mmWaveLink Interface Control Document

xWRLx432

Version: 1.2

10th July, 2025

Table of contents

List of figures

List of tables

Chapter 1
Introduction

1.1 Overview

mmWave DFP is collection of foundational API software, firmware, drivers, and ROM components associated with mmWave devices. mmWave DFP provides seamless control and configuration of Radar Frontend Controller Sub System (FECSS) in real-time either from an external host device in case of Front End (FE) mmWave sensors or from integrated M4 APPSS core in case of SOC.

The mmWave DFP is responsible to provide API function calls to all RF front end related functionalities and configurations using these APIs application can calibrate the RF device, generate radar waveforms and get functional safety monitoring results from FECSS. These APIs enables the RF transceiver to be self-contained and capable of adapting itself to dynamic conditions, such as temperature changes, functional safety capable and minimize the RF related knowledge/dependencies and algorithms run part of application software.

TI low cost and low power, low end mmWave radar products are highly-integrated 60GHz and 77GHz CMOS millimeter wave devices for Automotive and Industrial applications. The device integrated with all of the RF and Analog functionality, including VCO, PLL, PA, LNA, Mixer and ADC for multiple TX/RX channels into a single chip in a low power digital architecture.

1.2 mmWave DFP Architecture

The mmWave DFP Low consists of 3 major components, these components closely associated with the functionality of underlying device sensor front-end hardware IPs and implementation.

i. mmWaveLink Library

This is an open-source, OS and device agnostic API functions for user control and configuration of the RF analog FECSS in the device. The mmWaveLink is the user interface to the FECSS. The mmWaveLink layer can be integrated with SDK(SoC)/External Host (Frontend)

ii. FECSSLib Library

The FECSS open-source ROM/RAM drivers library is responsible for management and operation of the Radar sensor frontend functions and waveform generation in the device. The mmWaveLink APIs internally call these FECSSLib driver APIs to configure the FECSS.

iii. RFS Firmware

The ROM/patch binary is responsible for executing pre-defined TI device specific RF analog sequences, the dedicated FECSS M3 core runs these sequence based on commands from FECSS library or mmWaveLink. The patch file is loaded in the device patch RAM during boot which is used to implement bug fixes and minor feature additions in RFS ROM.

1.2.1 mmWaveLink Architecture

mmWaveLink AIPs are categorized in four sections:

- mmWaveLink Initialization APIs: APIs to initialize and de-initialize mmWaveLink interface

- mmWaveLink Device Configuration APIs: APIs to control and configure FECSS
- mmWave Sensor APIs: APIs to control and configure FMCW Chirp, Burst and Frame parameters
- mmWaveLink Monitor Configuration APIs: APIs to control and configure monitors

These APIs require access to application and OS specific interfaces to execute desired functionality. These functions are accessed by callback function pointers configured using the mmWaveLink Initialization API. Callback configurations are categorized in following groups:

- Platform Configuration Details
- Debug Interface Callback Functions
- Communication Interface Callback Functions
- Application Specific Interface Callback Functions
- OS Interface Callback Functions

1.2.2 FECSSLib Architecture

FECSSLib drives are categorized in following three sections:

- Device Drivers: Driver function to control front-end hardware modules
- Sensor Drivers: Driver functions to control and configure sensor hardware modules like Frame Timer and Chirp Timer
- RFS Drivers: Driver functions to interact with RFS firmware

1.2.3 Supported Devices

APIs defined in this document support following mmWave Low devices

- xWRL6432
  - RF Frequency: 56GHz - 64GHz
  - Package Types: FCCSP, WCSP
- xWRL6432
  - RF Frequency: 56GHz - 63.5GHz
  - Package Types: AOP
- xWRL1432
  - RF Frequency: 76GHz - 81GHz
  - Package Types: FCCSP

1.3 Terms and Abbreviations

Abbreviation / Term	Description
DFP	Device Firmware Package
RFS	Radar Front-end Scripter

<table>
<tr><th>Abbreviation / Term</th><th>Description</th></tr>
<tr><td>FECSS</td><td>Front End Controller Sub-System</td></tr>
<tr><td>AppSS</td><td>Application Sub-System</td></tr>
<tr><td>XTAL</td><td>External Crystal Oscillator</td></tr>
<tr><td>DIG_PLL</td><td>Digital PLL</td></tr>
<tr><td>APLL</td><td>Analog PLL</td></tr>
<tr><td>XTAL_FREQ</td><td>External Crystal Oscillator</td></tr>
<tr><td>DIG_PLL_FREQ</td><td>DIG_PLL_FREQ is designed frequency and derived from the Oscillator frequency.
$DIG\_PLL\_FREQ = \frac{M}{2N} \times XTAL\_FREQ$
DIG_PLL_FREQ should be 160MHz and the integers M and N values should be selected accordingly.
The recommended values are as follows:
<table>
<tr><th>XTAL_FREQ</th><th>M</th><th>N</th><th>DIG_PLL_FREQ</th></tr>
<tr><td>25.0 MHz</td><td>128</td><td>10</td><td>160.0 MHz</td></tr>
<tr><td>26.0 MHz</td><td>124</td><td>10</td><td>161.2 MHz</td></tr>
<tr><td>38.4 MHz</td><td>168</td><td>20</td><td>161.28 MHz</td></tr>
<tr><td>40.0 MHz</td><td>160</td><td>20</td><td>160 MHz</td></tr>
</table></td></tr>
<tr><td>FAST clock</td><td>Fast clock is a clock derived out of the DIG_PLL.
The frequency of the clock is DIG_PLL_FREQ for the AppSS core and DIG_PLL_FREQ / 2 for the FECSS core.</td></tr>
<tr><td>APLL_FREQ</td><td>Analog PLL frequency.
The frequency of the APLL depends on the input XTAL frequency. Typical output frequencies are as follows:
<table>
<tr><th>XTAL_FREQ</th><th>APLL_FREQ</th></tr>
<tr><td>25.0 MHz</td><td>400.0 MHz</td></tr>
<tr><td>26.0 MHz</td><td>403.0 MHz</td></tr>
<tr><td>38.4 MHz</td><td>403.2 MHz</td></tr>
<tr><td>40.0 MHz</td><td>400 MHz</td></tr>
</table></td></tr>
<tr><td>CW Mode</td><td>Continuous Waveform Mode
This is a special debug mode which enables the transmitters at a constant frequency (w_ChirpRfFreqStart).</td></tr>
<tr><td>FT</td><td>Frame Timer</td></tr>
<tr><td>CT</td><td>Chirp Timer</td></tr>
</table>

Abbreviation / Term	Description
PD	Peak Detector
RDIF	Radar Data Interface
LODIST_BIAS_CODES	Lo Distribution bias current control codes
FECSS_MON_EVENT	FECSS Monitoring Done Interrupt Event. This interrupt is mapped to the 4th IRQ channel of the AppSS processor
IFA_GAIN_CODE & RFA_GAIN_CODE	Rx signal chain contains two amplifiers, one in RF section and another in IF section. Overall Rx gain is sum of gains from these two stages. These codes controls the gain of respective stages
TX_PA_CODE	Tx Power Amplifiers (PA) control codes This is a 2-byte value which controls the output power

1.4 Scope and Intended Audience

Scope of this document is to describe the mmWaveLink API interface for TI mmWave Low devices.

The intended audience for this document is firmware, application software, and validation engineers needing to understand the format and contents of all API interfaces between the mmWave DFP and the application.

Chapter 2
mmWaveLink APIs

mmWaveLink APIs provide seamless API interface to mmWave Sensor FECSS RF Front End. mmWaveLink Library is a high level software API layer in DFP, which interface with FECSS Library, FECSSLib provides low level driver functions to configure/control FECSS hardware. mmWaveLink API internally calls FECSSLib drivers to perform the API job and returns the status to application.

The Application software or SDK interfaces with mmWave DFP using mmWaveLink APIs. The communication interface between Application software and mmWave DFP in mmWaveLink layer uses command-response protocol, there will be a response for all commands to mmWave DFP (FECSS RF Front End). The command normally contains configuration message data structure and response contains return information.

mmWaveLink APIs can be broadly categorized in two types:

A. Set APIs

Set APIs are commands which configures the FECSS with some predefined parameters, the response to this command will hold only status info, error codes and few generic API execution information. The Device and sensor configuration APIs are typical SET APIs.

B. Get APIs

Get APIs are commands which seek certain info from FECSS, command can have predefined parameters, the response to this command will hold status info, error codes, few generic API information along with GET info. The Monitor, GPADC measurement APIs are typical GET APIs.

2.1 mmWaveLink API Structure and Grouping

mmWaveLink APIs are defined using following elements:

A. API Command Id

Each mmWave API has a unique 16-bit unsigned command id which can be used by the external application to implement API based communication.

Note Command Ids are not used in the mmWaveLink DFP API functions.

B. API Function

Unique name of the mmWaveLink API function. All of these function start with prefix `rl_`. Example: rl_fecssDevPwrOn

C. API Configuration Structure

C-structures defined in the mmWaveLink header files to hold the inputs for the API. If applicable, mmWaveLink APIs accept pointer to the corresponding configuration structure.

D. API Response Structure

C-structures defined in the mmWaveLink header files to return the inputs for the API. If applicable, mmWaveLink APIs accept pointers to the corresponding response structure.

APIs for mmWave Low devices are categorized in four groups:

A. Initialization APIs

These APIs initialize and de-initialize mmWaveLink layer in the control application. Initialization involves critical tasks such as registering necessary interrupts, getting call-back functions from the application. De-initialize clears the handles created during the Initialization.

B. Device Configuration APIs

Device control APIs provide device level configuration and control options for the mmWave Low front end such as power control, calibrations, clock settings, etc.

C. Sensor Configuration APIs

Sensor APIs provide interface to configure and control the FMCW chirp parameters.

D. Monitor Configuration APIs

Monitor APIs provide interface to configure and trigger monitors on Functional Safety enabled devices along with few debug APIs.

2.2 mmWaveLink Initialization APIs

mmWaveLink initialization APIs configure an interface between the application and mmWaveLink handler. These APIs accept number of call-back functions required for the operation of mmWaveLink in application environment. Link layer assumes correctness of the callback functions provided by application.

Warning Incorrect callback function configuration may result in undefined behavior.

mmWaveLink Initialization includes following APIs:

- mmWaveLink Initialization API
- mmWaveLink De-Initialization API

2.2.1 mmWaveLink Initialization API

This API initializes all the interface call-back function and client context handlers for mmWaveLink and FECSSLib.

Note Callback functions are implementation defined and provided by the application. mmWaveLink callback functions just standardize the interface by defining function pointers

Assumptions

- Host application should call this API one for every front-end device in multi-chip cascade configuration with same call-back function configuration.
- mmWaveLink callback function structures assumes return type to be `T_RETURNTYPE`, which is a `SINT32` number indicating execution status of the API.
  - **0**: Success
  - **Any Negative Number**: Error / Failure

API Definition

API Index	0x0001
API Function Name	`rl_mmWaveLinkInit`
API Configuration Structure	`T_DFP_CLIENT_CB_DATA`
API Response Structure	None

▶ API Function

mmWaveLink C function declaration

```
T_RETURNTYPE rl_mmWaveLinkInit(UINT32 w_deviceMap, T_DFP_CLIENT_CB_DATA z_clientCbData);
```

Configuration C Structure

Table 2.1: T_DFP_CLIENT_CB_DATA

Field	Data Type	Size	Offset
c_PlatformType	UINT8	1	0
c_DeviceType	UINT8	1	1
c_ApiErrorCheckDis	UINT8	1	2
c_ApiDbglogEn	UINT8	1	3
h_ApiRespTimeoutMs	UINT16	2	4
h_Reserved1	UINT16	2	6
z_OsiCb	T_DFP_OSI_CB_DATA	36	8
z_PltfCb	T_DFP_PLATFORM_CB_DATA	36	44
z_DbgCb	T_DFP_DBG_PRINT_CB_DATA	8	80
z_ComIfCb	T_DFP_COMIF_CB_DATA	16	88
Total		**104**	

Configuration Fields

c_PlatformType

Device Platform Type.
This value defines the mode of operation of the mmWaveLink.

- M_DFP_DEVICE_PLATFORM_SOC : For mmWaveLink running on the AppSS
- M_DFP_DEVICE_PLATFORM_FE : For mmWaveLink running on external HOST

Note: mmWave Low devices only support SOC platform and appropriate value of this field is defined in macro M_DFP_DEVICE_PLATFORM_TYPE.

c_DeviceType

Device Type
This field defines the exact device type. Following table shows the list of supported devices:

Device	mmWaveLink Macro
xWRL6432	M_DFP_DEVICETYPE_6432
xWRL1432	M_DFP_DEVICETYPE_1432

- `c_ApiErrorCheckDis`

mmWaveLink API Error Check Disable.
Setting odd value (preferably 1) disables the error checks performed in mmWaveLink APIs. This feature can be used to reduce the API delay in production SW on non safety devices.

Recommendation It is recommended to do enable check always on safety devices.

- `c_ApiDbglogEn`

mmWaveLink API debug logger enable control.
Setting odd value (preferably 1) enables the debug log prints from mmWaveLink APIs. This feature relies on a callback function provided by the application (configured in `p_Print`) to print the logs.

- `h_ApiRespTimeoutMs`

mmWaveLink API response wait timeout in milliseconds.
mmWaveLink API returns an error if the API fails to execute or generate response in the specified time period.
Valid Range: $[0, 65535]$
Format: $1LSB = 1ms$, Typical value should be 10ms

- `z_OsiCb`

Operating system callbacks.
mmWaveLink APIs require number of interface functions provided by the application operating system. These functions are called via function pointers inside the mmWaveLink and FECSSLib drivers. Following table defines the structure which holds these OS callbacks.

Table 2.2: `T_DFP_OSI_CB_DATA`

Field	Data Type	Size	Offset
`z_Mutex`	`T_DFP_OSI_MUTEX_CB_DATA`	16	0x0
`z_Semp`	`T_DFP_OSI_SEMP_CB_DATA`	16	0x10
`z_Queue`	`T_DFP_OSI_MSGQ_CB_DATA`	4	0x20
Total		**36**	

`z_Mutex`

OS Mutex callback function data
This Structure hold callback function pointers for the functions required for handling mutexes.

Table 2.3: T_DFP_OSI_MUTEX_CB_DATA

Field	Data Type	Size	Offset
p_OsiMutexCreate	T_RETURNTYPE (*p_OsiMutexCreate)(T_DFP_OSI_MUTEX_HDL *, UINT8 *)	4	0x0
p_OsiMutexLock	T_RETURNTYPE (*p_OsiMutexLock)(T_DFP_OSI_MUTEX_HDL, UINT32)	4	0x4
p_OsiMutexUnLock	T_RETURNTYPE (*p_OsiMutexUnLock)(T_DFP_OSI_MUTEX_HDL)	4	0x8
p_OsiMutexDelete	T_RETURNTYPE (*p_OsiMutexDelete)(T_DFP_OSI_MUTEX_HDL)	4	0xc
Total		**16**	

- p_OsiMutexCreate :
  This function should create a mutex object and initialize the handler using reference pointer. The function should accept a pointer to uninitialized mutex handler and name for the mutex and return T_RETURNTYPE response.
- p_OsiMutexLock :
  This functions should attempt to lock the initialized mutex using the handler and timeout value parameters. The function should return M_DFP_RET_CODE_RADAR_OSIF_ERROR in case of failure to acquire lock.
- p_OsiMutexUnLock :
  This function should unlock the mutex using the handler and return M_DFP_RET_CODE_RADAR_OSIF_ERROR in case of failure.
- p_OsiMutexDelete :
  This function should de-initialize the mutex using the handler and return T_RETURNTYPE response.

z_Semp

OS Semaphore callback function data

This Structure hold callback function pointers for the functions required for handling semaphores.

Table 2.4: T_DFP_OSI_SEMP_CB_DATA

Field	Data Type	Size	Offset
p_OsiSemCreate	T_RETURNTYPE (*p_OsiSemCreate)(T_DFP_OSI_SEM_HDL *, UINT8 *)	4	0x0
p_OsiSemWait	T_RETURNTYPE (*p_OsiSemWait)(T_DFP_OSI_SEM_HDL, UINT32)	4	0x4
p_OsiSemSignal	T_RETURNTYPE (*p_OsiSemSignal)(T_DFP_OSI_SEM_HDL)	4	0x8
p_OsiSemDelete	T_RETURNTYPE (*p_OsiSemDelete)(T_DFP_OSI_SEM_HDL)	4	0xc
Total		**16**	

- p_OsiSemCreate :
  This function should create a semaphore object and initialize the handler using reference pointer. The function should accept a pointer to uninitialized semaphore handler and name for the mutex and return T_RETURNTYPE response.
- p_OsiSemWait :
  This functions should wait on the initialized semaphore using the handler and timeout value parameters. The function should return after the timeout.
- p_OsiSemSignal :
  This function should perform post on the semaphore using the handler and return M_DFP_RET_CODE_RADAR_OSIF_ERROR in case of failure.
- p_OsiSemDelete :
  This function should de-initialize the semaphore using the handler and return T_RETURNTYPE response.

z_Queue

OS message queue and task spawn data
This Structure hold callback function pointers for the functions required for creating message queues and spawn tasks.

Table 2.5: T_DFP_OSI_MSGQ_CB_DATA

Field	Data Type	Size	Offset
p_OsiSpawn	T_RETURNTYPE (*p_OsiSpawn)(T_DFP_OSI_SPAWN_HANDLER , const void* , UINT32)	4	0x0
Total		**4**	

- p_OsiSpawn :
  This function should spawn a totally independent new task. This callback function is reserved for future use.

▶ z_PltfCb

Platform callback function pointers.
This structure contains callbacks for application platform specific functions like serial interface handlers, delay, etc.

Table 2.6: T_DFP_PLATFORM_CB_DATA

Field	Data Type	Size	Offset
p_Delay	T_RETURNTYPE (*p_Delay)(UINT32)	4	0x0
p_TimeStamp	UINT32 (*p_TimeStamp)(void)	4	0x4
p_RegisterIsr	T_RETURNTYPE (*p_RegisterIsr)(UINT8 , T_DFP_PLT_ISR_HANDLER , void* , T_DFP_PLT_HWI_HDL*)	4	0x8
p_DeRegisterIsr	T_RETURNTYPE (*p_DeRegisterIsr)(T_DFP_PLT_HWI_HDL)	4	0xc
p_MaskIsr	T_RETURNTYPE (*p_MaskIsr)(T_DFP_PLT_HWI_HDL)	4	0x10
p_UnMaskIsr	T_RETURNTYPE (*p_UnMaskIsr)(T_DFP_PLT_HWI_HDL)	4	0x14
p_EnterCriticalRegion	UINT64 (*p_EnterCriticalRegion)(void)	4	0x18
p_ExitCriticalRegion	void (*p_ExitCriticalRegion)(UINT64)	4	0x1c
w_TimeStampCounterMask	UINT32	4	0x20
Total		**36**	

- p_Delay :
  This is a function pointer to the delay function which receives delay value with $1LSB = 1us$. Application may choose to implementation in non-blocking mode.
- p_TimeStamp :
  This is a function pointer which could be used to get a time-stamp, preferably SysTick based time-stamp.
- p_RegisterIsr :
  This is function pointer to the function which registers the FECSS mailbox interrupt (INT_FEC_INTR0) for mmWaveLink APIs to function. mmWaveLink Initialization API calls this function internally. THe function should perform following tasks:
  - Register the ISR handler provided by mmWaveLink for FECSS mailbox interrupt
  - Configure FECSS mailbox interrupt priority (Priority Recommendation: Highest)
  - Clear pending mailbox interrupt
  - Enable mailbox interrupt
- p_DeRegisterIsr :
  This is function pointer to the function which de-registers the FECSS mailbox interrupt. This function is called in mmWaveLink De-Initialization API.
- p_MaskIsr :
  Callback function to mask required interrupt. As of now callback is not used in mmWave Low devices.
- p_UnMaskIsr :
  Callback function to un-mask required interrupt. As of now callback is not used in mmWave Low devices.

- p_EnterCriticalRegion :
  Callback function to enter a critical region (Uninterrupted mode with all interrupts and task switching disabled). As of now callback is not used in mmWave Low devices.
- p_ExitCriticalRegion :
  Callback function to exit a critical region. As of now callback is not used in mmWave Low devices.
- w_TimeStampCounterMask :
  This value represents a 32-bit mask for the timestamp returned from p_TimeStamp. The value of this mask should be $2^w - 1$ where w is a bit-width of the time-stamp counter.

▶ z_DbgCb

mmWaveLink debug configuration data

Table 2.7: T_DFP_DBG_PRINT_CB_DATA

Field	Data Type	Size	Offset
p_Print	T_DFP_PRINT_FUNCP	4	0x0
p_RfsdbgData	UINT32*	4	0x4
Total		8	

- p_Print :
  printf like function to print the debug logs from mmWaveLink and FECSSLib.
  Note: Debug logs will be disable if the callback function pointer is NULL.
- p_RfsdbgData :
  Pointer to the 64-byte buffer to store RFS debug log which is read at the every API. This debug data is in TI specific encoded format. Application should provide a separate buffer for every device in a multi-chip configuration.

▶ z_ComIfCb

Communication interface callback data
This structure holds the callback configuration for communication interfaces such as SPI, UART etc. These callbacks are required only in multi-chip configuration where the mmWaveLink runs on the host.

Table 2.8: T_DFP_COMIF_CB_DATA

Field	Data Type	Size	Offset
p_ComIfOpen	T_DFP_COMIF_HDL (*p_ComIfOpen)(UINT8, UINT32)	4	0x0
p_ComIfRead	SINT32 (*p_ComIfRead)(T_DFP_COMIF_HDL, UINT8 *, const UINT8 *, UINT32)	4	0x4
p_ComIfWrite	SINT32 (*p_ComIfWrite)(T_DFP_COMIF_HDL, UINT8 *, const UINT8 *, UINT32)	4	0x8
p_ComIfClose	T_RETURNTYPE (*p_ComIfClose)(T_DFP_COMIF_HDL)	4	0xc
Total		**16**	

- p_ComIfOpen :
  Callback function to open requested communication interface.
- p_ComIfRead :
  Callback function to read data using the communication interface handle.
- p_ComIfWrite :
  Callback function to write data using the communication interface handle.
- p_ComIfClose :
  Callback function to close the communication interface.

2.2.2 mmWaveLink De-Initialization API

This API de-initializes the mmWaveLink call-back function and client context handlers for mmWaveLink and FECSSLib. The API should be called at the end of the application to free-up the resources used by mmWaveLink and FECSSLib.

API Definition

API Index	
API Function Name	rl_mmWaveLinkDeInit
API Input	Device map with 1-bit per device
API Response Structure	None

▶ API Function

mmWaveLink C function declaration

```
T_RETURNTYPE rl_mmWaveLinkDeInit(UINT32 w_deviceMap);
```

2.3 mmWaveLink Device Configuration APIs

mmWaveLink Device APIs provide interface to configure an control the FECSS. Following diagram shows functional blocks of the device module.

mmWaveLink provides following APIs to control device:

- FECSS Device Power On API
- FECSS Device Power OFF API
- DFP Version Get API
- FECSS RF Power On Off API
- FECSS Factory Calibration API
- FECSS RF Runtime Calibration API
- FECSS RF Clock Bandwidth Configuration API
- FECSS GPADC Measurement Configuration API
- FECSS GPADC Measurement Trigger API
- FECSS Temperature Measurement Config API
- FECSS Temperature Measurement Trigger API
- FECSS Factory Calibration Data Get API
- FECSS Factory Calibration Data Set API
- FECSS Device Clock Ctrl API
- FECSS Device RDIF Control API
- FECSS Device Status Get API
- FECSS RF Status Get API
- FECSS RF Fault Status Get API
- FECSS Die Id Get API
- FECSS Rx-Tx Calibration Data Get API
- FECSS Rx-Tx Calibration Data Set API
- FECSS RFS Debug Configuration API
- FECSS RF Runtime Tx CLPC API

2.3.1 FECSS Device Power On API

This API function contains the device power ON control, power modes and clock configuration settings. The application should use this API to powerup the FECSS before issuing other functional APIs.

API Definition

API Index	0x0003U
API Function Name	rl_fecssDevPwrOn
API Configuration Structure	T_RL_API_FECSS_DEV_PWR_ON_CMD
API Response Structure	None

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_DEV_PWR_ON            ((UINT16)0x0003U)
// API Function Definition
    T_RETURNTYPE rl_fecssDevPwrOn(UINT8 c_devIndex, const T_RL_API_FECSS_DEV_PWR_ON_CMD
    ↪  *p_apiCmdData);
```

Configuration C Structure

Table 2.9: T_RL_API_FECSS_DEV_PWR_ON_CMD

Field	Data Type	Size	Offset
h_XtalClkFreq	UINT16	2	0
c_ClkSourceSel	UINT8	1	2
c_PowerMode	UINT8	1	3
c_ChirpTimerResol	UINT8	1	4
c_FecBootCfg	UINT8	1	5
h_Reserved2	UINT16	2	6
w_Reserved3	UINT32	4	8
Total		**12**	

Configuration Fields

h_XtalClkFreq

Device XTAL clock frequency
The RFS software requires operating XTAL frequency to configure front-end and maintain time.

Unit: 1 LSB = 1/256 MHz

XTAL Frequency	Value
25MHz	6400
26MHz	6656
38.4MHz	9830
40MHz	10240

Warning Following table lists the supported XTAL frequencies per device:

Device	Supported XTALs
xWRL6432	40 MHz
xWRL1432	40 MHz

- `c_ClkSourceSel`

FECSS clock source selection during power up. FECSS can power up on either XTAL for FAST clock. Application choose the desired source based on requirements.

Value	Definition
0x00	XTAL clock source
0x0A	Fast clock source, fast clock can be either DIG_PLL or APLL clock set by application. Clock Frequency: DIG_PLL/2 (Typical 80MHz with 40MHz XTAL).

Recommendation Fast Clock should be used to reduce FECSS power-up time

- `c_PowerMode`

FECSS power up mode, The power up mode can be either cold or warm boot type.

Value	Definition	mmWaveLink Macro
0x00	**Cold Boot** If c_PowerMode is 0x00, FECSS power up is performed in cold boot and the RFS data memory is re-initialized during power up. The RFS code memory should be initialized by boot loader/application before calling this API. Typical use case: First time boot or nReset.	M_RL_FECSS_PWR_ON_MODE_COLD
0x0A	**Warm Boot** If c_PowerMode is 0x0A, FECSS power up is performed in warm boot and the RFS data memory is retained un-initialized during power up. To support warm boot mode, the FECSS power down must be done with c_RetentionMode = 0x0A, this will retain the FECSS memory in deep sleep. Typical use case: Deep sleep exit and software reset.	M_RL_FECSS_PWR_ON_MODE_WARM

Warning The warm boot can be done only if power down is done with c_RetentionMode = 0x0A to retain the FECSS memory.

- c_ChirpTimerResol

The RF start frequency resolution and timing related chirp parameters resolution selection.

Value	Definition
0x0	Low resolution mode.
0x1	High resolution mode.

Bit Definition:

Value	Definition
0	Frequency Resolution • Low resolution mode supports 16-bit frequency value. • High resolution mode supports 24-bit frequency value. This resolution setting is applicable for w_ChirpRfFreqStart.
1	Time Resolution • Low resolution mode supports 1 LSB = 40/APLL_FREQ (Typical 100ns with APLL_FREQ = 400MHz). • High resolution mode supports 1 LSB = 8/APLL_FREQ (Typical 20ns with APLL_FREQ = 400MHz). This resolution setting is applicable for h_ChirpIdleTime, h_ChirpRampEndTime and w_BurstPeriodicity.

Warning c_ChirpTimerResol should configure at boot once per power cycle. RFS subsystem configures the hardware based on the setting so it should not be changed dynamically. Also the profile configuration should be updated accordingly.

- `c_FecBootCfg`

The FEC RFS M3 boot mode configuration.

Bit definition:

Bit Field	Definition
bits [0]	FEC RFS M3 boot self-test disable flag. This field is applicable only for safety enabled devices. Value 0 : Self test enable (default) Value 1 : Self test disable
bits [7:1]	Reserved

2.3.2 FECSS Device Power OFF API

This API function contains the device power off mode setting. The application should use this API to perform power down of the FECSS before entering deep sleep mode or software reset.

API Definition

API Index	0x0004U
API Function Name	rl_fecssDevPwrOff
API Configuration Structure	T_RL_API_FECSS_DEV_PWR_OFF_CMD
API Response Structure	None

- API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_DEV_PWR_OFF          ((UINT16)0x0004U)
// API Function Definition
    T_RETURNTYPE rl_fecssDevPwrOff(UINT8 c_devIndex, const T_RL_API_FECSS_DEV_PWR_OFF_CMD
    ↪  *p_apiCmdData);
```

- Configuration C Structure

Table 2.10: T_RL_API_FECSS_DEV_PWR_OFF_CMD

Field	Data Type	Size	Offset
c_RetentionMode	UINT8	1	0
c_Reserved1	UINT8	1	1
h_Reserved2	UINT16	2	2
w_Reserved3	UINT32	4	4
Total		**8**	

Configuration Fields

- c_RetentionMode

FECSS memory retention mode, the mode can be either RETENTION OFF or RETENTION ON type.

Warning The warm boot can be done only if power down is done with c_RetentionMode = 0x0A to retain the FECSS memory.

Value	Definition	mmWaveLink Macro
0x00	**RETENTION OFF**. If c_RetentionMode is 0x00, then FECSS RAM is not retained across deep sleep cycle. The successive FEC power up c_PowerMode supposed to be cold boot type. Typical use case: Hibernate.	M_RL_FECSS_PWR_DOWN_RET_OFF
0x0A	**RETENTION ON**. If c_RetentionMode is 0x0A, then FECSS RAM is retained across deep sleep cycle. The successive FEC power up c_PowerMode supposed to be warm boot type. Typical use case: Deep sleep Entry, software reset.	M_RL_FECSS_PWR_DOWN_RET_ON

2.3.3 DFP Version Get API

This API function returns a data structure containing the version information of mmWaveLink Library, FECSS Library and RFS ROM and Patch firmware.

API Definition

API Index	0x0005U
API Function Name	rl_mmWaveDfpVerGet
API Configuration Structure	None
API Response Structure	T_RL_API_DFP_FW_VER_GET_RSP

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_DFP_VER_GET                 ((UINT16)0x0005U)
// API Function Definition
T_RETURNTYPE rl_mmWaveDfpVerGet(UINT8 c_devIndex, T_RL_API_DFP_FW_VER_GET_RSP *p_apiResData);
```

Response C Structure

Table 2.11: T_RL_API_DFP_FW_VER_GET_RSP

Field	Data Type	Size	Offset
z_MmwlLibVersion	T_DFP_COMP_FW_VER	8	0
z_FecssLibVersion	T_DFP_COMP_FW_VER	8	8
z_RfsRomVersion	T_DFP_COMP_FW_VER	8	16
z_RfsPatchVersion	T_DFP_COMP_FW_VER	8	24
Total		**32**	

Table 2.12: T_DFP_COMP_FW_VER

Field	Data Type	Size	Offset
c_GenVerNum	UINT8	1	0
c_MajorVerNum	UINT8	1	1
c_MinorVerNum	UINT8	1	2
c_BuildVerNum	UINT8	1	3
c_Year	UINT8	1	4
c_Month	UINT8	1	5
c_Date	UINT8	1	6
c_Reserved	UINT8	1	7
Total		**8**	

Response Fields

- `z_MmwlLibVersion`

mmWaveLink Lib version, 8 bytes.

- `z_FecssLibVersion`

FECSSLib version, 8 bytes.

- `z_RfsRomVersion`

RFS ROM version, 8 bytes.

- `z_RfsPatchVersion`

RFS Patch version, 8 bytes.

2.3.4 FECSS RF Power On Off API

This API configures the device RF channels and miscellaneous control settings. Application should issue this API before any other functional API which configures the RF settings.

Warning All the enabled configurations must be disabled by issuing this API with zero input before FECSS power off. This ensures the correct firmware states after wake up and saves power.

API Definition

API Index	0x0006U
API Function Name	rl_fecssRfPwrOnOff
API Configuration Structure	T_RL_API_FECSS_RF_PWR_CFG_CMD
API Response Structure	None

- API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_RF_PWR_ONOFF          ((UINT16)0x0006U)
// API Function Definition
T_RETURNTYPE rl_fecssRfPwrOnOff(UINT8 c_devIndex, const T_RL_API_FECSS_RF_PWR_CFG_CMD
↪  *p_apiCmdData);
```

- Configuration C Structure

Table 2.13: T_RL_API_FECSS_RF_PWR_CFG_CMD

Field	Data Type	Size	Offset
h_RxChCtrlBitMask	UINT16	2	0
h_TxChCtrlBitMask	UINT16	2	2
c_Reserved	UINT8	1	4
c_MiscCtrl	UINT8	1	5
h_Reserved1	UINT16	2	6
w_Reserved2	UINT32	4	8
Total		12	

Configuration Fields

- h_RxChCtrlBitMask

FECSS RFS RX channels ON/OFF control, 1 bit per channel.

Value	Definition
0x0	OFF, RX channel will not be enabled in any functional frame, calibration and monitor trigger.
0x1	ON, RX channel will be enabled automatically.

Recommendation The RFS enables the RX channels before executing below functionalities and disables it in the end.

- FECSS Factory Calibration API
- FECSS RF Runtime Calibration API
- FECSS Monitor Enable Trigger API
- Sensor Start API

In idle state, the Tx, RX channels are by default in power down state. In TX Calibrations, RX Calibrations and Monitors at least one RX channel should be enabled.

Bit Field	Definition	mmWaveLink Macro
bits [0]	Rx Channel 0	M_RL_RX_CHANNEL_0
bits [1]	Rx Channel 1	M_RL_RX_CHANNEL_1
bits [2]	Rx Channel 2	M_RL_RX_CHANNEL_2
bits [15:3]	Reserved	

- h_TxChCtrlBitMask

FECSS RFS TX channels ON/OFF control, 1 bit per channel.

