAttr, or LORAWAN_GetAttr APIs, respectively. Refer to 2.1.5 LORAWAN_SetAttr and 2.1.6 LORAWAN_GetAttr.

Table 2-38. LoRaWAN/MAC Attributes

Name (attrType)	Type	Range	Address	Default
ACKTIMEOUT	uint16	0x0000-0xffff	Read/Write	0x7D0
ADR	bool	True (Enabled) False (Disabled)	Read/Write	–
ADR_ACKDELAY	uint8	0x00-0xff	Read/Write	0x20
ADR_ACKLIMIT	uint8	0x00-0xff	Read/Write	0x40
APPS_KEY	uint8[16]	–	Read/Write	–
APP_EUI	uint8[8]	–	Read/Write	–
APP_KEY	uint8[16]	–	Read/Write	–
AUTOREPLY	bool	True, False	Read/Write	–
BATTERY	uint8	0x00-0xff	Read/Write	0xff (Battery level invalid)
CH_PARAM_FREQUENCY	uint32	0x00000000-0xffffffff	Read/Write	–
CH_PARAM_DR_RANGE	uint8	–	Read/Write	–
CH_PARAM_STATUS	bool	True, False	Read/Write	–

...........continued

Name (attrType)	Type	Range	Address	Default
CURRENT_DATARATE	uint8	DR0-DR7	Read/Write	DR0
DEV_ADDR	uint32	0x00000000-0xffffffff	Read/Write	–
DEV_EUI	uint8[8]	–	Read/Write	–
DOWNLINK_COUNTER	uint32	0x00000000-0xffffffff	Read/Write	–
EDCLASS	uint8	0 (Class A), 1 (Class B), 2 (Class C)	Read/Write	0 (Class A)
EDCLASS_SUPPORTED	uint8	1 (Class A), 5 (Class A and Class C)	Read/Write	LORAWAN_SUPPORTED_ED_CLASSES
FHSS_CALLBACK	FHSSCallback_t (function pointer)	Valid function address	Read/Write	–
ISMBAND	uint8	0x00-0xff	Read Only	ISM_EU868
JOINACCEPT_DELAY1	uint16	0x0000-0xffff	Read/Write	0x1388
JOINACCEPT_DELAY2	uint16	0x0000-0xffff	Read/Write	0x1770
LINK_CHECK_GWCNT	uint8	0x00-0xff	Read Only	0x00
LINK_CHECK_MARGIN	uint8	0x00-0xff	Read Only	0xff
LINK_CHECK_PERIOD	uint16	0x0000-0xffff	Read/Write	0x0000
LORAWAN_STATUS	uint32	0x00000000-0xffffffff	Read Only	0x00
MAX_FCOUNT_GAP	uint16	0x0000-0xffff	Read/Write	0x4000
MCAST_APPS_KEY	uint8[16]	–	Read/Write	–
MCAST_ENABLE	bool	True, False	Read/Write	0x00
MCAST_FCNT_DOWN	uint16	0x0000-0xffff	Read Only	–
MCAST_GROUP_ADDR	uint32	0x00000000-0xffffffff	Read/Write	–
MCAST_NWKS_KEY	uint8[16]	–	Read/Write	–
NWKS_KEY	uint8[16]	–	Read/Write	–
CNF_RETRANSMISSION_NUM	uint8	0x00-0xff	Read/Write	–
UNCNF_REPETITION_NUM	uint8	0x00-0xff	Read/Write	–
RX2_WINDOW_PARAMS	ReceiveWindow2Params_t (ENUM)	–	Read/Write	Specific to region
RX_DELAY1	uint16	0x0000-0xffff	Read/Write	0x3e8

...........continued

Name (attrType)	Type	Range	Address	Default
RX_DELAY2	uint16	0x0000-0xffff	Read/Write	0x7d0
SYNC_WORD	uint8	0x00-0xff	Read/Write	0x34
TX_POWER	uint8	For EU: 0 to 5, for NA: 5,7,8,9,10	Read/Write	For EU: 1, For NA: 7
UPLINK_COUNTER	uint32	0x00000000-0xffffffff	Read/Write	0x00000000

2.3.3 Radio/TAL Attributes

Note: All radio/TAL Attributes are set or read using the RADIO_SetAttr, or RADIO_GetAttr APIs respectively. Refer to 2.2.6 RADIO_SetAttr and 2.2.7 RADIO_GetAttr.

