TMS320C672x
Chip Support Libraries (CSL) v3.x
API Reference Guide

July 2007

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.
TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement

specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related

requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
Low Power Wireless	www.ti.com/lpw	Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address:
Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

Preface

Read This First

About This Manual

The API reference guide serves as a software programmer's handbook for working with the TMS 320c672x/DA7xx CSL.

This product supports the following devices: C6722, C6726, C6727, DA705, DA707, DA708, DA710.

Not all APIs and modules in this guide are supported by all devices in the family. Please check the specific product data sheet to determine which peripherals and features are supported by your device.

Abbreviations

Table of Abbreviations

Abbreviation	Description
API	Application Programming Interface
CSL	Chip Support Library
DMAX	Data Movement Accelerator
EMIF	External Memory Interface
I2C	Inter Integrated Circuit
ICACHE	Instruction Cache
INTC	Interrupt Controller
McASP	Multi channel Audio Serial Port
PLLC	Phase Lock Loop Controller
RTI	Real Time Interrupt
SPI	Serial Port Interface
UHPI	Universal Host Port Interface

TABLE OF CONTENTS

LIST OF TABLE

LIST OF FIGURES

Chapter 1
Introduction

Topics

1.1 Introduction

The Chip Support Layer constitutes a set of well-defined API that abstracts low-level details of the underlying SoC device so that user can configure, control (start/stop etc.) and have read/write access to peripherals without having to worry about register bit-field details.

The CSL services are implemented as distinct modules that correspond with the underlying SoC device modules themselves. By design, CSL APIs follow a consistent style, uniformly across Processor Instruction Set Architecture and are independent of OS. This helps in improving portability of code written using CSL.

1.2 Overview

CSL is realized as twin-layer – a basic register-layer and a more abstracted functional-layer. The lower register layer comprises of a very basic set of macros and type definitions. The upper functional layer comprises of "C" functions that provide an increased degree of abstraction, but intended to provide "directed" control of underlying hardware.

It is important to note that CSL does not manage data-movement over underlying h/w devices. Such functionality is considered a prerogative of a device-driver and serious effort is made to not blur the boundary between device-driver and CSL services in this regard.

CSL does not model the device state machine. However, should there exist a mandatory (hardware dictated) sequence (possibly atomically executed) of register reads/writes to setup the device in chosen "operating modes" as per the device datasheet, then CSL does indeed support services for such operations.

The CSL services are decomposed into modules, each following the twin-layer of abstraction described above. The APIs of each such module are completely orthogonal (one module's API does not internally call API of another module) and do not allocate memory dynamically from within. This is key to keeping CSL scalable to fit the specific usage scenarios and ease the effort to ROM a CSL based application.

1.3 CSL Interface

CSL is organized into modules by peripheral. Each module contains a twin-layer user interface, the register layer and the functional layer.

The register layer header file for a peripheral <module> is provided in a header file called cslr_<module>.h The functional layer header file for a given peripheral <module> is provided in a header file called csl_<module>.h
In addition to modules for individual peripherals, CSL provides some chip-level modules that perform system and device-level services. These modules are described in the table below.

Table 1: Chip Level Modules

Module	Description
CSL	Includes basic system management and utility functions for register configuration.
CHIP	Contains the generic device-specific information that is not specific to a peripheral or module. It includes the chip register IDs, field definitions, register read and write functions,
VERSION	Provides for version management, such as chip ID and version ID.
INTC	The interrupt module provides interrupt management services and a dispatcher. This module is delivered as a separate library.

These modules follow the same naming convention for header files.

1.4 Functional Layer

The CSL Functional Layer is provided as the recommended application programmer's interface to the peripheral. To take advantage of hardware abstraction and maintain maximum forward compatibility in the future, users are encouraged to make use of the Functional Layer APIs in their application and driver code.

CSL 3.x supports a core set of Functional Layer APIs across all modules.

Table 2 General format for Functional layer APIs

API	Description
CSL_<mod>Init()	Peripheral initialization function. Optional; does not affect hardware.
CSL_<mod>Open()	Returns a handle to the peripheral instance
CSL_<mod>Close()	Releases handle to peripheral instance
CSL_<mod>HwSetup()	Configures all peripheral registers with values passed in CSL_<mod>HwSetup structure
CSL_<mod>GetHwStatus()	Queries the current peripheral configuration with CSL_<mod>GetHwSetup structure.
CSL_<mod>HwControl()	Performs modification of one or more parameters according to passed command parameter.
CSL_<mod<HwSetupRaw()	Initializes the device registers with the register-values passed in the Config Data structure.
CSL_<mod>Read()	Data read (I/O peripherals)
CSL_<mod>Write()	Data write (I/O peripherals)

In addition, there may be unique APIs implemented for a peripheral to perform specific operations, as needed. For example "CSL_dmaxGetNextFreeParamEntry" API of DMAX module searches for the next free parameter entry in resource table.

Interface functions exported by this layer are "run to completion", meaning, they shall not support asynchronous behavior or deferred completion. If the peripheral hardware has ability to initiate a transaction and assert its completion at a later point in time via designated CPU interrupt, the

same should be accommodated by higher level software (typically device drivers). In general, CSL APIs do not perform resource management or memory allocation; this is managed by the application code or device drivers.

1.4.1 CSL Basic Data Types

The following basic data types are defined in CSL

Table 3: CSL basic Data types

Type	Defined as.
Bool	**Unsigned short**
Int	**int**
Char	**char**
String	**char ***
Ptr	**void ***
Uint32	**unsigned int**
Uint16	**unsigned short**
Uint8	**unsigned char**
Int32	**int**
Int16	**short**
Int8	**char**
CSL_BitMask16	**Uint16**
CSL_BitMask32	**Uint32**
CSL_Reg16	**volatile Uint16**
CSL_Reg32	**volatile Uint32**
CSL_Status	**Int16**

1.4.2 Functional Layer Naming Conventions

The CSL reserved names fall into two categories, those that are ***declared*** (ex: Functions, variables and so on) and those that are ***symbolic constants*** and ***macros*** that are implemented via enum or #defines. The declarative names should strictly be avoided from redefining by user. The #defines however, are open for redefinition via the standard C supported #undef construct. Regardless, user is encouraged not to redefine/conflict with CSL Namespace, as side effects are hard to predict.

The following table illustrates the CSL naming conventions:

Table 4: CSL naming conventions

Format	Namespace	Type
CSL_<MODULE>_<STRING>	Symbolic constant specified as either a #define or an enum. The entire name must be in upper case the <MODULE> denotes peripheral module name and the <STRING> denotes any name, representative of the item being specified or defined. The <STRING> part	**CSL_INTC_EVENTID_CNT** Here **INTC** is <MODULE>

	can have one or more underscores embedded for improved readability.	
CSL_<**P**eri**T**itle**C**ase**N**ame>	Peripheral module data type. The CSL_ prefix will be in upper case. The module name string is capitalized and follows title case convention without any underscores. Upper case is used to denote start of a new word or phrase.	**CSL_T**imer**O**bj **CSL_I**ntc**E**vent**I**d **CSL_I**2c**H**w**S**etup **CSL_U**art**H**andle
CSL<**P**eri**T**itle**C**ase**N**ame>	Peripheral module's private data type. The leading underscore denotes that this is not a part of published API. The naming style is similar to the peripheral module's data type.	**_CSL_I**ntc2**D**ispatch**T**able
CSL_<**p**eri**A**rbitrary**N**ame>	Peripheral module's published API function. Naming style is similar to that of module data type discussed above with the exception that first character following the CSL_ prefix is in lower case. The same style is also applicable to global data variable. The context of occurrence is key to telling one from the other. Example, CSL_sysDataObj is a global data variable, not a function.	**CSL_u**art**H**w**C**ontrol(…)
CSL<**p**eri**A**rbitrary**N**ame>	Peripheral module's private function. The leading underscore denotes that this is not a part of published API. The naming style is similar to the peripheral module's private data type. The same style is also applicable to private data variable. The context of occurrence is key to telling one from the other.	**_CSL_t**imer**G**et**A**ttrs(…)

1.4.3 Symbolic Constants

This section documents the symbolic (#define) constants that constitute part of published CSL APIs. The table only lists the common symbols that are applicable to all peripheral modules. However, there exists a whole host of symbolic constants that are very specific to each particular module and are **not** listed here.

Table 5: Symbolic constants naming conventions

Name	Description
CSL_<MODULE>_<n>_REGS	Base address of hardware registers for instance <n> of said peripheral module. This is used only for modules that do not support multiple channels or resources. Ex: CSL_TIMER_1_REGS for Timer
CSL_<MODULE>_CHA<m>_REGS	Base address of hardware registers for channel <m> of peripheral, for modules that do support multiple channels or resources.

1.4.4 Error Codes

The CSL3.x will extend minimal support for error handling. Essentially, CSL will only report success or failure of APIs via their return types and/or separate status parameter passed to the call itself.

The error codes are 16bit signed binary numbers that allows us to represent 32 K unique errors. The entire space is divided into 1024 groups, each of size 32. The first group is reserved for CSL generic system errors, the second through last are distributed amongst individual CSL modules. A positive number is regarded as OK status and/or successful operation of a CSL API. All error states are represented as negative integers only.

The following table documents the base set of CSL error codes, **not** specific to any given peripheral.

Table 6: Common error codes

Error Code	Number	Description
CSL_SOK	+1	Success
CSL_ESYS_FAIL	-1	Generic failure
CSL_ESYS_INUSE	-2	Peripheral resource is already in use
CSL_ESYS_XIO	-3	Encountered a shared I/O (XIO) pin conflict
CSL_ESYS_OVFL	-4	Encountered CSL system resource overflow
CSL_ESYS_BADHANDLE	-5	Handle passed to CSL was invalid
CSL_ESYS_INVPARAMS	-6	Invalid parameters.
CSL_ESYS_INVCMD	-7	Command passed to the CSL was invalid.
CSL_ESYS_INVQUERY	-8	Query passed to the CSL was invalid.
CSL_ESYS_NOTSUPPORTED	-9	Action not supported by CSL.
CSL_ESYS_ALREADY_INITIALIZED	-10	Module already initialized

1.5 Register Layer

1.5.1 Register Layer Naming Conventions

All names are alphanumeric except for use of underscores as delimiters.

Table 7: Register layer naming conventions

Convention	Description
CSL Module Identifiers	CSL_<MOD>_ID, where <MOD> is name of the CSL module for a specific peripheral Ex: CSL_TIMER_ID, CSL_MCBSP_ID
Peripheral Instance Identifiers	CSL_<MOD>_<*NUM*>, where <NUM> is Instance number as per Device Data Sheet or Peripheral Reference Guide Ex: CSL_TIMER_1, CSL_MCBSP_1, CSL_I2C_1 For CSL modules, which have single instance, the convention followed is CSL_<MOD> Ex: CSL_DMA, CSL_SSW
Peripheral Instance Count Identifiers	CSL_<MOD>_CNT, where <MOD> is peripheral module whose number of instances is defined Ex: CSL_EMIFS_CNT, CSL_CAN_CNT
Peripheral Register Identifiers	<MOD>_<REG>, where <REG> is register name. Names used with specific peripheral instance overlays. Ex: TIMER_CNTL, CPMAC_TX_CONTROL, CPMAC_RX_CONTROL
Peripheral Register Continuous Bit-field Identifiers	<MOD>_<REG>_<FIELD>, where <FIELD> is bit-field name. Names are instance-dependent. Ex: TIMER_CNTL_CLKEN, CPMAC_TX_CONTROL_EN
Peripheral Register Bit-field Symbols	<MOD>_<REG>_<FIELD>_<SYM>, where <SYM> is bit-field setting symbolic token Names are instance-dependent. Ex: CPMAC_TX_CONTROL_EN_RESETVAL, TIMER_CNTL_CLKEN_SHIFT

1.5.2 Register Overlay Structure

SoC peripherals are typically programmed by reading/writing to one or more registers in the peripheral's IO address space. In order to allow for clean and intuitive access to all the registers belonging to a given peripheral instance, CSL implements a technique called Register Overlay Structure. A C data structure template is defined with structure members corresponding to each of the registers of the peripheral device in the order in which they occur. The member types are chosen to correspond to the widths of the register they represent. Appropriate padding is introduced to ensure alignment for proper addressing of these registers from "C". The structure members use names that correspond to those used in the peripheral datasheet, to ease programming. Since there exists a well-formed C structure, the registers can be viewed in IDE watch windows and presumably recognized by smart-editors that can do auto-completion while typing.

It should be noted that register overlays do not consume memory, as they are not instantiated. The purpose of these structures is mainly to typify the "C" pointers

Example:

The figure below shows the layout of a TIMER peripheral device. Assuming there exist two instances of the device, one at address 0x01940000 and the other at address 0x01944000, the register overlay for such a device is specified as follows:

```
typedef struct {
  Uint32 CTL; /* Timer Control Register */
  Uint32 PRD; /* Timer Period Register */
  Uint32 CNT; /* Timer Counter Register */
} CSL_TimerRegs;
typedef volatile CSL_TimerReg *CSL_TimerRegsOvly
```

CSL_TIMER_0_REGS	**CTL [+0x00]**
(0x01940000)	**PRD [+0x04]**
	CNT [+0x08]
CSL_TIMER_1_REGS	**CTL [+0x00]**
(0x01944000)	**PRD [+0x04]**
	CNT [+0x08]

Figure 1: Register Layer overlay structure

1.5.3 Register Layer Symbolic Constants

The CSL register layer file for a given peripheral device (cslr_<module>.h) will define certain standard symbols for each peripheral register/bit field. These symbolic constants are declared with the following convention:

Notational convention: CSL_<MODULE>_<REG>_<FIELD>_<SYMBOL>

The semantics of the various parts of the symbolic name is shown in table below:

Table 8: Symbolic names used in Register layer

Convention	Description
<**MODULE**>	The CSL Peripheral module Ex: **TIMER**
<**REG**>	The peripheral device register Ex: **CNT**
<**FIELD**>	Bit-field of interest Ex: **ST**
<**SYMBOL**>	Operational symbol for constant being defined Ex: **STOP, START**

Ex: CSL_TIMER_CNT_ST_STOP

The table below summarizes the standard symbols used with register bit fields:

Table 9: Standard Symbols used with register bit fields

#define Symbolic Constant	Semantics of the Value Assigned
CSL_<MODULE>_<REG>_SHIFT	The number of left shift positions to reach the register bit-field of interest
CSL_<MODULE>_<REG>_MASK	The binary *and* mask useful to extract register bit-field of interest
CSL_<MODULE>_<REG>_RESETVAL	The power-on reset value assumed by the register or bit-field of interest

NOTE: The above defines specified in **cslr_<module>.h** have math bit ordering of MSB: LSB and are regardless of what Endian-flips occur as these are read over processor memory busses. Typically, the processor hardware wiring will be such that CPU always gets to read/write its memory mapped peripherals in "Native Endian" format i.e., ready for CPU interpretation. Should there be Endian mismatch between CPU and memory-mapped peripheral, then necessary corrections (Swaps) must be handled outside, before applying the **_MASK**, **_SHIFT** etc., symbols shown in this file.

1.5.4 Register Layer Macros

Table 10: Register Bit Field Manipulation Macro services

Service	Description
CSL_FMK(field, val)	Creates an AND mask of value (val) moved to specified field location.
CSL_FMKT (field, token)	Same as CSL_FMK, but allows predefined symbolic tokens to be used as value.
CSL_FMKR (msb, lsb, val)	Same as CSL_FMK, but allows raw bit positions (msb:lsb) to specify bit-field.
CSL_FEXT(reg, field)	Evaluates the arithmetic value of bits gathered from specified field.
CSL_FEXTR(reg, msb, lsb)	Same as CSL_FEXT, but allows raw bit positions (msb:lsb) to specify bit-field.
CSL_FINS(reg, field, val)	Inserts the specified value (val) at the specified field in the register.
CSL_FINST(reg, field, token)	Same as CSL_FINS, but allows predefined symbolic tokens to be used as value.
CSL_FINSR(reg, msb, lsb, val)	Same as CSL_FINS, but allows raw bit positions (msb:lsb) to specify bit-field.

1.6 C++ Compatibility

CSL Functional Layer APIs are, for the most part, implemented in C, with small parts implemented in native assembly to work around some difficulties of realizing the same in C. Regardless, the APIs are declared appropriately so as to allow C++ applications to call them.

Unlike C++ functions, the CSL APIs will not support specification of default values for formal arguments passed to them.

Also, in places where CSL API semantics require the user to specify function pointers, CSL3.x design does not allow the user to input a C++ function pointer. To work around above limitation, a wrapper function in C, encapsulating the C++ member function needs to be written by the user. This function can be designed to input the class instance as an argument (along with any other parameters that it requires), and invoke the appropriate class member function internally for achieving the desired objectives.

1.7 INTC Software Architecture

The INTC module in CSL3.x is designed to provide an abstraction for all the basic interrupt controller functions, such as enabling and disabling interrupts, specifying the user function to be called in response to interrupts, and setting desired hardware properties. These functions are not specific to any given peripheral. In other words, the INTC module abstracts the generic interrupt capabilities supported by the processor.

This section describes the CSL INTC functionality for C672x.

1.7.1 The Interrupt Controller

The figure below shows a multi-level interrupt controller, within the dashed line boundary. The Level-0 controller is considered part of the CPU itself. This level implements the primary Interrupt Vector Table for the processor. The remaining controllers help expand these vectors to handle many more hardware events. These events, shown as arrows, might be externally triggered via device input pins and/or internally asserted by on-chip peripheral devices. The peripheral devices themselves, represented by cross-hatched boxes, might host an event controller (checkered box) that helps map the hardware events to outgoing lines that assert the CPU interrupts. Each of the interrupt controllers would have programmable control registers to enable/disable the hardware events from proceeding towards the CPU Interrupts. The controllers also allow the user to configure the interrupt capture circuitry to a specified polarity, edge sensitivity, priority, etc. The Level-0 interrupt controller is shown as having a “selector” capability that maps a given CPU interrupt vector to one-of-N input hardware events. This scheme, although available at Level-0 in today’s TI processors, is technically possible to be also present at other levels. It is also possible for an interrupt controller at a given level to be comprised of more than one logical block. (See 2.0, 2.1 blocks in the figure.) All of the blocks that comprise the interrupt controllers Level-0 through level-N are part of the INTC module (bounded by dashed line) in CSL3.x. Only the top level INTC abstraction will be seen by the user. The internal INTC0 through INTCn are hidden from user. The wiring between individual INTC sub-blocks shall be done during INTC module initialization and dispatcher setup as detailed in ensuing sub-sections of this document. It is important to note that any “custom controllers” (refer to checkered box in figure) embedded in specific peripheral devices (ex: C64x EDMA Controller) are not considered part of INTC functionality.

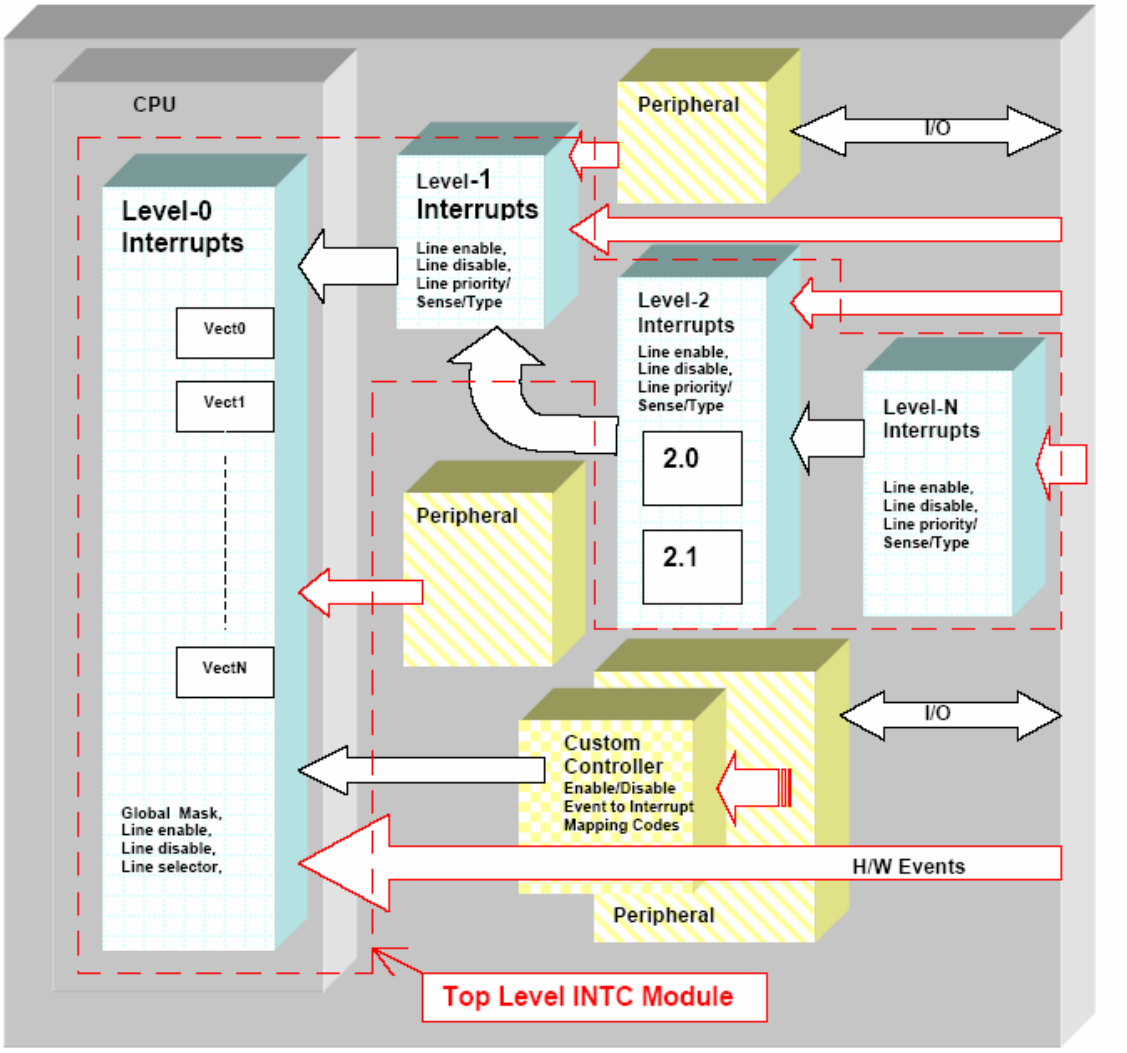

Figure 2: INTC Controller block diagram

1.7.2 INTC Module Initialization

The INTC module maintains an array of bit masks that enable INTC to keep track of interrupts that are active or in-use by the application. Each bit position corresponds to a single hardware event that can be processed by the INTC. The total bit positions maintained corresponds to the maximum number of hardware events that the INTC can recognize and handle at any given time. At level-0, this corresponds to the CPU primary interrupt vectors; at other levels, it corresponds to the capacity of fan-in and priority resolution implemented in the controllers. The INTC module will assign unique IDs to each hardware event and has knowledge of the range of such IDs applicable at each level (ie., INTC0 thru' INTCn).

During the CSL initialization phase, the INTC module initialization function CSL_intcInit() is called. This will reset the global variable CSL_IntcContext.eventAllocMask[] to zeros. This implies that all the interrupts are available for use by the application. This array stands responsible for resolving any conflicts on the same interrupt resource. Only one interrupt service routine can be plugged in for each interrupt. Thus when the same interrupt line is shared between different events,

synchronization of the events has to be taken care in the application program and the event that caused the interrupt has to be identified in the ISR to get the proper function executed.

1.7.3 Interrupt Dispatcher Specifics

Following successful initialization of INTC module (via CSL_intcInit()), the user can choose to initialize the INTC built-in dispatcher by calling CSL_intcDispatcherInit(). The dispatcher record argument passed for CSL_intcDispatcherInit()) is used as a record of which ISR is hooked to a particular CPU interrupt at a particular point in time. Once the dispatcher record is created and initialized, CSL_plugEventHandler() will internally perform recording the ISR in the dispatcher record and hooking up the appropriate primary ISRs in the interrupt vector table.

Typically, when an operating system (OS) is running, the primary interrupt vector table is under control of the OS Scheduler. The OS Scheduler will hook its own dispatcher function at this level-0. OS ports can choose to either do away completely with CSL dispatchers or implement their own for the desired levels. They can choose to first initialize the CSL interrupt dispatcher and then swap-in their own interrupt handlers at desired levels and/or vectoring slots. When an OS port used with CSL will use its own dispatcher, the CSL_IntcContext.flags must be equal to CSL_INTC_CONTEXT_DISABLECOREVECTORWRITES. The CSL dispatch code/data may either not be loaded at all, or be overlaid for optimizing on memory footprint. Typically, the event dispatch record constitutes the context information. This table will hold the address of the event handler function and pointer to arbitrary data object (void*) to be passed to event handler as a lone argument.

When INTC Dispatcher will not be used, the CSL_intcHookIsr() can be used to hook the right fetch packet in the Interrupt Vector Table, which in turn leads the CPU control to the right ISR on occurrence of the interrupt.

1.7.4 INTC API Call Sequence

The sequence of calls made by INTC user will be as follows –

```
// Initialize other required CSL3.x Peripherals
CSL_intcSetVectorPtr(DEST_ADDR);  //If relocation of interrupt vector
                                  //required. DEST_ADDR is new location
CSL_intcInit();                   // INTC Module Initialize
CSL_intcDispatcherInit();         // Dispatcher Initialize (if reqd.)
CSL_intcGlobalDisable(..);        // Disable global interrupts.
handle = CSL_intcOpen(..);        // Ready an interrupt for use
CSL_intcHwSetup(handle..);        // Setup interrupt attributes
CSL_intcPlugEventHandler(..);     // Bind the interrupt with the
                                  // corresponding ISR.
CSL_intcHwControl(handle..);      // assorted control, ex: ISR hookup
CSL_intcEventEnable(..);          // Enables the event of interest.
CSL_intcGlobalEnable(..);         // Enable global interrupts.
:
CSL_intcClose(handle);            // End of interrupt use
//                          Terminate                          Program
```

Chapter 2
CHIP Module

Topics

2.1 Overview

This module deals with all System On Chip (SOC) configurations. It constitutes of Configuration Registers specific for the chip. Following are the Registers associated with the CHIP module:

- The two registers related to the L1P Cache
    - L1P Invalidate Start Address Register - This register stores the byte address of the start location of the block in cache which is to be invalidated
    - L1P Invalidate Control Register - This register stores the number of words in the block in cache which is to be invalidated starting from the address specified in L1PSAR
- Memory Control and Status Register - This register gives the memory error status along with a provision for setting the priority between CSP Slave Port and L1P for RAM access
- Program Counter Export Register - This register holds the value of the Program Counter available at the CPU boundary as the PCDP Bus
- CGFPIN0 Register - This register captures the value of the boot mode pins on the rising edge of the RESET (active low)
- CGFPIN1 Register - This register captures the value of the boot mode pins (those not included in CGFPIN0 Register) on the rising edge of the RESET (active low)
- CFGHPI Register - This register controls the HPI Enable and Mode
- CFGHPIAMSB Register - This register gives the Most Significant Byte of the HPI Address (Non-Multiplexed mode)
- CFGHPIAUMB Register - This register gives the Upper Middle Byte of the HPI Address (Non-Multiplexed mode)
- CFGRTI Register - This register selects the sources for the RTI CAPEVT [1:0] inputs
- CFGMCASP0 Register - This register selects which device pin drives McASP0 AMUTEIN
- CFGMCASP1 Register - This register selects which device pin drives McASP1 AMUTEIN
- CFGMCASP2 Register - This register selects which device pin drives McASP2 AMUTEIN
- CFGBRIDGE Register - This register contains the software reset bits for VBUSP bridge(s)
- IDREG Register - This register contains the JTAG id
    - DFT_read_write Register - This register is for DFT

- Addressing Mode Register - This register specifies the addressing mode for the registers which can perform linear or circular addressing, also contain sizes for circular addressing

- Control Status Register - This register contains the control and status bits. This register is used to control the mode of cache. This is also used to enable or disable all the interrupts except reset and non maskable interrupt.

