Sciclient Design Document

Introduction

Overview

Traditional Texas Instruments SoCs implement system control functions such as power management within operating systems on each of the processing units (ARM/DSP etc). However the traditional approach has had tremendous challenges to ensure system stability. Few of the challenges include:

- Complex interactions between Operating Systems on heterogeneous SoCs for generic features.
- Lack of centralized knowledge of system state.
- Complex implementation challenges when implementing workarounds for SoC errata.
- Equivalent SoC power or device management entitlement on all variations of Operating Systems.

DMSC controls the power management of the device, hence is responsible for bringing the device out of reset, enforce clock and reset rules. DMSC power management functions are critical to bring device to low power modes, for example DeepSleep, and sense wake-up events to bring device back online to active state. There is one instance of DMSC in this family of devices - WKUP_DMSC0.

Texas Instruments' System Control Interface defines the communication protocol between various processing entities to the System Control Entity on TI SoCs. This is a set of message formats and sequence of operations required to communicate and get system services processed from System Control entity in the SoC.

More information regarding the TI-SCI is given here .

The SCIClient is an interface to the TI-SCI protocol for RTOS and non-OS based applications. It exposes the core message details, valid module/clock IDs to the higher level software and abstracts the communication with the firmware based on the TI-SCI protocol. These APIs can be called by power, resource and security RTOS drivers or any other non-OS or RTOS based higher level software to be able to communicate with DMSC for its services. The higher level software is expected to populate the necessary message core parameters. The SCIClient would wrap the core message with the necessary protocol header and forward this to the DMSC. The SCIClient relies on the CSL-FL layer to program and interact with the Secure Proxy Threads. The SCIClient's goal is to ensure there is no duplication of the interface to the DMSC from different software components which need to interact with the DMSC or other System Control Entities in future devices.

Text Conventions

style/bullet	definition or explanation
•	This bullet indicates important information. Please read such text carefully.
•	This bullet indicates additional information.

Terms and Abbreviation

term	definition or explanation
DMSC	Device Management and Security Controller
SCI	System Control Interface
SYSFW	System Firmware

RA	Ring accelerator
PM	Power Management
RM	Resource Management

Assumptions

1. The higher level software will populate the core message payload based on the message headers which will be exposed by the SCIClient. The higher level software should include these headers and would populate the core message and send this to the SCIClient.
2. The current implementation of the SCIClient is assumed to be blocking. Until decided later the SCIClient APIs will wait for a completion response from the DMSC firmware on completion of processing before exiting. The API will allow for context switch to other tasks while it waits for the service to complete. In the non-OS case this will be a spinlock.

Constraints

The host can have multiple outstanding messages to the DMSC firmware. In order to keep track of what messages were being sent out we use the message count and an array to read back the response corresponding to the particular message count. This is especially important when interrupts are being used to understand if the message being received corresponds to the message that we sent. The array size is chosen to be a maximum of how many messages the core can possibly sent out based on the thread ID allocation from DMSC firmware. This may not be optimal for all cores for DDR less systems but is exposed through a macro which if required the user can optimize and re-build the library for. Static allocation is considered for the array hence the macro.

Design Description

The SCIClient has two major functions:

1. Interact with DMSC ROM and load the DMSC Firmware.
2. Pass on service requests from higher level software to the DMSC firmware and forward the response from DMSC firmware to the higher level software.

The Sciclient_loadfirmware API is used to cater to the first requirement and the Sciclient_service is used to cater to the second. The SCIClient library requires initialization of the a handle which is used by the subsequent API calls. This handle is allocated by the higher level software and is initialized by the [Sciclient_init] function. Once the application/higher level software is being torn down or exiting the Sciclient_deinit can be used to de-initialize this handle.

The SCIClient can operate in the following combinations:

1. Non-OS, Polling based message completion.
2. Non-OS, Interrupt Based message completion.
3. RTOS, Polling based message completion.
4. RTOS, Interrupt based message completion.

The SCIClient depends on the PDK OSAL layer to differentiate between the Non-OS and the RTOS implementation of Semaphores and Interrupts (HWIs). The build parameter of the OSAL library would determine if the application is bare metal or RTOS based. The polling versus interrupt based wait for message completion is a run time configuration passed during the SCIClient_init initialization.

All the APIs for interacting with the firmware are blocking with a specified timeout . A common API Sciclient_service is implemented for all types of calls to the firmware which takes 3 arguments :

1. **pHandle**

2. **pInPrm**
3. **pOutPrm**

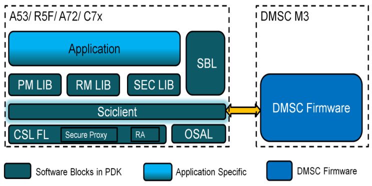
The API serves a particular request, based on the value of messageType parameter in **pInPrms**, whose response is given to the higher level API through **pOutPrms** . The **pInPrms** contains the required inputs from the higher level software corresponding to the message_type, timeout value and the core message as a byte stream. A pointer **pOutPrms** has to be passed to the sciclient ,which shall be modified by sciclient.

