cadence

SD Host Firmware Driver User Guide

Product Version 5.7.1

April 2020

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

SD Host Firmware Driver User Guide

Cadence Design Systems

Hardware Identifier:82f725edc9078f92a4ed7cea3ba0961e

Table of Contents

List of Figures

List of Tables

Chapter 1. Acronyms

API	Application Programming Interface.
Core Driver	Cadence Firmware component that provides IP programming abstraction.
CPS	Cadence Platform Services. A set of basic platform specific access functions acting as hardware abstraction layer for the core driver to operate.
IP	Intelectual Property
SD	Secure Digital
SDIO	Secure Digital Input/Output
MMC	Multi-Media Card
eMMC	embedded Multi-Media Card
UHSI	Ultra High Speed Phase I
SDMA	Single-operation Direct Memory Address
ADMA	Advanced-operation Direct Memory Address
PHY	Physical layer

Chapter 2. Document Purpose

This document is intended to serve as an overview and a reference for customers looking to use the Cadence SD Host Core Driver for the Cadence SD Host Core.

Chapter 3. Description

3.1. General firmware info

The FW provided by Cadence is of 2 types:

1. Production Code: Code built to production standards. i.e. it is fully verified with positive and negative testing.
2. Reference Code: This is example code for the platform specific portions of the customers' code base. Reference code is only provided as an example to help the customer with his environment. Ideally code snippets from this will be looked at, and the customer will create his own versions of the reference code that is specific to his environment.

3.2. Core driver features

Supported feautres:

- Execution card command.
- Support PIO/SDMA/ADMA2/ADMA3 engines.
- Support for emmc boot.
- Support UHSI modes up to SDR104 including tuning procedure.
- Support eMMC high speed modes up to HS400ES.
- Programmable bus width.
- Support for PHY configuration.
- Support for eMMC command queuing

The Init function configures a number of parameters on creation of the driver instance, e.g. register base address, the size of descriptor lists, the callback functions

Provide event polling API for upper layer ISR - the driver's isr() function should be called in response to a hardware interrupt

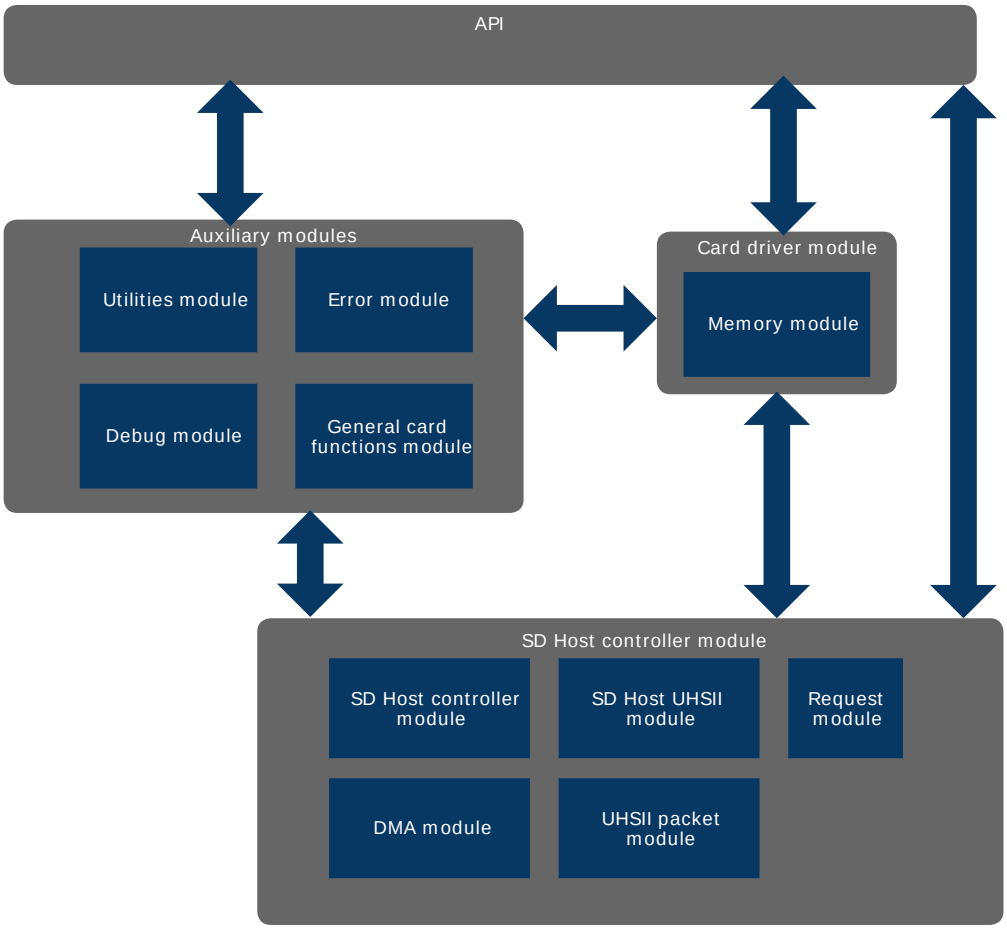
3.2.1. SD Host driver structure

Figure presents driver structure. The driver can be divided into four blocks: SD Host controller module, auxiliary modules, memory card driver module, and user API module

- The SD Host module is the lowest and the most important lyaer and it consists of:
  - SD Host controller core module is responsible for initializing, configuring SD Host controller, initializing the newly inserted card, and managing other card drivers (e.g. memory card driver), executing commands on cards, handling interrupts,
  - Request module is responsible for preparing requests objects
  - DMA module is responsible for preparing, configuring and finishing DMA transfers
- The memory card driver module is responsible for transferring data to/from card and for configuring the memory card.

- The Auxiliary modules comprise of:
  - The debug module can be helpful during driver testing and debugging
  - The error module defines all error codes.
  - The general card function module implements such useful functions as read SDIO/SD/MMC card registers, change card state, read card status, reset card etc
- API module shares user interface

Figure 3.1. Structure of the SD Host controller driver



3.3. Naming convensions

The SD Host Core Driver API will use the prefix CSDD_ for public top-level names of functions and data objects/ definitions.

3.4. System Usage

The SD Host Core Driver is supplied with reference code which illustrates the use of the core driver. It is provided as an example of how to use the driver API for basic operations. See the README file for further information.

Chapter 4. Core Driver Usage Model

4.1. Description

In order to support multiple instances and provide the minimum name space pollution, the Core Driver is implemented as a structure that contains a function pointer table.

A public header is provided to define the API object structure and the data structures used. The file is contained in the include directory of the distribution as *csdd.h*.

The API requires a private data pointer that provides the driver with instance information. It is critical that the upper software layer abides by the programming model.

Every API call apart from *probe* requires the private data address as the first parameter, to identify the driver instance and provide access to the internal state of that instance.

4.2. Probing the Hardware

The client code must obtain the object pointer to the core driver. It can be retrieved by issuing the *CSDD_GetInstance()* function. This provides access to all of the API functions, but initially only *probe* may be called.

Before initialising the core driver, the client must first invoke the *probe* function. The *probe* function returns the dynamic memory requirements for the driver, depending on the configuration specified. The configuration information is passed to *probe* in an *CSDD_Config* struct, although at this stage only those fields which affect the requirements need to be set, as indicated in the struct description. The client may repeat *probe* calls to customise configuration depending on available system resources.

The client code must then allocate the memory to the sizes returned by the probe function. .

```
...
CSDD_Config cfg;
CSDD_SysReq req;

/* get API function pointers for this driver instance */
CSDD_OBJ *csddObj = CSDD_GetInstance();

/* Initialise config structure fields required by probe */
cfg.regBase = 0x50000000;    /* IP core registers base address */

csddObj->probe(&cfg, &req);

printf("privateData object requires %u bytes\n", req.pDataSize);
...
```

4.3. SD Host driver initialisation

After this stage, the client must reserve the necessary memory areas for the driver's private data and hardware configuration (if required). The CSDD_Config struct should be filled out with the values for all initial configuration fields.

The driver can now be initialised with an *init* call, which will:

- initialise the driver instance

- configure the hardware as specified in the *config* parameter fields

e.g. *init* example, following on from *probe* example above:

```
...

CardInsertedCallback(void* pd, uint8_t slotIndex)
{
...
}

CardRemovedCallback(void* pd, uint8_t slotIndex)
{
...
}

AxiErrorCallback(void* pd, uint8_t axiError)
{
...
}

CSIH_Callbacks callbacks = {
    .cardInsertedCallback = CardInsertedCallback,
    .cardRemovedCallback = CardRemovedCallback,
    .axiErrorCallback = AxiErrorCallback
};

CSDD_SDIO_Host *privData = malloc(req.privDataSize);
cfg.descLogAddress = malloc(req.descSize);
cfg.descPhyAddress = config.descLogAddress;
cfg.idDescLogAddress = (uint32_t*)malloc(req.idDescSize);
cfg.idDescPhyAddress = cfg.idDescLogAddress;

if (sizeof(void*) > 4)
    config.dma64BitEn = 1;
else
    config.dma64BitEn = 0;

...


if (0 != (result = csddObj->init(privData, &cfg, &callbacks))) {
  printf("Error in csddObj->init: returned %d\n", result);
}
```

4.4. Card initialisation

The memory card driver is loaded and the user is able to use the memory card driver. Next checking slots operations should be executed to check if a card is present in a slot and attach found cards.

```
// load memory card driver, it is necessary to use this driver
csddObj->memoryCardLoadDriver(privData);

// Check if there is a card in a slot
// if card is present then it will be initialized in this function
```

```
status = csddObj->checkSlots(privData);
if (status){
    // handle error ....
}

// function gets info about a device status
csddObj->getDeviceState(privData, slotIndex, &deviceState);

if(!deviceState.inserted){
    //There is no card in slot
    //return 1;
}

if (deviceState.deviceType == CSDD_CARD_TYPE_MMC){
    //mmc device found
}
if (deviceState.deviceType == CSDD_CARD_TYPE_SDMEM){
    //sd device found
}
```

4.5. Access mode configuration

To change access mode user should use configure function.

```
// 100Mhz
uint32_t freq = 100000;
unsigned char Size = sizeof(freq);

// access mode selection
status = csddObj->configureAccessMode(privData, slotIndex,
                                      CSDD_ACCESS_MODE_SDR50);
if (status){
    //handle error
}

// clock frequency configuration
status = csddObj->configure(privData, slotIndex,
 CSDD_CONFIG_SET_CLK, &freq, &Size);
if (status){
    //handle error
}
```

4.6. Data transfer

Transfer data for all memory cards are executed by *memoryCardDataTransfer* and *memoryCardDataTransfer2* functions. These functions work on all memory devices (SD or eMMC).

4.6.1. Using ADMA with manually defined sub buffers

User can use sub buffers for two commands from API. There are: *execCardCommand* and *memoryCardDataTransfer2*. Sub buffer is a pair of values describing one buffer :an address and a buffer size. It can be used when one data transmission is split by a few buffers.

```
SDIO_SubBuffer subBuffers[2];
// define address of sub buffer 1
```

```
subBuffers[0].address = (uintptr_t)buffer1;
// define size of sub buffer 1
subBuffers[0].size = 512;
// define address of sub buffer 2
subBuffers[1].address = (uintptr_t)buffer2;
// define size of sub buffer 2
subBuffers[1].size = 1024;
// point data buffer to sub buffer array
pRequest->pDataBuffer = SubBuffers;
// set number of sub buffers
pRequest->SubBuffersCount = 2
.
.
.
// using sub bufers by execCardCommand function
csddObj->execCardCommand(privData, slotIndex, pRequest);
.
.
.
// using sub bufers by memoryCardDataTransfer2 function
csddObj->memoryCardDataTransfer2(privData, slotIndex, address, subBuffers,
     buffersSize, SDIO_TRANSFER_WRITE, 2);
```

4.7. Device specific functions and feaures

4.7.1. eMMC devices

4.7.1.1. Card access modes

Supported modes for eMMC devices.