- Floating Point Adder Configuration Register - This register contains the fields that specify underflow or overflow, the rounding mode, NANs, denormalized numbers, and inexact results for the instructions that use .L functional units

- Floating Point Auxiliary Configuration Register - This register contains the fields that specify underflow or overflow, the rounding mode, NANs, denormalized numbers, and inexact results for the instructions that use .S functional units

- Floating Point Multiplier Configuration Register - This register contains the fields that specify underflow or overflow, the rounding mode, NANs, denormalized numbers, and inexact results for the instructions that use .M functional units

2.2 Functions

This section lists the functions available in the CHIP module.

2.2.1 CSL_chipInit

CSL_Status CSL_chipInit (CSL_ChipContext * *pContext*)

Description
This is the initialization function for the CHIP CSL. The function must be called before calling any other API from this CSL. This function does not modify any registers or check status. It returns status CSL_SOK. It has been kept for future use.

Arguments

pContext	Pointer to module-context. As CHIP doesn't have any context based information user is expected to pass NULL.

Return Value
CSL_Status

- CSL_SOK - Always returns

Pre Condition
None

Post Condition
None.

Modifies
None

Example

```
CSL_Statu    status;
...
status = CSL_chipInit(NULL);
```

2.2.2 CSL_chipWriteReg

Uint32 CSL_chipWriteReg (CSL_ChipReg *regId*, Uint32 *val*)

Description
This function may be used to write to the CHIP registers. The register that can be specified could be one of those enumerated in CSL_ChipReg.

Arguments

```
regId   This is the register id specified for the register
        through the enum

val     Value to be written into the register
```

Return Value

- Uint32 - The value in the register before the new value being written

Pre Condition
CSL_chipInit () must be called.

Post Condition
The specified register will be modified.

Modifies
The specified register will be modified.

Example:

```
Uint32 oldRegVal;
...
oldRegVal = CSL_chipWriteReg( CSL_CHIP_REG_AMR, 56);
```

2.2.3 CSL_chipReadReg

Uint32 CSL_chipReadReg (CSL_ChipReg *regId*)

Description
This function may be used to read the CHIP registers. The register that can be specified could be one of those enumerated in CSL_ChipReg.

Arguments

```
regId   This is the register id specified for the register
        through the enum
```

Return Value

- Uint32 - The value read from the register

Pre Condition
CSL_chipInit () must be called.

Post Condition
None

Modifies

None

Example:

```
    Uint32 readValue;
    ...
    readValue = CSL_chipReadReg(CSL_CHIP_REG_CSR);
```

2.3 Data Structures

This section lists the data structures available in the CHIP module.

2.3.1 CSL_ChipContext

Detailed Description
CHIP Module Context.

Field Documentation

Uint32 CSL_ChipContext::contextInfo
Context information of CHIP. This is only a placeholder for future implementation.

2.3.2 CSL_ChipBaseAddress

Detailed Description
CHIP registers base address.

Field Documentation

CSL_ChipRegsOvly CSL_ChipBaseAddress::regs
Base-address of the memory mapped CHIP registers

2.3.3 CSL_ChipRegs

Detailed Description
Register Overlay structure for memory mapped chip registers.

Field Documentation

volatile Uint32 CSL_ChipRegs::CFGHPIAMSB
CFGHPIAMSB Register

volatile Uint32 CSL_ChipRegs::CFGHPIAUMB
CFGHPIAUMB Register

volatile Uint32 CSL_ChipRegs::CFGPIN0
CFGPIN0 Register

volatile Uint32 CSL_ChipRegs::CFGPIN1
CFGPIN1 Register

volatile Uint32 CSL_ChipRegs::CGFBRIDGE
CGFBRIDGE Register

volatile Uint32 CSL_ChipRegs::CGFHPI
CGFHPI Register

volatile Uint32 CSL_ChipRegs::CGFMCASP0
CGFMCASP0 Register

volatile Uint32 CSL_ChipRegs::CGFMCASP1
CGFMCASP1 Register

volatile Uint32 CSL_ChipRegs::CGFMCASP2
CGFMCASP2 Register

volatile Uint32 CSL_ChipRegs::CGFRTI
CGFRTI Register

volatile Uint32 CSL_ChipRegs::DFT_READ_WRITE
DFT_READ_WRITE Register

volatile Uint32 CSL_ChipRegs::IDREG
IDREG Register

volatile Uint32 CSL_ChipRegs::RSVD2[22]
Reserved memory mapped Registers

2.4 Enumerations

This section lists the enumerations available in the CHIP module.

2.4.1 CSL_ChipReg

enum CSL_ChipReg
Enumeration of the System registers.
This enumeration contains the list of registers that can be manipulated using the CSL_chipReadReg(..) and CSL_chipWriteReg(..) APIs.

Enumeration values:

CSL_CHIP_REG_L1PISAR	L1P Invalidate Start Address Register
CSL_CHIP_REG_L1PICR	L1P Invalidate Control Register
CSL_CHIP_REG_MEMCSR	Memory Control and Status Register
CSL_CHIP_REG_PCER	Program Counter Export Register
CSL_CHIP_REG_CFGPIN0	CFGPIN0 register
CSL_CHIP_REG_CFGPIN1	CFGPIN1 register
CSL_CHIP_REG_CFGHPI	CGFHPI register
CSL_CHIP_REG_CFGHPIAMSB	CFGHPIAMSB register
CSL_CHIP_REG_CFGHPIAUMB	CFGHPIAUMB register
CSL_CHIP_REG_CFGRTI	CGFRTI register
CSL_CHIP_REG_CFGMCASP0	CGFMCASP0 register
CSL_CHIP_REG_CFGMCASP1	CGFMCASP1 register
CSL_CHIP_REG_CFGMCASP2	CGFMCASP2 register
CSL_CHIP_REG_CFGBRIDGE	CGFBRIDGE register
CSL_CHIP_REG_IDREG	IDREG register
CSL_CHIP_REG_DFT_READ_WRITE	DFT_READ_WRITE
CSL_CHIP_REG_AMR	Addressing Mode Register
CSL_CHIP_REG_CSR	Control Status Register
CSL_CHIP_REG_FADCR	Floating Point Adder Configuration register
CSL_CHIP_REG_FAUCR	Floating Point Auxiliary Configuration register
CSL_CHIP_REG_FMCR	Floating Point Multiplier Configuration register

Chapter 3
DMAX Module

Topics

3.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within DMAX module.

The dMAX is a module designed to perform Data Movement Acceleration. The Data Movement Accelerator (dMAX) controller handles all user-programmed data transfers between the internal data memory controller and the device peripherals on the C672x DSP. The dMAX allows movement of data to/from any addressable memory space, including internal memory, peripherals, and external memory. The dMAX controller in the C672x DSP has a different architecture from the previous EDMA controller in the C621x/C671x devices.

The dMAX controller includes features, such as capability to perform three dimensional data transfers for advanced data sorting, capability to manage a section of the memory as a circular buffer/FIFO with delay tap based reading and writing data. The dMAX controller is capable of concurrently processing two transfer requests (provided that they are to/from different source/destinations).

The dMAX controller comprises:

- Event and interrupt processing registers.
- Event encoder.
- High priority event Parameter RAM (PaRAM).
- Low priority event Parameter RAM (PaRAM).
- Address generation hardware for High Priority Events – MAX0 (HiMAX)
- Address generation hardware for Low Priority Events – MAX1 (LoMAX).

3.2 Functions

This section lists the functions available in the DMAX module.

3.2.1 CSL_dmaxInit

CSL_Status CSL_dmaxInit (CSL_DmaxContext * *pContext*)

Description
This is the initialization function for the DMAX CSL. This function needs to be called before any other DMAX CSL functions are called. This function does not modify any registers.It Initializes the events and parameters of High and Low priorities with zero's. It returns status CSL_SOK.
Arguments

```
pContext   Context information for the instance. Should be NULL
```

Return Value C
SL_Status

- CSL_SOK - Always returns

Pre Condition
CSL_sysInit() must be called.

Post Condition
None

Modifies
None

Example

```
CSL_Status       status;
...
status = CSL_dmaxInit(NULL);
...
```

3.2.2 CSL_dmaxOpen

CSL_DmaxHandle CSL_dmaxOpen

(CSL_DmaxObj * ***pDmaxObj***,
CSL_InstNum ***dmaxNum***,
CSL_DmaxParam * ***pDmaxParam***,
CSL_Status * ***pStatus***
)

Description
This function populates the peripheral data object for the instance and returns a handle to the instance. The open call sets up the data structures for the particular instance of DMAX device. The device can be re-opened anytime after it has been normally closed if so required. The handle

returned by this call is input as an essential argument for rest of the APIs described for this module.

Arguments

```
pDmaxObj     Pointer to the DMAX instance object

dmaxNum      Instance of the DMAX to be opened.

pDmaxParam   Pointer to module specific parameters

pStatus     Pointer for returning status of the function call
```

Return Value
CSL_DmaxHandle

- Valid Dmax instance handle will be returned if status value is equal to CSL_SOK else CSL_DMAX_BADHANDLE

Pre Condition
CSL_dmaxInit() has to be called.

Post Condition
1. The status is returned in the status variable. If status returned is

- CSL_SOK - Open call is successful
- CSL_ESYS_INVPARAMS - If invalid parameter passed
- CSL_ESYS_FAIL - If invalid instance number

2. DMAX object structure is populated

Modifies

- The status variable
- DMAX object structure

Example

```
CSL_Status        status;
CSL_DmaxObj       dmaxObj;
CSL_DmaxHandle    hDmax;
...
hDmax = CSL_dmaxOpen (&dmaxObj,
                      CSL_DMAX_0,
                      NULL,
                      &status
                     );
...
```

3.2.3 CSL_dmaxClose

CSL_Status CSL_dmaxClose (CSL_DmaxHandle *hDmax*)

Description
This function closes the specified instance of DMAX.

Arguments

```
hDmax     Handle to the DMAX instance
```

Return Value
CSL_Status

- CSL_SOK - DMAX close successful
- CSL_ESYS_BADHANDLE - Invalid handle

Pre Condition
Both CSL_dmaxInit() and CSL_dmaxOpen() must be called successfully in order before calling this function.

Post Condition
CSL for the DMAX instance is closed.

Modifies
CSL_DmaxObj structure values for the instance

Example

```
CSL_Status        status;
CSL_DmaxHandle    hDmax;
...
CSL_dmaxClose (hDmax);
...
```

3.2.4 CSL_dmaxHwSetup

CSL_Status CSL_dmaxHwSetup (CSL_DmaxHandle *hDmax*, CSL_DmaxHwSetup * *setup*)

Description
This function initializes the device registers with the appropriate values provided through the HwSetup Data structure. After the Setup is completed, the device is ready for operation. For information passed through the HwSetup Data structure refer *CSL_dmaxHwSetup*.

Arguments

```
hdmax      Handle to the Dmax instance
```

```
        setup       Pointer to setup structure that programs DMAX
```

Return Value
CSL_Status

- CSL_SOK - Successful completion of hardware setup
- CSL_ESYS_ INVPARAMS - Hardware structure is not properly initialized
- CSL_ESYS_ BADHANDLE - Invalid CSL handle

Pre Condition
Both CSL_dmaxInit() and CSL_dmaxOpen() must be called successfully in order before calling this function.

Post Condition
DMAX registers are configured.

Modifies
DMAX registers

Example

```
CSL_dmaxHandle hDmax;
CSL_dmaxHwSetup hwSetup;
CSL_Status       status;
...

hwSetup.mode           = CSL_DMAX_MODE_MASTER;
hwSetup.dir            = CSL_DMAX_DIR_TRANSMIT;
hwSetup.addrMode       = CSL_DMAX_ADDRSZ_SEVEN;
hwSetup.clksetup       = &clksetup;
...
status = CSL_dmaxHwSetup(hDmax, &hwSetup);
...
```

3.2.5 CSL_dmaxGetHwSetup

CSL_Status CSL_dmaxGetHwSetup (CSL_DmaxHandle *hDmax*, CSL_DmaxHwSetup * *setup*)

Description
It retrieves the hardware setup parameters of the DMAX specified by the given handle.

Arguments

```
            hDmax       Handle to the DMAX

            setup       Pointer to the hardware setup structure
```

Return Value
CSL_Status

- CSL_SOK - Retrieving the hardware setup parameters is successful
- CSL_ESYS_BADHANDLE - The handle is passed is invalid
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition
Both CSL_dmaxInit() and CSL_dmaxOpen() must be called successfully in order before calling this function.

Post Condition
The hardware setup structure is populated with the hardware setup parameters.

Modifies
setup variable

Example

```
CSL_DmaxHandle   hDmax;
CSL_DmaxHwSetup  hwSetup;
CSL_Status       status;
...
status = CSL_dmaxGetHwSetup(hDmax, &hwSetup);
...
```

3.2.6 CSL_dmaxHwControl

CSL_Status CSL_dmaxHwControl **(** CSL_DmaxHandle ***hDmax***,
CSL_DmaxHwControlCmd ***cmd***,
void * ***arg***
)

Description
Takes a command of DMAX with an optional argument and implements it.

Arguments

hDmax	Handle to the DMAX instance
cmd	The command to this API indicates the action to be taken on DMAX.
arg	An optional argument.

Return Value
CSL_Status

- CSL_SOK - Status info return successful.

- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition
Both CSL_dmaxInit() and CSL_dmaxOpen() must be called successfully in order before calling this function.

Post Condition
Device register programmed accordingly.

Modifies
The hardware registers of DMAX.

Example

```
CSL_DmaxHandle         hDmax;
CSL_DmaxHwControlCmd   cmd;
void*                 arg;
CSL_Status       status;
...
status = CSL_dmaxHwControl (hDmax, cmd, &arg);
...
```

3.2.7 CSL_dmaxHwSetupRaw

CSL_Status CSL_dmaxHwSetupRaw **(** CSL_DmaxHandle ***hDmax*,**
CSL_DmaxConfig * ***config***
)

Description
This function initializes the device registers with the register-values provided through the Config Data structure. This configures registers based on a structure of register values, as compared to HwSetup, which configures registers based on structure of bit field values and may perform other functions (delays, etc.)

Arguments

```
hDmax       Handle to the DMAX

config      Pointer to config structure
```

Return Value
CSL_Status

- CSL_SOK - Configuration successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Configuration is not properly initialized

Pre Condition

Both CSL_dmaxInit() and CSL_dmaxOpen() must be called successfully in order before calling this function.

Post Condition
The registers of the specified DMAX instance will be setup according to value passed.

Modifies
Hardware registers of the specified DMAX instance.

Example

```
CSL_DmaxHandle       hDmax;
CSL_DmaxConfig       config = CSL_DMAX_CONFIG_DEFAULTS;
CSL_Status           status;
...
status = CSL_dmaxHwSetupRaw (hDmax, &config);
...
```

3.2.8 CSL_dmaxGetHwStatus

CSL_Status CSL_dmaxGetHwStatus **(** CSL_DmaxHandle ***hDmax***,
CSL_DmaxHwStatusQuery ***query***,
void * ***response***
)

Description
Gets the status of the different operations of DMAX.

Arguments

```
hDmax       Handle to the DMAX instance

query       The query to this API of DMAX which indicates the
            status to be returned.

response    Placeholder to return the status.
```

Return Value
CSL_Status

- CSL_SOK - Status info return successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Invalid query command
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition
Both CSL_dmaxInit() and CSL_dmaxOpen() must be called successfully in order before calling this function.

Post Condition

None

Modifies
"response" variable holds the queried value.

Example

```
CSL_DmaxHandle          hDmax;
CSL_DmaxHwStatusQuery   query;
void*                   reponse;
CSL_Status       status;
...
status = CSL_GetdmaxHwStatus (hDmax, query, &response);
...
```

3.2.9 CSL_dmaxGetBaseAddress

CSL_Status CSL_dmaxGetBaseAddress (CSL_InstNum *dmaxNum*,
CSL_DmaxParam * *pDmaxParam*,
CSL_DmaxBaseAddress * *pBaseAddress*
)

Description
Function to get the base address of the peripheral instance. This function is used for getting the base address of the peripheral instance. This function will be called inside the CSL_dmaxOpen() function call. This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral MMRss go to an alternate location.

Arguments

dmaxNum	Specifies the instance of DMAX to be opened.
pDmaxParam	Module specific parameters.
pBaseAddress	Pointer to baseaddress structure containing base address details.

Return Value
CSL_Status

- CSL_SOK Open call is successful
- CSL_ESYS_FAIL The instance number is invalid.
- CSL_ESYS_INVPARAMS Invalid parameter

Pre Condition
None

Post Condition
Base Address structure is populated.

Modifies
1. The status variable
2. Base address structure is modified.

Example

```
CSL_Status           status;
CSL_DmaxBaseAddress  baseAddress;
...
status = CSL_dmaxGetBaseAddress(CSL_DMAX_0, NULL,
                                &baseAddress);
...
```

3.2.10 CSL_dmaxSetupFifoDesc

CSL_Status CSL_dmaxSetupFifoDesc (CSL_DmaxFifoDescriptor * *fifoObj*,
CSL_DmaxFifoDescriptorSetup * *setup*
)

Description
This function formats a FIFO Descriptor object. *CSL_dmaxSetupFifoDesc*.

Arguments

```
fifoObj      Pointer to data area in RAM where descriptor is to
             be stored.

setup        Pointer to a dMax FIFO descriptor Object.
```

Return Value
CSL_Status

- CSL_SOK - Successful completion of hardware setup
- CSL_ESYS_ INVPARAMS - Hardware structure is not properly initialized
- CSL_ESYS_ BADHANDLE - Invalid CSL handle

Pre Condition
Both CSL_dmaxInit() and CSL_dmaxOpen() must be called successfully in order before calling this function.

Post Condition
DMAX registers are configured.

Modifies
DMAX registers

Example

```
CSL_DmaxFifoDescriptor        fifoObj;
CSL_DmaxFifoDescriptorSetup   setup;
CSL_Status       status;
```

```
setup.fifoBaseAddr      = 0x20000100;
...
setup.wPtr              = 0x10000000;
...
CSL_dmaxSetupFifoDesc(&fifoObj, &setup);
...
```

3.2.11 CSL_dmaxStartAsyncTransferMulti

CSL_Status CSL_dmaxStartAsyncTransferMulti (CSL_DmaxHandle * *hDmax*)

Description
This function initiates a dMax data transfer by toggling the appropriate bit in the ESR (Event Set Register) register for multiple events. *CSL_dmaxStartAsyncTransferMulti*.

Arguments

```
 hDmax            Pointer to the dMax Handle Instance
```

Return Value
CSL_Status

- CSL_SOK - Successful completion
- CSL_EDMAX_BAD_ESR_FLAG - Bad ESR Flag
- CSL_ESYS_ BADHANDLE - Invalid CSL handle

Pre Condition
Both CSL_dmaxInit() and CSL_dmaxOpen() must be called successfully in order before calling this function.

Post Condition
DMAX registers are configured.

Modifies
DMAX registers

Example

```
CSL_DmaxHandle   hDmax;
CSL_Status       status;
...
CSL_dmaxStartAsyncTransferMulti(hDmax);
...
```

3.2.12 CSL_dmaxGetHwSetupFifoXFRParamEntry

CSL_Status CSL_dmaxGetHwSetupFifoXFRParamEntry (CSL_DmaxParameterEntry * *paramEntry*, CSL_DmaxFifoParameterSetup * *setup*

)

Description
This function initializes the device registers with the appropriate values provided through the HwSetup Data structure. After the Setup is completed, the device is ready for operation. For information passed through the HwSetup Data structure refer *CSL_dmaxHwSetup*.

Arguments

paramEntry	Pointer to Parameter Table Entry where data is to be stored.
setup	Pointer to a dmax FIFO Transfer Setup Object

Return Value
CSL_Status

- CSL_SOK - Successful completion of hardware setup.
- CSL_ESYS_ INVPARAMS - Hardware structure is not properly initialized.
- CSL_ESYS_ BADHANDLE - Invalid CSL handle

Pre Condition
Both CSL_dmaxInit() and a CSL_dmaxOpen() must be called.

Post Condition
DMAX registers are configured.

Modifies
DMAX registers

Example

```
CSL_DmaxParameterEntry  paramEntry;
CSL_dmaxHwSetup         hwSetup;
CSL_Status              status;
hwSetup.linearReload0 = paramEntry[0];
...
hwSetup.delayTabPtr1  = paramEntry[9];
...
status = CSL_dmaxGetHwSetupFifoXFRParamEntry(hDmax, &hwSetup);
...
```

3.2.13 CSL_dmaxGetHwSetupGenXFRParamEntry

CSL_Status CSL_dmaxGetHwSetupGenXFRParamEntry	**(**	**CSL_DmaxParameterEntry ***	***paramEntry*,**
		Uint8	***cc*,**
		CSL_DmaxGPXFRParameterSetup *	***setup***

)

Description
This function formats the parameter table entry for a General Purpose transfer.

Arguments

```
paramEntry    Pointer to Parameter Table Entry where data is to
              be stored.

cc            Counter configuration

setup         Pointer to a dmax General Purpose Transfer Object.
```

Return Value
CSL_Status

- CSL_SOK - Successful completion of hardware setup
- CSL_ESYS_ INVPARAMS - Hardware structure is not properly initialized
- CSL_ESYS_ BADHANDLE - Invalid CSL handle

Pre Condition
Both CSL_dmaxInit() and a CSL_dmaxOpen() must be called.

Post Condition
DMAX registers are configured.

Modifies
DMAX registers

Example

```
CSL_DmaxParameterEntry        paramEntry;
Uint8                         cc;
CSL_DmaxGPXFRParameterSetup   hwSetup;
CSL_Status                    status;

hwSetup.srcReloadAddr0  = paramEntry[0];
...
hwSetup.dstReloadAddr0  = paramEntry[1];
...

status = CSL_dmaxGetHwSetupGenXFRParamEntry(paramEntry, cc,
                                            &hwSetup);
...
```

3.2.14 CSL_dmaxGetHwSetup1dXFRParamEntry

CSL_Status CSL_dmaxGetHwSetup1dXFRParamEntry **(CSL_DmaxParameterEntry * *paramEntry*,**

CSL_Dmax1dParameterSetup * *setup*

)

Description
This function initializes the device registers with the appropriate values provided through the HwSetup Data structure. After the Setup is completed, the device is ready for operation. For information passed through the HwSetup Data structure refer *CSL_dmaxHwSetup*.

Arguments

paramEntry	Pointer to Parameter Table Entry where data is to be stored.
setup	Pointer to a dmax 1d Burst Transfer Setup Object

Return Value
CSL_Status

- CSL_SOK - Successful completion of hardware setup.
- CSL_ESYS_ INVPARAMS - Hardware structure is not properly initialized.
- CSL_ESYS_ BADHANDLE - Invalid CSL handle

Pre Condition
Both CSL_dmaxInit() and a CSL_dmaxOpen() must be called.

Post Condition
DMAX registers are configured.

Modifies
DMAX registers

Example

```
CSL_DmaxParameterEntry  paramEntry;
CSL_dmaxHwSetup         hwSetup;
CSL_Status              status;
hwSetup.linearReload0 = paramEntry[0];
...
hwSetup.delayTabPtr1  = paramEntry[9];
...
status = CSL_dmaxGetHwSetup1dXFRParamEntry(hDmax, &hwSetup);
...
```

3.2.15 CSL_dmaxGetHwSetupSpiXFRParamEntry

CSL_Status CSL_dmaxGetHwSetupSpiXFRParamEntry (CSL_DmaxParameterEntry * ***paramEntry***,

CSL_DmaxSpiParameterSetup *setup*
*

)

Description
This function initializes the device registers with the appropriate values provided through the HwSetup Data structure. After the Setup is completed, the device is ready for operation. For information passed through the HwSetup Data structure refer *CSL_dmaxHwSetup*.

Arguments

paramEntry	Pointer to Parameter Table Entry where data is to be stored.
setup	Pointer to a dmax Spi Transfer Setup Object

Return Value
CSL_Status

- CSL_SOK - Successful completion of hardware setup.
- CSL_ESYS_ INVPARAMS - Hardware structure is not properly initialized.
- CSL_ESYS_ BADHANDLE - Invalid CSL handle

Pre Condition
Both CSL_dmaxInit() and a CSL_dmaxOpen() must be called.

Post Condition
DMAX registers are configured.

Modifies
DMAX registers

Example

```
CSL_DmaxParameterEntry  paramEntry;
CSL_dmaxHwSetup         hwSetup;
CSL_Status              status;
hwSetup.linearReload0 = paramEntry[0];
...
hwSetup.delayTabPtr1  = paramEntry[9];
...
status = CSL_dmaxGetHwSetupSpiXFRParamEntry(hDmax, &hwSetup);
...
```

3.2.16 CSL_dmaxSetupGeneralXFRParameterEntry

CSL_Status CSL_dmaxSetupGeneralXFRParameterEntry	**(**	**CSL_DmaxParameterEntry ***	***paramEntry*,**
		Uint8	***cc*,**
		CSL_DmaxGPXFRParameterSetup *	***setup***
	)		

Description
This function formats the parameter table entry for a General Purpose transfer.

Arguments

```
paramEntry    Pointer to Parameter Table Entry where data is to
              be stored.

cc            Counter configuration

setup         Pointer to a dmax General Purpose Transfer Object.
```

Return Value
CSL_Status

- CSL_SOK - Successful completion of hardware setup
- CSL_ESYS_ INVPARAMS - Hardware structure is not properly initialized
- CSL_ESYS_ BADHANDLE - Invalid CSL handle

Pre Condition
Both CSL_dmaxInit() and a CSL_dmaxOpen() must be called.

Post Condition
DMAX registers are configured.

Modifies
DMAX registers

Example

```
CSL_DmaxParameterEntry        paramEntry;
Uint8                         cc;
CSL_DmaxGPXFRParameterSetup   hwSetup;
CSL_Status                    status;
hwSetup.srcReloadAddr0  = 0x10000100;
hwSetup.dstReloadAddr0  = 0x20000100;
...
hwSetup.count0          = 3;
hwSetup.count1          = 4;
hwSetup.count2          = 2;
...
status  =  CSL_dmaxHwSetupGeneralXFRParameterEntry(paramEntry,  cc,
                                        &hwSetup);
```

```
...
```

3.2.17 CSL_dmaxSetupFifoXFRParameterEntry

CSL_Status CSL_dmaxSetupFifoXFRParameterEntry (CSL_DmaxParameterEntry * *paramEntry*,
CSL_DmaxFifoParameterSetup * *setup*
)

Description
This function initializes the device registers with the appropriate values provided through the HwSetup Data structure. After the Setup is completed, the device is ready for operation. For information passed through the HwSetup Data structure refer *CSL_dmaxHwSetup*.

Arguments

```
paramEntry    Pointer to Parameter Table Entry where data is to
              be stored

setup         Pointer to a dmax FIFO Transfer Setup Object
```

Return Value
CSL_Status

- CSL_SOK - Successful completion of hardware setup
- CSL_ESYS_ INVPARAMS - Hardware structure is not properly initialized
- CSL_ESYS_ BADHANDLE - Invalid CSL handle

Pre Condition
Both CSL_dmaxInit() and a CSL_dmaxOpen() must be called.

