CEC/MEC Family Devices ROM API User's Guide

ISBN: 9781522411666

Table of Contents

Preface

NOTICE TO CUSTOMERS

All documentation becomes dated, and this manual is no exception. Microchip tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our web site (www.microchip.com) to obtain the latest documentation available.

Documents are identified with a "DS" number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is "DSXXXXXA", where "XXXXX" is the document number and "A" is the revision level of the document.

For the most up-to-date information on development tools, see the MPLAB® IDE online help. Select the Help menu, and then Topics to open a list of available online help files.

INTRODUCTION

This chapter contains general information that will be useful to know before using the CEC/MEC Family Devices ROM API, which include the following:

- CEC1702

- MEC170x Family Devices.

Items discussed in this chapter include:

- Document Layout
- Conventions Used in this Guide
- The Microchip Web Site
- Development Systems Customer Change Notification Service
- Customer Support
- Document Revision History

DOCUMENT LAYOUT

This document describes how to use the CEC/MEC Family Devices ROM API as a development tool with the CEC/MEC family devices. The manual layout is as follows:

- **Chapter 1. "Introduction"** - Explains purpose and scope of this guide.
- **Chapter 2. "Overall Description"** - Provides an overview of getting started with the CEC/MEC Family Devices ROM API.
- **Chapter 3. "External Interface Requirements"** - Provides external interface requirements for the CEC/MEC Family Devices ROM API.
- **Chapter 4. "Usage"** - Describes functions available.
- **Chapter 5. "API Usage"** - This section lists the APIs available and their usage.

- **Chapter 6. “Build and Link”** - Provides instructions for proper linking of application code with bootrom.
- **Chapter 7. “Timing Analysis”** - Provides results of timing measurement.
- **Chapter 8. “PKE Slot Usage”** - Provides a table which lists the usage of the slots for various operations.
- **Appendix A. “Rom Symdef Table for API Support”** - Provides a table for SPI support.

CONVENTIONS USED IN THIS GUIDE

This manual uses the following documentation conventions:

DOCUMENTATION CONVENTIONS

Description	Represents	Examples
Arial font:		
Italic characters	Referenced books	*MPLAB® IDE User’s Guide*
	Emphasized text	...is the *only* compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	select Enable Programmer
Quotes	A field name in a window or dialog	“Save project before build”
Underlined, italic text with right angle bracket	A menu path	*File>Save*
Bold characters	A dialog button	Click **OK**
	A tab	Click the **Power** tab
N‘Rnnnn	A number in verilog format, where N is the total number of digits, R is the radix and n is a digit.	4‘b0010, 2‘hF1
Text in angle brackets < >	A key on the keyboard	Press <Enter>, <F1>
Courier New font:		
Plain Courier New	Sample source code	`#define START`
	Filenames	`autoexec.bat`
	File paths	`c:\mcc18\h`
	Keywords	`_asm, _endasm, static`
	Command-line options	`-Opa+, -Opa-`
	Bit values	`0, 1`
	Constants	`0xFF, ‘A’`
Italic Courier New	A variable argument	*`file`*`.o`, where *`file`* can be any valid filename
Square brackets [ ]	Optional arguments	`mcc18 [options]` *`file`* `[options]`
Curly brackets and pipe character: { | }	Choice of mutually exclusive arguments; an OR selection	`errorlevel {0|1}`
Ellipses...	Replaces repeated text	`var_name [, var_name...]`
	Represents code supplied by user	`void main (void) { ... }`

THE MICROCHIP WEB SITE

Microchip provides online support via our web site at www.microchip.com. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

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

DEVELOPMENT SYSTEMS CUSTOMER CHANGE NOTIFICATION SERVICE

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

To register, access the Microchip web site at www.microchip.com, click on Customer Change Notification and follow the registration instructions.

The Development Systems product group categories are:

- **Compilers** – The latest information on Microchip C compilers, assemblers, linkers and other language tools. These include all MPLAB C compilers; all MPLAB assemblers (including MPASM assembler); all MPLAB linkers (including MPLINK object linker); and all MPLAB librarians (including MPLIB object librarian).
- **Emulators** – The latest information on Microchip in-circuit emulators.This includes the MPLAB REAL ICE and MPLAB ICE 2000 in-circuit emulators.
- **In-Circuit Debuggers** – The latest information on the Microchip in-circuit debuggers. This includes MPLAB ICD 3 in-circuit debuggers and PICkit 3 debug express.
- **MPLAB IDE** – The latest information on Microchip MPLAB IDE, the Windows Integrated Development Environment for development systems tools. This list is focused on the MPLAB IDE, MPLAB IDE Project Manager, MPLAB Editor and MPLAB SIM simulator, as well as general editing and debugging features.
- **Programmers** – The latest information on Microchip programmers. These include production programmers such as MPLAB REAL ICE in-circuit emulator, MPLAB ICD 3 in-circuit debugger and MPLAB PM3 device programmers. Also included are nonproduction development programmers such as PICSTART Plus and PIC-kit 2 and 3.

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or field application engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

DOCUMENT REVISION HISTORY

Revision	Section/Figure/Entry	Correction
DS50002517B (12-06-16)	Chapter 4, 5, 7 and 8	Changes to chapters 4 and 5; added chapters 7 and 8.
DS50002517A (06-14-16)	Initial document release	

Chapter 1. Introduction

1.1 PURPOSE

This document illustrates the usage of ROM APIs available in the CEC/MEC family.

1.2 SCOPE

This document will serve as a usage manual for the functions provided by the ROM. It presents the reader with the function header, a description of the function's operations and its input and output parameters. It also entails the pseudo code and steps to use the APIs provided in the CEC/MEC family ROM.

1.3 GLOSSARY OF TERMS AND ACRONYMS

SPI – Serial Peripheral Interface

AES – Advanced Encryption Standard

RSA - Rivest-Shamir-Adleman cryptosystem

PKE – Public Key Encryption

SHA – Secure Hash Algorithm

RNG – Random Number Generator

SCM – Shared Crypto Memory

CRT – Chinese Remainder Theorem

KCDSA – Korean Elliptic Curve Digital Signature Algorithm

ECDSA – Elliptic Curve Digital Signature Algorithm

EC25519 – Elliptic Curve 25519

Chapter 2. Overall Description

2.1 PRODUCT PERSPECTIVE

The Boot ROM API allows software access to certain hardware features that facilitates easy development of applications. These APIs serve the function of providing easy access to the underlying hardware features.

2.2 PRODUCT FUNCTIONS

The Boot ROM APIs provide software access features like access to SPI/FLASH, AES encryption, RSA Crypt engine, Public Key Encryption. All operations are abstracted by Application program interfaces and the programmer need only to use the APIs to leverage these device specific operations.

2.3 USER CLASSES AND CHARACTERISTICS

This document is intended for programmers. It illustrates the use of certain software features that would facilitate easy development of applications.

2.4 DESIGN AND IMPLEMENTATION CONSTRAINTS

The APIs are all ROM resident. They only use stack dynamic variables and internal reference/pointers (these constraints discount any pointers passed by the user to the functions). The APIs do not use heap dynamic or global space to store data.

Many of the APIs require buffers and memory to be specified. The onus of maintaining the proper buffers and memory is on the caller of the APIs.

2.5 ASSUMPTIONS AND DEPENDENCIES

The efficacy of this user manual may be contingent upon the knowledge of the target hardware. Certain references in this document may require the usage of device data sheets.

2.6 OPERATING ENVIRONMENT

The pseudo code provided is OS independent and can only be used with CEC/MEC family.

Chapter 3. External Interface Requirements

3.1 USER INTERFACES

This document will describe the user interface to the ROM resident APIs.

3.2 HARDWARE INTERFACES

CEC/MEC EVB

Keil µVision Ulink Pro Debugger tool –

MCHP Trace Debugger Tool

Dediprog SPI programmer

3.3 SOFTWARE INTERFACES

Keil Compiler IDE-Version:

µVision V5.15

Tool Version Numbers:

Toolchain: MDK-ARM Standard Cortex-M Version: 5.15.0

Toolchain Path: C:\Keil_v5\ARM\ARMCC\Bin

C Compiler: Armcc.exe V5.05 update 2 (build 169)

Assembler: Armasm.exe V5.05 update 2 (build 169)

Linker/Locator: ArmLink.exe V5.05 update 2 (build 169)

Library Manager: ArmAr.exe V5.05 update 2 (build 169)

Hex Converter: FromElf.exe V5.05 update 2 (build 169)

CPU DLL: SARMCM3.DLL V5.15.0

Dialog DLL: DCM.DLL V1.13.2.0

Target DLL: ULP2CM3.DLL V2.200.17.0

Dialog DLL: TCM.DLL V1.14.5.0

For sample code, please contact your Microchip representative for more information.

Chapter 4. Usage

4.1 QMSPI FUNCTIONS

Note 1: All blocks need to be powered ON with corresponding APIs before usage of any of the API for crypto operations.

2: Ensure that Input Capture and compare timer is activated and running which may be required by some of the crypto API.

Set bits 0,1 of Capture and Compare Timer Control register (at address 0x40001000).

4.1.1 spi_port_sel

Function Header:

void spi_port_sel (uint8_t port, uint8_t pin_mask, bool en);

Description:

This function controls SPI port control. It facilitates the selection of ports and offers enable/disable control. By selection of ports, the GPIO's and chip selects are configured as necessary.

If any port numbers other that the one's mentioned below are used, the function will not perform any operation.

Inputs:

Input Parameter	Description
Port	An 8 bit unsigned integer indicating port number. The permitted port numbers are * 0 (Port 0, External shared) * 1 (Port 1, external private (Recovery)) * 2 (Port 2, Internal).
Pin_mask	Specifies the pin(s) of the selected QMSPI port that needs to be modified b[0]=chip-select, b[1]=clock, b[2]=IO0, b[3]=IO1, b[4]=IO2, b[5]=IO3.
En	A boolean input. The permitted values are * 1 (Enable) * 0 (Disable)

Outputs:

None

4.1.2 spi_port_drv_slew

Function Header:

void spi_port_sel (uint8_t port, uint8_t pin_mask, bool en);

Description:

This function controls SPI port control. It facilitates the selection of ports and offers enable/disable control. By selection of ports, the GPIO's and chip selects are configured as necessary.

If any port numbers other that the one's mentioned below are used, the function will not perform any operation.

Inputs:

Input Parameters	Description
Port	An 8 bit unsigned integer indicating port number. The permitted port numbers are: * 0 (Port 0, External shared) * 1 (Port 1, external private (Recovery)) * 2 (Port 2, Internal).
Pin_mask	Specifies the pin(s) of the selected QMSPI port that needs to be modified b[0]=chip-select, b[1]=clock, b[2]=IO0, b[3]=IO1, b[4]=IO2, b[5]=IO3.
Drv_slew	An 8 bit unsigned integer indicating drv slew values. The permitted values for Drive strength and slew rate are Drive strength - 1 for 3.3V, 2 for 1.8V, Slew Rate - 0 = Slow, 1 = Fast. The parameter drv_slew corresponds to a hardware register. Please refer the User Manual of target device for description.

Outputs:

None

4.1.3 rom_dis_lock_shd_spi

Function Header:

void rom_dis_lock_shd_spi(uint8_t lock_shd_spi);

Description:

Apply GPIO Locks as specified in customer section of EFUSE

Inputs:

Input Parameters	Description
lock_shd_spi	0 (do not modify lock values) 1(insure Shared SPI GPIO's are disabled (tri-state input) and these pins are locked.

Outputs: None

4.1.4 qmpsi_init

Function Header:

void qmspi_init(uint32_t freqHz, uint8_t spi_signalling, uint8_t ifctrl);

Description:

This function configures the frequency of SPI, the mode of operation and interface control.

The permitted frequencies for the SPI are 48 MHz, 24 MHz, 16 MHz, and 12 MHz.

The SPI supports 4 modes of operation (SPI_MODE_0, SPI_MODE_1, SPI_MODE_2, SPI_MODE_3).

Inputs:

Input Parameters	Description
Freq_hz	An unsigned 32 bit integer indicating frequency. The following frequencies are supported - 48 MHz, 24 MHz, 16 MHz, and 12 MHz.
Spi_mode	An unsigned 8 bit integer indicating the mode. The following modes of operation are permitted. * SPI_MODE_0 * SPI_MODE_1 * SPI_MODE_2 * SPI_MODE_3.
If_ctrl	An unsigned 8 bit integer indicating interface control. Refer the Data sheet of target for bit definitions.

Macro Values for SPI Modes Field:

Macro Name	Value
SPI_MODE_0	0
SPI_MODE_1	6
SPI_MODE_2	1
SPI_MODE_3	7

Outputs:

None

4.1.5 qmspi_freq_get

Function Header:

uint32_t qmspi_freq_set(void);

Description:

The function call is used to get the frequency of SPI.

Inputs:

None

Outputs:

Returns the SPI operating frequency.

4.1.6 qmspi_freq_set

Function Header:

void qmspi_freq_set (uint32_t freq_hz);

Description:

This function configures the frequency of SPI. The required frequency is passed to the function as an input parameter (freq_hz). The permitted frequencies for the SPI are 48 MHz, 24 MHz, 16 MHz, and 12 MHz.

Inputs:

Input Parameters	Description
Freq_hz	A 32 bit unsigned integer indicating the required frequency of operation.

Outputs:

None

4.1.7 qmspi_xfr_done_status

Function Header:

bool qmspi_xfr_done_status(uint32_t* qmspi_status);

Description:

This function gets the status of spi, updates the status into the pointer passed as argument, and returns the done status by evaluating the status register value. If done status is set, the bool value true is returned if not the value false is returned.

Bit Number	Definition
0	XFR_COMPLETE
1	DMA_COMPLETE
2	TX_BUFF_ERR
3	RX_BUFF_ERR
4	PROG_ERR
8	TX_BUFF_FULL
9	TX_BUFF_EMPTY
10	TX_BUFF_REQ
11	TX_BUFF_STALL
12	RX_BUFF_FULL

Bit Number	Definition
13	RX_BUFF_EMPTY
15	RX_BUFF_STALL
16	XFR_ACTIVE

Inputs:

Input Parameters	Description
Qmspi_status	A pointer to an unsigned 32 bit integer where the status of qmspi is stored.