- CSDD_ACCESS_MODE_HS_SDR
- CSDD_ACCESS_MODE_HS_DDR
- CSDD_ACCESS_MODE_HS_200
- CSDD_ACCESS_MODE_HS_400
- CSDD_ACCESS_MODE_HS_400_ES

4.7.1.2. Command queuing

Command queuing is a part of eMMC 5.1 standard. By default it is disabled. To enable it cQEnable function must be called. To transfer data in CQ mode mode yu need to use another set of functions.

```
/* enabling CQ */
uint8_t status;
CSDD_CQInitConfig cQConfig;

cQConfig.sendStatBlckCount = 1;
cQConfig.sendStatIdleTimer = 0x1000;

status = sdHostDriver->cQEnable(sdHost, &cQConfig);
if (status){
    // handle error
}

/* executing data transfer in CQ mode*/
CSDD_CQRequest request;
```

```
CSDD_CQRequestData buffers;

memset(&request, 0, sizeof(request));

status = sdHostDriver->cQGetUnusedTaskId(sdHost, &request.taskId);
if (status){
    // handle error
}

buffers.buffPhyAddr = (uintptr_t)writeBuffer;
buffers.bufferSize = DataSize;

request.blockAddress = 0;
request.blockCount = 4;
request.buffers = &buffers;
request.numberOfBuffers = 1;
request.transferDirection = CSDD_TRANSFER_WRITE;

status = sdHostDriver->cQAttachRequest(sdHost, &request);
if (status){
    // handle error
}

status = sdHostDriver->cQStartExecuteTask(sdHost, request.taskId);
if (status){
    // handle error
}

while(request.cQReqStat == CSDD_CQ_REQ_STAT_PENDING){
    /*if interrupts are disabled then we need to call
     *  interrupt handler manually in polling mode*/
    if (!withInt){
        bool handled;
        sdHostDriver->isr(sdHost, &handled);
    }
}
```

4.7.1.3. eMMC Boot

There are a few functions in the SD host driver to configure and execute eMMC boot operation.

There are:

- mmcGetParitionBootSize - get to know what is partiion size before program it
- mmcSetPartAccess - select partion which will be accessed. After selection standard *memoryCardDataTransfer* function are used to transfer data to selected partition.
- mmcSetBootPartition - set active boot partition for boot operation
- mmcExecuteBoot - execute boot operation

4.7.2. SD devices (UHSI and legacy modes)

4.7.2.1. Card access modes

Card access modes for SD UHSI devices:

- CSDD_ACCESS_MODE_SDR12

- CSDD_ACCESS_MODE_SDR25
- CSDD_ACCESS_MODE_SDR50
- CSDD_ACCESS_MODE_DDR50
- CSDD_ACCESS_MODE_SDR104

4.8. Interrupt mode

Driver can work with interrupts enabled or disabled. When user calls *start* function then interrupts are enabled and driver works in interrupt mode. If this function is not called the interrupts are disabled and driver works in polling mode. The most of communication with a card takes place inside the *execCardCommand* function. User should be aware how to use this function. When driver works in interrupt mode then user need only to check the status flag of *CSDD_Request* parameter passed to *execCardCommand* function and wait until it is 0xFF. If driver works in polling mode user need to keep executing isr function untill status flag of a *CSDD_Request* is not 0xFF. When execution card commands is made in other driver functions then polling is automatically made in these functions.

4.9. PHY configuration

For PHY configuration for SD6 and newer, combo PHY driver is recommended to be used. Please refer to CCP Core Driver user guide to get more information about combo PHY driver functionality. There are a few functions in Core Driver which are used together with combo PHY driver for PHY configuration. All of them contains CPhy in its name. For older SD host controller there is no Combo PHY so the previous PHY functions should be used: *writePhySet* and *readPhySet*. Please refer to reference code to get to know how to configure PHY.

4.10. Configuration of SD Host Controller Driver

The configurability of the CSDD driver helps to change CSDD driver options easily. All configuration options are defined in the CSDD_Config.h file. Below there is a description of all driver options:

Table 4.1. Configuration options

Configuration options	Description
SDIO_SDMA_SUPPORT	If the value is 0, then SDMA mode is disabled; if the value is 1, then SDMA mode is enabled
SDIO_ADMA1_SUPPORT	If the value is 0, then ADMA1 mode is disabled; if the value is 1, then ADMA1 mode is enabled
SDIO_ADMA2_SUPPORT	If the value is 0, then ADMA2 mode is disabled; if the value is 1, then ADMA2 mode is enabled
SDIO_ADMA3_SUPPORT	If the value is 0, then ADMA3 mode is disabled; if the value is 1, then ADMA3 mode is enabled
DEBOUNCING_TIME	This defines the debouncing time period.
COMMANDS_TIMEOUT	This is an iteration count, after which the timeout error will be reported if a command does not execute (mainly used in WaitForValue function).
ENABLE_CARD_INTERRUPT	Enables or disables, detecting a card interrupt (in interrupt mode). User would enable this option only if a procedure to handle card interrupt is implemented.
SYTEM_CLK_KHZ	It defines system clock in kHz. It is used as base clock if base clock in SRS16 register is 0

Configuration options	Description
MAX_DESCR_BUFF_SIZE	Buffer size for ADMA descriptors
MAX_COMMAND_DESCR_BUFF_SIZE	buffer size for ADMA 3 command descriptors
MAX_INTEGREATED_DESCR_BUFF_SIZE	buffer size for ADMA 3 integrated descriptors
SDIO_SLOT_COUNT	SDIO slot count in controller
MAX_SUPPORTED_DEVICE_COUNT	Maximum supported card/device drivers count. Default it is 2 SD and MMC.
USE_AUTO_CMD	Maximum supported card/device drivers count. Default it is 2 SD and MMC.
CHANGE_DATA_ENDIANITY_NODMA	This option can be used to change endianity of PIO transferred data form/to a card device.
POWER_UP_DELAY_US	This option defines delay in microseconds after power up before enabling clock.
SDIO_CFG_SUB_BUFFERS_COUNT	Number of sub-buffers used temporary by DMA module to create ADMA descriptors
SDIO_CFG_ENABLE_IO	Option enables support for IO cards. It should be disabled it was not tested yet
SDIO_CFG_RESET_COUNT	Configuration of how many times each reset operation shall be executed
SDIO_CFG_ENABLE_MMC	This option disables or enables support for MMC devices
SDIO_CFG_HOST_VER	Version of SD Host controller. It can be 3, 4 or 6 depending on which version of SD Host controller is used.

4.11. Platform depending configuration

The most important platform depending macros are located in CPS. But some extra macros are implemented in environment.h file. User can change this macros to adjust the Core Driver to own platform.

Endianess macros: CpuToLe32(a), LeToCpu32(a) macros are used during creation the DMA descriptors. If byte swapping is necessary it should be done in these macros.

GetByte macro is used to get one byte from 4 bytes word

Chapter 5. Configuration and Hardware Operation Information

5.1. Introduction

The following definitions specify the driver operation environment that is defined by hardware configuration or client code.

These defines are located in the header file of the core driver.

Table 5.1. Configuration and Hardware Operation Information

#define	value	description
CSDD_MAX_NUMBER_COMMAND	10U	CommandField structure maximum array size.
CSDD_MAX_DEV_PER_SLOT	1U	Max devices per one slot.
CSDD_CQ_REQUEST_PTR	CSDD_CQRequest*	

Chapter 6. Dynamic Data Structures

6.1. Introduction

This section defines the data structures used by the driver to provide hardware information, modification and dynamic operation of the driver.

These data structures are defined in the header file of the core driver and utilized by the API.

6.2. CSDD_SysReq_s

Type: struct

Fields

`pDataSize` private data size

`descSize` ADMA descriptors buffer size.

`idDescSize` ADMA3 Integrated descriptors buffer size.

6.3. CSDD_Config_s

Type: struct

Description

Structure defining all configuration parameters applied on driver start-up.

Fields

`regBase` base address of CSDD registers

`descLogAddress` logical address of ADMA2descriptors

`descPhyAddress` physical address of ADMA2descriptors

`idDescLogAddress` logical address of ADMA3 integrated descriptors

`idDescPhyAddress` physical address of ADMA3 integrated descriptors

`dma64BitEn` Enable DMA width 64bit.

6.4. CSDD_RequestFlags_s

Type: struct

Description

Structure describes a parameters of SD request.

Fields

`commandType`	Type of command.
`dataPresent`	This field specifies if request will transfer the data. 1 - data is present, 0 - no data
`responseType`	Response type.
`dataTransferDirection`	This variable specifies the data direction transfer.
`autoCMD12Enable`	if the flag has a value of 1, then AutoCMD12 is enabled; value of 0 means otherwise (flag to be taken into consideration only in data transfer commands)
`autoCMD23Enable`	1 - enable Auto CMD23, 0 - disable Auto CMD23
`hwResponseCheck`	1 - use hardware response checking
`appCmd`	1 - APP_CMD application-specific command
`isInfinite`	1 - means infinite transfer

6.5. CSDD_CommandField_s

Type: struct

Description

Structure holds data transfer details.

Fields

`argument`	Argument of command to execute.
`requestFlags`	Request flags.
`command`	Command to execute.
`blockCount`	number of blocks to send or receive. Field to be taken into consideration only in data transfer commands
`blockLen`	size of block data in bytes to send or receive; field to be taken into consideration only in data transfer commands
`subBuffersCount`	Number of sub-buffers.
`pDataBuffer`	data buffer or buffers for read/write data; field to be taken into consideration only in data transfer commands. If the SubBuffersCount parameter is 0 then it is just pointer to data. But if the SubBuffersCount is more than 0 then it is pointer to array of sub buffers.

6.6. CSDD_Request_s

Type: struct

Description

Structure describes a request which should be executed by host.

Fields

`response`	Response pointer (used in SD layer)
`responseTab`	Buffer for command response.
`pBufferPos`	Current buffer position.
`dataRemaining`	Number of bytes remaining in the transfer.
`status`	Request status.
`admaDescriptorTable`	ADMA descriptor table.
`IdDescriptorTable`	ADMA3 integrated descriptor table.
`pSdioHost`	Pointer to SDIO host object.
`busyCheckFlags`	Internal field.
`requestType`	Request type.
`respSize`	Size of response packet in bytes.
`respRegOffset`	Response register.
`infiniteStatus`	
`slotIndex`	slot index
`subCommandRequest`	pointer to subcommand
`commandCategory`	Indicate request is executed as a subcommand or Main command.
`cmdCount`	Number of command in a request. For noDMA, SDMA, ADMA1, ADMA2 must be 1 and for ADMA3 - 1 to CSDD_MAX_NUMBER_COMMAND
`pCmd`	Array to hold set of commands.

6.7. CSDD_SubBuffer_s

Type: struct

Description

structure contains info about one buffer.

Used to transfer group of buffers

Fields

`address` Address of sub buffer.

`size` Size in bytes of sub buffer.

6.8. CSDD_DeviceInfo_s

Type: struct

Description

Structure contains information that will be used to recognize type of supported devices.

Fields

`manufacturerCode`	Manufacturer code read from SDIO card CIS registers.
`manufacturerInformation`	Manufacturer information(Part Number and/or Revision) read from SDIO card CIS register.
`deviceType`	Type of device (SDIO card, SD/MMC memory card or combo card)
`pCardInterruptHandler`	Function pointer which should point to interrupt function which handle SDIO card interrupt.
`pCardInitialize`	Function pointer which should point to function which initialize the card device.
`pCardDeinitialize`	Function pointer which should point to function which de-initialize the card device.

6.9. CSDD_PhyDelaySettings_s

Type: struct

Fields

`highSpeed` PHY delay for SD High speed.

`defaultSpeed` PHY delay for SD default speed.

`uhsiSdr12` PHY delay for SDR12 mode.

`uhsiSdr25` PHY delay for SDR25 mode.

`uhsiSdr50` PHY delay for SDR50 mode.

`uhsiDdr50`	PHY delay for DDR50 mode.
`mmcLegacy`	PHY delay for eMMC legacy mode.
`mmcSdr`	PHY delay for eMMC High speed SDR mode.
`mmcDdr`	PHY delay for eMMC High speed DDR mode.
`mmcHs400`	PHY delay for eMMC High speed HS400 mode.

6.10. CSDD_MemoryCardInfo_s

Type: struct

Fields

`blockSize`	block size
`commandClasses`	card command classes
`deviceSizeMB`	device size in MB
`partialReadAllowed`	Defines whether partial block sizes can be used in block read commands.
`partialWriteAlloed`	Defines whether partial block sizes can be used in block write commands.
`writeBlkMisalign`	Defines if the data block to be written by one command can be spread over more than one physical block of the memory device.
`readBlkMisalign`	Defines if the data block to be read by one command can be spread over more than one physical block of the memory device.
`sectorSize`	The size of an erasable sector.
	The content of this register is a 7-bit binary coded value, defining the number of write blocks
`eraseBlkEn`	The EraseBlkEn defines the granularity of the unit size of the data to be erased.
`readBlkLen`	The maximum read data block length.
`writeBlkLen`	The maximum write data block length.