Table 2-39. Radio/TAL Attributes

Name (attribute)	Type	Range	Access	Default
BANDWIDTH	RadioLoRaBandwidth_t (ENUM)	enumerated values	Read/Write	BW_125KHZ
CHANNEL_FREQUENCY	uint32	valid frequency value	Read/Write	EU: FREQ_868100KHZ, NA: FREQ_923300KHZ
CHANNEL_FREQUENCY_DEVIATION	uint32	0x00000000 – 0xffffffff	Read/Write	0x61a8
CRC_ON	uint8	0x00 (Disabled), 0x01 (Enabled)	Read/Write	0x01 (Enabled)
ERROR_CODING_RATE	RadioErrorCodingRate_t (ENUM)	enumerated values	Read/Write	CR_4_5
FREQUENCY_HOP_PERIOD	uint16	0x0000 - 0xffff	Read Only	0x0000
FSK_AFC_BW	RadioFSKShaping_t (ENUM)	enumerated values	Read/Write	FSKBW_83_3KHZ
FSK_BIT_RATE	uint32	0x00000000 – 0xffffffff	Read/Write	0xc350
FSK_DATA_SHAPING	RadioFSKShaping_t (ENUM)	enumerated values	Read/Write	FSK_SHAPING_GAUSS_BR_0_5
FSK_RX_BW	RadioFSKShaping_t (ENUM)	enumerated values	Read/Write	FSK_BW_50_0KHZ
FSK_SYNC_WORD	uint8 [8]	0x00 - 0xff	Read/Write	{0xc1, 0x94, 0xc1}
FSK_SYNC_WORD_LEN	uint8	0x00 - 0x08	Read/Write	0x03

...........continued

Name (attribute)	Type	Range	Access	Default
IQINVERTED	uint8	0x00 (Disabled), 0x01 (Enabled)	Read/Write	0x00 (Disabled)
LORA_SYNC_WORD	uint8	0x00 - 0xff	Read/Write	0x34
MODULATION	RadioModulation_t (ENUM)	enumerated values	Read/Write	MODULATION_LORA
OUTPUT_POWER	uint8	0x00 - 0xff	Read/Write	0x01
PABOOST	uint8	0x00 - 0xff	Read/Write	0x01
PACKET_SNR	int8	-128 to +127	Read Only	-128
PREAMBLE_LEN	uint16	0x0000 - 0xffff	Read/Write	0x08
RADIO_CALLBACK	RadioCallback_t (function pointer)	Valid function address	Read/Write	–
RADIO_LBT_PARAMS	RadioLBT_t (structure)	structure member type	Read/Write	{0x00}
SPREADING_FACTOR	RadioDataRate_t (ENUM)	enumerated values	Read/Write	SF_12
WATCHDOG_TIMEOUT	uint32	0x00000000 – 0xffffffff	Read/Write	0x3a98

3. Supporting MAC Layers

3.1 Regional Band Layer

Regional band APIs are not to be used by an application directly. All API and attribute configuration of a regional band must happen via MAC layer. Necessary attributes are available as part of MAC attribute list and the application can use that to configure the regional band layer.

3.2 PMM Layer

MLS provides Power Management Module (PMM) in the stack. An application running on top of the MLS can choose to use PMM to save power during idle times. Power saving is done by switching the MCU to one of the available low-power modes. Currently, PMM is supported only on the SAM R34 microcontroller. In SAM R34, currently, STANDBY and BACKUP Sleep modes are supported by PMM.

