Microsemi VPROC API

V1.0.0

Generated by Doxygen 1.8.6

Fri Jul 1 2016 13:58:34

Contents

Chapter 1

Main Page

1.1 Introduction

This is an API specification document for Microsemi Voice Processor Class of Device SDK. This document covers platform-independent Host Bus Interface(HBI) And platform-dependent System Service Layer(SSL). HBI API are implemented over SSL. Below Diagram gives an overview of different components in system

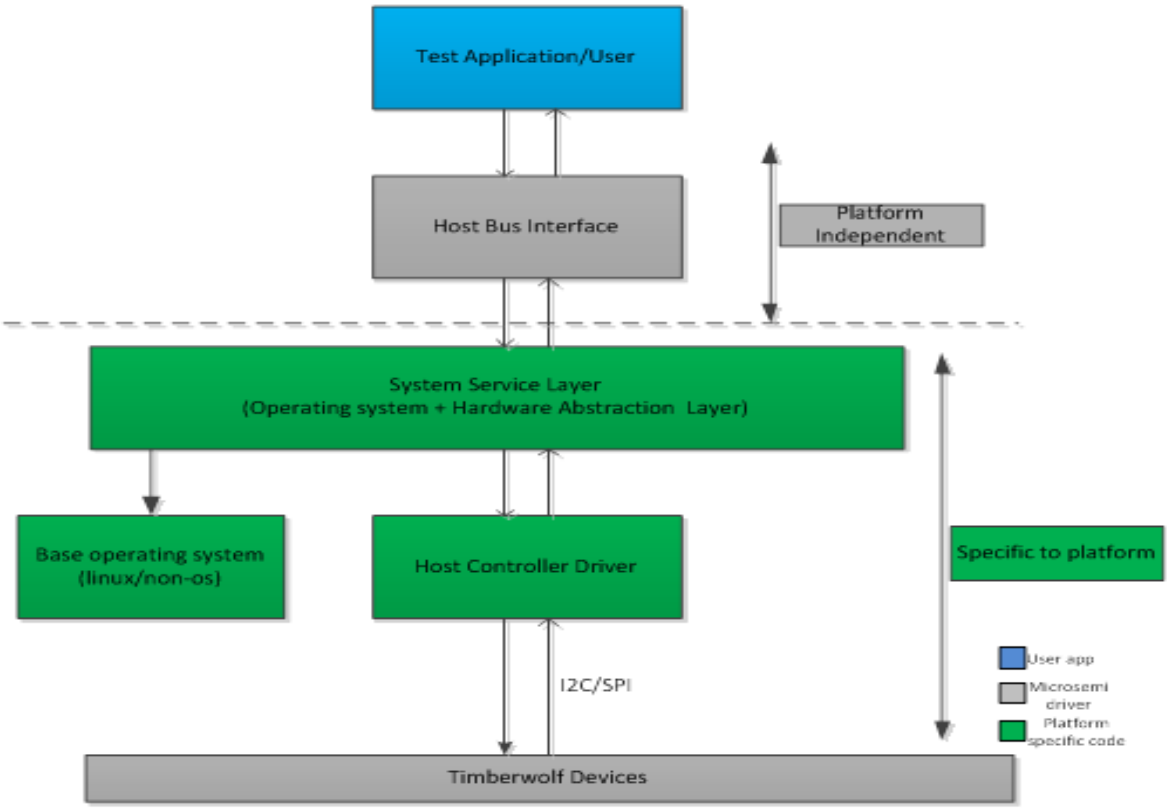

Figure 1.1: Microsemi VPROC SDK FLOW

Microsemi only define System Service Layer as an abstract API set . It should be ported by developer responsible for porting HBI Driver for a particular host platform. HBI Driver and Microsemi VPROC SDK is assisted by various

compile-time options which gives flexibility to host to configure Microsemi Voice Processor Device and software as per their system requirement ex. SPI/I2C, endianness, memory resources. Below tables summarizes both HBI and System-wide various compile time options

Option	Value	Description
BOOT_FROM_HOST	yes no	If set to yes or defined, then HBI driver will support booting voice processor device with specific firmware image from host
HBI_MAX_INST_PER_DEV	decimal value as per system	This defines maximum number of simultaneous user on a single voice processor device
FLASH_PRESENT	yes no	If defined, indicates the Microsemi Voice Processor device is connected to slave flash. In this case HBI driver will support all flash related operations
VPROC_DEV_ENDIAN	big	This defines endian-ness of VPROC Device as big.
HBI_MAX_INSTANCES	Any decimal value starting from 1	Maximum number of users of driver
HBI_BUFFER_SIZE	Any decimal value > 0	Size of buffer used by HBI for any transactions
HBI_ENABLE_PROCFS	0,1	Optional. Applicable for linux based system

