­­­

|  |  |
| --- | --- |
| MEC2016 ROM API’s | |
| MEC2016 Rom RevB API Reference Manual | |
| Rev 2.10 | 04/06/2017 |

MICROCHIP CONFIDENTIAL

|  |
| --- |
| Copyright © 2013 Microchip or its subsidiaries. All rights reserved.  The information contained herein is confidential and proprietary to Microchip, shall be used solely in accordance with the agreement pursuant to which it is provided, and shall not be reproduced or disclosed to others without the prior written consent of Microchip. Although the information is believed to be accurate, no responsibility is assumed for inaccuracies. Microchip reserves the right to make changes to this document and to specifications and product descriptions at any time without notice. Neither the provision of this information nor the sale of the described semiconductor devices conveys any licenses under any patent rights or other intellectual property rights of Microchip or others. The product may contain design defects or errors known as anomalies, including but not necessarily limited to any which may be identified in this document, which may cause the product to deviate from published specifications. Microchip products are not designed, intended, authorized or warranted for use in any life support or other application where product failure could cause or contribute to personal injury or severe property damage. Any and all such uses without prior written approval of an officer of Microchip will be fully at the risk of the customer.    MICROCHIP DISCLAIMS AND EXCLUDES ANY AND ALL WARRANTIES, INCLUDING WITHOUT LIMITATION ANY AND ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND AGAINST INFRINGEMENT AND THE LIKE, AND ANY AND ALL WARRANTIES ARISING FROM ANY COURSE OF DEALING OR USAGE OF TRADE. IN NO EVENT SHALL SMSC BE LIABLE FOR ANY DIRECT, INCIDENTAL, INDIRECT, SPECIAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES; OR FOR LOST DATA, PROFITS, SAVINGS OR REVENUES OF ANY KIND; REGARDLESS OF THE FORM OF ACTION, WHETHER BASED ON CONTRACT; TORT; NEGLIGENCE OF SMSC OR OTHERS; STRICT LIABILITY; BREACH OF WARRANTY; OR OTHERWISE; WHETHER OR NOT ANY REMEDY OF BUYER IS HELD TO HAVE FAILED OF ITS ESSENTIAL PURPOSE, AND WHETHER OR NOT SMSC HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. |

Table of Contents

[1 Introduction 5](#_Toc472591171)

[1.1 Purpose 5](#_Toc472591172)

[1.2 Scope 5](#_Toc472591173)

[1.3 References 5](#_Toc472591174)

[1.4 Glossary of Terms and Acronyms 5](#_Toc472591175)

[2 Overall Description 6](#_Toc472591176)

[2.1 Product Perspective 6](#_Toc472591177)

[2.2 Product Functions 6](#_Toc472591178)

[2.3 User Classes and Characteristics 6](#_Toc472591179)

[2.4 Design and Implementation Constraints 6](#_Toc472591180)

[2.5 Assumptions and Dependencies 6](#_Toc472591181)

[3 External Interface Requirements 7](#_Toc472591182)

[3.1 User Interfaces 7](#_Toc472591183)

[3.2 Hardware Interfaces 7](#_Toc472591184)

[3.3 Software Interfaces 7](#_Toc472591185)

[4 Usage 8](#_Toc472591186)

[4.1 QMSPI Functions 8](#_Toc472591187)

[4.1.1 spi\_port\_sel 8](#_Toc472591188)

[4.1.2 spi\_port\_drv\_slew 8](#_Toc472591189)

[4.1.3 rom\_dis\_lock\_shd\_spi 9](#_Toc472591190)

[4.1.4 qmpsi\_init 9](#_Toc472591191)

[4.1.5 qmspi\_freq\_get 10](#_Toc472591192)

[4.1.6 qmspi\_freq\_set 10](#_Toc472591193)

[4.1.7 qmspi\_xfr\_done\_status 10](#_Toc472591194)

[4.1.8 qmspi\_start 11](#_Toc472591195)

[4.1.9 qmspi\_start\_dma 11](#_Toc472591196)

[4.1.10 qmspi\_cfg\_spi\_cmd 12](#_Toc472591197)

[4.1.11 qmspi\_read\_dma 13](#_Toc472591198)

[4.1.12 qmspi\_read\_fifo 13](#_Toc472591199)

[4.2 AES Functions 14](#_Toc472591200)