Recommendation The RFS enables the TX channels before executing below functionalities and disables it in the end.

- FECSS Factory Calibration API
- FECSS RF Runtime Calibration API
- FECSS Monitor Enable Trigger API
- Sensor Start API

In idle state, the TX channels are by default in power down state. In TX Calibrations and Monitors, the corresponding TX channel should be enabled. The RFS FW enables corresponding TX channel if this mask value is zero.

Value	Definition	mmWaveLink Macro
0x0	OFF, TX channel will not be enabled in any functional frame, calibration and monitor trigger.	M_RL_TX_CHANNEL_0
0x1	ON, TX channel will be enabled automatically.	M_RL_TX_CHANNEL_1

Bit field definition:

Bit Field	Definition
bits [0]	Tx Channel 0 Power Control
bits [1]	Tx Channel 1 Power Control
bits [15:2]	Reserved

▶ c_MiscCtrl

FECSS RFS FECSS Miscellaneous block / clock control, 1 bit per control.

Value	Definition
0x0	Block Disable
0x1	Block Enable height

Bit field definition:

Bit Field	Definition	mmWaveLink Macro
Bit [0]	RDIF clock enable control. The clock to the RDIF IP block should be enabled using this control before configuring and enabling the RDIF. Note: This clock is derived from APLL clock. APLL must be ON while enabling this clock.	M_RL_RF_MISC_CTRL_RDIF_CLK
Bit [1]	RF ANA 1V LDO bypass control. By default the internal LDO is used to drive 1V supply. To bypass the internal LDO, set this bit. Note: External 1V supply should be connected in case of bypass. Note: Reserved for internal test only.	M_RL_RF_MISC_CTRL_RF_1V_LDO_BYPASS
Bit [7:2]	Reserved	

2.3.5 FECSS Factory Calibration API

This API function configures and triggers the FECSS RF factory calibrations. The application should use this API to perform one time RF configuration dependent calibrations (factory calibration) and store the results in non-volatile memory with the help of FECSS Factory Calibration Data Get API. The results can be restored in field using FECSS Factory Calibration Data Set API.

Factory calibration API supports following calibrations:

i. VCO Delta Trim Calibration

Update Synthesizer VCO C-trim to compensate for temperature and aging

ii. PD Offset Calibration

Compensate temperature dependent PD offsets

iii. LoDIST LUT Calibration

Update LoDist and other PVT LUT based settings

iv. Rx IFA Calibration

Calibrate HPF and LPF in the Rx IFA.

v. Rx Gain Calibration

Derive Rx gain codes for desired gain for 3 temperature bins

vi. Tx Power Calibration

Derive Tx PA codes for desired backoff(s) for 3 temperature bins

Assumptions

- APLL must be enabled before using FECSS Device Clock Ctrl API
- Tx and Rx channel configuration is set using FECSS RF Power On Off API
- Live monitor faults and loopbacks are disabled

Recommendation Factory calibrations are recommended to run within junction temperature range of 10^oC to 50^oC.

API Definition

API Index	0x0007U
API Function Name	rl_fecssRfFactoryCal
API Configuration Structure	T_RL_API_FECSS_RF_FACT_CAL_CMD
API Response Structure	T_RL_API_FECSS_RF_FACT_CAL_RSP

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_RF_FACTORY_CAL        ((UINT16)0x0007U)
// API Function Definition
    T_RETURNTYPE rl_rlfecssRfFactoryCal(UINT8 c_devIndex, const
    ↪  T_RL_API_FECSS_RF_FACT_CAL_CMD *p_apiCmdData, T_RL_API_FECSS_RF_FACT_CAL_RSP*
    ↪  p_apiResData);
```

- Configuration C Structure

Table 2.14: T_RL_API_FECSS_RF_FACT_CAL_CMD

Field	Data Type	Size	Offset
h_CalCtrlBitMask	UINT16	2	0
c_MiscCalCtrl	UINT8	1	2
c_CalRxGainSel	UINT8	1	3
c_CalTxBackOffSel	UINT8[4]	4	4
w_Reserved0	UINT32	4	8
h_CalRfFreq	UINT16	2	12
xh_CalRfSlope	SINT16	2	14
c_TxPwrCalTxEnaMask	UINT8[4]	4	16
w_Reserved1	UINT32	4	20
xc_CalTempBinOverrides	SINT8[3]	3	24
c_Reserved1	UINT8	1	27
w_Reserved2	UINT32[2]	8	28
Total		**36**	

- Response C Structure

Table 2.15: T_RL_API_FECSS_RF_FACT_CAL_RSP

Field	Data Type	Size	Offset
h_CalRunStatus	UINT16	2	0
h_CalResStatus	UINT16	2	2
xc_CalibTemp	SINT8	1	4
c_Reserved1	UINT8	1	5
h_Reserved2	UINT16	2	6
w_Reserved4	UINT32	4	8
Total		**12**	

Configuration Fields

- h_CalCtrlBitMask

FECSS RFS factory calibration enable control mask with 1 bit per calibration. Setting the corresponding bit will enable the calibration.

Recommendation Tx and Rx calibrations are recommended to run along with the VCO, PD and LoDIST calibrations to get better calibration accuracy.

Bit-definitions are as follows:

Bit(s)	Definition	mmWaveLink Macro
Bit[0]	Reserved	
Bit[1]	Enable synthesizer VCO calibration	`M_RL_FECSS_DEV_CAL_VCO`
Bit[2]	Enable PD calibration	`M_RL_FECSS_DEV_CAL_PD`
Bit[3]	Enable LoDIST LUT calibration	`M_RL_FECSS_DEV_CAL_LODIST`
Bit[4]	Reserved	
Bit[5]	Enable Rx IFA calibration Note: Required only for pre-production samples and requires ATE calibration firmware	`M_RL_FECSS_DEV_CAL_RX_IFA`
Bit[6]	Enable Rx gain calibration	`M_RL_FECSS_DEV_CAL_RX_GAIN`
Bit[7]	Enable Tx power calibration	`M_RL_FECSS_DEV_CAL_TX_PWR`

- `c_MiscCalCtrl`

Miscellaneous calibration control bits:

Bit(s)	Definition	mmWaveLink Macro
Bit[0]	Disable RF measurements in the Rx gain calibration and re-use data from previous calibration. DFP stores the measured RF values when Rx gain calibration is executed with this bit disabled. Firmware re-uses the stored data to compute `c_RxGainCodes` when the feature is enabled	`M_RL_FECSS_CAL_RF_GAIN_CAL_DIS`
Bit[1]	Disable Tx closed loop calibration (CLPC) while calibrating Txs	`M_RL_FECSS_CAL_TX_CLPC_CAL_DIS`
Bit[2]	Override bin temperatures for Tx calibration	`M_RL_FECSS_CAL_TX_CAL_TEMP_OVERRIDE`
Bits[7-3]	Reserved	

Recommendation Use of override bin temperatures is not recommended for xWRLx432 devices

- `c_CalRxGainSel`

Common Rx channel gain setting for all Rx channels in dB. $1LSB = 1dB$
Factory calibration API computes `c_CalRxGainCodes` for the three temperature bins bins to achieve similar RF performance across temperature range.

Valid Range: [30, 40]

Bit-definitions are as follows:

Bit(s)	Definition	mmWaveLink Macro
Bit[5-0]	RX gain selection for each RX channel	`M_RL_FECSS_CAL_RX_GAIN_MASK`
Bit[7-6]	Reserved	

- `c_CalTxBackOffSel`

Chirp Profile TX channels power back-off setting for calibration in dB. $1LSB = 0.5dB$

Byte definition is as follows:

Byte Field	Definition
Byte[0]	Output power back-off value for Tx0
Byte[1]	Output power back-off value for TX1
Byte[3-2]	Reserved

Note Accuracy of the Tx factory calibration may degrade with higher back-off.

Device	Backoff-Range
xWRL6432	[0dB, 26dB]
xWRL1432	[0dB, 20dB]

Warning Please check DFP release notes for release specific constraints

- `h_CalRfFreq`

Calibration chirp RF start Frequency.

Recommendation Run calibrations around the mid-band frequency with a very small slope

RF Frequency	Resolution
60GHz	$1LSB = (3 \times APLL\_FREQ)/2^{10} \approx 1.172MHz$ Valid Range: 57 GHz to 64 GHz for APLL_FREQ = 400MHz: 0xBE00U (57GHz) to 0xD555U (64 GHz)* *Variant specific limitations in Release Notes
77GHz	$1LSB = (4 \times APLL\_FREQ)/2^{10} \approx 1.562MHz$ Valid Range: 76 GHz to 81 GHz for APLL_FREQ = 400MHz: 0x0000BE00 (76GHz) to 0x0000CA80 (81GHz)

- `xh_CalRfSlope`

RFS calibration chirp RF Frequency Slope.

Recommendation The typical ramp time used in FW for calibration chirps are 55us.
For 60GHz: (APLL_FREQ = 400MHz): 2.2MHz/us (0x4D).
For 77GHz: (APLL_FREQ = 400MHz): 2.2MHz/us (0x3A).

RF Slope	Resolution
60GHz	$1LSB = (3 \times 400 \times APLL\_FREQ)/2^{24} \approx 28.610kHz/us$,signed number Valid Range: -3495 to 3495 (+/- 100MHz/us (APLL_FREQ = 400MHz))
77GHz	$1LSB = (4 \times 400 \times APLL\_FREQ)/2^{24} \approx 38.147kHz/us$, signed number Valid Range: -3495 to 3495 (+/- 100MHz/us (APLL_FREQ = 400MHz))

- `c_TxPwrCalTxEnaMask`

Per channel Tx enable mask for calibrations.

Recommendation

- In case more than 1 TXs are enabled during the chirp, then enabling the same TXs during calibration will result in better TX output power accuracy.
- Recommended Calibration Masks: 0x3, 0x1. This setting will enable both Tx channels while calibrating Tx 0 and copy the results for Tx1 without separately calibrating it.

Calibrated bias code for all Tx channels.

Byte Field	Definition
Byte[0]	Tx enable mask for calibrating Tx0. If Tx0 is disabled in this mask, calibration data from the lowest enabled Tx channel from this mask will be applied to Tx0. Tx0 calibration will be skipped from the calibration if the mask is 0.
Byte[1]	Tx enable mask for calibrating Tx1. If Tx1 is disabled in this mask, calibration data from the lowest enabled Tx channel from this mask will be applied to Tx1. Tx1 calibration will be skipped from the calibration if the mask is 0.
Byte[3-2]	Reserved

Examples Following list describes the Tx enable masks for a 2 channel device.

- `c_TxPwrCalTxEnaMask` = 3, 1
  Tx0 is calibrated with backoff from `c_CalTxBackOffSel`[0] with both Tx0 and Tx1 enabled during calibration
  Results from Tx0 calibration are copied to Tx1
  This mode is recommended to keep Tx mismatches to minimum
- `c_TxPwrCalTxEnaMask` = 2, 3
  Results from Tx1 calibration are copied to Tx0
  Tx1 is calibrated with backoff from `c_CalTxBackOffSel`[1] with both Tx0 and Tx1 enabled during calibration
- `c_TxPwrCalTxEnaMask` = 1, 2
  Tx0 is calibrated with backoff from `c_CalTxBackOffSel`[0] with Tx1 disabled
  Tx1 is calibrated with backoff from `c_CalTxBackOffSel`[1] with Tx0 disabled
  This mode is not recommended because of higher Tx mismatch
- `c_TxPwrCalTxEnaMask` = 3, 3
  Tx0 is calibrated with backoff from `c_CalTxBackOffSel`[0] with both Tx0 and Tx1 enabled during calibration
  Tx1 is calibrated with backoff from `c_CalTxBackOffSel`[1] with both Tx0 and Tx1 enabled during calibration
  Since both Tx channels are calibrated separately, Tx mismatch will be higher
- `c_TxPwrCalTxEnaMask` = 1, 0
  Tx0 is calibrated with backoff from `c_CalTxBackOffSel`[0] with Tx1 disabled
  Tx1 will not be calibrated (results will not be updated either)
  This mode is recommended if only Tx0 is being used

▶ `xc_CalTempBinOverrides`

TX temperature override values.

This field is valid if `M_RL_FECSS_CAL_TX_CAL_TEMP_OVERRIDE` bit is set in `c_MiscCalCtrl`.

Response Fields

▶ `h_CalRunStatus`

Status for the current calibration trigger.
This mask indicates pass/ fail status for all the calibrations in the current trigger.

Value	Definition
0x0	FAIL
0x1	PASS

Bit field definition:

Bit Field	Definition
bits [0]	RESERVED
bits [1]	VCO calibration status
bits [2]	PD calibration status
bits [3]	LODIST calibration status
bits [4]	RESERVED
bits [5]	RX IFA calibration status
bits [6]	RX Gain calibration status
bits [7]	TX power calibration status
bits [15:8]	Reserved

- `h_CalResStatus`

Calibration data validity status.
This mask indicates the validity of calibration data in `T_RL_API_FECSS_FACT_CAL_DATA` . This mask could remain set even after the corresponding calibration fails in a current run because mmWaveLink calibration APIs retain the previous calibration data in case of failure.

Value	Definition
0x0	INVALID - Calibration data is invalid.
0x1	VALID - Calibration data is valid.

Bit field definition:

Bit Field	Definition
bits [0]	APLL calibration validity status
bits [1]	VCO calibration validity status
bits [2]	PD calibration validity status
bits [3]	LODIST calibration validity status
bits [4]	RESERVED
bits [5]	RX IFA calibration validity status
bits [6]	RX Gain calibration validity status
bits [7]	TX power calibration validity status
bits [15:8]	Reserved

- `xc_CalibTemp`

The device temperature at which calibration is executed.
1LSB = $2°C$
Range : -127 to 127 ($-254°C$ to $254°C$)

2.3.6 FECSS RF Runtime Calibration API

The FECSS RF runtime calibration API configuration and trigger API. Runtime calibrations primarily compensates for the runtime variations like temperature. The calibration data is updated in the IPC memory directly and can be stored/ restored using the FECSS Factory Calibration Data Get API and FECSS Factory Calibration Data Set API.

Assumptions

- APLL must be enabled before using FECSS Device Clock Ctrl API
- Tx and Rx channel configuration is set using FECSS RF Power On Off API
- Live monitor faults and loopbacks are disabled

API Definition

API Index	`0x0008U`
API Function Name	`rl_fecssRfRuntimeCal`
API Configuration Structure	`T_RL_API_FECSS_RF_RUN_CAL_CMD`
API Response Structure	`T_RL_API_FECSS_RF_RUN_CAL_RSP`

- API Function

mmWaveLink C function declaration

```
// API Index Macro
```

```
#define M_RL_API_ID_FECSS_RF_RUN_CAL              ((UINT16)0x0008U)
// API Function Definition
T_RETURNTYPE rl_fecssRfRuntimeCal(UINT8 c_devIndex, const T_RL_API_FECSS_RF_RUN_CAL_CMD
↪  *p_apiCmdData, T_RL_API_FECSS_RF_RUN_CAL_RSP* p_apiResData);
```

- Configuration C Structure

Table 2.16: T_RL_API_FECSS_RF_RUN_CAL_CMD

Field	Data Type	Size	Offset
h_CalCtrlBitMask	UINT16	2	0
h_Reserved0	UINT16	2	2
c_TempBinIndex	UINT8	1	4
c_Reserved1	UINT8	1	5
h_Reserved2	UINT16	2	6
w_Reserved3	UINT32	4	8
w_Reserved4	UINT32	4	12
w_Reserved5	UINT32	4	16
Total		**20**	

- Response C Structure

Table 2.17: T_RL_API_FECSS_RF_RUN_CAL_RSP

Field	Data Type	Size	Offset
h_CalRunStatus	UINT16	2	0
h_CalResStatus	UINT16	2	2
w_Reserved1	UINT32	4	4
w_Reserved4	UINT32	4	8
Total		**12**	

Configuration Fields

- h_CalCtrlBitMask

FECSS RFS runtime calibration control, 1 bit per channel.

Bit-definitions for the fields are as follows:

Bit Field(s)	Definition
bits[0]	Reserved
bits[1]	VCO calibration ON/OFF control.
bits[2]	PD calibration ON/OFF control. Enable this calibration only if any PD measurement is done in runtime (TX power and ball break monitors, Tx CLPC calibration).
bits[3]	LODIST calibration ON/OFF control. The LODIST_BIAS_CODES are updated to HW based on `c_TempBinIndex`.
bits[4]	Reserved
bits[5]	Reserved
bits[6]	RX Gain calibration ON/OFF control. The `c_CalRxGainCodes` are applied to HW based on `c_TempBinIndex`.
bits[7]	TX power OLPC calibration ON/OFF control. The `h_CalTxBiasCodes` are applied to HW based on `c_TempBinIndex`.
bits[15:8]	Reserved

- `c_TempBinIndex`

FECSS RF calibration temperature bin index.
The application should read the device temperature using FECSS Temperature Measurement Trigger API and provide the bin index during the calibration. Bit-definitions for the field are as follows:

Value(s)	Definition
0x0	LOW , The Device temperature is < 0^oC (Default Target Temperature: -20^oC)
0x8	MID, The Device temperature is >= 0^oC and < 85^oC (Default Target Temperature: 42^oC)
0x10	HIGH, The Device temperature is >= 85^oC (Default Target Temperature: 105^oC)

Response Fields

- `h_CalRunStatus`

FECSS RFS runtime calibration run status, 1 bit per calibration.

This is a status of the current calibration command trigger.

Value(s)	Definition
0x0	FAIL - Calibration trigger is disabled / Enabled calibration has failed.
0x1	PASS - Enabled calibration has Passed.

Bit-definitions are as follows:

Bit Field(s)	Definition
bits[0]	Reserved
bits[1]	VCO calibration status.
bits[2]	PD calibration status.
bits[3]	LODIST calibration status.
bits[4]	Reserved
bits[5]	Reserved
bits[6]	RX Gain calibration status.
bits[7]	TX power calibration status.
bits[15:8]	Reserved

- `h_CalResStatus`

FECSS RFS run-time calibration result validity status, 1 bit per calibration. This is the latest validity status of all the calibrations. The previous validity status is retained if latest calibration trigger fails.

Value(s)	Definition
0x0	INVALID - calibration data is invalid.
0x1	VALID - calibration data is valid.

Bit-definitions are as follows:

Bit Field(s)	Definition
bits[0]	APLL calibration validity status.Updates status of APLL calibration.
bits[1]	VCO calibration validity status.
bits[2]	PD calibration validity status.
bits[3]	LODIST calibration validity status.
bits[4]	Reserved
bits[5]	RX IFA calibration validity status.Updates status of RX IFA calibration.
bits[6]	RX Gain calibration validity status.
bits[7]	TX power calibration validity status.
bits[15:8]	Reserved

2.3.7 FECSS RF Clock Bandwidth Configuration API

This API controls bandwidth for APLL and Synth clock modules.

warning The firmware does not store the bandwidth control settings and applies the default values after every boot. If required, application should re configure the bandwidth control settings after FECSS Device Power On API and before FECSS Device Clock Ctrl API API.

Default values for the bandwidth controls are as follows:

<table>
<tr><th>Parameter</th><th>xWRL6432</th><th>xWRL1432</th></tr>
<tr><td>c_SynthIcpTrim</td><td>0x5</td><td>0x1</td></tr>
<tr><td>c_SynthRzTrim</td><td>0x4</td><td>0x4</td></tr>
<tr><td>c_ApllIcpTrim</td><td><table><tr><th>XTAL_FREQ</th><th>Value</th></tr><tr><td>25 MHz</td><td>0x12</td></tr><tr><td>26 MHz</td><td>0x12</td></tr><tr><td>38.4 MHz</td><td>0x1C</td></tr><tr><td>40 MHz</td><td>0x1C</td></tr></table></td><td><table><tr><th>XTAL_FREQ</th><th>Value</th></tr><tr><td>25 MHz</td><td>0x12</td></tr><tr><td>26 MHz</td><td>0x12</td></tr><tr><td>38.4 MHz</td><td>0x1C</td></tr><tr><td>40 MHz</td><td>0x1C</td></tr></table></td></tr>
<tr><td>c_ApllRzTrim</td><td>0x11</td><td>0x11</td></tr>
<tr><td>c_ApllVcoRtrim</td><td>0x0C</td><td>0x12</td></tr>
<tr><td>Synth Bandwidth</td><td>1.2MHz</td><td>900Hz</td></tr>
<tr><td>APLL Bandwidth</td><td>250kHz</td><td>250kHz</td></tr>
</table>

API Definition

API Index	0x0009U
API Function Name	rl_fecssRfClockBwCfg
API Configuration Structure	T_RL_API_FECSS_CLK_BW_CFG_CMD
API Response Structure	None

▸ API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_RF_CLKBW_CFG          ((UINT16)0x0009U)
// API Function Definition
    T_RETURNTYPE rl_fecssRfClockBwCfg(UINT8 c_devIndex, const T_RL_API_FECSS_CLK_BW_CFG_CMD
    ↪  *p_apiCmdData);
```

- Configuration C Structure

Table 2.18: T_RL_API_FECSS_CLK_BW_CFG_CMD

Field	Data Type	Size	Offset
c_SynthIcpTrim	UINT8	1	0
c_SynthRzTrim	UINT8	1	1
c_ApllIcpTrim	UINT8	1	2
c_ApllRzTrim	UINT8	1	3
c_ApllVcoRtrim	UINT8	1	4
c_Reserved1	UINT8	1	5
c_Reserved2	UINT8	1	6
c_Reserved3	UINT8	1	7
w_Reserved4	UINT32	4	8
w_Reserved5	UINT32	4	12
Total		**16**	

Configuration Fields

- c_SynthIcpTrim

FECSS device RF synth clock ICP trim value.

- c_SynthRzTrim

FECSS device RF synth clock RZ trim value.

- c_ApllIcpTrim

FECSS device RF APLL clock ICP trim value.

- c_ApllRzTrim

FECSS device RF APLL clock RZ LPF trim value.

- c_ApllVcoRtrim

FECSS device RF APLL clock VCO RTrim value.

2.3.8 FECSS GPADC Measurement Configuration API

This API configures GPADC channels to measure the DC signals. GPADC signals are configured using a combination of Param Value and Config Value. The API supports multiple channels which can be enabled/ disabled individually. The FECSS GPADC Measurement Trigger API uses these settings to measure the desired signals.

Note Configurations are stored in the FECSS IPC memory and retained if memory retention is enabled in FECSS Device Power OFF API.

GPADC Signals Config and Param Values:

GPADC signal	Param Value	Config Value
External GPADC signal 1 buffered	CHAN_CTRL = 124, NUM_MEAS = 4, SKIP_SAMPLES = 10, SIG_TYPE = 1, OFFSET = 0, ENABLE = 1	0x00200000
External GPADC signal 1 non-buffered	CHAN_CTRL = 124, NUM_MEAS = 4, SKIP_SAMPLES = 10, SIG_TYPE = 2, OFFSET = 0, ENABLE = 1	0x00008000
External GPADC signal 2 buffered	CHAN_CTRL = 124, NUM_MEAS = 4, SKIP_SAMPLES = 10, SIG_TYPE = 1, OFFSET = 0, ENABLE = 1	0x00010000
External GPADC signal 2 non-buffered	CHAN_CTRL = 124, NUM_MEAS = 4, SKIP_SAMPLES = 10, SIG_TYPE = 2, OFFSET = 0, ENABLE = 1	0x00004000

API Definition

API Index	0x000AU
API Function Name	rl_fecssGpadcMeasCfg
API Configuration Structure	T_RL_API_FECSS_GPADC_MEAS_CMD
API Response Structure	None

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_GPADC_MEAS_CFG        ((UINT16)0x000AU)
// API Function Definition
    T_RETURNTYPE rl_fecssGpadcMeasCfg(UINT8 c_devIndex, const T_RL_API_FECSS_GPADC_MEAS_CMD
    ↪  *p_apiCmdData);
```

Configuration C Structure

Table 2.19: T_RL_API_FECSS_GPADC_MEAS_CMD

Field	Data Type	Size	Offset
w_ParamVal	UINT32[8]	32	0
w_ConfigVal	UINT32[8]	32	32
w_Reserved1	UINT32	4	64
w_Reserved2	UINT32	4	68
Total		**72**	

Configuration Fields

w_ParamVal

FECSS GPADC channel param value, 32bit value per channel. The param value is a unique value includes, channel control , num of collect and skip samples for internal or external signals to measure the corresponding signal in any GPADC channel.

Bit field definition:

Field Name	Bits	Definition
CHANNEL_CTRL	bits [7:0]	GPADC channel control value, 8bit value per channel. The control value is a unique parameter value for a internal or external signal to link it with the GPADC channel.

COLLECT_SAMPLES	bits [15:8]	This setting can be used to measure average value of the signal across multiple samples. • 1LSB = 1 collect sample Overall measurement time is (COLLECT_SAMPLES * GPADC Collect Sample Period) • Range: 0 to 255
SKIP_SAMPLES	Bit[22:16]	Clock cycles skipped before the measurement • Total skip samples is computed as $Mantissa \times 2^{Exponent}$ • where Mantissa = SKIP_SAMPLES[3:0] and Exponent = SKIP_SAMPLES[6:4] • Recommendation: 10 Bit-Field definition: • Bits[3:0]: 1LSB = 1 skip sample (Mantissa) • Valid Values: 0 • skip samples: 0 to 15 • Bits[6:4] : $1LSB = 2^N skipsamples(Exponent)$ • Range: 0 to 7 • skip samples: 1 to 128 • 1 skip sample period = Time period of operating clock frequency
Reserved	bits [23]	Reserved
SIGNAL_TYPE	bits [26:24]	GPADC signal type, 3 bits value. The GPADC measurement signal type should be appropriately selected to configure the analog and to get accurate measurement.
BUFFER_OFFSET	Bit[29:27]	GPADC buffer offset type index, 3 bits value. The GPADC buffer offset index type should be appropriately selected to add device process specific offset to the measurement.
Reserved	bits [30]	Reserved
CHANNEL_EN	bits [31]	GPADC channel ON/OFF control, 1 bit control.

The GPADC measurement channel enable control:

Value	Definition
0x0	OFF
0x1	ON

The GPADC collect sample period definition:

XTAL clock Source	GPADC Operating Clock	1LSB collect sample Period
25MHz	12.5MHz	1.28us
26MHz	13MHz	1.231us
38.4MHz	12.8MHz	1.25us
40MHz	13.3MHz	1.203us

The GPADC skip sample period definition:

XTAL clock Source	GPADC Operating Clock	1LSB skip sample Period
25MHz	12.5MHz	80ns
26MHz	13MHz	76.92ns
38.4MHz	12.8MHz	78.125ns
40MHz	13.3MHz	75.19ns

The GPADC Signal types:

Value	Definition
0	Internal Buffered Signal
1	External Buffered Signal
2	External Non-Buffered Signal
3	Temperature Sensor Differential Signal
4	Temperature Sensor Single Ended Signal
7-5	Reserved

The GPADC buffer offset index types:

Value	Definition
0	NO Offset (Default)
1	RX Offset
2	TX Offset
3	CLK Offset
4	PM Offset
7-5	Reserved

- `w_ConfigVal`

FECSS GPADC channel config value, 32bit value per channel. Max 8 GPADC signals supported. The config value is a unique value for a internal or external signal to configure and measure the corresponding signal in any

GPADC channel.

2.3.9 FECSS GPADC Measurement Trigger API

This API triggers GPADC measurements for the signals configured using the FECSS GPADC Measurement Configuration API.

Note FECSS must be in power-on state to trigger GPADC measurement

API Definition

API Index	0x000BU
API Function Name	rl_fecssGpadcMeasTrig
API Configuration Structure	None
API Response Structure	T_RL_API_FECSS_GPADC_MEAS_RSP

▶ API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_GPADC_MEAS_TRIG       ((UINT16)0x000BU)
// API Function Definition
    T_RETURNTYPE rl_fecssGpadcMeasTrig(UINT8 c_devIndex, T_RL_API_FECSS_GPADC_MEAS_RSP
    ↪  *p_apiResData);
```

▸ Response C Structure

Table 2.21: T_RL_API_FECSS_GPADC_MEAS_RSP

Field	Data Type	Size	Offset
c_GpadcChRunStatus	UINT8	1	0
c_Reserved1	UINT8	1	1
h_Reserved2	UINT16	2	2
c_GpadcMeasVal	UINT8[8]	8	4
w_Reserved3	UINT32	4	12
w_Reserved4	UINT32	4	16
Total		**20**	

Response Fields

- c_GpadcChRunStatus

Note Max 8 GPADC signal channels supported.

Value	Definition
0x0	FAIL (Enabled GPADC channel has failed.)
0x1	PASS (Enabled GPADC channel has Passed.)

Bit field definition:

Bit Field	Definition
bits [0]	GPADC channel 0 status
bits [1]	GPADC channel 1 status
bits [2]	GPADC channel 2 status
bits [3]	GPADC channel 3 status
bits [4]	GPADC channel 4 status
bits [5]	GPADC channel 5 status
bits [6]	GPADC channel 6 status
bits [7]	GPADC channel 7 status

- `c_GpadcMeasVal`

FECSS GPADC measurement average value: 1 LSB = 1.8V / 256.

2.3.10 FECSS Temperature Measurement Config API

This API configures on-chip temperature sensor measurements. The FECSS Temperature Measurement Trigger API uses these settings to measure the desired temperature sensors.

Note Configurations are stored in the FECSS IPC memory and retained if memory retention is enabled in FECSS Device Power OFF API.

API Definition

API Index	`0x000CU`
API Function Name	`rl_fecssTempMeasCfg`
API Configuration Structure	`T_RL_API_FECSS_TEMP_MEAS_CMD`
API Response Structure	None

- API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_TEMP_MEAS_CFG          ((UINT16)0x000CU)
// API Function Definition
```

```
T_RETURNTYPE rl_fecssTempMeasCfg(UINT8 c_devIndex, const T_RL_API_FECSS_TEMP_MEAS_CMD
↪  *p_apiCmdData);
```

Configuration C Structure

Table 2.22: T_RL_API_FECSS_TEMP_MEAS_CMD

Field	Data Type	Size	Offset
h_TempCtrlBitMask	UINT16	2	0
h_Reserved1	UINT16	2	2
w_Reserved2	UINT32	4	4
Total		8	

Configuration Fields

- h_TempCtrlBitMask

FECSS RFS temperature measurement control, 1 bit per channel.

Notes Max 4 temperature sensors supported in this device.

Value	Definition
0x0	OFF
0x1	ON

Bit field definition:

Bit Field	Definition
bits [0]	RX temperature sensor ON/OFF control
bits [1]	Reserved
bits [2]	Reserved
bits [3]	Reserved
bits [4]	TX temperature sensor ON/OFF control
bits [5]	Reserved
bits [6]	Reserved
bits [7]	Reserved
bits [8]	PM temperature sensor ON/OFF control
bits [9]	DIG temperature sensor ON/OFF control
bits [15:10]	Reserved

2.3.11 FECSS Temperature Measurement Trigger API

This API triggers temperature sensor measurements configured using the FECSS Temperature Measurement Config API.

API Definition

API Index	0x000DU
API Function Name	rl_fecssTempMeasTrig
API Configuration Structure	None
API Response Structure	T_RL_API_FECSS_TEMP_MEAS_RSP

▶ API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_TEMP_MEAS_TRIG        ((UINT16)0x000DU)
// API Function Definition
T_RETURNTYPE rl_fecssTempMeasTrig(UINT8 c_devIndex, T_RL_API_FECSS_TEMP_MEAS_RSP
↪  *p_apiResData);
```

▸ Response C Structure

Table 2.23: T_RL_API_FECSS_TEMP_MEAS_RSP

Field	Data Type	Size	Offset
h_TempStsBitMask	UINT16	2	0
h_Reserved1	UINT16	2	2
xh_TempValue	SINT16[10]	20	4
w_Reserved2	UINT32	4	24
Total		**28**	

Response Fields

▶ h_TempStsBitMask

FECSS RFS temperature measurement status, 1 bit per channel.

Notes Max 4 temperature sensors supported in this device.

Value	Definition
0x0	FAIL (Enabled temperature sensor measurement has failed.)
0x1	PASS (Enabled temperature sensor measurement has passed.)

Bit field definition:

Bit Field	Definition
bits [0]	RX temperature sensor status
bits [1]	Reserved
bits [2]	Reserved
bits [3]	Reserved
bits [4]	TX temperature sensor status
bits [5]	Reserved
bits [6]	Reserved
bits [7]	Reserved
bits [8]	PM temperature sensor status
bits [9]	DIG temperature sensor status
bits [15:10]	Reserved

- `xh_TempValue`

FECSS RFS temperature measured value, 1LSB = 1^oC signed.

2.3.12 FECSS Factory Calibration Data Get API

This API is used to store complete calibration data to flash or external memory. This is an optional API which can be used in the following scenarios:

- Store the TI factory calibration data in external flash during early evaluation.
- Use caseswhich do not retain FECSS memory during sleep and performs always perform cold boot for FECSS. This API can be used to store the latest calibration data and avoid re-running of all the factory and runtime calibrations including APLL.