Outputs:

TRUE if set, FALSE otherwise.

4.1.8 qmspi_start

Function Header:

void qmspi_start(uint16_t ien_mask);

Description:

This function starts the SPI operation with the specified interrupt mask.

Inputs:

Input Parameters	Description
Ien_mask	An unsigned 16 bit integer specifying the interrupt mask. The bit definition of interrupt enable mask corresponds to Status register bit definitions mentioned in qmspi_xfr_-done_status. Refer data sheet for available interrupts.

Outputs:

None

4.1.9 qmspi_start_dma

Function Header:

void qmspi_start_dma(uint8_t dmach_id, uint16_t ien_mask);

Description:

The function starts SPI operations along with a DMA channel. Ien_mask represents the Interrupt Enable mask.

The dmach_id is used to select the DMA Channel. There are 14 DMA channels and the channels along with their associated values are presented below.

Channel Name	Value
DMA_CH00_ID	0
DMA_CH01_ID	1
DMA_CH02_ID	2
DMA_CH03_ID	3
DMA_CH04_ID	4
DMA_CH05_ID	5
DMA_CH06_ID	6
DMA_CH07_ID	7
DMA_CH08_ID	8
DMA_CH09_ID	9
DMA_CH10_ID	10
DMA_CH11_ID	11
DMA_CH12_ID	12
DMA_CH13_ID	13

Inputs:

Input Parameters	Type
Dmach_id	An 8 bit unsigned integer indicating the DMA channel. The available channels are present above.
Ien_mask	An unsigned 16 bit integer specifying the interrupt mask. The bit definition of interrupt enable mask corresponds to Status register bit definitions mentioned in qmspi_xfr_-done_status. Refer data sheet for available interrupts.

Outputs:

None

4.1.10 qmspi_cfg_spi_cmd

Function Header:

uint8_t qmspi_cfg_spi_cmd(uint32_t spi_cmd, uint32_t spi_address);

Description:

This routine configures the QMSPI controller.

The bit definitions of the argument spi_cmd are presented below.

1. b[7:0] = SPI op-code
2. b[15:8] = flags
3. b[9:8] = cmd bus width 0=1X, 1=2X, 2=4X
4. b[11:10] = address bus width
5. b[13:12] = data bus width
6. b[14] = 0 (24-bit address), 1(32-bit address)
7. b[15] = 1 use mode byte
8. b[23:16] = mode byte
9. b[31:24] = number of dummy clocks expressed as number of bytes where
   a)Clocks = bytes * clocks/byte. Clocks per byte depend upon data bus width.
   b)Data bus width – 1X -> 8clocks/byte, 2X -> 4 clocks/byte, 4X -> 2 clocks/byte.
   c)Example: 4X 24bit read 0x6B requires 8 dummy clocks. At 2 clocks/byte, 4 bytes are required.

The SPI address can be either 24 bit address or 32 bit address.

Inputs:

Input Parameters	Description
Spi_cmd	An unsigned 32 bit integer. The bit definitions are presented above
Spi_address	An unsigned 32 bit integer specifying the SPI address

Outputs:

The function returns the ID of the Last Descriptor used (Descriptor is a Hardware register, refer data sheet for more details).

4.1.11 qmspi_read_dma

Function Header:

uint32_t qmspi_ read_dma(uint32_t spi_cmd,

uint32_t spi_address,

uint32_t mem_addr,

uint32_t nbytes,

uint8_t dmach_id);

Description:

This routine configures the QMSPI controller to read a specified number of bytes form a specified address.

If nbytes is 0, the value returned will be zero.

If mem_addr is specified as zero, the function will return a zero.

Inputs:

Input Parameters	Description
Spi_cmd	An unsigned 32 bit integer specifying the SPI Command. For spi_cmd bit definitions, please refer to qmspi_cfg_spi_cmd.
Spi_address	An unsigned 32 bit integer specifying the SPI address
Mem_addr	An unsigned 32 bit integer which specifies the 32 bit address from where the data is to be read.
Nbytes	An unsigned 32 bit integer which refers to the number of bytes to be read.
Dmach_id	An 8 bit unsigned integer which is used to refer to the DMA Channel to be used. Refer to qmspi_start_dma section for a description regarding dmach_id.

Outputs:

An unsigned 32 bit integer reflecting the number of bytes read.

4.1.12 qmspi_write_dma

Function Header:

```
uint32_t qmspi_write_dma(    uint32_t spi_cmd,
                             uint32_t spi_address,
                             uint32_t mem_addr,
                             uint32_t nbytes,
                             uint8_t dmach_id);
```

Description:

The function initiates a DMA write operation at the specified address.

Inputs:

Input Parameters	Description
Spi_cmd	An unsigned 32 bit integer specifying the SPI Command. For spi_cmd bit definitions, please refer to qmspi_cfg_spi_cmd.
Spi_address	An unsigned 32 bit integer specifying the SPI address.
Mem_addr	An unsigned 32 bit integer used to specify the 32 bit address at which the data ought to be written.

Input Parameters	Description
Nbytes	An unsigned 32 bit integer which refers to the number of bytes to be written.
Dmach_id	An 8 bit unsigned integer which is used to refer to the DMA Channel to be used. Refer to qmspi_start_dma section for a description regarding dmach_id.

Outputs:

The function returns a 32 bit value indicating the number of bytes written.

4.1.13 qmspi_xmit_cmd

Function Header:

```
Bool qmspi_xmit_cmd(        uint8_t* cmd_params,
                            uint8_t ntx,
                            uint8_t nresponse);
```

Description:

This function is used to send small commands to the flash. The pointer cmd_params has a list of commands to be sent. The possible SPI commands to flash are listed. below.

Command	Value
Write Enable	0x06
Volatile SR Write Enable	0x50
Write Disable	0x04
Read Status-1	0x05
Write Status-1	0x01
Read Status -2	0x35
Write Status-2	0x31
Read JEDEC-ID	0x9F
Read Data	0x03
Fast Read Data	0x0B
Fast Read Dual Data	0x3B
Fast Read Quad Data	0x6B
Read Data 4-byte address	0x13
Fast Read Data 4-byte	0x0C
Fast Read Dual Data 4-byte	0x3C
Fast Read Quad Data 4-byte	0x6C
Fast Read Dual IO Addr4	0xBC
Fast Read Quad IO Addr4	0xEC
Page Program	0x02
Quad Page Program	0x32
Sector Erase (4KB)	0x20
Block Erase (32KB)	0x52
Block Erase (64KB)	0xD8
Read SFDP	0x5A

Note:	This routine can be used by any commands required. They are not restricted to the commands listed above.

Inputs:

Input Parameters	Description
Cmd_params	An unsigned 8 bit pointer to a set of Flash commands.
Ntx	An 8 bit unsigned integer values stating the number of commands to be sent.
Nresponse	An unsigned 8 bit integer indicating the number of responses to be read after transmitting commands.

Outputs:

The function returns “true” if a previous transfer is not active and cmd_params is not a null pointer and ntx is not zero. If any of the aforementioned conditions are false, “false” value is returned.

4.1.14 qmspi_read_fifo

Function Header:

Uint32_t qmspi_read_fifo(uint8_t * data,
**   uint32_t buff_len);**

Description:

The function is used to read data from the qmspi FIFO.

The number of bytes read will always be equal to or less than the buffer length specified.

Inputs:

Input Parameters	Description
Data	An unsigned a-bit integer pointer to a buffer.
Buff_len	An unsigned 32 bit integer specifying the length.

Outputs:

The function returns a 32 bit value indicating the number of bytes read.

4.2 AES FUNCTIONS

4.2.1 aes_hash_power

Function Header:

Void aes_hash_power(uint8_t pwr_state);

Description:

This function is used to enable or disable AES and Hash Hardware Block.

Note: AES and Hash hardware accelerators do not implement a block level clock gate control and share AHB resources (master, internal DMA, clocking). A single PCR sleep control will sleep both blocks. Before setting the PCR sleep bit, clear AES and Hash Control registers to stop both engines. To wake (ungated clocks) clear the PCR sleep enable bit.

Inputs:

Input Parameter	Description
Pwr_state	An unsigned 8-bit integer specifying the power state. pwr_state * Non-zero = ungate clocks to block * 0 gate clocks to block.

Outputs:

None

4.2.2 aes_hash_reset

Function Header:

Void aes_hash_reset(void);

Description:

This function is used to reset the AES and Hash block.

Inputs:

None

Outputs:

None

4.2.3 aes_busy

Function Header:

Bool aes_busy (void);

Description:

This function is used to check if the AES block is running. “true” is returned if the block is running and “false” is returned if the block is not running.

Inputs:

None

Outputs:

True if AES block is running, False otherwise.

4.2.4 aes_status

Function Header:

uint32_t aes_status(void);

Description:

This function is used to read the status of the AES block.

Inputs:

None

Outputs:

The status of the AES block is reflected in the return value. The bit definitions of AES Status value is presented below.

Bit Number	Definition (all bits are read only)
0	AES Busy Status (1 if busy, 0 otherwise)
1	AES CCM Mode MAC_T Calculation is valid (1 if valid, 0 otherwise *not supported*)
2	DMA Error. (1 if error, indicates that the AES DMA Master received AHB Bus error.)

4.2.5 aes_done_status

Function Header:

Bool aes_done_status(uint32_t * status_value);

Description:

The function updates the status value of the AES block in the pointer argument. The done status is evaluated and it is returned as a Boolean value.

Inputs:

Input Parameter	Description
Status_value	An unsigned 32 bit integer pointer where the status value will be stored. Refer to aes_status for bit definition of status register.

Outputs:

The status of the AES is updated in the pointer argument.

The return value is TRUE if done status is set, FALSE otherwise.

4.2.6 aes_stop

Function Header:

Bool aes_stop(void);

Description:

This function is used to stop the AES block. The function accesses the AES control register to stop the AES Operations.

Inputs:

None

Outputs:

The busy status of AES block is returned as a Boolean value.

4.2.7 aes_start

Function Header:

Bool aes_start(bool ien);

Description:

This function is used to start the AES Block. The input argument controls interrupt enable.

Inputs:

Input Parameter	Description
Ien	Boolean value specifying the interrupt status. This value, if true enables interrupt and if false, disables interrupt.

Outputs:

The busy status of the AES block is returned as a Boolean value.

4.2.8 aes_iclr

Function Header:

Uint32_t aes_iclr(void);

Description:

The function is used to clear Hash interrupts. The function will return the status of AES Block. AES Status register is a read-to-clear register. If it is zero, no status is set.

Inputs:

None

Outputs:

The AES Status will be reflected in the unsigned 32 bit return value. Refer to aes_status for bit definitions.

4.2.9 aes_set_key

Function Header:

```
Uint8_t aes_set_key( const uint32_t *pkey,
                     const uint32_t *piv,
                     uint8_t key_len,
                     bool msbf);
```

Description:

The function programs the AES accelerator with key and optional initialization. AES key size and pre calculation modes are set. The AES engine is not started. Do not call this routine if the AES engine is busy.

Inputs:

Input Parameter	Description
Pkey	Pointer to a word (32-bit) aligned buffer containing the AES key, LSB first.
Piv	Pointer to a word (32-bit) aligned buffer containing AES initialization vector, LSB first. NULL if no Initialization vector is required.
Key_len	A uint8_t indicating the key length. The permitted values are: * AES_KEYLEN_128 * AES_KEYLEN_192 * AES_KEYLEN_256.
Msbf	A Boolean value indicating most significant bit first.

Macro Values for Key Length:

Macro Name	Value
AES_KEYLEN_128	0
AES_KEYLEN_192	1
AES_KEYLEN_256	2

Outputs:

AES_OK (Success), AES_ERR_BAD_POINTER (pkey is NULL), AES_ER-R_BAD_KEY_LEN (key len is not supported).

Macro Values for Return Codes:

Macro Name	Value
AES_OK	0
AES_ERR_BUSY	1
AES_ERR_BAD_KEY_LEN	2
AES_ERR_BAD_POINTER	3
AES_ERR_MISALIGNED_DATA	4
AES_ERR_UNSUPPORTED_OP	5

4.2.10 aes_crypt

Function Header:

```
Uint8_t aes_crypt (const uint32 *data_in,
                   uint32_t *data_out,
                   uint32_t num_128bit_blocks,
                   uint8_t modes);
```

Description:

This function programs specified AES operations using the currently programmed key. The hardware permits the following modes of operation; ECB, CBC, CTR and OFB modes.

Inputs:

Input Parameters	Description
Data_in	A pointer to a word (32-bit) aligned input data buffer.
Data_out	A pointer to a word (32-bit) aligned output data buffer.
Num_128bit_blocks	Size of input data as a number of 126-bit (16-byte) blocks.
Modes	8 bit integer indicating the mode of operation. The permitted modes of operation are • AES_MODE_ECB • AES_MODE_CBC • AES_MODE_CTR • AES_MODE_OFB.

Macro Values for Modes Field:

Macro Name	Value
AES_MODE_ECB	0
AES_MODE_CBC	1
AES_MODE_CTR	2
AES_MODE_CFB	3
AES_MODE_OFB	4

Outputs:

AES_OK (AES HW Programmed), AES_ERR_BAD_POINTER (NULL pointers or number of blocks is 0 or buffers cross CEC/MEC DMA boundary).

Macro Values for Return Codes:

Macro Name	Value
AES_OK	0
AES_ERR_BUSY	1
AES_ERR_BAD_KEY_LEN	2
AES_ERR_BAD_POINTER	3
AES_ERR_MISALIGNED_DATA	4
AES_ERR_UNSUPPORTED_OP	5

4.3 RSA FUNCTIONS

Note: Before using RSA functions the PKE block needs to be powered ON explicitly using pke_power() API.

4.3.1 rsa_load_key

Function Header:

```
uint8_t rsa_load_key(uint16_t rsa_bit_len,
                     const BUFF8_T *private_exponent,
                     const BUFF8_T *public_modulus,
                     const BUFF8_T *public_exponent,
                     bool msbf)
```

Description:

This function loads a given key into the PKE shared crypto memory. If moduli pointers are NULL, no copy operations are performed.