Figure 1.2: Microsemi HBI Compile Time Options

Option	Value	Description
VPROC_ MAX_NUM_DEVS	Any decimal value	This defines maximum number of voice processor devices in a system
TARGET	TW	This defines voice processor device i.e. value will “tw” for Timberwolf
HBI	I2C, SPI	This defines physical bus in use for host control data communication to device. Value can be SPI or I2C
BUILD_TYPE	DEBUG	Release or Debug. If release, all Print functions would be faked or none. If debug, then print functions would be enabled. User can further select a debug level.
HOST_ENDIAN	big, little	Tell endianness of host platform
VPROC_DEV_NAME_SIZE	Any decimal number	Optional. Tells device name maximum size limit
NUM_MAX_LOCKS	Any decimal number	Optional. Indicates maximum locks that can be used by driver.
DEBUG_LEVEL	Bitmask value of VPROC_DBG_LVL 0x0 - None 0x1 – function entry/exit 0x2 - information 0x4 - warning 0x8 - Err 0xF - All	

Figure 1.3: Microsemi VPROC SDK Compile Time Options

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Module Documentation

5.1 Host Bus Interface

Host Bus Interface define control path communication to Microsemi VPROC Devices.

- enum hbi_cmd_t {
  HBI_CMD_LOAD_FWR_FROM_HOST, HBI_CMD_LOAD_CFGREC_FROM_HOST, HBI_CMD_LOAD_F-
  WR_COMPLETE, HBI_CMD_LOAD_FWRCFG_FROM_FLASH,
  HBI_CMD_SAVE_FWRCFG_TO_FLASH, HBI_CMD_ERASE_WHOLE_FLASH, HBI_CMD_ERASE_FWR-
  CFG_FROM_FLASH, HBI_CMD_START_FWR,
  HBI_CMD_END }

  Enumerates various HOST commands that can be issued to device.
- enum hbi_status_t {
  HBI_STATUS_NOT_INIT, HBI_STATUS_INTERNAL_ERR, HBI_STATUS_RESOURCE_ERR, HBI_STAT-
  US_INVALID_ARG,
  HBI_STATUS_BAD_HANDLE, HBI_STATUS_BAD_IMAGE, HBI_STATUS_FLASH_FULL, HBI_STATUS_-
  NO_FLASH_PRESENT,
  HBI_STATUS_COMMAND_ERR, HBI_STATUS_INCOMPAT_APP, HBI_STATUS_INVALID_STATE, HBI_-
  STATUS_OP_INCOMPLETE,
  HBI_STATUS_SUCCESS }

  enumerates various status codes of HBI Driver
- enum hbi_img_type_t { HBI_IMG_TYPE_FWR =0, HBI_IMG_TYPE_CR =1, HBI_IMG_TYPE_LAST }

  enumerates types of data as passed by application
- enum hbi_rst_mode_t { HBI_RST_POR }

  enumerates reset mode of device
- typedef uint32_t hbi_handle_t

  HBI Handle updated by driver during HBI_Open() call and used in further driver calls.
- typedef unsigned char user_buffer_t

  basic data type for buffer as should be used by user in call to HBI_read() ,HBI_write()
- hbi_status_t **HBI_init** (hbi_init_cfg_t *pCfg)
- hbi_status_t HBI_term (void)

  Driver termination function.
- hbi_status_t HBI_open (hbi_handle_t *pHandle, hbi_dev_cfg_t *pDevCfg)

  function to open a device.
- hbi_status_t HBI_close (hbi_handle_t handle)

  function to close a device opened using HBI_open().
- hbi_status_t HBI_reset (hbi_handle_t handle, hbi_rst_mode_t mode)

  function to reset a device.
- hbi_status_t HBI_read (hbi_handle_t handle, reg_addr_t reg, user_buffer_t *pData, size_t length)

Reads the data from device memory starting from register address up to specified length.

- hbi_status_t HBI_write (hbi_handle_t handle, reg_addr_t reg, user_buffer_t *pData, size_t length)
  Writes the data from user buffer to device memory starting from register address up to specified length.
- hbi_status_t HBI_set_command (hbi_handle_t handle, hbi_cmd_t cmd, void *pCmdArgs)
  Set the Host Commands as listed in hbi_cmd_t enum.
- hbi_status_t HBI_wake (hbi_handle_t handle)
  wakes up the device from sleep mode.
- hbi_status_t HBI_sleep (hbi_handle_t handle)
  This function puts device to sleep mode.
- hbi_status_t HBI_get_header (hbi_data_t *pImg, hbi_img_hdr_t *pHeaderInfo)
  This function parses header of the image passed by user.

5.1.1 Detailed Description

Microsemi VPROC device control path communcation is divided in to transport and physical layer where Transport Layer sits on top of Physical Layer. Physical layer define actual bus protocol to use at ground level.Current Microsemi VPROC Devices support SPI and I2C as physical interface. Transport layer define a device specific protocol and termed as Host Bus Interface (HBI). Following section covers API of HBI Driver to configure VPROC Devices.