[4.2.1 aes\_hash\_power 14](#_Toc472591201)

[4.2.2 aes\_hash\_reset 14](#_Toc472591202)

[4.2.3 aes\_busy 15](#_Toc472591203)

[4.2.4 aes\_status 15](#_Toc472591204)

[4.2.5 aes\_done\_status 15](#_Toc472591205)

[4.2.6 aes\_stop 16](#_Toc472591206)

[4.2.7 aes\_start 16](#_Toc472591207)

[4.2.8 aes\_iclr 16](#_Toc472591208)

[4.2.9 aes\_set\_key 16](#_Toc472591209)

[4.2.10 aes\_crypt 17](#_Toc472591210)

[4.3 RSA Functions 18](#_Toc472591211)

[4.3.1 rsa\_load\_key 19](#_Toc472591212)

[4.3.2 rsa\_keygen 20](#_Toc472591213)

[4.3.3 rsa\_modular\_exp 21](#_Toc472591214)

[4.3.4 rsa\_encrypt 21](#_Toc472591215)

[4.3.5 rsa\_decrypt 22](#_Toc472591216)

[4.3.6 rsa\_crt\_gen\_params 22](#_Toc472591217)

[4.3.7 rsa\_load\_crt\_params 24](#_Toc472591218)

[4.3.8 rsa\_crt\_decrypt 24](#_Toc472591219)

[4.3.9 rsa\_signature\_gen 25](#_Toc472591220)

[4.3.10 rsa\_signature\_verify 26](#_Toc472591221)

[4.4 PKE Functions 27](#_Toc472591222)

[4.4.1 pke\_power 27](#_Toc472591223)

[4.4.2 pke\_reset 27](#_Toc472591224)

[4.4.3 pke\_status 27](#_Toc472591225)

[4.4.4 pke\_done\_status 28](#_Toc472591226)

[4.4.5 pke\_start 28](#_Toc472591227)

[4.4.6 pke\_busy 29](#_Toc472591228)

[4.4.7 pke\_set\_operand\_slot 29](#_Toc472591229)

[4.4.8 pke\_get\_operand\_slot 29](#_Toc472591230)

[4.4.9 pke\_set\_operand\_slots 30](#_Toc472591231)

[4.4.10 pke\_get\_slot\_addr 30](#_Toc472591232)

[4.4.11 pke\_fill\_slot 30](#_Toc472591233)

[4.4.12 pke\_scm\_clear\_slot 31](#_Toc472591234)

[4.4.13 pke\_read\_scm 31](#_Toc472591235)

[4.4.14 pke\_write\_scm32 32](#_Toc472591236)

[4.4.15 pke\_write\_scm 32](#_Toc472591237)

[4.4.16 pke\_clear\_scm 32](#_Toc472591238)

[4.4.17 pke\_ists\_clear 33](#_Toc472591239)

[4.4.18 modular\_arithm 33](#_Toc472591240)

[4.5 Elliptic Curve Functions 34](#_Toc472591241)

[4.5.1 ec\_point\_double 34](#_Toc472591242)

[4.5.2 ec\_point\_add 35](#_Toc472591243)

[4.5.3 ec\_point\_scalar\_mult2 35](#_Toc472591244)

[4.5.4 ec\_point\_scalar\_mult3 36](#_Toc472591245)

[4.5.5 ec\_check\_poc2 36](#_Toc472591246)

[4.5.6 ec\_check\_poc3 37](#_Toc472591247)

[4.5.7 ec\_check\_point\_less\_prime 37](#_Toc472591248)

[4.5.8 ec\_check\_ab 38](#_Toc472591249)

[4.5.9 ec\_check\_n 38](#_Toc472591250)

[4.5.10 ec\_prog\_curve 39](#_Toc472591251)

[4.5.11 ecdsa\_verify 40](#_Toc472591252)

[4.5.12 ec\_kcdsa\_keygen 41](#_Toc472591253)

[4.5.13 ec\_kcdsa\_sign 42](#_Toc472591254)

[4.5.14 ec\_kcdsa\_verify 42](#_Toc472591255)

[4.5.15 src\_sc 43](#_Toc472591256)

[4.5.16 ec25519\_point\_mult 44](#_Toc472591257)

[4.5.17 ec25519\_xrecover 44](#_Toc472591258)

[4.5.18 ed25519\_scalar\_mult 45](#_Toc472591259)