API Definition

API Index	0x000EU
API Function Name	`rl_fecssRfFactoryCalDataGet`
API Configuration Structure	None
API Response Structure	`T_RL_API_FECSS_FACT_CAL_DATA`

- API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_FACTORY_CAL_DATA_GET  ((UINT16)0x000EU)
// API Function Definition
T_RETURNTYPE rl_fecssRfFactoryCalDataGet(UINT8 c_devIndex, T_RL_API_FECSS_FACT_CAL_DATA
↪  *p_apiResData);
```

2.3.13 FECSS Factory Calibration Data Set API

This API is used to re-store complete calibration data from flash or external memory. This is an optional API which can be used in the following scenarios:

- Restore the TI factory calibration data from external flash during early evaluation.
- Use caseswhich do not retain FECSS memory during sleep and performs always perform cold boot for FECSS. This API can be used to restore the latest calibration data and avoid re-running of all the factory and runtime calibrations including APLL.

API Definition

API Index	0x000FU
API Function Name	rl_fecssRfFactoryCalDataSet
API Configuration Structure	T_RL_API_FECSS_FACT_CAL_DATA
API Response Structure	None

▶ API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_FACTORY_CAL_DATA_SET  ((UINT16)0x000FU)
// API Function Definition
T_RETURNTYPE rl_fecssRfFactoryCalDataSet(UINT8 c_devIndex, const T_RL_API_FECSS_FACT_CAL_DATA
↪  *p_apiCmdData);
```

- Configuration C Structure

Table 2.24: T_RL_API_FECSS_FACT_CAL_DATA

Field	Data Type	Size	Offset
h_CalResValidity	UINT16	2	0
h_CalResValRed	UINT16	2	2
w_ApllCapCodes	UINT32	4	4
h_SynthCapCodes	UINT16	2	8
h_IfaTrimCodes	UINT16	2	10
h_Reserved0	UINT16	2	12
h_PdFactCalTemp	UINT16	2	14
c_PdCalCodes	UINT8[76]	76	16
z_RxTxCalData	T_RL_API_FECSS_RXTX_CAL_DATA	36	92
Total		**128**	

Configuration Fields

- h_CalResValidity

FECSS RFS factory calibration result validity status, 1 bit per calibration.
Latest validity status of all the calibrations.

Value	Definition
0x0	INVALID - Calibration data is invalid.
0x1	VALID - Calibration data is valid.

Bit field definition:

Bit Field	Definition
bits [0]	APLL calibration validity status
bits [1]	VCO calibration validity status
bits [2]	PD calibration validity status
bits [3]	LODIST calibration validity status
bits [4]	RESERVED
bits [5]	RX IFA calibration validity status
bits [6]	RX Gain calibration validity status
bits [7]	TX power calibration validity status
bits [15:8]	Reserved

- h_CalResValRed

FECSS RFS factory calibration result validity status redundancy.

- w_ApllCapCodes

FECSS RFS APLL calibrated cap codes

- h_SynthCapCodes

FECSS RFS SYNTH calibrated cap codes.

- h_IfaTrimCodes

FECSS RFS IFA calibrated trim codes.

- h_PdFactCalTemp

FECSS PD β-DC Calibration Temperature

- c_PdCalCodes

FECSS RFS PD calibrated codes. Total 76 bytes

- z_RxTxCalData

FECSS RFS RX-TX calibration data. Total 36 bytes. Please refer T_RL_API_FECSS_RXTX_CAL_DATA for details.

2.3.14 FECSS Device Clock Ctrl API

This API function configures the device clock source selection setting. The application can use this API to switch the FECSS clock source based on device power state.

API Definition

API Index	0x0010U
API Function Name	rl_fecssDevClockCtrl
API Configuration Structure	T_RL_API_FECSS_DEV_CLK_CTRL_CMD
API Response Structure	None

- API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_DEV_CLK_CTRL          ((UINT16)0x0010U)
// API Function Definition
T_RETURNTYPE rl_fecssDevClockCtrl(UINT8 c_devIndex, const T_RL_API_FECSS_DEV_CLK_CTRL_CMD
↪  *p_apiCmdData);
```

Configuration C Structure

Table 2.25: T_RL_API_FECSS_DEV_CLK_CTRL_CMD

Field	Data Type	Size	Offset
c_DevClkCtrl	UINT8	1	0
c_FtClkCtrl	UINT8	1	1
c_ApllClkCtrl	UINT8	1	2
c_Reserved1	UINT8	1	3
w_Reserved2	UINT32	4	4
Total		**8**	

Configuration Fields

- c_DevClkCtrl

FECSS run time core clock source selection type.

The application can switch the clock source of FECSS based on power state to XTAL clock or Fast clock or can gate the FECSS system clock.

Recommendation FECSS clock state:
- XTAL Clock - Short idle time.
- Fast Clock - Data acquisition mode, calibration or monitoring.
- Clock Gate - Long frame idle time.

This API configures FECSS clock based on this setting.

Value	Definition	mmWaveLink Macro
0x00	XTAL clock source. Application can switch FECSS to the XTAL clock (XTAL_FREQ) in short idle period when FECSS is in idle state.	M_RL_FECSS_DEV_CLK_SRC_XTAL
0x0A	Fast clock source Fast clock is derived from (DIG_PLL_FREQ/2)	M_RL_FECSS_DEV_CLK_SRC_FCLK
0x05	Clock gate (OFF) The application can gate FECSS clock in inter-frame idle time.	M_RL_FECSS_DEV_CLK_SRC_OFF

- `c_FtClkCtrl`

FECSS frame timer (FT) clock gate option. The application can gate the FT clock source based on low power state.

Recommendation The frame timer always runs on XTAL clock.
- XTAL Clock - Normal operating mode
- Clock Gate - Device idle time (NOT frame idle time)

This API configures FECSS clock based on this setting.

Value	Definition	mmWaveLink Macro
0x00	XTAL clock source Default operating frequency. Supported clock Frequencies are: 25MHz, 26MHz, 38.4MHz or 40MHz.	`M_RL_FECSS_FT_CLK_SRC_XTAL`
0x05	Clock gate (OFF) The FECSS Frame Timer clock can be gated using this option. The application can gate FT clock in device idle state and FT is not active.	`M_RL_FECSS_FT_CLK_SRC_OFF`

- `c_ApllClkCtrl`

FECSS RF APLL clock block ON / OFF control.

Warning The APLL must be ON before:
- FECSS Factory Calibration API
- FECSS RF Runtime Calibration API
- FECSS Monitor Enable Trigger API
- Sensor Start API

Please refer sequences for more details.

Value	Definition	mmWaveLink Macro
0x00	Turn OFF APLL clock. Turn OFF APLL before entering deep sleep or APLL functionality is not needed by HOST. Typical use case : After data processing of a frame.	M_RL_FECSS_APLL_CTRL_OFF
0x0A	Turn ON APLL clock. Use this ON mode in typical APLL ON after deep sleep exit or whenever APLL functionality is needed by HOST. Typical use case: After deep sleep exit or before analog / sensor functionalities turn ON.	M_RL_FECSS_APLL_CTRL_ON
0xAA	Calibrate and turn ON APLL clock. This ON mode should be used in following cases: • Device is powered on after a long time. (Typically cold boot) • Significant temperature difference between subsequent boot cycles. • Whenever APLL block needs calibrations. This API turns the APLL off and returns error code if calibration fails.	M_RL_FECSS_APLL_CTRL_ON_CAL

2.3.15 FECSS Device RDIF Control API

This API is used for RDIF block configuration and enable/disable control.

The application can use this API to collect the radar Data over CMOS interface pins.

- Max 3 channels are supported in RDIF, which are mapped to the 3 RX channels of the sensor.
- it is possible to capture any one or two or all three of the DFE RX channels output.
- h_RxChCtrlBitMask is used to program the DFE RX channel enable which is mapped to RDIF channels.
- If less than 3 channels are enabled in DFE then only enabled channels will be mapped to lower RDIF channels without any holes.
- The max bits per sample supported is only 12 bits in RDIF, the number of bits and samples per chirp and per channel should match the DFE profile configuration.
- The DDR clock is 50MHz (APLL_FREQ/8) and data is sampled at both raising and falling edge of DDR clock.
- The max data rate of RDIF is $(50M \times 2 \times 4Datalines) = 400Mbps$
- The one frame clock duration is 12 DDR clock edges corresponds to 48bits.
- SATURATION_CNT_CH3 (sample 4), SATURATION_CNT_CH2 (sample 5) and SATURATION_CNT_CH1 (sample 6) - each 12b samples.

- In case of less number of enabled DFE channels, lower slots will have valid saturation count and upper slot will be padded with zeros.

Sideband Data Format:
The 96 bits (8 samples) sideband data can be appended at end of each chirp data (API enable). The side band data consists of the following information:

Value	Definition
FRAME_CNT[11:0]	12bits, 1st sample.
BURST_CNT[11:0]	12bits, 2nd sample.
CHIRP_CNT[11:0]	12bits, 3rd sample.
SATURATION_CNT[35:0]	12b per channel, there are three fixed slots named as below: • `SATURATION_CNT_CH3 (sample 4), SATURATION_CNT_CH2` (sample 5) and `SATURATION_CNT_CH1` (sample 6) - each 12b samples. • In case of less number of enabled DFE channels, lower slots will have valid saturation count and upper slot will be padded with zeros.
CRC-16[23:0]	16 bits CRC [15:0], MSB 8 bits are padded with zeros. (CRC is calculated for Chirp data + Sideband data) - sample 7 (LSB) and 8 (MSB).

CW Mode:

- In CW mode the DFE output samples will be sent out continuously for all the enabled channels.
- The max data rate supported is 400 Mbps. In case of max 3 enabled RX channels, the max DFE rate can be supported is 11.11 Mbps.
- The sideband data is not supported in CW mode.

API Definition

API Index	0x0011U
API Function Name	`rl_fecssDevRdifCtrl`
API Configuration Structure	`T_RL_API_FECSS_RDIF_CTRL_CMD`
API Response Structure	None

API Function

mmWaveLink C function declaration

- Configuration C Structure

Table 2.26: T_RL_API_FECSS_RDIF_CTRL_CMD

Field	Data Type	Size	Offset
c_RdifEnable	UINT8	1	0
c_RdifCfg	UINT8	1	1
h_RdifSampleCount	UINT16	2	2
c_TestPatternEn	UINT8	1	4
c_LaneRateCfg	UINT8	1	5
h_Reserved2	UINT16	2	6
h_TestPatrnInitCode	UINT16[4]	8	8
h_TestPatrnIncrCode	UINT16[4]	8	16
w_Reserved5	UINT32	4	24
Total		**28**	

Configuration Fields

- c_RdifEnable

FECSS RDIF block ON / OFF control. This control should be used to reset and enable/disable the RDIF data transfer.

Notes RDIF IP always goes through reset in every ON command.

Value	Definition
0x00	RDIF block is turned OFF
0x0A	RDIF block is turned ON

- c_RdifCfg

FECSS RDIF configuration settings can be used to enable various modes of operation of the RDIF IP. This API configures RDIF based on below settings.

Value	Definition
0x0	Disabled
0x1	Enabled

Bit Definition:

Bit Fields	Definition
Bit[0]	RDIF sideband data enable control. • The 96 bits (8 samples) sideband data will be appended at end of each chirp data when this mode is enabled. • **Note**: FECSS Rx Saturation Live Monitor Configuration API must be enabled in `c_FrameLivMonEn` to receive sideband data
Bit[1]	RDIF CW mode enable control. • The 96 bits (8 samples) sideband data will be appended at end of each chirp data when this mode is enabled. • In CW mode the DFE output samples will be sent out continuously for all the enabled channels at max rate of 400 Mbps. • The sideband data is not supported in CW mode.
Bit[3:2]	RDIF data swizzling mode enable control. • The RDIF channel data and the side band sample 12 bits data swizzling mode.
Bit[4]	RDIF Scrambler Mode enable control Scrambler Polynomial: $1 = z^{-18} + z^{-23}$
Bit[5]	Enable RDIF lane rate update. • By default per lane data rate is set to max 100 Mbps. • Max 400Mbs for 4 lanes. • The lane rate can be updated if this bit is set using `RDIF_LANE_RATE`.
Bit[7:6]	Reserved.

Warning Rx saturation monitor must be enabled to stream sideband data

Recommendation Scrambler should be enabled to avoid spurs during RDIF transmission

RDIF data swizzling mode control:

Value	Definition
b00	Pin0-bit0-Cycle1 Mode
b01	Pin3-bit0-Cycle1 Mode
b10	Pin0-bit0-Cycle3 Mode
b11	Pin3-bit0-Cycle3 Mode

▶ `h_RdifSampleCount`

FECSS RDIF number of samples per channel per chirp. This field should be exactly same as the `h_NumOfAdcSamples`. The config value should be a multiple of 4. This field does not include samples from sideband data.

Notes

- Variable chirp size is not supported. All the chirps will have the same size.
- It is not possible to send 512 samples in first chirp and 1024 samples in next chirp.
- Valid Range: 4 to 2048.

- `c_TestPatternEn`

FECSS test pattern block ON / OFF control. This control should be used to enable or disable the test pattern in DFE. This test pattern can be used to validate the RDIF and Data path buffer.

Value	Definition
0x0	TEST_PATTERN block is turned OFF.
0xA	TEST_PATTERN block is turned ON.

- `c_LaneRateCfg`

FECSS RDIF lane rate. The max lane rate supported 400 Mbps for 4 lanes (each lane max 100Mbps).

Value	Definition
0	400 Mbps
1	320 Mbps
2	200 Mbps
3	160 Mbps

Notes

- This configuration will be updated only if c_RdifCfg[bit 5] is set.
- By default Lane rate is set to 400Mbps.
- $[RDIFLaneRate > (RxChannels - (1152/ADCSamples)) \times 12 - bits \times ADCSamplingRate]$

- `h_TestPatrnInitCode`

RX channel [3:0] Test pattern init code. The offset value code to be used for the first sample of the test pattern data. RX3 is reserved.

- `h_TestPatrnIncrCode`

RX channel [3:0] Test pattern increment code. The value to be added for each successive sample for the test pattern data. RX3 is reserved.

2.3.16 FECSS Device Status Get API

This API response returns FECSS power and clock status.

API Definition

API Index	0x0012U
API Function Name	rl_fecssDevStatusGet
API Configuration Structure	None
API Response Structure	T_RL_API_FECSS_DEV_STS_RSP

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_DEV_STS_GET            ((UINT16)0x0012U)
// API Function Definition
T_RETURNTYPE rl_fecssDevStatusGet(UINT8 c_devIndex, T_RL_API_FECSS_DEV_STS_RSP
↪  *p_apiResData);
```

Response C Structure

Table 2.27: T_RL_API_FECSS_DEV_STS_RSP

Field	Data Type	Size	Offset
h_XtalClkFreq	UINT16	2	0
c_DevClkSrcState	UINT8	1	2
c_PowerModeState	UINT8	1	3
c_ChirpTimerResType	UINT8	1	4
c_FtClkState	UINT8	1	5
c_RfsBootSelfTest	UINT8	1	6
c_Reserved1	UINT8	1	7
h_RfsFwState	UINT16	2	8
c_PatchAndSilEna	UINT8	1	10
c_RfAndSensType	UINT8	1	11
w_Reserved2	UINT32	4	12
w_Reserved3	UINT32	4	16
Total		**20**	

Response Fields

h_XtalClkFreq

FECSS configures device XTAL clock frequency if c_DevClkCtrl =0x00. This value should match the application programmed value. Unit - 1LSB = 1/256 MHz Typical values are: 6400, 6656, 9830 and 10240

- `c_DevClkSrcState`

FECSS clock source state.
Valid configurations are as below:

Value	Definition
0x00	XTAL clock source. Clock Frequency: 25MHz, 26MHz, 38.4MHz or 40MHz (XTAL_FREQ)
0x0A	Fast clock source. Clock Frequency: 80MHz (DIG_PLL_FREQ/2).

- `c_PowerModeState`

FECSS `c_PowerMode` type.

- `c_ChirpTimerResType`

The RF start frequency resolution and timing related chirp parameters resolution selection. Please refer it to `c_ChirpTimerResol`

- `c_FtClkState`

FECSS frame timer clock source type. Please refer it to `c_FtClkCtrl`

- `c_RfsBootSelfTest`

RFS core IP boot self test status.

Value	Definition
0	Test Failed
1	Test Passed

Bit Field Definition:

Bit Field	Definition
Bit[0]	Self test status of RFS M3 core MPU.
Bit[1]	Self test status of RFS BIST FFT IP.
Bit[2]	Self test status of RFS GPADC IP.
Bit[7:3]	Reserved

- `h_RfsFwState`

RFS FW State.

Value	Definition
0x0000	Halt
0x4004	Booting
0x8008	Idle
0xFFFF	Fault
Cmd Id	API Execution

- `c_PatchAndSilEna`

RFS patch status in lower nibble:

Value	Definition
0x0	Disabled
0xA	Enabled

RFS SIL status in upper nibble:

Value	Definition
0x0	Disabled
0xA	Enabled

- `c_RfAndSensType`

RFS RF type in lower nibble:

Value	Definition
0x0	60GHz
0xA	77GHz

RFS sensor type in upper nibble:

Value	Definition
0x0	SOC
0xA	FE

2.3.17 FECSS RF Status Get API

This API returns FECSS RF channels and IP modules power state data.

API Definition

API Index	0x0013U
API Function Name	rl_fecssRfStatusGet
API Configuration Structure	None
API Response Structure	T_RL_API_FECSS_RF_STS_GET_RSP

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_RF_STS_GET            ((UINT16)0x0013U)
// API Function Definition
T_RETURNTYPE rl_fecssRfStatusGet(UINT8 c_devIndex, T_RL_API_FECSS_RF_STS_GET_RSP
↪  *p_apiResData);
```

Response C Structure

Table 2.28: T_RL_API_FECSS_RF_STS_GET_RSP

Field	Data Type	Size	Offset
w_RxChCtrlSts	UINT16	2	0
h_TxChCtrlSts	UINT16	2	2
c_Reserved0	UINT8	1	4
c_MiscCtrlSts	UINT8	1	5
c_ApllClkCtrlSts	UINT8	1	6
c_Reserved1	UINT8	1	7
w_Reserved2	UINT32	4	8
Total		**12**	

Response Fields

- w_RxChCtrlSts

FECSS RFS RX channels control status, 1 bit per channel. This field contains read-back of information configured in h_RxChCtrlBitMask.

- h_TxChCtrlSts

FECSS RFS TX channels control status, 1 bit per channel. This field contains read-back of information configured in h_TxChCtrlBitMask.

- c_MiscCtrlSts

FECSS RFS miscellaneous block / clock control status, 1 bit per control. This field contains read-back of information configured in c_MiscCtrl.

- c_ApllClkCtrlSts

FECSS APLL status.

Value	Definition
0x00	APLL block is OFF.
0x0A	APLL block is ON.

2.3.18 FECSS RF Fault Status Get API

This API returns the CPU Firmware status.

Notes Application should read the fault status of the FECSS device when FECSS_RFS_FAULT interrupt is received on FEC_INTR1 (IRQ[3]) in application M4 core.

API Definition

API Index	0x0014U
API Function Name	rl_fecssRfFaultStatusGet
API Configuration Structure	None
API Response Structure	T_RL_API_RFS_FAULT_STS_GET_RSP

- API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_FAULT_STS_GET         ((UINT16)0x0014U)
// API Function Definition
    T_RETURNTYPE rl_fecssRfFaultStatusGet(UINT8 c_devIndex, T_RL_API_RFS_FAULT_STS_GET_RSP
    ↪  *p_apiResData);
```

- Response C Structure

Table 2.29: T_RL_API_RFS_FAULT_STS_GET_RSP

Field	Data Type	Size	Offset
h_RfsFwState	UINT16	2	0
c_PatchAndSilEna	UINT8	1	2
c_RfAndSensType	UINT8	1	3
w_Reserved1	UINT32	4	4
c_FaultType	UINT8	1	8
c_ErrorCode	UINT8	1	9
h_LineNum	UINT16	2	10
w_AbortPC	UINT32	4	12
w_AbortLR	UINT32	4	16
w_AbortApsr	UINT32	4	20
w_AbortSp	UINT32	4	24
w_CfsrReg	UINT32	4	28
w_HfsrReg	UINT32	4	32
w_MmFarReg	UINT32	4	36
w_BFarReg	UINT32	4	40
w_ShcsrReg	UINT32	4	44
c_ExceptionCount	UINT8	1	48
c_Reserved2	UINT8	1	49
h_Reserved3	UINT16	2	50
Total		**52**	

Response Fields

- h_RfsFwState

RFS FW state:

Value	Definition
0x0000	Halt
0x4004	Booting
0x8008	Idle
0xFFFF	Fault
Cmd Id	API Execution

- `c_PatchAndSilEna`

RFS patch status in lower nibble:

Value	Definition
0x0	Disabled
0xA	Enabled

RFS SIL status in upper nibble:

Value	Definition
0x0	Disabled
0xA	Enabled

- `c_RfAndSensType`

RFS RF type in lower nibble:

Value	Definition
0x0	60GHz
0xA	77GHz

RFS sensor type in upper nibble:

Value	Definition
0x0	SOC
0xA	FE

- `c_FaultType`

RFS CPU fault type:

Value	Definition
0x00	No fault
0x81	CPU Fault
0x82	FW ASSERT

- `c_ErrorCode`

RFS FW fault error codes. This field is valid only if the `c_FaultType` = `FW_ASSERT`.

Id	Error Code
0x00	No Fault
0x81	Stray MBOX Interrupt Error in FEC
0x82	FEC Register Read back Error
0x83	Functional Frame triggered in CalMon duration
0x84	Invalid data
0x85	Array index out of bound
0x86	Attempted to de reference null pointer
0x87	Resource busy
0x88	Timeout error
0x89	Stack overflow
0x8A	Undefined error

- `h_LineNum`

RFS firmware fault line number. This field is valid only if the `c_FaultType` = `FW_ASSERT`.

- `w_AbortPC`

RFS PC (Program Counter) value where fault/ assert is detected.

- `w_AbortLR`

RFS LR (Link Register, Typically contains caller's address) register value when fault/ assert is detected.

- `w_AbortApsr`

RFS APSR (Application Program Status Register) register value when fault/ assert is detected.

- `w_AbortSp`

RFS SP (Stack Pointer) when fault/ assert is detected.

- `w_CfsrReg`

RFS CFSR (Configurable Fault Status Register) register value when fault/ assert is detected.

- w_HfsrReg

RFS HFSR (Hard Fault Status Register) register value when fault/ assert is detected.

- w_MmFarReg

RFS MMFAR (Memory Fault Address Status Register) register value when fault/ assert is detected.

- w_BFarReg

RFS BFAR (Bus Fault Status Address Register) register value when fault/ assert is detected.

- w_ShcsrReg

RFS SHCSR (Exception Configuration and Status Register) register value when fault/ assert is detected.

- c_ExceptionCount

RFS exception count.

2.3.19 FECSS Die Id Get API

This API returns sensor device die id registers. Die-id can be used to uniquely identify the TI mmWave Radar devices.

API Definition

API Index	0x0015U
API Function Name	rl_fecssDieIdGet
API Configuration Structure	None
API Response Structure	T_RL_API_SENSOR_DIEID_RSP

- API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_DIEID_GET             ((UINT16)0x0015U)
// API Function Definition
    T_RETURNTYPE rl_fecssDieIdGet(UINT8 c_devIndex, T_RL_API_SENSOR_DIEID_RSP *p_apiResData);
```

▶ Response C Structure

Table 2.30: T_RL_API_SENSOR_DIEID_RSP

Field	Data Type	Size	Offset
w_DieIdData	UINT32[4]	16	0
Reserved	UINT32[4]	16	16
Total		32	

Response Fields

▶ w_DieIdData

Sensor die id data array.

2.3.20 FECSS Rx-Tx Calibration Data Get API

This API can be used to store the RX-TX factory calibration data to flash or external memory if calibration is done in the factory. The FECSS Rx-Tx Calibration Data Set API can be used to restore the stored data back to the device.

API Definition

API Index	
API Function Name	rl_fecssRfRxTxCalDataGet
API Configuration Structure	None
API Response Structure	T_RL_API_FECSS_RXTX_CAL_DATA

▶ API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_RXTX_CAL_DATA_GET     ((UINT16)0x0016U)
// API Function Definition
    T_RETURNTYPE rl_fecssRfRxTxCalDataGet(UINT8 c_devIndex, T_RL_API_FECSS_RXTX_CAL_DATA
    ↪  *p_apiResData);
```

2.3.21 FECSS Rx-Tx Calibration Data Set API

This API can be used to restore/ update the Rx-Tx calibration data stored using the FECSS Rx-Tx Calibration Data Get API. The update can be done dynamically or at the FECSS power up.

Use FECSS Rx-Tx Calibration Data Get API and FECSS Rx-Tx Calibration Data Set API to update c_CalRxGainCodes and h_CalTxBiasCodes to support dynamic update of multiple chirp profiles in inter-frame time.

Warning This API **does not** apply the calibration data to the hardware registers. Please use the FECSS RF Runtime Calibration API to apply the updated calibration data.

API Definition

API Index	0x0017
API Function Name	rl_fecssRfRxTxCalDataSet
API Configuration Structure	T_RL_API_FECSS_RXTX_CAL_DATA
API Response Structure	None

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_RXTX_CAL_DATA_SET     ((UINT16)0x0017U)
// API Function Definition
    T_RETURNTYPE rl_fecssRfRxTxCalDataSet(UINT8 c_devIndex, const
    ↪  T_RL_API_FECSS_RXTX_CAL_DATA *p_apiCmdData);
```

Configuration C Structure

Table 2.31: T_RL_API_FECSS_RXTX_CAL_DATA

Field	Data Type	Size	Offset
c_CalRxTxResValidity	UINT8	1	0
c_CalRxGainCodes	UINT8[3]	3	1
c_CalTxBackOffMap	UINT8[4]	4	4
h_CalTxBiasCodes	UINT16[4][3]	24	8
w_Reserved	UINT32	4	32
Total		**36**	

Configuration Fields

Rx and Tx calibration data structure. This structure is sub-structure in the T_RL_API_FECSS_FACT_CAL_DATA .

c_CalRxTxResValidity

Rx and Tx calibration data validity status bits. This field also includes duplicate validity bits for data redundancy.

Value	Definition
0x0	INVALID - Calibration data is invalid.
0x1	VALID - Calibration data is valid.

bit definitions are as follows:

Bit Field	Definition
bits [0]	RX Gain calibration validity status
bits [1]	TX power calibration validity status
bits [3:2]	RESERVED
bits [4]	RX Gain calibration redundant validity status
bits [5]	TX power calibration redundant validity status
bits [7:6]	Reserved

- `c_CalRxGainCodes`

Calibrated Rx gain codes for all temperature bins. Rx gain codes byte contains following fields:

Bit Field	Definition
bits [4:0]	IFA_GAIN_CODE IFA gain code to achieve desired overall Rx gain $IFA\_GAIN(dB) = IFA\_GAIN\_CODE \times 2dB - 10dB$ Valid Code Rage: [0, 10]
bits [7:5]	RFA_GAIN_CODE Optimal RFA gain code for the calibrated Rx Gain in respective temperature bin. Valid Range: [0, 3]

- `c_CalTxBackOffMap`

Calibrated backoff value for each Tx channel. This map is updated based on the `c_TxPwrCalTxEnaMask`. The backoff information is used by various monitoring and internal configuration APIs to derive control parameters. Hence this information must be coherent with the `h_CalTxBiasCodes`.

- 1 LSB = 0.5 dB backoff
- Valid Range: [0, 56] i.e. [0dB , 26dB]

The array stored 1-byte per Tx channel as follows:

Byte	Definition
Byte [0]	Tx0 Backoff Code
Byte [1]	Tx1 Backoff Code
Bytes [3:2]	Reserved

- `h_CalTxBiasCodes`

2-Byte TX_PA_CODE for each Tx channel per temperature bin for given `c_CalTxBackOffMap`. Each TX_PA_CODE field contains a byte per Tx PA stage, which are packed as follows:

Byte	Definition
Byte [0]	PA stage 1 bias code
Byte [1]	PA stage 2 bias code

and the overall TX_PA_CODE array is as follows:

Bytes	Definition
Bytes [1:0]	`h_CalTxBiasCodes[Tx0][LOW_TEMP_BIN]`
Bytes [3:2]	`h_CalTxBiasCodes[Tx0][MID_TEMP_BIN]`
Bytes [5:4]	`h_CalTxBiasCodes[Tx0][HIGH_TEMP_BIN]`
Bytes [7:6]	`h_CalTxBiasCodes[Tx1][LOW_TEMP_BIN]`
Bytes [9:8]	`h_CalTxBiasCodes[Tx1][MID_TEMP_BIN]`
Bytes [11:10]	`h_CalTxBiasCodes[Tx1][HIGH_TEMP_BIN]`
Bytes [23:12]	Reserved

2.3.22 FECSS RFS Debug Configuration API

This is an optional API which can be used to configure the RFS internal debug data dump for calibrations and monitors.

API Definition

API Index	0x0018U
API Function Name	`rl_fecssRfsDbgCfg`
API Configuration Structure	`T_RL_API_FECSS_RFS_DBG_CFG_CMD`
API Response Structure	None

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_RFS_DBG_CFG           ((UINT16)0x0018U)
// API Function Definition
    T_RETURNTYPE rl_fecssRfsDbgCfg(UINT8 c_devIndex, const T_RL_API_FECSS_RFS_DBG_CFG_CMD
    ↪  *p_apiCmdData);
```

Configuration C Structure

Table 2.32: `T_RL_API_FECSS_RFS_DBG_CFG_CMD`

Field	Data Type	Size	Offset
w_RfsDbgLogAddress	UINT32	4	0
w_Reserved0	UINT32	4	4
w_Reserved1	UINT32	4	8
Total		**12**	

Configuration Fields

`w_RfsDbgLogAddress`

FECSS RFS debug log address. A non-zero of address of format `0x224XXXXX` enables the debug data store and `0x0` disables the debug data dump.

Warning

- Application must allocate at least 4kB and address aligned data buffer starting from this address
- FECSS does not retain this configuration across power cycles, hence the debug data will be disabled after every FECSS power-up.

2.3.23 FECSS RF Runtime Tx CLPC API

This API can be used to calibrate the Tx channels during in-field operation. The API calibrates Tx channels using CLPC algorithm and immediately applies the results to the hardware registers.