5.1.2 Enumeration Type Documentation

5.1.2.1 enum hbi_cmd_t

HBI commands accepts either of these types of arguments: mandatory: means this command need an input to process optional : inputs are desirable but not mandatory. none : this command accepts no arguments. Details of argument w.r.t command is detailed as below

Enumerator

HBI_CMD_LOAD_FWR_FROM_HOST Loads firmware image to device. [in] input args: pointer to hbi_data_buf_t (populated with pointer to image buffer and length of buffer) [out] output args: none

HBI_CMD_LOAD_CFGREC_FROM_HOST Loads configuration record from host to device. [in] input args: pointer to hbi_data_buf_t (populated with pointer to config record buffer and length of buffer) [out] output args: none

HBI_CMD_LOAD_FWR_COMPLETE Signal device that host data transfer is complete. Should be called after HBI_CMD_LOAD_FWR_FROM_HOST or HBI_CMD_LOAD_CFGREC_FROM_HOST input args : none output args : none

HBI_CMD_LOAD_FWRCFG_FROM_FLASH Loads requested firmware and associated configuration record from flash connected to voice processor device. Need image number from user. input args: an unsigend int image_num output args: none Saves current firmware and configuration record in device memory to flash. connected to voice processor device.Should be called after HBI_CMD_LOAD_FWR_FROM_HOST and HBI_CMD_HOST_LOAD_COMPLETE. input args : none output args: pointer to an unsigned int to pass back uploaded image number (optional)

HBI_CMD_ERASE_WHOLE_FLASH Erases whole flash connected to voice processor device input args: none output args: none.

HBI_CMD_ERASE_FWRCFG_FROM_FLASH Erases specific firmware and associated configuration record from flash connected to voice processor device. Need an image number from user Input args: an unsigned int firmware image number output args: none

HBI_CMD_START_FWR Starts executing current firmware in RAM after it is being loaded either from flash or host. Please note this should be called ONLY and IMMEDIATLY after HBI_CMD_LOAD_FROM_HOST and HBI_CMD_HOST_LOAD_COMPLETE OR HBI_CMD_LOAD_FWRCFG_FROM_FLASH. input args: none output args: none

HBI_CMD_END End marker for Host Command List.

Definition at line 74 of file hbi.h.

5.1.2.2 enum hbi_status_t

Enumerator

HBI_STATUS_NOT_INIT driver not initilised

HBI_STATUS_INTERNAL_ERR platform specific layer reported an error

HBI_STATUS_RESOURCE_ERR request resource unavailable

HBI_STATUS_INVALID_ARG invalid argument passed to a function call

HBI_STATUS_BAD_HANDLE a bad reference handle passed

HBI_STATUS_BAD_IMAGE requested firmware image not present on flash

HBI_STATUS_FLASH_FULL no more space left on flash

HBI_STATUS_NO_FLASH_PRESENT no flash connected to device

HBI_STATUS_COMMAND_ERR HBI Command failed.

HBI_STATUS_INCOMPAT_APP firmware image is incompatible

HBI_STATUS_INVALID_STATE driver is in invalid state for current action

HBI_STATUS_OP_INCOMPLETE operation incomplete

HBI_STATUS_SUCCESS driver call successful

Definition at line 142 of file hbi.h.

5.1.2.3 enum hbi_img_type_t

Enumerator

HBI_IMG_TYPE_FWR firmware image to load on Device

HBI_IMG_TYPE_CR Configuration Record to load Device.

HBI_IMG_TYPE_LAST Limiter on input type.

Definition at line 162 of file hbi.h.

5.1.2.4 enum hbi_rst_mode_t

Enumerator

HBI_RST_POR This simulates Power On Reset where it will clear everything including DSP memory and restart device with invocation of internal boot ROM code followed by loading of firmware from flash, if present. If flash not present in system then device will stop at Boot ROM prompt waiting on host

Definition at line 172 of file hbi.h.

5.1.3 Function Documentation

5.1.3.1 hbi_status_t HBI_term (void)

Deallocates and deinitializes all resources as acquired in HBI_init() function. Make sure to close all of the devices and users before terminating the driver else it will return an error. function is not interrupt safe and should be called from process context.

5.1.3.2 hbi_status_t HBI_open (hbi_handle_t * *pHandle*, hbi_dev_cfg_t * *pDevCfg*)

Inputs one-time device configuration as passed by user. User can call this function multiple time for different or same device. if called on already open device, driver will pass back already opened device reference with existing configuration and that user passed configuration will be ignored. This function is not interrupt safe.

Parameters

in	*pHandle*	- Updated by driver after successful open call. Reference handle of device to be used by user in subsequent HBI driver device access call.
in	*pDevCfg*	- Pointer to device configuration.
out	*pHandle*	- Device handle

5.1.3.3 hbi_status_t HBI_close (hbi_handle_t *handle*)

Deallocate and free up resources acquired during open call. If there are multiple user on the device, device physically doesn't close until user count reaches to 0 otherwise it just free up user reference and access to device. This function is not interrupt safe.

Parameters

in	*handle*	- Device handle as passed by HBI_open()

Return values

HBI_STATUS_SUCCESS	
HBI_STATUS_BAD_HANDLE	
HBI_STATUS_NOT_INIT	

5.1.3.4 hbi_status_t HBI_reset (hbi_handle_t *handle*, hbi_rst_mode_t *mode*)

This function equivalent to Power On Reset.This function is not interrupt safe.