The Sciclient shall be responsible for abstracting all interaction with proxy and RA .

Please refer TISCI for details of message manager protocol which is used for the requests and responses.

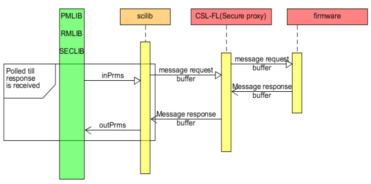
Component Interaction

Sciclient interacts with CSL-FL modules, secure proxy and RA, to interact with the DMSC firmware . Higher level libraries like PM LIB , RM LIB and SECURITY LIB will use Sciclient_service API for interaction and will be responsible for filling the core buffer of the request.



Dynamic behaviour

The high level sequence of operations and its interaction with over components of the Sciclient_service API is described in fig.2:



Key Steps of the [Sciclient_service] API are:

Construct Message

The message(header+payload) is constructed in normal memory instead of secure proxy memory .

- Find hostId . Refer section 2.2 of SYSFW for hostIds .
- Populate header flags(current support only for TI_SCI_FLAG_REQ_ACK_ON_PROCESSED) and create Message Header. For details on parameters in header, refer TISCI.
- Append payload to header.

Identify Secure Proxy threads

- Select proxy thread for Tx and Rx based on the pHandle->map . Here, pHandle is a pointer of type ([Sciclient_ServiceHandle_t] *) initialized by Sciclient_init .

Send Message

- Wait for binary semaphore pHandle->proxySem . Refer [Sciclient_ServiceHandle_t] for definition of proxySem .

- (pHandle->currSeqId ++)%[SCICLIENT_MAX_QUEUE_SIZE] . This is for differentiating different [Sciclient_service] calls.
- seqId = pHandle->currSeqId .
- Wait till queue has space till timeout (THREAD[a]_STATUS.curr_cnt > 0)
- initialCount = Rx thread message count .
- Write to Tx thread via secure proxy CSL-FL(CSL_secProxyAccessTarget) .
- Release semaphore pHandle->proxySem.

Wait for response

Sciclient waits for response when *flags* parameter in [Sciclient_ServiceInPrm_t] is TI_SCI_FLAG_REQ_ACK_ON_PROCESSED. Depending on the value of *opModeFlag* parameter in [Sciclient_ServiceHandle_t], there may be polling based or interrupt based execution . A global pointer *gSciclientHandle* = *pHandle* is also maintainded for ISR .

opModeFlag=0,Polling	**pHandle->opModeFlag=1, Interrupt based**

isMsgReceived = 0; do { Rx count = Rx thread message count if (Rx count > initialCount) { Peek into Rx thread;
 	
if(sequenceId received == sequenceId sent) { isMsgReceived=1;
 	
Read full message to pHandle->respMsgArr;
 	
}
 	
} while(!isMsgReceived)
	retVal=SemaphoreP_pend(pHandle->semSeqId[seqId],
	pInPrm->timeOut);
	ISR: {Peek for message SeqId gSciclientHandle->respMsgArr[seqId] = Read Message SemaphoreP_post(gSciclientHandle->semSeqId[SeqId]) }

Construct outPrms

- Extract response header to construct **outPrm** structure
- Copy payload from pHandle->respMsgArr[pHandle->currSeqId] to pOutPayload.

Resource Consumption

Sciclient uses the following resources:

- A global pointer *gSciclientHandle* is allocated for ISR. Refer [Sciclient_ServiceHandle_t] .
- A structure for secure proxy base addreses *gSciclient_secProxyCfg* is defined.
- The linker command file for an application using the lib must allocate a section *boardcfg_data* in OC-MSRAM for the default board configuration data .

Low Level Definitions

Constants and Enumerations

Sciclient_ServiceOperationMode

@ { Sciclient Service API Operation Mode. The different types of modes supported are:n (1) Polled Mode : no interrupts are registered. The completion of a message is via polling on the Proxy registers.n (2) Interrupt Mode : Interrupt are registered and the response message would be via a interrupt routine. Default mode in case #Sciclient_ConfigPrms_t is NULL is polled.