Post Condition
DMAX registers are configured

Modifies
DMAX registers

Example

```
CSL_DmaxParameterEntry  paramEntry;
CSL_dmaxHwSetup         hwSetup;

hwSetup.linearReload0  = 0x10000100;
...
hwSetup.count0         = 3;
...
status = CSL_dmaxHwSetupFifoXFRParameterEntry(hDmax, &hwSetup);
...
```

3.2.18 CSL_dmaxSetup1dXFRParameterEntry

CSL_Status CSL_dmaxSetup1dXFRParameterEntry (CSL_DmaxParameterEntry * *paramEntry*, CSL_Dmax1dXFRParameterSetup * *setup*)

Description
This function formats the parameter table entry for a 1d Burst transfer.

Arguments

paramEntry	Pointer to Parameter Table Entry where data is to be stored.
setup	Pointer to a dmax 1d Burst Transfer Object.

Return Value
CSL_Status

- CSL_SOK - Successful completion of hardware setup
- CSL_ESYS_ INVPARAMS - Hardware structure is not properly initialized
- CSL_ESYS_ BADHANDLE - Invalid CSL handle

Pre Condition
Both CSL_dmaxInit() and a CSL_dmaxOpen() must be called.

Post Condition
DMAX registers are configured.

Modifies
DMAX registers

Example

```
CSL_DmaxParameterEntry        paramEntry;
Uint8                         cc;
CSL_Dmax1dXFRParameterSetup   hwSetup;
CSL_Status                    status;
hwSetup.srcAddr         = 0x10000100;
hwSetup.dstAddr         = 0x20000100;
...
hwSetup.count           = 3;
hwSetup.burstlen        = 32;
hwSetup.nburst          = 3;
...
status = CSL_dmaxHwSetup1dXFRParameterEntry(paramEntry, cc,
                                &hwSetup);
...
```

3.2.19 CSL_dmaxSetupSpiXFRParameterEntry

CSL_Status CSL_dmaxSetupSpiXFRParameterEntry (CSL_DmaxParameterEntry * *paramEntry*, CSL_DmaxSpiXFRParameterSetup * *setup*)

Description
This function formats the parameter table entry for a SPI Slave transfer.

Arguments

paramEntry	Pointer to Parameter Table Entry where data is to be stored.
setup	Pointer to a dmax SPI Slave Transfer Object.

Return Value
CSL_Status

- CSL_SOK - Successful completion of hardware setup
- CSL_ESYS_ INVPARAMS - Hardware structure is not properly initialized
- CSL_ESYS_ BADHANDLE - Invalid CSL handle

Pre Condition
Both CSL_dmaxInit() and a CSL_dmaxOpen() must be called.

Post Condition
DMAX registers are configured.

Modifies
DMAX registers

Example

```
CSL_DmaxParameterEntry        paramEntry;
Uint8                         cc;
CSL_DmaxSpiXFRParameterSetup  hwSetup;
CSL_Status                    status;
hwSetup.srcRldAddr0  = 0x10000100;
hwSetup.dstRldAddr0  = 0x20000100;
...
hwSetup.count0          = 3;
hwSetup.rldCount        = 4;
...
status = CSL_dmaxHwSetupSpiXFRParameterEntry(paramEntry, cc,
                              &hwSetup);
...
```

3.2.20 CSL_dmaxGetNextFreeParamEntry

Uint32 CSL_dmaxGetNextFreeParamEntry (Uint32 *uid*,
CSL_Status * *st*
)

Description
CSL_dmaxGetNextFreeParamEntry searches for next free resource given the resource type.

Arguments

```
uid         Pointer to Parameter Table Entry where data is stored

st          status
```

Return Value
CSL_Status

- CSL_SOK – Found next free Parameter Entry.
- CSL_ESYS_ OVFL – Next free Parameter Entry not found

Pre Condition
Both CSL_dmaxInit() and a CSL_dmaxOpen() must be called.

Post Condition
None

Modifies
None

Example

```
CSL_Status       status;
...
CSL_dmaxGetNextFreeParameterEntry(hDmax->paramUid, &status);
...
```

3.3 Data Structures

This section lists the data structures available in the DMAX module.

3.3.1 CSL_DmaxObj

Detailed Description
This object contains the reference to the instance of DMAX opened using the *CSL_dmaxOpen()*. The pointer to this is passed to all DMAX CSL APIs.

Field Documentation

Uint32 CSL_DmaxObj::eventUid
Event Number

Uint32* CSL_DmaxObj::hiTableEventEntryPtr
Pointer to entry in High Event Table reserved for this event

Uint32* CSL_DmaxObj::loTableEventEntryPtr
Pointer to entry in Low Event Table reserved for this event

Uint8 CSL_DmaxObj::paramPtr
Pointer to the associated parameter entry

Uint32 CSL_DmaxObj::paramUid
Parameter Entry Number

CSL_InstNum CSL_DmaxObj::perNum
This is the instance of DMAX being referred to by this object

CSL_DmaxRegsOvly CSL_DmaxObj::regs
Pointer to the dmax Registers

3.3.2 CSL_DmaxConfig

Detailed Description
Config structure of DMAX. This is used to configure DMAX using CSL_HwSetupRaw function.

Field Documentation

Uint32 CSL_DmaxConfig::eventCtrl
Event Control

Uint8 CSL_DmaxConfig::eventType
Dmax Event Type

CSL_DmaxParameterEntry* CSL_DmaxConfig::paramEntry
DMAX parameter Entry Pointer

Uint8 CSL_DmaxConfig::polarity
DMAX Polarity

Uint8 CSL_DmaxConfig::priority
DMAX priority

3.3.3 CSL_DmaxContext

Detailed Description
DMAX specific parameters. Present implementation doesn't have any specific parameters. The below declaration is just a placeholder for future implementation.

Field Documentation

Uint16 CSL_DmaxContext::contextInfo
Context information of DMAX. The below declaration is just a placeholder for future implementation.

3.3.4 CSL_DmaxHwSetup

Detailed Description
This has all the fields required to configure DMAX at Power Up (After a Hardware Reset) or a Soft Reset. This structure is used to setup or obtain existing setup of DMAX using *CSL_dmaxHwSetup()* and *CSL_dmaxGetHwSetup()* functions respectively.

Field Documentation

CSL_DmaxEventSetup CSL_DmaxHwSetup::eventSetup
Generic Pointer for setup

Uint8 CSL_DmaxHwSetup::polarity
DMAX Polarity

Uint8 CSL_DmaxHwSetup::priority
DMAX Priority

3.3.5 CSL_DmaxParam

Detailed Description
DMAX specific parameters. Present implementation doesn't have any specific parameters.

Field Documentation

CSL_BitMask16 CSL_DmaxParam::flags
Bit mask to be used for module specific parameters. The below declaration is just a place-holder for future implementation.

3.3.6 CSL_DmaxBaseAddress

Detailed Description
This will have the base-address information for the peripheral instance.

Field Documentation

CSL_DmaxRegsOvly CSL_DmaxBaseAddress::regs

Base-address of the Configuration registers of DMAX.

3.3.7 CSL_DmaxAlloc

Detailed Description
This object contains the reference to the instance of DMAX opened using in the *CSL_dmaxOpen()*.

Field Documentation

Uint32 CSL_DmaxAlloc::allocMap
Allocation Map

Uint16 CSL_DmaxAlloc::maxCount
Maximum Count

3.3.8 CSL_DmaxResourceAlloc

Detailed Description
This object contains the reference to the instance of DMAX opened using in the *CSL_dmaxOpen()*.

Field Documentation

CSL_DmaxAlloc CSL_DmaxResourceAlloc::cslDmaxHpecrAlloc
Dmax High Priority Event

CSL_DmaxAlloc CSL_DmaxResourceAlloc::cslDmaxHpptAlloc
Dmax High Priority Parameter

CSL_DmaxAlloc CSL_DmaxResourceAlloc::cslDmaxLpecrAlloc
Dmax Low Priority Event

CSL_DmaxAlloc CSL_DmaxResourceAlloc::cslDmaxLpptAlloc
Dmax Low Priority Parameter

3.3.9 CSL_DmaxActiveSet

Detailed Description
This structure contains information for setting the active set of parameters in a general purpose transfer parameter entry.

Field Documentation

Uint16 CSL_DmaxActiveSet::activeCount0
Active Count0 Value

Uint16 CSL_DmaxActiveSet::activeCount1
Active Count1 Value

Uint16 CSL_DmaxActiveSet::activeCount2
Active Count2 Value

Uint32 CSL_DmaxActiveSet::activeDst
Active Destination Address

Uint8 CSL_DmaxActiveSet::activePP
Active Ping Pong Value

Uint32 CSL_DmaxActiveSet::activeSrc
Active Source Address

3.3.10 CSL_DmaxFifoParam

Detailed Description
This structure contains the necessary information to define the FIFO parameters for the reload address for the data transmission.

Field Documentation

Uint32 CSL_DmaxFifoParam::count
Count Value

Uint32 CSL_DmaxFifoParam::dtabPtr0
Pointer to Delay Table

Uint32 CSL_DmaxFifoParam::dtabPtr1
Pointer to Delay Table

Uint32 CSL_DmaxFifoParam::indx0
Index0 Value

Uint32 CSL_DmaxFifoParam::indx1
Index1 Value

Uint32 CSL_DmaxFifoParam::linearAddr
Linear Address

Uint32 CSL_DmaxFifoParam::linearReload0
Linear Reload0 Value

Uint32 CSL_DmaxFifoParam::linearReload1
Linear Reload1 Value

Uint32 CSL_DmaxFifoParam::pfd
Pointer to FIFO Descriptor

3.3.11 CSL_DmaxFifoDesc

Detailed Description
This structure contains the necessary information to define the FIFO parameters for any FIFO that is used in dmax FIFO read/write operations.

Field Documentation

Uint32 CSL_DmaxFifoDesc::efield

FIFO Error Field

Uint32 CSL_DmaxFifoDesc::emark
FIFO Empty Watermark

Uint32 CSL_DmaxFifoDesc::fifoBaseAddr
FIFO Base Address

Uint32 CSL_DmaxFifoDesc::fifoSize
FIFO Size

Uint32 CSL_DmaxFifoDesc::fmark
FIFO Full Watermark

Uint32 CSL_DmaxFifoDesc::rPtr
FIFO Read Pointer

Uint32 CSL_DmaxFifoDesc::wPtr
FIFO Write Pointer

3.3.12 CSL_DmaxFifoDescriptorSetup

Detailed Description
This structure contains the necessary information to define the FIFO parameters for any FIFO that is used in dmax FIFO read/write operations.

Field Documentation

Uint32 CSL_DmaxFifoDescriptorSetup::eMark
FIFO Empty Watermark

Uint8 CSL_DmaxFifoDescriptorSetup::emsc
FIFO Empty Mark Status

Uint8 CSL_DmaxFifoDescriptorSetup::esize
FIFO Error Field

Uint32 CSL_DmaxFifoDescriptorSetup::fifoBaseAddr
FIFO Base Address

Uint32 CSL_DmaxFifoDescriptorSetup::fifoSize
FIFO Size

Uint32 CSL_DmaxFifoDescriptorSetup::fMark
FIFO Full Watermark

Uint8 CSL_DmaxFifoDescriptorSetup::fmsc
FIFO Full Mark Status

Uint32 CSL_DmaxFifoDescriptorSetup::rPtr
FIFO Read Pointer

Uint32 CSL_DmaxFifoDescriptorSetup::wPtr
FIFO Write Pointer

3.3.13 CSL_DmaxFifoDescriptor

Detailed Description
This structure contains blank form for a FIFO descriptor object.

Field Documentation

Uint32 CSL_DmaxFifoDescriptor::word0
Word0

Uint32 CSL_DmaxFifoDescriptor::word1
Word1

Uint32 CSL_DmaxFifoDescriptor::word2
Word2

Uint32 CSL_DmaxFifoDescriptor::word3
Word3

Uint32 CSL_DmaxFifoDescriptor::word4
Word4

Uint32 CSL_DmaxFifoDescriptor::word5
Word5

Uint32 CSL_DmaxFifoDescriptor::word6
Word6

3.3.14 CSL_DmaxGPXFRParameterSetup

Detailed Description
This structure contains the information necessary to perform HW setup for a general purpose data transfer request for dmax.

Field Documentation

Uint16 CSL_DmaxGPXFRParameterSetup::count0
Dimension 0 Count Value

Uint16 CSL_DmaxGPXFRParameterSetup::count1
Dimension 1 Count Value

Uint16 CSL_DmaxGPXFRParameterSetup::count2
Dimension 2 Count Value

Int16 CSL_DmaxGPXFRParameterSetup::dstIndex0
Destination Index0 Value

Int16 CSL_DmaxGPXFRParameterSetup::dstIndex1
Destination Index1 Value

Int16 CSL_DmaxGPXFRParameterSetup::dstIndex2
Destination Index2 Value

Uint32 CSL_DmaxGPXFRParameterSetup::dstReloadAddr0
Destination Reload Address0

Uint32 CSL_DmaxGPXFRParameterSetup::dstReloadAddr1
Destination Reload Address1

Int16 CSL_DmaxGPXFRParameterSetup::srcIndex0
Source Index0 Value

Int16 CSL_DmaxGPXFRParameterSetup::srcIndex1
Source Index1 Value

Int16 CSL_DmaxGPXFRParameterSetup::srcIndex2
Source Index2 Value

Uint32 CSL_DmaxGPXFRParameterSetup::srcReloadAddr0
Source Reload Address0

Uint32 CSL_DmaxGPXFRParameterSetup::srcReloadAddr1
Source Reload Address1

3.3.15 CSL_DmaxFifoParameterSetup

Detailed Description
This structure contains the data items necessary to represent an FIFO R/W Parameter Entry in dmax Parameter RAM.

Field Documentation

Uint16 CSL_DmaxFifoParameterSetup::count0
Count0 Value

Uint16 CSL_DmaxFifoParameterSetup::count1
Count1 Value

Uint32 CSL_DmaxFifoParameterSetup::delayTabPtr0
Pointer to Delay Table0

Uint32 CSL_DmaxFifoParameterSetup::delayTabPtr1
Pointer to Delay Table1

Int16 CSL_DmaxFifoParameterSetup::linearIndx0
Linear Index0 Value

Int16 CSL_DmaxFifoParameterSetup::linearIndx1
Linear Index1 Value

Uint32 CSL_DmaxFifoParameterSetup::linearReload0
Linear Reload0 Value

Uint32 CSL_DmaxFifoParameterSetup::linearReload1
Linear Reload1 Value

Uint32 CSL_DmaxFifoParameterSetup::pfd
Pointer to FIFO Descriptor

3.3.16 CSL_Dmax1dParameterSetup

Detailed Description
This structure contains the data items necessary to represent an 1d Burst Parameter Entry in dmax Parameter RAM.

Field Documentation

Uint16 CSL_Dmax1dParameterSetup::event
event Value

Uint16 CSL_Dmax1dParameterSetup::tcc
tcc Value

Uint16 CSL_Dmax1dParameterSetup::nburst
nburst Value

Uint16 CSL_Dmax1d ParameterSetup::tccint
tccint value

Uint32 CSL_Dmax1dParameterSetup::srcAddr
srcAddr Value

Int16 CSL_Dmax1dParameterSetup::dstAddr
dstAddr Value

Uint16 CSL_Dmax1dParameterSetup::burstlen
burstlen Value

Uint32 CSL_Dmax1dParameterSetup::count
count Value

3.3.17 CSL_DmaxSpiParameterSetup

Detailed Description
This structure contains the information necessary to perform HW setup for a SPI Slave data transfer request for dmax.

Field Documentation

Uint32 CSL_DmaxSpiParameterSetup::srcAddr
srcAddr Value

Uint32 CSL_DmaxSpiParameterSetup::dstAddr
dstAddr Value

Uint32 CSL_DmaxSpiParameterSetup::srcRldAddr0
srcRldAddr0 Value

UInt32 CSL_DmaxSpiParameterSetup::dstRldAddr0

dstRldAddr0 Value

UInt32 CSL_DmaxSpiParameterSetup::srcRldAddr1
srcRldAddr1 Value

UInt32 CSL_DmaxSpiParameterSetup::dstRldAddr1
dstRldAddr1 Value

Uint16 CSL_DmaxSpiParameterSetup::pp
pp Value

Uint16 CSL_DmaxSpiParameterSetup::count
count Value

UInt16 CSL_DmaxSpiParameterSetup::rldCount
Reload Count value

3.3.18 CSL_DmaxGPTransferEventSetup

Detailed Description
This structure contains all data necessary to setup dmax for a General Purpose data transfer request.

Field Documentation

Uint8 CSL_DmaxGPTransferEventSetup::atcint
Alternate Transfer Mode Interrupt

Uint8 CSL_DmaxGPTransferEventSetup::cc
Counter Configuration

Uint8 CSL_DmaxGPTransferEventSetup::esize
Element Size

Uint32 CSL_DmaxGPTransferEventSetup::etype
Event Type

CSL_DmaxParameterSetup CSL_DmaxGPTransferEventSetup::paramSetup
Pointer to parameterSetup

Uint8 CSL_DmaxGPTransferEventSetup::pte
Pointer to Transfer Entry

Uint8 CSL_DmaxGPTransferEventSetup::qtsl
Quantum Transfer Size Limit

Uint8 CSL_DmaxGPTransferEventSetup::rload
Reload

Uint8 CSL_DmaxGPTransferEventSetup::sync
Transfer Synchronization

Uint8 CSL_DmaxGPTransferEventSetup::tcc
Transfer Complete Code

Uint8 CSL_DmaxGPTransferEventSetup::tcint
Transfer Completion Interrupt

3.3.19 CSL_DmaxFifoTransferEventSetup

Detailed Description
This structure contains all data necessary to setup dmax for a data transfer request.

Field Documentation

Uint8 CSL_DmaxFifoTransferEventSetup::atcint
Alternate Transfer Mode Interrupt

Uint32 CSL_DmaxFifoTransferEventSetup::etype
Event Type

CSL_DmaxParameterSetup **CSL_DmaxFifoTransferEventSetup::paramSetup**
Pointer to parameterSetup

Uint8 CSL_DmaxFifoTransferEventSetup::pte
Pointer to Transfer Entry

Uint8 CSL_DmaxFifoTransferEventSetup::qtsl
Quantum Transfer Size Limit

Uint8 CSL_DmaxFifoTransferEventSetup::rload
Reload

Uint8 CSL_DmaxFifoTransferEventSetup::sync
Transfer Synchronization

Uint8 CSL_DmaxFifoTransferEventSetup::tcc
Transfer Complete Code

Uint8 CSL_DmaxFifoTransferEventSetup::tcint
Transfer Completion Interrupt

Uint8 CSL_DmaxFifoTransferEventSetup::wmenab
Water Mark Enable

3.3.20 CSL_Dmax1dBurstTransferEventSetup

Detailed Description
This structure contains all data necessary to setup dmax for a data transfer request.

Field Documentation

Uint32 CSL_Dmax1dBurstTransferEventSetup::etype
Event Type

CSL_DmaxParameterSetup **CSL_Dmax1dBurstTransferEventSetup::paramSetup**
Pointer to parameterSetup

Uint8 CSL_Dmax1dBurstTransferEventSetup::pte
Pointer to Transfer Entry

3.3.21 CSL_DmaxSpiSlaveTransferEventSetup

Detailed Description
This structure contains all data necessary to setup dmax for a data transfer request.

Field Documentation

Uint32 CSL_DmaxSpiSlaveTransferEventSetup::etype
Event Type

CSL_DmaxParameterSetup **CSL_DmaxSpiSlaveTransferEventSetup::paramSetup**
Pointer to parameterSetup

Uint8 CSL_DmaxSpiSlaveTransferEventSetup::pte
Pointer to Transfer Entry

Uint8 CSL_DmaxSpiSlaveTransferEventSetup::spi
Spi peripheral used for data transfer

Uint8 CSL_DmaxSpiSlaveTransferEventSetup::rload
Reload

Uint8 CSL_DmaxSpiSlaveTransferEventSetup::tcc
Transfer Complete Code

Uint8 CSL_DmaxSpiSlaveTransferEventSetup::tcint
Transfer Completion Interrupt

Uint8 CSL_DmaxSpiSlaveTransferEventSetup::esize

Eelement size

3.3.22 CSL_DmaxCpuintEventSetup

Detailed Description
This structure contains the information needed to perform HW setup for CPU interrupt requests.

Field Documentation

Uint8 CSL_DmaxCpuintEventSetup::cpuInt
CPU Interrupt Line

Uint32 CSL_DmaxCpuintEventSetup::etype
Event Type

3.3.23 CSL_DmaxParameterEntry

Detailed Description
This structure defines the overlay for accessing entries in the Hi Priority Parameter Table.

Field Documentation

volatile Uint32 CSL_DmaxParameterEntry::word0
Parameter Table 0

volatile Uint32 CSL_DmaxParameterEntry::word1
Parameter Table 1

volatile Uint32 CSL_DmaxParameterEntry::word2
Parameter Table 2

volatile Uint32 CSL_DmaxParameterEntry::word3
Parameter Table 3

volatile Uint32 CSL_DmaxParameterEntry::word4
Parameter Table 4

volatile Uint32 CSL_DmaxParameterEntry::word5
Parameter Table 5

volatile Uint32 CSL_DmaxParameterEntry::word6
Parameter Table 6

volatile Uint32 CSL_DmaxParameterEntry::word7
Parameter Table 7

volatile Uint32 CSL_DmaxParameterEntry::word8
Parameter Table 8

volatile Uint32 CSL_DmaxParameterEntry::word9
Parameter Table 9

volatile Uint32 CSL_DmaxParameterEntry::word10
Parameter Table 10

3.3.24 CSL_DmaxEtype

Detailed Description
This structure is used to setup the Event Type.

Field Documentation

Uint32 CSL_DmaxEtype::etype
Event Type

3.4 Enumerations

This section lists the enumerations available in the DMAX module.

3.4.1 CSL_DmaxHwStatusQuery

enum CSL_DmaxHwStatusQuery
Enumeration for queries passed to *CSL_dmaxGetHwStatus()*.
This is used to get the status of different operations or to get the existing setup of DMAX.

Enumeration values:

CSL_DMAX_QUERY_EVENTFLAG	Get status of the Event Flag. **Parameters:** *(* Uint32*)
CSL_DMAX_QUERY_GET_TCC	Get the status of the Transfer Complete flag. **Parameters:** *(Uint32**)
CSL_DMAX_QUERY_GET_TC	Get the status of TC. **Parameters:** *(* Uint32*)
CSL_DMAX_QUERY_GET_FMSC	Get the status of FIFO full. **Parameters:** *(* Uint32*)
CSL_DMAX_QUERY_GET_EMSC	Get the status of FIFO empty. **Parameters:** *(* Uint32*)
CSL_DMAX_QUERY_GET_FIFO_ERROR	Get the status of the FIFO error Flag/Interrupt. **Parameters:** *(* Uint32*)
CSL_DMAX_QUERY_GET_FIFO_ERROR_CODE	Get the status of the FIFO error Code. **Parameters:** *(* Uint32*)
CSL_DMAX_QUERY_GET_EVENT_ENTRY	Get the status of the Event entry of the specified Event. **Parameters:** *(* Uint32*)
CSL_DMAX_QUERY_GET_PARAMETER_ENTRY	Get the status of the parameter entry of the specified Event. **Parameters:** *(* Uint32*)
CSL_DMAX_QUERY_GET_HBC	Get the status of the High dmax. **Parameters:** *(* Uint32*)
CSL_DMAX_QUERY_GET_LBC	Get the status of the Low dmax. **Parameters:** *(* Uint32*)
CSL_DMAX_QUERY_GET_FIFO_ADDR	Get the status of the FIFO Address.

	Parameters: *(* Uint32*)
CSL_DMAX_QUERY_GET_EVENT_ENTRY_ADDR	Get the Event entry Address. **Parameters:** *(* Uint32*)
CSL_DMAX_QUERY_GET_PARAMETER_ENTRY_ADDR	Get the Parameter entry Address. **Parameters:** *(* CSL_DmaxParameterEntry*)
CSL_DMAX_QUERY_GET_FIFO_FULL	Get the FIFO full status. **Parameters:** *(* Uint32*)

3.4.2 CSL_DmaxHwControlCmd

enum CSL_DmaxHwControlCmd
Enumeration for queries passed to *CSL_dmaxHwControl()*.
This is used to select the commands to control the operations existing setup of DMAX. The arguments to be passed with each enumeration if any are specified next to the enumeration.

Enumeration values:

CSL_DMAX_CMD_SETPRIORITY	Set Priority of Event. **Parameters:** *(None)*
CSL_DMAX_CMD_SETPOLARITY	Sets Polarity of Event. **Parameters:** *(None)*
CSL_DMAX_CMD_EVENTENABLE	Enables Event. **Parameters:** *(None)*
CSL_DMAX_CMD_CLEARTCC	Clears TCC Event flag. **Parameters:** *(None)*
CSL_DMAX_CMD_EVENTDISABLE	Disables Event. **Parameters:** *(None)*
CSL_DMAX_CMD_CLEAREVENTENTRY	Clears the Event Entry. **Parameters:** *(Uint16 *)*
CSL_DMAX_CMD_CLEARPARAMENTRY	Clears Parameter Entry. **Parameters:** *(Uint16 *)*
CSL_DMAX_CMD_STARTASYNCTRANSFER	Set the start Sync Transfer. **Parameters:** *(None)*
CSL_DMAX_CMD_CLEARFIFOSTATUS	Set the FIFO status clear bit. **Parameters:** *(None)*
CSL_DMAX_CMD_WATERMARK_ENABLE	Set the Water Mark Enable.