Parameters

in	*handle*	- Device handle as returned by HBI_open()
in	*mode*	- Reset mode of device

Return values

HBI_STATUS_SUCCESS	
HBI_STATUS_BAD_HANDLE	
HBI_STATUS_NOT_INIT	
HBI_STATUS_INTERNAL_ERR	

5.1.3.5 hbi_status_t HBI_read (hbi_handle_t *handle*, reg_addr_t *reg*, user_buffer_t * *pData*, size_t *length*)

Reads the data from device memory starting from register address up to specified length. It is caller responsibility to ensure that user buffer is sufficiently large enough to hold requested length of data else it may result in system crash. driver may expect length in multiple of 16-bit depending upon device family it is compiled for. This function is not interrupt safe.

Parameters

in	*handle*	- Device handle as returned by HBI_open()
in	*reg*	- Device register address to read from

in	*pData*	- Pointer to user buffer to read data into
in	*length*	- length of the data to be read in bytes. should be passed based on number_of_elements_to_be_read∗sizeof(user_buffer_t)
out	*pData*	- updated by driver on success

Return values

HBI_STATUS_SUCCESS	
HBI_STATUS_BAD_HANDLE	
HBI_STATUS_NOT_INIT	
HBI_STATUS_INTERNAL_ERR	

5.1.3.6 hbi_status_t HBI_write (hbi_handle_t *handle*, reg_addr_t *reg*, user_buffer_t ∗ *pData*, size_t *length*)

Writes the data from user buffer to device memory starting from register address up to specified length. It is caller responsibility to ensure that user buffer is sufficiently allocated and filled to number of elements to be written. Else it may result in system crash. driver may expect length in multiple of 16-bit depending upon device family it is compiled for. This function is not interrupt safe.

Parameters

in	*handle*	- Device handle as returned by HBI_open()
in	*reg*	- Device register address to write to
in	*pData*	- Pointer to user buffer to read data from
in	*length*	- length of the data to be written in bytes. should be calculated based on sizeof(user_buffer_t)∗number_of_elements_to_fill in buffer. lets say, user allocate a user_buffer_t reg[2] and what to read 2 entries of the buffer then length length = 2∗sizeof(user_buffer_t)

Return values

HBI_STATUS_SUCCESS	
HBI_STATUS_BAD_HANDLE	
HBI_STATUS_NOT_INIT	
HBI_STATUS_INTERNAL_ERR	

5.1.3.7 hbi_status_t HBI_set_command (hbi_handle_t *handle*, hbi_cmd_t *cmd*, void ∗ *pCmdArgs*)

Set the Host Commands as listed in hbi_cmd_t enum.This function is not interrupt safe.

Parameters

in	*handle*	- Device handle as returned by HBI_open()
in	*cmd*	- Host Command as listed in hbi_cmd_t
in	*pCmdArgs*	- Pointer to respective command arguments
out	*pCmdArgs*	- May be updated by driver depending upon command issued.

Return values

HBI_STATUS_SUCCESS	

HBI_STATUS_BAD_HAND-LE	
HBI_STATUS_NOT_INIT	
HBI_STATUS_INTERNAL_-ERR	
HBI_STATUS_INVALID_A-RG	

5.1.3.8 hbi_status_t HBI_wake (hbi_handle_t *handle*)

Wakes up the device from sleep mode.This function is not interrupt safe.

Parameters

in	*handle*	- Device handle as returned by HBI_open()

Return values

HBI_STATUS_SUCCESS	
HBI_STATUS_BAD_HAND-LE	
HBI_STATUS_NOT_INIT	
HBI_STATUS_INTERNAL_-ERR	

5.1.3.9 hbi_status_t HBI_sleep (hbi_handle_t *handle*)

Puts device to sleep mode.This function is not interrupt safe.

Parameters

in	*handle*	- Device handle as returned by HBI_open()

Return values

HBI_STATUS_SUCCESS	
HBI_STATUS_BAD_HAND-LE	
HBI_STATUS_NOT_INIT	
HBI_STATUS_INTERNAL_-ERR	

5.1.3.10 hbi_status_t HBI_get_header (hbi_data_t * *pImg*, hbi_img_hdr_t * *pHeaderInfo*)

Parses Image Header. It is caller responsibility to ensure that user buffer is sufficiently allocated and filled to number of elements to be written. Else it may result in system crash. driver may expect length in multiple of 16-bit depending upon device family it is compiled for. This function is not interruptsafe.

Parameters

in	*pImg*	- pointer to a structure containing image buffer and its length
out	*pHeaderInfo*	- pointer updated by function with information extracted from header

Return values

HBI_STATUS_SUCCESS	

5.2 System Service Layer

System Service Layer is Platform specific Abstraction Layer.