The shared crypto memory of the CEC/MEC device family is divided into 31 slots (slot0-slot30) each of 512 bytes.

The following modes of operation are possible

RSA Encryption with Public Key

- Pointer to private exponent = Not used
- Pointer to public modulus = your public key modulus -> slot 0
- Pointer to public exponent = your public key exponent -> slot 8

RSA Decryption with Private Key

- Pointer to private exponent = your private key modulus -> slot 6
- Pointer to public modulus = your public key modulus -> slot 0
- Pointer to public exponent = your public key exponent -> slot 8

Alternate

RSA Encryption with Private Key

- Pointer to private exponent = Not used
- Pointer to public modulus = your public key modulus -> slot 0
- Pointer to public exponent = your private key exponent -> slot 8

RSA Encryption with Private Key

- Pointer to private exponent = your private exponent -> slot 6
- Pointer to public modulus = your public modulus -> slot 0
- Pointer to Public Exponent = not used

<u>Inputs</u>:

Structure definition BUFF8_T

```
typedef struct buff8_s
{
   uint32_t len;
   uint8_t *pd;
} BUFF8_T;
```

Input parameters	Description
Rsa_bit_len	An unsigned 16 bit integer reflecting the bit size. The permitted values are 1024, 2048 or 4096 bits
Private_exponent	A pointer to the structure BUFF8_T having members as length field and the UINT8 pointer to the RSA private exponent. Length field indicates the size of private exponent in terms of bytes
Public_modulus	A pointer to the structure BUFF8_T having members as length field and the UINT8 pointer to the RSA public modulus. Length field indicates the size of public modulus in terms of bytes
Public_exponent	A pointer to the structure BUFF8_T having members as length field and the UINT8 pointer to the RSA public exponent. Length field indicates the size of public exponent in terms of bytes
Msbf	A Boolean value if true, indicates Most significant byte first. If false, it indicates least significant byte first.

<u>Outputs</u>:

The return values and their description are presented below.

PKE_RET_ERR_BUSY – if the RSA module is busy

PKE_RET_ERR_INVALID_BIT_LENGTH – if the rsa_bit_len parameter has a value that is not permitted.

PKE_RET_OK – if the operation requested was successful.

Return Code Macro Values:

Macro Name	Value
PKE_RET_OK	0
PKE_RET_ERR_BUSY	1
PKE_RET_ERR_BAD_PARAM	2
PKE_RET_ERR_BAD_ADDR	3
PKE_RET_ERR_UNKNOWN_OP	4
PKE_RET_ERR_INVALID_BIT_LENGTH	5
PKE_RET_ERR_INVALID_MSG_LENGTH	6

4.3.2 rsa_keygen

Function Header:

```
uint8_t rsa_keygen (uint16_t rsa_bit_len,
                    const BUFF8_T *p,
                    const BUFF8_T *q,
                    const BUFF8_T *e,
                    bool msbf);
```

Description:

This routine generates the private and public key. Output is Private-Public key pair: Slot 0 = Public Modulus, Slot 6 = Private Modulus, Public key is (Public Modulus, Public Exponent), Private key is Private Modulus

Inputs:

Input Parameters	Description
Rsa_bit_len	An unsigned 16 bit integer indicating the bit length. Permitted values are 1024, 2048 or 4096 bits
p	Pointer to structure containing prime p of length (rsa_bit_len/8)
q	Pointer to structure containing prime q of length (rsa_bit_len/8)
e	Pointer to structure containing RSA public exponent
Msbf	A Boolean value if true, indicates Most significant byte first. If false, it indicates least significant byte first.

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_ADDR – this error value is returned if Mesg points to NULL.

PKE_RET_ERR_BUSY – this error value is returned if pke is busy.

PKE_RET_ERR_INVALID_BIT_LENGTH – this error value is returned if rsa_bit_len has invalid data.

PKE_RET_OK – this value indicates a successful execution of requested operation.

4.3.3 rsa_modular_exp

Function Header:

```
uint8_t rsa_modular_exp(uint16_t rsa_bit_len,
                        const BUFF8_T *M,
                        const BUFF8_T *e,
                        const BUFF8_T *n,
                        bool msbf)
```

Description:

This routine calculates modular exponentiation. Parameters are loaded into SCM as follows:

OptPtrA specifies slot number of M = 1

OptPtrB specifies slot number of e = 2

OptPtrC specifies slot number of result, C = (M^e) mod n = Slot 3

n is located in Slot 0

Inputs:

Input Parameters	Description
Rsa_bit_len	An unsigned 16 bit integer indicating the bit length. Permitted values are 1024, 2048 or 4096 bits
M	pointer to BUFF8_T structure containing number to exponentiate
E	pointer to BUFF8_T structure containing exponent
n	pointer to BUFF8_T structure containing modulus
Msbf	A Boolean value if true, indicates Most significant byte first. If false, it indicates least significant byte first.

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_ADDR – this error value is returned if Mesg points to NULL.

PKE_RET_ERR_BUSY – This error value is returned if pke is busy.

PKE_RET_ERR_INVALID_BIT_LENGTH – this error value is returned if rsa_bit_len has invalid data.

PKE_RET_OK – this value indicates a successful execution of requested operation.

4.3.4 rsa_encrypt

Function Header:

```
uint8_t rsa_encrypt(uint16_t rsa_bit_len,
                    const BUFF8_T * mesg,
                    bool msbf);
```

Description:

This routine starts the encryption process. It requires the RSA keys to have been previously loaded. If encrypting with public key then, slot 8 must contain the public exponent and slot 0 contains the public modulus. If encrypting with a private key then slot 8 must contain the private exponent. Encrypted output is in slot 5. Message length is limited due to PKCS#1 v1.5 (recommended way to pad input and output to/from RSA Algorithm) padding. The maximum message length is (rsa_bit_len/8) – 11.

Inputs:

Input Parameters	Description
Rsa_bit_len	An unsigned 16 bit integer indicating the bit length. Permitted values are 1024, 2048 or 4096 bits
Mesg	A pointer to the structure BUFF8_T having members as length field and the UINT8 pointer to the input message. Length field indicates the size of the input message in terms of bytes
Msbf	A Boolean value if true, indicates most significant byte first. If false, it indicates least significant byte first.

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_ADDR – this error value is returned if Mesg points to NULL.

PKE_RET_ERR_BUSY – This error value is returned if pke is busy.

PKE_RET_ERR_INVALID_BIT_LENGTH – this error value is returned if rsa_bit_len has invalid data.

PKE_RET_OK – this value indicates a successful execution of requested operation.

4.3.5 rsa_decrypt

Function Header:

Uint8_t rsa_decrypt (uint16_t rsa_bit_len,
const BUFF8_T * encrypted_mesg,
bool msbf);

Description:

The routine is used to decrypt messages. It computes M = (encrypted_mesg)^(slot 6). Slot 6 contains either the private or public exponent. If the message was signed with a private key, then slot 6 should contain public exponent. If the message was signed with a public key then slot 6 should contain the private exponent. Switch the order of parameters in rsa_load_keys() to store the appropriate exponent in slot 6. The decrypted output will be in slot 5 and will contain PKCS#1 v1.5 padding.

Inputs:

Input parameter	Description
Rsa_bit_len	An unsigned 16 bit integer indicating the bit length. Permitted values are 1024, 2048 or 4096 bits
Encrypted_mesg	A pointer to the structure BUFF8_T having members as length field and the UINT8 pointer to the encrypted message. Length field indicates the size of the encrypted message in terms of bytes
Msbf	A Boolean value if true, indicates most significant byte first. If false, it indicates least significant byte first.

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_ADDR – this error value is returned if Encrypted_mesg points to NULL.

PKE_RET_ERR_BUSY – this error value is returned if pke is busy.

PKE_RET_ERR_INVALID_BIT_LENGTH – this error value is returned if rsa_bit_len has invalid data.

PKE_RET_OK – this value indicates a successful execution of requested operation.

4.3.6 rsa_crt_gen_params

Function Header:

```
uint8_t pke_rsa_crt_gen_params(uint16_t rsa_bit_len,
                               const BUFF8_T* p,
                               const BUFF8_T* q,
                               const BUFF8_T* pubmod,
                               const BUFF8_T* prvexp,
                               bool msbf)
```

Description:

This routine is used for RSA CRT parameter generation. This routine requires RSA keys to have been previously loaded. Public Modulus must be in slot 0 and public exponent in slot 8. Private exponent must be in slot 6. Prime p will be loaded into slot 2 and prime q into slot 3. After the engine is done, the three output parameters are: dp in slot 0xA, dq in slot 0xB and I in slot 0xC.

Inputs:

Input Parameters	Description
Rsa_bit_len	An unsigned 16 bit integer indicating the bit length. Permitted values are 1024 or 2048 bits

p	A pointer to structure of type BUFF8_T defined by
	typedef struct BUFF8_T
	{
	uint32_t len;
	uint8_t *pd;
	};
	The structure variable contains pointer to prime number 1, and length in bytes.

q	A pointer to structure of type BUFF8_T defined by typedef struct BUFF8_T { uint32_t len; uint8_t *pd; } ; The structure variable contains pointer to prime number 2, and length in bytes.

Input Parameters	Description

pubmod	A pointer to structure of type BUFF8_T defined by
	typedef struct BUFF8_T
	{
	uint32_t len;
	uint8_t *pd;
	};
	The structure variable contains pointer to public modulus (n) used for encryption, and length in bytes.

prvexp	A pointer to structure of type BUFF8_T defined by
	typedef struct BUFF8_T
	{
	uint32_t len;
	uint8_t *pd;
	};
	The structure variable contains pointer to private exponent (d), and length in bytes.

Msbf	A boolean value if true, indicates most significant byte first. If false, it indicates least significant byte first.

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_ADDR – this error value is returned if either p or q points to NULL.

PKE_RET_ERR_BUSY – this error value is returned if pke is busy.

PKE_RET_ERR_INVALID_BIT_LENGTH – this error value is returned if rsa_bit_len has invalid data.

PKE_RET_OK – this value indicates a successful execution of requested operation.

4.3.7 rsa_load_crt_params

Function Header:

```
uint8_t pke_rsa_load_crt_params(uint16_t rsa_bit_len,
                                const BUFF8_T *dp,
                                const BUFF8_T *dq,
                                const BUFF8_T *I,
                                bool msbf)
```

Description:

This routine loads CRT Key parameters. If a parameter pointer is not NULL load it into its SCM slot.

Inputs:

Input Parameters	Description
Rsa_bit_len	An unsigned 16 bit integer indicating the bit length. Permitted values are 1024 or 2048 bits

Dp	A pointer to structure of type BUFF8_T defined by
	typedef struct BUFF8_T
	{
	   uint32_t len;
	   uint8_t *pd;
	};
	The structure variable contains pointer to RSA key CRT parameter dp (first exponent), and length of the dp parameter.

Dq	A pointer to structure of type BUFF8_T defined by typedef struct BUFF8_T { uint32_t len; uint8_t *pd; } ; The structure variable contains pointer to RSA key CRT parameter dq (second exponent), and length of the dq parameter.

I	A pointer to structure of type BUFF8_T defined by
	typedef struct BUFF8_T
	{
	   uint32_t len;
	   uint8_t *pd;
	};
	The structure variable contains pointer to RSA key CRT parameter I (coefficient), and length of the I parameter.

Msbf	A Boolean value if true, indicates Most significant byte first. If false, it indicates least significant byte first.

Outputs:

The return values and description are presented below.

PKE_RET_OK if successful

PKE_RET_ERR_BUSY if pke is busy

PKE_RET_ERR_INVALID_BIT_LENGTH if the rsa_bit_len has invalid value.

4.3.8 rsa_crt_decrypt

Note: This API should not be used for bootrom versions v.A0 and v.A1. An alternative is adding 3 lines of code after calling rsa_crt_decrypt() as follows:

```
rsa_crt_decrypt( rsa bit length, mesg, msbf);
pke_command &= ~(0x17)
pke_command |= 0x13
pke_start(false);
```

For bootrom versions v.A2, this API can be directly used.

Function Header:

uint8_t pke_rsa_crt_decrypt(uint16_t rsa_bit_len,
const BUFF8_T* encrypted_mesg,
bool msbf)

Description:

This routine performs RSA decryption using Chinese remainder theorem. It computes M = (encrypted_mesg) ^ (slot 6). Slot 6 contains wither the private or public exponent. If the message was signed with a private key then slot 6 should contain the public exponent. If the message was signed with a public key then slot 6 should c

Inputs:

Input parameter	Description
Rsa_bit_len	An unsigned 16 bit integer indicating the bit length. Permitted values are 1024 or 2048 bits

Encrypted_mesg	A pointer to structure of type BUFF8_T defined by
	typedef struct BUFF8_T
	{
	uint32_t len;
	uint8_t *pd;
	};
	The structure variable contains pointer to encrypted message (ciphertext), and length in bytes.

Msbf	A boolean value if true, indicates most significant byte first. If false, it indicates least significant byte first.

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_ADDR – this error value is returned if Encrypted_mesg points to NULL.

PKE_RET_ERR_BUSY – this error value is returned if pke is busy.

PKE_RET_ERR_INVALID_BIT_LENGTH – this error value is returned if rsa_bit_len has invalid data.

PKE_RET_OK – this value indicates a successful execution of requested operation.

4.3.9 rsa_signature_gen

Function Header:

uint8_t rsa_ signature_gen(uint16_t rsa_bit_len,
const BUFF8_T* hash_digest,
bool msbf);

Description:

This routine is used for RSA signature generation. It Computes M = (hash)^(Slot 6). Slot 6 contains either the RSA key private or public exponent. The opposite exponent must be used for signature verification. Slot 0 contains the RSA key public modulus. The hash is loaded into Slot 4. Signature output is in Slot 5. The hash digest is truncated to (rsa_bit_len / 8) bytes.