Definitions

#define SCICLIENT_SERVICE_OPERATION_MODE_POLLED (0U)

#define SCICLIENT_SERVICE_OPERATION_MODE_INTERRUPT (1U)

Comments None

Constraints None

See Also None

Sciclient_ServiceOperationTimeout

@ { Sciclient Service API Timeout Values. The different types are:n (1) Wait forever for an operation to complete. n (2) Do not wait for the operation to complete. n (3) Wait for a given time interface for the operation to complete.

Definitions

#define SCICLIENT_SERVICE_WAIT_FOREVER (0xFFFFFFFFU)

#define SCICLIENT_SERVICE_NO_WAIT (0x0U)

TISCI_PARAM_UNDEF

Undefined Param Undefined

Definition

#define TISCI_PARAM_UNDEF (0xFFFFFFFFU)

Comments None

Constraints None

See Also None

SCICLIENT_FIRMWARE_ABI_MAJOR

ABI Major revision - Major revision changes

- indicate backward compatibility breakage

Definition

#define SCICLIENT_FIRMWARE_ABI_MAJOR (2U)

Comments None

Constraints None

See Also None

SCICLIENT_FIRMWARE_ABI_MINOR

ABI Minor revision - Minor revision changes

- indicate backward compatibility is maintained,
- however, new messages OR extensions to existing
- messages might have been adde

Definition

#define SCICLIENT_FIRMWARE_ABI_MINOR (4U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_R5_NONSEC_0

r5 (Non Secure) : Cortex R5 Context 0 on MCU island

Definition

#define SCICLIENT_CONTEXT_R5_NONSEC_0 (0U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_R5_SEC_0

r5 (Secure) : Cortex R5 Context 1 on MCU island (Boot)

Definition

#define SCICLIENT_CONTEXT_R5_SEC_0 (1U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_R5_NONSEC_1

r5 (Non Secure) : Cortex R5 Context 2 on MCU island

Definition

#define SCICLIENT_CONTEXT_R5_NONSEC_1 (2U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_R5_SEC_1

r5 (Secure) : Cortex R5 Context 3 on MCU island

Definition

#define SCICLIENT_CONTEXT_R5_SEC_1 (3U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_A53_SEC_0

a53 (Secure) : Cortex A53 context 0 on Main island

Definition

#define SCICLIENT_CONTEXT_A53_SEC_0 (4U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_A53_SEC_1

a53 (Secure) : Cortex A53 context 1 on Main island

Definition

#define SCICLIENT_CONTEXT_A53_SEC_1 (5U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_A53_NONSEC_0

a53 (Non Secure) : Cortex A53 context 2 on Main island

Definition

#define SCICLIENT_CONTEXT_A53_NONSEC_0 (6U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_A53_NONSEC_1

a53 (Non Secure) : Cortex A53 context 3 on Main island

Definition

#define SCICLIENT_CONTEXT_A53_NONSEC_1 (7U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_A53_NONSEC_2

a53 (Non Secure) : Cortex A53 context 4 on Main island

Definition

#define SCICLIENT_CONTEXT_A53_NONSEC_2 (8U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_A53_NONSEC_3

a53 (Non Secure) : Cortex A53 context 5 on Main island

Definition

#define SCICLIENT_CONTEXT_A53_NONSEC_3 (9U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_A53_NONSEC_4

a53 (Non Secure) : Cortex A53 context 6 on Main island

Definition

#define SCICLIENT_CONTEXT_A53_NONSEC_4 (10U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_A53_NONSEC_5

a53 (Non Secure) : Cortex A53 context 7 on Main island

Definition

#define SCICLIENT_CONTEXT_A53_NONSEC_5 (11U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_GPU_NONSEC_0

gpu (Non Secure) : SGX544 Context 0 on Main island

Definition

#define SCICLIENT_CONTEXT_GPU_NONSEC_0 (12U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_GPU_NONSEC_1

gpu (Non Secure) : SGX544 Context 1 on Main island

Definition

#define SCICLIENT_CONTEXT_GPU_NONSEC_1 (13U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_ICSSG_NONSEC_0

icssg (Non Secure) : ICSS Context 0 on Main island

Definition

#define SCICLIENT_CONTEXT_ICSSG_NONSEC_0 (14U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_ICSSG_NONSEC_1

icssg (Non Secure) : ICSS Context 1 on Main island

Definition

#define SCICLIENT_CONTEXT_ICSSG_NONSEC_1 (15U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_ICSSG_NONSEC_2

icssg (Non Secure) : ICSS Context 2 on Main island

Definition

#define SCICLIENT_CONTEXT_ICSSG_NONSEC_2 (16U)

Comments None

Constraints None

See Also None

SCICLIENT_CONTEXT_MAX_NUM

Total number of possible contexts for application.