- enum ssl_status_t {
  SSL_STATUS_NOT_INIT = -1, SSL_STATUS_INVALID_ARG = -2, SSL_STATUS_INTERNAL_ERR = -3,
  SSL_STATUS_BAD_HANDLE = -4,
  SSL_STATUS_RESOURCE_ERR = -5, SSL_STATUS_TIMEOUT = -6, SSL_STATUS_FAILED = -7, SSL_-STATUS_OP_INCOMPLETE = -8,
  SSL_STATUS_OK =0 }

  Enumerates Status Codes supported by SSL.
- enum ssl_wait_t { SSL_WAIT_NONE, SSL_WAIT_FOREVER}

  Enumerates wait type on lock.
- enum ssl_op_t { SSL_OP_PORT_RD =0x01, SSL_OP_PORT_WR =0x02, **SSL_OP_PORT_RW** =(SSL_O-P_PORT_RD | SSL_OP_PORT_WR) }

  Enumerates port functions.
- ssl_status_t SSL_init (ssl_drv_cfg_t *pCfg)

  Driver initialization function.
- ssl_status_t SSL_lock_create (ssl_lock_handle_t *pLock, const char *pName, void *pOption)

  Lock Create Function.
- ssl_status_t SSL_lock (ssl_lock_handle_t lock_id, ssl_wait_t wait_type)

  Hold a lock.
- ssl_status_t SSL_unlock (ssl_lock_handle_tlock_id)

  Release a lock.
- ssl_status_t SSL_lock_delete (ssl_lock_handle_tlock_id)

  Deletes lock.
- ssl_status_t SSL_term (void)

  Terminates SSL Driver.
- ssl_status_t SSL_port_open (ssl_port_handle_t *pHandle, ssl_dev_cfg_t *pDevCfg)

  Opens an port to device.
- ssl_status_t SSL_port_close (ssl_port_handle_t handle)

  Close the port opened to device.
- ssl_status_t SSL_port_read (ssl_port_handle_t handle, void *pDst, size_t *pNread)

  Reads the data from port into user buffer pointed by pDst.
- ssl_status_t SSL_port_write (ssl_port_handle_t handle, void *pSrc, size_t *pNwrite)

  Write the data from user buffer to device.
- ssl_status_t SSL_port_rw (ssl_port_handle_t handle, ssl_port_access_t *pPort)

  Read and Writes the data from and to device.
- ssl_status_t SSL_memset (void *pDst, int32_t val, size_t size)

  Sets the content of memory to specified value.
- ssl_status_t SSL_memcpy (void *pDst, const void *pSrc, size_t size)

  Copy the content of memory from source to destination pointer.
- ssl_status_t SSL_delay (uint32_t tmsec)

  Implements delay in millisecond.

5.2.1 Detailed Description

This layer prototypes API and data structures. It should be implemented by developer responsible for porting HBI driver over host platform.

5.2.2 Enumeration Type Documentation

5.2.2.1 enum ssl_status_t

Enumerator

SSL_STATUS_NOT_INIT SSL Driver not initialised. Should be returned by any other call to SSL API if called before SSL_init()

SSL_STATUS_INVALID_ARG Invalid argument passed to SSL API. Should be returned by every AAL API, if given argument doesn't fall in supported range

SSL_STATUS_INTERNAL_ERR Indicates to caller than an error is reported from platform specific layer.

SSL_STATUS_BAD_HANDLE Invalid port handle passed.Should be returned by an API if called before successful SSL_port_open()

SSL_STATUS_RESOURCE_ERR Requested resource unavailable. Should be returned by SSL API if requested resource not available. Example memory, bus, device

SSL_STATUS_TIMEOUT Return if API times out waiting on a certain operation.

SSL_STATUS_FAILED SSL API failed. A more generic message to indicate a call failure for any reason than listed above

SSL_STATUS_OP_INCOMPLETE port read/write incomplete. May be returned when number of bytes read/written is less than actual requested. This is optional and depends upon developer choice of SSL_read() and SSL_write() implementation

SSL_STATUS_OK SSL API call successful.

Definition at line 41 of file ssl.h.

5.2.2.2 enum ssl_wait_t

Depends on System Service Layer implementation and support of locking mechanism.

if Port layer does not support locking, these can be don't care or stub.

Enumerator

SSL_WAIT_NONE Return immediately, if failed to get lock.

SSL_WAIT_FOREVER Wait until lock is attained.Please note this may block the call.

Definition at line 89 of file ssl.h.

5.2.2.3 enum ssl_op_t

Can be 'Read Only' , 'Write Only' or 'Read/Write' both

Enumerator

SSL_OP_PORT_RD Read only operation on Port.

SSL_OP_PORT_WR Write only operation on Port.

Definition at line 102 of file ssl.h.

5.2.3 Function Documentation

5.2.3.1 ssl_status_t SSL_init (ssl_drv_cfg_t * *pCfg*)

Developer should Allocate and initialize resources as required by driver in this function call. pCfg is optional argument and host system specific. if required, developer may add initialization configuration paramter. Definition of data type ssl_drv_cfg_t is developer-specific and should be defined in typedefs.h in sdk. if used, it should be one-time configuration parameter and should not be modified.

Parameters

in	*pCfg*	Pointer to driver init configuration. This is optional parameter.

5.2.3.2 ssl_status_t SSL_lock_create (ssl_lock_handle_t * *pLock*, const char * *pName*, void * *pOption*)

This function implements Locking and Unlocking mechanism. pName and pOption are optional parameter and are implementation specific. ssl_lock_handle_t is host specific data type and should be declared by developer in typedefs.h

Parameters

in	*pLock*	Pointer to Lock structure. Should not be NULL.
in	*pName*	Optional Null terminated string to identify lock
in	*pOption*	Optional pointer to options structure as declared by host system in typedef.h.
out	*pLock*	Updated by call on successful completion

5.2.3.3 ssl_status_t SSL_lock (ssl_lock_handle_t *lock_id*, ssl_wait_t *wait_type*)

This function is called to hold a Locking. This can act as stub function if implementation does not support a locking mechanism.