6.11. CSDD_DeviceState_s

Type: struct

Fields

`inserted`	is card physical located in slot
`attached`	is card properly detected and initialized
`deviceType`	device type

`uhsSupported` is Ultra High Speed I mode supported

6.12. CSDD_CQInitConfig_s

Type: struct

Description

Command queuing initial configuration.

Fields

`sendStatBlckCount` Send Status Command Block Counter.

if 0 then the CQE does not send CMD13 during data transfer. The value is 1, 2, or N means, the CQE sends CMD13 is transferred during last, one before last, or (N-1) before last block, respectively

`sendStatIdleTimer` time interval of polling device status (CMD13) when controller is idle

6.13. CSDD_CQRequestData_s

Type: struct

Description

Request describing single task.

Fields

`buffPhyAddr` physical address of data buffer

`bufferSize` Size of buffer in bytes.

6.14. CSDD_CQRequest_s

Type: struct

Description

Data transfer request.

Fields

`blockCount` Number of blocks to be read/written.

`blockAddress` Data block address.

`taskId` task ID - it must be unused

`buffers`	it define one data buffer with data
`numberOfBuffers`	number of buffers with data
`contextId`	context ID
`transferDirection`	Data transfer direction.
`highPriorityEn`	Enable disable high priority.
`queueBarrierEn`	Enable queue barrier.
`forceProgEn`	When 1 data shall be forcefully programmed to nonvolatile storage instead of volatile cache while cache is turned.
`tagRequestEn`	Enable tag request.
`intCoalEn`	If 1 Interrupt will be generated when interrupt coalescing conditions have been met. If 0 then interrupt will be generated when task is finished.
`reliableWriteEn`	Enable reliable write.
`descDataBuffer`	pointer to virtual memory area where descriptors can be hold. Virtual address. It is necessary if there is more than one data buffer is used.
`descDataPhyAddr`	address of physical memory area where descriptors can be hold. It is necessary if there is more than one data buffer is used.
`descDataSize`	size of memory area where descriptors can be hold. It is used by driver to verify if there is enough space to all transfer descriptors. It is necessary if there is more than one data buffer is used.
`cQReqStat`	Request status. Only driver can modify it.

6.15. CSDD_CQDcmdRequest_s

Type: struct

Description

Direct command request used to execute eMMC command by their index and argument.

It is not used to data transfer

Fields

`cmdIdx`	The index of the command to be sent to device.
`responseType`	response expected to be received from the device

`argument`	argument of the command to be sent to the device
`queueBarrierEn`	Enable queue barrier.
`cmdTiming`	1 - command may be sent to device during data activity or busy, 0 - command may not be sent to device during data activity or busy
`response`	command response get from device after command execution
`cQReqStat`	Request status. Only driver can modify it.

6.16. CSDD_CQIntCoalescingCfg_s

Type: struct

Description

Command queuing interrupt coalescing configuration.

Fields

`enable`	enable/disable Interrupt coalescing 1 - enable, 0 - disable
`threshold`	counter threshold number of tasks completions which are required in order to generate an interrupt
`timeout`	timeout value

6.17. CSDD_SDIO_SlotSettings_s

Type: struct

Fields

`DMA64_En`	DMA 64 bit enabled.
`HostVer4_En`	Host version 4 enabled.

6.18. CSDD_SDIO_CidRegister_s

Type: struct

Description

Structure defines CID register fields.

Fields

`manufacturerId`	An 8-bit binary number that identifies the card manufacturer.
`oemApplicationId`	A 2-character ASCII string that identifies the card OEM and/or the card contents.

`productName`	Product name is a string, 5-character ASCII string.
`productRevision`	The product revision is composed of two Binary Coded Decimal (BCD) digits.
`productSn`	Product serial number.
`manufacturingDate`	Manufacturing date (year and month)

6.19. CSDD_SDIO_Device_s

Type: struct

Description

Structure contains information about inserted card and functions to handle them.

Fields

`deviceType`	Type of device.
	Types of devices are defined here Devices
`DeviceCapacity`	Card capacity information.
	Types of device capacities are define here Capacities
`RCA`	Relative card address or NODE ID.
`SpecVersNumb`	Specification Version Number.
`SupportedBusWidths`	Card device supported bus width.
	They are defined here BusWidths. This parameter is logic sum of all supported bus widths
`CommonCISAddress`	Address of card's common Card Information Structure (CIS)
`IsSelected`	This variable defines if the card is selected or unselected.
`pCardInterruptHandler`	Function pointer which should point to interrupt function which hanle card interrupt.
`pCardDeinitialize`	Function pointer which should point to function which deinitialize the card device.
`CardDriverData`	Private driver data.
`pSlot`	Slot in which card is inserted.
`UhsiSupported`	Flag is set if UHS-I mode is supported by the device.
`CMD23Supported`	Flag is set if CMD23 commad is supported by the device.
`CMD20Supported`	Flag is set if CMD20 commad is supported by the device.
`cQDepth`	command queuing depth

6.20. CSDD_SDIO_Slot_s

Type: struct

Description

Structure contains information a SDIO Host slot.

Fields

`RegOffset`	start address of the slot registers
`DMABufferBoundary`	Size of buffer in the system memory which is used by DMA module.
`pDevice`	Current selected device.
`Devices`	These structure describes SD devices connected to the host.
`BusWidth`	Current set bus width.
	The bus widths are defined here BusWidths
`pCurrentRequest`	Pointer to current executing SD request.
`subCommandStatus`	
`CardInserted`	1 - card is inserted, 0 - there is no card in slot
`UhsiSelected`	voltage is switched, host and sd card work in UHSI mode
`AbortRequest`	Abort current transaction.
`IntSettings`	Settings of signaling the interrupts.
`ErrorRecorvering`	Flag is set if error recovering is already executing.
`pSdioHost`	pointer to the sdio host object
`SlotNr`	slot number
`DescriptorBuffer`	pointer to logical address of descriptor buffer
`DescriptorDMAAddr`	pointer to physical address of descriptor buffer
`IntegratedDescriptorBuffer`	
	pointer to logical address of ADMA3 integrated descriptor buffer
`IntegratedDescriptorDMAAddr`	
	pointer to physical address of ADMA3 integrated descriptor buffer
`ProgClockMode`	If this flag is set then programmable clock mode is enabled, if it is 0 then 10-bit divider clock mode is enabled.

`RetuningEnabled`	flag informs if re-tuning is currently enabled
`AccessMode`	fields keep info about current card access mode (UHS-I)
`RetuningRequest`	flags informs if host requests for re-tuning
`DataCount`	data count in bytes which are transfered since last re-tuning procedure
`DmaMode`	DMA mode.
`AuxBuff`	extra buffer used for read data during initialization (256 = 512/2)
`InterfaceType`	Interface type can be: - CSDD_INTERFACE_TYPE_SD.
`SlotSettings`	other slot settings
`dmaModeSelected`	indicate the current DMA mode
`NeedAttach`	flag is set when card is in slot but it is not attached by the driver
`CQDcmdEnabled`	CQ: it informs if DCMD mode is enabled.
	In DCMD mode the pointer CQCurrentReq[31] is always NULL
`CQEnabled`	is command queuing enabled
`CQHalted`	is command queuing halted
`CQIntCoalescingEn`	is interrupt coalescing enabled
`CQDescriptorBuffer`	pointer to logical address of task descriptor buffer
`CQDescriptorDmaAddr`	pointer to physical address of task descriptor buffer
`CQCurrentReq`	pointer to current request of each normal (not DCMD) task
`CQCurrentDcmdReq`	used only if DCMD mode is enabled
`CQDescSize`	task/transfer descriptor size in bytes

6.21. CSDD_SDIO_Host_s

Type: struct

Description

Structure contains information about inserted card and functions to handle them.

Fields

`RegOffset`	start address of the host registers
`Slots`	slots array which belong to the host
`HostBusMode`	host bus mode (SPI SD, in the current version only SD is supported)

`NumberOfSlots`	The host's slots count.
`SpecVersNumb`	Specification Version Number.
`pCardRemoved`	callback called when card removed interrupt occurs
`pCardInserted`	callback called when card inserted interrupt occurs
`pSetTuneVal`	function pointer which point to function which handle mmc tunning
`axiErrorCallback`	callback called when AXI error occurs
`drvData`	pointer to user driver data
`intEn`	interrupts enabled
`dma64BitEn`	DMA 64 bit enable.
`cqSupported`	emmc command queueing is supported
`hs400EsSupported`	HS 400ES supported.
`hostCtrlVer`	Host Controller Version.
`HostFixVer`	Fix Version Number : Number of the fix related to the Host Controller Version.

6.22. CSDD_Callbacks_s

Type: struct

Description

Function pointers to event notification callbacks issued by driver functions when interrupt occurs.

Each call passes the drivers privateData pointer and slotIndex for instance identification if necessary.

Fields

`cardInsertedCallback`

`cardRemovedCallback`

`axiErrorCallback`

`setTuneValCallback`

6.23. CSDD_CPhyConfigIoDelay_s

Type: struct

Description

Controls the IO delay related timings - Combo PHY.

Fields

`rwCompensate`

`idelayVal`

6.24. CSDD_CPhyConfigLvsi_s

Type: struct

Description

Controls LVSI related timings - Combo PHY.

Fields

`lvsiCnt`

`lvsiTcksel`

6.25. CSDD_CPhyConfigDfiRd_s

Type: struct

Description

Controls rddata and rdcmd parameters - Combo PHY.

Fields

`rddataSwap`

`rddataEn`

`rdcmdEn`

6.26. CSDD_CPhyConfigOutputDelay_s

Type: struct

Description

Controls CMD DAT output delay related timings - Combo PHY.

Fields

`wrdata1SdclkDly`

`wrdata0SdclkDly`

`wrcmd1SdclkDly`

`wrcmd0SdclkDly`

`wrdata1Dly`

`wrdata0Dly`

`wrcmd1Dly`

`wrcmd0Dly`

6.27. CSDD_CccrRegAddr

Type: enum

Description

CCCR card control registers definitions.

Values

`CSDD_CCCR_CCCR_SDIO_REV`	CCCR version number and SDIO specification version number register. Value: = 0U
`CSDD_CCCR_SD_SPEC_REV`	SD version number register. Value: = 1U
`CSDD_CCCR_IO_ENABLE`	IO enable function register. Value: = 2U
`CSDD_CCCR_IO_READY`	IO ready function register. Value: = 3U
`CSDD_CCCR_INT_ENABLE`	interrupt enable register Value: = 4U
`CSDD_CCCR_INT_PENDING`	interrupt pending register Value: = 5U
`CSDD_CCCR_ABORT`	IO Abort register. It used to stop a function transfer. Value: = 6U
`CSDD_CCCR_BUS_CONTROL`	Bus interface control register. Value: = 7U

`CSDD_CCCR_CARD_CAPABILITY`

Card capability register.

Value: = 8U

`CSDD_CCCR_CIS_POINTER` Pointer to card's common Card Information Structure (CIS)

Value: = 9U

`CSDD_CCCR_BUS_SUSPENDED` Bus suspend register.

Value: = 12U

`CSDD_CCCR_FUNCTION_SELECT`

Function select register.

Value: = 13U

`CSDD_CCCR_EXEC_FLAGS` Exec flags register.

The bits of this register are used by the host to determine the current execution status of all functions (1-7) and memory (0).

Value: = 14U

`CSDD_CCCR_READY_FLAGS` Ready flags register.

The bits of this register tell the host the read or write busy status for functions (1-7) and memory (0).

Value: = 15U

`CSDD_CCCR_FN0_BLOCK_SIZE` I/O block size for Function 0.

Value: = 16U

`CSDD_CCCR_POWER_CONTROL` Power control register.

Value: = 18U

`CSDD_CCCR_HIGH_SPEED` Bus speed select.

Value: = 19U

`CSDD_CCCR_UHSI_SUPPORT` UHS-I support info.

Value: = 20U

`CSDD_CCCR_DRIVER_STRENGTH`

Driver Strength.

Value: = 21U

`CSDD_CCCR_INT_EXT` Interrupt extension.

Value: = 22U

6.28. CSDD_TupleCode

Type: enum

Description

Tuple names definitions of SDIO card.

Values

`CSDD_TUPLE_CISTPL_NULL` NULL tuple.