Inputs:

Input Parameter	Description
Rsa_bit_len	An unsigned 16 bit integer indicating the bit length. Permitted values are 1024 or 2048 bits

hash_digest	A pointer to structure of type BUFF8_T defined by
	typedef struct BUFF8_T
	{
	uint32_t len;
	uint8_t *pd;
	};
	The structure variable contains pointer to hash to sign. It is the caller's responsibility to properly pad the Hash.

Msbf	A boolean value if true, indicates most significant byte first. If false, it indicates least significant byte first.

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_ADDR – this error value is returned if Encrypted_mesg points to NULL.

PKE_RET_ERR_BUSY – this error value is returned if pke is busy.

PKE_RET_ERR_INVALID_BIT_LENGTH – this error value is returned if rsa_bit_len has invalid data.

PKE_RET_OK – this value indicates a successful execution of requested operation.

4.3.10 rsa_signature_verify

Function Header:

```
uint8_t rsa_ signature_verify(uint16_t rsa_bit_len,
                        const BUFF8_T *signature,
                        const BUFF8_T* hash_digest_pkcs15,
                        bool msbf);
```

Description:

This routine is used for RSA signature verification. Computes h = (signature)^(Slot 8). Slot 8 contains either the RSA key private or public exponent. The opposite exponent must be used for signature generation. Slot 0 contains the RSA key public modulus. The expected hash is loaded into Slot 0xC. Recovered hash digest output is in Slot 5. PKE compares the contents of Slot 5 with Slot 0xC. The recovered hash digest will also contain PKCS#1 v1.5 padding. The expected hash digest must also contain the same padding.

Inputs:

Input Parameter	Description
Rsa_bit_len	An unsigned 16 bit integer indicating the bit length. Permitted values are 1024 or 2048 bits

hash_digest_pkcs15	A pointer to structure of type BUFF8_T defined by
	typedef struct BUFF8_T
	{
	    uint32_t len;
	    uint8_t *pd;
	};
	The structure variable contains pointer to hash to verify.

Msbf	A boolean value if true, indicates most significant byte first. If false, it indicates least significant byte first.

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_ADDR – this error value is returned if Encrypted_mesg points to NULL.

PKE_RET_ERR_BUSY – this error value is returned if pke is busy.

PKE_RET_ERR_INVALID_BIT_LENGTH – this error value is returned if rsa_bit_len has invalid data.

PKE_RET_OK – this value indicates a successful execution of requested operation.

4.4 PKE FUNCTIONS

4.4.1 pke_power

Function Header:

Void pke_power (bool pwr_on);

Description:

This routine controls the Gate on/off clocks to pke block. Before setting PCR PKE sleep enable, write 0 to the PKE control register to stop the engine.

Inputs:

Input Parameters	Description
Pwr_on	A Boolean input parameter. True wakes the block (ungates clocks), False puts the block in sleep mode (gates clock).

Outputs:

None

4.4.2 pke_reset

Function Header:

Void pke_reset(void);

Description:

This routine resets the PKE block.

Inputs:

None

Output:

None

4.4.3 pke_status

Function Header:

Uint32_t pke_status(void);

Description:

The routine returns the status of the PKE block.

Inputs:

None

Outputs:

The routine returns the PKE block status. The important status bits are listed below.

Bit Number	Description
16	PKE Busy Bit. This applies to all operations. (1 if busy, 0 otherwise).
9	This bit only applies to signature verify operations. 0 if signature is valid, 1 otherwise.
5	This bit applies to Elliptic Curve Operations. 0 if EC point not at infinity, 1 otherwise.
4	This bit applies to Elliptic Curve Operations. 0 if EC point is on curve, 1 otherwise.

4.4.4 pke_done_status

Function Header:

Bool pke_done_status(uint32_t * status_value);

Description:

The routine is used to check the done status of the PKE block.

Inputs:

Inputs Parameters	Description
Status_value	An unsigned 32 bit integer pointer where the status value will be stored. Refer to pke_status for bit definitions.

Outputs:

Returns True if done status is set, False Otherwise.

4.4.5 pke_start

Function Header:

void pke_start(bool ien);

Description:

The function starts the pke block

Inputs:

Input Parameters	Description
Ien	A Boolean value specifying the interrupt status. The interrupt enable mask if set, enables done and error interrupts.

Outputs:

None

4.4.6 pke_busy

Function Header:

Bool pke_busy(void);

Description:

This routine returns the PKE busy status. PKE status register bits are read-only and are cleared upon reading. Reading also clears PKE block interrupt signal.

Inputs:

None

Outputs:

This routine returns a true if PKE is busy, False otherwise.

4.4.7 pke_set_operand_slot

Function Header:

void pke_set_operand_slot (uint8_t operand, uint8_t slot_num);

Description:

This routine sets the slot for the selected operand

Inputs:

Input Parameters	Description
operand	Operand number
slot_num	An unsigned 8 bit integer specifying the slot number

Outputs:

None

4.4.8 pke_get_operand_slot

Function Header:

uint8_t pke_get_operand_slot (uint8_t operand);

Description:

This routine returns the slot number for the specified operand

Inputs:

Input Parameters	Description
operand	Operand number

Outputs:

Slot number

4.4.9 pke_set_operand_slots

Function Header:

void pke_set_operand_slots (uint32_t operand);

Description:

This routine sets the slots for the operand 0, 1, 2

Inputs:

Input Parameters	Description
operand	32 bit variable containing slots for operand 0, 1, 2

Outputs:

None

4.4.10 pke_get_slot_addr

Function Header:

uint32_t pke_get_slot_addr (uint8_t slot_num);

Description:

This routine returns the address of the slot in the crypto memory

Inputs:

Input Parameters	Description
slot_num	An unsigned 8 bit integer specifying the slot number

Outputs:

address

4.4.11 pke_fill_slot

Function Header:

void pke_fill_slot (const uint8_t slot_num, const uint32_t fill_val);

Description:

This routine fills the slot memory with fill_val

Inputs:

Input Parameters	Description
slot_num	An unsigned 8 bit integer specifying the slot number
fill_val	Value to be filled in slot memory

Outputs:

None

4.4.12 pke_scm_clear_slot

Function Header:

void pke_scm_clear_slot(uint8_t slot_num);

Description:

This routine clears a specified slot in SCM. That is, it fills the specified slot in SCM with 0’s.

Inputs:

Input Parameters	Description
slot_num	An unsigned 8 bit integer specifying the slot number.

Outputs:

None

4.4.13 pke_read_scm

Function Header:

```
uint16_t pke_read_scm ( uint8_t * dest,
                        uint16_t nbytes,
                        uint8_t slot_num,
                        bool reverse_byte_order);
```

Description:

This routine is used to read specified amount of data from a specified SCM slot.

Inputs:

Input Parameter	Description
Dest	An unsigned 8 bit integer pointer to the destination where data will be copied.
Nbytes	An unsigned 16 bit integer specifying the number of bytes to be read.
Slot_num	An unsigned 8 bit integer specifying the slot from which data is to be read.
Reverse_byte_order	A Boolean flag which if true reads data in reverse byte order.

Outputs:

The routine returns the number of bytes copied to the destination.

4.4.14 pke_write_scm32

Function Header:

```
void pke_write_scm32(const std::uint32_t* pdata,
                     uint16_t num_words,
                     uint8_t slot_num,
                     bool reverse_byte_order);
```

Description:

This routine is used to write specified amount of data to a specified SCM slot as DWORD. PKE command register operand size field must be programmed before this routine is called.

Inputs:

Input Parameter	Description
Pdata	An unsigned 32 bit integer pointer to data to be written
Num_bytes	An unsigned 16 bit integer indicating the number of DWord to written
Slot_num	An unsigned 8 bit integer indicating the slot num
Reverse_byte_order	A Boolean flag which if true writes data in reverse byte order

Outputs:

None

4.4.15 pke_write_scm

Function Header:

```
void pke_write_scm(   const uint8_t * pdata,
                      uint16_t num_bytes,
                      uint8_t slot_num,
                      bool reverse_byte_order);
```

Description:

This routine is used to write specified amount of data to a specified SCM slot. PKE command register operand size field must be programmed before this routine is called.

Inputs:

Input Parameter	Description
Pdata	An unsigned 8 bit integer pointer to data to be written.
Num_bytes	An unsigned 16 bit integer indicating the number of bytes to written.
Slot_num	An unsigned 8 bit integer indicating the slot num.
Reverse_byte_order	A Boolean flag which if true writes data in reverse byte order.

Outputs:

None

4.4.16 pke_clear_scm

Function Header:

void pke_clear_scm(void);

Description:

Clears the PKE block's shared crypto memory.

Note:	The caller must insure the PKE engine is idle and not in sleep state (clock gated) before calling this routine.

Inputs:

None

Output:

None

4.4.17 pke_ists_clear

Function Header:

uint32_t pke_ists_clear(void);

Description:

Read and clear Status bit for PKE and the interrupt source bits for PKE block

Inputs:

None

Outputs:

PKE status[31:0] – in which

Reserved [31:22] = 0

GIRQ16.Source[1:0] in bits[21:20]

PKE status register value in bits[16:0],

4.4.18 modular_arithm

Function Header:

```
uint8_t modular_arithm(uint32_t op_size,
                       const void *P,
                       uint16_t pnbytes,
                       const void *A,
                       uint16_t anbytes,
                       const void *B,
                       uint16_t bnbytes)
```

Description:

This routine is used to perform modular arithmetic

Inputs:

Input Parameter	Description
op_size	Operand size 64 bits to 4096 bits b[6:0]=operation, b[7]=Field, b[15:8]=size in units of 64 bits, bit[16]=0(parameters LSBF), 1(parameters MSBF) Operation in bits[6:0] * 0x00 Reserved * 0x01 C = (A+B) mod P * 0x02 C = (A-B) mod P * 0x03 C = (A*B) mod P (P odd) * 0x04 C = B mod P (P odd), A is ignored * 0x05 C = (A/B) mod P (P odd) * 0x06 C = (1/B) mod P (P odd) * 0x07 Reserved * 0x08 C = (A * B) F(p) only, P is ignored * 0x09 C = (1/B) mod P (P even), A is ignored * 0x0A C = B mod P (P even), A is ignored
P	Pointer to parameter P
pnbytes	Byte length of P
A	Pointer to parameter A
anbytes	Byte length of A
B	Pointer to parameter B
bnbytes	Byte length of parameter B

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_ADDR – this error value is returned if Encrypted_mesg points to NULL.

PKE_RET_ERR_BUSY – this error value is returned if pke is busy.

PKE_RET_ERR_INVALID_BIT_LENGTH – this error value is returned if rsa_bit_len has invalid data.

PKE_RET_OK – this value indicates a successful execution of requested operation.

Return Code Macro Values:

Macro Name	Value
PKE_RET_OK	0
PKE_RET_ERR_BUSY	1
PKE_RET_ERR_BAD_PARAM	2
PKE_RET_ERR_BAD_ADDR	3
PKE_RET_ERR_UNKNOWN_OP	4
PKE_RET_ERR_INVALID_BIT_LENGTH	5
PKE_RET_ERR_INVALID_MSG_LENGTH	6

4.5 ELLIPTIC CURVE FUNCTIONS

Note:	Before using ECDSA functions the PKE block needs to be powered ON explicitly using pke_power() API.

4.5.1 ec_point_double

Function Header:

```
uint8_t ec_point_double(const uint8_t* pxy,
                        const uint16_t coord_len,
                        const bool msbf);
```

Description:

This routine performs a point doubling operation

Inputs:

Input Parameter	Description
pxy	Pointer to the point on the EC curve
coord_len	length field for the coordinates passed in bytes (px+py)
msbf	A boolean flag which if true writes data in reverse byte order

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

PKE_RET_ERR_BUSY – this error value is returned if PKE is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation.

Return Code Macro Values:

Macro Name	Value
PKE_RET_OK	0
PKE_RET_ERR_BUSY	1
PKE_RET_ERR_BAD_PARAM	2
PKE_RET_ERR_BAD_ADDR	3
PKE_RET_ERR_UNKNOWN_OP	4
PKE_RET_ERR_INVALID_BIT_LENGTH	5
PKE_RET_ERR_INVALID_MSG_LENGTH	6

4.5.2 ec_point_add

Function Header:

```
uint8_t ec_point_add (const uint8_t* p1xy,
                      const uint8_t* p2xy,
                      const uint16_t coord_len,
                      const bool msbf)
```

Description:

This routine performs addition of two points on EC curve

Inputs:

Input Parameter	Description
p1xy	Pointer to the point1 coordinates on EC curve
p2xy	Pointer to the point2 coordinates on EC curve
coord_len	length field for the coordinates passed in bytes (px+py)
msbf	A Boolean flag which if true writes data in reverse byte order

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

PKE_RET_ERR_BUSY – this error value is returned if PKE is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

4.5.3 ec_point_scalar_mult2

Function Header:

```
uint8_t ec_point_scalar_mult2(const uint8_t* px,
                              const uint8_t* py,
                              const uint8_t* pscalar,
                              const uint16_t byte_len,
                              const bool msbf)
```

Description:

This routine performs multiplication of EC curve (P (px,py)) to a scalar data

Inputs:

Input Parameter	Description
px	Pointer to point EC curve for the px coordinates(128 bits on the EC curve Px)
py	Pointer to point EC curve for the py coordinates (128 bits on the EC curve Py)
pscalar	Pointer to scalar data to be multiplied with
byte_len	Length field for the scalar data transferred in byte
msbf	A Boolean flag which if true writes data in reverse byte order

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

PKE_RET_ERR_BUSY – this error value is returned if PKE is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

4.5.4 ec_point_scalar_mult3

Function Header:

```
uint8_t ec_point_scalar_mult3(const uint8_t* pxy,
                              const uint8_t* pscalar,
                              const uint16_t byte_len,
                              const bool msbf)
```

Description:

This routine performs multiplication of EC curve (P (px,py)) to a scalar data

Inputs:

Input Parameter	Description
Pxy	Pointer to point coordinates px followed by py
pscalar	Pointer to scalar data to be multiplied with
byte_len	Length field for the scalar data transferred in byte
msbf	A Boolean flag which if true writes data in reverse byte order

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

PKE_RET_ERR_BUSY – this error value is returned if PKE is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

4.5.5 ec_check_poc2

Function Header:

```
Uint8_t ec_check_poc2(      const uint8_t * px,
                            const uint8_t * py,
                            const uint16_t plen,
                            const bool msbf);
```

Description:

This routine check if the point lies on the EC curve.