Parameters

in	*lock_id*	handle as returned by call to SSL_lock_create()
in	*wait_type*	enum indicating WAIT_FOREVER or WAIT_NONE

5.2.3.4 ssl_status_t SSL_unlock (ssl_lock_handle_t *lock_id*)

This function is called to release a Lock. This can act as stub function if implementation doesnt support a locking mechanism.

Parameters

in	*lock_id*	handle as returned by call to SSL_lock_create()

5.2.3.5 ssl_status_t SSL_lock_delete (ssl_lock_handle_t *lock_id*)

This function is called to delete a Lock created by call to SSL_lock_create.

Parameters

in	*lock_id*	handle as returned by call to SSL_lock_create()

5.2.3.6 ssl_status_t SSL_term (void)

This function should deallocate and free all resources as acquired during SSL_init() call.

5.2.3.7 ssl_status_t SSL_port_open (ssl_port_handle_t * *pHandle*, ssl_dev_cfg_t * *pDevCfg*)

This function should initialise, allocate and setup all necessary infrastructure required to transmit data to and from device. Initialisation of SPI / I2C interface happen here. User should be able to do read/write to device after successful execution of this API

Parameters

in	*pHandle	Pointer to port handle. ssl_port_handle_t data type should be declared by developer in typedefs.h
in	*pDevCfg	Pointer to device configuration. User passed in bus number, chip address or id. Example for SPI, bus number can be spi bus number and device address can be chip select value. For i2c, device address can be allowable range of i2c addresses.
out	*pHandle*	Should be updated by driver on successful execution of call

5.2.3.8 ssl_status_t SSL_port_close (ssl_port_handle_t *handle*)

This function should deallocate and free all resources as acquired during SSL_port_open() call. After successful execution of this function, user should not be able to send/receive data to device.

Parameters

handle	Port Handle as returned by SSL_port_open()

5.2.3.9 ssl_status_t SSL_port_read (ssl_port_handle_t *handle*, void * *pDst*, size_t * *pNread*)

This function read data from device into user buffer pointed by pDst

Parameters

in	*handle*	Port Handle as returned by SSL_port_open()
in	*pDst*	Pointer to user passed in destination buffer to read data in to
in	*pNread*	Pointer to variable containing size of data to read
out	*pNread*	Optional.May be updated by driver to indicate size of data actually read

5.2.3.10 ssl_status_t SSL_port_write (ssl_port_handle_t *handle*, void * *pSrc*, size_t * *pNwrite*)

This function writes user passed data in pSrc buffer to device.

Parameters

in	*handle*	Port Handle as returned by SSL_port_open()
in	*pSrc*	Pointer to source buffer containing data to be sent to device
in	*pNwrite*	Pointer to buffer containing size of data to be written
out	*pNwrite*	Optional. May be updated by driver to indicate size of data actually written

5.2.3.11 ssl_status_t SSL_port_rw (ssl_port_handle_t *handle*, ssl_port_access_t * *pPort*)

This function performs both read/write transaction to device in a single call. May be supported by driver for combined transactions.

Parameters

in	*handle*	Port Handle as returned by SSL_port_open()
in	*pPort*	Pointer to structure populated with valid arguments for read/write transactions
out	*pPort*	May update size of data actually read/written in pPort structure

5.2.3.12 ssl_status_t SSL_memset (void * *pDst*, int32_t *val*, size_t *size*)

Sets the content of memory to specified value. May be ported over system specific memset call.

Parameters

in	*pDst*	Pointer to memory location to be updated
in	*val*	Value to be written to
in	*size*	size of the memory to be updated

5.2.3.13 ssl_status_t SSL_memcpy (void * *pDst*, const void * *pSrc*, size_t *size*)

Copies the content of source buffer to destination buffer. May be ported over system specific memcpy call.

Parameters

in	*pDst*	Pointer to memory location to be copied to
in	*pSrc*	Pointer to memory location to copy from
in	*size*	size of the data to be copied

5.2.3.14 ssl_status_t SSL_delay (uint32_t *tmsec*)

Implements delay in milliseconds . May be ported over system specific delay/sleep call.

Parameters

in	*tmsec*	- time in milliseconds

Chapter 6

Data Structure Documentation

6.1 hbi_data_t Struct Reference

describes buffer format to be used by user to pass any data to HBI Driver.

`#include <hbi.h>`

Data Fields

- unsigned char * pData
  pointer to user data buffer
- size_t size
  length of the buffer in bytes

6.1.1 Detailed Description

Definition at line 198 of file hbi.h.

The documentation for this struct was generated from the following file:

- drivers/hbi/inc/hbi.h

6.2 hbi_dev_cfg_t Struct Reference

user passed in device configuration parameters at the time of open.