Value: = 0U

`CSDD_TUPLE_CISTPL_CHECKSUM`

Checksum control.

Value: = 16U

`CSDD_TUPLE_CISTPL_VERS_1` Level 1 version/product information.

Value: = 21U

`CSDD_TUPLE_CISTPL_ALTSTR` Alternate language string tuple.

Value: = 22U

`CSDD_TUPLE_CISTPL_MANFID` Manufacturer identification string tuple.

Value: = 32U

`CSDD_TUPLE_CISTPL_FUNCID` Function identification tuple.

Value: = 33U

`CSDD_TUPLE_CISTPL_SDIO_STD`

Additional information for functions built to support application specifications for standard SDIO functions.

Value: = 145U

`CSDD_TUPLE_CISTPL_SDIO_EXT`

Reserved for future use with SDIO devices.

Value: = 146U

`CSDD_TUPLE_CISTPL_END` The End-of-chain Tuple.

Value: = 255U

6.29. CSDD_CmdType

Type: enum

Values

`CSDD_CMD_TYPE_NORMAL`	normal command Value: = 0U
`CSDD_CMD_TYPE_SUSPEND`	suspend command Value: = 1U
`CSDD_CMD_TYPE_RESUME`	resume command Value: = 2U
`CSDD_CMD_TYPE_ABORT`	abort command Value: = 3U
`CSDD_CMD_TYPE_TRANS_ABORT`	abort command in UHS-II native protocol Value: = 4U
`CSDD_CMD_TYPE_GO_DORMANT`	go into dormant state command (UHS-II command type) Value: = 5U

6.30. CSDD_CmdCat

Type: enum

Description

Command can be be MainCommand or SubCommand depending on the DAT line Busy during command execution.

Values

`CSDD_CMD_CAT_MAIN`	Command is called as Main command if it contains a subcommand. Value: = 0U
`CSDD_CMD_CAT_SUB`	Command is called as Sub command if it comes under a command. Value: = 1U
`CSDD_CMD_CAT_NORMAL`	Normal Command. Value: = 2U

6.31. CSDD_ResponseType

Type: enum

Values

CSDD_RESPONSE_NO_RESP	Value: = 0U
CSDD_RESPONSE_R1	Value: = 1U
CSDD_RESPONSE_R1B	Value: = 2U
CSDD_RESPONSE_R2	Value: = 3U
CSDD_RESPONSE_R3	Value: = 4U
CSDD_RESPONSE_R4	Value: = 5U
CSDD_RESPONSE_R5	Value: = 6U
CSDD_RESPONSE_R5B	Value: = 7U
CSDD_RESPONSE_R6	Value: = 8U
CSDD_RESPONSE_R7	Value: = 9U

6.32. CSDD_CardType

Type: enum

Values

CSDD_CARD_TYPE_NONE	Value: = 0U
CSDD_CARD_TYPE_SDIO	Value: = 1U
CSDD_CARD_TYPE_SDMEM	Value: = 2U
CSDD_CARD_TYPE_COMBO	Value: = 3U
CSDD_CARD_TYPE_MMC	Value: = 4U

6.33. CSDD_Capacity

Type: enum

Values

CSDD_CAPACITY_NORMAL	Value: = 1U
CSDD_CAPACITY_HIGH	Value: = 2U

6.34. CSDD_BusWidth

Type: enum

Values

`CSDD_BUS_WIDTH_1` Value: = 1U

`CSDD_BUS_WIDTH_4` Value: = 4U

`CSDD_BUS_WIDTH_8` Value: = 8U

`CSDD_BUS_WIDTH_4_DDR` Value: = 5U

`CSDD_BUS_WIDTH_8_DDR` Value: = 6U

6.35. CSDD_BusMode

Type: enum

Values

`CSDD_BUS_MODE_SPI` Value: = 1U

`CSDD_BUS_MODE_SD` Value: = 2U

6.36. CSDD_InterfaceType

Type: enum

Values

`CSDD_INTERFACE_TYPE_SD` Value: = 1U

`CSDD_INTERFACE_TYPE_UNKNOWN`

Value: = 2U

6.37. CSDD_RequestType

Type: enum

Values

`CSDD_REQUEST_TYPE_SD` SD layer request type (SD)

Value: = 1U

6.38. CSDD_ConfigCmd

Type: enum

Values

`CSDD_CONFIG_SET_CLK` set host clock frequency

Value: = 1U

`CSDD_CONFIG_SET_BUS_WIDTH`

set host bus width

Value: = 2U

`CSDD_CONFIG_SET_DAT_TIMEOUT`

set timeout on DAT line.

Argument is timeout in microseconds

Value: = 3U

`CSDD_CONFIG_DISABLE_SIGNAL_INTERRUPT`

disable signal interrupts

Value: = 4U

`CSDD_CONFIG_RESTORE_SIGNAL_INTERRUPT`

restore signal interrupts

Value: = 5U

`CSDD_CONFIG_SET_DMA_MODE` set DMA mode

Value: = 6U

6.39. CSDD_TransferDirection

Type: enum

Values

`CSDD_TRANSFER_READ` Value: = 1U

`CSDD_TRANSFER_WRITE` Value: = 0U

6.40. CSDD_SpeedMode

Type: enum

Values

`CSDD_ACCESS_MODE_SDR12` access mode - SDR12 default (CLK: max 25MHz, DT: max 12MB/s)

Value: = 0U

`CSDD_ACCESS_MODE_SDR25` access mode - SDR15 default (CLK: max 50MHz, DT: max 25MB/s)

Value: = 1U

`CSDD_ACCESS_MODE_SDR50` access mode - SDR50 default (CLK: max 100MHz, DT: max 50MB/s)

Value: = 2U

`CSDD_ACCESS_MODE_SDR104` access mode - SDR104 default (CLK: max 208MHz, DT: max 104MB/s)

Value: = 3U

`CSDD_ACCESS_MODE_DDR50` access mode - DDR50 default (CLK: max 50MHz, DT: max 50MB/s)

Value: = 4U

`CSDD_ACCESS_MODE_MMC_LEGACY`

MMC access mode - legacy mode (CLK: max 26MHz, DT: max 26MB/s)

Value: = 6U

`CSDD_ACCESS_MODE_HS_SDR` MMC access mode - high speed SDR mode (CLK: max 26MHz, DT: max 26MB/s)

Value: = 7U

`CSDD_ACCESS_MODE_HS_DDR` MMC access mode - high speed DDR mode (CLK: max 52MHz, DT: max 104MB/s)

Value: = 8U

`CSDD_ACCESS_MODE_HS_200` MMC access mode - HS200 mode (CLK: max 200MHz, DT: max 200MB/s)

Value: = 9U

`CSDD_ACCESS_MODE_HS_400` MMC access mode - HS400 mode (CLK: max 200MHz, DT: max 400MB/s)

Value: = 10U

`CSDD_ACCESS_MODE_HS_400_ES`

MMC access mode - HS400 using Enhanced Strobe (CLK: max 200MHz, DT: max 400MB/s)

Value: = 11U

6.41. CSDD_DmaMode

Type: enum

Values

`CSDD_SDMA_MODE` Standard DMA mode.

Value: = 0U

`CSDD_ADMA1_MODE` Advanced DMA Mode Version 1.

Value: = 1U

`CSDD_ADMA2_MODE` Advanced DMA Mode Version 2.

Value: = 2U

`CSDD_ADMA3_MODE` Advanced DMA Mode Version 3.

Value: = 3U

`CSDD_AUTO_MODE` DMA mode is selected automatically.

Value: = 4U

`CSDD_NONEDMA_MODE` DMA is disabled.

Value: = 255U

6.42. CSDD_ParitionAccess

Type: enum

Description

Partition name to set access.

Values

`CSDD_EMMC_ACCCESS_BOOT_NONE`

none partition

Value: = 0U

`CSDD_EMMC_ACCCESS_BOOT_1` boot partition 1

Value: = 1U

`CSDD_EMMC_ACCCESS_BOOT_2` boot partition 2

Value: = 2U

`CSDD_EMMC_ACCCESS_BOOT_GENERAL_PURP_1`

general purpose partition 1

Value: = 3U

`CSDD_EMMC_ACCCESS_BOOT_GENERAL_PURP_2`

general purpose partition 2

Value: = 4U

`CSDD_EMMC_ACCCESS_BOOT_GENERAL_PURP_3`

general purpose partition 3

Value: = 5U

`CSDD_EMMC_ACCCESS_BOOT_GENERAL_PURP_4`

general purpose partition 4

Value: = 6U

`CSDD_EMMC_ACCCESS_BOOT_RPMB`

Replay-protected memory-block partition.

Value: = 7U

6.43. CSDD_ParitionBoot

Type: enum

Description

Partition to boot.

Values

`CSDD_EMMC_BOOT_NONE`	none partition
	Value: = 0U
`CSDD_EMMC_BOOT_1`	set boot partition 1 for boot
	Value: = 1U
`CSDD_EMMC_BOOT_2`	set boot partition 2 for boot
	Value: = 2U
`CSDD_EMMC_BOOT_USER`	set user partition for boot
	Value: = 3U

6.44. CSDD_MmcConfigCmd

Type: enum

Description

Partition to boot.

Values

`CSDD_MMC_CARD_CONF_CARD_LOCK`

Lock device.

Value: = 4U

`CSDD_MMC_CARD_CONF_CARD_UNLOCK`

Unlock device.

Value: = 0U

`CSDD_MMC_CARD_CONF_SET_PASSWORD`

Set device password.

Value: = 1U

`CSDD_MMC_CARD_CONF_RESET_PASSWORD`

Reset device password.

Value: = 2U

6.45. CSDD_DriverStrengthType

Type: enum

Description

Driver strength type.

Values

`CSDD_SWITCH_DRIVER_STRENGTH_TYPE_A`

Device driver strength A.

Value: = 1U

`CSDD_SWITCH_DRIVER_STRENGTH_TYPE_B`

Device driver strength B.

Value: = 2U

`CSDD_SWITCH_DRIVER_STRENGTH_TYPE_C`

Device driver strength C.

Value: = 3U

`CSDD_SWITCH_DRIVER_STRENGTH_TYPE_D`

Device driver strength D.

Value: = 4U

6.46. CSDD_DriverCurrentLimit

Type: enum

Description

SD card driver current limit.

Values

`CSDD_SDCARD_SWITCH_CURRENT_LIMIT_200`

Card driver current limit - 200mA default.

Value: = 0U

`CSDD_SDCARD_SWITCH_CURRENT_LIMIT_400`

Card driver current limit - 400mA.

Value: = 1U

`CSDD_SDCARD_SWITCH_CURRENT_LIMIT_600`

Card driver current limit - 600mA.

Value: = 2U

`CSDD_SDCARD_SWITCH_CURRENT_LIMIT_800`

Card driver current limit - 800mA.

Value: = 3U

6.47. CSDD_EmmcCmdqTaskDescSize

Type: enum

Description

Command queuing task descriptor size.

Values

`CSDD_CQ_TASK_DESC_SIZE_64BIT`

Task descriptor size 64 bit.

Value: = 0U

`CSDD_CQ_TASK_DESC_SIZE_128BIT`

Task descriptor size 128 bit.

Value: = 1U

6.48. CSDD_CQReqStat

Type: enum

Description

Command queuing request status.

Values

`CSDD_CQ_REQ_STAT_FINISHED`

Task executed without errors.

Value: = 0U

`CSDD_CQ_REQ_STAT_ATTACHED`

Task is ready to execute.

Value: = 1U

`CSDD_CQ_REQ_STAT_PENDING` Task is currently executed.

Value: = 2U

`CSDD_CQ_REQ_STAT_FAILED` Task execution failed.

Value: = 3U

6.49. CSDD_PhyDelay

Type: enum

Description

PHY configuration delay type.

Values

`CSDD_PHY_DELAY_INPUT_HIGH_SPEED`

delay in the input path for High Speed work mode

Value: = 0U

`CSDD_PHY_DELAY_INPUT_DEFAULT_SPEED`

delay in the input path for Default Speed work mode

Value: = 1U

`CSDD_PHY_DELAY_INPUT_SDR12`

delay in the input path for SDR12 work mode

Value: = 2U

`CSDD_PHY_DELAY_INPUT_SDR25`

delay in the input path for SDR25 work mode

Value: = 3U

`CSDD_PHY_DELAY_INPUT_SDR50`

delay in the input path for SDR50 work mode

Value: = 4U

`CSDD_PHY_DELAY_INPUT_DDR50`

delay in the input path for DDR50 work mode

Value: = 5U

`CSDD_PHY_DELAY_INPUT_MMC_LEGACY`

delay in the input path for eMMC legacy work mode

Value: = 6U

`CSDD_PHY_DELAY_INPUT_MMC_SDR`

delay in the input path for eMMC SDR work mode

Value: = 7U

`CSDD_PHY_DELAY_INPUT_MMC_DDR`

delay in the input path for eMMC DDR work mode

Value: = 8U

`CSDD_PHY_DELAY_DLL_SDCLK` Value of the delay introduced on the sdclk output for all modes except HS200, HS400 and HS400_ES.