	Parameters: *(None)*
CSL_DMAX_CMD_WATERMARK_DISABLE	Set the Water Mark Disable. **Parameters:** *(None)*

3.5 Macros

#define _CSL_DMAX_PARAM_ENTRY_SIZE (11)
DMAX Parameter Entry Size

#define CSL_DMAX_BAD_ESR_FLAG (0x000000FFu)
DMAX Bad ESR Flag

#define CSL_DMAX_CONFIG_DEFAULTS
Value:

```
{   \
                0x0,     \
                CSL_DMAX_EVENT0_ETYPE_FIFOWRITE, \
                0x0, \
                0x0, \
                0x0, \
                0x0, \
                0x0, \
                0x0, \
                0x0, \
                0x0, \
                0x0, \
                0x0  \
}
```

Default Values for Config structure

#define CSL_DMAX_ESR_FIFOERR_MASK (0x00000080u)
DMAX ESR Fifo Error

#define CSL_DMAX_ESR_FIFOERR_SHIFT (0x00000007u)
DMAX ESR Fifo Error

#define CSL_DMAX_EVENT_HI_PRIORITY (0x00000001u)
DMAX Event High Priority

#define CSL_DMAX_EVENT_LO_PRIORITY (0x00000000u)
DMAX Event Low Priority

#define CSL_DMAX_FIFO_FULL_MASK (0x01000000u)
DMAX FIFO Full

#define CSL_DMAX_FIFO_FULL_SHIFT (0x00000018u)
DMAX FIFO Full shift

#define CSL_DMAX_FIFO_NONSEQUENTIAL_UNDERFLOW_MASK (0x00000100u)
DMAX FIFO Non-sequential Underflow mask

#define CSL_DMAX_FIFO_NONSEQUENTIAL_UNDERFLOW_SHIFT (0x00000008u)
DMAX FIFO Non-sequential Underflow shift

#define CSL_DMAX_FIFO_OVERFLOW_MASK (0x00010000u)
DMAX FIFO OverFlow mask

#define CSL_DMAX_FIFO_OVERFLOW_SHIFT (0x00000010u)
DMAX FIFO Overflow shift

#define CSL_DMAX_FIFO_UNDERFLOW_MASK (0x00000001u)
DMAX FIFO UnderFlow mask

#define CSL_DMAX_FIFO_UNDERFLOW_SHIFT (0x00000000u)
DMAX FIFO Underflow shift

#define CSL_DMAX_GET_PARAM_ENTRY_OFFSET (uid)
(((uid) & 0xFFFFu)*_CSL_DMAX_PARAM_ENTRY_SIZE)
DMAX Get Parameter Entry offset

#define CSL_DMAX_HI_PRIORITY (0x00000001u)
DMAX High Priority

#define CSL_DMAX_HIPRIORITY_EVENT_ANY (0x1000FFFFu)
DMAX High Priority Event

#define CSL_DMAX_HIPRIORITY_PARAMETERENTRY_ANY (0x3000FFFFu)
DMAX High Priority Parameter entry

#define CSL_DMAX_ID (1)
DMAX Module ID

#define CSL_DMAX_LOPRIORITY_EVENT_ANY (0x2000FFFFu)
DMAX Low Priority Event

#define CSL_DMAX_LOPRIORITY_PARAMETERENTRY_ANY (0x4000FFFFu)
DMAX Low Priority Parameter entry

#define CSL_DMAX_POLARITY_FALLING_EDGE (0x00000000u)
DMAX polarity Falling Edge

#define CSL_DMAX_POLARITY_RISING_EDGE (0x00000001u)
DMAX Polarity Rising Edge

#define CSL_DMAX_PTE_BASE_OFFSET (0x00000028u)
DMAX Parameter table entry Base offset

#define CSL_DMAX_TCC_SPLIT (0x000000FFu)
DMAX Transfer Complete split

#define CSL_DMAX_WORD_SIZE (0x00000004u)
DMAX word size

#define CSL_EDMAX_BAD_ESR_FLAG (CSL_EDMAX_FIRST-1)
DMAX Bad ESR Flag

#define CSL_EDMAX_BAD_ETYPE (CSL_EDMAX_FIRST-2)
DMAX Bad Event Type

#define CSL_EDMAX_FIRST (-(((CSL_DMAX_ID+1)<<5)+1))
DMAX First

#define CSL_EDMAX_LAST (-((CSL_DMAX_ID+1)<<6))
DMAX Last

#define CSL_EDMAX_PARAM_ENTRY_FULL_ERROR (CSL_EDMAX_FIRST-0)
DMAX Parameter Entry Full Error

3.6 Typedef

typedef void * CSL_DmaxEventSetup
Dmax Event Setup

typedef void * CSL_DmaxParameterSetup
DMAX Prameter Setup

Chapter 4
EMIF Module

Topics

4.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within EMIF module.

The External Memory Interface (EMIF) is a TI developed re-usable IP component targeted for SoC designs. The EMIF provides a glue less interface to external memory devices including SDR and a wide variety of asynchronous devices.

The EMIF features supports following functionality:

- SDRAM Controller
- ASync Controller
- Nand Flash Controller
- Little Endian Operation
- Full Rate Operation

4.2 Functions

This section lists the functions available in the EMIF module.

4.2.1 CSL_emifInit

CSL_Status CSL_emifInit (CSL_EmifContext * *pContext*)

Description
This is the initialization function for the emif CSL. The function must be called before calling any other API from this CSL. This function does not modify any registers or check status. It returns status CSL_SOK. It has been kept for future use.

Arguments

```
pContext    Context information to EMIF
```

Return Value
CSL_Status

- CSL_SOK - Always returns

Pre Condition
CSL_sysInit() must be called.

Post Condition
None

Modifies
None

Example

```
CSL_Status       status;
...
status = CSL_emifInit(NULL);
```

4.2.2 CSL_emifOpen

CSL_EmifHandle CSL_emifOpen (CSL_EmifObj * *pEmifObj*,
CSL_InstNum *emifNum*,
CSL_EmifParam * *pEmifParam*,
CSL_Status * *pStatus*
)

Description
This function opens the EMIF CSL. It returns a handle to the CSL for the EMIF instance. This handle is passed to all other CSL APIs, as the reference to the EMIF instance.

Arguments

```
pEmifObj      Pointer to EMIF object.

emifNum       Instance of EMIF to be opened.

pEmifParam    Module specific parameters.

pStatus       The status of the function call.
```

Return Value
CSL_EmifHandle

- Valid EMIF handle will be returned if status value is equal to CSL_SOK.

Pre Condition
CSL_emifInit() must be called before calling this function.

Post Condition
1. The status is returned in the status variable. If status returned is

- CSL_SOK - Open call is successful
- CSL_ESYS_FAIL - Module instance is invalid.

2. EMIF object structure is populated.

Modifies
1. The status variable
2. EMIF object structure

Example

```
CSL_EmifObj      emifObj;
CSL_EmifHandle   hEmif;
CSL_EmifHwSetup  hwSetup;
CSL_Status       status;

...

hEmif = CSL_emifOpen (&emifObj, CSL_EMIF, NULL, &status);

...
```

4.2.3 CSL_emifClose

CSL_Status CSL_emifClose (CSL_EmifHandle *hEmif*)

Description
This function marks that CSL for the EMIF instance is closed. CSL for the EMIF instance need to be reopened before using any EMIF CSL API again.

Arguments

```
hEmif           Handle to EMIF
```

Return Value
CSL_Status

- CSL_SOK - EMIF is closed Successfully
- CSL_ESYS_BADHANDLE - The handle passed is invalid

Pre Condition
Both *CSL_emifInit()* and *CSL_emifOpen()* must be called successfully in order before calling *CSL_emifClose()*.

Post Condition
The external memory interface CSL APIs cannot be called until the external memory interface CSL is reopened again using CSL_emifOpen().

Modifies
Obj structure values

Example

```
CSL_EmifHandle   hEmif;
CSL_Status       status;
...

status = CSL_emifClose (hEmif);

...
```

4.2.4 CSL_emifGetHwSetup

CSL_Status CSL_emifGetHwSetup (CSL_EmifHandle *hEmif*, CSL_EmifHwSetup * *setup*)

Description
It retrieves the hardware setup parameters of the emif specified by the given handle.

Arguments

```
hEmif        Handle to the emif

setup      Pointer to the hardware setup structure
```

Return Value
CSL_Status

- CSL_SOK - Retrieving the hardware setup parameters is successful
- CSL_ESYS_BADHANDLE - The handle passed is invalid

- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition
Both *CSL_emifInit()* and *CSL_emifOpen()* must be called successfully in order before calling *CSL_emifGetHwSetup()*.

Post Condition
None

Modifies
Second parameter setup value

Example

```
CSL_EmifHandle   hEmif;
CSL_EmifHwSetup  hwSetup;
CSL_Status       status;
...

status = CSL_emifGetHwSetup(hEmif, &hwSetup);

...
```

4.2.5 CSL_emifHwControl

CSL_Status CSL_emifHwControl (CSL_EmifHandle ***hEmif***,
CSL_EmifHwControlCmd ***cmd***,
void * ***arg***
)

Description
This function performs various control operations on the EMIF, based on the command passed.

Arguments

```
hEmif        Handle to the EMIF instance

cmd          The command to this API indicates the action to be
             taken on EMIF.

arg          An optional argument.
```

Return Value
CSL_Status

- CSL_SOK - Status info return successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition

Both *CSL_emifInit()* and *CSL_emifOpen()* must be called successfully in order before calling *CSL_emifHwControl()* can be called. For the argument type that can be *void** casted and passed with a particular command refer to *CSL_EmifHwControlCmd.*

Post Condition
EMIFA registers are configured according to the command passed.

Modifies
EMIFA registers

Example

```
CSL_EmifHandle         hEmif;
CSL_EmifHwControlCmd   cmd = CSL_EMIF_CMD_SELF_REFRESH;
CSL_EmifSdramSelfRefresh arg = CSL_EMIF_SELF_REFRESH_DISABLE;
CSL_Status               status;
...
status = CSL_emifHwControl (hEmif, cmd, &arg);
```

4.2.6 CSL_emifGetHwStatus

CSL_Status CSL_emifGetHwStatus (CSL_EmifHandle ***hEmif***,
CSL_EmifHwStatusQuery ***query***,
void * ***response***
)

Description
Gets the status of the different operations of EMIF and depends on the query passed.

Arguments

```
hEmif      Handle to the EMIF instance

query      The query to this API of EMIF that indicates the
           status to be returned.

response   Placeholder to return the status.
```

Return Value
CSL_Status

- CSL_SOK - Status info return successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Invalid query command
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition
Both *CSL_emifInit()* and *CSL_emifOpen()* must be called successfully in order before calling *CSL_emifGetHwStatus()* can be called. For the argument type that can be *void** casted and passed with a particular command refer to *CSL_EmifHwStatusQuery.*

Post Condition
None

Modifies
Third parameter response value

Example

```
CSL_EmifHandle          hEmif;
CSL_EmifHwStatusQuery   query = CSL_EMIF_QUERY_REFRESH_RATE;
Uint16                  response;
CSL_Status       status;
...
status = CSL_emifGetHwStatus (hEmif, query, &response);
...
```

4.2.7 CSL_emifGetBaseAddress

CSL_Status CSL_emifGetBaseAddress (CSL_InstNum *emifNum*,
CSL_EmifParam * *pEmifParam*,
CSL_EmifBaseAddress * *pBaseAddress*
)

Description
This function gets the base address of the EMIF instance.

Arguments

```
emifNum         Instance number of the EMIF

pEmifParam      Module specific parameters.

pBaseAddress    Pointer to base address structure
```

Return Value
CSL_Status

- CSL_SOK - Successfully retrieved base address.
- CSL_ESYS_FAIL - Failed to get the base address
- CSL_ESYS_INVPARAMS - If the input parameter is not valid

Pre Condition
None

Post Condition
Base address structure is populated.

Modifies
1. The status variable
2. Base address structure

Example

```
   CSL_Status           status;
   CSL_EmifParam        emifParam;
   CSL_EmifBaseAddress  baseAddress;


   ...
   status     =     CSL_emifGetBaseAddress     (CSL_EMIF,     &emifParam,
&baseAddress);
   ...
```

4.2.8 CSL_emifHwSetup

CSL_Status CSL_emifHwSetup (CSL_EmifHandle *hEmif*, CSL_EmifHwSetup * *hwSetup*)

Description
It configures the emif registers to the values passed in the hardware setup structure.

Arguments

```
        hEmif           Handle to the emif

        hwSetup         Pointer to hardware setup structure
```

Return Value
CSL_Status

- CSL_SOK - Hardware setup successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Hardware structure is not properly initialized

Pre Condition
Both *CSL_emifInit()* and *CSL_emifOpen()* must be called successfully in order before calling this function.

Post Condition
EMIF controller registers are configured according to the hardware setup parameters.

Modifies
EMIF controller registers

Example

```
           CSL_EmifHandle   hEmif;
           CSL_EmifObj      emifObj;
           CSL_EmifHwSetup  hwSetup;
           CSL_Status       status;
```

```
...
hEmif = CSL_emifOpen(&emifObj, CSL_EMIF, NULL, &status);
status = CSL_emifHwSetup(hEmif, &hwSetup);
```

4.3 Data Structures

This section lists the data structures available in the EMIF module.

4.3.1 CSL_EmifObj

Detailed Description
EMIF object structure.

Field Documentation

CSL_InstNum CSL_EmifObj::emifNum
Instance of EMIF being referred by this object

CSL_EmifRegsOvly CSL_EmifObj::regs
Pointer to the register overlay structure of the EMIF

4.3.2 CSL_EmifContext

Detailed Description
Module specific context information. Present implementation of emif doesn't have any context information.

Field Documentation

Uint16 CSL_EmifContext::contextInfo
Context information of emif CSL. The declaration is just a placeholder for future implementation.

4.3.3 CSL_EmifHwSetup

Detailed Description
Top level h/w setup structure containing pointers to individual structures used for setup and the type of SDRAM used.

Field Documentation

CSL_EmifAsyncConfig CSL_EmifHwSetup::asyncBankCfg1
To configure Asynchronous Bank configuration

CSL_EmifSdramTimingConfig CSL_EmifHwSetup::emClkCycles
To configure SDRAM timing Configure Register

CSL_EmifIntrConfig CSL_EmifHwSetup::emifIntrCfg
To enable or disable interrupts on EMIF

Uint8 CSL_EmifHwSetup::maxExtWait
Maximum extended wait cycles

CSL_EmifNarrowMode CSL_EmifHwSetup::narrowMode
Data Bus width

Uint16 CSL_EmifHwSetup::refreshRate

Defines refresh rate for the SDRAM connected device

CSL_EmifSdramCasLatency CSL_EmifHwSetup::sdramCasLaten
SDRAM CAS Latency

CSL_EmifSdramInternalBankSetUp CSL_EmifHwSetup::sdramIntBankSetup
SDRAM Internal bank config

CSL_EmifSdramPageSize CSL_EmifHwSetup::sdramPageSize
SDRAM memory page size
`

Bool CSL_EmifHwSetup::sdramSelfRefreshEn
Self refresh enable/disable

Uint8 CSL_EmifHwSetup::txs
Minimum EM_CLK cycles from self refresh command to any command.

CSL_EmifWaitPolarity CSL_EmifHwSetup::waitPolarityCfg
Wait polarity for pad wait

CSL_EmifNandSetup CSL_EmifHwSetup::nandSetup
NAND flash configuration

4.3.4 CSL_EmifParam

Detailed Description
Module specific parameters. Present implementation of Emiff CSL doesn't have any module specific parameters.

Field Documentation

CSL_BitMask16 CSL_EmifParam::flags
Bit mask to be used for module specific parameters. The declaration is just a placeholder for future implementation.

4.3.5 CSL_EmifBaseAddress

Detailed Description
This structure contains the base-address information for the peripheral instance of the EMIF.

Field Documentation

CSL_EmifRegsOvly CSL_EmifBaseAddress::regs
Base-address of the configuration registers of the peripheral

4.3.6 CSL_EmifIntrConfig

Detailed Description
Used for enabling or disabling Interrupt operation for EMIF.

Field Documentation

Bool CSL_EmifIntrConfig::atIntrEn
Enables or Disables AT Interrupt on EMIF

4.3.7 CSL_EmifAsyncConfig

Detailed Description
Defines ths EMIF Async controller configuration.

Field Documentation

Uint8 CSL_EmifAsyncConfig::asize
Asynchronous Bank Size

Bool CSL_EmifAsyncConfig::ew
Extend Wait mode

Uint8 CSL_EmifAsyncConfig::r_hold
Read Strobe Hold cycles

Uint8 CSL_EmifAsyncConfig::r_setup
Read Strobe Setup cycles

Uint8 CSL_EmifAsyncConfig::r_strobe
Read Strobe Duration cycles

Bool CSL_EmifAsyncConfig::ss
Select Strobe mode

Uint8 CSL_EmifAsyncConfig::ta
Turn Around cycles

Uint8 CSL_EmifAsyncConfig::w_hold
Write Strobe Hold cycles

Uint8 CSL_EmifAsyncConfig::w_setup
Write Strobe Setup cycles

Uint8 CSL_EmifAsyncConfig::w_strobe
Write Strobe Duration cycles

4.3.8 CSL_EmifSdramTimingConfig

Detailed Description
Defines the SDRAM timing register bit fields for EM_CLK configurations

Field Documentation

Uint8 CSL_EmifSdramTimingConfig::tras
Min number of EM_CLK cycles from Activate to Precharge minus 1

Uint8 CSL_EmifSdramTimingConfig::trc
Min number of EM_CLK cycles from Activate to Activate minus 1

Uint8 CSL_EmifSdramTimingConfig::trcd
Min number of EM_CLK cycles from Activate to Read / Write minus 1

Uint8 CSL_EmifSdramTimingConfig::trfc
Min number of EM_CLK cycles from Refresh to Refresh minus 1

Uint8 CSL_EmifSdramTimingConfig::trp
Min number of EM_CLK cycles from Precharge to Activate or Refresh minus 1

Uint8 CSL_EmifSdramTimingConfig::trrd
Min number of EM_CLK cycles from Activate to Activate of different bank minus 1

Uint8 CSL_EmifSdramTimingConfig::twr
Min number of EM_CLK cycles from last write transfer to Precharge minus 1

4.3.9 CSL_EmifNandSetup

Detailed Description
This structure contains the information for the peripheral interface of NAND device

Field Documentation

Uint8 CSL_CSL_EmifNandSetup:: ecc_support
Controls the ECC calculation

Uint8 CSL_CSL_EmifNandSetup:: nand_support
Controls the Nand support is required or not

4.4 Enumerations

This section lists the enumerations available in the EMIF module.

4.4.1 CSL_EmifSdramSelfRefresh

enum CSL_EmifSdramSelfRefresh
This enum used in SR bit field of SDRAM Bank Config Register. Can set SDRAM to enter or exit self refresh mode.

Enumeration values:

CSL_EMIF_SELF_REFRESH_ENABLE	SDRAM exit Self refresh mode
CSL_EMIF_SELF_REFRESH_DISABLE	SDRAM enter Self refresh mode

4.4.2 CSL_EmifSdramCasLatency

enum CSL_EmifSdramCasLatency
This enum used to set the CAS latency to be used when accessing connected SDRAM devices.

Enumeration values:

CSL_EMIF_CAS_LATENCY_SET_TO_2	CAS latency = 2 EM_CLK cycles
CSL_EMIF_CAS_LATENCY_SET_TO_3	CAS latency = 3 EM_CLK cycles

4.4.3 CSL_EmifSdramInternalBankSetUp

enum CSL_EmifSdramInternalBankSetUp
This enum used in setting the bit field ibank of SDRAM Bank Config Register to set the number of SDRAM internal banks.

Enumeration values:

CSL_EMIF_1_SDRAM_BANKS	1 Bank SDRAM device
CSL_EMIF_2_SDRAM_BANKS	2 Banks SDRAM device
CSL_EMIF_4_SDRAM_BANKS	4 Banks SDRAM device

4.4.4 CSL_EmifHwControlCmd

enum CSL_EmifHwControlCmd
This enum used define the command required for EMIF registers setup.

Enumeration values:

CSL_EMIF_CMD_SELF_REFRESH	Enables self refresh
CSL_EMIF_CMD_SELF_REFRESH_RATE	Enables self refresh
CSL_EMIF_CMD_CLEAR_ASYNC_TIMEOUT	Clears Async Timeout
CSL_EMIF_CMD_CLEAR_MASKED_ASYNC_TIMEOUT	Clears Masked Async Timeout
CSL_EMIF_CMD_SET_ASYNC_TIMEOUT_INTR	Set Async Timeout Interrupt
CSL_EMIF_CMD_CLR_ASYNC_TIMEOUT_INTR	Clear Async Timeout Interrupt

4.4.5 CSL_EmifHwStatusQuery

enum CSL_EmifHwStatusQuery
This enum used define the query required for EMIF registers status.

Enumeration values:

CSL_EMIF_QUERY_SELF_REFRESH	Get status if SDRAM is in Self refresh
CSL_EMIF_QUERY_REFRESH_RATE	Get the Refresh rate of SDRAM
CSL_EMIF_QUERY_ASYNC_TIMEOUT_STATUS	Get timeout status
CSL_EMIF_QUERY_MASKED_ASYNC_TIMEOUT_STATUS	Get masked timeout status
CSL_EMIF_QUERY_MASKED_ASYNC_TIMEOUT_INTR_STATUS	Get masked timeout Intr status

4.4.6 CSL_EmifSdramPageSize

enum CSL_EmifSdramPageSize
This enum used in setting the bit field page size of SDRAM Bank Config Register.

Enumeration values:

CSL_EMIF_256WORD_8COL_ADDR	8 column address bits (256 elements per row)
CSL_EMIF_512WORD_9COL_ADDR	9 column address bits (512 elements per row)
CSL_EMIF_1024WORD_10COL_ADDR	10 column address bits (1024 elements per row)

4.4.7 CSL_EmifNarrowMode

enum CSL_EmifNarrowMode
This enum used to define data bus width usage.
It can be in either 32 Bit or 16 Bit modes.

Enumeration values:

CSL_EMIF_CAS_LATENCY_SET_TO_2	CAS latency = 2 EM_CLK cycles
CSL_EMIF_CAS_LATENCY_SET_TO_3	CAS latency = 3 EM_CLK cycles

4.4.8 CSL_EmifWaitPolarity

enum CSL_EmifWaitPolarity
This enum used to define the polarity of the AWAIT pin.

Enumeration values:

CSL_EMIF_WAIT_POLARITY_LOW	Insert wait cycles if AWAIT is low
CSL_EMIF_WAIT_POLARITY_HIGH	Insert wait cycles if AWAIT is high

4.4.9 CSL_EmifNANDStatus

enum CSL_EmifNANDStatus

his enum used to Enable or disable NAND Flash usage.

Enumeration values:

CSL_EMIF_NAND_DISABLE	Disable NAND
CSL_EMIF_NAND_ENABLE	Enable NAND

4.4.10 CSL_EmifNANDECCStatus

enum CSL_EmifNANDECCStatus
his enum used to Enable or disable NAND Flash usage.

Enumeration values:

CSL_EMIF_NAND_ECC_START	Start NAND ECC Calculation
CSL_EMIF_NAND_NO_ECC	Do not Start NAND ECC Calculation

4.5 Macros

#define CSL_EMIF_HWSETUP_DEFAULTS
Value:

```
{ \
    CSL_EMIF_SELF_REFRESH_DISABLE,\
    CSL_EMIF_256WORD_8COL_ADDR,\
    CSL_EMIF_1_SDRAM_BANKS,\
    CSL_EMIF_CAS_LATENCY_SET_TO_2,\
    CSL_EMIF_WAIT_POLARITY_HIGH,\
    0x0,\
    CSL_EMIF_16BIT_USAGE,\
    0x0,\
    0x0,\
    0x0,\
    {0,0,0x2,0x01,0x0c,0x0f,0x0},\
    {0,0,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x0B},\
    { CSL_EMIF_NAND_DISABLE, CSL_EMIF_NAND_NO_ECC}\
}
```

Chapter 5
I2C Module

Topics

5.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within I2C module.

The catalog DSP multi-master I 2 C peripheral provides an interface between a TI DSP device and I2C-bus compatible devices that are connected via the I2C serial bus. External components attached to the I2C bus can serially transmit/receive up to 8-bit data to/from the TI DSP device through the two-wire I 2 C interface.

The following are I2C features supported:

- Compliance of Philips I2C specification (reference to Philips I2C specification v2.1)
  - Byte format transfer
  - 7 bit and 10-bit device addressing modes
  - General call
  - Start byte
  - Multi-master transmitter/slave receiver mode
  - Multi-master receiver/slave transmitter mode
  - Combined master transmit/receive and receive/transmit mode
  - I2C data transfer rate of from 10kbps up to 400kbps (Philips I 2 C rate)
- 2 to 7 bit format transfer
- Free data format
- Has one read and one write DMA event that can be used by the DMA
- Has seven interrupts that can be used by the CPU
- Operate with DSP core frequency from 12 MHz up
- Operate with module frequency of 12 MHz
- Interface to V-bus (32-bit synchronously slave bus)
- Module enable/disable capability
- Conform to the Peripheral Design Guide Lines

5.2 Functions

This section lists the functions available in the I2C module.

5.2.1 CSL_i2cInit

CSL_Status CSL_i2cInit (CSL_I2cContext * *pContext*)

Description
This is the initialization function for the I2C. This function is idempotent in that calling it many times is same as calling it once. This function doesn't affect the H/W.

Arguments

```
       pContext   Context information for the instance. Should be NULL
```

Return Value
CSL_Status

- CSL_SOK - Always returns

Pre Condition
CSL_sysInit() must be called.

Post Condition
The CSL for I2C is initialized

Modifies
None

Example

```
    CSL_Status       status;
    ...
    status = CSL_i2cInit(NULL);
```

5.2.2 CSL_i2cOpen

CSL_I2cHandle CSL_i2cOpen

(
CSL_I2cObj * ***pI2cObj*****,**
CSL_InstNum ***i2cNum*****,**
CSL_I2cParam * ***pI2cParam*****,**
CSL_Status * ***pStatus***
)

Description
This function populates the peripheral data object for the instance of the I2C device and returns a handle to the instance. The device can be re-opened anytime after it has been normally closed if

so required. The handle returned by this call is input as an essential argument for rest of the APIs described for this module.

Arguments

```
pI2cObj     Pointer to the I2C instance object

i2cNum      Instance of the I2C to be opened

pI2cParam   Pointer to module specific parameters

pStatus     Pointer for returning status of the function call
```

Return Value
CSL_I2cHandle

- Valid I2C instance handle will be returned if status value is equal to CSL_SOK else CSL_I2C_BADHANDLE

Pre Condition
CSL_i2cInit() must be called successfully.

Post Condition
1. The status is returned in the status variable. If status returned is

- CSL_SOK - Open call is successful
- CSL_ESYS_INVPARAMS - If invalid parameter passed
- CSL_ESYS_FAIL - If invalid instance number

2. I2C object structure is populated

Modifies

- The status variable
- I2C object structure

Example

```
CSL_Status        status;
CSL_I2cObj        i2cObj;
CSL_I2cHandle     hI2c;

hI2c = CSL_i2cOpen (&i2cObj,
                    CSL_I2C_0,
                    NULL,
                    &status
                    );
```

5.2.3 CSL_i2cClose

CSL_Status CSL_i2cClose (CSL_I2cHandle *hI2c*)

Description
This function closes the specified instance of I2C.

Arguments

```
hI2c    Handle to the I2C instance
```

Return Value
CSL_Status

- CSL_SOK - I2C close successful
- CSL_ESYS_BADHANDLE - Invalid handle

Pre Condition
Both *CSL_i2cInit()* and *CSL_i2cOpen()* must be called successfully in order before calling *CSL_i2cClose()*.

Post Condition
The I2C CSL APIs can not be called until the I2C CSL is reopened again using *CSL_i2cOpen()*.

Modifies
CSL_I2cObj structure values for the instance

Example

```
CSL_I2cHandle   hI2c;
CSL_Status      status;
...
status = CSL_i2cClose(hI2c);
```

5.2.4 CSL_i2cHwSetup

CSL_Status CSL_i2cHwSetup (CSL_I2cHandle *hI2c*, CSL_I2cHwSetup * *setup*)

Description
This function initializes the device registers with the appropriate values provided through the HwSetup Data structure. After the Setup is completed, the device is ready for operation. For information passed through the HwSetup Data structure, refer *CSL_i2cHwSetup*.

Arguments

```
hI2c   handle to the I2c instance

setup  Pointer to setup structure that programs I2C
```

Return Value
CSL_Status

- CSL_SOK - Successful completion of hardware setup
- CSL_ESYS_ INVPARAMS - Hardware structure is not properly initialized
- CSL_ESYS_ BADHANDLE - Invalid CSL handle

Pre Condition
Both *CSL_i2cInit()* and *CSL_i2cOpen()* must be called successfully in order before calling this function. The user has to allocate space for and fill in the main setup structure appropriately before calling this function.

Post Condition
I2C registers are configured according to the hardware setup parameters.

Modifies
I2C registers will be setup according to value passed.

Example

```
CSL_i2cHandle  hI2c;
CSL_i2cHwSetup hwSetup;
CSL_Status       status;
...

hwSetup.mode           = CSL_I2C_MODE_MASTER;
hwSetup.dir            = CSL_I2C_DIR_TRANSMIT;
hwSetup.addrMode       = CSL_I2C_ADDRSZ_SEVEN;
.
.
hwSetup.clksetup       = &clksetup;

CSL_i2cHwSetup(hI2c, &hwSetup);
```

5.2.5 CSL_i2cGetHwSetup

CSL_Status CSL_i2cGetHwSetup (CSL_I2cHandle *hI2c*, CSL_I2cHwSetup * *setup*)

Description
It retrieves the hardware setup parameters of the I2C specified by the given handle.