3.2.1 PMM Files

Table 3-1. PMM Files

Files	Description
`pmm/inc/pmm.h`	This file contains the public API for the Power Management Module. This needs to be included by the application if power management is needed.
`pmm/src/pmm.c`	This file contains the implementation of power management that is the conditions for entering to sleep, calculating the sleep time, configuring and backing up the timers and so on.
`hal/inc/sleep_timer.h`	This file contains the APIs for the sleep timer. It is used by the PMM to keep track of the sleep duration and also to wake-up the device in timed sleep scenarios.
`hal/src/sleep_timer/sam0/sleep_timer.c`	This file contains the implementation of sleep timer. Currently, sleep timer uses RTC internally.
`hal/inc/sleep.h`	This file contains the API to switch the MCU to desired Sleep mode.
`hal/src/sleep/sam0/src/sleep.c`	This file contains the implementation to switch the MCU to Sleep mode. Currently, the SAM R34 supports the STANDBY and BACKUP mode.

3.2.2 PMM APIs

Note: All PMM APIs are present in the included file `pmm/inc/pmm.h`.

3.2.2.1 PMM_Sleep

Definition: This function puts the system to sleep if possible.

Syntax

```
PMM_Status_t PMM_Sleep(PMM_SleepReq_t *req);
```

Input Parameter

Table 3-2. Input Parameter

Parameter Name	Parameter Type	Description
`req`	`PMM_SleepReq_t`	Pointer to sleep request structure when being called from application

Note: `PMM_SleepReq_t` – Definition is available in `pmm.h`.

Return Type and Values

Table 3-3. Return Type

Parameter Name	Parameter Type	Description
`PMM_Status_t`	ENUM	Describes the status of power manager for a sleep request

Note: `PMM_Status_t` – Definition is available in `pmm.h`.

Table 3-4. Return Values

Return Value	Reason
`PMM_SLEEP_REQ_DENIED`	PMM denies the request because system is not ready to sleep at the instance of sleep call
`PMM_SLEEP_REQ_PROCESSED`	Power manager accepted and have already processed the request

API Type – Synchronous

3.3 PDS Layer

Persistent Data Server (PDS) module facilitates storing of stack parameters/attributes in Nonvolatile Memory (NVM) of MCU. The PDS module interfaces between NVM driver and stack.

3.3.1 PDS Files

Table 3-5. PDS Files

Files	Description
`services/pds/inc/pds_common.h`	Header file for PDS common typedefs and structures
`services/pds/inc/pds_interface.h`	The PDS interface file which provides the API definitions for external layers
`services/pds/inc/pds_nvm.h`	The PDS NVM header file which contains NVM abstractions for PDS
`services/pds/inc/pds_task_handler.h`	The PDS driver task manager header file which calls PDS task scheduler
`services/pds/inc/pds_wl.h`	The PDS wear leveling header file which contains PDS wear leveling headers
`services/pds/src/pds_interface.c`	The PDS interface source file which has implementations for all public API's
`services/pds/src/pds_nvm.c`	The PDS NVM source file which contains NVM abstraction for PDS
`services/pds/src/pds_task_handler.c`	The PDS driver task manager source file which contains PDS task scheduler

...........continued

Files	Description
`services/pds/src/pds_wl.c`	The PDS wear leveling source file which contains PDS wear leveling implementation

3.3.2 PDS_Init

Definition: Initialize the PDS software module.

Syntax

```
PdsStatus_t PDS_Init(void);
```

Input Parameter

<None>

Return Type and Values

Table 3-6. Return Type

Parameter Name	Parameter Type	Description
`PdsStatus_t`	ENUM	PDS status codes. Definition available in `pds_interface.h`

Table 3-7. Return Values

Return Value	Reason
`PDS_OK`	PDS operation is success
`PDS_ERROR`	NVM driver initialization failed
`PDS_NOT_ENOUGH_MEMORY`	EEPROM_SIZE configured in `conf_nvm.h` is greater than HW configured size in user page

API Type – Synchronous

3.3.3 PDS_UnInit

Definition: This API will disable storing the data in PDS.

Syntax

```
PdsStatus_t PDS_UnInit(void);
```

Input Parameter

<None>

Return Type and Value

Table 3-8. Return Type

Parameter Name	Parameter Type	Description
`PdsStatus_t`	ENUM	PDS status codes. Definition available in `pds_interface.h`

Table 3-9. Return Value

Return Value	Reason
`PDS_OK`	PDS operation is success

API Type – Synchronous

3.3.4 PDS_Store

Definition: This function sets the store operation bit in the file-marks for the item in PDS.