Inputs:

Input Parameters	Description
Px	Pointer to parameter px
Py	Pointer to parameter py
plen	Byte length of coordinates passed px , py.
msbf	A Boolean flag which if true writes data in reverse byte order.

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

PKE_RET_ERR_BUSY – this error value is returned if PKE is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

4.5.6 ec_check_poc3

Function Header:

```
uint8_t ec_check_poc3(const uint8_t* p,
                      const uint16_t plen,
                      const bool msbf)
```

Description:

This routine check if the point lies on the EC curve

Inputs:

Input Parameter	Description
P	Pointer to point to check
plen	length for the EC curve passed for the coordinates (Px+py)
msbf	A Boolean flag which if true writes data in reverse byte order

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

PKE_RET_ERR_BUSY – this error value is returned if PKE is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

4.5.7 ec_check_point_less_prime

Function Header:

```
uint8_t ec_check_point_less_prime(const uint8_t* pxy,
                                  const uint16_t plen,
                                  const bool msbf);
```

Description:

This routine check Points coordinates are less than prime. Requires curve has been programmed into PKE via ec_prog_curve()

Inputs:

Input Parameter	Description
Pxy	Pointer to point coordinates
plen	length for the EC curve passed for the coordinates (Px+py)
msbf	A Boolean flag which if true writes data in reverse byte order

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

PKE_RET_ERR_BUSY – this error value is returned if PKE is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

4.5.8 ec_check_ab

Function Header:

Uint8_t ec_check_ab(void);

Description:

This routine checks the parameters a and b of the curve. It requires that the curve be programmed into the PKE via eom_pke_ec_prog_curve().

Inputs:

None

Outputs:

The return values and their descriptions are presented below.

ECC_ERR_BUSY – This error value is returned when PKE is busy.

PKE_RET_OK – this value indicates that the requested operation was successful.

4.5.9 ec_check_in

Function Header:

Uint8_t ec_check_in(void);

Description:

This routine checks the EC Curve order (parameter n). This function requires that the curve be programmed into PKE via pke_ec_prog_curve().

Inputs:

None

Outputs:

The return values and their descriptions are presented below.

ECC_ERR_BUSY – this error value is returned when PKE is busy.

PKE_RET_OK – this value indicates that the requested operation was successful.

4.5.10 ec_prog_curve

Function Header:

Uint8_t ec_prog_curve(const ELLIPTIC_CURVE* curve_p);

Description:

This routine programs the elliptic curve parameters into the PKE shared crypto memory. It programs the EC parameters prime to slot 0, order to slot 1, generator point x-coordinate to slot 2, generator point y-coordinate to slot 3, curve parameter a to slot 4 and curve parameter b to slot 5. All parameters are zero extended to end of the slot. The PKE slots size is 512 bytes. This routine also programs the following register fields in command register.

Input Parameters	Description

Curve_p	A pointer to ELLIPTIC_CURVE structure. The structure definition is presented below.
	Struct ELLIPTIC_CRUVE{
	Uint32_t * param[ELLIPTIC_CURVE_NPARAMS];
	Uint16_t byte_len;
	Uint8_t flags;
	Uint8_t rsvd1;
	};
	Curve parameters are most-significant-byte first. EC firmware will byte reverse before writing to PKE SCM.
	#define EC_FLAG_LSB (0u << 0)
	#define EC_FLAG_MSB (1u << 0)
	#define EC_FLAG_F2M (1u << 1) // Curve is binary
	
	Elliptic Curve parameters a and b can be supplied in a negative encoding OR in a positive encoding. If the parameter is negative but encoded as positive then we use these flags to inform the PKE engine to negate the parameter before use. For example, common curves use a = -3. It's usually supplied in negative encoding with leading 1's. We could encode it as 3 (0x0000....0003) and use C_FLAG_ANEG to configure PKE to negate the value.
	#define EC_FLAG_ANEG (1u << 2)
	#define EC_FLAG_BNEG (1u << 3)

Input Parameters	Description
	PKE stored curve parameters in units of 64-bits(8-bytes) padded with zeros. #define EC_192_PKE_LEN (192ul / 8ul) #define EC_224_PKE_LEN (256ul / 8ul) #define EC_256_PKE_LEN (256ul / 8ul) #define EC_384_PKE_LEN (384ul / 8ul) #define EC_512_PKE_LEN (512ul / 8ul) NOTE: PKE HW must round P-521 length up to next even multiple of 64-bits (10) #define EC_640_PKE_LEN (640ul / 8ul)

	Actual Elliptic curve parameter lengths. The above ELLIPTIC_CURVE_xxx_LEN parameters are the size of the PKE engine slots used for the curve. Slot lengths are in units of 8-bytes and may be larger than actual curve parameters. PKE requires zero padding of data smaller than the slot length. The symbols below are the actual curve coordinate byte lengths.
	
	#define EC_P192_LEN (0x18ul) // same as above, multiple of 64
	#define EC_P224_LEN (0x1Cul) // not a multiple of 64
	#define EC_P256_LEN (0x20ul) // same as above, multiple of 64
	#define EC_P384_LEN (0x30ul) // same as above, multiple of 64
	#define EC_P521_LEN (0x42ul) // not a multiple of 64
	//
	#define EC_B163_LEN (0x15ul)
	#define EC_B233_LEN (0x1Eul)
	#define EC_B283_LEN (0x24ul)
	#define EC_B409_LEN (0x34ul)
	#define EC_B571_LEN (0x48ul)

	The order of parameters and their description are presented below. ELLIPTIC_CURVE_NPARAMS – 6u (max parameters - 7) 0 – ELLIPTIC_CURVE_PARAM_P 1 – ELLIPTIC_CURVE_PARAM_N 2 – ELLIPTIC_CURVE_PARAM_GX 3 – ELLIPTIC_CURVE_PARAM_GY 4 – ELLIPTIC_CURVE_PARAM_A 5 – ELLIPTIC_CURVE_PARAM_B The indices of parameters correspond to SCM slot numbers.

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – this error value is returned if curve_p points to NULL or if any of the pointers in the curve parameters point to NULL.

PKE_RET_OK – if the requested operation is successful.

Return Code Macro Values:

Macro Name	Value
ECC_OK	0
ECC_ERR_BUSY	1
ECC_ERR_BAD_PARAM	2
ECC_ERR_ZERO_LEN_PARAM	3
ECC_ERR_PARAM_LEN_MISMATCH	4
ECC_ERR_BAD_ADDR	5
ECC_ERR_BAD_CMD	6

4.5.11 ecdsa_verify

Function Header:

```
Uint8_t ecdsa_verify(  const uint8_t * Q,
                       const uint8_t * S,
                       const uint8_t * digest,
                       uint16_t elen,
                       uint16_t dlen,
                       bool msbf);
```

Description:

This routine verifies a signature using standard EC Digital Signature Algorithm.

Inputs:

Input Parameters	Description
Q	An unsigned 8 bit integer pointer to constant data containing Public Key Q.
S	An unsigned 8 bit integer pointer to constant data consisting of signature point S.
digest	An unsigned 8 bit integer pointer to constant data consisting of hash digest of message.
Elen	An unsigned 16 bit integer specifying the length of Qx and Qy. Qx in 0<= index < elen, Qy in elen <= index < 2* elen. Sx in 0<= index < elen, Sy in elen <= index < 2* elen.
dlen	An unsigned 16 bit integer specifying the length of digest.
Msbf	A Boolean value is true, indicates that all parameters are most significant byte first.

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – if Q,S or digest is a NULL pointer, if elen or dlen is zero.

ECC_ERR_BUSY – this error value is returned if PKE is busy.

ECC_ERR_ZERO_LEN_PARAM – This error value is returned if not equal to curve parameter length.

PKE_RET_OK – this value reflects that the requested operation was successful.

Return Code Macro Values:

Macro Name	Value
ECC_OK	0
ECC_ERR_BUSY	1
ECC_ERR_BAD_PARAM	2
ECC_ERR_ZERO_LEN_PARAM	3
ECC_ERR_PARAM_LEN_MISMATCH	4
ECC_ERR_BAD_ADDR	5
ECC_ERR_BAD_CMD	6

4.5.12 ec_kcdsa_keygen

Function Header:

uint8_t ec_kcdsa_keygen(const uint8_t* d, uint16_t plen, uint16_t flags);

Description:

This routine generates EC private key. Caller must have previously programmed elliptic curve into PKE e.g. ec_prog_curve()

Inputs:

Input Parameters	Description
d	Pointer to array containing EC private key
plen	Byte length of d
flags	Byte order bit[0]=0(d is LSBF), 1(d is MSBF)

Outputs:

The return values and their description are presented below.

0=success(PKE started), non-zero=error(PKE not started)

4.5.13 ec_kcdsa_sign

Function header:

```
uint8_t ec_kcdsa_sign(const uint8_t* prv_key, uint16_t plen,
                      const uint8_t* r, uint16_t rlen,
                      const uint8_t* hash, uint16_t hlen,
                      uint16_t flags)
```

Description:

This routine performs signature generation operation. Caller must have previously programmed elliptic curve into PKE e.g. ec_prog_curve()

Inputs:

Input Parameters	Description
Prv_key	Pointer to array containing EC private key
plen	Byte length of prv_key
r	Pointer to array containing r component of signature
rlen	Byte length of r
hash	Pointer to hash digest of message
hlen	Byte length of hash digest

flags	bit[0]=0(prv_key is LSBF), 1(prv_key is MSBF)
	bit[1]=0(r is LSBF), 1(r is MSBF)
	bit[2]=0(digest is LSBF), 1(digest is MSBF)

Outputs:

The return values and their description are presented below.

0=success(PKE started), non-zero=error(PKE not started)

4.5.14 ec_kcdsa_verify

Function Header:

```
uint8_t ec_kcdsa_verify (        const uint8_t * q,
                                 uint16_t qlen,
                                 const uint8_t * sig,
                                 uint16_t slen,
                                 const uint8_t * hash,
                                 uint16_t hlen,
                         uint16_t flags);
```

Description:

This routine performs signature verification operation. Caller must have previously programmed elliptic curve into PKE e.g. ec_prog_curve().

Inputs:

Input Parameters	Description
Q	An unsigned integer pointer to constant data consisting of Qx and Qy.
Qlen	An unsigned 16 bit integer specifying the length of Qx/Qy.
sig	An unsigned integer pointer to constant data consisting of signature.
Slen	An unsigned 16 bit integer specifying the length of signature.
hash	An unsigned integer pointer to constant data consisting of hash digest of message.

Input Parameters	Description
hlen	An unsigned 16 bit integer specifying the length of hash digest.
flags	The bit definitions of flags is presented below: Bit[0] – 0 (Qx,y is LSBF), 1 (Qx,y is MSBF) Bit[1] – 0 (r,s is LSBF), 1 (r,s is MSBF) Bit[2] – 0 (digest is LSBF), 1 (digest is MSBF)

Outputs:

The return values and their description are presented below.

0=success(PKE started), non-zero=error(PKE not started).

4.5.15 src_sc

Function Header:

Uint8_t src_sc(PKE_SRP_DATA * psrp);

Description:

This routine uses PKE to generate step 4 of SRP algorithm. The length provided in the inputs is in units of 64 bytes and in range [0x02, 0x40]

Inputs:

Input Parameters	Description

Psrp	A pointer to the PKE_SRP_DATA structure containing the length and pointers to each of the seven parameters. The structure definition is presented below.
	struct PKE_SRP_DATA{
	uint16_t len64b;
	uint16_t flags;
	uint8_t * param[PKE_MAX_SRP_PARAM];
	};
	The permitted values of flags are
	PKE_SRP_FLAG_LSBF – 0
	PKE_SRP_FLAG_MSBF – 1
	The bit definitions of param are listed below.
	PKE_MAX_SRP_PARAM – The maximum number of parameters (7)
	PKE_SRP_PARAM_P – 0
	PKE_SRP_PARAM_G – 1
	PKE_SRP_PARAM_A - 2
	PKE_SRP_PARAM_B - 3
	PKE_SRP_PARAM_X - 4
	PKE_SRP_PARAM_K - 5
	PKE_SRP_PARAM_U - 6

Outputs:

The return values and their description is presented below.

PKE_RET_ERR_BAD_ADDR – this error value is returned if input parameters points to NULL or if length parameter is invalid (see description).

PKE_RET_OK – this value is returned if requested operation was successful

4.5.16 ec25519_point_mult

Function Header:

```
uint8_t ec25519_point_mult ( const uint8_t * p1x,
                             uint16_t p1x_len,
                             const uint8_t * k,
                             uint16_t k_len,
                             uint16_t flags);
```

Description:

This routine performs an elliptic curve scalar point multiple using the Elliptic Curve 25519.

Inputs:

Input Parameters	Description
P1x	An unsigned 8 bit integer pointer to constant data consisting of p1x.
P1x_len	An unsigned 16 bit integer specifying the length of p1x.
k	An unsigned 8 bit integer pointer to constant data consisting of Scalar k.
K_len	An unsigned 16 bit integer specifying the length of k.
flags	bit[0] = 0(px byte array is LSBF), 1(point is MSBF) bit[1] = 0(k byte array is LSBF), 1(MSBF)

Outputs:

The return values and their description are presented below.