Definition

#define SCICLIENT_CONTEXT_MAX_NUM (17U)

Comments None

Constraints None

See Also None

Sciclient_PmDeviceIds

Comments None

Constraints None

See Also None

Sciclient_PmModuleClockIds

Comments None

Constraints None

See Also None

Typedefs and Data Structures

Sciclient_ConfigPrms_t

Initialization parameters for sciclient. Pointer to this is passed to #Sciclient_init.

Definition

typedef struct {

uint32_t opModeFlag; Sciclient_BoardCfgPrms_t * pBoardCfgPrms;

} Sciclient_ConfigPrms_t;

Fields

- opModeFlag : Operation mode for the Sciclient Service API. Refer to
  ref Sciclient_ServiceOperationMode for valid values.
- pBoardCfgPrms : NULL will result in using default board configuration.
  Refer #Sciclient_BoardCfgPrms_t

Comments None

Constraints None

See Also None

Sciclient_ReqPrm_t

Input parameters for #Sciclient_service function.

Definition

typedef struct {

uint16_t messageType; uint32_t flags; const uint8_t * pReqPayload; uint32_t reqPayloadSize; uint32_t timeout;

} Sciclient_ReqPrm_t;

Fields

- messageType : [IN] Type of message.
- flags : [IN] Flags for messages that are being transmitted.
  (Refer ref Tisci_ReqFlags)
- pReqPayload : [IN] Pointer to the payload to be transmitted
- reqPayloadSize : [IN] Size of the payload to be transmitted (in bytes)
- timeout : [IN] Timeout in ms for receiving response
  (Refer ref Sciclient_ServiceOperationTimeout)

Comments None

Constraints None

See Also None

Sciclient_RespPrm_t

Output parameters for #Sciclient_service function.

Definition

typedef struct {

uint32_t flags; uint8_t * pRespPayload; uint32_t respPayloadSize;

} Sciclient_RespPrm_t;

Fields

- flags : [OUT] Flags of message: Refer ref Tisci_RespFlags.
- pRespPayload : [IN] Pointer to the received payload. The pointer is an input. The

  API will populate this with the firmware response upto the size mentioned in respPayloadSize. Please ensure respPayloadSize bytes are allocated.
- respPayloadSize : [IN] Size of the response payload (in bytes)

Comments None

Constraints None

See Also None

API Definition

Sciclient_loadFirmware

Loads the DMSC firmware. This is typically called by SBL. Load firmware does not require calling the #Sciclient_init function.

Requirement: DOX_REQ_TAG (PDK-2137) , DOX_REQ_TAG (PDK-2138)

Syntax

int32_t Sciclient_loadFirmware(const uint32_t *pSciclient_firmware);

Arguments

pSciclient_firmware : [IN] Pointer to signed SYSFW binary

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_init

This API is called once for registering interrupts and creating semaphore handles to be able to talk to the firmware. The application should assume that the firmware is pre-loaded while calling the #Sciclient_init API. The firmware should have been loaded either via GEL or via the SBL prior to the application calling the #Sciclient_init. If a void pointer is passed, default values will be used, else the values passed will be used.

Requirement: DOX_REQ_TAG (PDK-2146)

Syntax

int32_t Sciclient_init(const Sciclient_ConfigPrms_t * pCfgPrms);

Arguments

pCfgPrms : [IN] Pointer to #Sciclient_ConfigPrms_t

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_service

This API allows communicating with the System firmware which can be called to perform various functions in the system. Core sciclient function for transmitting payload and recieving the response. The caller is expected to allocate memory for the input request parameter (Refer #Sciclient_ReqPrm_t) . This involves setting the message type being communicated to the firmware, the response flags, populate the payload of the message based on the inputs in the files sciclient_fmwPmMessages.h,sciclient_fmwRmMessages.h, sciclient_fmwSecMessages.h and sciclient_fmwCommonMessages.h. Since the payload in considered a stream of bytes in this API, the caller should also populate the size of this stream in reqPayloadSize. The timeout is used to determine for what amount of iterations the API would wait for their operation to complete.

To make sure the response is captured correctly the caller should also allocate the space for #Sciclient_RespPrm_t parameters. The caller should populate the pointer to the pRespPayload and the size respPayloadSize. The API would populate the response flags to indicate any firmware specific errors and also populate the memory pointed by pRespPayload till the size given in respPayloadSize.