Syntax

```
dsStatus_t PDS_Store(PdsFileItemIdx_t pdsFileItemIdx, uint8_t item);
```

Input Parameters

Table 3-10. Input Parameters

Parameter Name	Parameter Type	Description
`pdsFileItemIdx`	`PdsFileItemIdx_t`	PDS file ID of the item
`item`	`uint8_t`	PDS item ID

Return Type and Values

Table 3-11. Return Type

Parameter Name	Parameter Type	Description
`PdsStatus_t`	ENUM	PDS status codes. Definition available in `pds_interface.h`

Table 3-12. Return Values

Return Value	Reason
`PDS_OK`	PDS operation is success
`PDS_INVALID_FILE_IDX`	File ID is invalid

API Type – Asynchronous

3.3.5 PDS_Restore

Definition: This function restores an item from PDS to RAM.

Syntax

```
PdsStatus_t PDS_Restore(PdsFileItemIdx_t pdsFileItemIdx, uint8_t item);
```

Input Parameters

Table 3-13. Input Parameters

Parameters Name	Parameters Type	Description
`pdsFileItemIdx`	`PdsFileItemIdx_t`	PDS file ID of the item
`item`	`uint8_t`	PDS item ID

Return Type and Values

Table 3-14. Return Type

Parameter Name	Parameter Type	Description
`PdsStatus_t`	ENUM	PDS status codes. Definition available in `pds_interface.h`

Table 3-15. Return Values

Return Value	Reason
PDS_OK	PDS operation is success
PDS_INVALID_FILE_IDX	File ID is invalid
PDS_NOT_FOUND	Item ID is not found in File ID or File reference is not in NVM
PDS_ITEM_DELETED	Item is deleted from the PDS

API Type – Synchronous

3.3.6 PDS_Delete

Definition: This function will set the delete operation for the item in the file ID bit mask.

Syntax

```
PdsStatus_t PDS_Delete(PdsFileItemIdx_t pdsFileItemIdx, uint8_t item);
```

Input Parameters

Table 3-16. Input Parameters

Parameter Names	Parameter Types	Description
pdsFileItemIdx	PdsFileItemIdx_t	PDS file ID of the item
item	uint8_t	PDS item ID

Return Type and Values

Table 3-17. Return Type

Parameter Name	Parameter Type	Description
PdsStatus_t	ENUM	PDS status codes. Definition available in pds_interface.h

Table 3-18. Return Values

Return Value	Reason
PDS_OK	PDS operation is success
PDS_INVALID_FILE_IDX	File ID is invalid

API Type – Asynchronous

3.3.7 PDS_IsRestorable

Definition: This function checks if all the registered files are restorable.

Syntax

```
bool PDS_IsRestorable(void);
```

Input Parameters

<None>

Return Type and Values

Table 3-19. Return Type

Parameter Name	Parameter Type	Description
`bool`	Boolean	• 'true' – Valid PDS data is available in EEPROM • 'false' – No valid data is available in EEPROM

API Type – Synchronous

3.3.8 PDS_DeleteAll

Definition: This function will erase all the items stored in the PDS.

Syntax

```
PdsStatus_t PDS_DeleteAll(void);
```

Input Parameters

<None>

Return Type and Value

Table 3-20. Return Type

Parameter Name	Parameter Type	Description
`PdsStatus_t`	ENUM	PDS status codes. Definition available in `pds_interface.h`

Table 3-21. Return Value

Return Value	Reason
`PDS_OK`	PDS operation is success

API Type – Synchronous

3.3.9 PDS_RestoreAll

Definition: This function will restore all the items from the PDS to RAM from all registered files.

Syntax

```
PdsStatus_t PDS_RestoreAll(void);
```

Input Parameters

<None>

Return Type and Values

Table 3-22. Return Type

Parameter Name	Parameter Type	Description
`PdsStatus_t`	ENUM	PDS status codes. Definition available in `pds_interface.h`

Table 3-23. Return Values

Return Value	Reason
`PDS_OK`	PDS operation is success
`PDS_NOT_FOUND`	PDS read from NVM failed

API Type – Synchronous

3.3.10 PDS_StoreAll

Definition: This function will set the store operation to all the items stored in all the registered files in PDS.