PKE_RET_OK on success, ECC_ERR_BUSY if PKE engine busy, ECC_ER-R_BAD_PARAM if parameters have invalid values

4.5.17 ec25519_xrecover

Function Header:

```
uint8_t ec25519_xrecover(const uint8_t* y, uint16_t ylen, uint16_t flags);
```

Description:

This routine recovers X-coordinate given Y-coordinate

Inputs:

Input Parameters	Description
y	Pointer to parameter y
ylen	Byte length of y
flags	0(point byte array is LSBF), 1(point is MSBF)

Outputs:

0(PKE started), Non-zero(bad parameter(s) error)

4.5.18 ed25519_scalar_mult

Function Header:

```
uint8_t ed25519_scalar_mult(const uint8_t* px, uint16_t pxlen,
                const uint8_t* py, uint16_t pylen,
                const uint8_t* e,
                uint16_t elen,
                uint16_t flags);
```

Description:

Multiply point by a scalar for Elliptic Curve 25519. When done, result located in SCM at Slot[0xA]=x-coordinate, Slot[0xB]=y-coordinate

Inputs:

Input Parameters	Description
Px	Pointer to parameter px
Pxlen	Byte length of px
Py	Pointer to parameter py
Pylen	Byte length of py
E	Pointer to parameter e
Elen	Byte length of e
flags	bit[0] = 0(px byte array is LSBF), 1(point is MSBF) bit[1] = 0(py byte array is LSBF), 1(MSBF) bit[2] = 0(e byte array is LSBF), 1(MSBF)

Outputs:

The return values and their description are presented below.

0(PKE started), Non-zero (bad parameter(s) error)

4.5.19 ed25519_valid_sig

Function Header:

uint8_t pke_ed25519_valid_sig(const Ed25519_SIG_VERIFY* psv)

Description:

Check signature (point) against message string (hash)

Inputs:

Input Parameters	Description

psv	pointer to structure Ed25519_SIG_VERIFY
	
	#define ED_PARAM_AX (0u)
	#define ED_PARAM_AY (1u)
	#define ED_PARAM_RX (2u)
	#define ED_PARAM_RY (3u)
	#define ED_PARAM_SIG (4u)
	#define ED_PARAM_HASH (5u)
	#define ED_PARAM_MAX (6u)
	
	typedef struct {
	    uint8_t* params[ED_PARAM_MAX];
	    uint16_t paramlen[ED_PARAM_MAX];
	    uint16_t flags;
	    uint16_t rsvd;
	} Ed25519_SIG_VERIFY;
	
	Load appropriate parameter to the params[] array
	Params[ED_PARAM_AX] = pointer to the Param Ax
	Params[ED_PARAM_AY] = pointer to the Param Ay
	Params[ED_PARAM_RX] = pointer to the Param Rx
	Params[ED_PARAM_RY] = pointer to the Param Ry
	Params[ED_PARAM_SIG] = pointer to the Param Signature
	Params[ED_PARAM_HASH] = pointer to the Param A
	
	Flags - A Boolean flag which if true writes data in reverse byte order
	        Bit[0] corresponds to Ax
	        Bit[1] corresponds to Ay
	        Bit[2] corresponds to Rx
	        Bit[3] corresponds to Ry
	        Bit[4] corresponds to Signature
	        Bit[5] corresponds to Hash

Outputs:

PKE_RET_OK on success, ECC_ERR_BUSY if PKE engine busy, ECC_ER-R_BAD_PARAM if parameters have invalid values

4.6 RNG FUNCTIONS

4.6.1 rng_power

Function Header:

Void rng_power(bool pwr_on);

Description:

This routine is used for power control of the RNG block.

Inputs:

Input Parameters	Description
Pwr_on	A boolean value if false puts the module to sleep (1 gate off clocks to block), if true enables the block (gate on clocks to block).

Outputs:

None

4.6.2 rng_reset:

Function Header:

Void rng_reset(void);

Description:

This routine resets the RNG block.

Inputs:

None

Outputs:

None

4.6.3 rng_mode

Function Header:

Void rng_mode(uint8_t tmode_pseudo);

Description:

The function controls the mode of RNG. The possible modes are Asynchronous (true random mode), and pseudo random mode.

Inputs:

Input Parameters	Description
Tmode_pseudo	An 8 bit unsigned integer if zero, enables asynchronous mode and if 1 enables pseudo random mode.

Outputs:

None

4.6.4 rng_is_on

Function Header:

Bool rng_is_on(void);

Description:

This function is used to check if the NDRNG block is powered on.

Inputs:

None

Outputs:

Returns true if block is on, false otherwise.

4.6.5 rng_start

Function Header:

Void rng_start(void);

Description:

This routine is used to start the NDRNG engine. Once started, the NDRNG will fill its internal 1Kbit internal FIFO with random bits. The NDRNG block will hang if its FIFO is read while empty. Firmware must poll the NDRNG's FIFO level and only read data from the 32-bit FIFO data register when NOT empty.

Inputs:

None

Outputs:

None

4.6.6 rng_stop

Function Header:

Void rng_stop(void);

Description:

This routine stops the NDRNG engine. When the engine is stopped, the NDRNG will not re-fill its FIFO when data is removed.

Inputs:

None

Outputs:

None

4.6.7 rng_get_fifo_level

Function Header:

Uint32_t rng_get_fifo_level(void);

Description:

This routine reads the NDRNG FIFO level register and returns the number of 32-bit words of random data currently in FIFO. This call must be issued before reading the FIFO and only read FIFO if this call returns a non-zero number.

Inputs:

None

Outputs:

The call returns the number of 32-bit words in the NDRNG FIFO. Maximum value is 32 (32x32 = 1024 bits).

4.6.8 rng_get_bytes

Function Header:

```
uint32_t rng_get_bytes( uint8_t * pbuff8,
                        uint32_t num_bytes);
```

Description:

This routine fills a buffer with random bytes.

Note:	The API reads 32 bits at a time from FIFO. If bytes are requested, a 32 bit word is read and 4 bytes are retrieved. However, only the number of bytes requested is returned.

Inputs:

Input Parameters	Description
Pbuff8	An unsigned 8 bit integer pointer to a buffer where the data will be stored.
Num_bytes	An unsigned 32 bit integer indicating the number of random bytes to be retrieved.

Output:

The number of bytes retrieved is returned.

4.6.9 rng_get_words

Function Header:

Uint32_t rng_get_words(uint32_t * pbuff32,
uint32_t num_words);

Description:

This function reads a specified number of words (32-bit data) into the buffer specified by the caller. This function is an all-in-one routine. Powers on the NDRNG, starts the NDRNG, polls the FIFO level and reads words from the FIFO only if it is not empty. It loops until the specified number of words is read. No time out is implemented, if the NDRNG FIFO hardware stops filling the FIFO, this routine will loop forever.

Inputs:

Input Parameters	Description
Pbuff32	A pointer to word (32-bit) aligned SRAM buffer.
Num_words	Number of 32-bit words of random data to read.

Output:

Returns the actual number of bytes read.

4.7 HASH FUNCTIONS

4.7.1 hash_status

Function Header:

uint32_t hash_status(void);

Description:

This routine returns the status of the HASH block.

Inputs:

None

Outputs:

Returns the status of the HASH block. The status register is read only. The bit definition is provided below.

Bit Number	Description
0	This bit reflects AHB error. If 0, there is no error. If 1, it indicates that AHB error has occurred.

4.7.2 hash_busy

Function Header:

Bool hash_busy(void);

Description:

This routine is used to check if the HASH block is busy.

Inputs:

None

Outputs:

Returns a Boolean value which, if true, indicates that the block is busy.

4.7.3 hash_start

Function Header:

void hash_start(bool ien);

Description:

This routine is used to start the HASH engine. Once started, the GIRQ16 bit 4 reflects the done status (1 if done). It must be cleared after it is set.

Inputs:

Input Parameters	Description
ien	A Boolean value indicating the state of interrupts.

Outputs:

None

4.7.4 hash_done_status

Function Header:

Bool hash_done_status(uint32_t * status_value);

Description:

This routine is used to check the done status of HASH block. The status register value is updated into the pointer passed by the buffer.

Input:

Input Parameter	Description
Status value	An unsigned 32 bit integer pointer where the status value will be stored. The bit definitions are listed below. Bit Numbers Description 31:16 Hash Status Register Value 15:0 Hash Control Register Value

Outputs:

The return value is true if the done status, false otherwise.

4.8 SHA FUNCTIONS

4.8.1 sha12_init

Function Header:

uint8_t sha12_init(SHA12_CONTEXT_T * sha12_ctx,
uint8_t sha_mode);

Description:

This routine initializes the SHA12_CONTEXT_T Data structure for the specified mode. This routine does not effect a change on the Hash Hardware.

Inputs:

Input Parameters	Description

Sha12_ctx	A pointer to the SHA context structure. The structure definition is given below.
	Struct SHA12_CONTEXT_T{
	SHA12_DIGEST_U hash;
	    Union{
	            uint32_t w[(SHA12_BLOCK_WLEN) * 2]; // SHA12_-BLOCK_WLEN = 16
	            uint8_t b[(SHA12_BLOCK_BLEN)*2]; // SHA12_-BLOCK_BLEN = 64
	            }block;
	Uint8_t mode;
	Uint8_t block_len;
	Uint8_t rsvd[2];
	Uint64_t total_msg_len;/uint32_t total_msg_len;
	};
	
	The structure definition of SHA12_DIGEST_U is given below.
	Union{
	Uint32_t w[SHA2_WLEN]; //SHA2_WLEN = 8
	Uint8_t b[SHA2_BLEN]; //SHA2_BLEN = 32
	};

Sha_mode	This indicates the mode of SHA. The permitted modes are SHA_MODE_256 SHA_MODE_1

Outputs:

The return values and their descriptions are presented below.

SHA_RET_ERR_BAD_ADDR – this error is returned if sha12_ctx points to NULL.

SHA_RET_ERR_UNSPPORTED – this error value is returned as sha_mode and has a value other than the permitted ones.

SHA_RET_OK – this value is returned if the operation requested is successful.

Return Code Macro Values:

Macro Name	Value
SHA_RET_OK	0
SHA_RET_START	1
SHA_RET_ERR_BUSY	0x80
SHA_RET_ERR_BAD_ADDR	0x81
SHA_RET_ERR_TIMEOUT	0x82
SHA_RET_ERR_MAX_LEN	0x82
SHA_RET_ERR_UNSUPPORTED	0x84

4.8.2 sha12_update

Function Header:

uint8_t sha12_update(SHA12_CONTEXT_T * sha12_ctx,
const uint32_t * data,
uint32_t data_byte_len);

Description:

This routine runs Hash block on data and updates the SHA12_CONTEXT_T data structure with the number of bytes processed. The data must be aligned to a 4-byte boundary for SHA1 or SHA256. If data length is not a multiple of 64-bytes, then the remaining bytes will be copied into SHA12_CONTEXT_T data structure to be processed by sha12_finalize.

Inputs:

Input Parameters	Description

Sha12_ctx	A pointer to the SHA context structure. The structure definition is given below.
	Struct SHA12_CONTEXT_T{
	SHA12_DIGEST_U hash;
	Union{
	Uint32_t w[(SHA12_BLOCK_WLEN) * 2]; // SHA12_-BLOCK_WLEN = 16
	Uint8_t b[(SHA12_BLOCK_BLEN)*2]; // SHA12_-BLOCK_BLEN = 64
	}block;
	Uint8_t mode;
	Uint8_t block_len;
	Uint8_t rsvd[2];
	Uint64_t total_msg_len;/uint32_t total_msg_len;
	};
	The structure definition of SHA12_DIGEST_U is given below.
	Union{
	Uint32_t w[SHA2_WLEN]; //SHA2_WLEN = 8
	Uint8_t b[SHA2_BLEN]; //SHA2_BLEN = 32
	};

Data	An unsigned 32 bit integer pointer to constant data consisting of data to be updated
Data_byte_len	An unsigned 32 bit integer specifying the length of data.

Output:

The return values and description is presented below.

SHA_RET_ERR_BAD_ADDR – this error value is returned if either sha12_ctx or data points to NULL.

SHA_RET_ERR_BUSY – this error value is returned if the HASH module is busy.

SHA_RET_ERR_MAX_LEN – this error value is returned if total_msg_len is greater than SHA12_MSG_LEN_MAX.

SHA_RET_OK – this value is returned if the requested operation is successful.

4.8.3 sha12_finalize

Function Header:

Uint8_t sha12_finalize(SHA12_CONTEXT_T * sha12_ctx);

Description:

This routine applies FIPS padding to SHA256 and performs final hash calculations. It must be used in sequence, sha256_init, sha256_update_start, wait for hash engine to finish, sha256_finalize. The SHA256_CONTEXT_T object will be filled in with any remaining bytes on the last call to sha256_update_start. SHA engine is approximately 1 cycle per byte for SHA1 and SHA256. It is 64 cycles per 64-byte block.

If the original message length has greater than 56 remaining bytes, the API will need to hash two additional blocks, one for the remaining 56 bytes and one to hold the message bit length.

Inputs:

Input Parameter	Description

Sha12_ctx	A pointer to the SHA context structure. The structure definition is given below.
	Struct SHA12_CONTEXT_T{
	SHA12_DIGEST_U hash;
	Union{
	Uint32_t w[(SHA12_BLOCK_WLEN) * 2]; // SHA12_-BLOCK_WLEN = 16
	Uint8_t b[(SHA12_BLOCK_BLEN)*2]; // SHA12_-BLOCK_BLEN = 64
	}block;
	Uint8_t mode;
	Uint8_t block_len;
	Uint8_t rsvd[2];
	Uint64_t total_msg_len;/uint32_t total_msg_len;
	};
	The structure definition of SHA12_DIGEST_U is given below.
	Union{
	Uint32_t w[SHA2_WLEN]; //SHA2_WLEN = 8
	Uint8_t b[SHA2_BLEN]; //SHA2_BLEN = 32
	};

Outputs:

The return values and description are presented below.

SHA_RET_ERR_BAD_ADDR – this error value is returned if sha12_ctx points to NULL.

SHA_RET_ERR_BUSY – this error value is returned if HASH block is busy.

SHA_RET_START – this value is returned if the operation was successful.

4.8.4 sha35_init

Function Header:

**uint8_t sha35_init(SHA35_CONTEXT_T * sha35_ctx,
uint8_t sha35_mode);**

Description:

This routine initializes the SHA35_CONTEXT_T data structure for the mode specified.