Requirement: DOX_REQ_TAG (PDK-2142) , DOX_REQ_TAG (PDK-2141) , DOX_REQ_TAG (PDK-2140) , DOX_REQ_TAG (PDK-2139)

Syntax

int32_t Sciclient_service(const Sciclient_ReqPrm_t * pReqPrm,Sciclient_RespPrm_t * pRespPrm);

Arguments

pReqPrm : [IN] Pointer to #Sciclient_ReqPrm_t

pRespPrm : [OUT] Pointer to #Sciclient_RespPrm_t

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_deinit

De-initialization of sciclient. This de-initialization is specific to the application. It only de-initializes the semaphores, interrupts etc. which are initialized in #Sciclient_init. It does not de-initialize the system firmware.

Requirement: DOX_REQ_TAG (PDK-2146)

Syntax

int32_t Sciclient_deinit(void);

Arguments

void :

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_pmSetModuleState

Message to set the hardware block/module state This is used to request or release a device. For example: When the device is requested for operation, state is set to MSG_DEVICE_SW_STATE_ON. When the usage of the device is complete and released, the same request with state set as MSG_DEVICE_SW_STATE_AUTO_OFF is invoked. Based on exclusive access request, multiple processing entities can share a specific hardware block, however, this must be carefully used keeping the full system view in mind.

nMessage: #TISCI_MSG_SET_DEVICE nRequest: #tisci_msg_set_device_req nResponse: #tisci_msg_set_device_resp

Syntax

int32_t Sciclient_pmSetModuleState(uint32_t moduleId,uint32_t state,uint32_t additionalFlag,uint32_t timeout);

Arguments

moduleId : Module for which the state should be set. Refer ref Sciclient_PmDeviceIds.

state : Module State requested. Refer ref Sciclient_PmSetDevice.

additionalFlag : Certain flags can also be set to alter the device state. Refer ref Sciclient_PmSetDeviceMsgFlags.

timeout : Gives a sense of how long to wait for the operation. Refer ref Sciclient_ServiceOperationTimeout.

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_pmGetModuleState

Message to get the hardware block/Module state. This request does not require the processing entity to have control of the device via a set device state request.

nMessage: #TISCI_MSG_GET_DEVICE nRequest: #tisci_msg_get_device_req nResponse: #tisci_msg_get_device_resp

Syntax

int32_t Sciclient_pmGetModuleState(uint32_t moduleId,uint32_t * moduleState,uint32_t * resetState,uint32_t * contextLossState,uint32_t timeout);

Arguments

moduleId : Module for which the state should be set. Refer ref Sciclient_PmDeviceIds.

moduleState : Module State returned. Refer ref Sciclient_PmGetDeviceMsgResp.

resetState : Programmed state of the reset lines.

contextLossState : Indicates how many times the device has lost context. A driver can use this monotonic counter to determine if the device has lost context since the last time this message was exchanged.

timeout : Gives a sense of how long to wait for the operation. Refer ref Sciclient_ServiceOperationTimeout.

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_pmSetModuleRst

Set the device reset state. This is used to set or release various resets of the hardware block/module

nMessage: #TISCI_MSG_SET_DEVICE_RESETS nRequest: #tisci_msg_set_device_resets_req nResponse: #tisci_msg_set_device_resets_resp

Syntax

int32_t Sciclient_pmSetModuleRst(uint32_t moduleId,uint32_t resetBit,uint32_t timeout);

Arguments

moduleId : Module for which the state should be set. Refer ref Sciclient_PmDeviceIds.

resetBit : Module Reset Bit to be set. TODO: Get reset IDs. Refer ref Sciclient_PmGetDeviceMsgResp. 1 - Assert the reset 0 - Deassert the reset Note this convention is opposite of PSC MDCTL

timeout : Gives a sense of how long to wait for the operation. Refer ref Sciclient_ServiceOperationTimeout.

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_pmModuleClkRequest

Message to set the clock state: This requests for finer control of hardware device's clocks. This allows for configuration for hardware blocks that require customization of the specific input clocks. NOTE: each of the clock IDs are relative to the hardware block.

nMessage: #TISCI_MSG_SET_CLOCK nRequest: #tisci_msg_set_clock_req nResponse: #tisci_msg_set_clock_resp

Syntax

int32_t Sciclient_pmModuleClkRequest(uint32_t moduleId,uint32_t clockId,uint32_t state,uint32_t additionalFlag,uint32_t timeout);

Arguments

moduleId : Module for which the state should be set. Refer ref Sciclient_PmDeviceIds.

clockId : Clock Id for the module. Refer ref Sciclient_PmModuleClockIds.

state : Clock State requested. Refer ref Sciclient_PmSetClockMsgState.

additionalFlag : Certain flags can also be set to alter the clock state. Refer ref Sciclient_PmSetClockMsgFlag.

timeout : Gives a sense of how long to wait for the operation. Refer ref Sciclient_ServiceOperationTimeout.