Assumptions

- APLL must be enabled before using FECSS Device Clock Ctrl API
- Tx and Rx channel configuration is set using FECSS RF Power On Off API
- Live monitor faults and loopbacks are disabled

API Definition

API Index	0x0019U
API Function Name	rl_fecssRlRuntimeTxClpcCal
API Configuration Structure	T_RL_API_FECSS_RUNTIME_TX_CLPC_CAL_CMD
API Response Structure	T_RL_API_FECSS_RUNTIME_TX_CLPC_CAL_RSP

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_FECSS_RUNTIME_TX_CLPC_CAL    ((UINT16)0x0019U)
// API Function Definition
    T_RETURNTYPE rl_fecssRlRuntimeTxClpcCal(UINT8 c_devIndex, const
    ↪  T_RL_API_FECSS_RUNTIME_TX_CLPC_CAL_CMD *p_apiCmdData,
    ↪  T_RL_API_FECSS_RUNTIME_TX_CLPC_CAL_RSP* p_apiResData);
```

Configuration C Structure

Table 2.33: T_RL_API_FECSS_RUNTIME_TX_CLPC_CAL_CMD

Field	Data Type	Size	Offset
c_CalMode	UINT8	1	0
xc_OverrideCalTemp	SINT8	1	1
c_Reserved0	UINT8[2]	2	2
c_CalTxBackOffSel	UINT8[4]	4	4
w_Reserved1	UINT32	4	8
h_CalRfFreq	UINT16	2	12
xh_CalRfSlope	SINT16	2	14
c_TxPwrCalTxEnaMask	UINT8[4]	4	16
w_Reserved2	UINT32	4	20
h_TxStgOverrideCodes	UINT16[4]	8	24
w_Reserved3	UINT32[4]	16	32
Total		**48**	

▶ Response C Structure

Table 2.34: T_RL_API_FECSS_RUNTIME_TX_CLPC_CAL_RSP

Field	Data Type	Size	Offset
c_CalRunStatus	UINT8	1	0
xc_CalibrationTemp	SINT8	1	1
h_Reserved0	UINT16	2	2
h_TxStgCodes	UINT16[4]	8	4
w_Reserved1	UINT32[3]	12	12
Total		**24**	

Configuration Fields

Tx CLPC calibration API configuration Structure.

▶ c_CalMode

TX CLPC calibration mode flags. The API supports following modes of operations:

Mode	Description
Calibrate Mode	This mode performs Tx CLPC calibration immediately and applies results to the hardware registers. The API returns calibration results as a response to the API and are not stored.
Override Mode	This mode can be used to skip the calibrations and apply the results provided in the configuration directly

The bitwise details of the flags is as follows:

Bits	Description
bit 0	Calibration Mode • 0: [Default] Calibrate Mode • 1: Bypass Mode
bits [7:1]	Reserved

▶ xc_OverrideCalTemp

Override mode calibration temperature value. This field is applicable only when Override Mode is enabled. The information is used to correctly set the internal temperature dependent parameters.

- $1\ LSB = 2^oC$
- Valid Range: $[-40^oC, 120^oC]$

- `c_CalTxBackOffSel`

Tx backoff value per Tx channel. This filed is used in both calibration and override modes. Functionally it is similar to the `c_CalTxBackOffSel` in FECSS Factory Calibration API.

Byte definition is as follows, where 1LSB = 0.5dB:

Byte Field	Definition
Byte[0]	Output power back-off value for Tx0
Byte[1]	Output power back-off value for TX1
Byte[3-2]	Reserved

- `h_CalRfFreq`

Calibration chirp RF start Frequency.

Recommendation Configuration of the operating band should be Mid frequency .

RF Frequency	Resolution
60GHz	$1LSB = (3 \times APLL\_FREQ)/2^{10} \approx 1.172MHz$ Valid Range: 57 GHz to 64 GHz for APLL_FREQ = 400MHz: 0xBE00U (57GHz) to 0xD555U (64 GHz)* *Variant specific limitations in Release Notes
77GHz	$1LSB = (4 \times APLL\_FREQ)/2^{10} \approx 1.562MHz$ Valid Range: 76 GHz to 81 GHz for APLL_FREQ = 400MHz: 0x0000BE00 (76GHz) to 0x0000CA80 (81GHz)

- `xh_CalRfSlope`

RFS calibration chirp RF Frequency Slope.

Recommendation The typical ramp time used in FW for calibration chirps are 55us.
For 60GHz: (APLL_FREQ = 400MHz): 2.2MHz/us (0x4D).
For 77GHz: (APLL_FREQ = 400MHz): 2.2MHz/us (0x3A).

RF Slope	Resolution
60GHz	$1LSB = (3 \times 400 \times APLL\_FREQ)/2^{24} \approx 28.610kHz/us$,signed number Valid Range: -3495 to 3495 (+/- 100MHz/us (APLL_FREQ = 400MHz))
77GHz	$1LSB = (4 \times 400 \times APLL\_FREQ)/2^{24} \approx 38.147kHz/us$, signed number Valid Range: -3495 to 3495 (+/- 100MHz/us (APLL_FREQ = 400MHz))

- `c_TxPwrCalTxEnaMask`

Per channel Tx enable mask for calibrations.

Recommendation

- In case more than 1 TXs are enabled during the chirp, then enabling the same TXs during calibration will result in better TX output power accuracy.
- Recommended Calibration Masks: 0x3, 0x1. This setting will enable both Tx channels while calibrating Tx 0 and copy the results for Tx1 without separately calibrating it.

Calibrated bias code for all Tx channels.

Byte Field	Definition
Byte[0]	Tx enable mask for calibrating Tx0. If Tx0 is disabled in this mask, calibration data from the lowest enabled Tx channel from this mask will be applied to Tx0. Tx0 calibration will be skipped from the calibration if the mask is 0.
Byte[1]	Tx enable mask for calibrating Tx1. If Tx1 is disabled in this mask, calibration data from the lowest enabled Tx channel from this mask will be applied to Tx1. Tx1 calibration will be skipped from the calibration if the mask is 0.
Byte[3-2]	Reserved

Warning This This mask is applicable even in the override mode. This helps to selectively update Tx stage codes.

- `h_TxStgOverrideCodes`

Override mode TX_PA_CODE for each Tx channel.

Response Fields

Tx Runtime CLPC calibration API response Structure.

- `c_CalRunStatus`

Calibration run status.

- `xc_CalibrationTemp`

Tx CLPC calibration temperature in $1\ LSB = 2^oC$. In override mode, `xc_OverrideCalTemp` is reported back in this field.

- h_TxStgCodes

Calibrated TX_PA_CODE for each Tx channel. In override mode, this table is updated according to the c_TxPwrCalTxEnaMask.

2.4 mmWave Sensor APIs

mmWave sensor APIs provide interface to configure, verify and control the FMCW chirp, burst and frame parameters. Following diagram describes various chirp configuration parameters for the mmWave Low devices.

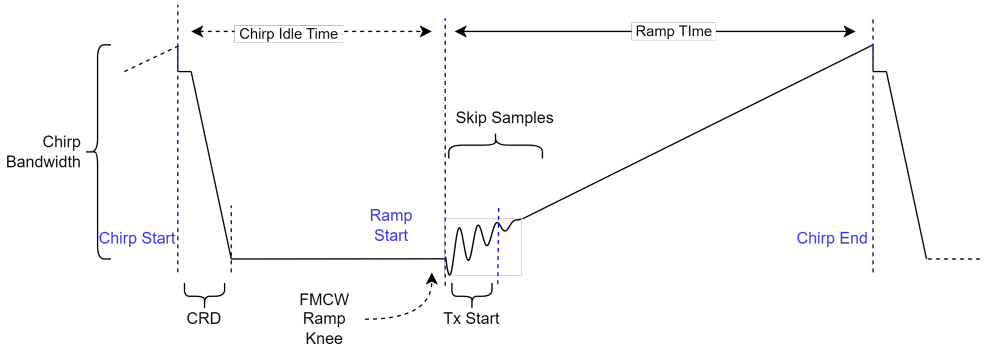


Figure 2.1: FMCW Chirp Parameters

Parameter	Description and Notes
Chirp Idle Time	Time period between end of one chirp's frequency ramp and next chirp's frequency ramp
Chirp Start	Start of a chirp idle time
ADC Start Time	Time from the knee of the ramp after which ADC samples are captured in the Rx buffer
Tx Start Time	Time from knee of the ramp after which transmitters are enabled This time could be negative, i.e. transmitters can be enabled in idle time itself
Chirp Ramp End Time	Time period of frequency ramp in a chirp
Chirp End	End of the FMCW RAMP This event coincides with the chirp start of the following chirp CRD is included in the chirp idle time of the next chirp
Chirp Period	Sum of chirp idle time and chirp ramp time
Chirp Bandwidth	Difference between the start frequency and end frequency of a chirp

mmWave devices pack multiple chirps in a single transmission called burst. A set of bursts in called as frame. Sensor APIs allow transmission of multiple frames with a single trigger. Following diagram and table describes important terms and parameters related to bursts and frames.

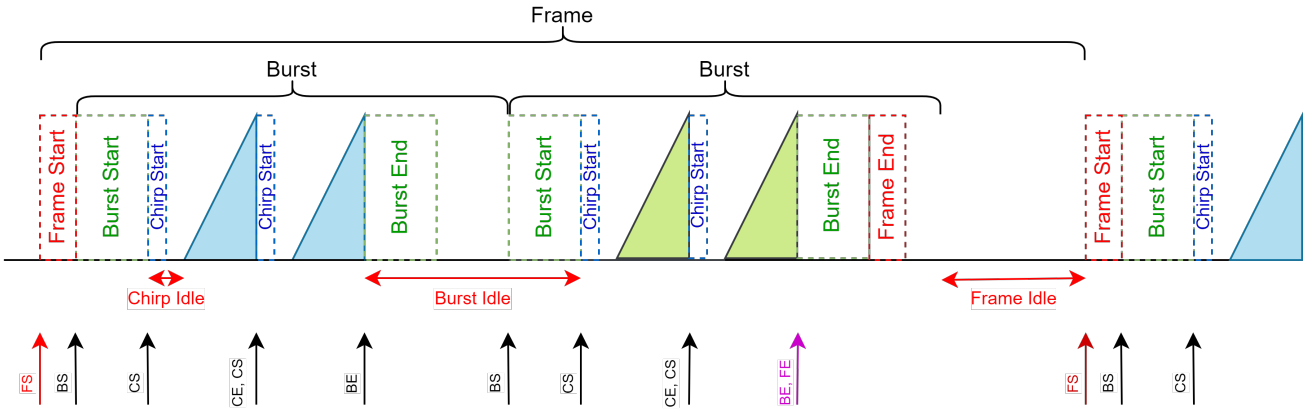


Figure 2.2: Frame and Burst Events

Parameter	Description and Notes
Chirp	Single FMCW transmission with defined slope and start frequency
Burst	Set of consecutive chirps
Frame	Set of bursts
Burst Start (BS)	Start of a burst This event is periodically generated by the Chirp Timer hardware module. RFS firmware powers up and configures analog front end upon receiving this event.
Burst End (BE)	End of a burst This event is generated at the end of the last chirp of a burst. The RFS firmware powers down the analog front end components enabled during Burst Start upon receiving this event.
Burst Idle Time	Time between the Chirp End of the last chirp in a burst and the first Chirp Start of the next burst. This time should be sufficient to execute Burst Start (for next burst) and Burst End sequences.
Burst Periodicity	Time period between two consecutive Burst Start events in a frame
Frame Start (FS)	Start of a frame This event is periodically generated by the Frame Timer hardware module. RFS firmware powers up and configures minimal set of front end modules required for Chirp Timer upon receiving this event.
Frame End (FE)	End of a frame This event is generated by the Chirp Timer hardware module upon completion of the last chirp of a last burst in a frame. RFS firmware turns off all the front end modules enabled during frame start. The actual hardware event coincides with the Burst End of the last burst in a frame, but the Burst End takes higher priority and executes first
Frame Idle Time	Idle time between end of the last burst in first burst to the start of the first burst in following frame This time should be sufficient to execute Frame Start (for the following frame) and Frame End sequences.
Frame Periodicity	Time between two consecutive Frame Start events

Note Chirp timing parameters like chirp idle time, skip samples, and chirp ramp end time, along with parameters such as burst and frame times are to be set according to conditions mentioned as in Chirp, Burst and Frame Timing Constraints.

The sensor module provides following APIs to configure and control the mmWave Low sensors:

- Sensor Profile Common Configuration API
- Sensor Chirp Profile Common Configuration Get API
- Sensor Chirp Profile Time Configuration API
- Sensor Chirp Profile Time Configuration Get API
- Sensor Per-Chirp Configuration API
- Sensor Per-Chirp Configuration Get API
- Sensor Per-Chirp Control API
- Sensor Per-Chirp Control Get API
- Sensor Frame Configuration API
- Sensor Frame Configuration Get API
- Sensor Start API
- Sensor Stop API
- Sensor Status Get API
- Sensor Dynamic Power Save Disable Configuration API
- Sensor Dynamic Power Save Disable Configuration Get API
- Sensor Loopback Configuration API
- Sensor Loopback Enable API

Warning Sensor configuration APIs must be issued outside the frame active time (Outside Frame Start and Frame End event boundaries). The application should issue Sensor Stop API if reconfiguration is required in frame active time.

2.4.1 Sensor Profile Common Configuration API

This API function configures the FMCW radar chirp profile common parameters like sample rate, num of samples, ramp end time, TX/RX gain, etc. These parameters will be common for all the chirps in a frame.

Warning Profile configurations are directly written to the hardware registers and hence will not be retained across deep-sleep. Application must re-issue this API after waking up from deep sleep.

API Definition

API Index	0x0040U
API Function Name	`rl_sensChirpProfComnCfg`
API Configuration Structure	`T_RL_API_SENS_CHIRP_PROF_COMN_CFG`
API Response Structure	None

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_SENS_PROF_COMN_CFG              ((UINT16)0x0040U)
// API Function Definition
    T_RETURNTYPE rl_sensChirpProfComnCfg(UINT8 c_devIndex, const
    ↪  T_RL_API_SENS_CHIRP_PROF_COMN_CFG *p_apiCmdData)
```

Configuration C Structure

Table 2.35: T_RL_API_SENS_CHIRP_PROF_COMN_CFG

Field	Data Type	Size	Offset
c_DigOutputSampRate	UINT8	1	0
c_DigOutputBitsSel	UINT8	1	1
c_DfeFirSel	UINT8	1	2
c_VcoMultiChipMode	UINT8	1	3
h_NumOfAdcSamples	UINT16	2	4
c_ChirpTxMimoPatSel	UINT8	1	6
c_MiscSettings	UINT8	1	7
c_HpfFastInitDuration	UINT8	1	8
c_Reserved	UINT8	1	9
h_CrdNSlopeMag	UINT16	2	10
w_Reserved	UINT32	4	12
h_ChirpRampEndTime	UINT16	2	16
c_ChirpRxHpfSel	UINT8	1	18
c_Reserved2	UINT8	1	19
w_Reserved3	UINT32[4]	16	20
Total		**36**	

Configuration Fields

c_DigOutputSampRate

Digital output sampling rate of the Rx ADC.
The ADC sample rate depends on the APLL clock and is computed uses the following formula:

$$ADC\_SAMPLING\_RATE(MSPS) = \frac{APLL\_FREQ(MHz)}{c\_DigOutputSampRate \times 4}$$

Valid Range: [8, 100]

recommended sampling rates are as follows (assuming APLL_FREQ = 400MHz):

c_DigOutputSampRate	sampling Rate (MSPS)
8	12.5
10	10
13	7.692
15	6.667
20	5
25	4
40	2.5
50	2
60	1.667
80	1.25
100	1

- c_DigOutputBitsSel

Digital ADC output bit width select control.
This field controls the section of bits from the internal 16-bit output, which will be stored to the ADC buffer. Rx ADC always sends 16-bits to the output, depending on the configuration up to 4-bits are dropped from the result and the result is sign extended to map to a valid signed 16-bit number. xWRLx432 devices support following output options:

c_DigOutputBitsSel	Description
0	Digital sample output is 12 MSB bits of DFE after rounding 4 LSBs
1	Digital sample output is 12 bits after rounding 3 LSBs and clipping 1 MSB
2	Digital sample output is 12 bits after rounding 2 LSBs and clipping 2 MSB
3	Digital sample output is 12 bits after rounding 1 LSBs and clipping 3 MSB
4	Digital sample output is 12 LSB bits after clipping 4 MSB
5	Digital sample output is 16 bits

Valid Range: [0,5]

- `c_DfeFirSel`

DFE FIR filter selection.
This option selects the configuration of the final DFE FIR filter. The options are as follows:

c_DfeFirSel	Description
0	Long Filter (90% visibility) This provides visibility to a larger range of IF frequencies: 0 to 0.45 ×Sampling Rate. Beyond that, the filter response starts drooping and enters filter transition band.
1	Short Filter (80% visibility) This provides faster settled outputs but the IF frequency range visible is 0 to 0.40 ×Sampling Rate. Beyond that, the filter response starts drooping and enters filter transition band.

- `c_VcoMultiChipMode`

Sensor VCO selection control.

c_VcoMultiChipMode	Description
0	SINGLE_CHIP_VCO **Default Mode**: The device is in non-cascaded configuration
1	MULTI_CHIP_PRIMARY_VCO The device is controlling the VCOs in cascaded mode
2	MULTI_CHIP_SECONDARY_VCO The device is controlled by the VCO from a primary device in a cascaded configuration

Warning xWRLx432 devices **do not** support cascaded configuration

- `h_NumOfAdcSamples`

Number of ADC samples to capture in a chirp for each RX channels.
Valid Range: [2, 2048]

- `c_ChirpTxMimoPatSel`

Tx output MIMO pattern control.
This field controls the various BPM, TDMA output patterns supported by the device.

c_ChirpTxMimoPatSel	Description
0	Disabled Tx enable and BPM is controlled by h_ChirpTxEnSel and h_ChirpTxBpmEnSel registers respectively for all the chirps.
1	TDMA_2TX h_ChirpTxEnSel register or settings corresponding to the bit M_RL_SENS_PER_CHIRP_PARAM_INDEX_IDLE_TIME enabled in h_PerChirpParamCtrl are ignored and internally varied as [01; 10; 01; 10; …] in chirp 0, 1, 2, 3, … and so on in the frame. h_ChirpTxBpmEnSel register continues to control the BPM for all enabled Tx channels.
4	BPM_2TX h_ChirpTxBpmEnSel register or settings corresponding to the bit M_RL_SENS_PER_CHIRP_PARAM_INDEX_TX_START_TIME enabled in h_PerChirpParamCtrl are ignored and is internally varied as [00; 10; 00; 10; …] in chirp 0, 1, 2, 3, … and so on in the frame. h_ChirpTxEnSel register continues to control Tx enable for all Tx channels.

- c_MiscSettings

Miscellaneous chirp control settings bit-mask.

Bit Index	Description	mmWaveLink Macro
0	**HPF Fast Init Disable** This bit **Disables** HPF fast init for all the chirps. When HPF fast init is enabled, Chirp Timer initializes the HPF by temporarily increasing HPF bandwith (20X BW) for faster settling at the start of the ramp. Width of the initialization window is controlled by c_HpfFastInitDuration and is starts 100ns before the xh_ChirpTxStartTime.	M_RL_SENS_MISC_HPF_FAST_INIT_DIS_BIT

Bit Index	Description	mmWaveLink Macro
1	**CRD Disable** This bit **Disables** the CRD for all chirps. When CRD is enabled, synthesizer is controlled in the ramp down portion of the chirp to avoid undershoot / overshoot of synthesizer and mitigate the fixed frequency RF spurs. Magnitude of the CRD ramp slope is controlled by h_CrdNSlopeMag	M_RL_SENS_MISC_CRD_DIS_BIT
2	**CRD Dither Disable** This bit **Disables** the CRD dither. The filed is applicable only of the CRD if enabled.	M_RL_SENS_MISC_CRD_DITHER_DIS_BIT
3	**PA Blanking Enable** This bit enables PA Blanking. Device supports PA blanking through an input signal to the device in order to stop the transmission from the device instantaneously. This feature can be particularly useful, when the device is used in presence of other RF transceiver devices for various purposes (e.g. 5G communication).	M_RL_SENS_MISC_PA_BLANK_EN_BIT

Recommended and Default Value: 0x00

- c_HpfFastInitDuration

This field controls the width of the HPF fast initialization pulse generated by the Chirp Timer. The pulse starts 100ns before xh_ChirpTxStartTime and the timing should be chosen in accordance to h_ChirpAdcStartTime.

- Unit: $1\ LSB = 40/APLL\_FREQ$, typically 100ns with 40MHz XTAL
- Range: [5, 100]
- Recommended Value: 15 (1.5us for 40MHz XTAL)

- h_CrdNSlopeMag

Magnitude if the CRD slope. The sign of the ramp is detected automatically based on the start frequency.

Device RF Type	Resolution
60GHz	$1LSB = (3 \times 400 \times APLL\_FREQ)/2^{20} \approx 457.763kHz/us$,unsigned number Valid Range: 0x0 to 0xC00 (1406 MHz/us with APLL_FREQ = 400MHz)
77GHz	$1LSB = (4 \times 400 \times APLL\_FREQ)/2^{20} \approx 610.351kHz/us$,unsigned number Valid Range: 0x0 to 0x925 (1428 MHz/us with APLL_FREQ = 400MHz)

Recommendation

- Recommended method to compute the CRD slope magnitude is

$$CRD\_Slope = Chirp\_Ramp\_End\_Time \times Chirp\_Slope/min(Chirp\_Idle\_Time - 1us, 6us)$$

  where Chirp_Ramp_End_Time, Chirp_Slope and Chirp_Idle_Time are controlled by `h_ChirpRampEndTime`, `xh_ChirpRfFreqSlope` and `h_ChirpIdleTime` fields respectively,
- CRD Ramp down time must be $<= min(Chirp\_Idle\_Time - 1U, 6us)$

▶ h_ChirpRampEndTime

Chirp ramp end time.
This is a common ramp end time for all the chirps in the frame. The time value of the ramp end time depends on the Time Resolution selected by `c_ChirpTimerResol` during the FECSS powerup.

Time Resolution	Ramp End Time Resolution
High	1 LSB = 8/ APLL_FREQ (Typically 20ns with 40MHz XTAL) Valid Range: [1, 65535 (≈1.3ms)]
Low	1 LSB = 40/ APLL_FREQ (Typically 100ns with 40MHz XTAL) Valid Range: [1, 65535 (≈6.5ms)]

Warning User should ensure that the complete frequency sweep is within the valid frequency range. DFP does not perform checks on the end frequency to detect frequency overflow. Valid frequency ranges are as follows:

- 60 GHz Device: 57 GHz - 64 GHz
- 77 GHz Device: 76 GHz - 81 GHz

Note This filed is not applicable for `M_RL_SENS_FRAME_CW_CZ_TRIG` mode.

▶ c_ChirpRxHpfSel

DFE HPF cut-off frequency control. This is a common HPF cutoff frequency for all the chirps.

c_ChirpRxHpfSel	HPF Cutoff Frequency
0	175kHz
1	350kHz
2	700kHz
3	1400kHz

2.4.2 Sensor Chirp Profile Common Configuration Get API

This API reads the hardware registers and returns the configuration done using the Sensor Profile Common Configuration API.

API Definition

API Index	0x0041
API Function Name	rl_sensChirpProfComnCfgGet
API Configuration Structure	None
API Response Structure	T_RL_API_SENS_CHIRP_PROF_COMN_CFG

▶ API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_SENS_PROF_COMN_CFG_GET        ((UINT16)0x0041U)
// API Function Definition
    T_RETURNTYPE rl_sensChirpProfComnCfgGet(UINT8 c_devIndex,
    ↪  T_RL_API_SENS_CHIRP_PROF_COMN_CFG *p_apiResData);
```

2.4.3 Sensor Chirp Profile Time Configuration API

This API function configures the FMCW radar chirp profile timing parameters like idle time, ADC start time, TX start time, RF Slope, RF start freq, TX and BPM enable. These parameters are common for all chirps in a frame. Sensor Per-Chirp Configuration API can be used to dither/ vary certain parameters in every chirp.

Warning Profile configurations are directly written to the hardware registers and hence will not be retained across deep-sleep. Application must re-issue this API after waking up from deep sleep.

API Definition

API Index	0x0042U
API Function Name	rl_sensChirpProfTimeCfg
API Configuration Structure	T_RL_API_SENS_CHIRP_PROF_TIME_CFG
API Response Structure	None

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_SENS_PROF_TIME_CFG              ((UINT16)0x0042U)
// API Function Definition
    T_RETURNTYPE rl_sensChirpProfTimeCfg(UINT8 c_devIndex, const
    ↪  T_RL_API_SENS_CHIRP_PROF_TIME_CFG *p_apiCmdData);
```

Configuration C Structure

Table 2.37: T_RL_API_SENS_CHIRP_PROF_TIME_CFG

Field	Data Type	Size	Offset
h_ChirpIdleTime	UINT16	2	0
h_ChirpAdcStartTime	UINT16	2	2
xh_ChirpTxStartTime	SINT16	2	4
xh_ChirpRfFreqSlope	SINT16	2	6
w_ChirpRfFreqStart	UINT32	4	8
h_ChirpTxEnSel	UINT16	2	12
h_ChirpTxBpmEnSel	UINT16	2	14
w_Reserved1	UINT32	4	16
w_Reserved2	UINT32[4]	16	20
Total		**36**	

Configuration Fields

h_ChirpIdleTime

Chirp idle time code
The time value of the chirp idle time depends on the timer resolution (c_ChirpTimerResol) selected during the FECSS boot.
This value will be ignored if the bit M_RL_SENS_PER_CHIRP_PARAM_INDEX_IDLE_TIME is enabled in h_PerChirpParamCtrl.
Refer Sensor Per-Chirp Configuration API and Sensor Per-Chirp Control API for more details.

Time Resolution Mode	Resolution Definition
High Resolution (c_ChirpTimerResol:b1=1)	$1LSB = 8/APLL\_FREQ$, unsigned number. Valid Range: 1 to 65535 (1.3ms)
Low Resolution (c_ChirpTimerResol:b1=0)	$1LSB = 40/APLL\_FREQ$, unsigned number. Valid Range: 1 to 65535 (6.5ms)

▶ h_ChirpAdcStartTime

Rx ADC start time from the knee of the ramp
This parameter configures the number of samples to be skipped at the start of the chirp. These samples are addition to the h_NumOfAdcSamples and should be considered while computing h_ChirpRampEndTime.
This value will be ignored if the bit M_RL_SENS_PER_CHIRP_PARAM_INDEX_TX_START_TIME is enabled in h_PerChirpParamCtrl. Refer Sensor Per-Chirp Configuration API and Sensor Per-Chirp Control API for more details.

Bit Field	Definition
Bits[9:0]	Reserved
Bits[15:10]	$AdcSkipSamples$ Number of ADC Samples to be Skipped. • 1 LSB= 1 ADC sample • Valid range: 0 to 63

▶ xh_ChirpTxStartTime

Tx start time from the knee of the ramp.
This value determines when the Tx transmission is enabled with respect to the knee of the ramp. Negative values could be used to start the Tx transmissions before the FMCW chirp ramp starts.
This value will be ignored if the bit M_RL_SENS_PER_CHIRP_PARAM_INDEX_TX_START_TIME is enabled in h_PerChirpParamCtrl. Refer Sensor Per-Chirp Configuration API and Sensor Per-Chirp Control API for more details.

- Format: 16-bit signed number with 1 LSB = 8/APLL_FREQ (Typical 20us with 40MHz XTAL)
- Recommended Range: -150 to 150 (+/- 3us)

Notes The c_ChirpTimerResol is **NOT** applicable for this parameter.

▶ xh_ChirpRfFreqSlope

Chirp frequency ramp slope.
This parameter controls the slope of the FMCW ramp. The signed slope value code and range depend on the operating frequency range of the device. This value will be ignored if the bit M_RL_SENS_PER_CHIRP_PARAM_INDEX_FREQ_SLOPE is enabled in h_PerChirpParamCtrl. Refer Sensor Per-Chirp Configuration API and Sensor Per-Chirp Control API for more details.

RF Frequency	Resolution
60GHz	$1LSB = 3 \times 400 \times APLL\_FREQ/2^{24}$ ≈28.610 kHz/us , signed number. Valid range: -13972 to 13972 (+/- 399MHz/us)
77GHz	$1LSB = 4 \times 400 \times APLL\_FREQ/2^{24}$ ≈38.147 kHz/us , signed number. Valid range: -13972 to 13972 (+/- 533MHz/us)

Warning

- Firmware does not check validity of the ramp end frequency of a chirp. Application should ensure that h_ChirpRampEndTime, w_ChirpRfFreqStart and xh_ChirpRfFreqSlope are configured without exceeding the operating frequency range of the device.
- Chirp slope will be forced to 0 internally in CW mode.

▶ w_ChirpRfFreqStart

Starting frequency of the FMCW chirp ramp.
This parameter controls the start frequency of the FMCW chirp or the wave transmitted in CW mode. The code value of the start frequency depends on the c_ChirpTimerResol in the FECSS boot and operating frequency of the device.
This value will be ignored if the bit M_RL_SENS_PER_CHIRP_PARAM_INDEX_FREQ_START is enabled in h_PerChirpParamCtrl. Refer Sensor Per-Chirp Configuration API and Sensor Per-Chirp Control API for more details.

RF Frequency	Encoding
60 GHz	**High Frequency Resolution** (c_ChirpTimerResol:b0=1) • $1LSB = (3 \times APLL\_FREQ/2^{24}) \times 2^6$ ≈4.578 kHz • Valid Range: 0x00BE0000 (57GHz) to 0x00D55555 (64GHz)* *Variant specific limitations in Release Notes • 24-bit unsigned format **Low Frequency Resolution** (c_ChirpTimerResol:b0=0) • $1LSB = (3 \times APLL\_FREQ/2^{16}) \times 2^6$ ≈1.172 MHz • Valid Range: 0x0000BE00 (57GHz) to 0x0000D555 (64GHz)* *Variant specific limitations in Release Notes • 16-bit unsigned format
77 GHz	**High Frequency Resolution** (c_ChirpTimerResol:b0=1) • $1LSB = (4 \times APLL\_FREQ/2^{24}) \times 2^6$ ≈6.103 kHz • Valid Range: 0x00BE0000 (76GHz) to 0x00CA8000 (81GHz) • 24-bit unsigned format **Low Frequency Resolution** (c_ChirpTimerResol:b0=0) • $1LSB = (4 \times APLL\_FREQ/2^{16}) \times 2^6$ ≈1.562 MHz • Valid Range: 0x0000BE00 (76GHz) to 0x0000CA80 (81GHz) • 16-bit unsigned format

▶ h_ChirpTxEnSel

Tx enable control bit mask with 1 bit per Tx.
This parameter controls which Tx channels are enabled during the transmission. This parameter will be ignored if either c_ChirpTxMimoPatSel=1 or M_RL_SENS_PER_CHIRP_PARAM_INDEX_TX_EN is enabled in h_PerChirpParamCtrl. The application should also ensure that all the required channels are also enabled in h_TxChCtrlBitMask.

Bit Field	Definition
bits [0]	Tx Channel 0 Enable Control
bits [1]	Tx Channel 1 Enable Control
bits [15:2]	Reserved

- `h_ChirpTxBpmEnSel`

Tx BPM enable control bit mask with 1 bit per Tx.
This parameter controls binary phase inversion for enabled Tx channels during the transmission. This parameter will be ignored if either `c_ChirpTxMimoPatSel`=4 or `M_RL_SENS_PER_CHIRP_PARAM_INDEX_BPM_EN` is enabled in `h_PerChirpParamCtrl`. The application should also ensure that all the required channels are also enabled in `h_TxChCtrlBitMask`.

Bit Field	Definition
bits [0]	Tx Channel 0 BPM Control
bits [1]	Tx Channel 1 BPM Control
bits [15:2]	Reserved

2.4.4 Sensor Chirp Profile Time Configuration Get API

This API reads the hardware registers and returns the configuration done using the Sensor Chirp Profile Time Configuration API.

API Definition

API Index	0x0043
API Function Name	`rl_sensChirpProfTimeCfgGet`
API Configuration Structure	None
API Response Structure	`T_RL_API_SENS_CHIRP_PROF_TIME_CFG`

- API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_SENS_PROF_TIME_CFG_GET        ((UINT16)0x0043U)
// API Function Definition
    T_RETURNTYPE rl_sensChirpProfTimeCfgGet(UINT8 c_devIndex,
    ↪  T_RL_API_SENS_CHIRP_PROF_TIME_CFG *p_apiResData);
```

2.4.5 Sensor Per-Chirp Configuration API

mmWave sensors provide option to dither/ vary selected set of parameters for every chirp. These parameters are stored in a dedicated memory section, referred as PER_CHIRP_RAM, in a look-up-table (LUT) format. Each of the per-chirp parameters' LUT is stored in the memory as an array of contiguous values in a specific format. Chirp timer hardware automatically updates the enabled parameters from the LUT corresponding to that parameter. mmWave Low devices support following per-chirp parameters which override the configurations from Sensor Chirp Profile Time Configuration API.

Table 2.38: Sensor Per-Chirp Parameters

Parameter	Description
Chirp Start Frequency M_RL_SENS_PER_CHIRP_PARAM_INDEX_FREQ_START (0)	LUT entries are stored as **32-bit unsigned** numbers similar to the w_ChirpRfFreqStart
Chirp Slope M_RL_SENS_PER_CHIRP_PARAM_INDEX_FREQ_SLOPE (1)	LUT entries are stored as **16-bit signed** numbers similar to the xh_ChirpRfFreqSlope
Chirp Idle Time M_RL_SENS_PER_CHIRP_PARAM_INDEX_IDLE_TIME (2)	LUT entries are stored as **16-bit unsigned** numbers similar to the h_ChirpIdleTime
ADC Start Time M_RL_SENS_PER_CHIRP_PARAM_INDEX_ADC_START_TIME (3)	LUT entries are stored as **16-bit unsigned** numbers similar to the h_ChirpAdcStartTime
Tx Start Time M_RL_SENS_PER_CHIRP_PARAM_INDEX_TX_START_TIME (4)	LUT entries are stored as **16-bit signed** numbers similar to the xh_ChirpTxStartTime
Tx Enable Mask M_RL_SENS_PER_CHIRP_PARAM_INDEX_TX_EN (5)	LUT entries are stored as **4-bit masks** with 1-bit per Tx channel, i.e. each byte contains 2 LUT entries with upper nibble representing the latter configuration.
Tx BPM Enable Mask M_RL_SENS_PER_CHIRP_PARAM_INDEX_BPM_EN (6)	LUT entries are stored as **4-bit masks** with 1-bit per Tx channel, i.e. each byte contains 2 LUT entries with upper nibble representing the latter configuration.

All the LUT arrays are stored in PER_CHIRP_RAM, which is a generic memory buffer accessible by the front end. The application can directly access the PER_CHIRP_RAM and update the LUTs on the boot or in frame idle time. Following table defines the various attributes of the PER_CHIRP_RAM.

Attribute	xWRL6432	xWRL1432
Start Address	0x21880000	0x21880000
Size	8192 Bytes	8192 Bytes
Address Offset Alignment for LUTs	4 Bytes	4 Bytes

This API should be used along with the Sensor Per-Chirp Control API to control and enable per-chirp configurations.

Warning

- Application must not modify the PER_CHIRP_RAM in active frame duration.
- If application requires dynamic configuration of the PER_CHIRP_RAM, it should split the buffer in two sections and use it in ping-pong manner.
- PER_CHIRP_RAM could be retained across deep sleep, but not the configuration. Application has to re-issue the per-chirp configuration command after waking up from the deep sleep.
- This API configurations will be ignored in CW Mode

API Definition

API Index	0x0044U
API Function Name	rl_sensPerChirpCfg
API Configuration Structure	T_RL_API_SENS_PER_CHIRP_CFG
API Response Structure	None

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_SENS_PER_CHIRP_CFG            ((UINT16)0x0044U)
// API Function Definition
    T_RETURNTYPE rl_sensPerChirpCfg(UINT8 c_devIndex, const T_RL_API_SENS_PER_CHIRP_CFG
    ↪  *p_apiCmdData);
```

Configuration C Structure

Table 2.39: T_RL_API_SENS_PER_CHIRP_CFG

Field	Data Type	Size	Offset
h_ParamArrayLen	UINT16[12]	24	0
h_ParamRptCount	UINT16[12]	24	24
w_Reserved1	UINT32	4	48
Total		**52**	

Configuration Fields

h_ParamArrayLen

Array of the Per-Chirp LUT lengths for all the supported per-chirp parameters.
Each entry in this array represents number of configurations (not memory size) for the corresponding per-chirp LUT parameter.
Valid Range: [1, 4096]

Byte-wise description is as follows:

Bytes	Description
Bytes[1:0]	Number of entries in Start Frequency Per-Chirp LUT
Bytes[3:2]	Number of entries in Chirp Slope Per-Chirp LUT
Bytes[5:4]	Number of entries in Chirp Idle Time Per-Chirp LUT
Bytes[7:6]	Number of entries in ADC Start Time Per-Chirp LUT
Bytes[9:8]	Number of entries in Tx Start Time Per-Chirp LUT
Bytes[11:10]	Number of entries in Tx Enable Per-Chirp LUT
Bytes[13:12]	Number of entries in Tx BPM Enable Per-Chirp LUT
Bytes[23:14]	Reserved

LUT length should be 0 for the disable per-chirp parameters.

- `h_ParamRptCount`

Array of repetition count for each entry in the per-chirp LUT for all the supported per-chirp parameters.
This parameter defines the number of time a particular entry in the LUT will be repeated before reading a new value from the buffer. This feature can be used to reduce the LUT size if same configuration is used for N consecutive chirps.

Valid Range: [1, 32768]

Byte-wise description is as follows:

Bytes	Description
Bytes[1:0]	Repetition count for Start Frequency Per-Chirp LUT
Bytes[3:2]	Repetition count for Chirp Slope Per-Chirp LUT
Bytes[5:4]	Repetition count for Chirp Idle Time Per-Chirp LUT
Bytes[7:6]	Repetition count for ADC Start Time Per-Chirp LUT
Bytes[9:8]	Repetition count for Tx Start Time Per-Chirp LUT
Bytes[11:10]	Repetition count for Tx Enable Per-Chirp LUT
Bytes[13:12]	Repetition count for Tx BPM Enable Per-Chirp LUT
Bytes[23:14]	Reserved

Repetition count should be 0 for the disable per-chirp parameters.