Inputs:

Input Parameters	Description

Sha35_ctx	A pointer to the SHA35_CONTEXT_T data structure. The structure definition is given below
	Struct SHA35_CONTEXT_T{
	SHA35_DIGEST_U hash;
	union {
	uint32_t w[(SHA35_BLOCK_WLEN) * 2]; // 32
	uint32_t b[(SHA35_BLOCK_BLEN) * 2]; // 128
	}blocks;
	Uint8_t mode;
	Uint8_t block_len;
	Uint8_t rsvd[2];
	Uint64_t total_msg_len; / uint32_t total_msg_len;
	};

Sha35_mode	The permitted mode for this operation is SHA_MODE_512

Outputs:

The return values and their descriptions are given below.

SHA_RET_ERR_BAD_ADDR – This error value is returned as sha35_ctx and points to NULL.

SHA_RET_ERR_UNSUPPORTED – This error is returned if sha35_mode contains an invalid mode.

SHA_RET_OK – This value is returned if the function was successful.

4.8.5 sha35_update

Function Header:

**uint8_t sha35_update(SHA35_CONTEXT_T * sha35_ctx,
const uint32_t * data,
uint32_t data_byte_len);**

Description:

This routine runs HASH block on data and updates SHA35_CONTEXT_T data structure with the number of bytes. The data must be aligned to a 4-byte boundary for SHA1 or SHA256. If data length is not a multiple of 64-bytes, then the remaining bytes will be copied into SHA35_CONTEXT_T data structure to be processed by sha35_finalize.

Inputs:

Input Parameters	Description

Sha35_ctx	A pointer to the SHA35_CONTEXT_T data structure. The structure definition is given below
	Struct SHA35_CONTEXT_T{
	SHA35_DIGEST_U hash;
	union {
	uint32_t w[(SHA35_BLOCK_WLEN) * 2]; // 32

uint32_t b[(SHA35_BLOCK_BLEN) * 2]; // 128
}blocks;
Uint8_t mode;
Uint8_t block_len;
Uint8_t rsvd[2];
Uint64_t total_msg_len; / uint32_t total_msg_len;
};	

Data	An unsigned 32 bit integer pointer consisting of data.
Data_byte_len	An unsigned 32 bit integer containing the length of data.

Outputs:

The return values and their descriptions are presented below.

SHA_RET_ERR_BAD_ADDR – This error value is returned if sha35_ctx or data points to NULL.

SHA_RET_ERR_BUSY – This error value is returned if HASH block is busy.

SHA_RET_OK – This value is returned if the operation is successful.

4.8.6 sha35_finalize

Function Header:

Uint8_t sha35_finalize(SH35_CONTEXT_T * sha35_ctx);

Description:

Finalizes the Hash operations by running Hash engine if bytes are left over and adds FIPS padding.

Inputs:

Input Parameters	Description

Sha35_ctx	A pointer to the SHA35_CONTEXT_T data structure. The structure definition is given below
	Struct SHA35_CONTEXT_T{
	SHA35_DIGEST_U hash;
	union {
	uint32_t w[(SHA35_BLOCK_WLEN) * 2]; // 32
	uint32_t b[(SHA35_BLOCK_BLEN) * 2]; // 128
	}blocks;
	Uint8_t mode;
	Uint8_t block_len;
	Uint8_t rsvd[2];
	Uint64_t total_msg_len; / uint32_t total_msg_len;
	};

Outputs:

The return values and their descriptions are presented below.

SHA_RET_ERR_BAD_ADDR – this error value is returned if sha35_ctx points to NULL.

SHA_RET_ERR_BUSY – this error value is returned if HASH block is busy.

SHA_RET_OK – this value is returned if the operation is successful.

4.8.7 hash_iclr

Function Header:

uint32_t hash_iclr(void)

Description:

This function is used to clear HASH interrupts.

Inputs:

None

Output:

The return value is the status register of HASH block.

Hash status – bit 0 =1(hash block is busy, status bit set), bit 0 = 0 (hash status is clear).

4.8.8 sha_init

Function Header:

```
Uint8_t sha_init(        uint8_t mode,
                         Uin32_t * digest);
```

Description:

This routine initializes the HASH engine for SHA operation. Programs supported SHA operation's initial value, digest address and operation (SHA1, SHA256, SHA384 or SHA512). Hash engine does not need to be started. SHA1 and SHA256 require 4 byte alignment. SHA384 and SHA512 require 8-byte alignment.

Inputs:

Input Parameters	Description
Mode	An unsigned 8 bit integer indicating the mode. Permitted modes are SHA_MODE_1, SHA_MODE_256, SHA_-MODE_512.
Digest	An unsigned 32 bit integer pointer to digest.

Outputs:

The return values and their description is presented below.

0 = Success

1 = Hash Engine Busy

2 = Unsupported SHA operation

3 = Bad digest pointer, NULL or mis-aligned.

4.8.9 sha_update

Function Header:

```
uint8_t sha_update(  uint32_t * pdata,
                     uint16_t nblocks,
                     uint8_t flags);
```

Description:

This routine programs HASH engine with data address and the number of data blocks to process. Sha block must be initialized before this routine is called. SHA1 and SHA256 require 4 byte alignment. SHA384 and SHA512 require 8-byte alignment. If HASH engine is not started and if return value is non-zero caller must call hash_start().

Inputs:

Input Parameters	Description
Pdata	An unsigned 32 bit integer pointer to data.
Nblocks	An unsigned 16 bit integer specifying the number of blocks.
Flags	Bit 0 indicates clear status(1-clear), bit enable interrupt status(1-enable) and bit indicates start/stop (1-start).

Outputs:

The return values and their description are presented below.

0 = Success

1 = Hash engine busy

2 = pdata is a null pointer

3 = Data is misaligned

4.8.10 sha_final

Function Header:

```
uint8_t sha_final(   uint32_t * padbuf,
                     uint32_t total_msg_len,
                     const uint8_t * perm,
                     uint8_t flags);
```

Description:

This routine implements the standard SHA padding described in the FIPS standard.

Inputs:

Input Parameters	Description
Padbuf	An unsigned 32 bit integer pointer consisting of buffer for padding.
Total_msg_len	An unsigned 32 bit integer specifying the length of message.
Perm	An unsigned 8 bit integer pointer to constant data pointer.
Flags	An unsigned 8 bit integer. Bit 0 indicates clear status(1-clear), bit enable interrupt status(1-enable) and bit indicates start/stop (1-start).

Outputs:

The return values and their descriptions are presented below.

0 = Success

1 = Hash engine busy

2 = pdata is a null pointer

3 = Data is misaligned

4.9 MISCELLANOUS ROM API

4.9.1 version

Function Header:

uint32_t version(void);

Description:

This routine returns the version number of the ROM API's.

Inputs:

None

Outputs:

The return value is an unsigned 32 bit integer reflecting the build information of ROM API's.

4.9.2 loader

Function Header:

```
uint32_t loader(uint32_t config,
                LOAD_DESCR* pldr,
                uint32_t* p256_ecdsa_pub,
                uint32_t* p256_ecdh_prv,
                uint32_t* buff2k);
```

Description:

This routine performs the firmware load process.

Inputs:

Input Parameters	Description
config	b[7:0] = interface 0 = Use ROM POR load interface 1 = Shared SPI 2 = Private SPI 3 = eSPI 4 = Internal SPI b[9:8] = SPI Freq MHz 0=48, 1=24, 2=16, 3=12 (N/A for eSPI) b[11:10] = SPI Drive Strength (N/A for eSPI) b[12] = SPI Slew Rate (N/A for eSPI) b[15:13] = 0 reserved b[19:16] = DMA channel (0-13) (N/A for eSPI) b[23:20] = 0 reserved b[30] = 0 Do not return to caller. 1 Return to caller b[31] = 0 ROM takes over interrupts (vector table set to ROM table) 1 caller retains ownership of interrupt vector table

pldr	structure variable of type LOAD_DESCR
	typedef struct {
	uint32_t ld_addr;
	uint32_t byte_len;
	uint32_t spi_addr;
	uint32_t entry_addr;
	} LOAD_DESCR;

p256_ecdsa_pub	pointer to ecdsa public key , used in case firmware is autenticated
p256_ecdh_prv	pointer to ecdh private key , used in case firmware is encrypted.
buff2k	buffer required for implementing the function, aligned on a 16 byte boundary; minimum required size is 4 bytes.

Outputs:

The return value is 32 bit integer and the description is presented below.

bit[31] == 1 indicates an error

bit[31] == 0, bits[30:0] = loaded application entry point address

Chapter 5. API Usage

The following section lists the APIs available and their usage.

5.1 SHA APIS

1. Power on SHA block with *aes_hash_power ().*
2. Check if Hash Block is busy, if not busy proceed. If busy, wait until busy status is false (*hash_busy()*)
3. Run *sha_init* with the required mode (1,256 or 512) and the pointer to the buffer where the digest will be stored. The buffer must be 4 byte aligned for SHA1 and SHA256 and 8 byte aligned for SHA512.
4. Run *sha_update* with a buffer pointer to message on which digest is to be calculated. The message must align with the block size requirement for the mode being calculated (64 byte block for SHA1 and SHA256, 128 byte block for SHA512). The number of blocks of data provided must be mentioned.
5. If start hash block was not specified in the flags in the previous call, call *hash_start*() to start the Hash engine.
6. Wait until hash operation is complete (*hash_busy()*)
7. Run *sha_final* with a buffer of size at least one block and message length.
8. Wait until hash operation is complete (*hash_busy()*).
9. The digest calculated will be in the buffer specified in step 2.

5.2 SHA12 APIS

1. Power on SHA block with *aes_hash_power ().*
2. Check if Hash Block is busy, if not busy proceed. If busy, wait until busy status is false (*hash_busy())*
3. Call *sha12_init* with the SHA12_CONTEXT_T data structure and the mode required. SHA12 API's only support SHA1 and SHA256.
4. Call *sha12_update* with the data structure, the message for which digest is to be calculated and the length of the message. The input need not adhere to block size requirement.
5. Wait on hash busy status (*hash_busy()*). Once hash block is free, proceed.
6. Call *sha12_finalize* with the data structure provided previously.
7. Wait for Hash operation to complete (*hash_busy()*).
8. The calculated digest will be in the data structure provided (sha12_ctx.hash.b or sha12_ctx.hash.w).

5.3 SHA35 APIS

1. Power on SHA block with *aes_hash_power ().*
2. Check if Hash Block is busy, if not busy proceed. If busy, wait until busy status is false (*hash_busy())*

3. Call *sha35_init* with SHA35_CONTEXT_T data structure and the mode required. Sha35 API's only support SHA512.
4. Call *sha35_update* with the data structure, the message on which hash is to be calculated and the length of the message. The input need not adhere to block size requirement.
5. Wait until hash block is not busy (*hash_busy()*).
6. Call *sha35_finalize* with the data structure provided previously.
7. Wait until Hash operation is complete (*hash_busy()*).
8. The calculated digest will be available in the provided data structure (sha35_ctx.hash.b or sha35_ctx.hash.w).

5.4 RSA APIS

1. Power on PKE block with *pke_power().*
2. Call the *rsa_load_key()* to load Public-Private Key pairs into the PKE engine. Specify the RSA byte length (1024, 2048 or 4096) and the byte order of data provided. 4 combinations are possible according to which keys may be loaded. The slot numbers are handled by the API. Keys can be explicitly programmed with *pke_write_scm()* calls.
   a) RSA Encryption with Public Key
      i. Pointer to private exponent = Not used
      ii. Pointer to public modulus = your public key modulus -> Slot 0
      iii. Pointer to public exponent = your public key exponent -> Slot 8
   b) RSA Decryption with Private Key
      i. Pointer to private modulus = your private key modulus -> Slot 6
      ii. Pointer to public modulus = your public key modulus -> Slot 0
      iii. Pointer to public exponent = your public key exponent -> Slot 8
   c) RSA Encryption with Private Key
      i. Pointer to private exponent = Not used
      ii. Pointer to public modulus = your public key modulus -> Slot 0
      iii. Pointer to public exponent = your private key exponent -> Slot 8
   d) RSA Decryption with Public Key
      i. Pointer to private exponent = your private exponent -> Slot 6
      ii. Pointer to public modulus = your public modulus -> Slot 0
      iii. Pointer to public exponent = Not used
3. If data is to be encrypted, call *rsa_encrypt()* with the rsa bit len (1024, 2048 or 4096), pointer to structure having byte length of input data & pointer to input data. Start the PKE engine by calling *pke_start()*.
4. Wait for PKE engine to complete operation (*pke_busy()*). Once complete, the encrypted message can be found in slot 5 of crypto memory. The data can be read into a local buffer with *pke_read_scm()*.
5. If data is to be decrypted, there are two methods – RSA decryption and CRT RSA decryption.
   RSA Decryption:
   a) call rsa_decrypt() with rsa bit len (1024, 2048 or 4096), pointer to structure having byte length of encrypted data & pointer to encrypted data. Start the PKE engine by calling *pke_start()*.

CRT RSA Decryption:

b) call rsa_load_crt_params() api with first exponent, second exponent, coefficient to load these parameters into appropriate slots in the shared crypto memory(scm slots). Folllowing this, call pke_write_scm api to load the two prime numbers to scm slots 2 and 3. Finally call pke_rsa_crt_decrypt with bit len (1024, 2048 or 4096), input data, byte length of input data. Pke engine can be started by specifying the appropriate flag parameter. If not started, the pke engine may be explicitly started with *pke_start().*

c) call rsa_crt_gen_params() api with first prime number, second prime number and private exponent. Wait for PKE engine to complete operation (*pke_busy()*). Call pke_rsa_crt_decrypt with bit len (1024, 2048 or 4096), input data, byte length of input data.The pke engine is to be started with *pke_start().*

6. Wait for PKE engine to complete operation (*pke_busy()*). Once complete, the decrypted data will be in slot 5 of Shared Crypto memory. This data can be read into a local buffer with *pke_read_scm()*.
7. RSA Signature Generation and Verification:
   i) Call the *rsa_load_key()* to load Public-Private Key pairs into the PKE engine. Specify the RSA bite length (1024, 2048 or 4096) and the byte order of data provided.
   ii) Wait for PKE engine to complete operation(pke_busy()).
   iii) Call rsa_signature_gen() with bit length, hash digest, specifying the byte order of data provided.
   iv) Signature for the given hash digest will be generated in scm slot 5.
   v) After completion of PKE operation(pke_busy()), copy expected hash digest to slot C using pke_write_scm().
   vi) Call rsa_signature verify() with pointer to signature generated in the above process.
   vii) After completion of PKE operation(pke_busy()), read PKE status register by api pke_done_status(). If bit 9 is set in the status register, it indicates that the signature is not valid for the given expected hash digest.
   viii) The regenerated hash digest calculated by the PKE engine may be read from slot 5.