Value: = 11U

`CSDD_PHY_DELAY_DLL_HS_SDCLK`

Value of the delay introduced on the sdclk output for HS200, HS400 and HS400_ES speed mode.

Value: = 12U

`CSDD_PHY_DELAY_DLL_DAT_STROBE`

Value of the delay introduced on the dat_strobe input used in HS400 / HS400_ES speed mode.

Value: = 13U

6.50. CSDD_MEMORY_CARD_INFO

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct MEMORY_CARD_INFO_s` **`CSDD_MEMORY_CARD_INFO`**

6.51. CSDD_PMEMORY_CARD_INFO

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct MEMORY_CARD_INFO_s *` **`CSDD_PMEMORY_CARD_INFO`**

6.52. CSDD_SysReq

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_SysReq_s` **`CSDD_SysReq`**

6.53. CSDD_Config

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_Config_s` **`CSDD_Config`**

6.54. CSDD_RequestFlags

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_RequestFlags_s` **`CSDD_RequestFlags`**

6.55. CSDD_CommandField

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_CommandField_s` **`CSDD_CommandField`**

6.56. CSDD_Request

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_Request_s` **`CSDD_Request`**

6.57. CSDD_SubBuffer

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_SubBuffer_s` **`CSDD_SubBuffer`**

6.58. CSDD_DeviceInfo

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_DeviceInfo_s` **`CSDD_DeviceInfo`**

6.59. CSDD_PhyDelaySettings

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_PhyDelaySettings_s` **`CSDD_PhyDelaySettings`**

6.60. CSDD_MemoryCardInfo

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_MemoryCardInfo_s` **`CSDD_MemoryCardInfo`**

6.61. CSDD_DeviceState

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_DeviceState_s` **`CSDD_DeviceState`**

6.62. CSDD_CQInitConfig

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_CQInitConfig_s` **`CSDD_CQInitConfig`**

6.63. CSDD_CQRequestData

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_CQRequestData_s` **`CSDD_CQRequestData`**

6.64. CSDD_CQRequest

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_CQRequest_s` **`CSDD_CQRequest`**

6.65. CSDD_CQDcmdRequest

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration
`struct CSDD_CQDcmdRequest_s` **`CSDD_CQDcmdRequest`**

6.66. CSDD_CQIntCoalescingCfg

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration
`struct CSDD_CQIntCoalescingCfg_s` **`CSDD_CQIntCoalescingCfg`**

6.67. CSDD_SDIO_SlotSettings

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration
`struct CSDD_SDIO_SlotSettings_s` **`CSDD_SDIO_SlotSettings`**

6.68. CSDD_SDIO_CidRegister

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration
`struct CSDD_SDIO_CidRegister_s` **`CSDD_SDIO_CidRegister`**

6.69. CSDD_SDIO_Device

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration
`struct CSDD_SDIO_Device_s` **`CSDD_SDIO_Device`**

6.70. CSDD_SDIO_Slot

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration
`struct CSDD_SDIO_Slot_s` **`CSDD_SDIO_Slot`**

6.71. CSDD_SDIO_Host

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_SDIO_Host_s` **`CSDD_SDIO_Host`**

6.72. CSDD_Callbacks

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_Callbacks_s` **`CSDD_Callbacks`**

6.73. CSDD_CPhyConfigIoDelay

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_CPhyConfigIoDelay_s` **`CSDD_CPhyConfigIoDelay`**

6.74. CSDD_CPhyConfigLvsi

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_CPhyConfigLvsi_s` **`CSDD_CPhyConfigLvsi`**

6.75. CSDD_CPhyConfigDfiRd

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_CPhyConfigDfiRd_s` **`CSDD_CPhyConfigDfiRd`**

6.76. CSDD_CPhyConfigOutputDelay

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct CSDD_CPhyConfigOutputDelay_s` **`CSDD_CPhyConfigOutputDelay`**

6.77. CSDD_CardInsertedCallback

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`void(*` **`CSDD_CardInsertedCallback`** `)(void *pd, uint8_t slotIndex)`

6.78. CSDD_CardRemovedCallback

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`void(*` **`CSDD_CardRemovedCallback`** `)(void *pd, uint8_t slotIndex)`

6.79. CSDD_CardInterruptHandlerCallback

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`uint8_t(*` **`CSDD_CardInterruptHandlerCallback`** `)(void *pd, uint8_t slotIndex)`

6.80. CSDD_CardInitializeCallback

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`uint8_t(*` **`CSDD_CardInitializeCallback`** `)(CSDD_SDIO_Host *pd, uint8_t slotIndex)`

6.81. CSDD_CardDeinitializeCallback

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint8_t(*  CSDD_CardDeinitializeCallback  )(CSDD_SDIO_Host  *pd,  uint8_t
slotIndex)
```

6.82. CSDD_AxiErrorCallback

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* CSDD_AxiErrorCallback )(void *pd, uint8_t axiError)
```

6.83. CSDD_SetTuneValCallback

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
uint8_t(* CSDD_SetTuneValCallback )(const CSDD_SDIO_Host *pd, uint8_t tune_val)
```

Chapter 7. Driver Function API

7.1. Introduction

Prototypes for the driver API functions.

The user application can link statically to the necessary API functions and call them directly.

7.2. CSDD_Probe

Function Declaration

`uint32_t CSDD_Probe(`*`registerBase`*`, `*`req`*`);`

`uintptr_t registerBase`

`CSDD_SysReq *req`

Parameter List

`registerBase` [in]

base address of controller registers

`req` [out]

Description

Return Value

0 on success

ENOTSUP if configuration cannot be supported due to driver/hardware constraints

7.3. CSDD_Init

Function Declaration

`uint32_t CSDD_Init(`*`pD`*`, `*`config`*`, `*`callbacks`*`);`

`CSDD_SDIO_Host *pD`

`CSDD_Config *config`

`CSDD_Callbacks *callbacks`

Parameter List

`pD` [in]

private data

`config` [in]

startup configuration

`callbacks` [in]

pointer to callbacks functions

Description

Function initializes all objects.

It must be called as the first function before calling other driver functions

Return Value

0 on success or error code otherwise

7.4. CSDD_Start

Function Declaration

`uint32_t CSDD_Start(`*`pD`*`);`

`CSDD_SDIO_Host *pD`

Parameter List

`pD` [in]

private data

Description

Enable signaling interrupts.

Return Value

0 on success or error code otherwise

7.5. CSDD_Stop

Function Declaration

`uint32_t CSDD_Stop(`*`pD`*`);`

`CSDD_SDIO_Host *pD`

Parameter List

`pD` [in]

private data

Description

Disable signaling interrupts.

Return Value

0 on success or error code otherwise

7.6. CSDD_ExecCardCommand

Function Declaration

`uint32_t CSDD_ExecCardCommand(`*`pD`*`, `*`slotIndex`*`, `*`request`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

CSDD_Request *request
```

Parameter List

`pD`	[in] private data
`slotIndex`	[in] slot index
`request`	[in] request to execute

Description

Function executes a request which is given as the request parameter.

Return Value

0 on success or error code otherwise

7.7. CSDD_DeviceDetach

Function Declaration

`uint32_t CSDD_DeviceDetach(`*`pD`*`, `*`slotIndex`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex
```

Parameter List

`pD`	[in]

private data

`slotIndex` [in]

slot index

Description

Function detaches card, meaning that all information about card is removed in this function.

Return Value

0 on success or error code otherwise

7.8. CSDD_DeviceAttach

Function Declaration

`uint32_t CSDD_DeviceAttach(`*`pD`*`, `*`slotIndex`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

Description

Function initializes a newly inserted card, gathers information about it, and searches for card driver.

Return Value

0 on success or error code otherwise

7.9. CSDD_Abort

Function Declaration

`uint32_t CSDD_Abort(`*`pD`*`, `*`slotIndex`*`, `*`isSynchronous`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

uint8_t isSynchronous
```

Parameter List

`pD`	[in]
	private data
`slotIndex`	[in]
	slot index
`isSynchronous`	[in]
	abort type: synchronous or asynchronous

Description

function aborts data transfer

Return Value

0 on success or error code otherwise

7.10. CSDD_StandBy

Function Declaration

`uint32_t CSDD_StandBy(`*`pD`*`, `*`slotIndex`*`, `*`wakeupCondition`*`);`

`CSDD_SDIO_Host *pD`

`uint8_t slotIndex`

`uint8_t wakeupCondition`

Parameter List

`pD`	[in]
	private data
`slotIndex`	[in]
	slot index
`wakeupCondition`	[in]
	conditions of wakeup from standby mode

Description

sets slot of SD host controller to standby mode and waits for wakeup condition; in standby mode the clock is disabled

Return Value

0 on success or error code otherwise

7.11. CSDD_Configure

Function Declaration

`uint32_t CSDD_Configure(`*`pD`*`, `*`slotIndex`*`, `*`cmd`*`, `*`data`*`, `*`sizeOfData`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

CSDD_ConfigCmd cmd

void *data

uint8_t *sizeOfData
```

Parameter List

`pD`	[in]	private data
`slotIndex`	[in]	slot index
`cmd`	[in]	it describes what should be configured
`data`	[in]	buffer with configuration data
`sizeOfData`	[in]	size of data buffer in bytes

Description

configures the SD Host controller

Return Value

0 on success or error code otherwise

7.12. CSDD_Isr

Function Declaration

`void CSDD_Isr(`*`pD`*`, `*`handled`*`);`

```
CSDD_SDIO_Host *pD

bool *handled
```

Parameter List

`pD` [in]

private data

`handled` [in]

informs if interrupt occurred

Description

interrupt handler function, it should be called to handle SD host interrupts

7.13. CSDD_ConfigureHighSpeed

Function Declaration

`uint32_t CSDD_ConfigureHighSpeed(`*`pD`*`, `*`slotIndex`*`, `*`setHighSpeed`*`);`

```
CSDD_SDIO_Host *pD
```

```
uint8_t slotIndex
```

```
bool setHighSpeed
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`setHighSpeed` [in]

defines if high speed mode will be enabled SetHighSpeed = 1) or disabled (SetHighSpeed = 0)

Description

switches a card and host to work either in high speed mode or in normal mode

Return Value

0 on success or error code otherwise

7.14. CSDD_CheckSlots

Function Declaration

`uint32_t CSDD_CheckSlots(`*`pD`*`);`

`CSDD_SDIO_Host *pD`

Parameter List

`pD` [in]

private data

Description

checks slots states; depending on the slots status, calls deviceAttach or deviceDettach, or does nothing if slot state remains unchanged

Return Value

0 on success or error code otherwise

7.15. CSDD_CheckInterrupt

Function Declaration

`void CSDD_CheckInterrupt(`*`pD`*`, `*`slotIndex`*`, `*`status`*`);`

`CSDD_SDIO_Host *pD`

`uint8_t slotIndex`

`uint32_t status`

Parameter List

`pD`	[in]
	private data
`slotIndex`	[in]
	slot index
`status`	[in]

Description

7.16. CSDD_ConfigureAccessMode

Function Declaration

`uint32_t CSDD_ConfigureAccessMode(`*`pD`*`, `*`slotIndex`*`, `*`accessMode`*`);`

`CSDD_SDIO_Host *pD`

`uint8_t slotIndex`

`CSDD_SpeedMode accessMode`

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`accessMode` [in]

access mode to set

Description

Function configures card access mode.