2.4.6 Sensor Per-Chirp Configuration Get API

This API reads the hardware registers and returns the per-chirp LUT configuration done using the Sensor Per-Chirp Configuration API.

API Definition

API Index	0x0045
API Function Name	rl_sensPerChirpCfgGet
API Configuration Structure	None
API Response Structure	T_RL_API_SENS_PER_CHIRP_CFG

▶ API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_SENS_PER_CHIRP_CFG_GET        ((UINT16)0x0045U)
// API Function Definition
    T_RETURNTYPE rl_sensPerChirpCfgGet(UINT8 c_devIndex, T_RL_API_SENS_PER_CHIRP_CFG
    ↪  *p_apiResData);
```

2.4.7 Sensor Per-Chirp Control API

This API defines which per-chirp parameters are controlled using the LUTs. Refer Sensor Per-Chirp Configuration API for more details about the per-chirp LUTs and configurations.

Warning

- Per-chirp configurations are directly written to the hardware registers and hence will not be retained across deep-sleep. Application must re-issue this API after waking up from deep sleep.
- This API configurations will be ignored in CW Mode

API Definition

API Index	
API Function Name	rl_sensPerChirpCtrl
API Configuration Structure	T_RL_API_SENS_PER_CHIRP_CTRL
API Response Structure	None

▶ API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_SENS_PER_CHIRP_CTRL           ((UINT16)0x0046U)
// API Function Definition
    T_RETURNTYPE rl_sensPerChirpCtrl(UINT8 c_devIndex, const T_RL_API_SENS_PER_CHIRP_CTRL
    ↪  *p_apiCmdData);
```

Configuration C Structure

Table 2.40: T_RL_API_SENS_PER_CHIRP_CTRL

Field	Data Type	Size	Offset
h_PerChirpParamCtrl	UINT16	2	0
h_Reserved1	UINT16	2	2
h_ParamArrayStartAdd	UINT16[12]	24	4
w_Reserved2	UINT32	4	28
w_Reserved3	UINT32	4	32
Total		**36**	

Configuration Fields

h_PerChirpParamCtrl

Per-chirp parameter enable control mask.
This is a bit-mask to select which per-chirps parameters should be read from the LUTs.

Bit Value	Description
0	Per-chirp parameter is disabled and the parameter value will be used from Configuration Fields
1	Per-chirp parameter is enabled and the parameter value will be picked from the PER_CHIRP_RAM based on configurations in Configuration Fields

Bitwise description of the field is as follows:

Bit	Description
Bit[0]	Enable control for Start Frequency Per-Chirp LUT
Bit[1]	Enable control for Chirp Slope Per-Chirp LUT
Bit[2]	Enable control for Chirp Idle Time Per-Chirp LUT
Bit[3]	Enable control for ADC Start Time Per-Chirp LUT
Bit[4]	Enable control for Tx Start Time Per-Chirp LUT
Bit[5]	Enable control for Tx Enable Per-Chirp LUT
Bit[6]	Enable control for Tx BPM Enable Per-Chirp LUT
Bits[15:7]	Reserved

- h_ParamArrayStartAdd

Array of LUT offset offsets for all supported per-chirp parameters.
Entries in this array defines the starting memory location of the per-chirp LUT of the corresponding parameter. These offsets must be multiple of 4 and defined with respect to the start of the PER_CHIRP_RAM.

Byte-wise description of the array is as follows:

Bytes	Description
Bytes[1:0]	Offset for Start Frequency Per-Chirp LUT
Bytes[3:2]	Offset for Chirp Slope Per-Chirp LUT
Bytes[5:4]	Offset for Chirp Idle Time Per-Chirp LUT
Bytes[7:6]	Offset for ADC Start Time Per-Chirp LUT
Bytes[9:8]	Offset for Tx Start Time Per-Chirp LUT
Bytes[11:10]	Offset for Tx Enable Per-Chirp LUT
Bytes[13:12]	Offset for Tx BPM Enable Per-Chirp LUT
Bytes[23:14]	Reserved

Offset count should be 0 for the disable per-chirp parameters.

2.4.8 Sensor Per-Chirp Control Get API

This API reads the hardware registers and returns the per-chirp LUT configuration done using the Sensor Per-Chirp Control API.

API Definition

API Index	0x0047
API Function Name	rl_sensPerChirpCtrlGet
API Configuration Structure	None
API Response Structure	T_RL_API_SENS_PER_CHIRP_CTRL

- API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_SENS_PER_CHIRP_CTRL_GET       ((UINT16)0x0047U)
// API Function Definition
    T_RETURNTYPE rl_sensPerChirpCtrlGet(UINT8 c_devIndex, T_RL_API_SENS_PER_CHIRP_CTRL
    ↪  *p_apiResData);
```

2.4.9 Sensor Frame Configuration API

Frame configuration defines how the chirps are grouped in a transmission. The API configures parameters like number of chirps, burst and frames and timings. Please refer the important terms and descriptions in the mmWave Sensor APIs for more details.

Warning Profile configurations are directly written to the hardware registers and hence will not be retained across deep-sleep. Application must re-issue this API after waking up from deep sleep.

API Definition

API Index	0x0048
API Function Name	rl_sensFrameCfg
API Configuration Structure	T_RL_API_SENS_FRAME_CFG
API Response Structure	None

▶ API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_SENS_FRAME_CFG                ((UINT16)0x0048U)
// API Function Definition
    T_RETURNTYPE rl_sensFrameCfg(UINT8 c_devIndex, const T_RL_API_SENS_FRAME_CFG
    ↪  *p_apiCmdData);
```

- Configuration C Structure

Table 2.41: T_RL_API_SENS_FRAME_CFG

Field	Data Type	Size	Offset
h_NumOfChirpsInBurst	UINT16	2	0
c_NumOfChirpsAccum	UINT8	1	2
c_MiscSetupMask	UINT8	1	3
w_BurstPeriodicity	UINT32	4	4
h_NumOfBurstsInFrame	UINT16	2	8
h_Reserved1	UINT16	2	10
w_FramePeriodicity	UINT32	4	12
h_NumOfFrames	UINT16	2	16
h_Reserved2	UINT16	2	18
w_FrameEvent0TimeCfg	UINT32	4	20
w_FrameEvent1TimeCfg	UINT32	4	24
w_Reserved3	UINT32	4	28
Total		**32**	

Configuration Fields

- h_NumOfChirpsInBurst

Number of chirps in a burst.
Valid Range: [1, 65535]

- c_NumOfChirpsAccum

Number of chirp accumulations per chirp.
mmWave Low devices provide hardware level time domain accumulation for the chirps. When this feature is enabled, the hardware internally triggers multiple chirps with identical configurations and sums the corresponding ADC samples before storing them to ADC buffer. This feature can be used to increase the SNR without increasing the number of chirps to be processed.
Valid values for the chirp accumulation are as follows:

Value	Description
0 (Default)	Chirp accumulation is disabled
[2,64]	Chirp accumulation is enabled The hardware is internally trigger multiple chirps, same as this value, and accumulate the ADC data
1 (INVALID)	Invalid Value

Warning Actual number of chirps in a burst is `h_NumOfChirpsInBurst` $\times max(1,$ `c_NumOfChirpsAccum`$)$.

- `c_MiscSetupMask`

Miscellaneous Setup Mask
This mask controls the following miscellaneous features:

Bit	Description
Bit [0]	**Skip Frame Timer Configuration** • Value 0 (Default): Configure both Frame Timer and Chirp Timer • Value 1: Configure only Chirp Timer and skip Frame Timer Use Case: Frame configuration API configures burst related information to Chirp Timer, which is responsible for triggering bursts and chirps. This peripheral is associated with FECSS Sub-system. All the frame related parameters are passed to a separate peripheral called Frame Timer which is associated with AppSS. Since frame timer runs on the XTAL, it can run in background even when FECSS and AppSS are powered down in inter-framer duration. Chirp timer (CT) runs on the APLL clock and hence loses all the configurations ones FECSS is powered off. CT must be reconfigured after exit from sleep. This option should be enabled only to configure CT while FT is still running
Bits [7:1]	Reserved

- `w_BurstPeriodicity`

Burst periodicity count.
This parameter configures the 24-bit counter which generates the Burst Start event. Tick duration of this counter depends on the time resolution configured in FECSS Device Power On API.

Timer Resolution	Burst Periodicity Definition
High Resolution (`c_ChirpTimerResol`:b1=1)	$1LSB = 8/APLL\_FREQ$, (20ns With 40 MHz XTAL). Valid Range: [10, 16777215] ([0.2 us, 335 ms] for 40 MHz XTAL)
Low Resolution (`c_ChirpTimerResol`:b1=0)	$1LSB = 40/APLL\_FREQ$, (100ns With 40 MHz XTAL). Valid Range: [10, 16777215] ([1 us, 1677 ms] for 40 MHz XTAL)

- `h_NumOfBurstsInFrame`

Number of bursts in a frame.
Valid Range: [1, 4096]

- `w_FramePeriodicity`

Frame periodicity count.
This parameters configures the 32-bit counter running on XTAL which generates the Frame Start event.

XTAL_FREQ	1 LSB of Frame Periodicity Timer
25.0 MHz	40 ns
26.0 MHz	38.461 ns
38.4 MHz	26.042 ns
40.0 MHz	25 ns

Valid Range: $[100, 2^{32})$

Note This parameter is not applicable for hardware triggered frames

- `h_NumOfFrames`

Number of frames in a trigger.
Valid Range: [0, 65535]
Setting number of frames to 0 results in infinite frames.

- `w_FrameEvent0TimeCfg`

Frame Timer Event 0 Interrupt Time Period.
Frame timer hardware modules two user configurable events generated periodically with respect to the Frame Start event. These events can be used to schedule tasks with respect to the Frame Start of current frame, e.g. waking up FECSS from sleep just before the next frame starts and re-configure it.
These event counters, just like Frame Start run on XTAL. The application should disable these interrupts in AppSS core if they are not being used.
Valid Range: $[100, 2^{32})$

- `w_FrameEvent1TimeCfg`

Frame Timer Event 1 Interrupt Time Period.
Please refer `w_FrameEvent0TimeCfg` for detailed explanation.

Calculating Burst and Frame Period

Following table describes the recommended method to calculate burst and frame periods.

Note All the time variables used in the following table represent absolute time values in us and should be scaled appropriately.
e.g. `h_ChirpRampEndTime` represents the Ramp End Time in us, the actual variable value which is passed to the API depends on timer resolution.

Variable	Calculation Steps
Total Number of Chirps in a Burst $TotalNumChirps$	h_NumOfChirpsInBurst $\times max(1,$ c_NumOfChirpsAccum$)$
Burst Idle Time $BusrtIdleTime$	Note: This time should be grater then the $MinBurstIdleTime$
Burst Periodicity w_BurstPperiodicity	$TotalNumChirps \times$ (h_ChirpRampEndTime + h_ChirpIdleTime) + $BurstIdleTime$
Frame Idle Time $FrameIdleTime$	Note: This time should be grater then the $MinFrameIdleTime$
Frame Period w_FramePeriodicity	(h_NumOfBurstsInFrame $\times$ w_BurstPeriodicity) + $FrameIdleTime$

2.4.10 Sensor Frame Configuration Get API

This API reads the hardware registers and returns the frame configuration done using the Sensor Frame Configuration API.

API Definition

API Index	0x0049
API Function Name	rl_sensFrameCfgGet
API Configuration Structure	None
API Response Structure	T_RL_API_SENS_FRAME_CFG

▶ API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_SENS_FRAME_CFG_GET            ((UINT16)0x0049U)
// API Function Definition
    T_RETURNTYPE rl_sensFrameCfgGet(UINT8 c_devIndex, T_RL_API_SENS_FRAME_CFG *p_apiCmdData);
```

2.4.11 Sensor Start API

This API controls the frame trigger logic and associated configurations. The mmWave Low devices support three categories of trigger modes:

- Software Trigger Mode: Frames are triggered by the API call itself.
- Hardware Trigger Mode: Frame Timer waits for a rising edge on a DIG_SYNC_IN input pin to start the frames. This feature can be used to synchronize frames with external host. The mode h_NumOfFrames

is ignored and the hardware trigger only one frame at a time and the application software has to re-issue the command.

- CW Mode: In this mode enables the transmission in CW mode.

Warning Application should not issue this API while frames are running. Sensor Status Get API should be used to determine the current state.

Further details are mentioned in the API parameters.

API Definition

API Index	0x004A
API Function Name	rl_sensSensorStart
API Configuration Structure	T_RL_API_SENSOR_START_CMD
API Response Structure	None

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_SENS_START                    ((UINT16)0x004AU)
// API Function Definition
    T_RETURNTYPE rl_sensSensorStart(UINT8 c_devIndex, const T_RL_API_SENSOR_START_CMD
    ↪  *p_apiCmdData);
```

Configuration C Structure

Table 2.43: T_RL_API_SENSOR_START_CMD

Field	Data Type	Size	Offset
c_FrameTrigMode	UINT8	1	0
c_ChirpStartSigLbEn	UINT8	1	1
c_FrameLivMonEn	UINT8	1	2
c_Reserved	UINT8	1	3
w_FrameTrigTimerVal	UINT32	4	4
w_Reserved1	UINT32	4	8
Total		**12**	

Configuration Fields

- `c_FrameTrigMode`

Frame Trigger Mode Control.

The mmWave Low devices support following types of trigger mode:

Trigger Mode	mmWaveLink Macro	Value
Software Immediate Trigger Mode	`M_RL_SENS_FRAME_SW_TRIG`	0
Software Timer Trigger Mode	`M_RL_SENS_FRAME_SW_TIMER_TRIG`	1
Hardware Triggered Mode	`M_RL_SENS_FRAME_HW_LOW_PWR_TRIG`	2
Hardware Triggered Low Jitter Mode	`M_RL_SENS_FRAME_HW_LOW_JIT_TRIG`	3
CW Trigger Mode	`M_RL_SENS_FRAME_CW_CZ_TRIG`	4
Chirp Timer Override Mode	`M_RL_SENS_FRAME_CT_OVRD_TRIG`	5

Software Immediate Trigger Mode

In this mode, Frame Start event is generated immediately after receiving the API. Frame Trimer starts the Chirp Timer after RFS completes the first Frame Start event ($FrameStartTime$). First chirp starts after the RFS completes the first Burst Start event ($BurstStartTime$).

Software Timer Trigger Mode

This mode is almost same as the software trigger mode, except for the additional delay in the first Frame Start event. The Frame Timer waits for the internal free running counter to reach the value defined by `w_FrameTrigTimerVal` before generating the first Frame Start. This provides a controlled way to delay frame trigger. Subsequent Frame Starts are generated according to the `w_FramePeriodicity`.

Hardware Triggered Mode

In Hardware triggered mode, Frame Timer waits for a rising edge on the `DIG_SYNC_IN` input pin and then generates the Frame Start event. After that the burst and chirps are scheduled similar to the software triggered mode.

Low power mode samples the rising edge on the XTAL clock resulting into higher frame start jitter.

Warnings

- Application should ensure that the sensor start API is issued at least 10us before the rising edge on the `DIG_SYNC_IN` input pin.
- `w_FramePeriodicity` and `h_NumOfFrames` are not applicable in this mode.
- FECSS and APLL clocks must be powered on and configured before calling the sensor start API.

Hardware Triggered Low Jitter Mode

This mode is similar to the Hardware Triggered mode, except that the high frequency APLL clock is used to sample the `DIG_SYNC_IN` input pin. Higher sampling clock reduces frame start jitter.

Warnings

- Application should ensure that the sensor start API is issued at least 35us before the rising edge on the DIG_SYNC_IN input pin.
- w_FramePeriodicity and h_NumOfFrames are not applicable in this mode.
- FECSS and APLL clocks must be powered on and configured before calling the sensor start API.

CW Trigger Mode

This mode enables the CW mode transmission at a constant frequency defined by w_ChirpRfFreqStart with 0 slope. Sensor Stop API must be issued to stop the transmission.

Warnings

- This mode should be used only for debugging and testing purpose.
- Minimum 100us setup time is required before the transmission begins.

Chirp Timer Override Mode

This is an internal test mode in which a single frame triggered directly using the Chirp Timer. Application should not use this mode.

► c_ChirpStartSigLbEn

This filed is reserved in Single Chirp mode.

► c_FrameLivMonEn

Live monitor enable control.
This field controls which live monitors are enabled during the frame trigger. The mask contains 1-bit per live monitor, setting the corresponding bit high in the mask will enable the monitor. mmWave Low devices support following live monitors:

Bit	Description
Bit[0]	**Synthesizer Frequency Live Monitor** The monitor must be configured using FECSS Synthesizer Frequency Live Monitor Configuration API before calling sensor start mmWaveLink Macro: M_RL_SENS_LIVE_MON_SYNTH_FREQ
Bit[1]	**Rx ADC Saturation Live Monitor** The monitor must be configured using FECSS Rx Saturation Live Monitor Configuration API before calling sensor start. mmWaveLink Macro: M_RL_SENS_LIVE_MON_RX_SATURATION
Bit[7:2]	Reserved

► w_FrameTrigTimerVal

Frame trigger delay timer value.
This value is applicable only in Software Timer Trigger Mode. The Frame Timer hardware contains a reference free running 32-bit up-counter which runs on XTAL clock. The counter starts at value 0 at device power-up and rolls over after counting till $2^{32} - 1$. In timer trigger mode, Frame Timer waits for the internal reference

counter to reach the value specified in `w_FrameTrigTimerVal` before issuing first Frame Start. Current value of the reference counter can be read using Sensor Status Get API.
Valid Range: $[0, 2^{32})$

2.4.12 Sensor Stop API

Sensor stop API stops the device transmission by sending stop signals to the Frame Timer and Chirp Timer modules. The application should issue this API only after verifying the status of the Sensor using Sensor Status Get API.

API Definition

API Index	0x004B
API Function Name	`rl_sensSensorStop`
API Configuration Structure	`T_RL_API_SENSOR_STOP_CMD`
API Response Structure	None

▶ API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_SENS_STOP                    ((UINT16)0x004BU)
// API Function Definition
    T_RETURNTYPE rl_sensSensorStop(UINT8 c_devIndex, const T_RL_API_SENSOR_STOP_CMD
    ↪  *p_apiCmdData);
```

▶ Configuration C Structure

Table 2.44: `T_RL_API_SENSOR_STOP_CMD`

Field	Data Type	Size	Offset
`c_FrameStopMode`	`UINT8`	1	0
`c_Reserved`	`UINT8`	1	1
`h_Reserved1`	`UINT16`	2	2
`w_Reserved2`	`UINT32`	4	4
Total		**8**	

Configuration Fields

▶ `c_FrameStopMode`

mmWave Low devices support stopping frames in following modes:

Table 2.45: Frame Stop Modes

Value	mmWaveLink Macro	Description
0	M_RL_SENS_STOP_FRAME	**Stop at Frame Boundary** This modes stops the transmission after completion of currently on-going frame. Essentially it stops the Frame Timer from generating more Frame Start events. Application can use this mode to stop the frames early or interrupt infinite frame transmission
1	M_RL_SENS_STOP_CWCZ_MODE	**Stop CW Mode** This mode is used to stop the CW mode transfer.
2	M_RL_SENS_FORCE_STOP_FRAME	**Forced Stop** This is a error-recovery mode which immediately resets the Frame Timer and Chirp Timer. Chirp and Frame events may not be generated in this mode.

Warning

- Application should use the Sensor Status Get API API to verify after stopping the frames in M_RL_SENS_STOP_FRAME to verify the frames have stopped. The application may have to wait for one complete frame period before getting frame stop.
- Unnecessary Sensor Stop API should be avoided. The Frame Timer latches Frame Stop command in hardware registers which takes affect at the end of current frame. Frame Stop API issued when Frame Timer has already stopped will result in un-intended stop in the next frame trigger because of the latched stop bit.
- Application should wait for FECSS to complete Burst End ($BurstEndTime$) and Frame End ($FrameEndTime$) activities after receiving the Frame Stop conformation in any mode.

Most reliable method to stop the frames without going in deep sleep is as follows:

1. Stop the frames using M_RL_SENS_STOP_FRAME mode.
2. Wait for 1 frame period or poll for the status using Sensor Status Get API.
3. Reset the Frame Timer and Burst Timer IP forcefully using M_RL_SENS_FORCE_STOP_FRAME mode.

The last step ensures that the hardware registers are free of any latched signals which could impact the next transmission. The last step could be ignored if the device enters in a sleep before next transmission (As power cycle will automatically reset the hardware registers).

2.4.13 Sensor Status Get API

This API returns current values of various sensor state parameters such as frame count, chirp count, etc. The API could also be used to get time stamp from the Frame Timer reference counter. It is recommended to read the status using this API before issuing Sensor Start API.

API Definition

API Index	0x004C
API Function Name	rl_sensStatusGet
API Configuration Structure	None
API Response Structure	T_RL_API_SENSOR_STATUS_RSP

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_SENS_STATUS_GET                ((UINT16)0x004CU)
// API Function Definition
    T_RETURNTYPE rl_sensStatusGet(UINT8 c_devIndex, T_RL_API_SENSOR_STATUS_RSP
    ↪  *p_apiResData);
```

Response C Structure

Table 2.46: T_RL_API_SENSOR_STATUS_RSP

Field	Data Type	Size	Offset
w_ChirpCount	UINT32	4	0
h_BurstCount	UINT16	2	4
h_FrameCount	UINT16	2	6
w_FrameRefTimerVal	UINT32	4	8
w_BurstPeriodTimerVal	UINT32	4	12
w_FramePeriodTimerVal	UINT32	4	16
c_FrameStartStopStatus	UINT8	1	20
c_Reserved1	UINT8	1	21
h_Reserved2	UINT16	2	22
w_Reserved3	UINT32[2]	8	24
Total		**32**	

Response Fields

w_ChirpCount

Number of Chirps Triggered.
This value indicates the number of chirps triggered in the current burst so far. The value starts at 0 for the first chirp and counts up to h_NumOfChirpsInBurst -1 for the current burst.
Valid Range: $[0, 65534]$

h_BurstCount

Number of Bursts Triggered.
This value indicates the number of bursts triggered in the current frame so far. The value starts at 0 for the first burst and counts up to h_NumOfBurstsInFrame -1 for the current frame.
Valid Range: $[0, 4095]$

- h_FrameCount

Number of Frames Triggered.
This value indicates the number of frames triggered in the current trigger so far. The value starts at 0 for the first frame and counts up to h_NumOfFrames -1 for the current trigger.
In case of infinite frames, the counter will roll over after reaching the maximum value.
Valid Range: $[0, 65534]$

- w_FrameRefTimerVal

Frame Timer Reference Counter Value.
This value indicates the current count of the free running 32-bit up-counter present in the Frame Timer hardware module which runs on XTAL clock. The counter starts at value 0 at device power-up and rolls over after counting till $2^{32} - 1$.

XTAL_FREQ	1 LSB of the Counter
25.0 MHz	40 ns
26.0 MHz	38.461 ns
38.4 MHz	26.042 ns
40.0 MHz	25 ns

- w_BurstPeriodTimerVal

Remaining Burst Period Timer Value.
The Chirp Timer hardware contains a decrementing counter to track the burst period. This counter runs on a APLL derived clock which runs at $APLL/4$ (same as Rx ADC). The counter loads the start value according to the following table upon receiving Burst Start event and counts down to 0. Running value of this field indicates remaining time for the current burst (including idle time). If applicable, new Burst Start event will be generated one the counter reaches zero indicating start of a new burst.

Resolution: $1LSB = 4/APLL\_FREQ$

Time Resolution	Starting Value of the Counter
High Resolution (c_ChirpTimerResol:b1=1)	$(2 \times$ w_BurstPeriodicity$) - 1$
Low Resolution (c_ChirpTimerResol:b1=0)	$(10 \times$ w_BurstPeriodicity$) - 1$

- w_FramePeriodTimerVal

Remaining Frame Period Timer Value.
The Frame Timer hardware contains a decrementing counter to track the frame period. This counter runs on a XTAL clock. The counter loads the counter with w_FramePeriodicity -1 upon receiving Frame Start event and

counts down to 0. Running value of this field indicates remaining time for the current frame (including idle time). If applicable, new Frame Start event will be generated one the counter reaches zero indicating start of a new burst.

- `c_FrameStartStopStatus`

Frame Start and Stop Status.
These bits indicate if the frame start and stop commands are honored by the hardware modules. These bits are cleared in Sensor Start API before starting Frame Timer.

Bit	Description
Bit [0]	**Frame Start Status** High value of this bit indicates that the Frame Timer has started after receiving frame Sensor Start API in Software Immediate Trigger Mode.
Bit [1]	**Frame Stop Status** High value of this indicates that the frames have been stopped after receiving the Sensor Stop API.
Bits [7:2]	Reserved

- Detecting Burstand Frame Boundaries

There are two methods to detect burst and frame boundaries

- Polling Method: Application can poll on the Sensor Status Get API to determine current state
- Interrupt Method: The Application can enable following interrupts to detect required events.
  - Burst Start or Burst End Interrupt `IRQ_31`
  - Frame End Interrupt `IRQ_32`
  - Frame Start Interrupt `IRQ_33`

  Note: Burst Start and Burst End interrupts are Muxed. Please refer Technical Reference manual for more details

Note The RFS firmware also keep track of the burst and frame events and updates the count in shared memory. These counts can be accessed directly using mmWaveLink macro `M_FE_RFS_SENS_STATUS_START_ADDRESS`. These counts also include the calibration and monitoring events and hence should be used only for debug.

2.4.14 Sensor Dynamic Power Save Disable Configuration API

This API configures the dynamic power save disable settings. By default the Chirp Timer module turns off high power consuming modules in the inter-chirp idle time to save power. This API provides interface to override that feature.

mmWave Low devices do not provide option to disable inter-burst and inter-frame power saving, these features are enabled by default maximizing the power savings.

API Definition

API Index	0x004D
API Function Name	rl_sensDynPwrSaveDis
API Configuration Structure	T_RL_API_SENS_DYN_PWR_SAVE_DIS
API Response Structure	None

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_SENS_DYN_PS_DIS                ((UINT16)0x004DU)
// API Function Definition
    T_RETURNTYPE rl_sensDynPwrSaveDis(UINT8 c_devIndex, const T_RL_API_SENS_DYN_PWR_SAVE_DIS
    ↪  *p_apiCmdData);
```

Configuration C Structure

Table 2.47: T_RL_API_SENS_DYN_PWR_SAVE_DIS

Field	Data Type	Size	Offset
c_InterChirpPsDis	UINT8	1	0
c_Reserved	UINT8	1	1
h_Reserved	UINT16	2	2
w_Reserved1	UINT32	4	4
Total		**8**	

Configuration Fields

c_InterChirpPsDis

Inter-Chirp Dynamic Power Save Disable Mask.
Each bit in the mask represents a rapid enable signal controlled by the Chirp Timer. Setting the corresponding bit High will disable power saving associated with that signal. By default, all signals are enabled (Value: 0).

Bit	Description
Bit [0]	Tx PA Power Save Disable
Bit [1]	Tx LO Power Save Disable
Bit [2]	Rx RF Power Save Disable
Bit [3]	Rx BaseBand Power Save Disable
Bit [4]	Rx LO Power Save Disable
Bits [7:5]	Reserved

Recommendation Following table shows the recommended value of the `c_InterChirpPsDis` across use cases. Please note that the variables used in the following table represent absolute time values in us and should be scaled appropriately.

Use Case	`c_InterChirpPsDis`
IF ($h\_ChirpIdleTime >= max(6us - xh\_ChirpTxStartTime, 3.1us)$)	0x00 Power Save Enabled
ELSEIF ($h\_ChirpIdleTime >= max(4us - xh\_ChirpTxStartTime, 3.1us)$)	0x1C Rx Power Save Disabled Tx Power Save Enabled
ELSE	Unsupported Idle Time

Please refer Burst Timing Constraints for more details.

2.4.15 Sensor Dynamic Power Save Disable Configuration Get API

This API reads the hardware registers and returns the frame configuration done using the Sensor Dynamic Power Save Disable Configuration API.

API Definition

API Index	0x004E
API Function Name	`rl_sensDynPwrSaveStsGet`
API Configuration Structure	None
API Response Structure	`T_RL_API_SENS_DYN_PWR_SAVE_DIS`

▶ API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_SENS_DYN_PS_DIS_GET          ((UINT16)0x004EU)
// API Function Definition
    T_RETURNTYPE rl_sensDynPwrSaveStsGet(UINT8 c_devIndex, T_RL_API_SENS_DYN_PWR_SAVE_DIS
    ↪  *p_apiResData);
```

2.4.16 Sensor Loopback Configuration API

This API configures the parameters required for various analog and digital loopbacks in the mmWave Low devices. This configuration is retained across deep sleep in the FECSS memory. Hence application can avoid reconfiguring it across deep sleep.

[Gray]Note Digital loopback uses a full-scale amplitude square wave signal generated from the loopback signal generator. This signal has inherent delay of 80ns and 180ns for Rx1 and Rx2 with respect to the Rx0, which results into phase shift across Rx channels.

API Definition

API Index	0x004F
API Function Name	rl_sensLoopBackCfg
API Configuration Structure	T_RL_API_SENS_LOOPBACK_CFG
API Response Structure	None

▶ API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_SENS_LB_CFG                    ((UINT16)0x004FU)
// API Function Definition
    T_RETURNTYPE rl_sensLoopBackCfg(UINT8 c_devIndex, const T_RL_API_SENS_LOOPBACK_CFG
    ↪  *p_apiCmdData);
```

▸ Configuration C Structure

Table 2.48: T_RL_API_SENS_LOOPBACK_CFG

Field	Data Type	Size	Offset
c_LbFreqSel	UINT8	1	0
c_IfaLbGainIndex	UINT8	1	1
c_LoPaLbCmnGainIndex	UINT8	1	2
c_LoLbGainIndex	UINT8	1	3
c_PaLbGainIndex	UINT8	1	4
c_Reserved1	UINT8	1	5
h_ExtLbTxBpmEnSel	UINT16	2	6
w_Reserved3	UINT32[2]	8	8
Total		**16**	

Configuration Fields

- `c_LbFreqSel`

Loopback Signal Frequency Select.
This parameter controls the frequency of the loopback signal in IF domain across all loopbacks. The signal is derived from APLL clock according the following formula:

$$f_{LoopBack} = \frac{APLL\_FREQ}{16 \times \texttt{c\_LbFreqSel}}$$

Valid Range: $[2, 200]$
Following table shows the recommended loopback frequencies:

`c_LbFreqSel`	**Loopback Frequency**
2	12.5 MHz
4	6.25 MHz
5	5 MHz
8	3.125 MHz
9	2.778 MHz
10	2.5 MHz
20	1.25 MHz
25	1 MHz
40	625 kHz
50	500 kHz
80	312.5 kHz
100	250 kHz
200	125 kHz

- `c_IfaLbGainIndex`

IFA Loopback Gain Index.
This parameter selects the IFA loopback gain and is applied only if IFA loopback is enabled using `c_LbEnable`.

Following table lists the recommended values of the gain index for various gains.

IFA Gain (dB)	c_IfaLbGainIndex	Low Bias Enable c_IfaLbGainIndex[7]	Loopback DAC Code c_IfaLbGainIndex[6:0]
-10	40	0	40
-8	191	1	63
-6	178	1	50
-4	167	1	39
-2	159	1	31
0	153	1	25
2	148	1	20
4	144	1	16
6	140	1	12
8	138	1	10
10	136	1	8

- c_LoPaLbCmnGainIndex

LO and PA Loopback Common Loopback Buffer Gain Index.
This parameter determines the gain of the common buffer in PA and LO loopback paths. This filed is applicable for both PA and LO loopbacks. Following table shows the buffer gain in dB for nominal process corner and temperature.

Common Loopback Gain (dB)	c_LoPaLbCmnGainIndex
0	48
-1	29
-2	25
-3	24
-5	23
-6	22
-7	21
-8	20
-10	19
-11	18
-13	17

Common Loopback Gain (dB)	c_LoPaLbCmnGainIndex
-14	16
-18	15
-19	0

- c_LoLbGainIndex

LO Loopback Buffer Gain Index.
This parameter controls the gain of the LO loopback buffer which follows the common buffer. The parameter is applicable only in LO loopback mode. Following table shows the buffer gain in dB for nominal process corner and temperature.

LO Loopback Gain (dB)	c_LoLbGainIndex
0	96
-1	51
-2	47
-3	44
-4	42
-5	40
-6	38
-7	36
-8	35
-9	34
-10	33
-11	32
-12	31
-13	30
-14	29
-15	28
-16	27
-17	26
-18	25

LO Loopback Gain (dB)	c_LoLbGainIndex
-19	24
-21	23
-22	22
-23	21
-25	20
-26	19
-27	18
-30	0

- c_PaLbGainIndex

PA Loopback Buffer Gain Index.
This parameter controls the gain of the LO loopback buffer which follows the common buffer. The parameter is applicable only in PA loopback mode. Following table shows the buffer gain in dB for nominal process corner and temperature.

PA Loopback Gain (dB)	c_PaLbGainIndex
0	48
-1	27
-2	25
-3	24
-4	23
-5	22
-6	21
-7	20
-10	19
-11	18
-13	17
-15	16
-17	15
-18	14

PA Loopback Gain (dB)	c_PaLbGainIndex
-19	0

- h_ExtLbTxBpmEnSel

External Waveguide Loopback Tx BPM Enable Mask.
This bit-mask controls channels on which Tx channels loopback signals drives the BPM bit. This parameter is applicable only when external waveguide loopback is enabled using c_LbEnable ans sensor is started in CW mode.
Each bit controls BPM enable on corresponding Tx channel.

Bit	Description
Bit [0]	Enables CW Mode BPM on Tx0 Channel
Bit [1]	Enables CW Mode BPM on Tx1 Channel
Bits [7:2]	Reserved

2.4.17 Sensor Loopback Enable API

This API enables a selected loopback. The API assumes that the loopback settings are already configured using the Sensor Loopback Configuration API.

Warning

- Loopback enable settings are not retained across FECSS power cycles. Hence loopback must be disabled before FECSS Device Power OFF API.
- Loopback should be disabled before running all calibrations and monitors as well.

API Definition

API Index	0x0050
API Function Name	rl_sensLoopBackEna
API Configuration Structure	T_RL_API_SENS_LOOPBACK_ENA
API Response Structure	None

- API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_SENS_LB_ENA                    ((UINT16)0x0050U)
// API Function Definition
    T_RETURNTYPE rl_sensLoopBackEna(UINT8 c_devIndex, const T_RL_API_SENS_LOOPBACK_ENA
    ↪  *p_apiCmdData);
```

▸ Configuration C Structure

Table 2.54: T_RL_API_SENS_LOOPBACK_ENA

Field	Data Type	Size	Offset
c_LbEnable	UINT8	1	0
c_Reserved1	UINT8	1	1
h_Reserved2	UINT16	2	2
w_Reserved3	UINT32	4	4
Total		**8**	

Configuration Fields

▶ c_LbEnable

Loopback Enable Control.
This parameter selects the loopback mode which should be applied in the front end.

c_LbEnable	mmWaveLink Macro	Description
0	M_RL_SENS_LOOPBACK_DISABLE	Disable All Loopbacks (Default)
1	M_RL_SENS_LOOPBACK_DIG	Enable Digital Loopback
2	M_RL_SENS_LOOPBACK_IFA	Enable IFA Loopback
3	M_RL_SENS_LOOPBACK_LO	Enable LO Loopback
4	M_RL_SENS_LOOPBACK_PA	Enable PA Loopback
5	M_RL_SENS_LOOPBACK_EXTERNAL_WG	Enable External Waveguide Loopback

2.5 mmWaveLink Monitor Configuration APIs

mmWaveLink Monitor APIs provide interface to configure and run the various monitoring features supported by the mmWave Radar devices.