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_pmModuleGetClkStatus

Message to get the clock state to or from a hardware block

nMessage: #TISCI_MSG_GET_CLOCK nRequest: #tisci_msg_get_clock_req nResponse: #tisci_msg_get_clock_resp

Syntax

int32_t Sciclient_pmModuleGetClkStatus(uint32_t moduleId,uint32_t clockId,uint32_t * state,uint32_t timeout);

Arguments

moduleId : Module for which the state should be set. Refer ref Sciclient_PmDeviceIds.

clockId : Clock Id for the module. Refer ref Sciclient_PmModuleClockIds.

state : Clock State returned. Refer ref Sciclient_PmGetClockMsgState.

timeout : Gives a sense of how long to wait for the operation. Refer ref Sciclient_ServiceOperationTimeout.

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_pmSetModuleClkParent

Message to Set Clock Parent: This message allows SoC specific customization for setting up a specific clock parent ID for the various clock input options for a hardware block's clock. This is rarely used customization that may be required based on the usecase of the system where the reset input clock option may not suffice for the usecase attempted.

nMessage: #TISCI_MSG_SET_CLOCK_PARENT nRequest: #tisci_msg_set_clock_parent_req nResponse: #tisci_msg_set_clock_parent_resp

Syntax

int32_t Sciclient_pmSetModuleClkParent(uint32_t moduleId,uint32_t clockId,uint32_t parent,uint32_t timeout);

Arguments

moduleId : Module for which the state should be set. Refer ref Sciclient_PmDeviceIds.

clockId : Clock Id for the module. Refer ref Sciclient_PmModuleClockIds.

parent : Parent Id for the clock. TODO: Find what this is.

timeout : Gives a sense of how long to wait for the operation. Refer ref Sciclient_ServiceOperationTimeout.

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_pmGetModuleClkParent

Message to Get Clock Parent: Query the clock parent currently configured for a specific clock source of a hardware block This is typically used to confirm the current clock parent to ensure that the requisite usecase for the hardware block can be satisfied.

nMessage: #TISCI_MSG_GET_CLOCK_PARENT nRequest: #tisci_msg_get_clock_parent_req nResponse: #tisci_msg_get_clock_parent_resp

Syntax

int32_t Sciclient_pmGetModuleClkParent(uint32_t moduleId,uint32_t clockId,uint32_t * parent,uint32_t timeout);

Arguments

moduleId : Module for which the state should be set. Refer ref Sciclient_PmDeviceIds.

clockId : Clock Id for the module. Refer ref Sciclient_PmModuleClockIds.

parent : Returned Parent Id for the clock. TODO: Find what this is.

timeout : Gives a sense of how long to wait for the operation. Refer ref Sciclient_ServiceOperationTimeout.

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_pmGetModuleClkNumParent

Message to get the number of clock parents for a given module. This is typically used to get the max number of clock parent options available for a specific hardware block's clock.

nMessage: #TISCI_MSG_GET_NUM_CLOCK_PARENTS nRequest: #tisci_msg_get_num_clock_parents_req nResponse: #tisci_msg_get_num_clock_parents_resp

Syntax

int32_t Sciclient_pmGetModuleClkNumParent(uint32_t moduleId,uint32_t clockId,uint32_t * numParent,uint32_t timeout);

Arguments

moduleId : Module for which the state should be set. Refer ref Sciclient_PmDeviceIds.

clockId : Clock Id for the module. Refer ref Sciclient_PmModuleClockIds.

numParent : Returned number of parents.

timeout : Gives a sense of how long to wait for the operation. Refer ref Sciclient_ServiceOperationTimeout.

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_pmSetModuleClkFreq

Message to set the clock frequency. This is typically desired when the default frequency of the hardware block's clock is not appropriate for the usecase desired. NOTE: Normally clock frequency management is automatically done by TISCI entity. In case of specific requests, TISCI evaluates capability to achieve requested range and responds with success/failure message. This sets the desired frequency for a clock within an allowable range. This message will fail on an enabled clock unless MSG_FLAG_CLOCK_ALLOW_FREQ_CHANGE is set for the clock. Additionally, if other clocks have their frequency modified due to this message, they also must have the MSG_FLAG_CLOCK_ALLOW_FREQ_CHANGE or be disabled.

nMessage: #TISCI_MSG_SET_FREQ nRequest: #tisci_msg_set_freq_req nResponse: #tisci_msg_set_freq_resp

Syntax

int32_t Sciclient_pmSetModuleClkFreq(uint32_t moduleId,uint32_t clockId,uint64_t freqHz,uint32_t additionalFlag,uint32_t timeout);

Arguments

moduleId : Module for which the state should be set. Refer ref Sciclient_PmDeviceIds.

clockId : Clock Id for the module. Refer ref Sciclient_PmModuleClockIds.

freqHz : Frequency of the clock in Hertz.

additionalFlag : Additional flags for the request .Refer ref Tisci_PmSetClockMsgFlag .

timeout : Gives a sense of how long to wait for the operation. Refer ref Sciclient_ServiceOperationTimeout.