Return Value

0 on success or error code otherwise

7.17. CSDD_Tuning

Function Declaration

`uint32_t CSDD_Tuning(`*`pD`*`, `*`slotIndex`*`);`

`CSDD_SDIO_Host *pD`

`uint8_t slotIndex`

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

Description

function executes tuning operation on a card but only if it is needed

Return Value

0 on success or error code otherwise

7.18. CSDD_ClockGeneratorSelect

Function Declaration

`uint32_t CSDD_ClockGeneratorSelect(`*`pD`*`, `*`slotIndex`*`, `*`progClkMode`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

uint8_t progClkMode
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`progClkMode` [in]

parameter defines if programmable clock mode should be enabled. 1 programmable clock mode will be enabled; 0 10-bit divider clock mode is enabled

Description

function executes tuning operation on a card but only if it is needed

Return Value

0 on success or error code otherwise

7.19. CSDD_PresetValueSwitch

Function Declaration

`uint32_t CSDD_PresetValueSwitch(`*`pD`*`, `*`slotIndex`*`, `*`enable`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

bool enable
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`enable` [in]

enable/disable preset value switching

Description

Function enables or disables preset value switching.

Return Value

0 on success or error code otherwise

7.20. CSDD_ConfigureDriverStrength

Function Declaration

`uint32_t CSDD_ConfigureDriverStrength(`*`pD`*`, `*`slotIndex`*`, `*`driverStrength`*`);`

`CSDD_SDIO_Host *pD`

`uint8_t slotIndex`

`CSDD_DriverStrengthType driverStrength`

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`driverStrength` [in]

new driver strength

Description

function configures driver strength of a card and the slot

Return Value

0 on success or error code otherwise

7.21. CSDD_MemoryCardLoadDriver

Function Declaration

`void CSDD_MemoryCardLoadDriver(`*`pD`*`);`

`CSDD_SDIO_Host *pD`

Parameter List

`pD` [in]

private data

Description

Function loads driver for the card.

7.22. CSDD_MemoryCardDataTransfer

Function Declaration

`uint32_t CSDD_MemoryCardDataTransfer(`*`pD`*`,`*`slotIndex`*`,`*`address`*`,`*`buffer`*`,`*`size`*`,`*`direction`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

uint32_t address

void *buffer

uint32_t size

CSDD_TransferDirection direction
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`address` [in]

address in card memory to/from which data will be transferred; address in blocks (512 bytes)

`buffer` [inout]

buffer with data to be written or to save data that was read

`size` [in]

size of buffer in bytes, Buffer size should be divisible by 512

`direction` [in]

Description

Function transfers data to/from memory card, function operates on blocks (512 bytes);.

Return Value

0 on success or error code otherwise

7.23. CSDD_MemoryCardDataTransfer2

Function Declaration

`uint32_t CSDD_MemoryCardDataTransfer2(`*`pD`*`, `*`slotIndex`*`, `*`address`*`, `*`buffer`*`, `*`size`*`, `*`direction`*`, `*`subBufferCount`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

uint32_t address

void *buffer

uint32_t size

CSDD_TransferDirection direction

uint32_t subBufferCount
```

Parameter List

`pD`	[in]	private data
`slotIndex`	[in]	slot index
`address`	[in]	memory card address to/from which data will be transferred; address in blocks (512 bytes);
`buffer`	[inout]	buffer (or buffers when SubBufferCount > 0) with data to be written or to save data that was read
`size`	[in]	BufferSize size of buffer in bytes
`direction`	[in]	parameter defines data transfer direction
`subBufferCount`	[in]	If it is bigger than 0 it means that Buffer is pointer to CSDD_SubBuffer array where are defined buffers with data to transfer and data sizes of each buffer

Description

Function transfers data to/from memory card, function operates on blocks (512 bytes)

Return Value

0 on success or error code otherwise

7.24. CSDD_MemoryCardConfigure

Function Declaration

`uint32_t CSDD_MemoryCardConfigure(`*`pD`*`, `*`slotIndex`*`, `*`cmd`*`, `*`data`*`, `*`size`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

CSDD_MmcConfigCmd cmd

uint8_t *data

uint8_t size
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`cmd` [in]

this parameter defines what operation will be executed

`data` [in]

buffer with data

`size` [in]

size of data in bytes

Description

function executes configuration commands on memory card

Return Value

0 on success or error code otherwise

7.25. CSDD_MemoryCardDataErase

Function Declaration

`uint32_t CSDD_MemoryCardDataErase(`*`pD`*`, `*`slotIndex`*`, `*`startBlockAddress`*`, `*`blockCount`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

uint32_t startBlockAddress

uint32_t blockCount
```

Parameter List

`pD`	[in] private data
`slotIndex`	[in] slot index
`startBlockAddress`	[in] address of the first block to be erased
`blockCount`	[in] number of blocks to be erased.

Description

Function erases block or blocks specified by startBlockAddress and blockCount parameteres.

Return Value

0 on success or error code otherwise

7.26. CSDD_MemCardPartialDataXfer

Function Declaration

`uint32_t CSDD_MemCardPartialDataXfer(`*`pD`*`, `*`slotIndex`*`, `*`address`*`, `*`buffer`*`, `*`size`*`, `*`direction`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

uint32_t address

void *buffer

uint32_t size
```

```
CSDD_TransferDirection direction
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`address` [in]

address in card memory to/from which data will be transferred; address in bytes

`buffer` [inout]

buffer with data to be written or to save data that was read

`size` [in]

size of buffer in bytes

`direction` [in]

size of buffer in bytes

Description

Function transfers data to/from memory card (byte oriented)

Return Value

0 on success or error code otherwise

7.27. CSDD_MemCardInfXferStart

Function Declaration

`uint32_t CSDD_MemCardInfXferStart(`*`pD`*`, `*`slotIndex`*`, `*`address`*`, `*`buffer`*`, `*`size`*`, `*`direction`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

uint32_t address

void *buffer

uint32_t size

CSDD_TransferDirection direction
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`address` [in]

address in card memory to/from which data will be transferred; address in blocks (512 bytes)

`buffer` [inout]

buffer with data to be written or to save data that was read

`size` [in]

size of buffer in bytes

`direction` [in]

parameter defines transfer direction

Description

Function transfers data in infinite mode to/from memory card.

Function operates on 512 data blocks. Data transfer can be continiued by memCardInfXferContinue. Trnasmission must be finished by memCardInfXferFinish

Return Value

0 on success or error code otherwise

7.28. CSDD_MemCardInfXferContinue

Function Declaration

`uint32_t CSDD_MemCardInfXferContinue(`*`pD`*`, `*`slotIndex`*`, `*`buffer`*`, `*`size`*`, `*`direction`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

void *buffer

uint32_t size

CSDD_TransferDirection direction
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`buffer` [inout]

buffer with data to be written or to save data that was read

`size` [in]

size of buffer in bytes

`direction` [in]

parameter defines transfer direction

Description

Return Value

0 on success or error code otherwise

7.29. CSDD_MemCardInfXferFinish

Function Declaration

`uint32_t CSDD_MemCardInfXferFinish(`*`pD`*`, `*`slotIndex`*`, `*`direction`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

CSDD_TransferDirection direction
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`direction` [in]

parameter defines transfer direction

Description

Function finishes data transfer in infinite mode.

Return Value

0 on success or error code otherwise

7.30. CSDD_MemCardDataXferNonBlock

Function Declaration

`uint32_t CSDD_MemCardDataXferNonBlock(`*`pD`*`, `*`slotIndex`*`, `*`address`*`, `*`buffer`*`, `*`size`*`, `*`direction`*`, `*`*request`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

uint32_t address

void *buffer

uint32_t size

CSDD_TransferDirection direction

void **request
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`address` [in]

address in card memory to/from which data will be transferred; address in blocks (512 bytes)

`buffer` [inout]

buffer with data to be written or to save data that was read

`size` [in]

size of buffer in bytes

`direction` [in]

parameter defines transfer direction

`request` [out]

current executing request

Description

Function transfers data to/from memory card.

Function operates on 512 data blocks. Function does not wait for operation finish. Therefore it returns pointer to current request. User using MemoryCard_FinishXferNonBlock function and request pointer can wait until operation finish and get the status of operation. Function needs AUTO_CMD option enabled

Return Value

0 on success or error code otherwise

7.31. CSDD_MemCardFinishXferNonBlock

Function Declaration

`uint32_t CSDD_MemCardFinishXferNonBlock(`*`pD`*`, `*`pRequest`*`);`

```
CSDD_SDIO_Host *pD

CSDD_Request *pRequest
```

Parameter List

`pD` [in]

private data

`pRequest` [in]

request to wait for

Description

Function waits until request finish and returns status of request execution.

Return Value

0 on success or error code otherwise

7.32. CSDD_PhySettingsSd3

Function Declaration

`uint32_t CSDD_PhySettingsSd3(`*`pD`*`, `*`slotIndex`*`, `*`phyDelaySet`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

CSDD_PhyDelaySettings *phyDelaySet
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`phyDelaySet` [in]

PHY delay settings

Description

Function sets delay line in UHS-I PHY hardware module.

Function works for SD Host 3 Function is obsolete writePhySet function should be used to configure PHY delays.

Return Value

0 on success or error code otherwise

7.33. CSDD_PhySettingsSd4

Function Declaration

`uint32_t CSDD_PhySettingsSd4(`*`pD`*`, `*`phyDelaySet`*`);`

```
CSDD_SDIO_Host *pD
```

```
CSDD_PhyDelaySettings *phyDelaySet
```

Parameter List

`pD` [in]

private data

`phyDelaySet` [in]

PHY delay settings

Description

Function sets delay line in UHS-I PHY hardware module.

Function works for SD Host 4. Function is obsolete writePhySet function should be used to configure PHY delays.

Return Value

0 on success or error code otherwise

7.34. CSDD_WritePhySet

Function Declaration

`uint32_t CSDD_WritePhySet(`*`pD`*`, `*`slotIndex`*`, `*`phyDelayType`*`, `*`delayVal`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

CSDD_PhyDelay phyDelayType

uint8_t delayVal
```

Parameter List

`pD`	[in]
	private data
`slotIndex`	[in]
	slot index
`phyDelayType`	[in]
	it defines which PHY delay should be written
`delayVal`	[in]
	value to be written to selected PHY delay

Description

Function configures one PHY delay line in UHS-I PHY hardware module.

Return Value

0 on success or error code otherwise

7.35. CSDD_ReadPhySet

Function Declaration

`uint32_t CSDD_ReadPhySet(`*`pD`*`, `*`slotIndex`*`, `*`phyDelayType`*`, `*`delayVal`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

CSDD_PhyDelay phyDelayType

uint8_t *delayVal
```

Parameter List

`pD`	[in]
	private data
`slotIndex`	[in]

slot index

`phyDelayType` [in]

it defines which PHY delay should be read:

`delayVal` [in]

read value of selected PHY delay

Description

Function configures one PHY delay line in UHS-I PHY hardware module.

Return Value

0 on success or error code otherwise

7.36. CSDD_ReadCardStatus

Function Declaration

`uint32_t CSDD_ReadCardStatus(`*`pD`*`, `*`slotIndex`*`, `*`cardStatus`*`);`

```
CSDD_SDIO_Host *pD
```

```
uint8_t slotIndex
```

```
uint32_t *cardStatus
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`cardStatus` [out]

pointer to memory where read card status shall be written

Description

reads card status.

Function returns error if command executing error appears or card error bits are set

Return Value

0 on success or error code otherwise

7.37. CSDD_SelectCard

Function Declaration

`uint32_t CSDD_SelectCard(`*`pD`*`, `*`slotIndex`*`, `*`rca`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

uint16_t rca
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`rca` [in]

card address in selecting card case or 0 in deselecting card case

Description

selects or deselects a card; if RCA parameter is 0, then card will be deselected, otherwise,a card of the RCA value will be selected

Return Value

0 on success or error code otherwise

7.38. CSDD_ResetCard

Function Declaration

`uint32_t CSDD_ResetCard(`*`pD`*`, `*`slotIndex`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

Description

resets a card using CMD0 command

Return Value

0 on success or error code otherwise

7.39. CSDD_ExecCmd55Command

Function Declaration

`uint32_t CSDD_ExecCmd55Command(`*`pD`*`, `*`slotIndex`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

Description

executes CMD55 command

Return Value

0 on success or error code otherwise

7.40. CSDD_AccessCccr

Function Declaration

`uint32_t CSDD_AccessCccr(`*`pD`*`, `*`slotIndex`*`, `*`transferDirection`*`, `*`data`*`, `*`size`*`, `*`registerAddress`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

CSDD_TransferDirection transferDirection

void *data

uint8_t size
```

`CSDD_CccrRegAddr registerAddress`

Parameter List

`pD`	[in]
	private data
`slotIndex`	[in]
	slot index
`transferDirection`	[in]
	parameter defines transfer direction
`data`	[inout]
	pointer to write/read data buffer
`size`	[in]
	size of Data buffer in bytes
`registerAddress`	[in]