Arguments

```
hI2c        Handle to the I2C

setup       Pointer to the hardware setup structure
```

Return Value
CSL_Status

- CSL_SOK - Retrieving the hardware setup parameters is successful

- CSL_ESYS_BADHANDLE - The handle is passed is invalid
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition
Both *CSL_i2cInit()* and *CSL_i2cOpen()* must be called successfully in order before calling this function.

Post Condition
The hardware setup structure is populated with the hardware setup parameters.

Modifies
Second Parameter setup value

Example

```
CSL_I2cHandle   hI2c;
CSL_I2cHwSetup  hwSetup;
CSL_Status      status;
...
status = CSL_i2cGetHwSetup(hI2c, &hwSetup);
...
```

5.2.6 CSL_i2cHwControl

CSL_Status CSL_i2cHwControl (CSL_I2cHandle ***hI2c***,
CSL_I2cHwControlCmd ***cmd***,
void * ***arg***
)

Description
Takes a command of I2C with an optional argument and implements it.

Arguments

hI2c	Handle to the I2C instance
cmd	The command to this API indicates the action to be taken on I2C
arg	An optional argument.

Return Value
CSL_Status

- CSL_SOK - Status info return successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition

Both *CSL_i2cInit()* and *CSL_i2cOpen()* must be called successfully in order before calling this function.

Post Condition
I2C registers are configured according to the command passed

Modifies
The hardware registers of I2C.

Example

```
CSL_I2cHandle         hI2c;
CSL_I2cHwControlCmd   cmd;
void*                 arg;
CSL_Status            status;
...
cmd = CSL_I2C_CMD_RESET;
status = CSL_i2cHwControl (hI2c, cmd, &arg);
...
```

5.2.7 CSL_i2cRead

CSL_Status CSL_i2cRead (CSL_I2cHandle *hI2c*, void * *buf*)

Description
Reads the received data from the I2C Data Receive register

Arguments

```
hI2c   Handle of already opened peripheral

buf    Pointer to memory where data will be read and stored
```

Return Value
CSL_Status

- CSL_SOK - Data successfully extracted
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition
Both *CSL_i2cInit()* and *CSL_i2cOpen()* must be called successfully in order before calling this function.

Post Condition
RSFULL flag bit in I2CSTR register is cleared,
ICRRDY flag bit in I2CSTR register is cleared.

Modifies
buf variable holds the received data

Example

```
Uint16        inData;
CSL_Status    status;
CSL_I2cHandle hI2c;
...
// I2C object defined and HwSetup structure defined and
   initialized
...

// Init, Open, HwSetup successfully done in that order
...

status = CSL_i2cRead(hI2c, &inData);
```

5.2.8 CSL_i2cWrite

CSL_Status CSL_i2cWrite (CSL_I2cHandle *hI2c*, void * *buf*)

Description
Writes the specified data into I2C Data Transmit register.

Arguments

```
hI2c   Handle of already opened peripheral

buf    Pointer to data to be written
```

Return Value
CSL_Status

- CSL_SOK - Data successfully written
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition
Both *CSL_i2cInit()* and *CSL_i2cOpen()* must be called successfully in order before calling this function.

Post Condition
XSMT flag bit in I2CSTR register is set ICXRDY flag bit in I2CSTR register is cleared.

Modifies
ICDXR register

Example:

```
Uint16        inData;
CSL_Status    status;
```

```
CSL_I2cHandle hI2c;
...
// I2C object defined and HwSetup structure defined and
   initialized
...

// Init, Open, HwSetup successfully done in that order
...

status = CSL_i2cWtrite(hI2c, &inData);
```

5.2.9 CSL_i2cHwSetupRaw

CSL_Status CSL_i2cHwSetupRaw **(** CSL_I2cHandle ***hI2c***, CSL_I2cConfig ***** ***config*** **)**

Description
This function configures the icache using the register-values passed through the config-structure.

Arguments

```
hI2c        Handle to the I2C

config      Pointer to config structure
```

Return Value
CSL_Status

- CSL_SOK - Configuration successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Configuration is not properly initialized

Pre Condition
Both *CSL_i2cInit()* and *CSL_i2cOpen()* must be called successfully in order before calling this function.

Post Condition
The registers of the specified I2C instance will be setup according to value passed.

Modifies
Hardware registers of the specified I2C instance.

Example

```
CSL_I2cHandle       hI2c;
CSL_I2cConfig       config = CSL_I2C_CONFIG_DEFAULTS;
CSL_Status          status;

status = CSL_i2cHwSetupRaw (hI2c, &config);
```

5.2.10 CSL_i2cGetHwStatus

CSL_Status CSL_i2cGetHwStatus (CSL_I2cHandle *hI2c*,
CSL_I2cHwStatusQuery *query*,
void * *response*
)

Description
Gets the status of the different operations of I2C.

Arguments

```
hI2c        Handle to the I2C instance

query       The query to this API of I2C that indicates the
            status to be returned

response    Placeholder to return the status.
```

Return Value
CSL_Status

- CSL_SOK - Status info return successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Invalid query command
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition
Both *CSL_i2cInit()* and *CSL_i2cOpen()* must be called successfully in order before calling *CSL_i2cGetHwStatus()*.

Post Condition
None

Modifies
Third parameter, response value

Example

```
CSL_I2cHandle          hI2c;
CSL_I2cHwStatusQuery   query;
void*                  response;
CSL_Status             status;
...
status = CSL_Geti2cHwStatus (hI2c, query, &response);
```

5.2.11 CSL_i2cGetBaseAddress

CSL_Status CSL_i2cGetBaseAddress (CSL_InstNum *i2cNum*,

CSL_I2cParam * *pI2cParam*,

CSL_I2cBaseAddress * *pBaseAddress*

)

Description
Function to get the base address of the peripheral instance. This function is used for getting the base address of the peripheral instance. This function will be called inside the CSL_i2cOpen() function call.

Arguments

```
i2cNum        Specifies the instance of the I2C to be opened

pI2cParam     Module specific parameters

pBaseAddress  Pointer to baseaddress structure containing base
              address details
```

Return Value
CSL_Status

- CSL_SOK Open call is successful
- CSL_ESYS_FAIL The instance number is invalid.
- CSL_ESYS_INVPARAMS Invalid parameter

Pre Condition
None

Post Condition
Base Address structure is populated.

Modifies
1. The status variable
2. Base address structure is modified.

Example

```
CSL_Status          status;
CSL_I2cBaseAddress  baseAddress;

...
status = CSL_i2cGetBaseAddress(CSL_I2C_0, NULL, &baseAddress);
...
```

5.3 Data Structures

This section lists the data structures available in the I2C module.

5.3.1 CSL_I2cObj

Detailed Description
This object contains the reference to the instance of I2C opened using the *CSL_i2cOpen()*.
The pointer to this is passed to all I2C CSL APIs.

Field Documentation

CSL_InstNum CSL_I2cObj::perNum
This is the instance of I2C being referred to by this object

CSL_I2cRegsOvly CSL_I2cObj::regs
Base-address of the Configuration registers of I2C.

5.3.2 CSL_I2cParam

Detailed Description
I2C specific parameters. Present implementation doesn't have any specific parameters.

Field Documentation

CSL_BitMask16 CSL_I2cParam::flags
Bit mask to be used for module specific parameters. The declaration is just a placeholder for future implementation.

5.3.3 CSL_I2cContext

Detailed Description
I2C specific context information. Present implementation doesn't have any Context information.

Field Documentation

Uint16 CSL_I2cContext::contextInfo
Context information of I2C. The declaration is just a placeholder for future implementation.

5.3.4 CSL_I2cConfig

Detailed Description
Config structure of I2C. This is used to configure I2C using CSL_HwSetupRaw function. This is a structure of register values, rather than a structure of register *field* values like CSL_I2cHwSetup.

Field Documentation

volatile Uint16 CSL_I2cConfig::ICCLKH
I2C Clock Divider High Register

volatile Uint16 CSL_I2cConfig::ICCLKL
I2C Clock Divider Low Register

volatile Uint16 CSL_I2cConfig::ICCNT
I2C Data Count Register

volatile Uint16 CSL_I2cConfig::ICDXR
I2C Data Transmit Register

volatile Uint16 CSL_I2cConfig::ICEMDR
I2C Extended Mode Register

volatile Uint16 CSL_I2cConfig::ICIMR
I2C Interrupt Mask Register

volatile Uint16 CSL_I2cConfig::ICIVR
I2C Interrupt Vector Register

volatile Uint16 CSL_I2cConfig::ICMDR
I2C Mode Register

volatile Uint16 CSL_I2cConfig::ICOAR
I2C Own Address Register

volatile Uint32 CSL_I2cConfig::ICPDCLR
I2C Pin Data Clear Register

volatile Uint32 CSL_I2cConfig::ICPDIR
I2C Pin Direction Register

volatile Uint32 CSL_I2cConfig::ICPDOUT
I2C Pin Data Out Register

volatile Uint32 CSL_I2cConfig::ICPDSET
I2C Pin Data Set Register

volatile Uint32 CSL_I2cConfig::ICPFUNC
I2C Pin Function Register

volatile Uint16 CSL_I2cConfig::ICPSC
I2C Prescaler Register

volatile Uint16 CSL_I2cConfig::ICSAR
I2C Slave Address Register

volatile Uint16 CSL_I2cConfig::ICSTR
I2C Status Register

5.3.5 CSL_I2cClkSetup

Detailed Description
This has all the fields required to configure the I2C clock.

Field Documentation

Uint16 CSL_I2cClkSetup::clkhighdiv
High time period of the clock

Uint16 CSL_I2cClkSetup::clklowdiv
Low time period of the clock

Uint16 CSL_I2cClkSetup::prescalar
Prescalar to the input clock

5.3.6 CSL_I2cHwSetup

Detailed Description
This has all the fields required to configure I2C at Power Up (After a Hardware Reset) or a Soft Reset.
This structure is used to setup or obtain existing setup of I2C using *CSL_i2cHwSetup()* and *CSL_i2cGetHwSetup()* functions respectively.

Field Documentation

Uint16 CSL_I2cHwSetup::ackMode
ACK mode while receiver: 0==> ACK Mode, 1==> NACK Mode

Uint16 CSL_I2cHwSetup::addrMode
Addressing Mode: 0==> 7-bit Mode, 1==> 10-bit Mode

Uint16 CSL_I2cHwSetup::bcm
I2C Backward Compatibility Mode: 0==> Not compatible, 1==> Compatible

CSL_I2cClkSetup* CSL_I2cHwSetup::clksetup
Prescalar, Clock Low Time and Clock High Time for Clock Setup

Uint16 CSL_I2cHwSetup::dir
Transmitter Mode or Receiver Mode: 1==> Transmitter Mode, 0 ==> Receiver Mode

Uint16 CSL_I2cHwSetup::freeDataFormat
Free Data Format of I2C: 0==>Free data format disable, 1==> Free data format enable

Uint16 CSL_I2cHwSetup::inten
Interrupt Enable mask. The mask can be for one interrupt or for multiple interrupts.

Uint16 CSL_I2cHwSetup::loopBackMode
DLBack mode of I2C (master tx-er only): 0==> No loopback, 1==> Loopback Mode

Uint16 CSL_I2cHwSetup::mode
Master or Slave Mode: 1==> Master Mode, 0==> Slave Mode

Uint16 CSL_I2cHwSetup::ownaddr
Address of the own device

Uint16 CSL_I2cHwSetup::repeatMode
Repeat Mode of I2C: 0==> No repeat mode 1==> Repeat mode

Uint16 CSL_I2cHwSetup::resetMode
I2C Reset Mode: 0==> Reset, 1==> Out of reset

Uint16 CSL_I2cHwSetup::runMode
Run mode of I2C: 0==> No Free Run, 1==> Free Run mode

Uint16 CSL_I2cHwSetup::sttbyteen
Start Byte Mode: 1 ==> Start Byte Mode, 0 ==> Normal Mode

5.3.7 CSL_I2cBaseAddress

Detailed Description
This will have the base-address information for the peripheral instance.

Field Documentation

CSL_I2cRegsOvly CSL_I2cBaseAddress::regs
Base-address of the Configuration registers of I2C.

5.4 Enumerations

This section lists the enumerations available in the I2C module.

5.4.1 CSL_I2cHwControlCmd

enum CSL_I2cHwControlCmd
Enumeration for queries passed to *CSL_i2cHwControl()*.
This is used to select the commands to control the operations existing setup of I2C. The arguments to be passed with each enumeration if any are specified next to the enumeration.

Enumeration values:

CSL_I2C_CMD_ENABLE	Enable the I2C. **Parameters:** *(None)*
CSL_I2C_CMD_CONFIG_AS_GIO	Configure the I2C pins as GPIO. **Parameters:** *(None)*
CSL_I2C_CMD_RESET	Reset command to the I2C. **Parameters:** *(None)*
CSL_I2C_CMD_OUTOFRESET	Bring the I2C out of reset. **Parameters:** *(None)*
CSL_I2C_CMD_CLEAR_STATUS	Clear all the status bits that are set. **Parameters:** *(None)*
CSL_I2C_CMD_SET_SLAVE_ADDR	Set the address of the Slave device. **Parameters:** *(Uint16 *)*
CSL_I2C_CMD_SET_DATA_COUNT	Set the Data Count. **Parameters:** *(Uint16 *)*
CSL_I2C_CMD_START	Set the start condition. **Parameters:** *(None)*
CSL_I2C_CMD_STOP	Set the stop condition. **Parameters:** *(None)*
CSL_I2C_CMD_DIR_TRANSMIT	Set the transmission mode. **Parameters:** *(None)*
CSL_I2C_CMD_DIR_RECEIVE	Set the receiver mode. **Parameters:** *(None)*
CSL_I2C_CMD_RM_ENABLE	Set the Repeat Mode. **Parameters:** *(None)*

CSL_I2C_CMD_RM_DISABLE	Disable the Repeat Mode. **Parameters:** *(None)*
CSL_I2C_CMD_DLB_ENABLE	Set the loop back mode. **Parameters:** *(None)*
CSL_I2C_CMD_DLB_DISABLE	Set the loop back mode. **Parameters:** *(None)*
CSL_I2C_CMD_INTR_ENABLE	Unmask all interrupts. **Parameters:** *(None)*
CSL_I2C_CMD_INTR_DISABLE	Mask all interrupts. **Parameters:** *(None)*
CSL_I2C_CMD_SET_OWN_ADDR	Set the own address. **Parameters:** *(Uint16 *)*
CSL_I2C_CMD_SET_CLOCK	Set the Clock. **Parameters:** *(CSL_I2cClkSetup *)*
CSL_I2C_CMD_SET_MODE	Set the mode(Master/Slave) **Parameters:** (Uint16 *)

5.4.2 CSL_I2cHwStatusQuery

enum CSL_I2cHwStatusQuery
Enumeration for queries passed to *CSL_i2cGetHwStatus()*.
This is used to get the status of different operations or to get the existing setup of I2C.

Enumeration values:

CSL_I2C_QUERY_CLOCK_SETUP	Get current clock setup parameters. **Parameters:** *(CSL_I2cClkSetup*)*
CSL_I2C_QUERY_BUS_BUSY	Get the Bus Busy status information. **Parameters:** *(Uint16*)*
CSL_I2C_QUERY_RX_RDY	Get the Receive Ready status information. **Parameters:** *(Uint16*)*
CSL_I2C_QUERY_TX_RDY	Get the Transmit Ready status information. **Parameters:** *(Uint16*)*
CSL_I2C_QUERY_ACS_RDY	Get the Register Ready status information. **Parameters:** *(Uint16*)*
CSL_I2C_QUERY_SCD	Get the Stop Condition Data bit information.

	Parameters: *(Uint16*)*
CSL_I2C_QUERY_AD0	Get the Address Zero Status (General Call) detection status. **Parameters:** *(Uint16*)*
CSL_I2C_QUERY_RSFULL	Get the Receive overflow status information. **Parameters:** *(Uint16*)*
CSL_I2C_QUERY_XSMT	Get the Transmit underflow status information. **Parameters:** *(Uint16*)*
CSL_I2C_QUERY_AAS	Get the Address as Slave bit information. **Parameters:** *(Uint16*)*
CSL_I2C_QUERY_AL	Get the Arbitration Lost status information. **Parameters:** *(Uint16*)*
CSL_I2C_QUERY_RDONE	Get the Reset Done status bit information. **Parameters:** *(Uint16*)*
CSL_I2C_QUERY_BITCOUNT	Get no of bits of next byte to be received or transmitted. **Parameters:** *(Uint16*)*
CSL_I2C_QUERY_INTCODE	Get the interrupt code for the interrupt that occurred. **Parameters:** *(Uint16*)*
CSL_I2C_QUERY_REV	Get the revision level of the I2C. **Parameters:** *(Uint16*)*
CSL_I2C_QUERY_CLASS	Get the class of the peripheral. **Parameters:** *(Uint16*)*
CSL_I2C_QUERY_TYPE	Get the type of the peripheral. **Parameters:** *(Uint16*)*
CSL_I2C_QUERY_SDIR	Get the slave direction. **Parameters:** *(Uint16*)*
CSL_I2C_QUERY_NACKSNT	Get the acknowledgement status. **Parameters:** *(Uint16*)*

5.5 Macros

#define CSL_I2C_ACK_DISABLE (1)
For enabling the tx of a NACK to the TX-ER, while in the RECEIVER mode

#define CSL_I2C_ACK_ENABLE (0)
For enabling the tx of a ACK to the TX-ER, while in the RECEIVER mode

#define CSL_I2C_ACS_NOT_READY (0)
For indicating that the Access ready signal is low

#define CSL_I2C_ACS_READY (1)
For indicating that the Access ready signal is high

#define CSL_I2C_ADDRSZ_SEVEN (0)
For setting the 7-bit Addressing Mode for I2C

#define CSL_I2C_ADDRSZ_TEN (1)
For setting the 10-bit Addressing Mode

#define CSL_I2C_ARBITRATION_LOST (1)
For indicating Arbitration Lost signal is set

#define CSL_I2C_BCM_DISABLE (0)
For disabling the Backward Compatibility mode of I2C

#define CSL_I2C_BCM_ENABLE (1)
For enabling the Backward Compatibility mode of I2C

#define CSL_I2C_BUS_BUSY (1)
For indicating that the bus is busy

#define CSL_I2C_BUS_NOT_BUSY (0)
For indicating that the bus is not busy

#define CSL_I2C_CLEAR_AL 0x1
Clear the Arbitration Lost status bit

#define CSL_I2C_CLEAR_ARDY 0x4
Clear the Register access ready status bit

#define CSL_I2C_CLEAR_NACK 0x2
Clear the No acknowledge status bit

#define CSL_I2C_CLEAR_RRDY 0x8
Clear the Receive ready status bit

#define CSL_I2C_CLEAR_SCD 0x20
Clear the Stop Condition Detect status bit

#define CSL_I2C_CLEAR_XRDY 0x10
Clear the Transmit ready status bit

#define CSL_I2C_CONFIG_DEFAULTS
Value:

```
{   \
        CSL_I2C_I2COAR_RESETVAL,     \
        CSL_I2C_I2CIMR_RESETVAL,     \
        CSL_I2C_I2CSTR_RESETVAL,     \
        CSL_I2C_I2CCLKL_RESETVAL,    \
        CSL_I2C_I2CCLKH_RESETVAL,    \
        CSL_I2C_I2CCNT_RESETVAL,     \
        CSL_I2C_I2CSAR_RESETVAL,     \
        CSL_I2C_I2CDXR_RESETVAL,     \
        CSL_I2C_I2CMDR_RESETVAL,     \
        CSL_I2C_I2CIVR_RESETVAL,     \
        CSL_I2C_I2CEMDR_RESETVAL,    \
        CSL_I2C_I2CPSC_RESETVAL,     \
        CSL_I2C_I2CPFUNC_RESETVAL,   \
        CSL_I2C_I2CPDIR_RESETVAL,    \
        CSL_I2C_I2CPDOUT_RESETVAL,   \
        CSL_I2C_I2CPDSET_RESETVAL,   \
        CSL_I2C_I2CPDCLR_RESETVAL    \
}
```

Default Values for Config structure

#define CSL_I2C_DIR_RECEIVE (0)
For setting the RECEIVER Mode for I2C

#define CSL_I2C_DIR_TRANSMIT (1)
For setting the TRANSMITTER Mode for I2C

#define CSL_I2C_DLB_DISABLE (0)
For disabling DLB mode of I2C (applicable only in case of MASTER TX-ER)

#define CSL_I2C_DLB_ENABLE (1)
For enabling DLB mode of I2C (applicable only in case of MASTER TX-ER)

#define CSL_I2C_FDF_DISABLE (0)
For disabling the Free Data Format of I2C

#define CSL_I2C_FDF_ENABLE (1)
For enabling the Free Data Format of I2C

#define CSL_I2C_FREE_MODE_DISABLE (0)
For disabling the free run mode of the I2C

#define CSL_I2C_FREE_MODE_ENABLE (1)
For enabling the free run mode of the I2C

#define CSL_I2C_IRS_DISABLE (1)
For taking the I2C out of Reset

#define CSL_I2C_IRS_ENABLE (0)
For putting the I2C in Reset

#define CSL_I2C_MODE_MASTER (1)
For setting the MASTER Mode for I2C

#define CSL_I2C_MODE_SLAVE (0)
For setting the SLAVE Mode for I2C

#define CSL_I2C_RECEIVE_OVERFLOW (1)
For indicating Receive Overflow signal is set

#define CSL_I2C_REPEAT_MODE_DISABLE (0)
For disabling the Repeat Mode of the I2C

#define CSL_I2C_REPEAT_MODE_ENABLE (1)
For enabling the Repeat Mode of the I2C

#define CSL_I2C_RESET_DONE (1)
For indicating the completion of Reset

#define CSL_I2C_RESET_NOT_DONE (0)
For indicating the non-completion of Reset

#define CSL_I2C_RX_NOT_READY (0)
For indicating that the Receive ready signal is low

#define CSL_I2C_RX_READY (1)
For indicating that the Receive ready signal is high

#define CSL_I2C_SINGLE_BYTE_DATA (1)
For indicating Single Byte Data signal is set

#define CSL_I2C_STB_DISABLE (0)
For Disabling the Start Byte Mode for I2C (Normal Mode)

#define CSL_I2C_STB_ENABLE (1)
For Enabling the Start Byte Mode for I2C

#define CSL_I2C_TRANSMIT_UNDERFLOW (1)
For indicating Transmit Underflow signal is set

#define CSL_I2C_TX_NOT_READY (0)
For indicating that the Transmit Ready signal is low

#define CSL_I2C_TX_READY (1)
For indicating that the Transmit Ready signal is high

Chapter 6
ICACHE Module

Topics

6.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within ICACHE module.

The L1P Instruction Cache is designed to be direct-mapped and 32KB in size with a line size of 32 bytes. This corresponds to 1024 lines in the cache.

6.2 Functions

This section lists the functions available in the ICACHE module.

6.2.1 CSL_icacheInit

CSL_Status CSL_icacheInit (CSL_IcacheContext * *pContext*)

Description
This is the initialization function for the icache CSL. This function needs to be called before any other icache CSL functions are called. This function does not modify any registers or check status. It returns status CSL_SOK. It has been kept for future use.

Arguments

```
pContext   Pointer to module-context. As ICACHE doesn't have any
           context based information user is expected to pass NULL.
```

Return Value
CSL_Status

- CSL_SOK - Always returns

Pre Condition
CSL_sysInit() must be called.

Post Condition
None

Modifies
None

Example

```
CSL_Status      status;
...
status = CSL_icacheInit(NULL);
...
```

6.2.2 CSL_icacheOpen

CSL_IcacheHandle CSL_icacheOpen (CSL_IcacheObj * *pIcacheObj*,
CSL_InstNum *icacheNum*,
CSL_IcacheParam * *param*,
CSL_Status * *pStatus*
)

Description
This Function populates the peripheral data object for the instance and returns handle to it.

Arguments

```
pIcacheObj       Pointer to the data object for icache instance

icacheNum        Specifies the instance of the icache to be opened

pIcacheParam     Module specific parameter for icache instance

pStatus          Status of the function call
```

Return Value
CSL_IcacheHandle
Valid icache handle will be returned if status value is equal to CSL_SOK.

Pre Condition
I-cache module has to be initialized.

Post Condition
1. The status is returned in the status variable. If status returned is

- CSL_SOK Open call is successful
- CSL_ESYS_FAIL Open call failed

2. Icache object structure is populated.

Modifies
1. The status variable
2. The data object for the instance

Example

```
CSL_Status          status;
CSL_IcacheObj       icacheObj;
CSL_IcacheHandle    hIcache;
...
hIcache = CSL_IcacheOpen (&icacheObj, CSL_ICACHE, NULL, &status);
...
```

6.2.3 CSL_icacheClose

CSL_Status CSL_icacheClose (CSL_IcacheHandle *hIcache*)

Description
This function releases the instance of the icache.

Arguments

```
hIcache          Handle to the icache instance
```

Return Value
CSL_Status

- CSL_SOK - Close successful

- CSL_ESYS_BADHANDLE - Invalid handle

Pre Condition
icache would have opened properly.

Post Condition
Icache instance is closed and its usage is illegal until next open.

Modifies
peripheral data object

Example

```
CSL_IcacheHandle    hIcache;
...
status = CSL_icacheClose(hIcache);
...
```

6.2.4 CSL_icacheHwSetupRaw

CSL_Status CSL_icacheHwSetupRaw **(** CSL_IcacheHandle ***hIcache*****,**
CSL_IcacheConfig ***** ***icacheConfig***
)

Description
This function configures the icache using the register-values passed through the config-structure

Arguments

```
hIcache          Pointer to the data object for icache instance

icacheConfig     Pointer to config structure that contains the
                 information to program the icache to a useful state
```

Return Value
CSL_Status

- CSL_SOK - Configuration successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Invalid configuration parameters

Pre Condition
icache should have opened properly.

Post Condition
The registers of the specified icache instance is setup according to value passed.

Modifies
Hardware registers of the specified icache instance.

Example

```
CSL_IcacheHandle    hIcache;
CSL_IcacheConfig    icacheConfig;
CSL_Status          status;
...
status = CSL_icacheHwSetupRaw (hIcache, &icacheConfig);
...
```

6.2.5 CSL_icacheHwSetup

CSL_Status CSL_icacheHwSetup (CSL_IcacheHandle *hIcache*,
CSL_IcacheHwSetup * *hwSetup*
)

Description
This function initializes the L1P I-cache hardware to a known state.

Arguments

```
hIcache         Handle to the L1P I-cache instance

hwSetup         Pointer to hardware setup structure
```

Return Value
CSL_Status

- CSL_SOK - Hardware setup successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition
CSL_icacheInit () and CSL_icacheOpen () must be called.

Post Condition
The hardware set up structure will be populated with values for the setup parameters.

Modifies
Hardware registers

Example

```
CSL_IcacheHandle    hIcache;
CSL_IcacheHwSetup   icacheHwsetup;
CSL_Status          status;
icacheHwsetup = CSL_ICACHE_HWSETUP_DEFAULTS;
...
CSL_icacheHwSetup (hIcache, &icacheHwsetup);
...
```

6.2.6 CSL_icacheHwControl

CSL_Status CSL_icacheHwControl **(** CSL_IcacheHandle ***hIcache***,
CSL_IcacheHwControlCmd ***cmd***,
void * ***cmdArg***
)

Description
This function is used for controlling the L1P I-cache instance. It accepts the commands and data to configure/control the hardware. Some commands do not need any data.