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_pmQueryModuleClkFreq

Message to query the best clock frequency in the given range. This message does no real operation, instead, it requests the system control entity to respond with the best frequency that can match a frequency range provided. NOTE: This is a snapshot view. In a multi processing system, it is very well possible that another processing entity might change the configuration after one entity has queried for best match capability. Only a SET_CLOCK_FREQ will guarantee the frequency is configured.

nMessage: #TISCI_MSG_QUERY_FREQ nRequest: #tisci_msg_query_freq_req nResponse: #tisci_msg_query_freq_resp

Syntax

int32_t Sciclient_pmQueryModuleClkFreq(uint32_t moduleId,uint32_t clockId,uint64_t freqHz,uint32_t timeout);

Arguments

moduleId : Module for which the state should be set. Refer ref Sciclient_PmDeviceIds.

clockId : Clock Id for the module. Refer ref Sciclient_PmModuleClockIds.

freqHz : Frequency of the clock in Hertz.

timeout : Gives a sense of how long to wait for the operation. Refer ref Sciclient_ServiceOperationTimeout.

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_pmGetModuleClkFreq

Message to Get Clock Frequency This is most used functionality and is meant for usage when the driver controlling the hardware block requires to know the input clock frequency for configuring internal dividers / multipliers as required.

nMessage: #TISCI_MSG_GET_FREQ nRequest: #tisci_msg_get_freq_req nResponse: #tisci_msg_get_freq_resp

Syntax

int32_t Sciclient_pmGetModuleClkFreq(uint32_t moduleId,uint32_t clockId,uint64_t * freqHz,uint32_t timeout);

Arguments

moduleId : Module for which the state should be set. Refer ref Sciclient_PmDeviceIds.

clockId : Clock Id for the module. Refer ref Sciclient_PmModuleClockIds.

freqHz : Frequency of the clock returned in Hertz.

timeout : Gives a sense of how long to wait for the operation. Refer ref Sciclient_ServiceOperationTimeout.

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_pmEnableWdt

Enables the WDT controllers within the DMSC.

nMessage: #TISCI_MSG_ENABLE_WDT nRequest: #tisci_msg_enable_wdt_req nResponse: #tisci_msg_enable_wdt_resp

Syntax

int32_t Sciclient_pmEnableWdt(uint32_t timeout);

Arguments

timeout : Gives a sense of how long to wait for the operation. Refer ref Sciclient_ServiceOperationTimeout.

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_pmDisableWakeup

This message is part of the CPU Off sequence. The sequence is: - Mask interrupts - Send wake reset message to PMMC - Wait for wake reset ACK - Abort if any interrupts are pending - Disable all interrupts - Send goodbye to PMMC - Wait for goodbye ACK - Execute WFI

nMessage: #TISCI_MSG_WAKE_RESET nRequest: #tisci_msg_wake_reset_req nResponse: #tisci_msg_wake_reset_resp

Syntax

int32_t Sciclient_pmDisableWakeup(uint32_t timeout);

Arguments

timeout : Gives a sense of how long to wait for the operation. Refer ref Sciclient_ServiceOperationTimeout.

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_pmGetWakeupReason

Request wakeup reason After a wakeup, the host can request the deepest sleep/idle mode reached and the reason for the wakeup. The API also returns the time spent in idle state.

nMessage: #TISCI_MSG_WAKE_REASON nRequest: #tisci_msg_wake_reason_req nResponse: #tisci_msg_wake_reason_resp

Syntax

int32_t Sciclient_pmGetWakeupReason(uint8_t mode[32],uint8_t reason[32],uint32_t * time_ms,uint32_t timeout);

Arguments

mode[32] : Deepest sleep/idle mode 0x000C reached (ASCII)

reason[32] : Wakeup reason (ASCII)

time_ms : Time spent in idle state (ms)

timeout : Gives a sense of how long to wait for the operation. Refer ref Sciclient_ServiceOperationTimeout.

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_pmDevicePowerOff

Some processors have a special sequence for powering off the core that provides notification to the PMMC when that sequence has completed. For processors without such a sequence, the goodbye message exists. The exact sequence involved in the goodbye message depends on the SoC.

nMessage: #TISCI_MSG_GOODBYE nRequest: #tisci_msg_goodbye_req nResponse: #tisci_msg_goodbye_resp

Syntax

int32_t Sciclient_pmDevicePowerOff(uint32_t timeout);

Arguments

timeout : Gives a sense of how long to wait for the operation. Refer ref Sciclient_ServiceOperationTimeout.