Description

writes or reads the CCCR card registers; the CCCR is present in SDIO or combo cards only

Return Value

0 on success or error code otherwise

7.41. CSDD_ReadCsd

Function Declaration

`uint32_t CSDD_ReadCsd(`*`pD`*`, `*`slotIndex`*`, `*`buffer[4]`*`);`

`CSDD_SDIO_Host *pD`

`uint8_t slotIndex`

`uint32_t buffer[4]`

Parameter List

`pD`	[in]
	private data
`slotIndex`	[in]

slot index

`buffer` [out]

content of CSD register 16 bytes

Description

reads CSD register of the card

Return Value

0 on success or error code otherwise

7.42. CSDD_ReadExCsd

Function Declaration

`uint32_t CSDD_ReadExCsd(`*`pD`*`, `*`slotIndex`*`, `*`buffer`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

uint8_t *buffer
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`buffer` [out]

Description

function reads Extended CSD register from the card

Return Value

0 on success or error code otherwise

7.43. CSDD_GetTupleFromCis

Function Declaration

`uint32_t CSDD_GetTupleFromCis(`*`pD`*`, `*`slotIndex`*`, `*`tupleAddress`*`, `*`tupleCode`*`, `*`buffer`*`, `*`bufferSize`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

uint32_t tupleAddress

CSDD_TupleCode tupleCode

void *buffer

uint8_t bufferSize
```

Parameter List

`pD`	[in]	private data
`slotIndex`	[in]	slot index
`tupleAddress`	[in]	start address of the CIS register
`tupleCode`	[in]	code name of tuple
`buffer`	[out]	buffer for read tuple
`bufferSize`	[in]	size of buffer in bytes

Description

function reads tuple from the CIS card register to buffer; CIS register is present only in SDIO or combo cards;

Return Value

0 on success or error code otherwise

7.44. CSDD_ReadSdStatus

Function Declaration

`uint32_t CSDD_ReadSdStatus(`*`pD`*`, `*`slotIndex`*`, `*`buffer[64]`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex
```

`uint8_t buffer[64]`

Parameter List

`pD`	[in]
	private data
`slotIndex`	[in]
	slot index
`buffer`	[out]
	card status

Description

function reads SD card status

Return Value

0 on success or error code otherwise

7.45. CSDD_SetDriverStrength

Function Declaration

`uint32_t CSDD_SetDriverStrength(`*`pD`*`, `*`slotIndex`*`, `*`driverStrength`*`);`

`CSDD_SDIO_Host *pD`

`uint8_t slotIndex`

`CSDD_DriverStrengthType driverStrength`

Parameter List

`pD`	[in]
	private data
`slotIndex`	[in]
	slot index
`driverStrength`	[in]
	new driver strength value

Description

Function configures driver strength of a card.

Return Value

0 on success or error code otherwise

7.46. CSDD_ExecSetCurrentLimit

Function Declaration

`uint32_t CSDD_ExecSetCurrentLimit(`*`pD`*`, `*`slotIndex`*`, `*`currentLimit`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

CSDD_DriverCurrentLimit currentLimit
```

Parameter List

`pD`	[in]
	private data
`slotIndex`	[in]
	slot index
`currentLimit`	[in]

Description

Function sets a card current limit.

Return Value

0 on success or error code otherwise

7.47. CSDD_MmcSwitch

Function Declaration

`uint32_t CSDD_MmcSwitch(`*`pD`*`, `*`slotIndex`*`, `*`argIndex`*`, `*`argValue`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

uint8_t argIndex

uint8_t argValue
```

Parameter List

`pD`	[in]
	private data

`slotIndex` [in]

slot index

`argIndex` [in]

index of EXT_CSD register field to be modified

`argValue` [in]

new value to be set

Description

Function executes mmc switch command to modify EXT_CSD register.

Return Value

0 on success or error code otherwise

7.48. CSDD_MmcSetExtCsd

Function Declaration

`uint32_t CSDD_MmcSetExtCsd(`*`pD`*`, `*`slotIndex`*`, `*`byteNr`*`, `*`newValue`*`, `*`mask`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

uint8_t byteNr

uint8_t newValue

uint8_t mask
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`byteNr` [in]

index of EXT_CSD register field to be modified

`newValue` [in]

new value to be "or"

`mask` [in]

mask showing which part of byte should be masked

Description

Function reads byte pointed by byteNr parameter.

Next it masks the value using ~mask, at the end function makes "or" operation with newValue. Such value is written to the EXT_CSD register

Return Value

0 on success or error code otherwise

7.49. CSDD_MmcSetBootPartition

Function Declaration

`uint32_t CSDD_MmcSetBootPartition(`*`pD`*`, `*`slotIndex`*`, `*`partition`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

CSDD_ParitionBoot partition
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`partition` [in]

partition to be active to boot

Description

Function sets active partition to boot operation.

Return Value

0 on success or error code otherwise

7.50. CSDD_MmcSetPartAccess

Function Declaration

`uint32_t CSDD_MmcSetPartAccess(`*`pD`*`, `*`slotIndex`*`, `*`partition`*`);`

```
CSDD_SDIO_Host *pD
```

`uint8_t slotIndex`

`CSDD_ParitionAccess partition`

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`partition` [in]

partition to be active to access

Description

Function sets active partition to access.

Return Value

0 on success or error code otherwise

7.51. CSDD_MmcSetBootAck

Function Declaration

`uint32_t CSDD_MmcSetBootAck(`*`pD`*`, `*`slotIndex`*`, `*`enableBootAck`*`);`

`CSDD_SDIO_Host *pD`

`uint8_t slotIndex`

`bool enableBootAck`

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`enableBootAck` [in]

enable/disable eMMC boot ACK

Description

Function sets active partition to access.

Return Value

0 on success or error code otherwise

7.52. CSDD_MmcExecuteBoot

Function Declaration

`uint32_t CSDD_MmcExecuteBoot(`*`pD`*`, `*`slotIndex`*`, `*`buffer`*`, `*`size`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

void *buffer

uint32_t size
```

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`buffer` [out]

buffer for read

`size` [in]

buffer size in bytes

Description

Function executes boot operation.

It reads data from active boot partition to the buffer

Return Value

0 on success or error code otherwise

7.53. CSDD_MmcGetParitionBootSize

Function Declaration

`uint32_t CSDD_MmcGetParitionBootSize(`*`pD`*`, `*`slotIndex`*`, `*`bootSize`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex
```

`uint32_t *bootSize`

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`bootSize` [out]

size of boot partition

Description

Function gets boot partition size.

Return Value

0 on success or error code otherwise

7.54. CSDD_GetInterfaceType

Function Declaration

`uint32_t CSDD_GetInterfaceType(`*`pD`*`, `*`slotIndex`*`, `*`interfaceType`*`);`

`CSDD_SDIO_Host *pD`

`uint8_t slotIndex`

`CSDD_InterfaceType *interfaceType`

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`interfaceType` [out]

interface type

Description

function gets current device interface type (SD legacy)

Return Value

0 on success or error code otherwise

7.55. CSDD_GetDeviceState

Function Declaration

`uint32_t CSDD_GetDeviceState(`*`pD`*`, `*`slotIndex`*`, `*`deviceState`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

CSDD_DeviceState *deviceState
```

Parameter List

`pD`	[in] private data
`slotIndex`	[in] slot index
`deviceState`	[out] device state and info

Description

function gets current device state/info

Return Value

0 on success or error code otherwise

7.56. CSDD_MemoryCardGetSecCount

Function Declaration

`uint32_t CSDD_MemoryCardGetSecCount(`*`pD`*`, `*`slotIndex`*`, `*`sectorCount`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

uint32_t *sectorCount
```

Parameter List

`pD`	[in] private data

`slotIndex` [in]

slot index

`sectorCount` [out]

sector count

Description

Function gets sectors count.

Return Value

0 on success or error code otherwise

7.57. CSDD_SetDriverData

Function Declaration

`uint32_t CSDD_SetDriverData(`*`pD`*`, `*`drvData`*`);`

```
CSDD_SDIO_Host *pD
```

```
void *drvData
```

Parameter List

`pD` [in]

private data

`drvData` [in]

driver data

Description

Function used to save pointer to driver data.

Return Value

0 on success or error code otherwise

7.58. CSDD_GetDriverData

Function Declaration

`uint32_t CSDD_GetDriverData(`*`pD`*`, `*`*drvData`*`);`

```
CSDD_SDIO_Host *pD
```

```
const void **drvData
```

Parameter List

`pD` [in]

private data

`drvData` [out]

driver data

Description

Function used get pointer to driver data saved by setDriverData function.

Return Value

0 on success or error code otherwise

7.59. CSDD_SimpleInit

Function Declaration

`uint32_t CSDD_SimpleInit(`*`pD`*`, `*`slotIndex`*`, `*`clk`*`);`

`CSDD_SDIO_Host *pD`

`uint8_t slotIndex`

`uint32_t clk`

Parameter List

`pD` [in]

private data

`slotIndex` [in]

slot index

`clk` [in]

clock frequency in hertz

Description

Function makes very simple host initialization just enables clock and supply the power.

Function is useful for emmc boot operation

Return Value

0 on success or error code otherwise

7.60. CSDD_ResetHost

Function Declaration

`uint32_t CSDD_ResetHost(`*`pD`*`);`

```
CSDD_SDIO_Host *pD
```

Parameter List

`pD` [in]

private data

Description

Function resets SD Host controller.

Return Value

0 on success or error code otherwise

7.61. CSDD_GetRca

Function Declaration

`uint32_t CSDD_GetRca(`*`pD`*`, `*`slotIndex`*`, `*`rca`*`);`

```
CSDD_SDIO_Host *pD
```

```
uint8_t slotIndex
```

```
uint16_t *rca
```

Parameter List

`pD`	[in]
	private data
`slotIndex`	[in]
	slot index
`rca`	[out]
	RCA address

Description

Function gets RCA address of conneted device.

Return Value

0 on success or error code otherwise

7.62. CSDD_CQEnable

Function Declaration

`uint32_t CSDD_CQEnable(`*`pD`*`, `*`cqConfig`*`);`

`CSDD_SDIO_Host *pD`

`CSDD_CQInitConfig *cqConfig`

Parameter List

`pD` [in]

private data

`cqConfig` [in]

Initial configuration.

Description

Function enables command queuing and set initialize configuration.

Return Value

0 on success or error code otherwise

7.63. CSDD_CQDisable

Function Declaration

`uint32_t CSDD_CQDisable(`*`pD`*`);`

`CSDD_SDIO_Host *pD`

Parameter List

`pD` [in]

private data

Description

Function disables command queuing.

Return Value

0 on success or error code otherwise

7.64. CSDD_CQGetInitConfig

Function Declaration

`uint32_t CSDD_CQGetInitConfig(`*`pD`*`, `*`cqConfig`*`);`

```
CSDD_SDIO_Host *pD

CSDD_CQInitConfig *cqConfig
```

Parameter List

`pD` [in]

private data

`cqConfig` [out]

current command queue configuration

Description

Function get command queuing initial configuration passed in in cQEnable function.

Return Value

0 on success or error code otherwise

7.65. CSDD_CQGetUnusedTaskId

Function Declaration

`uint32_t CSDD_CQGetUnusedTaskId(`*`pD`*`, `*`taskId`*`);`

```
CSDD_SDIO_Host *pD

uint8_t *taskId
```

Parameter List

`pD` [in]

private data

`taskId` [out]

unused task ID

Description

Function searches and gets unused task ID.

Return Value

0 on success or error code otherwise

7.66. CSDD_CQStartExecuteTask

Function Declaration

`uint32_t CSDD_CQStartExecuteTask(`*`pD`*`, `*`taskId`*`);`

```
CSDD_SDIO_Host *pD

uint8_t taskId
```

Parameter List

`pD` [in]

private data

`taskId` [in]

task ID which shall be executed

Description

Function starts task execution.

Task must be attached before by cQAttachRequest function

Return Value

0 on success or error code otherwise

7.67. CSDD_CQAttachRequest

Function Declaration

`uint32_t CSDD_CQAttachRequest(`*`pD`*`, `*`request`*`);`

```
CSDD_SDIO_Host *pD

CSDD_CQRequest *request
```

Parameter List

`pD` [in]

private data

`request` [in]

a request to be attached

Description

Function attaches new request to a task.

Function create new descriptors. Function cannot be used to attach direct requests. Execution of request is not started by this function.