Syntax

```
PdsStatus_t PDS_StoreAll(void);
```

Input Parameters

<None>

Return Type and Values

Table 3-24. Return Type

Parameter Name	Parameter Type	Description
PdsStatus_t	ENUM	PDS status codes. Definition available in pds_interface.h

Table 3-25. Return Value

Return Value	Reason
PDS_OK	PDS operation is success

API Type – Synchronous

3.3.11 PDS_RegFile

Definition: This function registers a file to the PDS.

Syntax

```
PdsStatus_t PDS_RegFile(PdsFileItemIdx_t argFileId, PdsFileMarks_t argFileMarks);
```

Input Parameters

Table 3-26. Input Parameters

Parameter Names	Parameter Types	Description
argFileId	PdsFileItemIdx_t	PDS file ID of the item.
argFileMarks	PdsFileMarks_t	This structure contains details about number of items in file, RAM address for each item and size of each item in a list form. This will be stored in PDS and used to retrieve data during PDS operations.

Return Type and Values

Table 3-27. Return Type

Parameter Name	Parameter Type	Description
PdsStatus_t	ENUM	PDS status codes. Definition available in pds_interface.h.

Table 3-28. Return Values

Return Value	Reason
`PDS_OK`	PDS operation is success
`PDS_INVALID_FILE_IDX`	File ID in the argument is not valid one

API Type – Synchronous

3.3.12 PDS_UnRegFile

Definition: This function un-registers a file to the PDS. This makes the data stored in the NVM invalid and any operation on this file ID will be invalid after this point.

Syntax

```
PdsStatus_t PDS_UnRegFile(PdsFileItemIdx_t argFileId);
```

Input Parameters

Table 3-29. Input Parameter

Parameter Name	Parameter Type	Description
`argFileId`	`PdsFileItemIdx_t`	PDS file ID of the item

Return Type and Values

Table 3-30. Return Type

Parameter Name	Parameter Type	Description
`PdsStatus_t`	ENUM	PDS status codes. Definition available in `pds_interface.h`.

Table 3-31. Return Values

Return Value	Reason
`PDS_OK`	PDS operation is success
`PDS_INVALID_FILE_IDX`	File ID in the argument is not valid one

API Type – Synchronous

3.4 Software Timer Module

Timer provides the facility to measure desired amount of time. The SAM R34 has five hardware timers that can measure only five individual amounts of time simultaneously. Every component in MLS such as radio, MAC and APP have timer requirements. Therefore, timers need to be efficiently shared by all the required components.

The software timer module provides the needed abstractions for MLS to use timers. It handles the operation of the hardware timer, for measuring the given amount of time, thus freeing the user from managing the hardware timers.

The software timer provides a set of interfaces to initialize, create, start, stop timers and so on. If the user calls timer start, the software timer module takes Timer duration and callback function as input. Once the duration is elapsed, user-supplied callback function is invoked.

Simultaneously, more than one software timer can be started, which is automatically sorted according to their duration and expires accordingly. Besides, running for user-desired duration, the software timer module also keeps track of system time. System time is measured starting from the initialization of the software timer during reset.

MLS currently supports a maximum of 25 software timer instances. It can be customized as per application requirements by changing the `TOTAL_NUMBER_OF_TIMERS` macro in the `conf_app.h` file.

Note: To change the number of software timers, the user must rebuild an application firmware.

3.4.1 Software Timer Files

Table 3-32. Software Timer Files

Files	Description
`services/sw_timer/inc/sw_timer.h`	Header file for software timer module
`services/ sw_timer/inc/sw_timer.c`	Software timer module source file

3.4.2 Software Timer APIs

3.4.2.1 SwTimerCreate

Definition: Returns a timer ID to be used before starting a timer. Timer ID is taken from common software timer free pool.