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_pmDeviceReset

Objective: Trigger a SoC level reset Usage: Used to trigger a system level reset. NOTE: Depending on permissions configured for the SoC, not all processing entities may be permitted to request a SoC reset. When permitted, the request once processed will not return back to caller.

nMessage: #TISCI_MSG_SYS_RESET nRequest: #tisci_msg_sys_reset_req nResponse: #tisci_msg_sys_reset_resp

Syntax

int32_t Sciclient_pmDeviceReset(uint32_t timeout);

Arguments

timeout : Gives a sense of how long to wait for the operation. Refer ref Sciclient_ServiceOperationTimeout.

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Sciclient_pmDomainReset

Trigger warm reset of a domain NOTE: Depending on permissions configured for the SoC, not all domains can be reset

nMessage: #TISCI_MSG_SYS_RESET nRequest: #tisci_msg_sys_reset_req nResponse: #tisci_msg_sys_reset_resp

Syntax

int32_t Sciclient_pmDomainReset(domgrp_t domGrp, uint32_t timeout);

Arguments

domGrp : The domain to be reset.

timeout : Gives a sense of how long to wait for the operation. Refer ref Sciclient_ServiceOperationTimeout.

Return Value CSL_PASS on success, else failure

Comments If domGrp = 0, this API functions like Sciclient_pmDeviceReset.

Constraints None

See Also None

Sciclient_pmIsModuleValid

This API would check if the given module Id is valid for the device. The module Id that is referred to is ref Sciclient_PmDeviceIds.

Syntax

int32_t Sciclient_pmIsModuleValid(uint32_t modId);

Arguments

modId : Module Index to be checked.

Return Value CSL_PASS on success, else failure

Comments None

Constraints None

See Also None

Design Analysis and Resolution(DAR)

DAR Criteria 1

SCIClient should support OS and non OS applications. However, SCIClient in an OS context should be able to prevent mutliple threads from writing to the secure proxy.

Available Alternatives

Alternative 1

Built the library separately for OS and non OS. Each API will have its own OS and Non-OS implementation.

Alternative 2

Use PDK OSAL and have the OSAL support for non-OS and OS implementation. The SCIClient APIs will have one implementation. The application should link OSAL as well.

Decision

Use Alternative 2 as the OSAL already has OS and non-OS implementation. The SCIClient need not duplicate code. Consistent with other libraries as well.

DAR Criteria 2

SCIClient constructing the SCI core message based on fields given as input by the higher level software.

Available Alternatives

Alternative 1

SCIClient maintains separate APIs for all the message types possible and the PM/RM/Security Libs pass parameters to this. Eg. So if a SCI message looks like

```
struct msg_rm_alloc_foo {
        struct message_hdr hdr;
        s32 param_a;
        u16 param_b;
        u16 param_c;
} __attribute__((__packed__));
```

The SCILIB API looks something like

```
int32_t sci_client_rm_alloc_foo(int32_t param_a, uint16_t param_b, uint16_t param_c);
```

SCILIB abstracts the packed message format and the header details.

Alternative 2

SCI message structure is instead of

```
struct msg_rm_alloc_foo {
        struct message_hdr hdr;
        s32 param_a;
        u16 param_b;
        u16 param_c;
} __attribute__((__packed__));

int32_t sci_client_rm_alloc_foo(int32_t param_a, uint16_t param_b, uint16_t param_c);
```

Do the following

```
struct msg_rm_alloc_foo_body {
        s32 param_a;
        u16 param_b;
        u16 param_c;
} __attribute__((__packed__));

int32_t sci_client_rm_alloc_foo(const struct msg_rm_alloc_foo_body * body);
```

Just to extend that a bit further:

```
int32_t sci_client_rm_alloc_foo(const struct msg_rm_alloc_foo_body * body);
```

becomes a common API

```
int32_t sci_client_send_message(uint32 message_type, const struct void * body);
```

Apart from message_type we have some more inputs which we finally capture in an inPrm structure.

Decision

Use alternate 2 to not have multiple APIs in the SCIClient HAL and the SCIClient HAL will expose the core message structure for the higher level software to work on . Above this, SCIClient FL will have seperate API definitions for each type of supported message separately for PM, RM and Security .

Document revision history

Version #	Date	Author Name	Revision History	Status
01.00	2-Jan-2018	SACHIN PUROHIT	**Added design info for** sciclient.h	Draft
01.01	6-Dec-2022	KUNAL LAHOTI	Removing internal Links from sciclient_designdoc	Draft