Return Value

0 on success or error code otherwise

7.68. CSDD_CQExecuteDcmdRequest

Function Declaration

`uint32_t CSDD_CQExecuteDcmdRequest(`*`pD`*`, `*`request`*`);`

```
CSDD_SDIO_Host *pD

CSDD_CQDcmdRequest *request
```

Parameter List

`pD` [in]

private data

`request` [in]

a direct request to be executed

Description

Function attaches new request to the direct task.

Function create new descriptors. Function start of execution direct task. Function can be only to direct requests.

Return Value

0 on success or error code otherwise

7.69. CSDD_CQGetDirectCmdConfig

Function Declaration

`uint32_t CSDD_CQGetDirectCmdConfig(`*`pD`*`, `*`enable`*`);`

```
CSDD_SDIO_Host *pD

uint8_t *enable
```

Parameter List

`pD` [in]

private data

`enable` [out]

it informs if direct command is enabled or disabled

Description

Function gets current configuration of command queuing direct command.

Return Value

0 on success or error code otherwise

7.70. CSDD_CQSetDirectCmdConfig

Function Declaration

`uint32_t CSDD_CQSetDirectCmdConfig(`*`pD`*`, `*`enable`*`);`

```
CSDD_SDIO_Host *pD

uint8_t enable
```

Parameter List

`pD` [in]

private data

`enable` [out]

if it is then direct command will be enabled, if it is 0 then direct command will be disabled

Description

Function sets new configuration of command queuing direct command.

Return Value

0 on success or error code otherwise

7.71. CSDD_CQSetIntCoalescingConfig

Function Declaration

`uint32_t CSDD_CQSetIntCoalescingConfig(`*`pD`*`, `*`config`*`);`

```
CSDD_SDIO_Host *pD

CSDD_CQIntCoalescingCfg *config
```

Parameter List

`pD` [in]

private data

[ ]

Description

Function sets new configuration of command queuing interrupt coalescing.

7.72. CSDD_CQGetIntCoalescingConfig

Function Declaration

`uint32_t CSDD_CQGetIntCoalescingConfig(`*`pD`*`, `*`config`*`);`

```
CSDD_SDIO_Host *pD

CSDD_CQIntCoalescingCfg *config
```

Parameter List

`pD` [in]

private data

`config` [out]

current interrupt coalescing configuration

Description

Function gets current configuration of command queuing interrupt coalescing.

Return Value

0 on success or error code otherwise

7.73. CSDD_CQGetIntCoalescingTimeoutBase

Function Declaration

`uint32_t CSDD_CQGetIntCoalescingTimeoutBase(`*`pD`*`, `*`clockFreqKHz`*`);`

```
CSDD_SDIO_Host *pD

uint32_t *clockFreqKHz
```

Parameter List

`pD` [in]

private data

`clockFreqKHz` [out]

interrupt coalescing timeout base clock frequency in KHz. It can be usefull to calculate interrupt coalescing timeout value

Description

Function gets current configuration of command queuing interrupt coalescing.

Return Value

0 on success or error code otherwise

7.74. CSDD_CQStartExecuteTasks

Function Declaration

`uint32_t CSDD_CQStartExecuteTasks(`*`pD`*`, `*`taskIds`*`);`

```
CSDD_SDIO_Host *pD
```

```
uint32_t taskIds
```

Parameter List

`pD` [in]

private data

`taskIds` [in]

task IDs OR'd combination of bit-flags selecting the tasks to be executed. Bit 0 - task ID 0, bit 1 task ID 1 ...

Description

Function starts tasks execution.

Tasks must be attached before by cQAttachRequest function

Return Value

0 on success or error code otherwise

7.75. CSDD_CQHalt

Function Declaration

`uint32_t CSDD_CQHalt(`*`pD`*`, `*`set`*`);`

```
CSDD_SDIO_Host *pD
```

```
uint8_t set
```

Parameter List

`pD` [in]

private data

`set` [in]

1 - enter to halt state, 0 - exit from halt state

Description

Function put command queueing to halt state or exit from halt state.

Behaviour depends on set parameter.

Return Value

0 on success or error code otherwise

7.76. CSDD_CQTaskDiscard

Function Declaration

`uint32_t CSDD_CQTaskDiscard(`*`pD`*`, `*`taskId`*`);`

```
CSDD_SDIO_Host *pD
```

```
uint8_t taskId
```

Parameter List

`pD` [in]

private data

`taskId` [in]

task ID to discard

Description

Function dicards one task.

Return Value

0 on success or error code otherwise

7.77. CSDD_CQAllTasksDiscard

Function Declaration

`uint32_t CSDD_CQAllTasksDiscard(`*`pD`*`);`

```
CSDD_SDIO_Host *pD
```

Parameter List

`pD` [in]

private data

Description

Function dicards all tasks.

Return Value

0 on success or error code otherwise

7.78. CSDD_CQResetIntCoalCounters

Function Declaration

`uint32_t CSDD_CQResetIntCoalCounters(`*`pD`*`);`

`CSDD_SDIO_Host *pD`

Parameter List

`pD` [in]

private data

Description

Function resets interrupt coalescing timer and counter.

Return Value

0 on success or error code otherwise

7.79. CSDD_CQSetResponseErrorMask

Function Declaration

`uint32_t CSDD_CQSetResponseErrorMask(`*`pD`*`, `*`errorMask`*`);`

`CSDD_SDIO_Host *pD`

`uint32_t errorMask`

Parameter List

`pD` [in]

private data

`errorMask` [in]

bit mask informs CQ engine which bits should be treated as error in command response

Description

Function writes error mask for command response.

Return Value

0 on success or error code otherwise

7.80. CSDD_CQGetResponseErrorMask

Function Declaration

`uint32_t CSDD_CQGetResponseErrorMask(`*`pD`*`, `*`errorMask`*`);`

```
CSDD_SDIO_Host *pD

uint32_t *errorMask
```

Parameter List

`pD`	[in] private data
`errorMask`	[out] bit mask informs CQ engine which bits should be treated as error in command response

Description

Function reads error mask for command response.

Return Value

0 on success or error code otherwise

7.81. CSDD_GetBaseClk

Function Declaration

`uint32_t CSDD_GetBaseClk(`*`pD`*`, `*`slotIndex`*`, `*`frequencyKHz`*`);`

```
CSDD_SDIO_Host *pD

uint8_t slotIndex

uint32_t *frequencyKHz
```

Parameter List

`pD`	[in] private data
`slotIndex`	[in] slot index
`frequencyKHz`	[out] base clock

Description

Function reads base clock.

Return Value

0 on success or error code otherwise

7.82. CSDD_WaitForRequest

Function Declaration

`uint32_t CSDD_WaitForRequest(`*`pD`*`, `*`pRequest`*`);`

```
CSDD_SDIO_Host *pD

CSDD_Request *pRequest
```

Parameter List

`pD` [in]

private data

`pRequest` [in]

request to wait for

Description

Function waits until request finish and returns status of request execution.

Return Value

0 on success or error code otherwise

7.83. CSDD_SetCPhyConfigIoDelay

Function Declaration

`uint32_t CSDD_SetCPhyConfigIoDelay(`*`pD`*`, `*`ioDelay`*`);`

```
CSDD_SDIO_Host *pD

CSDD_CPhyConfigIoDelay *ioDelay
```

Parameter List

`pD` [in]

private data

`ioDelay` [in]

configuration

Description

Return Value

0 on success or error code otherwise

7.84. CSDD_GetCPhyConfigIoDelay

Function Declaration

`uint32_t CSDD_GetCPhyConfigIoDelay(`*`pD`*`, `*`ioDelay`*`);`

```
CSDD_SDIO_Host *pD

CSDD_CPhyConfigIoDelay *ioDelay
```

Parameter List

`pD` [in]

private data

`ioDelay` [out]

configuration

Description

Return Value

0 on success or error code otherwise

7.85. CSDD_SetCPhyConfigLvsi

Function Declaration

`uint32_t CSDD_SetCPhyConfigLvsi(`*`pD`*`, `*`lvsi`*`);`

```
CSDD_SDIO_Host *pD

CSDD_CPhyConfigLvsi *lvsi
```

Parameter List

`pD` [in]

private data

`lvsi` [in]

configuration

Description

Return Value

0 on success or error code otherwise

7.86. CSDD_GetCPhyConfigLvsi

Function Declaration

`uint32_t CSDD_GetCPhyConfigLvsi(`*`pD`*`, `*`lvsi`*`);`

`CSDD_SDIO_Host *pD`

`CSDD_CPhyConfigLvsi *lvsi`

Parameter List

`pD` [in]

private data

`lvsi` [out]

configuration

Description

Return Value

0 on success or error code otherwise

7.87. CSDD_SetCPhyConfigDfiRd

Function Declaration

`uint32_t CSDD_SetCPhyConfigDfiRd(`*`pD`*`, `*`dfiRd`*`);`

`CSDD_SDIO_Host *pD`

`CSDD_CPhyConfigDfiRd *dfiRd`

Parameter List

`pD` [in]

private data

`dfiRd` [in]

configuration

Description

Return Value

0 on success or error code otherwise

7.88. CSDD_GetCPhyConfigDfiRd

Function Declaration

`uint32_t CSDD_GetCPhyConfigDfiRd(`*`pD`*`, `*`dfiRd`*`);`

```
CSDD_SDIO_Host *pD

CSDD_CPhyConfigDfiRd *dfiRd
```

Parameter List

`pD` [in]

private data

`dfiRd` [out]

configuration

Description

Return Value

0 on success or error code otherwise

7.89. CSDD_SetCPhyConfigOutputDelay

Function Declaration

`uint32_t CSDD_SetCPhyConfigOutputDelay(`*`pD`*`, `*`outputDelay`*`);`

```
CSDD_SDIO_Host *pD

CSDD_CPhyConfigOutputDelay *outputDelay
```

Parameter List

`pD` [in]

private data

`outputDelay` [in]

configuration

Description

Return Value

0 on success or error code otherwise

7.90. CSDD_GetCPhyConfigOutputDelay

Function Declaration

`uint32_t CSDD_GetCPhyConfigOutputDelay(`*`pD`*`, `*`outputDelay`*`);`

`CSDD_SDIO_Host *pD`

`CSDD_CPhyConfigOutputDelay *outputDelay`

Parameter List

`pD` [in]

private data

`outputDelay` [out]

configuration

Description

Return Value

0 on success or error code otherwise

7.91. CSDD_CPhyDllReset

Function Declaration

`uint32_t CSDD_CPhyDllReset(`*`pD`*`, `*`doReset`*`);`

`CSDD_SDIO_Host *pD`

`bool doReset`

Parameter List

`pD` [in]

private data

`doReset` [in]

enable/disable reset

Description

Return Value

0 on success or error code otherwise

7.92. CSDD_SetCPhyExtMode

Function Declaration

`uint32_t CSDD_SetCPhyExtMode(`*`pD`*`, `*`extendedWrMode`*`, `*`extendedRdMode`*`);`

`CSDD_SDIO_Host *pD`

`bool extendedWrMode`

`bool extendedRdMode`

Parameter List

`pD` [in]

private data

`extendedWrMode` [in]

`extendedRdMode` [in]

Description

Return Value

0 on success or error code otherwise

7.93. CSDD_GetCPhyExtMode

Function Declaration

`uint32_t CSDD_GetCPhyExtMode(`*`pD`*`, `*`extendedWrMode`*`, `*`extendedRdMode`*`);`

`CSDD_SDIO_Host *pD`

`bool *extendedWrMode`

`bool *extendedRdMode`

Parameter List

`pD` [in]

private data

`extendedWrMode` [out]

`extendedRdMode` [out]

Description

Return Value

0 on success or error code otherwise

7.94. CSDD_SetCPhySdclkAdj

Function Declaration

`uint32_t CSDD_SetCPhySdclkAdj(`*`pD`*`, `*`sdclkAdj`*`);`

```
CSDD_SDIO_Host *pD
```

```
uint8_t sdclkAdj
```

Parameter List

`pD` [in]

private data

`sdclkAdj` [in]

configuration

Description

Return Value

0 on success or error code otherwise

7.95. CSDD_GetCPhySdclkAdj

Function Declaration

`uint32_t CSDD_GetCPhySdclkAdj(`*`pD`*`, `*`sdclkAdj`*`);`

```
CSDD_SDIO_Host *pD
```

`uint8_t *sdclkAdj`

Parameter List

`pD` [in]

private data

`sdclkAdj` [out]

configuration

Description

Return Value

0 on success or error code otherwise