Syntax

```
StackRetStatus_t SwTimerCreate(uint8_t *timerId);
```

Input Parameters

Table 3-33. Input Parameters

Parameter Name	Parameter Type	Description
`timerId`	`uint8_t`	Timer ID of the item

Return Type and Values

Parameter Name	Parameter Type	Description
`StackRetStatus_t`	ENUM	List of enumerated values for return status

Return Value	Reason
`LORAWAN_SUCCESS`	New timerId is allocated
`LORAWAN_RESOURCE_UNAVAILABLE`	There is no more timerId to allocate

API Type – Synchronous

3.4.2.2 SwTimerGetTime

Definition: Get current system time. Returns the system time in micro seconds.

Syntax

```
uint64_t SwTimerGetTime(void);
```

Input Parameters

<None>

Return Type and Values

Table 3-34. Return Type

Parameter Name	Parameter Type	Description
`uint64_t`	64-bit unsigned integer	System time in micro seconds

API Type – Synchronous

3.4.2.3 SystemTimerInit

Definition: Initializes the software timer module. Timer free pool array gets initialized. Hardware (HW) timer callback assignment and TC initialization will be carried out.

Syntax

```
void SystemTimerInit(void);
```

Input Parameters

<None>

Return Type and Values

<None>

API Type – Synchronous

3.4.2.4 SwTimerIsRunning

Definition: Checks whether a given timer is running or not.

Syntax

```
bool SwTimerIsRunning(uint8_t timerid);
```

Input Parameters

Table 3-35. Input Parameter

Parameter Name	Parameter Type	Description
`timerId`	uint8_t	Timer identifier

Return Type and Values

Table 3-36. Return Type

Parameters Name	Parameter Type	Description
`bool`	Boolean	• ‘true’ – Timer is running • ‘false’ – Timer is stopped or not started

API Type – Synchronous

3.4.2.5 SwTimerReset

Definition: Resets the software timer module. Clears the SW timer pool.

Syntax

```
void SystemTimerReset(void);
```

Input Parameters

<None>

Return Type and Values

<None>

API Type – Synchronous

3.4.2.6 SwTimerStart

Definition: This function starts a regular timer and installs the corresponding callback function handling the timeout event.

Syntax

```
StackRetStatus_t SwTimerStart(uint8_t timerId, uint32_t timerCount,
SwTimeoutType_t timeoutType, void *timerCb, void *paramCb);
```

Input Parameters

Table 3-37. Input Parameters

Parameter Name	Parameter Type	Description
`timerId`	uint8_t	Timer identifier
`timerCount`	uint32_t	Timeout in microseconds
`timeoutType`	`SwTimeoutType_t`	• SW_TIMEOUT_RELATIVE – The timeout is relative to the current time • SW_TIMEOUT_ABSOLUTE – The timeout is an absolute value
`timerCb`	Void pointer	Callback handler invoked upon timer expiry
`paramCb`	Void pointer	Argument for the callback handler

Return Type and Values

Table 3-38. Return Type

Parameters Name	Parameter Type	Description
StackRetStatus_t	ENUM	List of enumerated values for return Status

Table 3-39. Return Values

Return Value	Reason
`LORAWAN_SUCCESS`	Timer is started successfully
`LORAWAN_INVALID_REQUEST`	Timer is already running
`LORAWAN_INVALID_PARAMETER`	Timer ID, timeout type or timeout value is wrong

API Type – Synchronous

3.4.2.7 SwTimerStop

Definition: Stops a running timer. This API stops a running timer with specified timerId - Timer identifier

Syntax

```
StackRetStatus_t SwTimerStop(uint8_t timerId);
```

Input **Parameters**

Table 3-40. Input Parameter

Parameter Name	Parameter Type	Description
timerId	uint8_t	Timer identifier

Return Type and Values

Table 3-41. Return Type

Parameter Name	Parameter Type	Description
StackRetStatus_t	ENUM	List of enumerated values for return status

Table 3-42. Return Values

Return Value	Reason
LORAWAN_SUCCESS	Timer is started successfully
LORAWAN_INVALID_REQUEST	Timer is not running
LORAWAN_INVALID_PARAMETER	Timer ID value is wrong

API Type – Synchronous

4. HAL APIs

4.1 HAL Files

Table 4-1. HAL Files

Files	Description
`hal/inc/radio_driver_hal.h`	Header file for HAL
`hal/src/radio_driver_hal.c`	HAL source file

4.2 HAL_RadioInit

Definition: This function initializes the radio hardware SPI interface, DIO and Reset pins.