Arguments

hIcache	Handle to the L1P I-cache instance
cmd	Command to be executed
cmdArg	Data pointer corresponding to command

Return Value
CSL_Status

- CSL_SOK - Command execution successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command

Pre Condition
icache should have been opened properly.

Post Condition
1. The status is returned. If status returned is

- CSL_SOK Command successful
- CSL_ESYS_BADHANDLE Invalid handle
- CSL_ESYS_INVCMD Invalid command

2. Appropriate hardware parameters are configured based on the command.

Modifies
Hardware registers

Example

```
CSL_Status                  status;
CSL_IcacheHandle            hIcache;

...
status  = CSL_icacheHwControl (hIcache,
                               CSL_ICACHE_CMD_CACHE_ENABLE,
                               NULL);
...
```

6.2.7 CSL_icacheGetHwStatus

CSL_Status CSL_icacheGetHwStatus **(** CSL_IcacheHandle ***hIcache***,
CSL_IcacheHwStatusQuery ***myQuery***,
void * ***response***
)

Description
This function is for querying the hardware status. It accepts the query and populates the data structure passed with the corresponding hardware status.

Arguments

hIcache	Handle to the DSP I-cache instance
myQuery	Query to be performed
response	Pointer to data structure to hold the status information

Return Value
CSL_Status

- CSL_SOK - Query successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Invalid query
- CSL_ESYS_INVPARAMS - Invalid parameters

Pre Condition
CSL_icacheInit () and CSL_icacheOpen () must be called.

Post Condition
1. The status is returned. If status returned is

- CSL_SOK Query is successful
- CSL_ESYS_BADHANDLE Invalid handle
- CSL_ESYS_INVQUERY Invalid query

2. The status information is copied to the corresponding data structure.

Modifies
Data structure pointed by response

Example

```
CSL_Status                  status;
CSL_IcacheMode              modarg;
CSL_IcacheHandle            hIcache;
...
```

```
            status = CSL_icacheGetHwStatus (hIcache,
                                            CSL_ICACHE_QUERY_MODE,
                                            (void*) &modarg);
            ...
```

6.2.8 CSL_icacheGetHwSetup

CSL_Status CSL_icacheGetHwSetup (CSL_IcacheHandle *hIcache*,
CSL_IcacheHwSetup * *hwSetup*
)

Description
It retrieves the hardware setup parameters of the I-cache module specified by the given handle.

Arguments

```
            hIcache         Handle to the instance of I-cache

            hwSetup         Pointer to the hardware setup structure
```

Return Value
CSL_Status

- CSL_SOK - Retrieving the hardware setup parameters is successful
- CSL_ESYS_BADHANDLE - The handle is passed is invalid
- CSL_ESYS_INVPARAMS - Invalid parameters

Pre Condition
icache would have opened properly.

Post Condition
The hardware setup structure is populated with the hardware setup parameters.

Modifies
hwSetup variable

Example

```
            CSL_IcacheHandle   hIcache;
            CSL_IcacheHwSetup  hwSetup;
            CSL_Status         status;
            ...
            status = CSL_icacheGetHwSetup(hIcache, &hwSetup);

            ...
```

6.2.9 CSL_icacheGetBaseAddress

CSL_Status CSL_icacheGetBaseAddress (CSL_InstNum *icacheNum*,

CSL_IcacheParam * *pIcacheParam*,

CSL_IcacheBaseAddress * *pBaseAddress*

)

Description
Function to get the base address of the peripheral instance. This function is used for getting the base address of the peripheral instance. This function will be called inside the CSL_icacheOpen() function call. This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral MMR's go to an alternate location.

Arguments

icacheNum	Specifies the instance of the Icache to be opened
pIcacheParam	Module specific parameters
pBaseAddress	Pointer to baseaddress structure containing base address details

Return Value
CSL_Status

- CSL_SOK Open call is successful
- CSL_ESYS_FAIL The instance number is invalid

Pre Condition
None

Post Condition
Base Address structure is populated

Modifies
Base address structure is modified

Example

```
CSL_Status              status;
CSL_IcacheBaseAddress   baseAddress;
...
status = CSL_icacheGetBaseAddress(CSL_ICACHE, NULL,
                                  &baseAddress);
...
```

6.3 Data Structures

This section lists the data structures available in the ICACHE module.

6.3.1 CSL_IcacheObj

Detailed Description
L1P ICACHE data object.

Field Documentation

CSL_InstNum CSL_IcacheObj::icacheNum
icache instance

CSL_IcacheRegsOvly CSL_IcacheObj::regs
Pointer to the I-CACHE Register Overlay structure

6.3.2 CSL_IcacheParam

Detailed Description
Icache Module specific parameters. Present implementation doesn't have any module specific parameters.

Field Documentation

CSL_BitMask16 CSL_IcacheParam::flags
Bit mask to be used for module specific parameters. The declaration is just a placeholder for future implementation.

6.3.3 CSL_IcacheContext

Detailed Description
Icache module specific context information. Present implementation doesn't have any Context information.

Field Documentation

Uint16 CSL_IcacheContext::contextInfo
Context information of Icache. The declaration is just a placeholder for future implementation.

6.3.4 CSL_IcacheConfig

Detailed Description
Icache Config structure.

Field Documentation

volatile Uint32 CSL_IcacheConfig:: CSR
Variable which holds the value for Control and Status register. Here it is used to read and write the PCC field that corresponds to the icache mode.

volatile Uint32 CSL_IcacheConfig:: MEMCSR
Variable which holds the value for Memory Control and Status register. Here it is used to read and write the P field that determines the priority between CSP Slave port and L1P cache for RAM access.

6.3.5 CSL_IcacheHwSetup

Detailed Description
This structure is used to provide I-cache setup parameters to the function CSL_icacheHwSetup.

Field Documentation

CSL_IcacheMode CSL_IcacheHwSetup::mode
The operating modes of the cache are CACHE ENABLE, CACHE FREEZE and CACHE BYPASS

CSL_IcachePrio CSL_IcacheHwSetup::ramAccessPrio
Determines the priority for RAM access between I-Cache and CSP Slave port

6.3.6 CSL_IcacheBaseAddress

Detailed Description
This contains the base-address information for the peripheral instance.

Field Documentation

CSL_IcacheRegsOvly CSL_IcacheBaseAddress::regs
Base-address of the Configuration registers of the peripheral

6.4 Enumerations

This section lists the enumerations available in the ICACHE module.

6.4.1 CSL_IcacheMode

enum CSL_IcacheMode
This enum is used to convey the information of the operating mode to be set using hardware control commands or the value returned using status query commands.

Enumeration values:

CSL_ICACHE_MODE_CACHE_ENABLE	For the hardware control commands, this enumeration value is used to set the mode parameter associated with the control command. For status query commands this enumeration value tells that queried field is set cache enabled.
CSL_ICACHE_MODE_CACHE_FREEZE	For the hardware control commands, this enumeration value is used to set the mode parameter associated with the control command. For status query commands, this enumeration value tells that queried field is set cache freezed.
CSL_ICACHE_MODE_CACHE_BYPASS	For the hardware control commands, this enumeration value is used to set the mode parameter associated with the control command. For status query commands this enumeration value tells that queried field is set cache bypassed.

6.4.2 CSL_IcachePrio

enum CSL_IcachePrio
This enum is used to convey the information regarding RAM access priority to be set using hardware control commands or the value returned using status query commands.

Enumeration values:

CSL_ICACHE_PRIO_CACHE	RAM access priority for I-Cache
CSL_ICACHE_PRIO_CSP	RAM access priority for CSP slave port

6.4.3 CSL_IcacheWait

enum CSL_IcacheWait
This enum is used to issue a wait till the cache operation is complete in accordance with the hardware control command.

Enumeration values:

CSL_ICACHE_NO_WAIT	Return immediately without any wait
CSL_ICACHE_WAIT	Wait till the cache operation is done

6.4.4 CSL_IcacheHwStatusQuery

enum CSL_IcacheHwStatusQuery

Status query commands that are supported by the L1P I-cache CSL.

Enumeration values:

CSL_ICACHE_QUERY_MODE	Gets the current configured value PCC field of CSR register. **Parameters:** *(CSL_IcacheMode *)*
CSL_ICACHE_QUERY_INV_WORDCNT	Gets the current value of the Invalidate Word Count field in the L1PICR register. **Parameters:** *(CSL_IcacheCount*)*
CSL_ICACHE_QUERY_INV_STARTADDR	Gets the current value of the Invalidate Start Address field in the L1PSAR register. **Parameters:** *(CSL_IcacheAddr*)*
CSL_ICACHE_QUERY_INV_CACHE	Gets the current value of the IP field in the L1PICR register. **Parameters:** *(Uint8 *)*
CSL_ICACHE_QUERY_PRIO	Get the priority setting in P bit of MEMCSR. **Parameters:** *(CSL_IcachePrio*)*

6.4.5 CSL_IcacheHwControlCmd

enum CSL_IcacheHwControlCmd
Enumeration for L1P ICACHE Control commands.

Enumeration values:

CSL_ICACHE_CMD_CACHE_ENABLE	Command for enabling L1P I-cache. **Parameters:** *None*
CSL_ICACHE_CMD_CACHE_FREEZE	Command for freezing I-cache. **Parameters:** *None*
CSL_ICACHE_CMD_CACHE_BYPASS	Command for bypassing I-cache. **Parameters:** *None*
CSL_ICACHE_CMD_INV_STARTADDR	Command for setting the start address for the cache region to be invalidated. **Parameters:** *CSL_IcacheAddr*
CSL_ICACHE_CMD_INV_BYTECNT	Command for setting the number of words to be invalidated starting from the Start Address. User needs to make a delay for the invalidation; if required. **Parameters:** *CSL_IcacheCount**
CSL_ICACHE_CMD_INV_CACHELINE	Command for invalidating a specific I-cache line in

	L1P I-cache. User needs to make a delay for the invalidation; if required. **Parameters:** *CSL_IcacheAddr*
CSL_ICACHE_CMD_INV_CACHE	Command for invalidating the entire L1P I-cache. User can make a request to wait until the invalidation completes along with the control command. **Parameters:** *CSL_IcacheWait**

6.5 Macros

#define CSL_ICACHE_HWSETUP_DEFAULTS
Value:

```
{ \
    CSL_ICACHE_MODE_CACHE_ENABLE,\
    CSL_ICACHE_PRIO_CSP     \
}
```

The default value for CSL_IcacheHwSetup structure

- Cache Mode - Cache enable
- Priority for RAM access - CSP.

Chapter 7
INTC Module

Topics

7.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within INTC module.

The Octave Mega module CPU supports 16 prioritized interrupts which are controlled by Interrupt Controller.

The registers that constitute the interrupt controller module are
Interrupt Enable Register - Enables/Disables the interrupts individually
Interrupt Flag Register - Indicates the occurrence of interrupts
Interrupt set Register - Sets the interrupt flags in the Interrupt Flag Register
Interrupt Clear Register - Clears the interrupt flags in the Interrupt Flag Register
Interrupt Service Table Pointer – Pointer to the Interrupt Vector Table

Non Maskable Interrupt Return Pointer – Holds the return address to the location after the servicing of the non maskable interrupt

Interrupt Return Pointer - Holds the return address to the location after the servicing of the maskable interrupt.

7.2 Functions

This section lists the functions available in the INTC module.

7.2.1 CSL_intcInit

CSL_Status CSL_intcInit (CSL_IntcContext * *pContext*)

Description This API performs any module-specific initialization. CSL_intcInit(..) must be invoked before calling any other API in the INTC module.

Arguments

```
        pContext    pointer to the intc context memory allocated by
                    the user
```

Return Value
CSL_Status

- CSL_SOK - returns on success
- CSL_ESYS_ALREADY_INITIALIZED - if initialized

Pre Condition
CSL_sysInit() must be called.
Memory has to be allocated and filled for the context variable.

Post Condition
Intc module is initialized.

Modifies
None

Example:

```
    ...
    if ((CSL_intcInit() != CSL_SOK) &&
       (CSL_intcInit() != CSL_ESYS_ALREADY_INITIALIZED)) {
       //module initialization failed! //
    }
```

7.2.2 CSL_intcOpen

CSL_IntcHandle CSL_intcOpen

(**CSL_IntcObj *** ***intcObj***,
CSL_IntcEventId ***eventId***,
CSL_IntcParams * ***params***,
CSL_Status * ***status***
)

Description
The API would reserve an interrupt-event for use. It returns a valid handle to the event only if the event is not currently allocated. The user could release the event after use by calling CSL_intcClose(). The CSL-object ('intcObj') that the user passes would be used to store information pertaining handle.

Arguments

```
        pIntcObj     Pointer to the CSL-object allocated by the user

        eventId      The event-id of the interrupt

        params       Module specific parameter

        pStatus      (Optional) pointer for returning status of the
                     function call
```

Return Value
CSL_IntcHandle
Valid INTC handle identifying the event

Pre Condition
1. INTC module must be initialized properly.
2. Memory allocated for obj structure

Post Condition
1. INTC object structure is populated
2. The status is returned in the status variable. If status returned is

- CSL_SOK Valid intc handle is returned
- CSL_ESYS_FAIL The open command failed

Modifies
1. The status variable
2. INTC object structure

Example:

```
    CSL_IntcObj     intcObj;
    CSL_IntcHandle  hIntc;
    CSL_Status      openStatus;

    hIntc = CSL_intcOpen(&intcObj, CSL_INTC_EVENTID_I2C, NULL,
                         NULL);
    if (openStatus != CSL_SOK)
    {
    // open failed //
    }
```

7.2.3 CSL_intcClose

CSL_Status CSL_intcClose (CSL_IntcHandle *hIntc*)

Description
Releases an allocated event. CSL_intcClose() must be called to release an event that has been previously allocated with a call to CSL_intcOpen().

Arguments

```
hIntc            Handle identifying the event
```

Return Value
CSL_Status

- CSL_SOK - Close successful
- CSL_ESYS_FAIL - Close failed

Pre Condition
1. INTC module must have been initialized properly.
2. Event must have been opened.

Post Condition
1. CPU interrupt could be used again.
2. The status is returned in the status variable.

Modifies
The status variable

Example

```
CSL_IntcHandle      hIntc;
CSL_Status          status;
...

status = CSL_intcClose(hIntc);
if (status != CSL_SOK) {
    /* close failed! */
}
...
```

7.2.4 CSL_intcHwSetup

CSL_Status CSL_intcHwSetup (CSL_IntcHandle *hIntc*, CSL_IntcHwSetup * *setup*)

Description
CSL_intcHwSetup () API is used to configure the interrupt controller for the event identified by the handle. The user must instantiate and initialize a setup-structure with appropriate configuration parameters before passing it to the function. As no setup parameters are identified for C672x\DA7xx INTC, this is a dummy API.

Arguments

```
hIntc        Handle to the INTC instance

hwSetup      Pointer to hardware setup structure
```

Return Value
CSL_Status

- CSL_SOK - always return

NOTE:
In C672x\DA7xx CSL_intcHwSetup() is a function which does nothing as no h/w configuration is to be done.

Pre Condition
1.INTC module must have been initialized properly.
2.Event must have been opened.

Post Condition
None

Modifies
None

Example:

```
CSL_IntcHandle  hIntc;
CSL_Status      status;
CSL_IntcHwSetup setup
...
/* Here just pass the CSL_IntcHwSetup* as NULL */
CSL_intcHwSetup(hIntc, &setup);
...
```

7.2.5 CSL_intcHwControl

CSL_Status CSL_intcHwControl **(** CSL_IntcHandle ***hIntc***,
CSL_IntcHwControlCmd ***controlCommand***,
void * ***commandArg***
)

Description
This API is used to invoke any of the supported control-operations supported by the module.

Note: Refer to the control-command documentation for details on the parameter (if any) that a specific command accepts.

Arguments

```
hIntc           Handle identifying the event
```

```
controlCommand  The command to this API indicates the action to
                be taken on INTC.

commandArg      An optional argument.
```

Return Value
CSL_Status

- CSL_SOK - HwControl successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command

Pre Condition
INTC should be initialized and opened properly.

Post Condition
None

Modifies
The hardware registers of INTC.

Example:

```
CSL_IntcHandle  hIntc;
CSL_Status      status;
...
status CSL_intcHwControl(hIntc, CSL_INTC_CMD_EVTSET, NULL);
...
```

7.2.6 CSL_intcGetHwStatus

CSL_Status CSL_intcGetHwStatus (CSL_IntcHandle ***hIntc***,
CSL_IntcHwStatusQuery ***myQuery***,
void * ***answer***
)

Description
The CSL_intcGetHwStatus() API could be used to retrieve status or configuration information from the peripheral. The user must allocate an object that would hold the retrieved information and pass a pointer to it to the function. The type of the object is specific to the query-command.

Arguments

```
hIntc           Handle identifying the event

myQuery         The query to this API of INTC that indicates
                the status to be returned.

response        Placeholder to return the status.
```

Return Value
CSL_Status

- CSL_SOK - Status info return successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Invalid query

Pre Condition
INTC should be initialized and opened properly.

Post Condition
None

Modifies
None

Example:

```
CSL_IntcHandle  hIntc;
CSL_Status      status;
...
Bool  evtPending = FALSE;
while (evtPending == FALSE) {
   CSL_intcGetHwStatus(hIntc, CSL_INTC_QUERY_ISEVENTPENDING,
                    &evtPending);
}
```

7.2.7 CSL_intcEventEnable

CSL_Status CSL_intcEventEnable (CSL_IntcEventId *eventId*,
CSL_IntcEventEnableState * *prevState*
)

Description
The API enables the specified event. If the user wishes to restore the enable-state of the event at a later point of time, they may store the current state using the parameter, which could be used with CSL_intcEventRestore(..). Note: The function directly works on the event and hence it is not necessary to "open" the event to invoke the API.

Arguments

```
eventId     Event-ID of interest

prevState   (Optional) Pointer to object that would store
            current state
```

Return Value C
SL_Status

- CSL_SOK on success
- CSL_INTC_EVENTID_INVALID

Example:

```
CSL_Status      status;
...

status = CSL_intcEventEnable(CSL_INTC_EVENTID_SPI, NULL);
```

7.2.8 CSL_intcEventDisable

CSL_Status CSL_intcEventDisable (CSL_IntcEventId *eventId*,
CSL_IntcEventEnableState * *prevState*
)

Description
The API disables the specified event. If the user wishes to restore the enable-state of the event at a later point of time, they may store the current state using the parameter, which could be used with CSL_intcEventRestore(..). Note: The function directly works on the event and hence it is not necessary to "open" the event to invoke the API.

Arguments

```
eventId     Event-ID of interest

prevState   (Optional) Pointer to object that would store
            current state
```

Return Value
CSL_Status

- CSL_SOK on success
- CSL_INTC_EVENTID_INVALID

Example:

```
...
CSL_IntcEventEnableState    oldState;
CSL_intcEventDisable(CSL_INTC_EVENTID_SPI, &oldState);
...
```

7.2.9 CSL_intcEventRestore

CSL_Status CSL_intcEventRestore (CSL_IntcEventId *eventId*,
CSL_IntcEventEnableState *prevState*
)

Description
The API restores the specified event to a previous enable-state as recorded by the event-enable state passed as an argument. Note: The function directly works on the event and hence it is not necessary to "open" the event to invoke the API.

Arguments

```
eventId     Event-ID of interest

prevState   Object that contains information about previous
            state
```

Return Value
CSL_Status

- CSL_SOK on success
- CSL_INTC_EVENTID_INVALID

Example:

```
...
CSL_intcEventRestore(CSL_INTC_EVENTID_SPI, oldState);
...
```

7.2.10 CSL_intcGlobalEnable

CSL_Status CSL_intcGlobalEnable (CSL_IntcGlobalEnableState * *prevState*)

Description
The API enables the global interrupt by manipulating the processor's global interrupt enable/disable flag. If the user wishes to restore the enable-state at a later point, they may store the current state using the parameter, which could be used with CSL_intcGlobalRestore(..).

Arguments

```
prevState   (Optional) Pointer to object that would store
            current stateObject that contains information about
            previous state
```

Return Value
CSL_Status

- CSL_SOK on success

Example:

```
CSL_IntcGlobalEnableState   prevGie;
CSL_Status status;
...
status = CSL_intcGlobalEnable (&prevGie);
...
```

7.2.11 CSL_intcGlobalDisable

CSL_Status CSL_intcGlobalDisable (CSL_IntcGlobalEnableState * *prevState*)

Description
The API disables the global interrupt by manipulating the processor's global interrupt enable/disable flag. If the user wishes to restore the enable-state at a later point, they may store the current state using the parameter, which could be used with CSL_intcGlobalRestore(..).

Arguments

```
prevState   (Optional) Pointer to object that would store
            current stateObject that contains information about
            previous state
```

Return Value
CSL_Status

- CSL_SOK on success

Example:

```
CSL_IntcGlobalEnableState   gieState;
...
CSL_intcGlobalDisable(&gieState);
  // critical-section code //
CSL_intcGlobalRestore(gieState);
...
```

7.2.12 CSL_intcGlobalRestore

CSL_Status CSL_intcGlobalRestore (CSL_IntcGlobalEnableState *prevState*)

Description
The API restores the global interrupt enable/disable state to a previous state as recorded by the global-event-enable state passed as an argument.

Arguments

```
prevState   Object containing information about previous state
```

Return Value
CSL_Status

- CSL_SOK on success

Example:

```
CSL_IntcGlobalEnableState   gieState;

...
CSL_intcGlobalDisable(&gieState);
```

```
    // critical-section code //
    CSL_intcGlobalRestore(gieState);
    ...
```

7.2.13 CSL_intcDispatcherInit

CSL_Status CSL_intcDispatcherInit (CSL_IntcDispatcherContext * *pContext*)

Description
The user should call CSL_intcDispatcherInit (..) if they wish to make use of the dispatcher built into the CSL INTC module.
Note: This API must be called before using CSL_intcPlugEventHandler(..).

Arguments

```
        pContext   Context relevant to Dispatcher
```

Return Value
CSL_Status

- CSL_SOK on success
- CSL_ESYS_ALREADY_INITIALIZED

Pre Condition
Memory has to be allocated for the context variable

Post Condition
None

Modifies
None

Example:

```
    if ((CSL_intcInit( ) != CSL_SOK) ||
        (CSL_intcInit() != CSL_ESYS_ALREADY_INITIALIZED)) {
           // module initialization failed! //
    }
    if ((CSL_intcDispatcherInit( ) != CSL_SOK) &&
        (CSL_intcDispatcherInit( ) != CSL_ESYS_ALREADY_INITIALIZED)){
       // CSL dispatcher setting up failed! //
    }
```

7.2.14 CSL_intcPlugEventHandler

CSL_IntcEventHandler CSL_intcPlugEventHandler(CSL_IntcHandle *hIntc*,
CSL_IntcEventHandlerRecord * *eventHandlerRecord*
)

Description
CSL_intcPlugEventHandler(..) ties an event-handler to an event; so that the occurrence of the event, would result in the event-handler being invoked.

Arguments

```
hIntc                Handle identifying the interrupt-event

eventHandlerRecord   Provides the details of the event-handler
```

Return Value
Returns the address of the previous handler or CSL_INTC_EVENTHANDLER_PLUG_ERROR if the isr was not properly plugged

Pre Condition
1. INTC has to be initialized and opened for the event.
2. DispatcherInit has to be called.
3. EventHandlerRecord has to be filled.

Post Condition
Handler plugged

Modifies
None

Example:

```
CSL_IntcEventHandlerRecord  evtHandlerRecord;
...
evtHandlerRecord.handler = myIsr;
evtHandlerRecord.arg     = (void *)hTimer;
CSL_intcPlugEventHandler(hIntc, &evtHandlerRecord);
...
```

7.2.15 CSL_intcHookIsr

CSL_Status CSL_intcHookIsr (CSL_IntcEventId *eventId*, Uint32 *isrAddr*)

Description
The CSL_intcHookIsr (..) hooks up the Interrupt Service Routine(ISR) to the specified event.

Note: Unlike CSL_intcPlugEventHandler (..), this is done without opening the module (INTC) for the event. Hence, the CSL dispatcher is also not made to use.

Arguments

```
eventId       Event Identifier
```

```
isrAddr       Pointer to the handler
```

Return Value
CSL_Status

- CSL_SOK - Isr plugged properly
- CSL_ESYS_FAIL - Isr plug failed.(NB: The DMax uid CSL_DMAX_LOPRIORITY_EVENT0_UID used in INTC, could be in use by some other application)

Pre Condition
INTC has to be initialized properly

Post Condition
Handler plugged

Modifies
None

Example:

```
...
CSL_intcHookIsr(CSL_INTC_EVENTID_RTI_INT_REQ0 , rtiIsr);
...
```

7.2.16 CSL_intcSetVectorPtr

Uint32 CSL_intcSetVectorPtr (Uint32 *vecPtr*)

Description
The API changes the base address of the Interrupt Vector table to the value passed.

Arguments

```
vecPtr   New base address of the Interrupt Vector Table. Care
         should be taken that first ten lower bits are all zeroes
```

Return Value

- Previously loaded ISTP value

Example:

```
#define VEC_ADDR  0x100c0000
Uint32 prevValue;
...
prevValue = CSL_intcSetVectorPtr(VEC_ADDR);
...
```

7.3 Data Structures

This section lists the data structures available in the INTC module.

7.3.1 CSL_IntcObj

Detailed Description
The interrupt handle object.
This object is used referenced by the handle to identify the event.

Field Documentation

CSL_IntcEventId CSL_IntcObj::eventId
The event-id

void* CSL_IntcObj::reserved
Reserved for the future

7.3.2 CSL_IntcContext

Detailed Description
INTC Module Context.

Field Documentation

CSL_BitMask16 CSL_IntcContext:: flags
Flags to indicate presence or absence of OS

CSL_BitMask16 CSL_IntcContext:: eventAllocMask[CSL_INTC_EVENT_CNT]
Masks that indicate the allocation of events

7.3.3 CSL_IntcHwSetup

Detailed Description
HwSetup.

Field Documentation

void* CSL_IntcHwSetup::reserved
Reserved for future

7.3.4 CSL_IntcEventHandlerRecord

Detailed Description
Event Handler Record.
Used to set-up the event-handler using CSL_intcPlugEventHandler(..)

Field Documentation

CSL_IntcEventHandler CSL_IntcEventHandlerRecord:: handler
Pointer to the event handler

void * CSL_IntcEventHandlerRecord:: arg
The argument to be passed to the handler when it is invoked. This should be NULL for an isr

7.3.5 CSL_IntcDispatcherContext

Detailed Description
Context relevant to the dispatcher.

Field Documentation

CSL_IntcEventHandler CSL_IntcEventHandlerRecord:: handler
Holds the isr records of the events

7.4 Enumerations

This section lists the enumerations available in the INTC module.

7.4.1 CSL_IntcEvent

enum CSL_IntcEvent
Interrupt Enable/Disable.

Enumeration values:

CSL_INTC_EVENT_ENABLE	Enables event
CSL_INTC_EVENT_DISABLE	Disables event

7.4.2 CSL_IntcGlobal

enum CSL_IntcGlobal
Global Interrupt Enable/Disable.

Enumeration values:

CSL_INTC_GLOBAL_ENABLE	Global Enable
CSL_INTC_GLOBAL_DISABLE	Global Disable

7.4.3 CSL_EventStatus

enum CSL_EventStatus
Hardware status response.