[4.5.19 ed25519\_valid\_sig 45](#_Toc472591260)

[4.6 RNG Functions 47](#_Toc472591261)

[4.6.1 rng\_power 47](#_Toc472591262)

[4.6.2 rng\_reset: 47](#_Toc472591263)

[4.6.3 rng\_mode 47](#_Toc472591264)

[4.6.4 rng\_is\_on 48](#_Toc472591265)

[4.6.5 rng\_start 48](#_Toc472591266)

[4.6.6 rng\_stop 48](#_Toc472591267)

[4.6.7 rng\_get\_fifo\_level 49](#_Toc472591268)

[4.6.8 rng\_get\_bytes 49](#_Toc472591269)

[4.6.9 rng\_get\_words 49](#_Toc472591270)

[4.7 HASH Functions 50](#_Toc472591271)

[4.7.1 hash\_status 50](#_Toc472591272)

[4.7.2 hash\_busy 50](#_Toc472591273)

[4.7.3 hash\_start 51](#_Toc472591274)

[4.7.4 hash\_done\_status 51](#_Toc472591275)

[4.8 SHA Functions 51](#_Toc472591276)

[4.8.1 sha12\_init 51](#_Toc472591277)

[4.8.2 sha12\_update 53](#_Toc472591278)

[4.8.3 sha12\_finalize 53](#_Toc472591279)

[4.8.4 sha35\_init 54](#_Toc472591280)

[4.8.5 sha35\_update 55](#_Toc472591281)

[4.8.6 sha35\_finalize 56](#_Toc472591282)

[4.8.7 hash\_iclr 56](#_Toc472591283)

[4.8.8 sha\_init 57](#_Toc472591284)

[4.8.9 sha\_update 57](#_Toc472591285)

[4.8.10 sha\_final 58](#_Toc472591286)

[4.9 Miscellanous ROM API 58](#_Toc472591287)

[4.9.1 version 58](#_Toc472591288)

[4.9.2 loader 59](#_Toc472591289)

[5 Appendix: 60](#_Toc472591290)

[6 Revision History 62](#_Toc472591291)

# Introduction

## Purpose

The document illustrates the usage of ROM API’s available in MEC 2016

## Scope

This document will serve as a usage manual for the functions provided by the function in the ROM. It presents the reader with the function header, a description of the functions operations, it inputs and output parameters.

## References

-

## 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

# Overall Description

## Product Perspective

The Boot Rom API allows software access to certain hardware features that facilitates easy development of application on the MEC2016 platform. These API’s serve the function of providing easy access to the underlying hardware features.

## Product Functions

The Boot Rom API’s 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 API’s to leverage these device specific operations.

## User Classes and Characteristics

This document is intended for programmers and users of the MEC2016 product. It illustrates the use of certain software features that would facilitate easy development of applications.

## Design and Implementation Constraints

The API’s 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 API’s do not use heap dynamic or global space to store data.

## Assumptions and Dependencies

The efficacy of this user manual may be contingent upon the knowledge of the MEC2016 target hardware. Certain references in this document may require the usage of device data sheets.

If testing with FPGA proper bootrom elf file is needed to be loaded suing incremental option

# External Interface Requirements

## User Interfaces

The document will describe the user interface to the rom resident API’s.

## Hardware Interfaces

MEC2016 EVB / FPGA with proper bit map

Keil µVision Ulink Pro Debugger tool –

MCHP Trace debugger Tool

Dediprog SPI programmer

## 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

-

# Usage

**Note1:** All blocks need to be powered ON with corresponding APIs before usage of any of the API for crypto operations.

***Note2:***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).

## QMSPI Functions

### 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

### spi\_port\_drv\_slew

Function Header:

void spi\_port\_drv\_slew(uint8\_t port, uint8\_t pin\_mask , uint8\_t drv\_slew);

Description:

This function configures the drive strength and slew rate for GPIO’s based on selected port.