Syntax

```
Void HAL_RadioInit (void)
```

Input Parameters

<None>

Return Type and Values

<None>

API Type - Synchronous

4.3 HAL_RadioDeInit

Definition: This function de-initializes the radio hardware SPI interface.

Syntax

```
Void HAL_RadioDeInit (void)
```

Input Parameters

<void>

Return Type and Values

<void>

API Type - Synchronous

4.4 RADIO_Reset

Definition: This function resets the Radio hardware by pulling the reset pin low.

Syntax

```
 Void RADIO_Reset (void)
```

Input Parameters

<void>

Return Type and Values

<void>

API Type - Synchronous

4.5 RADIO_RegisterWrite

Definition: This function is used to write a byte of data to the radio register

Syntax

```
void RADIO_RegisterWrite(uint8_t reg, uint8_t value)
```

Input Parameters

Parameter Names	Parameter Type	Description
reg	uint8_t	Register address to be written
value	uint8_t	Value to be written into the radio register

Return Type and Values

<void>

API Type - Synchronous

4.6 RADIO_RegisterRead

Definition: This function is used to read a byte of data from the radio register.

Syntax

```
Uint8_t RADIO_RegisterRead(uint8_t reg)
```

Input Parameters

Parameters Name	Parameter Type	Description
reg	uint8_t	Register address to be read

Return Type and Values

Parameters Name	Parameter Type	Description
Uint8_t	8-bit unsigned integer	Value read from the radio register

API Type - Synchronous

4.7 RADIO_FrameWrite

Definition: This function is used to write a stream of data into the radio frame buffer.

Syntax

```
void RADIO_FrameWrite(uint8_t offset, uint8_t* buffer, uint8_t bufferLen)
```

Input Parameters

Parameters Name	Parameter Type	Description
offset	Uint8_t	FIFO offset to be written to
buffer	Uint8_t *	Pointer to the data to be written into the frame buffer
bufferLen	Uint8_t	Length of the data to be written

Return Type and Values

<void>

API Type – Synchronous

4.8 RADIO_FrameRead

Definition: This function is used to read a stream of data from the radio frame buffer.

Syntax

```
void RADIO_FrameRead(uint8_t offset, uint8_t* buffer, uint8_t bufferLen)
```

Input Parameters

Parameters Name	Parameter Type	Description
offset	Uint8_t	FIFO offset to be read from
buffer	Uint8_t *	Pointer to the data to be read from the frame buffer
bufferLen	Uint8_t	Length of the data to be read from the frame buffer

Return Type and Values

<void>

API Type – Synchronous

4.9 HAL_DisableRFCtrl

Definition: This function is called by the stack to indicate HAL for resetting the Rfctrl GPIO pins to their inactive state. Based on RFCTRL1 and RFCTRL2 values, different GPIOs will be controlled.

Syntax

```
void HAL_DisableRFCtrl(RFCtrl1_t RFCtrl1, RFCtrl2_t RFCtrl2)
```

Input Parameters

Parameters Name	Parameter Type	Description

RFCtrl1	ENUM	RF Control 1 indicates the FREQUENCY band (Higher UHF or Lower UHF) or PA Boost enabled.
		RFO_LF = 0
		RFO_HF = 1
		PA_BOOST = 2

...........continued		
Parameters Name	**Parameter Type**	**Description**

RFCtrl2	ENUM	RF Control 2 indicates Transmit or Receive path
		RX = 0
		TX = 1

Return Types and Values

<void>

API Type - Synchronous

4.10 HAL_EnableRFCtrl

Definition: This function is called by the stack to inform HAL which RF path signal is used for TX/RX and, based on the RF front-end design, which respective GPIO pin will be controlled.

Usage of the RFCTRL1 variable:

The RFCtrl1 parameter is used to select the RF output frequency range (or band). The following values are applicable for the parameter.