Enumeration values:

CSL_INTC_EVENTPENDING	Event pending
CSL_INTC_EVENTNOTPENDING	Event not pending

7.4.4 CSL_IntcHwControlCmd

enum CSL_IntcHwControlCmd
Enumeration of the control commands.
These are the control commands that could be used with CSL_intcHwControl(..). Some of the commands expect an argument as documented along side the description of the command.

Enumeration values:

CSL_INTC_CMD_EVTENABLE	Enables the event. **Parameters:** *None*
CSL_INTC_CMD_EVTDISABLE	Disables the event. **Parameters:** *None*
CSL_INTC_CMD_EVTCLEAR	Clears the event. **Parameters:** *None*

CSL_INTC_CMD_EVTSET	Sets the event (software triggering). **Parameters:** *None*

7.4.5 CSL_IntcHwStatusQuery

enum CSL_IntcHwStatusQuery
Enumeration of the queries.
These are the queries that could be used with CSL_intcGetHwStatus(..). The queries return a value through the object pointed to by the pointer that it takes as an argument. The argument supported by the query is documented along side the description of the query.

Enumeration values:

CSL_INTC_QUERY_ISEVENTPENDING	Checks if event is pending. **Parameters:** *(Bool *)*

7.5 Macros

#define CSL_INTC_EVENT_CNT (16)
Count of the number of interrupt-events

#define CSL_INTC_EVENTMASK_ENABLE (1)
Event Enable

#define CSL_INTC_CONTEXT_DISABLECOREVECTORWRITES (1)
Flag to disable modification of the Interrupt Vector Area on the core specified using 'CSL_IntcContext' via `CSL_intcInit()'.

#define CSL_INTC_EVENTID_INVALID (-1)
Invalid EventId

#define CSL_INTC_EVTHANDLER_NONE ((CSL_IntcEventHandlerrecord *) 0)
Indicates there is no associated event-handler.

#define CSL_INTC_BADHANDLE (0)
Invalid handle

#define CSL_INTC_EVENTHANDLER_PLUG_ERROR ((CSL_IntcEventHandler)(-2))
Indicates that the interruptHandler passed as the argument for the function CSL_intcPlugEventHandler was not plugged properly in the vector table(The DMAX uid used for INTC ie CSL_DMAX_LOPRIORITY_EVENT0_UID might be in use by some other application)

Chapter 8
McASP Module

Topics

8.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within McASP module.

The multi-channel audio serial port (McASP) functions as a general-purpose audio serial port optimized for the needs of multichannel audio applications. The McASP is useful for time-division multiplexed (TDM) stream, Inter-Integrated Sound (I2S) protocols, and inter-component digital audio interface transmission (DIT).

The McASP consists of transmit and receive sections that may operate synchronized, or completely independently with separate master clocks, bit clocks, and frame syncs, and using different transmit modes with different bit stream formats. The McASP module may include up to 16 serializers that can be individually enabled to either transmit or receive. In addition, all McASP pins can be configured as GPIOs. This is done by setting the appropriate bit in the device configuration register.

The McASP includes the following pins:

- Serializers
- Transmit clock generator
- Receive clock generator
- Receive Frame Sync Generator
- Mute in/out

8.2 Functions

This section lists the functions available in the McASP module.

8.2.1 CSL_mcaspInit

CSL_Status CSL_mcaspInit (CSL_McaspContext * *pContext*)

Description
This is the initialization function for the multi channel audio serial port CSL. This function needs to be called before any other multi channel audio serial port CSL functions are to be called. This function does not modify any registers or check status. It returns status CSL_SOK. It has been kept for future use.

Arguments

```
pContext   Context information for the instance. Should be NULL
```

Return Value
CSL_Status

- CSL_SOK - Always returns

Pre Condition
CSL_sysInit() must be called.

Post Condition
None

Modifies
None

Example

```
CSL_Status  status;
...
status = CSL_mcaspInit (NULL);
...
```

8.2.2 CSL_mcaspOpen

CSL_McaspHandle CSL_mcaspOpen

(
CSL_McaspObj * *pMcaspObj*,
CSL_InstNum *mcaspNum*,
CSL_McaspParam * *pMcaspParam*,
CSL_Status * *pStatus*
)

Description
This function populates the peripheral data object for the instance and returns handle to it.

Arguments

```
pMcaspObj     Pointer to the data object for McASP instance

mcaspNum      Specifies the instance of the McASP to be opened.
              There are 3 instances of the McASP.

pMcaspParam   Module specific parameter for McASP instance

pStatus       Status of the function call
```

Return Value
CSL_McaspHandle
Valid McASP instance handle will be returned if status value is equal to CSL_SOK.

Pre Condition
None

Post Condition
1. The status is returned in the status variable. If status returned is

- CSL_SOK - Open call is successful.
- CSL_ESYS_FAIL - The McASP instance is invalid.

2. Multi channel Audio serial Port object structure is populated.

Modifies
1. The status variable
2. The data object for the instance

Example

```
CSL_Status          status;
CSL_McaspObj        mcaspObj;
CSL_McaspHandle     hMcasp;
...
hMcasp = CSL_McaspOpen (&mcaspObj,
                        CSL_MCASP_1, NULL
                        &status);
...
```

8.2.3 CSL_mcaspClose

CSL_Status CSL_mcaspClose (CSL_McaspHandle *hMcasp*)

Description
This function closes the McASP instance.

Arguments

```
hMcasp          Handle to the McASP instance
```

Return Value CSL_Status

- CSL_SOK - McASP is close Successful
- CSL_ESYS_BADHANDLE - The handle passed is invalid

Pre Condition
CSL_mcaspInit() and CSL_mcaspOpen() must be called successfully in that order before CSL_mcaspClose() can be called.

Post Condition
The McASP instance is closed.

Modifies
McASP Handle

Example

```
CSL_McaspHandle hMcasp;
CSL_Status      status;
...
status = CSL_mcaspClose (hMcasp);
...
```

8.2.4 CSL_mcaspHwSetup

CSL_Status CSL_mcaspHwSetup (CSL_McaspHandle *hMcasp*, CSL_McaspHwSetup * *myHwSetup*)

Description
It configures the McASP instance registers as per the values passed in the hardware setup structure.

Arguments

```
hMcasp      Handle to the McASP instance

myHwSetup   Pointer to hardware setup structure
```

Return Value
CSL_Status

- CSL_SOK - Hardware setup successful

- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Hardware structure is not properly initialized

Pre Condition
Both *CSL_mcaspInit()* and *CSL_mcaspOpen()* must be called successfully in that order before this function can be called. The user has to allocate space for & fill in the main setup structure appropriately before calling this function.

Post Condition
The specified instance will be setup according to value passed.

Modifies
Hardware registers for the specified instance

Example

```
CSL_Status        status;
CSL_McaspHwSetup  myHwSetup;
CSL_McaspHandle   hMcasp;

myHwSetup.glb.ctl = 0x00000000;
myHwSetup.glb.ditCtl = 0x00000003;
myHwSetup.glb.lbMode = 0x00000002;
myHwSetup.glb.amute = 0x00001234;
myHwSetup.glb.serSetup[1] = 0x00000012;
myHwSetup.glb.serSetup[2] = 0x00000012;
myHwSetup.rx.mask = 0x11111111;
myHwSetup.tx.mask = 0x11111111;
myHwSetup.rx.fmt = 0x00001111;
myHwSetup.tx.fmt = 0x00001111;
myHwSetup.rx.frSyncCtl = 0x00000001;
myHwSetup.tx.frSyncCtl = 0x00000001;
myHwSetup.rx.tdm = 0xFFFFFFFF;
myHwSetup.tx.tdm = 0xFFFFFFFF;
myHwSetup.rx.intCtl = 0x000000BF;
myHwSetup.tx.intCtl = 0x000000BF;
myHwSetup.rx.stat = 0x000001F7;
myHwSetup.tx.stat = 0x000001F7;
myHwSetup.rx.evtCtl = 0x00000000;
myHwSetup.tx.evtCtl = 0x00000000;
myHwSetup.rx.clk.clkSetupClk = 0x00000001;
myHwSetup.tx.clk.clkSetupClk = 0x00000001;
myHwSetup.rx.clk.clkSetupHiClk = 0x00000101;
myHwSetup.tx.clk.clkSetupHiClk = 0x00000101;
myHwSetup.rx.clk.clkChk = 0x00432100;
myHwSetup.tx.clk.clkChk = 0x00432100;
myHwSetup.emu = CSL_MCASP_PWRDEMU_FREE_ON;
...

status = CSL_mcaspHwsetup (hMcasp, &myHwSetup);
...
```

8.2.5 CSL_mcaspHwSetupRaw

CSL_Status CSL_mcaspHwSetupRaw **(** CSL_McaspHandle ***hMcasp*,**
CSL_McaspConfig ***** ***config***
)

Description
This function initializes the device registers with the register-values provided through the config data structure. This configures registers based on a structure of register values, as compared to HwSetup, which configures registers based on structure of bit field values and may perform other functions (delays, etc.)

Arguments

```
hMcasp        Handle to the McASP instance

config        Pointer to config structure
```

Return Value
CSL_Status

- CSL_SOK - Configuration successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Configuration structure is not properly initialized

Pre Condition
Both *CSL_mcaspInit()* and *CSL_mcaspOpen()* must be called successfully in that order before *CSL_mcaspHwSetupRaw()* can be called.

Post Condition
The registers of the specified McASP instance will be setup according to value passed.

Modifies
Hardware registers of the specified McASP instance

Example

```
CSL_McaspHandle       hMcasp;
CSL_McaspConfig       config;
CSL_Status            status;

config.PWREMUMGT=0x00000001;
config.PFUNC = 0x00000000;
config.PDIR = 0x00000000;
config.PDOUT = 0x00000000;
config.PDIN_PDSET = 0x00000000;
config.PDCLR = 0x00000000;
config.GBLCTL = 0x00000000;
config.AMUTE = 0x00000000;
config.LBCTL = 0x00000000;
```

```
config.TXDITCTL = 0x00000000;
config.RXMASK = 0x00000000;
config.RXFMT = 0x00000000;
config.RXFMCTL = 0x00000000;
config.ACLKRCTL = 0x00000020;
config.AHCLKRCTL = 0x00008000;
config.RXTDM = 0x00000000;
config.EVTCTLR = 0x00000000;
config.RXSTAT = 0x00000000;
config.RXCLKCHK = 0x00000000;
config.REVTCTL = 0x00000000;
config.TXMASK = 0x00000000;
config.TXFMT = 0x00000000;
config.TXFMCTL = 0x00000000;
config.ACLKXCTL = 0x00000060;
config.AHCLKXCTL = 0x00008000;
config.TXTDM = 0x00000000;
config.EVTCTLX = 0x00000000;
config.TXSTAT = 0x00000000;
config.TXCLKCHK = 0x00000000;
config.XEVTCTL = 0x00000000;
config.SRCTL0 = 0x00000000;
config.SRCTL1 = 0x00000000;
config.SRCTL2 = 0x00000000;
config.SRCTL3 = 0x00000000;
config.SRCTL4 = 0x00000000;
config.SRCTL5 = 0x00000000;
...
status = CSL_mcaspHwSetupRaw (hMcasp, &config);
...
```

8.2.6 CSL_mcaspGetHwSetup

CSL_Status CSL_mcaspGetHwSetup (CSL_McaspHandle *hMcasp*, CSL_McaspHwSetup * *myHwSetup*)

Description
This function retrieves the hardware setup parameters.

Arguments

hMcasp	Handle to the McASP instance
myHwSetup	Pointer to hardware setup structure

Return Value
CSL_Status

- CSL_SOK - Hardware setup retrieved successfully

- CSL_ESYS_BADHANDLE - Invalid handle

Pre Condition
Both *CSL_mcaspInit()* and *CSL_mcaspOpen()* must be called successfully in that order before *CSL_mcaspGetHwSetup()* can be called.

Post Condition
The hardware set up structure will be populated with values from the registers.

Modifies
setup variable

Example

```
CSL_Status        status;
CSL_McaspHwSetup  myHwSetup;
CSL_McaspHandle   hMcasp;
...
status = CSL_mcaspGetHwsetup (hMcasp, &myHwSetup);
...
```

8.2.7 CSL_mcaspHwControl

CSL_Status CSL_mcaspHwControl **(** CSL_McaspHandle ***hMcasp***, CSL_McaspHwControlCmd ***cmd***, **void *** ***arg*** **)**

Description
This function performs various control operations on McASP instance, based on the command passed.

Arguments

hMcasp	Handle to the McASP instance
cmd	Operation to be performed on the McASP
arg	Argument specific to the command

Return Value
CSL_Status

- CSL_SOK - Command execution successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command

Pre Condition
Both *CSL_mcaspInit()* and *CSL_mcaspOpen()* must be called successfully in that order before *CSL_mcaspHwContol()* can be called..

Post Condition
Registers of McASP instance are configured according to the command and the command arguments. The command determines which registers are modified.

Modifies
Registers determined by the command

Example

```
CSL_Status      status;
CSL_McaspHandle hMcasp;
...
status  = CSL_mcaspHwControl (hMcasp,
                              CSL_MCASP_CMD_CLK_RESET_XMT,
                              NULL);
...
```

8.2.8 CSL_mcaspGetHwStatus

CSL_Status CSL_mcaspGetHwStatus (CSL_McaspHandle ***hMcasp***,
CSL_McaspHwStatusQuery ***myQuery***,
void * ***response***
)

Description
This function is used to get the value of various parameters of the McASP instance. The value returned depends on the query passed.

Arguments

hMcasp	Handle to the McASP instance
myQuery	Query to be performed
response	Pointer to buffer to return the data requested by the query passed

Return Value
CSL_Status

- CSL_SOK - Successful completion of the query
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Query command not supported
- CSL_ESYS_FAIL - Generic failure

Pre Condition
Both *CSL_mcaspInit()* and *CSL_mcaspOpen()* must be called successfully in order before calling *CSL_mcaspGetHwStatus()*..

Post Condition
Data requested by query is returned through the variable "response".

Modifies
The input argument "response" is modified.

Example

```
CSL_McaspHandle hMcasp;
CSL_Status      status;
Bool            xmtUnderRun;
...
status = CSL_mcaspGetHwStatus(hMcasp,
                              CSL_MCASP_QUERY_XSTAT_XUNDRN,
                              &xmtUnderRun);
...
```

8.2.9 CSL_mcaspRead

CSL_Status CSL_mcaspRead (CSL_McaspHandle *hMcasp*, Uint32 * *data*)

Description
This reads the data from McASP. 32-bits of data will be read in the data object (variable); the pointer to which is passed as the third argument.

Arguments

```
hMcasp      Handle to the McASP instance

data        Buffer to store read data
```

Return Value
CSL_Status

- CSL_SOK - Successful completion of read
- CSL_ESYS_INVPARAMS - Invalid serializer number
- CSL_ESYS_BADHANDLE – Invalid handle

Pre Condition
Both *CSL_mcaspInit()*, *CSL_mcaspOpen()* and @ CSL_mcaspHwSetup() must be called successfully in that order before *CSL_mcaspRead()* can be called.

Post Condition
Input argument "data" will be modified.

Modifies
"data" argument.

Example

```
CSL_Status        status;
Uint32            inData;
CSL_McaspHandle   hMcasp;
...
status = CSL_mcaspRead (hMcasp, &inData);
...
```

8.2.10 CSL_mcaspWrite

CSL_Status CSL_mcaspWrite (CSL_McaspHandle *hMcasp*, Uint32 * *data*)

Description
This transmits the data from McASP. 32 bits of data will be transmitted in the data object (variable); the pointer to which is passed as the third argument.

Arguments

```
hMcasp      Handle to the McASP instance

data        Data to be written
```

Return Value
CSL_Status

- CSL_SOK - Successful completion of Write
- CSL_ESYS_INVPARAMS - Invalid serializer number
- CSL_ESYS_BADHANDLE - Invalid handle

Pre Condition
1. Both *CSL_mcaspInit()*, *CSL_mcaspOpen()* and @ CSL_mcaspHwSetup() must be called successfully in that order before *CSL_mcaspWrite()* can be called.
2. "data" argument should have the data to be written

Post Condition
XBUF Register will be written.

Modifies
XBUF Register

Example

```
CSL_Status        status;
Uint32            outData;
CSL_McaspHandle   hMcasp;
```

```
...
status = CSL_mcaspWrite (hMcasp, &outData);
...
```

8.2.11 CSL_mcaspRegReset

void CSL_mcaspRegReset (CSL_McaspHandle *hMcasp*)

Description
This function resets all the register values.

Arguments

```
hMcasp          Handle to the McASP instance
```

Return Value
None

Pre Condition
Both *CSL_mcaspInit()* and *CSL_mcaspOpen()* must be called successfully in that order before *CSL_mcaspRegReset()* can be called.

Post Condition
All register values will be reset.

Modifies
All McASP registers.

Example

```
CSL_McaspHandle hMcasp;
...
CSL_mcaspRegReset (hMcasp);
...
```

8.2.12 CSL_mcaspResetCtrl

void CSL_mcaspResetCtrl (CSL_McaspHandle *hMcasp*, CSL_BitMask32 *selectMask*)

Description
This function enables bit fields of GBLCTL register

Arguments

```
hMcasp          Handle to the McASP instance

selectMask      Selects the bits to enable
```

Return Value
None

Pre Condition
Both *CSL_mcaspInit()*, *CSL_mcaspOpen()* and @ CSL_mcaspHwSetup() must be called successfully in that order before *CSL_mcaspResetCtrl()* can be called.

Post Condition
Some GBLCTL register bit fields are enabled.

Modifies
GBLCTL register.

Example

```
CSL_McaspHandle hMcasp;
...
CSL_mcaspResetCtrl (hMcasp, CSL_MCASP_GBLCTL_TXSMRST_ACTIVE);
...
```

8.2.13 CSL_mcaspGetChipCtxt

CSL_Status CSL_mcaspGetChipCtxt (CSL_InstNum *mcaspNum*,
CSL_McaspParam * *pMcaspParam*,
CSL_McaspChipContext * *pChipContext*
)

Description
This function is used for getting the base address of the peripheral instance and to configure the number of serializers for a particular instance of McASP on a chip. This function will be called inside the CSL_mcaspOpen() function. This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral MMR's go to an alternate location.

Arguments

```
mcaspNum      Specifies the instance of the McASP to be opened.

pMcaspParam   Module specific parameters.

pChipContext  Pointer to hold the chip context details
```

Return Value
CSL_Status

- CSL_SOK - Successful completion

- CSL_ESYS_FAIL - The instance number is invalid.

Pre Condition
None

Post Condition
Base address structure is populated.

Modifies
Base address structure is modified.

Example

```
CSL_Status            status;
CSL_McaspChipContext  chipContext;
...
status = CSL_mcaspGetChipCtxt(CSL_MCASP, NULL, &chipContext);
...
```

8.3 Data Structures

This section lists the data structures available in the McASP module.

8.3.1 CSL_McaspObj

Detailed Description
Module specific Object Structure.
McASP object structure

Field Documentation

Bool CSL_McaspObj::ditStatus
Support for DIT mode

Int32 CSL_McaspObj::numOfSerializers
Number of serializers

Int16 CSL_McaspObj::perNo
Specifies a particular instance of McASP

CSL_McaspRegsOvly CSL_McaspObj::regs
Pointer to the register overlay structure for the peripheral

8.3.2 CSL_McaspConfig

Detailed Description
Module specific Configuration structure. This is used to configure McASP instance using CSL_mcaspHwSetupRaw function.

Field Documentation

Uint32 CSL_McaspConfig::ACLKRCTL
Receive clock control register

Uint32 CSL_McaspConfig::ACLKXCTL
Transmit clock control register

Uint32 CSL_McaspConfig::AFSRCTL
Receive frame sync control register

Uint32 CSL_McaspConfig::AFSXCTL
Transmit frame sync control register

Uint32 CSL_McaspConfig::AHCLKRCTL
Receive high-frequency clock control register

Uint32 CSL_McaspConfig::AHCLKXCTL
Transmit high-frequency clock control register

Uint32 CSL_McaspConfig::AMUTE
Audio mute control register

Uint32 CSL_McaspConfig::DITCTL
DIT mode control register

Uint32 CSL_McaspConfig::DLBCTL
Digital loopback control register

Uint32 CSL_McaspConfig::GBLCTL
Global control register

Uint32 CSL_McaspConfig::PDCLR
Pin data clear register

Uint32 CSL_McaspConfig::PDIN_PDSET
Pin data set register

Uint32 CSL_McaspConfig::PDIR
Pin direction register

Uint32 CSL_McaspConfig::PDOUT
Pin data output register

Uint32 CSL_McaspConfig::PFUNC
Pin function register

Uint32 CSL_McaspConfig::PWRDEMU
Power down and emulation management register

Uint32 CSL_McaspConfig::RCLKCHK
Receive clock check control register

Uint32 CSL_McaspConfig::REVTCTL
Receiver DMA event control register

Uint32 CSL_McaspConfig::RFMT
Receive bit stream format register

Uint32 CSL_McaspConfig::RINTCTL
Receiver interrupt control register

Uint32 CSL_McaspConfig::RMASK
Receive format unit bit mask register

Uint32 CSL_McaspConfig::RSTAT
Receiver status register

Uint32 CSL_McaspConfig::RTDM
Receive TDM time slot 0-31 register

Uint32 CSL_McaspConfig::SRCTL0
Serializer control register 0

Uint32 CSL_McaspConfig::SRCTL1
Serializer control register 1

Uint32 CSL_McaspConfig::SRCTL2
Serializer control register 2

Uint32 CSL_McaspConfig::SRCTL3
Serializer control register 3

Uint32 CSL_McaspConfig::SRCTL4
Serializer control register 4

Uint32 CSL_McaspConfig::SRCTL5
Serializer control register 5

Uint32 CSL_McaspConfig::SRCTL6
Serializer control register 6

Uint32 CSL_McaspConfig::SRCTL7
Serializer control register 7

Uint32 CSL_McaspConfig::SRCTL8
Serializer control register 8

Uint32 CSL_McaspConfig::SRCTL9
Serializer control register 9

Uint32 CSL_McaspConfig::SRCTL10
Serializer control register 10

Uint32 CSL_McaspConfig::SRCTL11
Serializer control register 11

Uint32 CSL_McaspConfig::SRCTL12
Serializer control register 12

Uint32 CSL_McaspConfig::SRCTL13
Serializer control register 13

Uint32 CSL_McaspConfig::SRCTL14
Serializer control register 14

Uint32 CSL_McaspConfig::SRCTL15
Serializer control register 15

Uint32 CSL_McaspConfig::XCLKCHK
Transmit clock check control register

Uint32 CSL_McaspConfig::XEVTCTL
Transmitter DMA event control register

Uint32 CSL_McaspConfig::XFMT
Transmit bit stream format register

Uint32 CSL_McaspConfig::XINTCTL
Transmitter interrupt control register

Uint32 CSL_McaspConfig::XMASK

Transmit format unit bit mask register

Uint32 CSL_McaspConfig::XSTAT
Transmitter status register

Uint32 CSL_McaspConfig::XTDM
Transmit TDM time slot 0-31 register

8.3.3 CSL_McaspParam

Detailed Description
Module specific parameters. Present implementation doesn't have any module specific parameters.

Field Documentation

CSL_BitMask32 CSL_McaspParam::flags
Bit mask to be used for module specific parameters. The declaration is just a placeholder for future implementation.

8.3.4 CSL_McaspContext

Detailed Description
Module specific context information. Present implementation doesn't have any context information.

Field Documentation

Uint16 CSL_McaspContext::contextInfo
Context information of McASP. The declaration is just a placeholder for future implementation.

8.3.5 CSL_McaspHwSetup

Detailed Description
Module specific Hardware Setup Structure.

Field Documentation

Uint32 CSL_McaspHwSetup::emu
Power down emulation mode params - PWRDEMU

CSL_McaspHwSetupGbl **CSL_McaspHwSetup::glb**
Value to be loaded in global control register (GLBCTL)

CSL_McaspHwSetupData **CSL_McaspHwSetup::rx**
Receiver settings

CSL_McaspHwSetupData **CSL_McaspHwSetup::tx**
Transmitter settings

8.3.6 CSL_McaspHwSetupGbl

Detailed Description
Module specific hardware setup global structure.

Field Documentation

Uint32 CSL_McaspHwSetupGbl::amute
Mute control register - AMUTE

Uint32 CSL_McaspHwSetupGbl::ctl
Global control register - GBLCTL

Uint32 CSL_McaspHwSetupGbl::ditCtl
Decides whether McASP operates in DIT mode - DITCTL

Uint32 CSL_McaspHwSetupGbl::dlbMode
Digital loopback mode setup - DLBEN

Uint32 CSL_McaspHwSetupGbl::pdir
Pin direction register

Uint32 CSL_McaspHwSetupGbl::pfunc
Pin function register

Uint32 CSL_McaspHwSetupGbl::serSetup[16]
Setup serializer control register (SRCTL0-5)

8.3.7 CSL_McaspHwSetupDataClk

Detailed Description
McASP Hardware Setup data clock structure.

Field Documentation

Uint32 CSL_McaspHwSetupDataClk::clkChk
Configures receive/transmit clock failure detection R/XCLKCHK

Uint32 CSL_McaspHwSetupDataClk::clkSetupClk
Clock details ACLK(R/X)CTL

Uint32 CSL_McaspHwSetupDataClk::clkSetupHiClk
High clock details AHCLK(R/X)CTL

8.3.8 CSL_McaspHwSetupData

Detailed Description
McASP Hardware Setup data structure.

Field Documentation

CSL_McaspHwSetupDataClk **CSL_McaspHwSetupData::clk**
Clock settings for rcv/xmt

Uint32 CSL_McaspHwSetupData::evtCtl
Event control register - R/XEVTCTL

Uint32 CSL_McaspHwSetupData::fmt
Format details as per - R/XFMT

Uint32 CSL_McaspHwSetupData::frSyncCtl
Configure the rcv/xmt frame sync - AFSR/XCTL

Uint32 CSL_McaspHwSetupData::intCtl
Controls generation of McASP interrupts – R/XINTCTL

Uint32 CSL_McaspHwSetupData::mask
To mask or not to mask - R/XMASK

Uint32 CSL_McaspHwSetupData::stat
Status register (controls writable fields of STAT register)-R/XSTAT

Uint32 CSL_McaspHwSetupData::tdm
Specifies which TDM slots are active - R/XTDM

8.3.9 CSL_McaspChStatusRam

Detailed Description
Module specific DIT channel status register structure.

Field Documentation

Uint32 CSL_McaspChStatusRam::chStatusLeft[6]
Left channel status registers (DITCSRA0-5)

Uint32 CSL_McaspChStatusRam::chStatusRight[6]
Right channel status register (DITCSRB0-5)

8.3.10 CSL_McaspUserDataRam

Detailed Description
Module specific DIT channel user data register structure.

Field Documentation

Uint32 CSL_McaspUserDataRam::userDataLeft[6]
Left channel user data registers (DITUDRA0-5)

Uint32 CSL_McaspUserDataRam::userDataRight[6]
Right channel user data registers (DITUDRB0-5)

8.3.11 CSL_McaspSerQuery

Detailed Description
Module specific structure used in CSL_QUERY_SRCTL_RRDY, and CSL_MCASP_QUERY_SRCTL_XRDY.