Note Only selected set of monitors are enabled on non-functional safety enabled devices. Please refer `w_MonitorEnable` for more details.

The mmWaveLink Monitor module provides following APIs:

- FECSS Monitor Enable Trigger API
- FECSS Peak-Detector Debug Monitor API
- FECSS Tx Power Debug Monitor API
- FECSS Synthesizer Frequency Live Monitor Configuration API
- FECSS Rx Saturation Live Monitor Configuration API
- FECSS PLL Control Voltage Monitor Configuration API
- FECSS TxN-Rx Loopback Monitor Configuration API
- FECSS TxN Power Monitor Configuration API
- FECSS Tx Ball-Break Monitor Configuration API
- FECSS TxN DC Signal Monitor Configuration API
- FECSS Rx HPF and DC Signal Monitor Configuration API
- FECSS PM and Clock DC Signals Monitor Configuration API

Firmware assigns a unique monitor index to all the monitors. All monitor trigger and status fields use the same indices to map the monitors. mmWaveLink monitoring indices are as follows:

Table 2.55: mmWaveLink Monitoring Indices

	mmWaveLink Macro	Definition
0	`M_RL_MON_PLL_CTRL_VOLT`	PLL Control Voltage Monitor
1	`M_RL_MON_TX0_RX_LB`	Tx0 Rx-Tx Loopback Monitor
2	`M_RL_MON_TX1_RX_LB`	Tx1 Rx Loopback Monitor
3	`M_RL_MON_TX2_RX_LB`	Tx2 Rx Loopback Monitor
4	`M_RL_MON_TX3_RX_LB`	Tx3 Rx Loopback Monitor
5	`M_RL_MON_TX4_RX_LB`	Tx4 Rx Loopback Monitor
6	`M_RL_MON_TX5_RX_LB`	Tx5 Rx Loopback Monitor
7	`M_RL_MON_TX6_RX_LB`	Tx6 Rx Loopback Monitor
8	`M_RL_MON_TX7_RX_LB`	Tx7 Rx Loopback Monitor
9	`M_RL_MON_TX0_POWER`	Tx0 Power Monitor

	mmWaveLink Macro	Definition
10	M_RL_MON_TX1_POWER	Tx1 Power Monitor
11	M_RL_MON_TX2_POWER	Tx2 Power Monitor
12	M_RL_MON_TX3_POWER	Tx3 Power Monitor
13	M_RL_MON_TX4_POWER	Tx4 Power Monitor
14	M_RL_MON_TX5_POWER	Tx5 Power Monitor
15	M_RL_MON_TX6_POWER	Tx6 Power Monitor
16	M_RL_MON_TX7_POWER	Tx7 Power Monitor
17	M_RL_MON_TX0_BB	Tx0 Ball-Break Monitor
18	M_RL_MON_TX1_BB	Tx1 Ball-Break Monitor
19	M_RL_MON_TX2_BB	Tx2 Ball-Break Monitor
20	M_RL_MON_TX3_BB	Tx3 Ball-Break Monitor
21	M_RL_MON_TX4_BB	Tx4 Ball-Break Monitor
22	M_RL_MON_TX5_BB	Tx5 Ball-Break Monitor
23	M_RL_MON_TX6_BB	Tx6 Ball-Break Monitor
24	M_RL_MON_TX7_BB	Tx7 Ball-Break Monitor
25	M_RL_MON_TX0_INTRNAL_DC_SIG	Tx0 Internal DC Signal Monitor
26	M_RL_MON_TX1_INTRNAL_DC_SIG	Tx1 Internal DC Signal Monitor
27	M_RL_MON_TX2_INTRNAL_DC_SIG	Tx2 Internal DC Signal Monitor
28	M_RL_MON_TX3_INTRNAL_DC_SIG	Tx3 Internal DC Signal Monitor
29	M_RL_MON_TX4_INTRNAL_DC_SIG	Tx4 Internal DC Signal Monitor
30	M_RL_MON_TX5_INTRNAL_DC_SIG	Tx5 Internal DC Signal Monitor
31	M_RL_MON_TX6_INTRNAL_DC_SIG	Tx6 Internal DC Signal Monitor
32	M_RL_MON_TX7_INTRNAL_DC_SIG	Tx7 Internal DC Signal Monitor
33	M_RL_MON_RX_HPF_INTRNAL_DC_SIG	Rx HPF and DC Signal Monitor
34	M_RL_MON_PM_CLK_INTRNAL_DC_SIG	PM and Clock DC Signal Monitor
35	M_RL_MON_DFE_BIST_FFT_CHECK	DFE and BIST-FFT Monitor
36	M_RL_MON_STATIC_REG_READBACK	Static Register Read-Back Monitor

Monitor responses are stored at a fixed address location in a shared memory between FECSS and AppSS cores. Application should read this memory upon monitor completion to get the results. Following structure shows the details about the response.

Table 2.56: T_RL_FE_RFS_IPC_MON_RESULT_DATA_STRUCT

Field	Data Type	Size	Offset
T_RL_API_MON_TRIG_RESPONSE	T_RL_API_MON_TRIG_RESPONSE	20	0x21200500
T_RL_API_MON_LIVE_SYNTH_FREQ_RESULT	T_RL_API_MON_LIVE_SYNTH_FREQ_RESULT	24	0x21200514
T_RL_API_MON_PLL_CTRL_VOLT_RESULT	T_RL_API_MON_PLL_CTRL_VOLT_RESULT	28	0x2120052c
T_RL_API_MON_TXN_RX_LB_RESULT	T_RL_API_MON_TXN_RX_LB_RESULT[4]	224	0x21200548
T_RL_API_MON_TXN_POWER_RESULT	T_RL_API_MON_TXN_POWER_RESULT[4]	64	0x21200628
T_RL_API_MON_TXN_BB_RESULT	T_RL_API_MON_TXN_BB_RESULT[4]	64	0x21200668
T_RL_API_MON_TXN_DCSIG_RESULT	T_RL_API_MON_TXN_DCSIG_RESULT[4]	64	0x212006a8
T_RL_API_MON_RX_HPF_DCSIG_RESULT	T_RL_API_MON_RX_HPF_DCSIG_RESULT	36	0x212006e8
T_RL_API_MON_PMCLK_DCSIG_RESULT	T_RL_API_MON_PMCLK_DCSIG_RESULT	24	0x2120070c
w_Reserved1	UINT32[39]	156	0x21200724
Total		**704**	

Note

- All monitor configurations are stored in the FECSS IPC memory and will be retained across deep sleep if c_RetentionMode is enabled. Application can reduce overhead by avoiding reconfiguration in warm boot (c_PowerMode).
- DFE and BIST-FFT Monitor and Static Register Read-Back Monitor do not have any configuration or generate any response. Their pass/fail status is indicated in w_MonitorStatus.

2.5.1 FECSS Monitor Enable Trigger API

This API is used to trigger the monitors.
The API triggers all the enabled monitors and stores results in a predefined IPC RAM with fixed address location. The API immediately returns a response by acknowledging the trigger to unblock the Application. The RFS generates an interrupt to the application core upon completion to indicate monitoring done event. Application should call this API in inter-frame idle time and enable FEC_MON_EVENT (IRQ 4) to receive and respond to this event.
The API also provides mask to inject faults. RFS firmware automatically injects the all supported and enabled faults before running the corresponding monitor and removes it immediately, except for the faults enabled for live monitors. These faults must be disabled explicitly by calling the API again.

Note

- The collective execution status of all the monitors are stored in the address 0x21200500, upon receiving FEC_MON_EVENT, the APPSS can read this collective status before reading each of the individual results.
- RFS Firmware will not generate FEC_MON_EVENT (IRQ 4) if the API is issues with all monitors disabled.

Warning

- The application is responsible for reading out all the results from the predefined address.
- Faults supported by live monitors are not removed automatically.
- All loopbacks must be disabled before triggering monitors.
- Application must ensure sufficient idle is present in the idle time to complete all the enabled monitors. Refer Monitor Execution Time for execution time required per monitor.
- Live monitor faults must be disabled before running calibrations.

Following table shows the supported faults for the monitors: Notation:

- **X**: Recommended fault
- **O**: Optional fault
- Blank Cell: Unsupported fault

Monitor	Synth Fault	Rx IFA Gain Fault	Rx High Noise Fault	HPF Cutoff Fault	Tx Gain Fault	Tx Phase Inversi Fault	GPAD Fault	BIST FFT Gain Fault	Static Reg Read-back Fault
PLL_CTRL_VOLT	**X**								
TX0_RX_LB		**X**		O	**X**	**X**			
TX1_RX_LB		**X**		O	**X**	**X**			
TX2_RX_LB		**X**		O	**X**	**X**			
TX3_RX_LB		**X**		O	**X**	**X**			
TX4_RX_LB		**X**		O	**X**	**X**			
TX5_RX_LB		**X**		O	**X**	**X**			
TX6_RX_LB		**X**		O	**X**	**X**			
TX7_RX_LB		**X**		O	**X**	**X**			
TX0_POWER					**X**				
TX1_POWER					**X**				
TX2_POWER					**X**				
TX3_POWER					**X**				
TX4_POWER					**X**				
TX5_POWER					**X**				
TX6_POWER					**X**				
TX7_POWER					**X**				
TX0_BB					**X**				

Monitor	Synth Fault	Rx IFA Gain Fault	Rx High Noise Fault	HPF Cutoff Fault	Tx Gain Fault	Tx Phase Inversi Fault	GPAD Fault	BIST FFT Gain Fault	Static Reg Read-back Fault
TX1_BB					X				
TX2_BB					X				
TX3_BB					X				
TX4_BB					X				
TX5_BB					X				
TX6_BB					X				
TX7_BB					X				
TX0_INTRNAL_DC_SIG							X		
TX1_INTRNAL_DC_SIG							X		
TX2_INTRNAL_DC_SIG							X		
TX3_INTRNAL_DC_SIG							X		
TX4_INTRNAL_DC_SIG							X		
TX5_INTRNAL_DC_SIG							X		
TX6_INTRNAL_DC_SIG							X		
TX7_INTRNAL_DC_SIG							X		
RX_HPF_INTRNAL_DC_SIG		O		X			X		
PM_CLK_INTRNAL_DC_SIG							X		
DFE_BIST_FFT_CHECK								X	
STATIC_REG_READBACK									X
RX_SATURATION_LIVE			X						
SYNTH_FREQUENCY_LIVE	X								

API Definition

API Index	0x0080U
API Function Name	rl_monEnableTrig
API Configuration Structure	T_RL_API_MON_ENABLE_TRIG
API Response Structure	None
Monitor Response Structure	T_RL_API_MON_TRIG_RESPONSE

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_MON_ENABLE_TRIG             ((UINT16)0x0080U)
// API Function Definition
    T_RETURNTYPE rl_monEnableTrig(UINT8 c_devIndex, const T_RL_API_MON_ENABLE_TRIG
    ↪  *p_apiCmdData);
```

Configuration C Structure

Table 2.58: T_RL_API_MON_ENABLE_TRIG

Field	Data Type	Size	Offset
w_MonitorEnable	UINT32[2]	8	0
w_FaultInjEnable	UINT32	4	8
w_Reserved1	UINT32	4	12
w_Reserved2	UINT32	4	16
w_Reserved3	UINT32	4	20
Total		**24**	

Monitor Response C Structure

Table 2.59: T_RL_API_MON_TRIG_RESPONSE

Field	Data Type	Size	Offset
w_MonitorStatus	UINT32[2]	8	0
w_FrameCount	UINT32	4	8
w_Reserved2	UINT32	4	12
w_Reserved3	UINT32	4	16
Total		**20**	

Configuration Fields

- `w_MonitorEnable`

FECSS Monitors enable control, 1-bit per monitor. Refer mmWaveLink Monitoring Indices for the bit indices.

Value	Definition
0	Monitor Disabled.
1	Monitor Enabled.

Following table shows the supported monitor enable mask for different devices:

Table 2.60: Supported Monitor Enable Masks

Device Type	Supported Monitor Mask	Description
Safety Enabled xWRL1432	$w\_MonitorEnable[1] = 0x1E$ $w\_MonitorEnable[0] = 0x06060607$	Tx[2:7] monitors are reserved
Safety Enabled xWRL6432	$w\_MonitorEnable[1] = 0x1E$ $w\_MonitorEnable[0] = 0x06060607$	Tx[2:7] monitors are reserved
Safety Disabled xWRL1432	$w\_MonitorEnable[1] = 0x00$ $w\_MonitorEnable[0] = 0x00060006$	Only Tx[0-1]-BallBreak and Tx[0:2]-Rx loopback monitors are supported
Safety Disabled xWRL6432	$w\_MonitorEnable[1] = 0x00$ $w\_MonitorEnable[0] = 0x00060006$	Only Tx[0-1]-BallBreak and Tx[0:2]-Rx loopback monitors are supported

- `w_FaultInjEnable`

FECSS monitors fault injection enable control, 1 bit control per monitor.

Value	Definition
0	Fault Injection Disabled.
1	Fault Injection Enabled.

Bit Field Definition:

Fault Index	mmWaveLink Macro	Description
0	`M_RL_MON_FAULT_SYNTH`	Synthesizer fault This fault affects output frequency of the Synthesizer
1	`M_RL_MON_FAULT_RX_IFA_GAIN_DROP`	Rx IFA gain drop fault
2	`M_RL_MON_FAULT_RX_HIGH_NOISE`	Rx high noise fault This fault injects high noise in the Rx chain which could result in Rx ADC saturation

Fault Index	mmWaveLink Macro	Description
3	M_RL_MON_FAULT_RX_HPF_HIGH_CUTOFF	Rx HPF cutoff fault
4	M_RL_MON_FAULT_TX_GAIN_DROP	Tx gain drop fault
5	M_RL_MON_FAULT_TX_PHASE_INVERSION	Tx phase inversion fault This fault enforces BPM for all monitoring chirps
6	M_RL_MON_FAULT_GPADC	GPADC fault This fault affects only the first measurement in DC signal monitors
7	M_RL_MON_FAULT_BIST_FFT_GAIN	BIST FFT gain drop fault
8	M_RL_MON_FAULT_FECSS_STATIC_REG	FECSS static register read-back fault

Warning M_RL_MON_FAULT_RX_IFA_GAIN_DROP and M_RL_MON_FAULT_TX_GAIN_DROP should not be enabled together.
Tx gain drop fault causes a drop in both input power (xh_RxInputPower) as well as output powers (xh_TxNRxLbPower and xh_TxNRxLbBpmPower). Rx gain (computed by application software as the difference between output and input powers) may not necessarily show a fault with M_RL_MON_FAULT_TX_GAIN_DROP. For seeing a deterministic fault in computed Rx gain, both fault should not be enabled together.

Monitor Response Fields

Monitor trigger response is stored in the IPC response structure. Refer T_RL_FE_RFS_IPC_MON_RESULT_DATA_STRUCT for the details.

- w_MonitorStatus

FECSS Monitors run status, 1 bit status per monitor. This is the response status of the current monitor enable command trigger. Application should check this response upon receiving the FEC_MON_EVENT (IRQ 4) event to check trigger stratus of the monitors.

Value	Definition
0	Not Updated / Failed Cleared bit indicated monitor run failed if the monitor is enabled in w_MonitorEnable or monitored is not enabled
1	Monitor is triggered Enabled Monitor has successfully executed and result in updated.

Refer mmWaveLink Monitoring Indices for the monitor indices.

- `w_FrameCount`

Monitor report frame count.
RFS firmware maintains internal 32-bit count of the number of frames triggered from Cold Boot. This count is reported in this field to uniquely identify monitor triggers.

Warning Application has to handle the roll-over of the 32-bit counter, if any.

2.5.2 FECSS Peak-Detector Debug Monitor API

This API provides interface to measure RF power at various points in the RF signal chain using on-chip peak-detectors (PD) when issued in CW mode. PDs are uniquely identified using two parameters, PD LNA index and PD LNA Mux index. RFS firmware maintains the mapping for frequently used PDs (critical PDs). The API provides options to measure critical PDs directly using the internal mapping. Non-critical PD measurements require inputs from the API to identify PD.

Warning

- This API is only meant for debug purpose and must not be used to measure/estimate Tx power or achieve functional safety objectives.
- PD measurements could be unpredictable if sensor is not enabled in CW mode or not properly calibrated.

Note This API does not update the sensor state. Use rlsensSensorStart and Sensor Stop API to enable or disable the CW mode transmission.

API Definition

API Index	0x0081U
API Function Name	rl_monDbgPdMeas
API Configuration Structure	T_RL_API_MON_DBG_PD_MEAS_CMD
API Response Structure	T_RL_API_MON_DBG_PD_MEAS_RSP

- API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_MON_DBG_PD_PWR_MEAS          ((UINT16)0x0081U)
// API Function Definition
    T_RETURNTYPE rl_monDbgPdMeas(UINT8 c_devIndex, const T_RL_API_MON_DBG_PD_MEAS_CMD
    ↪  *p_apiCmdData, T_RL_API_MON_DBG_PD_MEAS_RSP *p_apiResData);
```

- Configuration C Structure

Table 2.62: T_RL_API_MON_DBG_PD_MEAS_CMD

Field	Data Type	Size	Offset
c_PdId	UINT8	1	0
c_PdLnaIndx	UINT8	1	1
c_PdLnaGainIndx	UINT8	1	2
c_PdType	UINT8	1	3
c_PdSelect	UINT8	1	4
c_PdLnaOffsetDacVal	UINT8	1	5
c_PdNAccum	UINT8	1	6
c_PdNSamples	UINT8	1	7
w_Reserved1	UINT32	4	8
w_Reserved2	UINT32	4	12
Total		**16**	

- Response C Structure

Table 2.63: T_RL_API_MON_DBG_PD_MEAS_RSP

Field	Data Type	Size	Offset
w_SumRfON	UINT32	4	0
w_SumRfOff	UINT32	4	4
h_DeltaSum	UINT16	2	8
xh_PdPwrDb	SINT16	2	10
w_VpdRmsSqr	UINT32	4	12
c_PdMeasSts	UINT8	1	16
c_Reserved0	UINT8	1	17
h_Reserved1	UINT16	2	18
w_Reserved2	UINT32	4	20
Total		**24**	

Configuration Fields

- c_PdId

Unique firmware assigned index to PD. This index is assigned only to critical PDs.

- Applicable only for critical PDs.

- Use $0xFF$ for non-critical PDs.

▶ `c_PdLnaIndx`

Unique index assigned to the LNA to which PD is connected. The field will be ignored for the critical PDs.

▶ `c_PdLnaGainIndx`

PD LNA gain index for measurement.

- 1LSB = 4dB
- Range: $[1, 9]$

▶ `c_PdType`

PD Type

- 0 - HPPD
- 1 - LPPD

▶ `c_PdSelect`

This field selects the PD from the LNA selected using `c_PdLnaIndx`. The field will be ignored for the critical PDs.

▶ `c_PdLnaOffsetDacVal`

PD offset DAC value in 7-bit sign magnitude format. The field will be ignored for the critical PDs.

▶ `c_PdNAccum`

Number of GPADC measurements used for PD measurement (Software Accumulation).

▶ `c_PdNSamples`

Number of samples used in each GPADC measurement (Hardware Accumulation).

Response Fields

▶ `w_SumRfON`

Accumulated GPADC value across all RF on measurements in GPADC codes.

- `w_SumRfOff`

Accumulated GPADC value across all RF off measurements in GPADC codes.

- `h_DeltaSum`

Difference between `w_SumRfON` and `w_SumRfOff`. This value is used to compute power measured by the PD.

- `xh_PdPwrDb`

Power measured by the PD. $1LSB = 0.1dBm$.

- `w_VpdRmsSqr`

Square of RMS voltage of the signal measured by the PD. $1LSB = (31.25\mu v)^2$

- `c_PdMeasSts`

PD measurement status.

- 0 - Fail
- 1 - Pass

2.5.3 FECSS Tx Power Debug Monitor API

This API provides interface to measure Tx power in CW mode.

Warning

- This API is only meant for debug purpose and must not be used to achieve functional safety objectives.
- Tx measurements could be unpredictable if sensor is not enabled in CW mode or not properly calibrated.

Note This API does not update the sensor state. Use rlsensSensorStart and Sensor Stop API to enable or disable the CW mode transmission.

API Definition

API Index	0x0082U
API Function Name	`rl_monDbgTxPwrMeas`
API Configuration Structure	`T_RL_API_MON_DBG_TXPWR_MEAS_CMD`
API Response Structure	`T_RL_API_MON_DBG_TXPWR_MEAS_RSP`

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_MON_DBG_TX_PWR_MEAS          ((UINT16)0x0082U)
// API Function Definition
    T_RETURNTYPE rl_monDbgTxPwrMeas(UINT8 c_devIndex, const T_RL_API_MON_DBG_TXPWR_MEAS_CMD
    ↪  *p_apiCmdData, T_RL_API_MON_DBG_TXPWR_MEAS_RSP *p_apiResData)
```

Configuration C Structure

Table 2.64: T_RL_API_MON_DBG_TXPWR_MEAS_CMD

Field	Data Type	Size	Offset
c_TxIndex	UINT8	1	0
c_TxPwrNAccum	UINT8	1	1
c_TxBackoff	UINT8	1	2
c_Reserved0	UINT8	1	3
w_Reserved1	UINT32	4	4
w_Reserved2	UINT32	4	8
Total		**12**	

Response C Structure

Table 2.65: T_RL_API_MON_DBG_TXPWR_MEAS_RSP

Field	Data Type	Size	Offset
xh_TxIncidentPwr	SINT16	2	0
xh_TxReflectedPwr	SINT16	2	2
w_TxIncidentVolt	UINT32	4	4
w_TxReflectedVolt	UINT32	4	8
xh_TxPower	SINT16	2	12
h_Reserved1	UINT16	2	14
w_Reserved2	UINT32	4	16
Total		**20**	

Configuration Fields

- c_TxIndex

Index of the Tx channel to be measured.

- `c_TxPwrNAccum`

Number of GPADC measurements (Software accumulations) used to compute Tx power.

- `c_TxBackoff`

Tx backoff of the measured channel. This filed is used only to configure and compute the PD parameters, it does not update the output power.

- 1LSB: 0.5d
- Valid Range: Even values (1dB step) from 0 (0dB) to 52 (26dB)

Response Fields

- `xh_TxIncidentPwr`

Power measured at incident PD in dBm. $1LSB = 0.1dBm$.

- `xh_TxReflectedPwr`

Power measured at reflected PD in dBm. $1LSB = 0.1dBm$.
This is valid only for certain back-off ranges. Please refer monitoring application note for more details.

- `w_TxIncidentVolt`

Square of RMS voltage of signal measured by the incident Tx PD. $1LSB = (31.25\mu v)^2$

- `w_TxReflectedVolt`

Square of RMS voltage of signal measured by the reflected Tx PD. $1LSB = (31.25\mu v)^2$.
This is valid only for certain back-off ranges. Please refer monitoring application note for more details.

- `xh_TxPower`

Total Tx power measured by the PD. $1LSB = 0.1dBm$.

2.5.4 FECSS Synthesizer Frequency Live Monitor Configuration API

This API configures the Synthesizer frequency live monitor.
Live monitor runs in background when functional frames are running and can be enabled in Sensor Start API.
Monitor results are updated to the `T_RL_API_MON_LIVE_SYNTH_FREQ_RESULT` at the end of every frame.

Notes

- The application can avoid configuring the monitor repeatedly in every deep sleep cycle as this configuration is retained in FEC RAM during deep sleep.
- The result is updated in IPC RAM at the end of every frame (At the end of *FrameEndTime*), the application must read the results before the next Frame End Event.

API Definition

API Index	0x0090U
API Function Name	rl_monLiveSynthFreqCfg
API Configuration Structure	T_RL_API_MON_LIVE_SYNTH_FREQ_CFG
API Response Structure	None
Monitor Response Structure	T_RL_API_MON_LIVE_SYNTH_FREQ_RESULT

API Function

mmWaveLink C function declaration

Configuration C Structure

Table 2.66: T_RL_API_MON_LIVE_SYNTH_FREQ_CFG

Field	Data Type	Size	Offset
xc_ChirpMonStartTime	SINT8	1	0
c_Reserved1	UINT8	1	1
h_FreqErrThreshold	UINT16	2	2
w_Reserved2	UINT32	4	4
w_Reserved3	UINT32	4	8
Total		**12**	

- Response C Structure

Table 2.67: T_RL_API_MON_LIVE_SYNTH_FREQ_RESULT

Field	Data Type	Size	Offset
c_MonStatus	UINT8	1	0
c_Reserved1	UINT8	1	1
h_Reserved2	UINT16	2	2
xw_MaxSynthFreqErr	SINT32	4	4
w_FreqErrFailCount	UINT32	4	8
w_Reserved3	UINT32	4	12
w_FrameCount	UINT32	4	16
w_Reserved4	UINT32	4	20
Total		**24**	

Configuration Fields

- xc_ChirpMonStartTime

Synth frequency monitor start time.

- This field determines when the monitoring starts in each chirp relative to the start of the ramp.
- $1LSB = 80/APLL\_FREQ$ (Typical 0.2 µs with 40MHz XTAL), signed number.
- Valid range: -125 to +125 (-25 to 25 µs)
- Recommended value: 4µs or above.

- h_FreqErrThreshold

Synth frequency monitor error threshold.
In synth freq monitor the error of the measured instantaneous chirp frequency with respect to the desired value is continuously compared against this threshold during the chirp.
The result of the comparison is reported in the monitor result. An Error bit is set if the measurement ever exceeds the above threshold anytime during the active chirp.

RF Frequency	Resolution
60 GHz	$1LSB = 3 \times$ (APLL_FREQ)$/(144 \times 2^8) = 32.552kHz.$
77 GHz	$1LSB = 4 \times$ (APLL_FREQ)$/(144 \times 2^8) = 43.408kHz.$

Response Fields

Monitor response is stored in IPC memory for each Tx channel. Refer T_RL_FE_RFS_IPC_MON_RESULT_DATA_STRUCT for the memory addresses.

- `c_MonStatus`

Monitor PASS/ FAIL status. Response status flag indicating pass fail results corresponding to threshold checks under this monitor.

Value	Definition
0	Fail
1	Pass

- `xw_MaxSynthFreqErr`

Signed max synth frequency error value.

- This field indicates the maximum instantaneous frequency error measured during the last monitoring frame period for which frequency monitoring has been enabled.
- In Synth freq monitor the error of the measured instantaneous chirp frequency with respect to the desired value.

RF Frequency	Resolution
60 GHz	$1LSB = 3 \times$ (APLL_FREQ)$/(144 \times 2^13) = \ 1.017kHz.$
77 GHz	$1LSB = 4 \times$ (APLL_FREQ)$/(144 \times 2^13) = \ 1.356kHz.$

Valid Range: $[-2^{21}, 2^{21})$

- `w_FreqErrFailCount`

Frequency error failure count.
This field indicates the number of times during chirping in the previous monitoring frame period in which the measured frequency error violated the allowed threshold.

- Frequency error threshold violation is counted every 10 ns.
- Valid range: [0, 524287]

- `w_FrameCount`

RFS frame count when the monitor report is updated.

note Value of this counter lags by 1 with respect to other monitors as the results are generated before the Frame End Event

2.5.5 FECSS Rx Saturation Live Monitor Configuration API

This API configures Rx saturation live monitor. This monitor reports number of saturated Rx ADC samples per Rx channel for every chirp. Live monitor runs in background when functional frames are running and can be enabled in Sensor Start API. Monitor results are updated at a programmable offset in `PER_CHIRP_LUT` RAM.

API Definition

API Index	0x0091U
API Function Name	rl_monLiveRxSaturationCfg
API Configuration Structure	T_RL_API_MON_LIVE_RX_SATURATION_CFG
API Response Structure	None

▶ API Function

mmWaveLink C function declaration

▶ Configuration C Structure

Table 2.68: T_RL_API_MON_LIVE_RX_SATURATION_CFG

Field	Data Type	Size	Offset
h_PerChirpRamStartAdd	UINT16	2	0
h_Reserved1	UINT16	2	2
w_Reserved2	UINT32	4	4
Total		**8**	

Configuration Fields

▶ h_PerChirpRamStartAdd

The per chirp RAM start offset address for saturation monitor result storage. This field is used to program the start offset address of saturation monitor result ARRAY in PER_CHIRP_LUT RAM.

- The start address offset should be multiple of 4 bytes.
- The max addressable space 8kB.
- The size of this ARRAY should be sufficient to hold all chirp data in a frame.
- Total size of RAM = 4 bytes per chirp ×num of chirps in a burst ×num of bursts in frame.
- Each chirp will have 1 byte data for each RX, max 4 RXs are supported (Adding is added if device supports less number of Rx channels).
- Valid range: 0, 4, 8 to 8188.

2.5.6 FECSS PLL Control Voltage Monitor Configuration API

This API configures PLL control voltage monitor. The monitor measures control voltages for APLL and Synthesizer PLLs. The monitor can be triggered using FECSS Monitor Enable Trigger API. The measured values

of the PLL control voltages are reported in the response Structure T_RL_API_MON_PLL_CTRL_VOLT_RESULT in memory mapped monitor response T_RL_FE_RFS_IPC_MON_RESULT_DATA_STRUCT .

API Definition

API Index	0x0093U
API Function Name	rl_monPllCtrlVoltCfg
API Configuration Structure	T_RL_API_MON_PLL_CTRL_VOLT_CFG
API Response Structure	None
Monitor Response Structure	T_RL_API_MON_PLL_CTRL_VOLT_RESULT

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_MON_PLL_CTRL_VOLT_CFG              ((UINT16)0x0093U)
// API Function Definition
    T_RETURNTYPE rl_monPllCtrlVoltCfg(UINT8 c_devIndex, const T_RL_API_MON_PLL_CTRL_VOLT_CFG
    ↪  *p_apiCmdData);
```

Configuration C Structure

Table 2.69: T_RL_API_MON_PLL_CTRL_VOLT_CFG

Field	Data Type	Size	Offset
c_PllMonEnaMask	UINT8	1	0
c_Reserved1	UINT8	1	1
h_Reserved2	UINT16	2	2
w_Reserved3	UINT32	4	4
Total		**8**	

- Response C Structure

Table 2.70: T_RL_API_MON_PLL_CTRL_VOLT_RESULT

Field	Data Type	Size	Offset
xh_ErrorCode	SINT16	2	0
h_Reserved1	UINT16	2	2
h_ApllCtrlVolt	UINT16	2	4
h_SynthMinCtrlVolt	UINT16	2	6
h_SynthMaxCtrlVolt	UINT16	2	8
h_Reserved2	UINT16	2	10
w_Reserved3	UINT32	4	12
w_Reserved4	UINT32	4	16
w_Reserved5	UINT32	4	20
w_FrameCount	UINT32	4	24
Total		**28**	

Configuration Fields

- c_PllMonEnaMask

APLL Synth control voltage monitor enable mask.

Value	Definition
0	Monitor Disabled
1	Monitor Enabled

Bit Field Definition:

Bit-Field	Definition
Bit[0]	APLL control voltage monitor enable
Bit[1]	Synthesizer VCO control voltage monitor enable
Bit[7:2]	RESERVED

Monitor Response Fields

Monitor response is stored in the IPC response structure. Refer T_RL_FE_RFS_IPC_MON_RESULT_DATA_STRUCT for the details.

- xh_ErrorCode

Monitor Error code. 2's complement signed number.

Value	Definition
0	Monitor executed successfully
2's complement negative number	Monitor execution failed

- `h_ApllCtrlVolt`

APLL control voltage value in 10-bit unsigned code.

- This field is valid only if the APLL control voltage monitor is enabled.
- 1LSB = 1.8V / 1024.
- Valid range: 0 to 1023.

- `h_SynthMinCtrlVolt`

Synthesizer control voltage for the minimum supported RF frequency. The voltage is reported as 10-bit unsigned code.

Device Type	Min RF Frequency
60GHz Device	57GHz
77GHz Device	76GHz

- This field is valid only if the Synth control voltage monitor is enabled.
- 1LSB = 1.8V / 1024.
- Valid range: 0 to 1023.

- `h_SynthMaxCtrlVolt`

Synthesizer control voltage for the maximum supported RF frequency. The voltage is reported as 10-bit unsigned code.