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   * ROM\_PORT\_QMSPI\_SHD * ROM\_PORT\_QMSPI\_PVT * ROM\_PORT\_QMSPI\_INT |
| 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 - GPIO\_DRV\_STR\_2MA, GPIO\_DRV\_STR\_4MA, GPIO\_DRV\_STR\_8MA, GPIO\_DRV\_STR\_12MA. Slew Rate - GPIO\_DRV\_SLEW\_SLOW, GPIO\_DRV\_SLEW\_FAST. The parameter drv\_slew corresponds to a hardware register. Please refer the User Manual of target device for description. |

Outputs: None

### 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

### 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 | (0x00u << 8) |
| SPI\_MODE\_1 | (0x06u << 8) |
| SPI\_MODE\_2 | (0x01u << 8) |
| SPI\_MODE\_3 | (0x07u << 8) |

Outputs: None

### qmspi\_freq\_get

Function Header:

uint32\_t qmspi\_freq\_get(void);

Description:

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

Inputs: None

Outputs:

Returns the SPI operating frequency.

### 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

### 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 |
| 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.

### 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

### 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

### 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
   1. Clocks = bytes \* clocks/byte. Clocks per byte depend upon data bus width.
   2. Data bus width – 1X -> 8clocks/byte, 2X -> 4 clocks/byte, 4X -> 2 clocks/byte.
   3. 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 Parameter | 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).

### 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 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 qmspi\_start\_dma section for description regarding dmach\_id. |

Outputs:

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

### 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.

## AES Functions

### 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 also 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

### 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

### 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.

### 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). |

### 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 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 is done status is set, FALSE otherwise.

### 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.

### 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 to 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.

### 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 aes\_status for bit definitions.

### 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 mode 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\_ERR\_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 |

### 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 operation using 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 MEC140X 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 |

## RSA Functions

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

### 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 MEC2016 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

Inputs:

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. |

Outputs:

The return values and their description is 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 |

### 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.

### 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.

### 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.

### 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 = (emcrypted\_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.

### 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. |
| 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.

### 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 is 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.

### 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.

### 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.

### 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.

## PKE Functions

### 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

### pke\_reset

Function Header:

Void pke\_reset(void);

Description:

This routine resets the pke block.

Inputs:

None

Output:

None

### 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) |
| 10 | This bit applies to Elliptic curve, 0 if parameters a,b are valid, 1 otherwise. |
| 9 | This bit only applies to signature verify operations. 0 if signature is valid, 1 otherwise. |
| 7 | This bit applies to Elliptic curve, 0 if parameter n is valid, 1 otherwise. |
| 6 | This bit applies to Elliptic curve, 0 if couple x,y 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. |

### 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 pke\_status for bit definitions. |

Outputs:

Returns True if done status is set, False Otherwise.

### 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

### 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.

### 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

### 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

### 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

### 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

### 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

### 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

### 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 to |
| 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.

### 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

### 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 as bytes. 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

### 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.

### 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],

### 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 arithmatic

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 |

## Elliptic Curve Functions

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

### 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 ec curve |
| coord\_len | length filed 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 |

### 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 filed 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

### 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

### 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

### 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 Parameter | 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

### 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

### 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

### ec\_check\_ab

Function Header:

uint8\_t ec\_check\_ab(void);

Description:

This routine check EC Curve parameters a & b. Requires curve has been programmed into PKE via

ec\_prog\_curve()

Inputs:

None

Outputs:

The return values and their description are presented below.

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

### ec\_check\_n

Function Header:

uint8\_t ec\_check\_n(void);

Description:

This routine check EC Curve order (parameter n) Requires curve has been programmed into PKE via

ec\_prog\_curve().

Inputs:

None

Outputs:

The return values and their description are presented below.

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

### 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)  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 lenght 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 is 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 |

### 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 |

### 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)

### 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)

### ec\_kcdsa\_verify

Function header:

uint8\_t ec\_kcdsa\_verify(const uint8\_t\* q,

std::uint16\_t qlen,

const uint8\_t\* sig,

std::uint16\_t slen,

const uint8\_t\* hash,

std::uint16\_t hlen,

std::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 | Pointer to array containing Qx and Qy |
| qlen | Byte length of Q |
| Sig | Pointer to array containing r & s |
| slen | Byte length of signature |
| hash | Pointer to hash digest of message |
| hlen | Byte length of hash digest |
| Msbf | bit[0]=0(Qx,y are LSBF), 1(Qx,y are MSBF)  bit[1]=0(r,s are LSBF), 1(r,s are 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)