- RFO_LF is used when the RF output frequency range is lesser than 525 MHz (band 2/3).
- RFO_HF is used when the RF output frequency range is greater than 779 MHz (band 1).
- PA_BOOST is used regardless of the RF output frequency. But, when the TX power is greater than +15 dBm.
- When the TX power is between -4 dBm to +15 dBm, either RFO_LF or RFO_HF should be used. The following table describes the frequency range and TX power of RFCtrl1.

Note: For more details on the frequency limits and required bands, refer to the SX1276 data sheet.

Table 4-2. RFCtrl1 Frequency Range and Power

ENUM Value	Frequency Region	TX Power
RFO_LF	< 525 MHz	-4 to +15 dBm
RFO_HF	From 779 MHz	-4 to +15 dBm
PA_BOOST	All frequencies	+2 to +17 dBm

- All 3 ENUM values represent a pin out from the SX1276 transceiver.
- The RFCtrl1 value indicates which of these 3 pin-outs the signal will be received at. For example, if the frequency is 868MHz and the output power is +10dBm, then the signal will come from the RFO_HF pin.
- Refer to the SX1276 data sheet, RF power amplifiers section for more details on the transceiver operation with different frequencies and output power.
- Refer to the SAM R34 data sheet for the pin mapping details for these 3 pins.

Usage of RFCTRL2 variable:

The RFCtrl2 parameter is used to select either a transmit or receive operation by the transceiver.

- For a receive operation, front-end RF circuitry routes the received signal through either the RFI_LF or RFI_HF pin.
- Based on the frequency range, either RFI_LF or RFI_HF is selected.
- The frequency range can be determined from the RFCtrl1 parameter to select between RFI_LF and RFI_HF.

Table 4-3. Master Truth Table for RFCtrl1 and RFCtrl2

RFCtrl1	RFCtrl2	Operation
RFO_LF	TX	• Transmit operation • Transmit Frequency – < 525 MHz • Output power – -4 to +15 dBM
RFO_HF	TX	• Transmit operation • Transmit Frequency – >779 MHz • Output power – -4 to +15 dBM
PA_BOOST	Do not care	• Transmit operation • Transmit Frequency – All • Output power – >15 dBm
RFO_LF	RX	• Receive Operation • Receive Frequency – < 525 MHz
RFO_HF	RX	• Receive Operation • Receive Frequency – > 779 MHz

For the SAM R34 Xplained Pro board or WLR089 Xplained Pro board:

1. Only RFO_HF and PA_BOOST values of RFCtrl1 are valid and used to set the BAND_SELECT pin. Refer to the SAM R34 Xplained Pro board or WLR089 Xplained Pro board for more details on the front-end RF circuit design.
2. RFO_LF is not used because neither the SAM R34 Xplained Pro nor WLR089 Xplained Pro supports the frequency operation in < 525 MHz.
3. The RFCtrl2 input is not used. The SAM R34 Xplained Pro or WLR089 Xplained Pro front-end RF circuit does not need to control based on the TX and RX operation.

Syntax

```
void HAL_EnableRFCtrl(RFCtrl1_t RFCtrl1, RFCtrl2_t RFCtrl2)
```

Table 4-4. Input Parameters

Parameters Name	Parameter Type	Description
RFCtrl1	ENUM	RF Control 1 indicates the FREQUENCY band (Higher UHF or Lower UHF) or PA Boost enabled. • RFO_LF = 0 • RFO_HF = 1 • PA_BOOST = 2
RFCtrl2	ENUM	RF Control 2 indicates the Transmit or Receive path. • RX = 0 • TX = 1

Return Types and Values

<void>

API Type – Synchronous

5. Document Revision History

Revision	Date	Section	Description
A	10/2018	Document	Initial Revision
B	09/2020	1.1 Reference Documentation	Added WLR089 Xplained Pro user guide in the reference documentation list
		4.10 HAL_EnableRFCtrl	Added WLR089 Xplained Pro board support related changes

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable." Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication is provided for the sole purpose of designing with and using Microchip products. Information regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION. Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

ISBN: 978-1-5224-6657-4

Quality Management System

For information regarding Microchip’s Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4485-5910
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-72884388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820