Device Type	Max RF Frequency
60GHz Device	64GHz* (Variant specific limitations in Release Notes)
77GHz Device	81GHz

- Only the fields corresponding to the enabled monitors are valid.
- 1LSB = 1.8V / 1024.
- Valid range: 0 to 1023.

- `w_FrameCount`

RFS frame count when the monitor report is updated.

2.5.7 FECSS TxN-Rx Loopback Monitor Configuration API

This API configures the Tx-Rx loopback monitor for all Tx channels. Application should call this API multiple times to configure monitor for different Tx channels. The monitor enables PA loopback from the selected Tx channel to all enabled Rx channels and measures various tone parameters with IF mid-band tone of frequency $2.77MHz$. It also includes options to measure and BPM. Rx input power and loopback power with BPM enabled.
The monitor can be triggered using FECSS Monitor Enable Trigger API and the response is stored in memory mapped response T_RL_API_MON_TXN_RX_LB_RESULT .

Note

- This monitor enables corresponding Tx channel with transmission enabled.
- This monitor also enables Rx channels, hence results are susceptible to external interference.

API Definition

API Index	0x0094U
API Function Name	rl_monTxNRxLbCfg
API Configuration Structure	T_RL_API_MON_TXN_RX_LB_CFG
API Response Structure	None
Monitor Response Structure	T_RL_API_MON_TXN_RX_LB_RESULT

▶ API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_MON_TXn_RX_LB_CFG                    ((UINT16)0x0094U)
// API Function Definition
    T_RETURNTYPE rl_monTxNRxLbCfg(UINT8 c_devIndex, const T_RL_API_MON_TXN_RX_LB_CFG
    ↪  *p_apiCmdData);
```

- Configuration C Structure

Table 2.71: T_RL_API_MON_TXN_RX_LB_CFG

Field	Data Type	Size	Offset
c_TxIndSel	UINT8	1	0
c_MonEnaCtrl	UINT8	1	1
c_MonRxTxCodesSel	UINT8	1	2
c_MonRxGainCode	UINT8	1	3
h_MonTxBiasCodes	UINT16	2	4
h_RfFreqStart	UINT16	2	6
xh_RfFreqSlope	SINT16	2	8
h_Reserved1	UINT16	2	10
w_Reserved2	UINT32	4	12
Total		**16**	

- Response C Structure

Table 2.72: T_RL_API_MON_TXN_RX_LB_RESULT

Field	Data Type	Size	Offset
xh_ErrorCode	SINT16	2	0
xh_RxInputPower	SINT16	2	2
xh_TxNRxLbPower	SINT16[4]	8	4
h_TxNRxLbPhase	UINT16[4]	8	12
xh_TxNRxLbBpmPower	SINT16[4]	8	20
h_TxNRxLbBpmPhase	UINT16[4]	8	28
xc_TxNRxLbNoise	SINT8[4]	4	36
xc_TxNRxLbBpmNoise	SINT8[4]	4	40
w_Reserved2	UINT32	4	44
w_Reserved3	UINT32	4	48
w_FrameCount	UINT32	4	52
Total		**56**	

Configuration Fields

- c_TxIndSel

Index of the Tx channel for which the configuration is applied.

Value	Definition
0	The Monitor configuration will be applicable for TX0.
1	The Monitor configuration will be applicable for TX1.
2 to 255	Reserved

Note Tx channel will be enabled irrespective of the `h_TxChCtrlBitMask` if corresponding monitor is enabled in `w_MonitorEnable`. Rx channels will be enabled according to the `h_RxChCtrlBitMask`.

- `c_MonEnaCtrl`

TxN-Rx loopback monitor control.
The monitor supports measuring QPSK PD along with the $BPM = 0^o$ and $BPM = 180^o$ in a signal iteration. This field controls the enable masks for the required measurements.

Value	Definition
0	Disabled
1	Enabled

Bit Field	Definition
Bit[0]	`TxN_LB_MON` Enable bit Enable Txn$_RxLoopbackMonitorwith$BPM = 0^o
Bit[1]	`TxN_BPM_MON` enable bit. Enable Txn$_RxLoopbackMonitorwith$BPM = 180^o
Bit[2]	Rx input power measurement enable bit. **Warning** • Accuracy of the reported Rx input power degrades with higher backoff. • Either `TxN_LB_MON` or `TxN_BPM_MON` monitors must be enabled to measure Rx input power.

- `c_MonRxTxCodesSel`

Monitor RX gain and TX bias code selection controls.

Bit Field	Definition
Bit[0]	RX_GAIN_CODE override enable. • Value 0: Use calibrated Rx gain codes • Value 1: Use Rx gain codes from c_MonRxGainCode
Bit[1]	TX_PA_CODE override enable. • Value 0: Use calibrates TX_PA_CODE • Value 1: Use TX_PA_CODE from h_MonTxBiasCodes.
Bit[2]	Monitor loopback gain select. • Value 0: Use low PA loopback gain (-10 dB). • Value 1: Use loopback gain (0 dB). • Recommended value: 1 (0 dB)
Bit[7:3]	Reserved

Recommendation Recommended value for c_MonRxTxCodesSel is 0x5

▶ c_MonRxGainCode

Monitor RX channels gain code override setting. This is a common IFA code value for all chirps used in this monitor. The monitor compensates for the overridden IFA gain code internally and reports the tone powers for the currently applied calibrated values. THe IFA gain code value should be chosen appropriately to avoid Rx chain saturation.

Bit definition:

Bit Field	Definition
bits [4:0]	IFA_GAIN_CODE. • Override IFA gain code. • Valid range: [0, 10] • $IFA\_GAIN(dB) = IFA\_GAIN\_CODE \times 2 - 10dB$
bits [7:5]	Reserved

Recommendation Recommended value for IFA gain code is: c_MonRxGainCode $= 2 +$ (c_CalTxBackOffSel$/2$)

▶ h_MonTxBiasCodes

Override value for TX_PA_CODES.

▶ h_RfFreqStart

Monitor chirp start frequency in low resolution.

RF Frequency	Resolution
60GHz	$1LSB = (3 \times APLL\_FREQ)/2^{10} \approx 1.172MHz$ Valid Range: 57 GHz to 64 GHz for APLL_FREQ = 400MHz: 0xBE00U (57GHz) to 0xD555U (64 GHz)* *Variant specific limitations in Release Notes
77GHz	$1LSB = (4 \times APLL\_FREQ)/2^{10} \approx 1.562MHz$ Valid Range: 76 GHz to 81 GHz for APLL_FREQ = 400MHz: 0x0000BE00 (76GHz) to 0x0000CA80 (81GHz)

- `xh_RfFreqSlope`

RF Slope of the monitoring chirp.

RF Slope	Resolution
60GHz	$1LSB = (3 \times 400 \times APLL\_FREQ)/2^{24} \approx 28.610kHz/us$,signed number Valid Range: -3495 to 3495 (+/- 100MHz/us for APLL_FREQ = 400MHz)
77GHz	$1LSB = (4 \times 400 \times APLL\_FREQ)/2^{24} \approx 38.147kHz/us$, signed number Valid Range: -3495 to 3495 (+/- 100MHz/us for APLL_FREQ = 400MHz)

Warning Application should ensure that the Monitor chirp's end frequency is within the operating range of a respective device.

Response Fields

Monitor response is stored in IPC memory for each Tx channel. Refer `T_RL_FE_RFS_IPC_MON_RESULT_DATA_STRUCT` for the memory addresses.

- `xh_ErrorCode`

Monitor Error code. 2's complement signed number.

Value	Definition
0	Monitor executed successfully
2's complement negative number	Monitor execution failed

- `xh_RxInputPower`

Rx input power measured using PD at the loopback buffer in dB. $1LSB = 0.1dBm$.
This field is valid only if Rx input measurement is enabled and at least one of the `TxN_LB_MON` `TxN_BPM_MON` is enabled.

Note Accuracy of the Rx Input power will degrade with higher Tx back-off. It is recommended to measure this power with least backoff possible (preferably 0dB).

- `xh_TxNRxLbPower`

Array of loopback tone powers measured with $BPM = 0^o$ across Rx channels. $1LSB = 0.1dBm$
These values are valid only if the bit `TxN_LB_MON` is enabled in `c_MonEnaCtrl` and corresponding Rx channel is enabled in `h_RxChCtrlBitMask`.

- `h_TxNRxLbPhase`

Array of loopback tone phases measured with $BPM = 180^o$ across Rx channels. $1LSB = 360^o/2^16$
These values are valid only if the bit `TxN_LB_MON` is enabled in `c_MonEnaCtrl` and corresponding Rx channel is enabled in `h_RxChCtrlBitMask`.

- `xh_TxNRxLbBpmPower`

Array of loopback tone powers measured with $BPM = 180^o$ across Rx channels. $1LSB = 0.1dBm$
These values are valid only if the bit `TxN_BPM_MON` is enabled in `c_MonEnaCtrl` and corresponding Rx channel is enabled in `h_RxChCtrlBitMask`.

- `h_TxNRxLbBpmPhase`

Array of loopback tone phases measured with $BPM = 180^o$ across Rx channels. $1LSB = 360^o/2^16$
These values are valid only if the bit `TxN_BPM_MON` is enabled in `c_MonEnaCtrl` and corresponding Rx channel is enabled in `h_RxChCtrlBitMask`.

- `xc_TxNRxLbNoise`

Array of noise powers measured with $BPM = 0^o$ across Rx channels. $1LSB = 1dBm$
These values are valid only if the bit `TxN_LB_MON` is enabled in `c_MonEnaCtrl` and corresponding Rx channel is enabled in `h_RxChCtrlBitMask`.

- `xc_TxNRxLbBpmNoise`

Array of noise powers measured with $BPM = 180^o$ across Rx channels. $1LSB = 1dBm$
These values are valid only if the bit `TxN_BPM_MON` is enabled in `c_MonEnaCtrl` and corresponding Rx channel is enabled in `h_RxChCtrlBitMask`.

- `w_FrameCount`

RFS frame count when the monitor report is updated.

2.5.8 FECSS TxN Power Monitor Configuration API

This API configures the Tx power monitor for all Tx channels. Application should call this API multiple times to configure monitor for different Tx channels. The API enables the selected Tx channel and measures transmitted power using the on-chip PDs.

The monitor can be triggered using FECSS Monitor Enable Trigger API and the response is stored in memory mapped response T_RL_API_MON_TXN_POWER_RESULT .

Note This monitor enables corresponding Tx channel with transmission enabled.

API Definition

API Index	0x0095U
API Function Name	rl_monTxNPowerCfg
API Configuration Structure	T_RL_API_MON_TXN_POWER_CFG
API Response Structure	None
Monitor Response Structure	T_RL_API_MON_TXN_POWER_RESULT

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_MON_TXn_PWR_CFG                    ((UINT16)0x0095U)
// API Function Definition
    T_RETURNTYPE rl_monTxNPowerCfg(UINT8 c_devIndex, const T_RL_API_MON_TXN_POWER_CFG
    ↪  *p_apiCmdData);
```

Configuration C Structure

Table 2.73: T_RL_API_MON_TXN_POWER_CFG

Field	Data Type	Size	Offset
c_TxIndSel	UINT8	1	0
c_MonTxCodesSel	UINT8	1	1
h_MonTxBiasCodes	UINT16	2	2
h_RfFreqStart	UINT16	2	4
xh_RfFreqSlope	SINT16	2	6
c_TxBackoffMap	UINT8	1	8
c_Reserved1	UINT8	1	9
h_Reserved2	UINT16	2	10
w_Reserved3	UINT32	4	12
Total		**16**	

- Response C Structure

Table 2.74: T_RL_API_MON_TXN_POWER_RESULT

Field	Data Type	Size	Offset
xh_ErrorCode	SINT16	2	0
h_Reserved1	UINT16	2	2
xh_TxNPwrVal	SINT16	2	4
h_Reserved2	UINT16	2	6
w_Reserved3	UINT32	4	8
w_FrameCount	UINT32	4	12
Total		**16**	

Configuration Fields

- c_TxIndSel

Index of the Tx channel for which the configuration is applied.

Value	Definition
0	The Monitor configuration will be applicable for TX0.
1	The Monitor configuration will be applicable for TX1.
2 to 255	Reserved

Note Tx channel will be enabled irrespective of the h_TxChCtrlBitMask if corresponding monitor is enabled in w_MonitorEnable.

- c_MonTxCodesSel

Tx monitor code select control

Bit Field	Definition
Bit[0]	Reserved
Bit[1]	TX_PA_CODE override enable. • Value 0: Use calibrates TX_PA_CODE • Value 1: Use TX_PA_CODE from h_MonTxBiasCodes.
Bit[7:1]	Reserved

- `h_MonTxBiasCodes`

Override value for `TX_PA_CODES`.

- `h_RfFreqStart`

Monitor chirp start frequency in low resolution.

RF Frequency	Resolution
60GHz	$1LSB = (3 \times APLL\_FREQ)/2^{10} \approx 1.172MHz$ Valid Range: 57 GHz to 64 GHz for APLL_FREQ = 400MHz: 0xBE00U (57GHz) to 0xD555U (64 GHz)* *Variant specific limitations in Release Notes
77GHz	$1LSB = (4 \times APLL\_FREQ)/2^{10} \approx 1.562MHz$ Valid Range: 76 GHz to 81 GHz for APLL_FREQ = 400MHz: 0x0000BE00 (76GHz) to 0x0000CA80 (81GHz)

- `xh_RfFreqSlope`

RF Slope of the monitoring chirp.

RF Slope	Resolution
60GHz	$1LSB = (3 \times 400 \times APLL\_FREQ)/2^{24} \approx 28.610kHz/us$,signed number Valid Range: -3495 to 3495 (+/- 100MHz/us for APLL_FREQ = 400MHz)
77GHz	$1LSB = (4 \times 400 \times APLL\_FREQ)/2^{24} \approx 38.147kHz/us$, signed number Valid Range: -3495 to 3495 (+/- 100MHz/us for APLL_FREQ = 400MHz)

Warning Application should ensure that the Monitor chirp's end frequency is within the operating range of a respective device.

- `c_TxBackoffMap`

Backoff applied to the Tx channels when `TX_PA_CODE` override is enabled. $1LSB = 0.5dB$
This value is required to correctly setup PDs and compute internal variables when override mode is enabled.

Response Fields

Monitor response is stored in IPC memory for each Tx channel. Refer `T_RL_FE_RFS_IPC_MON_RESULT_DATA_STRUCT` for the memory addresses.

- `xh_ErrorCode`

Monitor Error code. 2's complement signed number.

Value	Definition
0	Success
2's complement negative number	Failure

- `xh_TxNPwrVal`

Measured Tx output power in dB. $1LSB = 0.1dBm$

- `w_FrameCount`

RFS frame count when the monitor report is updated.

2.5.9 FECSS Tx Ball-Break Monitor Configuration API

This API configures the Tx ball-break monitor for all Tx channels. Application should call this API multiple times to configure monitor for different Tx channels. The API enables the selected Tx channel and measures both incident and reflected PDs. Ratio of powers measured by these two PDs can be used to check if contact with the Tx antenna.
The monitor can be triggered using FECSS Monitor Enable Trigger API and the response is stored in memory mapped response `T_RL_API_MON_TXN_BB_RESULT` .

Note

- Accuracy of the ball-break monitor degrades with back-off. It it recommended to run this monitor with least backoff possible (preferably 0dB) for the most reliable results.
- This monitor enables corresponding Tx channel with transmission enabled.

API Definition

API Index	0x0096U
API Function Name	rl_monTxNBbCfg
API Configuration Structure	T_RL_API_MON_TXN_BB_CFG
API Response Structure	None
Monitor Response Structure	T_RL_API_MON_TXN_BB_RESULT

- API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_MON_TXn_BB_CFG                          ((UINT16)0x0096U)
// API Function Definition
    T_RETURNTYPE rl_monTxNBbCfg(UINT8 c_devIndex, const T_RL_API_MON_TXN_BB_CFG
    ↪  *p_apiCmdData);
```

- Configuration C Structure

Table 2.75: T_RL_API_MON_TXN_BB_CFG

Field	Data Type	Size	Offset
c_TxIndSel	UINT8	1	0
c_MonTxCodesSel	UINT8	1	1
h_MonTxBiasCodes	UINT16	2	2
h_RfFreqStart	UINT16	2	4
xh_RfFreqSlope	SINT16	2	6
c_TxBackoffMap	UINT8	1	8
c_Reserved1	UINT8	1	9
h_Reserved2	UINT16	2	10
w_Reserved2	UINT32	4	12
Total		**16**	

- Response C Structure

Table 2.76: T_RL_API_MON_TXN_BB_RESULT

Field	Data Type	Size	Offset
xh_ErrorCode	SINT16	2	0
h_Reserved1	UINT16	2	2
xh_TxNIncidPwrVal	SINT16	2	4
xh_TxNReflPwrVal	SINT16	2	6
w_Reserved2	UINT32	4	8
w_FrameCount	UINT32	4	12
Total		**16**	

Configuration Fields

- c_TxIndSel

Index of the Tx channel for which the configuration is applied.

Value	Definition
0	The Monitor configuration will be applicable for TX0.
1	The Monitor configuration will be applicable for TX1.
2 to 255	Reserved

Note Tx channel will be enabled irrespective of the h_TxChCtrlBitMask if corresponding monitor is enabled in w_MonitorEnable.

- c_MonTxCodesSel

Tx monitor code select control

Bit Field	Definition
Bit[0]	Reserved
Bit[1]	TX_PA_CODE override enable. • Value 0: Use calibrates TX_PA_CODE • Value 1: Use TX_PA_CODE from h_MonTxBiasCodes.
Bit[7:1]	Reserved

- h_MonTxBiasCodes

Override value for TX_PA_CODES.

- h_RfFreqStart

Monitor chirp start frequency in low resolution.

RF Frequency	Resolution
60GHz	$1LSB = (3 \times APLL\_FREQ)/2^{10} \approx 1.172MHz$ Valid Range: 57 GHz to 64 GHz for APLL_FREQ = 400MHz: 0xBE00U (57GHz) to 0xD555U (64 GHz)* *Variant specific limitations in Release Notes
77GHz	$1LSB = (4 \times APLL\_FREQ)/2^{10} \approx 1.562MHz$ Valid Range: 76 GHz to 81 GHz for APLL_FREQ = 400MHz: 0x0000BE00 (76GHz) to 0x0000CA80 (81GHz)

- xh_RfFreqSlope

RF Slope of the monitoring chirp.

RF Slope	Resolution
60GHz	$1LSB = (3 \times 400 \times APLL\_FREQ)/2^{24} \approx 28.610kHz/us$,signed number Valid Range: -3495 to 3495 (+/- 100MHz/us for APLL_FREQ = 400MHz)
77GHz	$1LSB = (4 \times 400 \times APLL\_FREQ)/2^{24} \approx 38.147kHz/us$, signed number Valid Range: -3495 to 3495 (+/- 100MHz/us for APLL_FREQ = 400MHz)

Warning Application should ensure that the Monitor chirp's end frequency is within the operating range of a respective device.

- `c_TxBackoffMap`

Backoff applied to the Tx channels when `TX_PA_CODE` override is enabled. $1LSB = 0.5dB$
This value is required to correctly setup PDs and compute internal variables when override mode is enabled.

Response Fields

Monitor response is stored in IPC memory for each Tx channel. Refer `T_RL_FE_RFS_IPC_MON_RESULT_DATA_STRUCT` for the memory addresses.

- `xh_ErrorCode`

Monitor error code. 2's complement signed number.

Value	Definition
0	Success
2's complement negative number	Failure

- `xh_TxNIncidPwrVal`

Measured Tx incident power in dB. $1LSB = 0.1dBm$

- `xh_TxNReflPwrVal`

Measured Tx reflected power in dB. $1LSB = 0.1dBm$

- `w_FrameCount`

RFS frame count when the monitor report is updated.

2.5.10 FECSS TxN DC Signal Monitor Configuration API

This API configures the Tx DC-Signal monitor for all the Tx channels. Application should call this API multiple times to configure monitor for different Tx channels. The monitor measures set of DC signals in the Tx chain for the selected channel and compares the results with know thresholds. 1-bit pass/ fail result is stored in the calibration result for every signal.
The monitor can be triggered using FECSS Monitor Enable Trigger API and the response is stored in memory mapped response `T_RL_API_MON_TXN_DCSIG_RESULT` .

Note This monitor enables corresponding Tx channel with transmission enabled.

API Definition

API Index	0x0097U
API Function Name	rl_monTxNDcSigCfg
API Configuration Structure	T_RL_API_MON_TXN_DCSIG_CFG
API Response Structure	None
Monitor Response Structure	T_RL_API_MON_TXN_DCSIG_RESULT

API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_MON_TXn_DCSIG_CFG            ((UINT16)0x0097U)
// API Function Definition
T_RETURNTYPE rl_monTxNDcSigCfg(UINT8 c_devIndex, const T_RL_API_MON_TXN_DCSIG_CFG
↪  *p_apiCmdData);
```

Configuration C Structure

Table 2.77: T_RL_API_MON_TXN_DCSIG_CFG

Field	Data Type	Size	Offset
c_TxIndSel	UINT8	1	0
c_MonTxCodesSel	UINT8	1	1
h_MonTxBiasCodes	UINT16	2	2
h_RfFreqStart	UINT16	2	4
xh_RfFreqSlope	SINT16	2	6
w_Reserved1	UINT32	4	8
w_Reserved2	UINT32	4	12
Total		**16**	

- Response C Structure

Table 2.78: T_RL_API_MON_TXN_DCSIG_RESULT

Field	Data Type	Size	Offset
xh_ErrorCode	SINT16	2	0
h_TxDcMonStatus	UINT16	2	2
w_Reserved1	UINT32	4	4
w_Reserved2	UINT32	4	8
w_FrameCount	UINT32	4	12
Total		**16**	

Configuration Fields

- c_TxIndSel

Index of the Tx channel for which the configuration is applied.

Value	Definition
0	The Monitor configuration will be applicable for TX0.
1	The Monitor configuration will be applicable for TX1.
2 to 255	Reserved

Note Tx channel will be enabled irrespective of the h_TxChCtrlBitMask if corresponding monitor is enabled in w_MonitorEnable.

- c_MonTxCodesSel

Tx monitor code select control

Bit Field	Definition
Bit[0]	Reserved
Bit[1]	TX_PA_CODE override enable. • Value 0: Use calibrates TX_PA_CODE • Value 1: Use TX_PA_CODE from h_MonTxBiasCodes.
Bit[7:2]	Reserved

- h_MonTxBiasCodes

Override value for TX_PA_CODES.

- `h_RfFreqStart`

Monitor chirp start frequency in low resolution.

RF Frequency	Resolution
60GHz	$1LSB = (3 \times APLL\_FREQ)/2^{10} \approx 1.172MHz$ Valid Range: 57 GHz to 64 GHz for APLL_FREQ = 400MHz: 0xBE00U (57GHz) to 0xD555U (64 GHz)* *Variant specific limitations in Release Notes
77GHz	$1LSB = (4 \times APLL\_FREQ)/2^{10} \approx 1.562MHz$ Valid Range: 76 GHz to 81 GHz for APLL_FREQ = 400MHz: 0x0000BE00 (76GHz) to 0x0000CA80 (81GHz)

- `xh_RfFreqSlope`

RF Slope of the monitoring chirp.

RF Slope	Resolution
60GHz	$1LSB = (3 \times 400 \times APLL\_FREQ)/2^{24} \approx 28.610kHz/us$,signed number Valid Range: -3495 to 3495 (+/- 100MHz/us for APLL_FREQ = 400MHz)
77GHz	$1LSB = (4 \times 400 \times APLL\_FREQ)/2^{24} \approx 38.147kHz/us$, signed number Valid Range: -3495 to 3495 (+/- 100MHz/us for APLL_FREQ = 400MHz)

Warning Application should ensure that the Monitor chirp's end frequency is within the operating range of a respective device.

Response Fields

Monitor response is stored in IPC memory for each Tx channel. Refer `T_RL_FE_RFS_IPC_MON_RESULT_DATA_STRUCT` for the memory addresses.

- `xh_ErrorCode`

Monitor error code. 2's complement signed number.

Value	Definition
0	Success
2's complement negative number	Failure

- `h_TxDcMonStatus`

The measured TX supply GPADC DC signals for selected monitor configuration is compared with internal thresholds and result status (Pass/fail) are stored for each signal.

Bit Field	Definition
Bit[0]	TXn_VDDA_SUPPLY signal Status
Bit[1]	TXn_1V_CLUSTER_SUPPLY signal Status
Bit[15:2]	Reserved

- w_FrameCount

RFS frame count when the monitor report is updated.

2.5.11 FECSS Rx HPF and DC Signal Monitor Configuration API

This API configures the Rx DC-Signal and HPF monitor. The DC-signal monitor measures set of DC signals from all the Rx chains and compares the results with know thresholds. 1-bit pass/ fail result is stored in the calibration result for every signal. HPF monitor enables IFA loopback and measures tone power at in-band IF frequency and selected HPF's cutoff IF frequency.
The monitor can be triggered using FECSS Monitor Enable Trigger API and the response is stored in memory mapped response T_RL_API_MON_RX_HPF_DCSIG_RESULT .

API Definition

API Index	0x0098U
API Function Name	rl_monRxHpfDcSigCfg
API Configuration Structure	T_RL_API_MON_RX_HPF_DCSIG_CFG
API Response Structure	None
Monitor Response Structure	T_RL_API_MON_RX_HPF_DCSIG_RESULT

- API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_MON_RX_HPF_DCSIG_CFG        ((UINT16)0x0098U)
// API Function Definition
    T_RETURNTYPE rl_monRxHpfDcSigCfg(UINT8 c_devIndex, const T_RL_API_MON_RX_HPF_DCSIG_CFG
    ↪  *p_apiCmdData);
```

- Configuration C Structure

Table 2.79: T_RL_API_MON_RX_HPF_DCSIG_CFG

Field	Data Type	Size	Offset
h_RfFreqStart	UINT16	2	0
c_MonEnaCtrl	UINT8	1	2
c_RxHpfSel	UINT8	1	3
w_Reserved1	UINT32	4	4
w_Reserved2	UINT32	4	8
Total		**12**	

- Response C Structure

Table 2.80: T_RL_API_MON_RX_HPF_DCSIG_RESULT

Field	Data Type	Size	Offset
xh_ErrorCode	SINT16	2	0
h_Reserved1	UINT16	2	2
w_RxDcMonStatus	UINT32	4	4
xh_RxHpfInBandPwr	SINT16[4]	8	8
xh_RxHpfCutOffPwr	SINT16[4]	8	16
w_Reserved2	UINT32	4	24
w_Reserved3	UINT32	4	28
w_FrameCount	UINT32	4	32
Total		**36**	

Configuration Fields

- h_RfFreqStart

Monitor chirp start frequency in low resolution.

RF Frequency	Resolution
60GHz	$1LSB = (3 \times APLL\_FREQ)/2^{10} \approx 1.172MHz$ Valid Range: 57 GHz to 64 GHz for APLL_FREQ = 400MHz: 0xBE00U (57GHz) to 0xD555U (64 GHz)* *Variant specific limitations in Release Notes
77GHz	$1LSB = (4 \times APLL\_FREQ)/2^{10} \approx 1.562MHz$ Valid Range: 76 GHz to 81 GHz for APLL_FREQ = 400MHz: 0x0000BE00 (76GHz) to 0x0000CA80 (81GHz)

- `c_MonEnaCtrl`

Monitor enable control.

Value	Definition
0	Disabled
1	Enabled

Bit Field Definition:

Value	Definition
Bit[0]	`RX_DC_MON` enable bit. A monitor chirp is triggered to measure the DC signals using GPADC.
Bit[1]	`HPF_CUTOFF_MON` enable bit. • Two monitor chirps with IFA loopback are triggered to measure the powers at in-band frequency and HPF cutoff frequency. • The first chirp for the in-band power measurement uses an IF signal frequency of 2.77MHz. • The second chirp for the HPF cutoff power measurement uses an IF signal frequency matching the user programmed HPF corner frequency.

- `c_RxHpfSel`

The `HPF_CUTOFF_MON` monitor Chirp HPF corner frequency.

Value	Definition
0	175kHz HPF corner frequency
1	350kHz HPF corner frequency
2	700kHz HPF corner frequency
3	1400kHz HPF corner frequency

Valid Range: 0 to 3.

Response Fields

Monitor response is stored in IPC memory. Refer T_RL_FE_RFS_IPC_MON_RESULT_DATA_STRUCT for the memory addresses.

- xh_ErrorCode

Monitor error code. 2's complement signed number.

Value	Definition
0	Success
2's complement negative number	Failure

- w_RxDcMonStatus

ALl measured RX channels supply and bias GPADC DC signals for selected monitor configuration is compared with internal thresholds and result (pass/fail) will be sent out for each signal type.
DC-BIST results are valid only for enabled RX channels in h_RxChCtrlBitMask and RX_DC_MON = 1 in configuration API.

Bit Field	Definition
Bit [0]	RX0_ADC_I_DIG_LDO_VOUT_SENSE status
Bit [1]	RX0_ADC_I_DIG_LDO_VIN_SENSE status
Bit [2]	RX0_IFA_I_LDO_VOUT_SENSE status
Bit [3]	RX0_IFA_I_LDO_VIN_SENSE status
Bit [4]	RX0_VDD_LNA status
Bit [5]	LODIST_RX01_VDDA_1P0V_SUPPLY status
Bit [6]	RX1_ADC_I_DIG_LDO_VOUT_SENSE status
Bit [7]	RX1_ADC_I_DIG_LDO_VIN_SENSE status
Bit [8]	RX1_IFA_I_LDO_VOUT_SENSE status
Bit [9]	RX1_IFA_I_LDO_VIN_SENSE status
Bit [10]	RX1_VDD_LNA status
Bit [11]	RX2_ADC_I_DIG_LDO_VOUT_SENSE status
Bit [12]	RX2_ADC_I_DIG_LDO_VIN_SENSE status
Bit [13]	RX2_IFA_I_LDO_VOUT_SENSE status
Bit [14]	RX2_IFA_I_LDO_VIN_SENSE status
Bit [15]	RX2_VDD_LNA status
Bit [16]	LODIST_RX23_VDDA_1P0V_SUPPLY status
Bit [31:17]	Reserved status

- xh_RxHpfInBandPwr

Array of measured tone power for in-band IFA frequency measured in the Rx HPF monitor per Rx channel. $1LSB = 0.1dB$.
DC-BIST results are valid only for enabled RX channels in h_RxChCtrlBitMask and HPF_CUTOFF_MON = 1 in configuration API.

- xh_RxHpfCutOffPwr

Array of measured tone power for cutoff IFA frequency measured in the Rx HPF monitor per Rx channel. $1LSB = 0.1dB$.
DC-BIST results are valid only for enabled RX channels in h_RxChCtrlBitMask and HPF_CUTOFF_MON = 1 in configuration API.

- w_FrameCount

RFS frame count when the monitor report is updated.

2.5.12 FECSS PM and Clock DC Signals Monitor Configuration API

This API configures the PM-clock DC-Signal monitorThe monitor measures set of DC signals in the Power Management and Clock modules and compares the results with know thresholds. 1-bit pass/ fail result is stored in the calibration result for every signal.
The monitor can be triggered using FECSS Monitor Enable Trigger API and the response is stored in memory mapped response T_RL_API_MON_PMCLK_DCSIG_RESULT .

API Definition

API Index	0x0099U
API Function Name	rl_monPmClkDcSigCfg
API Configuration Structure	T_RL_API_MON_PMCLK_DCSIG_CFG
API Response Structure	None
Monitor Response Structure	T_RL_API_MON_PMCLK_DCSIG_RESULT

- API Function

mmWaveLink C function declaration