### 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

### 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 curve25519 Point Multiplication

Inputs:

|  |  |
| --- | --- |
| Input Parameters | Description |
| P1x | Pointer to parameter px |
| P1x\_len | Byte length of p1\_x |
| K | Pointer to parameter K |
| K\_len | Byte 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\_ERR\_BAD\_PARAM if

parameters have invalid values

### ec25519\_xrecover

Function header:

uint8\_t ec25519\_xrecover(const uint8\_t\* y, uint16\_t ylen, uint16\_t flags);

Description:

This routine recover 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)

### 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. Part of Ed25519. 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)

### 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\_ERR\_BAD\_PARAM if

parameters have invalid values

## RNG Functions

### 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 (gate off clocks to block), if true enables the block (gate on clocks to block) |

Outputs:

None

### rng\_reset:

Function Header:

Void rng\_reset(void);

Description:

This routine resets the RNG block.

Inputs:

None

Outputs:

None

### 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

### 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.

### 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

### 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

### 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).

### 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.

### 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. If the Random number generator is stopped or no data available in the FIFO it’ll return a zero.

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.

## HASH Functions

Note: Before using hash functions, the hash block needs to be powered ON using aes\_hash\_power() API.

### 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. |

### 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.

### 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

### 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.

## SHA Functions

Note: Before using sha functions, the hash block needs to be powered ON using aes\_hash\_power() API.

### 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 is sha\_mode 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 |

### 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.

### 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 nay 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 is 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.

### 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 is sha35\_ctx 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.

### 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.

### 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.

### hash\_iclr

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).

### 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 and SHA512). Hash engine does not need to be started. SHA1 and SHA256 require 4 byte alignment. 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.

### sha\_update

Function Header:

uint8\_t sha\_update( const 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. SHA512 require 8-byte alignment. Has engine is not started, 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 is presented below.

0 = Success

1 = Hash engine busy

2 = pdata is a null pointer

3 = data is misaligned

### 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 implement 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

## Miscellanous ROM API

### 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.

### 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

# Appendix:

The symdef table for the A1 ROM API’s

#<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

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

Symdef continued…

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

# Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Revision Level | Date | Section | Remarks |
| Hemal Gujarathi – I15581 | 1.0 | 21 Apr 2016 | 4.3 | Created a new copy to include changes for Bootrom A1 version |
| Akshaya Karthikeyan - I17306 | 1.1 | 13 May 2016 | 4.3 | Added API for rsa crt decryption |
| Hemal Gujarathi – I15581 | 2.0 | 20 July 2016 |  | Add new APIs |
| Arun Krishnan C21798 | 2.1 | 20 July 2016 | 4.5.1, 4.5.2, 4.5.3, 4.5.4, 4.5.6, 4.5.7, 4.5.18 | Updated the following section for corrections  Added Appendix for the symdef table used |
| Arun Krishnan C21798 | 2.2 | 22 July 2016 | 4.4.14 | Added section 4.1.14 pke\_write\_scm32  Updated the Appendix for the symdef table used |
| Arun Krishnan C21798 | 2.3 | 22 July 2016 | 4.4.16 | Updated the comment for the correct bit filed details |
| Akshaya Karthikeyan - I17306 | 2.4 | 17 August 2016 | 4.1,4.3,4.5,4.2.10 | Added mode values for aes\_crypt()  Added note for block power enable, icct timer enable. |
| Akshaya Karthikeyan - I17306 | 2.5 | 06 September 2016 | 4.3,4.5,4.8 | Added return code values for ECC, SHA block APIs. |
| Akshaya Karthikeyan - I17306 | 2.6 | 09 September 2016 | 4.9 | Added section for miscellaneous ROM API - loader |
| Swastik Pramanik – I16169 | 2.7 | 22 November 2016 | Section 4 | Corrected API definitions as per BootROM A1. Also added the missing API definitions. |
| Swastik Pramanik – I16169 | 2.8 | 16 December 2016 | Section 4.1, 4.6, 5 | Removed qmspi\_cfg\_wrtie\_cmd(), qmspi\_write\_dma() APIs and corrected the rng\_power() API’s description. |
| Swastik Pramanik – I16169 | 2.9 | 19 January 2017 | Section 4.1 | Removed qmspi\_xmit\_cmd() and added back qmspi\_cfg\_wrtie\_cmd().. |
| Swastik Pramanik | 2.10 | 6 April 2017 | Section 4.6.9, Section 4.8.8, 4.8.9 | Updated the rng\_get\_word() description, removed the instances of SHA384 |