5.5 AES

1. Power on AES block with *aes_hash_power ().*
2. Reset the AES hash block with *aes_hash_reset*().
3. Check for AES block busy with *aes_busy():*If not busy perform the following
4. Set AES Private key LSB first random generated and optional initialization vector LSB first, also specify the AES key length used using api *aes_set_key()*
5. Check for AES status with ROM API *aes_status()*
   a) clear status for any leftover status bits if any using API *aes_iclr()*
6. Call API *aes_crypt()* for encryption or decryption providing the message should be aligned input data buffer and pointer buffer to load aligned output data buffer and mode of operation; supported mode ECB CBC CTR CFB OFB
7. Start the AES operation to be performed by calling function *aes_start()*
8. Wait for the done status by calling API *aes_done_status()*

9. Once done the operated data output will be in the buffer provided vis API *aes_crypt()*
10. *Stop AES block using the API aes_stop()*
11. For power saving and putting the AES block in the sleep state using routine *aes_hash_power(false);*

5.6 RANDOM NUMBER GENERATOR

1. Power on RND HW block with API *rng_power (true).*
2. Reset the RND HW block with *rng_reset()*
3. Two modes of random number are generated asynchronous/true random mode and Non-zero(pseudo-random mode)
4. For random number generation use the function call as follows
5. Select the mode of operation by calling *rng_mode(mode)*
   a) 0 – asynchronous
   b) 1 - pseudo-random mode
6. Start the HW block run state by calling function *rng_start();*
7. Wait for operation completion by polling *rng_get_fifo_level()* for data in the internal buffer.
8. Once complete, the internal buffer will have 1K bits of random data, then use APIs
   a) *rng_get_bytes()* number of random bytes to retrieve. Must be less or equal to the size of the buffer or
   b) *rng_get_words()* Reads the FIFO level register and returns the number of 32-bit words of random data currently in the FIFO – max value supported is 1024 bits

5.7 ECDSA VERIFICATION

1. Generate SHA Digest of the message to be validated. (Optional if digest already exists).
2. Check if PKE block is busy (*pke_busy()*). If not busy, proceed.
3. Call the ecdsa_verify() API with the Public key and the signature of the message and the digest calculated.
4. Start the PKE Engine by calling *pke_start()*.
5. Wait until PKE operations are done by polling on *pke_done_status(PKE_STATUS)*.
6. Check the 9th bit of *PKE_STATUS*. If it is reset, Signature is valid, if set, the Signature is Invalid.

<u>ECDSA Point Operations:</u>
<u>The procedure for EC point operations like ec_point_add, ec_point_double, ec_point_scalar_mult2, ec_point_scalar_mult3 is explained below.</u>

1. Call pke_power() to power on the block.
2. For ECDSA point operations, the curve should be programmed to slots using API ec_prog_curve().
3. Check PKE engine ready using API pke_busy().
4. Program curve parameters to slots using ec_prog_curve().
5. Set slot numbers to operand pointers A,B,C using pke_set_operand_slots() API. Operand pointers A and B correspond to input data to the EC point operations.

The output of the operation is pointed by pointer C. pke_set_operand_slots() API instructs the PKE engine the slot numbers where it should look for operands A and B, and also where it should store output C. The slot numbers used as defaults for pointers A,B,C are 6,8,C respectively.

6. Call API to appropriate point operation with required parameters, data byte order.
7. The output of operation may be read from slot number corresponding to pointer C (slot C if using default values).

Curve 25519 Operations:

ec25519_xrecover:

1. Call pke_power() to power on the block.
2. Call ec25519_xrecover() API with y coordinate, size and byte order.
3. Call pke_start() and wait for operation to complete(pke_busy()).
4. The recovered x coordinate is always loaded to scm slot 6. Read x coordinate using call to pke_read_scm().

ed25519_scalar_mult:

1. Call pke_power() to power on the block.
2. Call ed25519_scalar_mult() API with point on curve 25519, scalar and byte order.
3. Call pke_start() and wait for operation to complete(pke_busy()).
4. The x and y coordinates of product are always loaded to SCM slots A and B. Read using call to pke_read_scm().

ed25519_valid_sig:

1. Call pke_power() to power on the block.
2. Call ed25519_valid_sig() API with structure variable of type Ed25519_SIG_-VERIFY.
3. Call pke_start() and wait for operation to complete(pke_busy()).
4. The parameters P1x,P1y,P2x,P2y,P3x,P3y are loaded to slots A,B,C,D,E,F respectively.
5. Verify validity of signature by comparing P1 and P3.

Chapter 6. Build and Link

Use the provided symdef file and API header file for proper linking of the application code with the bootrom.

If running on FPGA download the bootcode for proper linking of the object binary on runtime for the API calls.

Use the linker script for loading the bootcode binary using the load incremental option.

Chapter 7. Timing Analysis

For all the crypto operations mentioned in section 4 timing measurement is done at CPU clock of 48MHz and the results are below.

7.1 AES

BLOCK	AES MESSAGE LENGTH = 2048 bytes KEY LENGTH = 256 bits									
BLOCK CONFIGURATION	**ECB**		**CBC**		**CTR**		**CFB**		**OFB**	
BLOCK OPERATION	ENCRYPTION	DECRYPTION	ENCRYPTION	DECRYPTION	ENCRYPTION	DECRYPTION	ENCRYPTION	DECRYPTION	ENCRYPTION	DECRYPTION
TIME (usec) = (1/CPU_CLK) * (CPU	146.55	215.64	146.61	215.61	143.85	215.19	143.88	215.16	143.85	214.05
TOTAL TIME (usec)	362.19		362.22		359.04		359.04		357.1	

7.2 ECDSA

	ECDSA MESSAGE LENGTH = 2048 bytes PUBLIC KEY LENGTH = 64 bytes SIGNATURE LENGTH = 64 bytes
BLOCK CONFIGURATION	SHA256
BLOCK OPERATION	
TIME (usec) = (1/CPU_CLK) * (CPU CYCLES)	9280
TOTAL TIME (usec)	9280

7.3 PKE

	PKE RSA BIT LENGTH = 1024 BITS INPUT MSG LENGTH = 2048 BYTES	
BLOCK CONFIGURATION	RSA ENCRYPTION WITH PUBLIC KEY DECRYPTION WITH PRIVATE KEY	RSA ENCRYPTION WITH PRIVATE KEY DECRYPTION WITH PUBLIC KEY
BLOCK OPERATION		
TIME (usec) = (1/CPU_CLK) * (CPU CYCLES)	156346	157719
TOTAL TIME (usec)	156346	157719

7.4 SHA

BLOCK	SHA INPUT MSG LENGTH = 2048 BYTES		
BLOCK CONFIGURATION	SHA1	SHA12 (MODE 256)	SHA35 (MODE 512)
BLOCK OPERATION			
TIME (usec) = (1/CPU_CLK) * (CPU	477.24	478.74	425.55
TOTAL TIME (usec)	477.24	478.74	425.55

7.5 RNG

	RNG	
BLOCK CONFIGURATION	TRUE RANDOM NUMBER	PSEUDO RANDOM NUMBER
BLOCK OPERATION		
CPU CYCLES	3285	6407
TIME (usec) = (1/CPU_CLK) * (CPU CYCLES)	68.43	133.47
TOTAL TIME (usec)	68.43	133.47

Chapter 8. PKE Slot Usage

CEC/MEC family devices have dedicated crypto SRAM (shared crypto memory - SCM) for PKE block usage. This memory is shared by various PKE operations and hence it limits the operations which can be carried out in parallel. Some of the operations supported by PKE are:

- Primitive arithmetic operation
- RSA Cryptosystem
- Curve25519
- ECDSA

The SCM is used to program parameters & keys and to upload/download operands/results from the host side. The shared crypto memory of CEC/MEC family devices is divided into 31 slots (slot0-slot30) each of 512 bytes.

The table below lists the usage of the slots for various operations.

PKE Operation	Slots Used
RSA Encryption with Public Key	Slot 8 - public exponent Slot 0 – public modulus Slot 5 – Encryption output
RSA Encryption with Private Key	Slot 8 – private exponent Slot 0 – public modulus Slot 5 – Encryption output
RSA Decryption with Private key	Slot 0 – public modulus Slot 6 – private exponent Slot 8 – public exponent Slot 5 – Decryption output
EC program curve	Slot 0 – prime Slot 1 – order Slot 2 - generator point x-coordinate Slot 3 - generator point y-coordinate Slot 4 – curve parameter a Slot 5 – curve parameter b
EC25519 recover x coordinate	Slot 6 – result
PKE clear slot	Specified at run time
PKE read slot	Specified at run time
PKE write slot	Specified at run time
EC Modular arithmetic	Specified at run time

Since above operations share multiple slots these PKE crypto operations cannot be run in parallel

Appendix A. Rom Symdef Table for API Support

```
#<SYMDEFS># ARM Linker, 5.05 [Build 169]: Last Updated: Tue Jul 19 19:39:39
2016
0x00006828 T spi_port_sel
0x00006898 T spi_port_drv_slew
0x00006c5c T rom_dis_lock_shd_spi
0x00006f5c T qmspi_init
0x00006cf8 T qmspi_freq_get
0x00006d18 T qmspi_freq_set
0x00006d30 T qmspi_xfr_done_status
0x00006edc T qmspi_start
0x00006f00 T qmspi_start_dma
0x00006f68 T qmspi_cfg_spi_cmd
0x00006f6c T qmspi_read_dma
0x00006f7c T qmspi_write_dma
0x00006f8c T qmspi_xmit_cmd
0x00006f28 T qmspi_read_fifo
0x00006f90 T aes_hash_power
0x00006fb4 T aes_hash_reset
0x00006ff0 T aes_busy
0x00006fe4 T aes_status
0x00007014 T aes_done_status
0x00007044 T aes_stop
0x0000705c T aes_start
0x00007370 T aes_iclr
0x000071e4 T aes_set_key
0x000072d4 T aes_crypt
0x000082dc T rsa_load_key
0x00008350 T rsa_load_crt_params
0x000083c4 T rsa_keygen
0x00008464 T rsa_modular_exp
0x0000850c T rsa_encrypt
0x0000856c T rsa_decrypt
0x000085cc T rsa_crt_gen_params
0x00008678 T rsa_crt_decrypt
0x000086d8 T rsa_signature_gen
0x00008738 T rsa_signature_verify
0x00007780 T pke_power
0x0000779c T pke_reset
0x00007700 T pke_status
0x00007758 T pke_done_status
0x000077cc T pke_start
0x0000770c T pke_ists_clear
```

```
0x00007730 T pke_busy
0x00007808 T pke_set_operand_slot
0x000077f0 T pke_get_operand_slot
0x00007830 T pke_set_operand_slots
0x0000786c T pke_get_slot_addr
0x00007880 T pke_fill_slot
0x000078a0 T pke_clear_scm
0x00007844 T pke_scm_clear_slot
0x000079ec T pke_read_scm
0x0000790c T pke_write_scm32
0x00007984 T pke_write_scm
0x00007a68 T ec_point_double
0x00007ad0 T ec_point_add
0x00007be4 T ec_point_scalar_mult2
0x00007c70 T ec_point_scalar_mult3
0x00007da4 T ec_check_poc2
0x00007e0c T ec_check_poc3
0x00007cf4 T ec_check_point_less_prime
0x00007d54 T ec_check_ab
0x00007d7c T ec_check_n
0x00008200 T modular_arithm
0x00007ee4 T ec_kcdsa_keygen
0x00007f30 T ec_kcdsa_sign
0x00007e70 T ec_prog_curve
0x00002434 T ec_kcdsa_verify
0x000024f0 T src_sc
0x00008158 T ecdsa_verify
0x000074c4 T ec25519_point_mult
0x0000753c T ec25519_xrecover
0x000075b4 T ed25519_scalar_mult
0x00007654 T ed25519_valid_sig
0x000073f0 T rng_power
0x00007388 T rng_reset
0x00007424 T rng_mode
0x00007404 T rng_is_on
0x000073c4 T rng_start
0x000073d4 T rng_stop
0x000073e4 T rng_get_fifo_level
0x00007440 T rng_get_bytes
0x00007488 T rng_get_words
0x000087ec T hash_status
0x000087dc T hash_busy
0x000087b8 T hash_start
0x0000881c T hash_done_status
0x00008860 T sha12_init
0x0000891c T sha12_update
0x000089dc T sha12_finalize
0x00008bb4 T sha35_init
0x00008bfc T sha35_update
0x00008cbc T sha35_finalize
```

```
0x00008da0 T hash_iclr
0x00008db4 T sha_init
0x00008e70 T sha_update
0x00008ef4 T sha_final
0x0000694c T version
0x00006954 T loader
0x00009028 D ec_sect571r1
0x00009044 D ec_sect409r1
0x00009060 D ec_sect283r1
0x0000907c D ec_sect233r1
0x00009098 D ec_sect163r2
0x000090d0 D ec_secp521r1
0x000090ec D ec_secp384r1
0x00009108 D ec_secp256r1
0x00009124 D ec_secp224r1
0x00009140 D ec_secp192r1
```

MICROCHIP

Worldwide Sales and Service

AMERICAS

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

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

Austin, TX
Tel: 512-257-3370

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

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

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

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

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

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

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

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

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

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

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

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

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-3326-8000
Fax: 86-21-3326-8021

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

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

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

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

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

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

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

France - Saint Cloud
Tel: 33-1-30-60-70-00

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-67-3636

Germany - Karlsruhe
Tel: 49-721-625370

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

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

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

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

Italy - Padova
Tel: 39-049-7625286

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

Norway - Trondheim
Tel: 47-7289-7561

Poland - Warsaw
Tel: 48-22-3325737

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

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

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

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

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

11/07/16