```
// API Index Macro
#define M_RL_API_ID_MON_PMCLK_DCSIG_CFG          ((UINT16)0x0099U)
// API Function Definition
    T_RETURNTYPE rl_monPmClkDcSigCfg(UINT8 c_devIndex, const T_RL_API_MON_PMCLK_DCSIG_CFG
    ↪  *p_apiCmdData);
```

- Configuration C Structure

Table 2.81: T_RL_API_MON_PMCLK_DCSIG_CFG

Field	Data Type	Size	Offset
h_RfFreqStart	UINT16	2	0
h_Reserved1	UINT16	2	2
w_Reserved2	UINT32	4	4
w_Reserved3	UINT32	4	8
Total		12	

- Response C Structure

Table 2.82: T_RL_API_MON_PMCLK_DCSIG_RESULT

Field	Data Type	Size	Offset
xh_ErrorCode	SINT16	2	0
h_Reserved1	UINT16	2	2
w_PmClkDcMonStatus	UINT32	4	4
c_VRef1Val	UINT8	1	8
c_VRef2Val	UINT8	1	9
c_VRef3Val	UINT8	1	10
c_VRef4Val	UINT8	1	11
w_Reserved2	UINT32	4	12
w_Reserved3	UINT32	4	16
w_FrameCount	UINT32	4	20
Total		**24**	

Configuration Fields

- h_RfFreqStart

Monitor chirp start frequency in low resolution.

RF Frequency	Resolution
60GHz	$1LSB = (3 \times APLL\_FREQ)/2^{10} \approx 1.172MHz$ Valid Range: 57 GHz to 64 GHz for APLL_FREQ = 400MHz: 0xBE00U (57GHz) to 0xD555U (64 GHz)* *Variant specific limitations in Release Notes
77GHz	$1LSB = (4 \times APLL\_FREQ)/2^{10} \approx 1.562MHz$ Valid Range: 76 GHz to 81 GHz for APLL_FREQ = 400MHz: 0x0000BE00 (76GHz) to 0x0000CA80 (81GHz)

Response Fields

Monitor response is stored in IPC memory. Refer T_RL_FE_RFS_IPC_MON_RESULT_DATA_STRUCT for the memory addresses.

- xh_ErrorCode

Monitor error code. 2's complement signed number.

Value	Definition
0	Success
2's complement negative number	Failure

- `w_PmClkDcMonStatus`

`PM_CLK_DC_MON` status.
The measured all PM, CLK supply and bias GPADC DC signals for selected monitor configuration is compared with internal thresholds and result (pass/fail) will be sent out for each signal type.

Value	Definition
Bit [0]	`PM_TOP_RF_LDO_1P2V_1P0V_VIN` status
Bit [1]	`PM_TOP_RF_LDO_1P2V_1P0V_VOUT` status
Bit [2]	`PM_TOP_VBG1P22V` status
Bit [3]	`CLK_TOP_VIN_1P8VCO_SENSE_SCALED_0P5` status
Bit [4]	`CLK_TOP_VIN_1P8CLK_SENSE_SCALED_0P5` status
Bit [5]	`CLK_TOP_VDDA_LDO_CLKTOP_IOBUF_1P4V_APLL_SCALED_0P6` status
Bit [6]	`CLK_TOP_VDDA_LDO_APLL_VCO_1P4V_SCALED_0P6` status
Bit [7]	`CLK_TOP_VDDA_LDO_APLL_CP_1P0_SCALED_0P6` status
Bit [8]	`CLK_TOP_VDDD_LDO_SDM_1P4V_SCALED_0P6` status
Bit [9]	`CLK_TOP_VDDA_LDO_SYNTH_VCO_SCALED_0P6` status
Bit [10]	`CLK_TOP_VDDA_LDO_SYNTH_DIV_SCALED_0P6` status
Bit [11]	`CLK_TOP_VDDA_LDO_SLICER_1P4V_SCALED_0P6` status
Bit [12]	`LODIST_COM_VDDA_1P0V_SUPPLY` status
Bit [13]	`LODIST_COM_LO_MULT_LDO_VOUT_SENSE_LOWV` status
Bit [14]	`PM_TOP_VREF0P45` status
Bit [15]	`PM_TOP_VFB_0P6V` status
Bit [16]	`PM_TOP_VREF_0P9V` status
Bit [17]	`PM_TOP_SCALED_VIOIN` status
Bit [18]	`CLK_TOP_SYNTH_DIVIDER_COMMON_MODE` status
Bit [19]	`PM_TOP_DBC_REFSYS` status
Bit [31:20]	`Reserved status` status

- `c_VRef1Val`

The REF1_0P45V reference voltage GPADC measurement value. 1LSB = 1.8V/256

- `c_VRef2Val`

The REF2_0P6V reference voltage GPADC measurement value. 1LSB = 1.8V/256

- `c_VRef3Val`

The REF3_0P9V reference voltage GPADC measurement value. 1LSB = 1.8V/256.

- `c_VRef4Val`

The REF4_VION_IN reference voltage GPADC measurement value. 1LSB = 1.8V/256.

- `w_FrameCount`

RFS frame count when the monitor report is updated.

2.6 mmWaveLink Error Codes

mmWaveLink APIs use data type `T_RETURNTYPE` to return the error code. Negative value of the `T_RETURNTYPE` indicates error in the API execution and 0 indicates success. This section describes all the error codes supported by mmWaveLink APIs.

mmWaveLink Error Codes are grouped in following categories. Categories are assigned with a non-overlapping range of values for ease of use. Application should define error code **excluding range** $[-2000, 0]$ to avoid overlap with mmWaveLink error codes.

- mmWaveLink Interface Error Codes
- mmWaveLink Device Error Codes
- mmWaveLink Sensor Error Codes
- mmWaveLink Monitor Error Codes
- mmWaveLink RFS API Error Codes
- mmWaveLink RFS Monitor Error Codes

2.6.1 mmWaveLink Interface Error Codes

This group defines the errors associated with mmWaveLink and application interface.
Range: $[-100, 0]$

T_RETURNTYPE	mmWaveLink Error Code and Description
0	M_DFP_RET_CODE_OK No-Error (Success)
0	M_DFP_OSI_RET_CODE_OK No-Error (Success)
0	M_DFP_CIF_RET_CODE_OK No-Error (Success)
-1	M_DFP_RET_CODE_INVALID_INPUT Invalid input - General
-2	M_DFP_RET_CODE_RESP_TIMEOUT API response timeout
-3	M_DFP_RET_CODE_FATAL_ERROR Fatal error in DFP
-4	M_DFP_RET_CODE_RADAR_OSIF_ERROR OS interface failure
-5	M_DFP_RET_CODE_INVALID_API Invalid API call
-6	M_DFP_RET_CODE_NULL_PTR_ERROR NULL pointer error

T_RETURNTYPE	mmWaveLink Error Code and Description
-7	M_DFP_RET_CODE_INTERFACE_CB_NULL Interface callback is NULL
-8	M_DFP_RET_CODE_INVALID_DEVICE Invalid device map/id
-9	M_DFP_RET_CODE_RADAR_PLTIF_ERROR Platform interface failure
-10	M_DFP_RET_CODE_RADAR_CIF_ERROR Communication interface failure
-11	M_DFP_RET_CODE_MMWL_INIT_NOT_DONE mmWaveLink Init not done
-12	M_DFP_RET_CODE_RFS_DBG_BUF_CB_NULL RFS debug buffer is NULL
[-100:-13]	Reserved

2.6.2 mmWaveLink Device Error Codes

This group defines the errors associated with mmWaveLink Device Configuration APIs.
Range: $[-300, -101]$

T_RETURNTYPE	mmWaveLink Error Code and Description
-101	M_DFP_RET_CODE_RFS_BOOT_ERROR RFS boot failure
-102	M_DFP_RET_CODE_RFS_MB_INIT_ERROR RFS MB Init failure
-103	M_DFP_RET_CODE_RFS_MB_BUSY_ERROR RFS MB busy while sending command
-104	M_DFP_RET_CODE_RFS_MB_RESP_ERROR RFS MB empty while reading result
-105	M_DFP_RET_CODE_RFS_PROTOCOL_ERROR RFS Protocol error
-106	M_DFP_RET_CODE_RFS_API_SIZE_ERR RFS cmd/res API size error
-107	M_DFP_RET_CODE_DEVICE_NOT_POWERED FECSS device is not powered up
-108	M_DFP_RET_CODE_REG_READBACK_ERROR Register readback error

T_RETURNTYPE	mmWaveLink Error Code and Description
-109	M_DFP_RET_CODE_RFS_RESP_TIMEOUT RFS Response timeout
-110	M_DFP_RET_CODE_RFS_CMDID_MISMATCH RFS cmd id mismatch
-111	M_DFP_RET_CODE_MEM_INIT_TIMEOUT Device Mem init failure
-112	M_DFP_RET_CODE_RFS_BOOT_TIMEOUT RFS boot timeout error
-113	M_DFP_RET_CODE_INVAL_DEV_PWR_MODE Invalid Device power mode
-114	M_DFP_RET_CODE_INVAL_DEV_CLK_SRC Invalid Device clock source
-115	M_DFP_RET_CODE_INVAL_FT_CLK_SRC Invalid FT clock source
-116	M_DFP_RET_CODE_INVAL_XTAL_CLK_SRC Invalid XTAL clock
-117	M_DFP_RET_CODE_INVAL_CT_RESOL Invalid CT resolution
-118	M_DFP_RET_CODE_FEC_POWERUP_TIMEOUT FECSS powerup timeout
-119	M_DFP_RET_CODE_FEC_PWRDOWN_TIMEOUT FECSS power down timeout
-120	M_DFP_RET_CODE_INVAL_APLL_CLK_CTRL Invalid APLL clock ctrl

2.6.3 mmWaveLink Sensor Error Codes

This group defines the errors associated with mmWave Sensor APIs
Range: $[-500, -301]$

T_RETURNTYPE	mmWaveLink Error Code and Description
-301	M_DFP_RET_CODE_SENS_INVAL_SAMP_RATE Invalid sample rate
-302	M_DFP_RET_CODE_SENS_INVAL_DIG_BITS Invalid Digout bits
-303	M_DFP_RET_CODE_SENS_INVAL_DFEFIR_SEL Invalid DFE FIR select

T_RETURNTYPE	mmWaveLink Error Code and Description
-304	M_DFP_RET_CODE_SENS_INVAL_VCO_MC_SEL Invalid VCO Multichip select
-305	M_DFP_RET_CODE_SENS_INVAL_NUMADC_SAMP Invalid num ADC samples
-306	M_DFP_RET_CODE_SENS_INVAL_TX_MIMO Invalid TX MIMO pattern
-307	M_DFP_RET_CODE_SENS_INVAL_RAMPEND Invalid rampend time
-308	M_DFP_RET_CODE_SENS_INVAL_HPF_SEL Invalid HPF select
-309	M_DFP_RET_CODE_SENS_INVAL_RX_GAIN Invalid RX gain select
-310	M_DFP_RET_CODE_SENS_INVAL_RF_GAIN Invalid RF gain select
-311	M_DFP_RET_CODE_SENS_INVAL_TX_BO Invalid TX Back off
-312	M_DFP_RET_CODE_SENS_INVAL_IDLE_TIME Invalid idle time
-313	M_DFP_RET_CODE_SENS_INVAL_ADCFRAC_TIME Invalid adc fract time
-314	M_DFP_RET_CODE_SENS_INVAL_ADCSKIP_SAMP Invalid adc skip samples
-315	M_DFP_RET_CODE_SENS_INVAL_TXSTART_TIME Invalid TX start time
-316	M_DFP_RET_CODE_SENS_INVAL_RF_SLOPE Invalid RF slope
-317	M_DFP_RET_CODE_SENS_INVAL_RF_FREQ Invalid RF Frequency
-318	M_DFP_RET_CODE_SENS_INVAL_TX_EN Invalid TX enable
-319	M_DFP_RET_CODE_SENS_INVAL_TXBPM_EN Invalid TX BPM enable
-320	M_DFP_RET_CODE_SENS_INVAL_NUM_CHIRPS Invalid Num of chirps
-321	M_DFP_RET_CODE_SENS_INVAL_NUM_ACCUM Invalid Num of accumulation

T_RETURNTYPE	mmWaveLink Error Code and Description
-322	M_DFP_RET_CODE_SENS_INVAL_BURST_PERD Invalid Burst period
-323	M_DFP_RET_CODE_SENS_INVAL_NUM_BURST Invalid Num of bursts
-324	M_DFP_RET_CODE_SENS_INVAL_FRAME_PERD Invalid Frame period
-325	M_DFP_RET_CODE_SENS_INVAL_NUM_FRAME Invalid num of frames
-326	M_DFP_RET_CODE_SENS_INVAL_FRAME_TRIG Invalid frame trigger mode
-327	M_DFP_RET_CODE_SENS_INVAL_CT_START_LB Invalid frame signal LB enable
-328	M_DFP_RET_CODE_SENS_INVAL_FRAME_STOP Invalid frame stop
-329	M_DFP_RET_CODE_SENS_INVAL_PERCHRP_LEN Invalid Per chirp array length
-330	M_DFP_RET_CODE_SENS_INVAL_PERCHRP_RPCNT Invalid per chirp repeat count
-331	M_DFP_RET_CODE_SENS_INVAL_PERCHRP_CTRL Invalid per chirp control
-332	M_DFP_RET_CODE_SENS_INVAL_PERCHRP_ADDR Invalid per chirp address
-333	M_DFP_RET_CODE_SENS_INVAL_HPF_FI_TIME Invalid HPF Fast Init TIme
-334	M_DFP_RET_CODE_SENS_INVAL_CRD_NSLOPE Invalid CRD Nslope
-335	M_DFP_RET_CODE_SENS_INVAL_LIVE_MON_CFG Inavlid Live Mon cfg
-336	M_DFP_RET_CODE_SENS_INVAL_LB_FREQ Inavlid LB freq

2.6.4 mmWaveLink Monitor Error Codes

This group defines the errors associated with mmWaveLink Monitor Configuration APIs.
Range: $[-700, -501]$

T_RETURNTYPE	mmWaveLink Error Code and Description
-501	M_DFP_RET_CODE_MON_INVAL_SYNTH_START Invalid SYNTH mon start time
-502	M_DFP_RET_CODE_MON_INVAL_SYNTH_THRSH Invalid SYNTH mon threshold
-503	M_DFP_RET_CODE_MON_INVAL_PERCHRP_ADDR Invalid Rx Sat mon result add
-504	M_DFP_RET_CODE_MON_INVAL_GPADC_INST Invalid GPADC num instructions
-505	M_DFP_RET_CODE_MON_INVAL_TX_IND Invalid TX index select

2.6.5 mmWaveLink RFS API Error Codes

This group defines error codes returned by the RFS firmware for mailbox APIs. Range: $[-1500, -1000]$

T_RETURNTYPE	mmWaveLink Error Code and Description
-1000	M_DFP_RET_CODE_RFS_INVALID_CMDID
-1002	M_DFP_RET_CODE_RFS_INVALID_MSG_LEN
-1003	M_DFP_RET_CODE_RFS_INVALID_RX_MASK
-1004	M_DFP_RET_CODE_RFS_INVALID_TX_MASK
-1005	M_DFP_RET_CODE_RFS_INVALID_APLL_CTRL
-1006	M_DFP_RET_CODE_RFS_INVALID_RF_CTRL_OVERRIDE_MASK
-1007	M_DFP_RET_CODE_RFS_INVALID_CAL_TEMP_BIN
-1008	M_DFP_RET_CODE_RFS_INVALID_CAL_FREQ
-1009	M_DFP_RET_CODE_RFS_INVALID_CAL_FREQ_SLOPE
-1010	M_DFP_RET_CODE_RFS_INVALID_CAL_EN_MASK
-1011	M_DFP_RET_CODE_RFS_INVALID_CHIRP_PS_DIS_MASK
-1012	M_DFP_RET_CODE_RFS_INVALID_BURST_PS_DIS_MASK
-1013	M_DFP_RET_CODE_RFS_INVALID_RFS_SENS_START_MODE
-1014	M_DFP_RET_CODE_RFS_INVALID_RFS_SENS_STOP_MODE
-1015	M_DFP_RET_CODE_RFS_INVALID_SENS_LB_CTRL
-1016	M_DFP_RET_CODE_RFS_INVALID_SENS_LB_ENA
-1017	M_DFP_RET_CODE_RFS_GPADC_BUSY

T_RETURNTYPE	mmWaveLink Error Code and Description
-1018	M_DFP_RET_CODE_RFS_INVALID_RDIF_ENA_CTRL
-1019	M_DFP_RET_CODE_RFS_INVALID_RDIF_SAMP_COUNT
-1020	M_DFP_RET_CODE_RFS_INVALID_ORBIT_RECORD
-1021	M_DFP_RET_CODE_RFS_APLL_ON_FAILED
-1022	M_DFP_RET_CODE_RFS_APLL_OFF_IN_CAL
-1023	M_DFP_RET_CODE_RFS_INVALID_TESTPATRN_ENA_CTRL
-1024	M_DFP_RET_CODE_RFS_INVALID_TEMP_MEAS_CHANNEL
-1025	M_DFP_RET_CODE_RFS_LOOPBACK_ACTIVE
-1026	M_DFP_RET_CODE_RFS_GPADC_IFM_CONV_ERROR
-1027	M_DFP_RET_CODE_RFS_GPADC_IFM_INVAL_SAMPLS
-1028	M_DFP_RET_CODE_RFS_GPADC_IFM_INVAL_BUF_INDX
-1029	M_DFP_RET_CODE_RFS_INVALID_MON_EN_MASK
-1030	M_DFP_RET_CODE_RFS_INVALID_MON_FAULT_MASK
-1031	M_DFP_RET_CODE_RFS_MON_MANAGER_BUSY
-1032	M_DFP_RET_CODE_RFS_INVALID_DEBUG_DATA_ADDRESS
-1033	M_DFP_RET_CODE_RFS_INVALID_TX_INDEX
-1034	M_DFP_RET_CODE_RFS_INVALID_PD_INDEX
-1035	M_DFP_RET_CODE_RFS_INVALID_SUM_SAMPLES
-1036	M_DFP_RET_CODE_RFS_INVALID_PD_LNA_INDEX
-1037	M_DFP_RET_CODE_RFS_GPADC_CTM_MEM_INIT_TIMEOUT
-1038	M_DFP_RET_CODE_RFS_INVALID_BOOT_CAL_DATA

2.6.6 mmWaveLink RFS Monitor Error Codes

This group defines the errors associated with RFS monitors and calibrations. These errors could be reported in the monitor results (e.g. xh_ErrorCode) and calibration APIs. Range: $[-1099, -1500]$

T_RETURNTYPE	mmWaveLink Error Code and Description
-1500	M_CALMON_ERROR_FFT_CHIRP_MEAS_FAILED
-1501	M_CALMON_ERROR_PD_MEAS_FAILED
-1502	M_CAL_ERROR_RXGAIN_OPTIMAL_PGA_CFG_FAILED

T_RETURNTYPE	mmWaveLink Error Code and Description
-1503	M_CAL_ERROR_RXGAIN_RF_GAIN_THRES_CHK_FAILED
-1504	M_CALMON_ERROR_SETUP_FAILED
-1505	M_CALMON_ERROR_DCBIST_MEAS_FAILED
-1506	M_CALMON_ERROR_INVALID_HPF_SEL

Chapter 3
Execution Time and Constraints

This sections describes API and important event execution timings. The section also includes timing restrictions which must be honored by the application across different APIs. mmWaveLink APIs may not report error for violating these settings. These values are also device specific.

3.1 Chirp, Burst and Frame Timing Constraints

Chirp timing parameters like chirp idle time, skip samples, and chirp ramp end time, along with parameters such as burst and frame times are to be set according to conditions mentioned in this section. Hardware functionality can be affected if these conditions are violated.

3.1.1 Burst Timing Constraints

Table 3.1: Burst Timing Constraints

Variable	Description	xWRL6432 xWRL1432
Burst Start Time $BurstStartTime$	Time required for RFS firmware to process the Burst Start event	65 us
Burst End Time $BurstEndTime$	Time required for RFS firmware to process the Burst End event	50 us
Minimum Burst Idle Time $MinBurstIdleTime$	$BurstStartTime + BurstEndTime$	115us

3.1.2 Frame Timing Constraints

Table 3.2: Frame Timing Constraints

Variable	Description	xWRL6432 xWRL1432
Frame Start Time $FrameStartTime$	Time required for RFS firmware to process the Frame Start event	25 us
Frame End Time $FrameEndTime$	Time required for RFS firmware to process the Frame End event	15 us
Minimum Frame Idle Time $MinFrameIdleTime$	$FrameStartTime + FrameEndTime$	40 us

3.1.3 Chirp Idle Time Constraints

Test

Table 3.3: Minimum Chirp Idle-Time Constraints

Inter-Chirp Power Save Configuration	Minimum Inter-Chirp Idle-Time
Power Save Enabled c_InterChirpPsDis = 0x00	$max(6us - \texttt{xh\_ChirpTxStartTime}, 3.1us)$ Example: • xh_ChirpTxStartTime = 4us • h_ChirpIdleTime = 3.1us
Power Save Disabled c_InterChirpPsDis = 0x1C	$max(4us - \texttt{xh\_ChirpTxStartTime}, 3.1us)$ Example: • xh_ChirpTxStartTime = 0us • h_ChirpIdleTime = 4us

3.1.4 Skip Samples Constraints

Note The term "skip samples" refers to the number of RX ADC samples at the beginning of each chirp which are skipped (i.e. not collected for radar processing) by programming the hardware via h_ChirpAdcStartTime.

In each chirp, the initial samples exhibit settling behavior, which arises in the synthesizer PLL and RX baseband high-pass and low-pass filters (analog and digital). It is mandatory to program skip samples parameter (see h_ChirpAdcStartTime) high enough to wait out these (potentially abrupt) initial transients.
Specifically,

Table 3.4: Skip Samples Constraints

Chirp Start frequency		c_DfeFirSel	Recommended $AdcSkipSamples$
60GHz devices	77GHz devices		
[57.5, 64] GHz	[76.5, 81] GHz	0 (long filter)	$\geq 12 + max\left((1.5us \times Fs) + 4, (2us \times Fs)\right)$
		1 (short filter)	$\geq 8 + max\left((1.5us \times Fs) + 3, (2us \times Fs)\right)$
Otherwise	Otherwise	0 (long filter)	$\geq 12 + max\left((1.5us \times Fs) + 4, (3us \times Fs)\right)$
		1 (short filter)	$\geq 8 + max\left((1.5us \times Fs) + 3, (3us \times Fs)\right)$

Note 1.5us in the above table corresponds to the recommended HPF Fast Init Duration (see c_HpfFastInitDuration). The other parameters correspond to the group delay and settling delay of LPFs (which depend on the user's filter type selection) and synthesizer PLL settling delay (which depends on RF start frequency (see w_ChirpRfFreqStart)).

3.1.5 Chirp Ramp End Time Constraints

The user should program Ramp End Time higher than the time necessary for RX ADC sampling and filtering. The time necessary is a function of the sampling rate, F_s, in Hz (c_DigOutputSampRate), as well as the number of skip samples ($AdcSkipSamples$) and number of ADC samples (h_NumOfAdcSamples) as per the following table:

Table 3.5: Ramp End Time Constraints

RX sampling rate (F_s)	c_DfeFirSel	Ramp End Time (in s)
< 1.5 MHz	0 (long filter)	$\geq (AdcSkipSamples + \texttt{h\_NumOfAdcSamples} - 4)/F_s$
	1 (short filter)	
>= 1.5 MHz	0 (long filter)	$\geq (AdcSkipSamples + \texttt{h\_NumOfAdcSamples} - 8)/F_s$
	1 (short filter)	$\geq (AdcSkipSamples + \texttt{h\_NumOfAdcSamples} - 5)/F_s$

3.1.6 Chirp Periodicity Constraints

In setting the chirp timing parameters and sample counts, the user should ensure that the programmed chirp periodicity, i.e., Ramp End Time + Idle Time (set by fields: h_ChirpRampEndTime, h_ChirpIdleTime) always exceeds the time necessary for RX ADC sampling.
Specifically, ensure that $(RampEndTime + IdleTime) \geq$ (AdcSkipSamples + h_NumOfAdcSamples + 4) / F_s

Warning Chirp and burst timing restrictions must be satisfied for all the chirps while using per chirp LUTs

3.2 API Execution Time

3.2.1 mmWaveLink API Execution Time

Following tables list the worst case execution times for all the mmWaveLink APIs measured.

Note
- Execution time also depend on the implementation of call-back functions and Application interrupt processing latencies
- API timings are measured with AppSS and FECSS clocked using Fast Clock

Table 3.6: Execution Time for Initialization API timings:apiTimings:apis:init

mmWaveLink API	Execution Time (us)
mmWaveLink Initialization API	30
mmWaveLink De-Initialization API	5

Table 3.7: Execution Time for Device Control APIs timings:apiTimings:apis:init

mmWaveLink API	Execution Time (us)
FECSS Device Power On API	Cold Boot: 850 Warm Boot: 450
FECSS Device Power OFF API	125
DFP Version Get API	10
FECSS RF Power On Off API	20

mmWaveLink API	Execution Time (us)
FECSS Factory Calibration API	xWRL6432: 1500 xWRL1432: 1900 with h_CalCtrlBitMask = 0xCE
FECSS RF Runtime Calibration API	xWRL6432: 450 xWRL1432: 450 with h_CalCtrlBitMask = 0xCE
FECSS RF Clock Bandwidth Configuration API	20
FECSS GPADC Measurement Configuration API	10
FECSS GPADC Measurement Trigger API	40 (Configuration dependent)
FECSS Temperature Measurement Config API	5
FECSS Temperature Measurement Trigger API	65 h_TempCtrlBitMask = 0x311
FECSS Factory Calibration Data Get API	20
FECSS Factory Calibration Data Set API	15
FECSS Device Clock Ctrl API	APLL On: 125 APLL Cal: 280 APLL Off: 10
FECSS Device RDIF Control API	20
FECSS Device Status Get API	20
FECSS RF Status Get API	20
FECSS RF Fault Status Get API	10
FECSS Die Id Get API	5
FECSS Rx-Tx Calibration Data Get API	10
FECSS Rx-Tx Calibration Data Set API	10
FECSS RFS Debug Configuration API	20
FECSS RF Runtime Tx CLPC API	710 (per Tx channel)

Table 3.8: Execution Time for Sensor Configuration APIs timings:apiTimings:apis:init

mmWaveLink API	Execution Time (us)
Sensor Profile Common Configuration API	15

mmWaveLink API	Execution Time (us)
Sensor Chirp Profile Common Configuration Get API	15
Sensor Chirp Profile Time Configuration API	10
Sensor Chirp Profile Time Configuration Get API	10
Sensor Per-Chirp Configuration API	15
Sensor Per-Chirp Configuration Get API	15
Sensor Per-Chirp Control API	10
Sensor Per-Chirp Control Get API	10
Sensor Frame Configuration API	10
Sensor Frame Configuration Get API	10
Sensor Start API	10
Sensor Stop API	5
Sensor Status Get API	10
Sensor Dynamic Power Save Disable Configuration API	20
Sensor Dynamic Power Save Disable Configuration Get API	20
Sensor Loopback Configuration API	5
Sensor Loopback Enable API	20

Table 3.9: Execution Time for Monitor Configuration APIs timings:apiTimings:apis:init

mmWaveLink API	Execution Time (us)
FECSS Monitor Enable Trigger API	20
FECSS Peak-Detector Debug Monitor API	350 (Configuration dependent)
FECSS Tx Power Debug Monitor API	450 (Configuration dependent)
FECSS Synthesizer Frequency Live Monitor Configuration API	5
FECSS Rx Saturation Live Monitor Configuration API	5

mmWaveLink API	Execution Time (us)
FECSS PLL Control Voltage Monitor Configuration API	5
FECSS TxN-Rx Loopback Monitor Configuration API	5
FECSS TxN Power Monitor Configuration API	5
FECSS Tx Ball-Break Monitor Configuration API	5
FECSS TxN DC Signal Monitor Configuration API	5
FECSS Rx HPF and DC Signal Monitor Configuration API	5
FECSS PM and Clock DC Signals Monitor Configuration API	5

3.2.2 Calibration Execution Time

Table 3.10: Factory Calibration Execution Time

Calibration	Execution Time (us)	
	xWRL6432	xWRL1432
Synth	180	180
PD	250	250
LoDist	10	10
Rx IFA	425	800
Tx PA `c_TxPwrCalTxEnaMask` = 0x3, 0x1	650	650

Table 3.11: Runtime Calibration Execution Time

Calibration	Execution Time (us)	
	xWRL6432	xWRL1432
Synth	180	180
PD	250	250
LoDist	10	10
Rx IFA	2	2
Tx PA	2	2

3.2.3 Monitor Execution Time

Following table tabulates **Worst Case** monitoring time for all the monitors supported by mmWave devices.

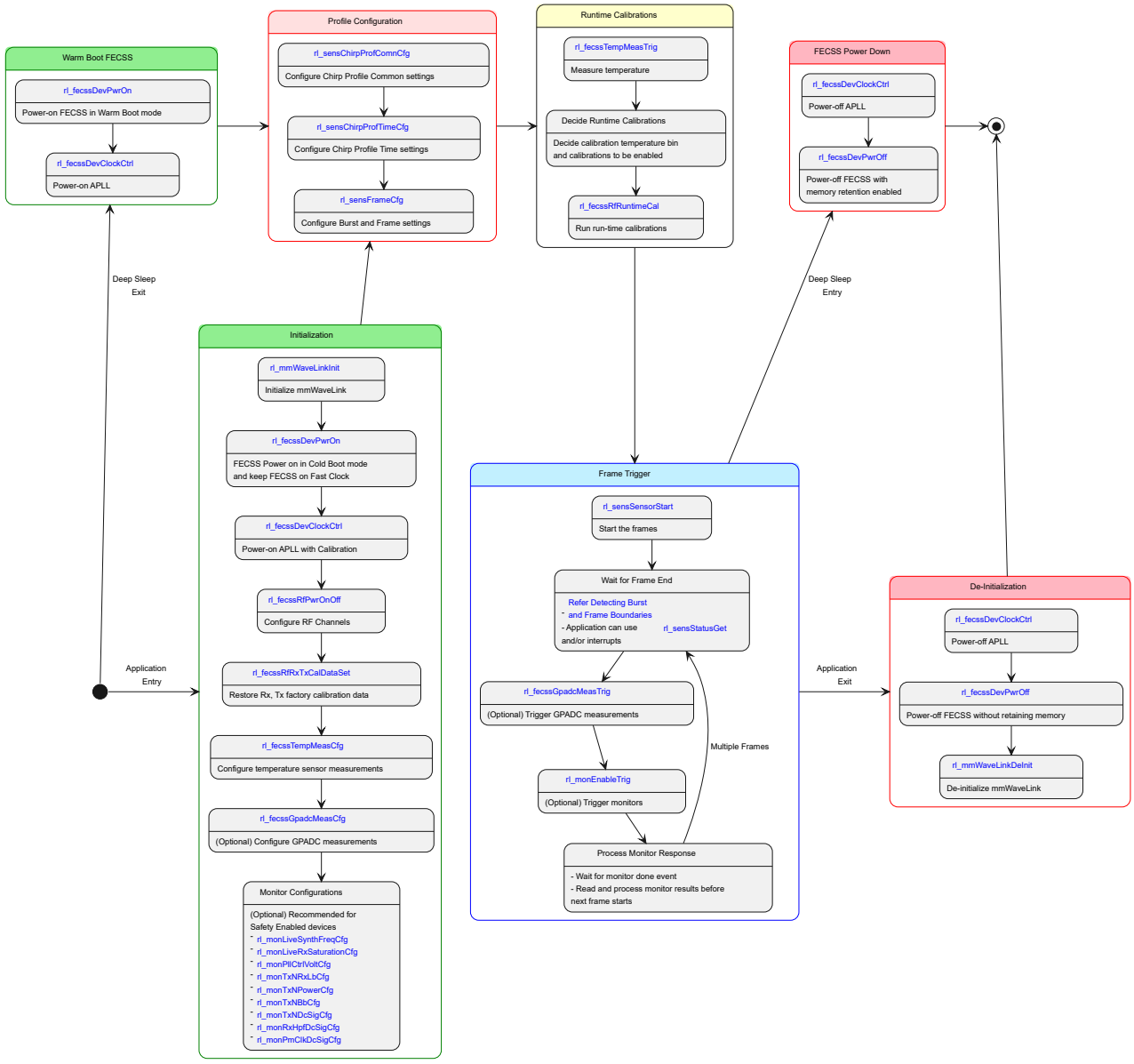
Note Monitoring durations are Configuration dependent

Monitor	Execution Time (us)	
	xWRL6432	xWRL1432
PLL V-Control	170	170
Tx-Rx Loopback	650	650
Tx Power (per Tx)	250	250
Tx Ball-Break (per Tx)	250	250
Tx DC Signal (per Tx)	250	250
Rx HPF and DC Signal	650	650
PM Clock DC Signal	250	250
DFE-FFT Monitor	300	300
Static Register Readback	50	50

4.1 In-Field Operation sequence

Following diagram describes a recommended mmWaveLink API sequence for in-field application which retains FECSS data memories across deep sleep cycles.

Figure 4.1: Recommended in-field operation API sequence



Notes:

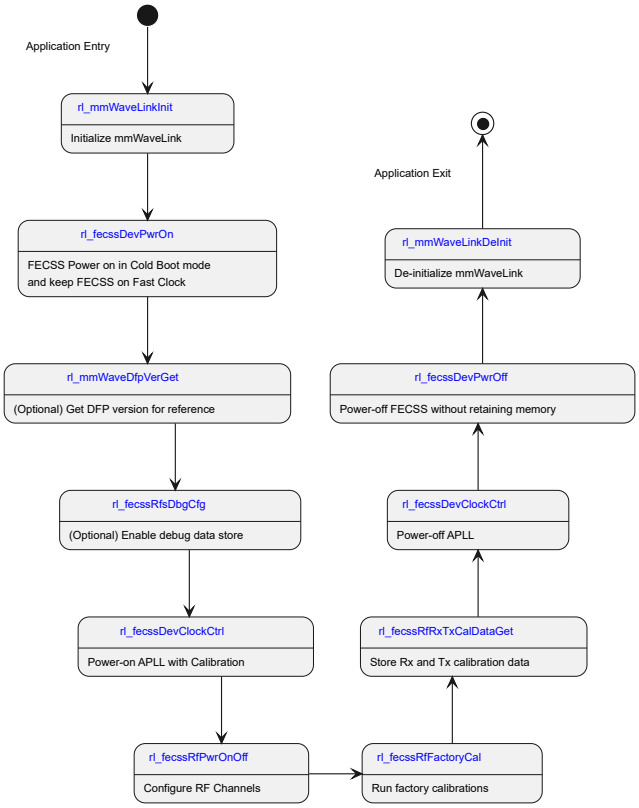
- DFP Version Get API and FECSS RFS Debug Configuration API could be used immediately after FECSS Device Power On API for debug

- Monitor config and FECSS Monitor Enable Trigger API are required only on Safety Enabled devices.

4.2 Factory Calibration sequence

Following diagram describes a recommended mmWaveLink API sequence for factory calibration application.

Figure 4.2: Recommended factory calibration API sequence



ICD Revision History

ICD Version 1.0

Release Date: 11th January, 2024
Description: xWRL6432 RTM Release
This is a baseline ICD for low end radar devices. All the API changes after this release will be tracked after this release.

ICD Version 1.1

Release Date: 11th June, 2024
Description: xWRL1432 RTM Release
Changes: xWRL1432 RTM Release

- Updated default value for `c_SynthIcpTrim` from 0x1 to 0x5
- MMWAVE_DFP_LOW-588: Added more details on minimum ramp end time in Skip Samples Constraints
- Added device specific backoff ranges in `c_CalTxBackOffSel`.

ICD Version 1.2

Release Date: 10th July, 2025
Description: xWRL6432 AOP RTM Release
Changes: xWRL6432 AOP RTM Release

- Called out variant specific constraints where applicable (To be referred to using Release notes).
- MMWAVE_DFP_LOW-813: Updated Per-Chirp and MIMO priorities in `c_ChirpTxMimoPatSel`

Notice

Texas Instruments Incorporated (“TI”) technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, “TI Resources”) are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI’s provision of TI Resources does not expand or otherwise alter TI’s applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED “AS IS” AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your noncompliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI’s standard terms for semiconductor products (http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265