`#include <hbi.h>`

Data Fields

- uint8_t dev_addr
  device address.
- uint8_t * pDevName
  an optional pointer to device name
- uint8_t bus_num
  bus number device physically present on
- ssl_lock_handle_t dev_lock
  lock to serialise device access

6.2.1 Detailed Description

Definition at line 186 of file hbi.h.

6.2.2 Field Documentation

6.2.2.1 uint8_t dev_addr

can be i2c complaint or spi chip select id

Definition at line 188 of file hbi.h.

The documentation for this struct was generated from the following file:

- drivers/hbi/inc/hbi.h

6.3 hbi_img_hdr_t Struct Reference

provide header information of firmware image passed by user

`#include <hbi.h>`

Data Fields

- int major_ver
  header version major num
- int minor_ver
  header version minor num
- hbi_img_type_t image_type
  firmware or configuration record
- int endianness
  endianness of the image excluding header
- int fwr_code
  if image_type == firmware, tells the firmware opn code
- size_t block_size
  block length in words image is divided into
- size_t img_len
  total length of the image excluding header
- int hdr_len
  length of header

6.3.1 Detailed Description

Definition at line 218 of file hbi.h.

The documentation for this struct was generated from the following file:

- drivers/hbi/inc/hbi.h

6.4 hbi_init_cfg_t Struct Reference

user-specific configuration passed at the time of driver initialization

`#include <hbi.h>`

Data Fields

- ssl_lock_handle_t lock
  Driver serialising lock.

6.4.1 Detailed Description

Definition at line 207 of file hbi.h.

6.4.2 Field Documentation

6.4.2.1 ssl_lock_handle_t lock

supposed to be created and passed by user. if passed, then driver will serialize all calls using this lock.

Definition at line 209 of file hbi.h.

The documentation for this struct was generated from the following file:

- drivers/hbi/inc/hbi.h

6.5 ssl_port_access_t Struct Reference

user passed in structure during read/write done by making a call to SSL_port_read(), SSL_port_write() and SSL_port_rw()

```
#include <ssl.h>
```

Data Fields

- void * pSrc
  Pointer to caller source buffer.
- void * pDst
  Pointer to caller destination buffer, Must be set to a valid value for PORT READ Operation.
- size_t nread
  Number of bytes to read.
- size_t nwrite
  Number of bytes to write.
- ssl_op_t op_type
  Enum indicating port operation: 'read','write' or 'read/write'.

6.5.1 Detailed Description

Definition at line 115 of file ssl.h.

6.5.2 Field Documentation

6.5.2.1 void* pSrc

Must be set to a valid value for PORT WRITE Operation

Definition at line 117 of file ssl.h.

6.5.2.2 size_t nread

Ignore in case of PORT Write operation

Definition at line 123 of file ssl.h.

6.5.2.3 size_t nwrite

Ignore in case of PORT Write operation

Definition at line 126 of file ssl.h.

The documentation for this struct was generated from the following file:

- include/ssl.h

Chapter 7

File Documentation

7.1 drivers/hbi/inc/hbi.h File Reference

Data Structures

- struct hbi_dev_cfg_t

  user passed in device configuration parameters at the time of open.
- struct hbi_data_t

  describes buffer format to be used by user to pass any data to HBI Driver.
- struct hbi_init_cfg_t

  user-specific configuration passed at the time of driver initialization
- struct hbi_img_hdr_t

  provide header information of firmware image passed by user

- enum hbi_cmd_t {
  HBI_CMD_LOAD_FWR_FROM_HOST, HBI_CMD_LOAD_CFGREC_FROM_HOST, HBI_CMD_LOAD_F-
  WR_COMPLETE, HBI_CMD_LOAD_FWRCFG_FROM_FLASH,
  HBI_CMD_SAVE_FWRCFG_TO_FLASH, HBI_CMD_ERASE_WHOLE_FLASH, HBI_CMD_ERASE_FWR-
  CFG_FROM_FLASH, HBI_CMD_START_FWR,
  HBI_CMD_END }

  Enumerates various HOST commands that can be issued to device.
- enum hbi_status_t {
  HBI_STATUS_NOT_INIT, HBI_STATUS_INTERNAL_ERR, HBI_STATUS_RESOURCE_ERR, HBI_STAT-
  US_INVALID_ARG,
  HBI_STATUS_BAD_HANDLE, HBI_STATUS_BAD_IMAGE, HBI_STATUS_FLASH_FULL, HBI_STATUS_-
  NO_FLASH_PRESENT,
  HBI_STATUS_COMMAND_ERR, HBI_STATUS_INCOMPAT_APP, HBI_STATUS_INVALID_STATE, HBI_-
  STATUS_OP_INCOMPLETE,
  HBI_STATUS_SUCCESS }

  enumerates various status codes of HBI Driver
- enum hbi_img_type_t { HBI_IMG_TYPE_FWR =0, HBI_IMG_TYPE_CR =1, HBI_IMG_TYPE_LAST }

  enumerates types of data as passed by application
- enum hbi_rst_mode_t { HBI_RST_POR }

  enumerates reset mode of device
- typedef uint32_t hbi_handle_t

  HBI Handle updated by driver during HBI_Open() call and used in further driver calls.
- typedef unsigned char user_buffer_t

  basic data type for buffer as should be used by user in call to HBI_read() ,HBI_write()
- hbi_status_t **HBI_init** (hbi_init_cfg_t *pCfg)

- hbi_status_t HBI_term (void)
  Driver termination function.
- hbi_status_t HBI_open (hbi_handle_t *pHandle, hbi_dev_cfg_t *pDevCfg)
  function to open a device.
- hbi_status_t HBI_close (hbi_handle_t handle)
  function to close a device opened using HBI_open().
- hbi_status_t HBI_reset (hbi_handle_t handle, hbi_rst_mode_t mode)
  function to reset a device.
- hbi_status_t HBI_read (hbi_handle_t handle, reg_addr_t reg, user_buffer_t *pData, size_t length)
  Reads the data from device memory starting from register address up to specified length.
- hbi_status_t HBI_write (hbi_handle_t handle, reg_addr_t reg, user_buffer_t *pData, size_t length)
  Writes the data from user buffer to device memory starting from register address up to specified length.
- hbi_status_t HBI_set_command (hbi_handle_t handle, hbi_cmd_t cmd, void *pCmdArgs)
  Set the Host Commands as listed in hbi_cmd_t enum.
- hbi_status_t HBI_wake (hbi_handle_t handle)
  wakes up the device from sleep mode.
- hbi_status_t HBI_sleep (hbi_handle_t handle)
  This function puts device to sleep mode.
- hbi_status_t HBI_get_header (hbi_data_t *pImg, hbi_img_hdr_t *pHeaderInfo)
  This function parses header of the image passed by user.

7.2 include/ssl.h File Reference

System Service Layer.

Data Structures

- struct ssl_port_access_t
  user passed in structure during read/write done by making a call to SSL_port_read(), SSL_port_write() and SSL_port_rw()
- enum ssl_status_t {
  SSL_STATUS_NOT_INIT = -1, SSL_STATUS_INVALID_ARG = -2, SSL_STATUS_INTERNAL_ERR = -3,
  SSL_STATUS_BAD_HANDLE = -4,
  SSL_STATUS_RESOURCE_ERR = -5, SSL_STATUS_TIMEOUT = -6, SSL_STATUS_FAILED = -7, SSL_STATUS_OP_INCOMPLETE = -8,
  SSL_STATUS_OK =0 }
  Enumerates Status Codes supported by SSL.
- enum ssl_wait_t { SSL_WAIT_NONE, SSL_WAIT_FOREVER}
  Enumerates wait type on lock.
- enum ssl_op_t { SSL_OP_PORT_RD =0x01, SSL_OP_PORT_WR =0x02, **SSL_OP_PORT_RW** =(SSL_OP_PORT_RD | SSL_OP_PORT_WR) }
  Enumerates port functions.
- ssl_status_t SSL_init (ssl_drv_cfg_t*pCfg)
  Driver initialization function.
- ssl_status_t SSL_lock_create (ssl_lock_handle_t *pLock, const char *pName, void *pOption)
  Lock Create Function.
- ssl_status_t SSL_lock (ssl_lock_handle_t lock_id, ssl_wait_t wait_type)
  Hold a lock.
- ssl_status_t SSL_unlock (ssl_lock_handle_t lock_id)

Release a lock.

- ssl_status_t SSL_lock_delete (ssl_lock_handle_tlock_id)
  Deletes lock.
- ssl_status_t SSL_term (void)
  Terminates SSL Driver.
- ssl_status_t SSL_port_open (ssl_port_handle_t *pHandle, ssl_dev_cfg_t *pDevCfg)
  Opens an port to device.
- ssl_status_t SSL_port_close (ssl_port_handle_t handle)
  Close the port opened to device.
- ssl_status_t SSL_port_read (ssl_port_handle_t handle, void *pDst, size_t *pNread)
  Reads the data from port into user buffer pointed by pDst.
- ssl_status_t SSL_port_write (ssl_port_handle_t handle, void *pSrc, size_t *pNwrite)
  Write the data from user buffer to device.
- ssl_status_t SSL_port_rw (ssl_port_handle_t handle, ssl_port_access_t *pPort)
  Read and Writes the data from and to device.
- ssl_status_t SSL_memset (void *pDst, int32_t val, size_t size)
  Sets the content of memory to specified value.
- ssl_status_t SSL_memcpy (void *pDst, const void *pSrc, size_t size)
  Copy the content of memory from source to destination pointer.
- ssl_status_t SSL_delay (uint32_t tmsec)
  Implements delay in millisecond.

7.2.1 Detailed Description

System Service Layer is combination of 3 header files :

ssl.h - Master file that prototypes functions and data structures to be ported

typedefs.h - Base file defining data types as used by layers above

vproc_dbg.h - Debug header file ported for debug messages. If no debugging supported, implementation to leave function/macro as stub but still should define

This is Platform-Specific layer and must be ported to host platform-specific function for correct functioning of HBI Driver over host platform

Expected and Intended use of data structures and functions are explained here. In case of confusion, developer may reach at <>

Definition in file ssl.h.

Index