Field Documentation

CSL_MaspSerializerNum_CSL_McaspSerQuery::SerNum
Serializer number

Bool CSL_McaspSerQuery::serStatus
Return Value of the query

8.3.12 CSL_McaspSerMmode

Detailed Description
The following structure will be used in CSL_MCASP_QUERY_SRCTL_SRMOD

Field Documentation

CSL_MaspSerMode CSL_McaspSerMmode::serMode
Serializer mode

CSL_McaspSerializerNum CSL_McaspSerMmode::serNum
Serializer number

8.3.13 CSL_McaspChipContext

Detailed Description
Module specific base-address information.
This contains the base-address information for the peripheral instance

Field Documentation

Bool CSL_McaspChipContext::ditStatus
Support for DIT mode

Int32 CSL_McaspChipContext::numOfSerializers
Number of serializers

CSL_McaspRegsOvly CSL_McaspChipContext::regs
Base-address of the configuration registers of the peripheral

8.4 Enumerations

This section lists the enumerations available in the McASP module.

8.4.1 CSL_McaspDITRegIndex

enum CSL_McaspDITRegIndex

DIT channel/user Left/right data structure.

Enumeration values:

DIT_REGISTER_0	1st DIT (channel/user data), (left/right) Register
DIT_REGISTER_1	2nd DIT (channel/user data), (left/right) Register
DIT_REGISTER_2	3rd DIT (channel/user data), (left/right) Register
DIT_REGISTER_3	4th DIT (channel/user data), (left/right) Register
DIT_REGISTER_4	5th DIT (channel/user data), (left/right) Register
DIT_REGISTER_5	6th DIT (channel/user data), (left/right) Register

8.4.2 CSL_McaspSerializerNum

enum CSL_McaspSerializerNum
Enumeration for the serializer numbers

Enumeration values:

SERIALIZER_1	SRCTL0
SERIALIZER_2	SRCTL1
SERIALIZER_3	SRCTL2
SERIALIZER_4	SRCTL3
SERIALIZER_5	SRCTL4
SERIALIZER_6	SRCTL5
SERIALIZER_7	SRCTL5
SERIALIZER_8	SRCTL5
SERIALIZER_9	SRCTL5
SERIALIZER_10	SRCTL5
SERIALIZER_11	SRCTL5
SERIALIZER_12	SRCTL5
SERIALIZER_13	SRCTL5
SERIALIZER_14	SRCTL5
SERIALIZER_15	SRCTL5
SERIALIZER_16	SRCTL5

8.4.3 CSL_McaspSerMode

enum CSL_McaspSerMode
Enumeration for various Serializer modes.

Enumeration values:

SERIALIZER_INACTIVE	Serializer is inactive

SERIALIZER_XMT	Serializer is transmitter
SERIALIZER_RCV	Serializer is receiver

8.4.4 CSL_McaspHwControlCmd

enum CSL_McaspHwControlCmd
Enumeration for hardware control commands passed to CSL_mcaspHwControl().
This is used to select the commands to control the operations existing setup of McASP. The arguments to be passed with each enumeration if any are specified next to the enumeration.

Enumeration values:

CSL_MCASP_CMD_SET_XMT	Configure transmitter global control register with parameters passed. **Parameters:** *(Uint32 *)*
CSL_MCASP_CMD_SET_RCV	Configure receiver global control register with parameters passed. **Parameters:** *(Uint32 *)*
CSL_MCASP_CMD_RESET_FSYNC_XMT	Reset transmit frame sync generator. **Parameters:** *(None)*
CSL_MCASP_CMD_RESET_FSYNC_RCV	Reset receive frame sync generator. **Parameters:** *(None)*
CSL_MCASP_CMD_REG_RESET	Reset all registers. **Parameters:** *(None)*
CSL_MCASP_CMD_AMUTE_ON	Mute enable. **Parameters:** *(Uint32 *)*
CSL_MCASP_CMD_DLB_ON	Enable digital loopback mode. **Parameters:** *(Uint32 *)*
CSL_MCASP_CMD_CONFIG_RTDM_SLOT	Configures receive slots. **Parameters:** *(Uint32 *)*
CSL_MCASP_CMD_CONFIG_XTDM_SLOT	Configures transmit slots. **Parameters:** *(Uint32 *)*
CSL_MCASP_CMD_CONFIG_INTERRUPT_RCV	Configures the interrupts on the receive side. **Parameters:** *(Uint32 *)*
CSL_MCASP_CMD_CONFIG_INTERRUPT_XMT	Configures the interrupts on the transmit side. **Parameters:** *(Uint32 *)*
CSL_MCASP_CMD_CLK_RESET_RCV	Reset clock circuitry for receive.

	Parameters: *(None)*
CSL_MCASP_CMD_CLK_RESET_XMT	Reset clock circuitry for transmit. **Parameters:** *(None)*
CSL_MCASP_CMD_CLK_SET_RCV	Set receive clock registers with value (CSL_McaspHwSetupDataClk*) passed. **Parameters:** *(CSL_McaspHwSetupDataClk *)*
CSL_MCASP_CMD_CLK_SET_XMT	Sets transmit clock registers with value (CSL_McaspHwSetupDataClk*) passed. **Parameters:** *(CSL_McaspHwSetupDataClk *)*
CSL_MCASP_CMD_CONFIG_XMT_SECTION	Configure the format, frame sync, and other parameters related to the transmit section. **Parameters:** *(CSL_McaspHwSetupData *)*
CSL_MCASP_CMD_CONFIG_RCV_SECTION	Configure the format, frame sync, and other parameters related to the receive section. **Parameters:** *(CSL_McaspHwSetupData *)*
CSL_MCASP_CMD_SET_SER_XMT	Sets a particular serializer to act as transmitter. **Parameters:** *(CSL_McaspSerializerNum *)*
CSL_MCASP_CMD_SET_SER_RCV	Sets a particular serializer to act as receiver. **Parameters:** *(CSL_McaspSerializerNum *)*
CSL_MCASP_CMD_SET_SER_INA	Sets a particular serializer as inactivated. **Parameters:** *(CSL_McaspSerializerNum *)*
CSL_MCASP_CMD_WRITE_CHAN_STAT_RAM	Writes to the channel status RAM. **Parameters:** *(CSL_McaspChStatusRam *)*
CSL_MCASP_CMD_WRITE_USER_DATA_RAM	Writes to the user data RAM. **Parameters:** *(CSL_McaspUserDataRam *)*
CSL_MCASP_CMD_RESET_XMT	Resets the bits related to transmit in transmitter global control register. **Parameters:** *(None)*
CSL_MCASP_CMD_RESET_RCV	Resets the bits related to receive in transmitter global control register. **Parameters:** *(None)*
CSL_MCASP_CMD_RESET_SM_FS_XMT	Resets transmit frame sync generator and transmit state machine in transmitter global control register.

	Parameters: *(None)*
CSL_MCASP_CMD_RESET_SM_FS_RCV	Resets receive frame sync generator and receive state machine in receiver global control register. **Parameters:** *(None)*
CSL_MCASP_CMD_ACTIVATE_XMT_CLK_SER	Sets the bits related to transmit in transmitter global control register. **Parameters:** *(None)*
CSL_MCASP_CMD_ACTIVATE_RCV_CLK_SER	Sets the bits related to receive in receiver global control register. **Parameters:** *(None)*
CSL_MCASP_CMD_ACTIVATE_SM_RCV_XMT	Activates receive and transmit state machine in global control register. **Parameters:** *(None)*
CSL_MCASP_CMD_ACTIVATE_FS_RCV_XMT	Activates receive and transmit frame sync generator in global control register. **Parameters:** *(None)*
CSL_MCASP_CMD_SET_DIT_MODE	Enables/disables the DIT mode. **Parameters:** *(Bool *)*

8.4.5 CSL_McaspHwStatusQuery

enum CSL_McaspHwStatusQuery
Enumeration for hardware status query commands passed to *CSL_mcaspGetHwStatus()*.This is used to get the status of different operations or to get the existing setup of McASP.

Enumeration values:

CSL_MCASP_QUERY_CURRENT_XSLOT	Return current transmit slot being transmitted. **Parameters:** *(Uint16 *)*
CSL_MCASP_QUERY_CURRENT_RSLOT	Return current receive slot being received. **Parameters:** *(Uint16 *)*
CSL_MCASP_QUERY_XSTAT_XERR	Return transmit error status bit. **Parameters:** *(Bool *)*
CSL_MCASP_QUERY_XSTAT_XCLKFAIL	Return transmit clock failure flag status. **Parameters:** *(Bool *)*
CSL_MCASP_QUERY_XSTAT_XSYNCERR	Return unexpected transmit frame sync flag status.

	Parameters: *(Bool *)*
CSL_MCASP_QUERY_XSTAT_XUNDRN	Return transmit underrun flag status. **Parameters:** *(Bool *)*
CSL_MCASP_QUERY_XSTAT_XDATA	Return transmit data ready flag status. **Parameters:** *(Bool *)*
CSL_MCASP_QUERY_RSTAT_RERR	Return receive error status bit. **Parameters:** *(Bool *)*
CSL_MCASP_QUERY_RSTAT_RCLKFAIL	Return receive clock failure flag status. **Parameters:** *(Bool *)*
CSL_MCASP_QUERY_RSTAT_RSYNCERR	Return unexpected receive frame sync flag status. **Parameters:** *(Bool *)*
CSL_MCASP_QUERY_RSTAT_ROVRN	Return receive overrun flag status. **Parameters:** *(Bool *)*
CSL_MCASP_QUERY_RSTAT_RDATA	Return receive data ready flag status. **Parameters:** *(Bool *)*
CSL_MCASP_QUERY_SRCTL_RRDY	Return status whether the serializer is ready to receive or not. **Parameters:** *(CSL_McaspSerQuery *)*
CSL_MCASP_QUERY_SRCTL_XRDY	Return status whether the serializer is ready to transmit or not. **Parameters:** *(CSL_McaspSerQuery *)*
CSL_MCASP_QUERY_SRCTL_SRMOD	Return status whether serializer is configured to transmit/receive/inactive. **Parameters:** *(CSL_McaspSerQuery *)*
CSL_MCASP_QUERY_XSTAT	Return the value of transmitter status register. **Parameters:** *(Uint16 *)*
CSL_MCASP_QUERY_RSTAT	Return the value of receiver status register. **Parameters:** *(Uint16 *)*
CSL_MCASP_QUERY_SM_FS_XMT	Return transmit state machine and transmit frame sync generator values in transmitter global control register. **Parameters:** *(Uint8 *)* • 0x00 - Both transmit frame generator sync and transmit state machine are reset.

- 0x1 - Only transmit state machine is active.
- 0x10 - Only transmit frame sync generator is active.
- 0x11 - Both transmit frame generator sync and transmit state machine are active.

CSL_MCASP_QUERY_SM_FS_RCV

Return receive state machine and receive frame sync generator values in receiver global control register.

Parameters:

*(Uint8 *)*

- 0x00 - Both receive frame generator sync and receive state machine are reset.
- 0x1 - Only receive state machine is active.
- 0x10 - Only receive frame sync generator is active.
- 0x11 - Both receive frame generator sync and receive state machine are active.

CSL_MCASP_QUERY_DIT_MODE

Queries whether DIT mode is set or not.

Parameters:

*(Bool *)*

8.5 Macros

#define MCASP0_TXBUF_ADDR 0x54000000
McASP0 Tx Buffer Address

#define MCASP0_RXBUF_ADDR 0x54000000
McASP0 Rx Buffer Address

#define MCASP1_TXBUF_ADDR 0x55000000
McASP1 Tx Buffer Address

#define MCASP1_RXBUF_ADDR 0x55000000
McASP1 Rx Buffer Address

#define MCASP2_TXBUF_ADDR 0x56000000
McASP2 Tx Buffer Address

#define MCASP2_RXBUF_ADDR 0x56000000
McASP2 Rx Buffer Address

#define CSL_MCASP_CONFIG_DEFAULTS
Value:

```
{\
     CSL_MCASP_PWRDEMU_RESETVAL,\
     CSL_MCASP_PFUNC_RESETVAL,\
     CSL_MCASP_PDIR_RESETVAL,\
     CSL_MCASP_PDOUT_RESETVAL,\
     CSL_MCASP_PDIN_PDSET_RESETVAL,\
     CSL_MCASP_PDCLR_RESETVAL,\
     CSL_MCASP_GBLCTL_RESETVAL,\
     CSL_MCASP_AMUTE_RESETVAL,\
     CSL_MCASP_DLBCTL_RESETVAL,\
     CSL_MCASP_DITCTL_RESETVAL,\
     CSL_MCASP_RMASK_RESETVAL,\
     CSL_MCASP_RFMT_RESETVAL,\
     CSL_MCASP_AFSRCTL_RESETVAL,\
     CSL_MCASP_ACLKRCTL_RESETVAL,\
     CSL_MCASP_AHCLKRCTL_RESETVAL,\
     CSL_MCASP_RTDM_RESETVAL,\
     CSL_MCASP_RINTCTL_RESETVAL,\
     CSL_MCASP_RSTAT_RESETVAL,\
     CSL_MCASP_RCLKCHK_RESETVAL,\
     CSL_MCASP_REVTCTL_RESETVAL,\
     CSL_MCASP_XMASK_RESETVAL,\
     CSL_MCASP_XFMT_RESETVAL,\
     CSL_MCASP_AFSXCTL_RESETVAL,\
     CSL_MCASP_ACLKXCTL_RESETVAL,\
     CSL_MCASP_AHCLKXCTL_RESETVAL,\
     CSL_MCASP_XTDM_RESETVAL,\
     CSL_MCASP_XINTCTL_RESETVAL,\
     CSL_MCASP_XSTAT_RESETVAL,\
     CSL_MCASP_XCLKCHK_RESETVAL,\
     CSL_MCASP_XEVTCTL_RESETVAL,\
```

```
    CSL_MCASP_SRCTL0_RESETVAL,\
    CSL_MCASP_SRCTL1_RESETVAL,\
    CSL_MCASP_SRCTL2_RESETVAL,\
    CSL_MCASP_SRCTL3_RESETVAL,\
    CSL_MCASP_SRCTL4_RESETVAL,\
    CSL_MCASP_SRCTL5_RESETVAL,\
    CSL_MCASP_SRCTL6_RESETVAL,\
    CSL_MCASP_SRCTL7_RESETVAL,\
    CSL_MCASP_SRCTL8_RESETVAL,\
    CSL_MCASP_SRCTL9_RESETVAL,\
    CSL_MCASP_SRCTL10_RESETVAL,\
    CSL_MCASP_SRCTL11_RESETVAL,\
    CSL_MCASP_SRCTL12_RESETVAL,\
    CSL_MCASP_SRCTL13_RESETVAL,\
    CSL_MCASP_SRCTL14_RESETVAL,\
    CSL_MCASP_SRCTL15_RESETVAL\
 }
```

Module specific Reset values for the config structure.

#define CSL_MCASP_HWSETUP_DEFAULTS
Value:

```
{ \
    {0,0,0,0, {0,0,0,0,0,0}},\
    {0,0,0,0,0,0,0,{0x20,0x8000,0}},\
    {0,0,0,0,0,0,0,{0x20,0x8000,0}},\
    0\
}
```

Module specific Default values for the Hardware setup structure.

Chapter 9
PLLC Module

Topics

9.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within PLLC module.

The PLL controller features software-configurable PLL multiplier controller (PLLM), dividers (D0, D1, D2, and D3), and reset controller. The PLL controller offers flexibility and convenience by way of software-configurable multiplier and dividers to modify the input signal internally. The resulting clock outputs are passed to the DSP core, peripherals, and other modules inside the DSP.

- The input reference clocks to the PLL controller:
  - OSCIN: output signal from on-chip oscillator

- The resulting output clocks from the PLL controller:
  - AUXCLK: internal clock output signal directly from CLKIN or OSCIN.
  - SYSCLK1: internal clock output of divider D1.
  - SYSCLK2: internal clock output of divider D2.
  - SYSCLK3: internal clock output of divider D3.

9.2 Functions

This section lists the functions available in the PLLC module.

9.2.1 CSL_pllcInit

CSL_Status CSL_pllcInit (CSL_PllcContext * *pContext*)

Description
This is the initialization function for the pllc CSL. The function must be called before calling any other API from this CSL. This function does not modify any registers or check status. It returns status CSL_SOK. It has been kept for future use.

Arguments

```
pContext    Pointer to module-context. As per doesn't have
            any context based information user is expected to
            pass NULL.
```

Return Value
CSL_Status

- CSL_SOK - Always returns

Pre Condition
CSL_sysInit() must be called.

Post Condition
None

Modifies
None

Example

```
CSL_Status  status;
...
CSL_pllcInit(NULL);
...
```

9.2.2 CSL_pllcOpen

CSL_PllcHandle CSL_pllcOpen

(CSL_PllcObj * *pPllcObj*,
CSL_InstNum *pllcNum*,
CSL_PllcParam * *pPllcParam*,
CSL_Status * *pStatus*
)

Description
This function returns the handle to the PLL controller instance. This handle is passed to all other CSL APIs.

Arguments

```
pllcObj     Pointer to pllc object.

pllcNum     Instance of pll CSL to be opened.
            There is only one instance of the pllc available.
            So, the value for this parameter will always be
            CSL_PLLC.

pPllcParam  Module specific parameters.

pStatus     Status of the function call
```

Return Value
CSL_PllcHandle
Valid pllc handle is returned, if status value is equal to CSL_SOK.

Pre Condition
CSL_pllcInit() must be called successfully before calling this function.

Post Condition
1. The status is returned in the status variable. If status returned is

- CSL_SOK - Valid pllc handle is returned
- CSL_ESYS_FAIL - The pllc instance is invalid
- CSL_ESYS_INVPARAMS - Invalid parameter

2. PLLC object structure is populated.

Modifies
1. The status variable
2. PLLC object structure

Example

```
CSL_Status          status;
CSL_PllcObj         pllcObj;
CSL_PllcHandle      hPllc;
...
hPllc = CSL_pllcOpen(&pllcObj, CSL_PLLC, NULL, &status);
...
```

9.2.3 CSL_pllcClose

CSL_Status CSL_pllcClose (CSL_PllcHandle *hPllc*)

Description
This function closes the specified instance of PLLC.

Arguments

```
hPllc            Handle to the pllc
```

Return Value
CSL_Status

- CSL_SOK - Close successful
- CSL_ESYS_BADHANDLE - Invalid handle

Pre Condition
Both CSL_pllcInit() and CSL_pllcOpen() must be called successfully in order before calling this function.

Post Condition
None

Modifies
The peripheral object structure.

Example

```
CSL_PllcHandle   hPllc;
CSL_Status       status;
...
status = CSL_pllcClose(hPllc);
...
```

9.2.4 CSL_pllcHwSetup

CSL_Status CSL_pllcHwSetup (CSL_PllcHandle *hPllc*, CSL_PllcHwSetup * *hwSetup*)

Description
It configures the pllc registers as per the values passed in the hardware setup structure.

Arguments

```
hPllc            Handle to the pllc

hwSetup          Pointer to hardware setup structure
```

Return Value
CSL_Status

- CSL_SOK - Hardware setup successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Hardware structure is not properly initialized

Pre Condition
Both CSL_pllcInit() and CSL_pllcOpen() must be called successfully in order before calling this function.

Post Condition
PLL controller registers are configured according to the hardware setup parameters

Modifies
PLL controller registers

Example

```
CSL_PllcHandle   hPllc;
CSL_PllcObj      pllcObj;
CSL_PllcHwSetup  hwSetup;
CSL_Status       status;

...

hPllc = CSL_pllcOpen(&pllcObj, CSL_PLLC, NULL, &status);

status = CSL_pllcHwSetup(hPllc, &hwSetup);
...
```

9.2.5 CSL_pllcHwControl

CSL_Status CSL_pllcHwControl (CSL_PllcHandle ***hPllc***,
CSL_PllcHwControlCmd ***cmd***,
void * ***arg***
)

Description
Takes a command of PLLC with an optional argument and implements it.

Arguments

```
hPllc       Handle to the PLLC instance

cmd         The command to this API indicates the action to be
            taken on PLLC.

arg         An optional argument.
```

Return Value
CSL_Status

- CSL_SOK - Status info return successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command

- CSL_ESYS_INVPARAMS - Invalid parameter
- CSL_ESYS_NOTSUPPORTED - Action Not Supported

Pre Condition
Both CSL_pllcInit() and CSL_pllcOpen() must be called successfully in order before calling this function.

Post Condition
PLLC registers are configured according to the command and the command arguments. The command determines which registers are modified.

Modifies
The hardware registers of PLLC.

Example

```
CSL_PllcHandle         hPllc;
CSL_PllcHwControlCmd   cmd = CSL_PLLC_CMD_SET_MODE;
CSL_PllcMode           arg = CSL_PLLC_PLL_MODE;
...
status = CSL_pllcHwControl (hPllc, cmd, &arg);
...
```

9.2.6 CSL_pllcGetHwStatus

CSL_Status CSL_pllcGetHwStatus **(** CSL_PllcHandle ***hPllc***,
CSL_PllcHwStatusQuery ***query***,
void * ***response***
)

Description
Gets the status of the different operations of PLLC.

Arguments

```
hPllc      Handle to the PLLC instance

query      The query to this API of PLLC that indicates the
           status to be returned.

response   Placeholder to return the status.
```

Return Value
CSL_Status

- CSL_SOK - Status info return successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Invalid query command
- CSL_ESYS_INVPARAMS - Invalid parameter
- CSL_ESYS_NOTSUPPORTED- Action Not Supported

Pre Condition
Both CSL_pllcInit() and CSL_pllcOpen() must be called successfully in order before calling this function

Post Condition
None

Modifies
The input argument "response" is modified.

Example

```
CSL_PllcHandle          hPllc;
CSL_PllcHwStatusQuery   query= CSL_PLLC_QUERY_OSC_STATE;
CSL_PllcState            response;
...
status = CSL_pllcGetHwStatus (hPllc, query, &response);
...
```

9.2.7 CSL_pllcHwSetupRaw

CSL_Status CSL_pllcHwSetupRaw **(** CSL_PllcHandle ***hPllc***, CSL_PllcConfig ***** ***config*** **)**

Description
This function initializes the device registers with the register-values provided through the Config Data structure. This configures registers based on a structure of register values, as compared to HwSetup, which configures registers based on structure of bit field values and may perform other functions (delays, etc.)

Arguments

hPllc	Handle to the PLLC instance
config	Pointer to config structure

Return Value
CSL_Status

- CSL_SOK - Configuration successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Configuration is not properly initialized
- CSL_ESYS_NOTSUPPORTED - Action not supported. (System not in bypass mode. If the mode is not bypass mode and the developer tries to write into D0 or PLLM registers or tries to set the pllc in powerdown state, then this error status is returned)

Pre Condition
Both CSL_pllcInit() and CSL_pllcOpen() must be called successfully in order before calling this function

Post Condition

The registers of the specified PLLC instance will be setup according to input configuration structure values.

Modifies
Hardware registers of the specified PLLC instance.

Example

```
CSL_PllcHandle       hPllc;
CSL_PllcConfig       config = CSL_PLLC_CONFIG_DEFAULTS;
CSL_Status           status;
...
status = CSL_pllcHwSetupRaw (hPllc, &config);
...
```

9.2.8 CSL_pllcGetHwSetup

CSL_Status CSL_pllcGetHwSetup (CSL_PllcHandle *hPllc*, CSL_PllcHwSetup * *hwSetup*)

Description
It retrieves the hardware setup parameters of the pllc specified by the given handle.

Arguments

```
hPllc        Handle to the pllc

hwSetup      Pointer to the hardware setup structure
```

Return Value
CSL_Status

- CSL_SOK - Retrieving the hardware setup parameters is successful
- CSL_ESYS_BADHANDLE - The handle is passed is invalid
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition
Both CSL_pllcInit() and CSL_pllcOpen() must be called successfully in order before calling this function.

Post Condition
The hardware setup structure is populated with the hardware setup parameters

Modifies
hwSetup variable

Example

```
CSL_PllcHandle   hPllc;
```

```
CSL_PllcHwSetup  hwSetup;
CSL_Status       status;
...
status = CSL_pllcGetHwSetup(hPllc, &hwSetup);
...
```

9.2.9 CSL_pllcGetBaseAddress

CSL_Status CSL_pllcGetBaseAddress (CSL_InstNum *pllcNum*,
CSL_PllcParam * *pPllcParam*,
CSL_PllcBaseAddress * *pBaseAddress*
)

Description
This function is used for getting the base address of the peripheral instance. This function will be called inside the CSL_pllcOpen() function call. This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral. MMRs go to an alternate location.

Arguments

```
pllcNum      Specifies the instance of the pllc to be opened

pPllcParam   Module specific parameters.

pBaseAddress Pointer to base address structure containing base
             address details.
```

Return Value
CSL_Status

- CSL_SOK Open call is successful
- CSL_ESYS_FAIL The instance number is invalid.
- CSL_ESYS_INVPARAMS Invalid parameter

Pre Condition
None

Post Condition
Base address structure is populated.

Modifies
1. The status variable
2. Base address structure is modified.

Example

```
CSL_Status           status;
CSL_PllcBaseAddress  baseAddress;
...
```

```
        status = CSL_pllcGetBaseAddress(CSL_PLLC_CNT, NULL,
                                        &baseAddress);
        ...
```

9.3 Data Structures

This section lists the data structures available in the PLLC module.

9.3.1 CSL_PllcObj

Detailed Description
PLLC object structure.

Field Documentation

CSL_InstNum CSL_PllcObj::pllcNum
Instance of pllc being referred by this object

CSL_PllcRegsOvly CSL_PllcObj::regs
Pointer to the register overlay structure of the pllc

9.3.2 CSL_PllcContext

Detailed Description
Module specific context information. Present implementation of pllc CSL doesn't have any context information.

Field Documentation

Uint16 CSL_PllcContext::contextInfo
Context information of pllc CSL. The declaration is just a placeholder for future implementation.

9.3.3 CSL_PllcParam

Detailed Description
Module specific parameters. Present implementation of pllc CSL doesn't have any module specific parameters.

Field Documentation

CSL_BitMask16 CSL_PllcParam::flags
Bit mask to be used for module specific parameters. The declaration is just a placeholder for future implementation.

9.3.4 CSL_PllcHwSetup

Detailed Description
Hardware setup structure for PLLC.

Field Documentation

CSL_PllcDivEnable CSL_PllcHwSetup::div0Enable
Divider 0 enable/disable and its divider ratio

CSL_PllcDivEnable CSL_PllcHwSetup::div1Enable

Divider 1 enable/disable and its divider ratio

CSL_PllcDivEnable **CSL_PllcHwSetup::div2Enable**
Divider 2 enable/disable and its divider ratio

CSL_PllcDivEnable **CSL_PllcHwSetup::div3Enable**
Divider 3 enable/disable and its divider ratio

void* CSL_PllcHwSetup::extendSetup
Setup that can be used for future implementation

CSL_PllcMode **CSL_PllcHwSetup::pllcMode**
PLLC Mode BYPASS or PLL

Uint32 CSL_PllcHwSetup::pllM
PLL Multiplier factor

9.3.5 CSL_PllcConfig

Detailed Description
Config-structure.
Used to configure the pllc using CSL_wdtHwSetupRaw()

Field Documentation

volatile Uint32 CSL_PllcConfig::ALNCTL
PLL Align control register

volatile Uint32 CSL_PllcConfig::CKEN
PLL Clock enable register

volatile Uint32 CSL_PllcConfig::PLLCMD
PLL Command register

volatile Uint32 CSL_PllcConfig::PLLCSR
pllc control/status register

volatile Uint32 CSL_PllcConfig::PLLDIV0
pllc divider 0 register
volatile Uint32 CSL_PllcConfig::PLLDIV1
pllc divider 1 register

volatile Uint32 CSL_PllcConfig::PLLDIV2
pllc divider 2 register

volatile Uint32 CSL_PllcConfig::PLLDIV3
pllc divider 3 register

volatile Uint32 CSL_PllcConfig::PLLM
pllc multiplier control register

9.3.6 CSL_PllcDivEnable

Detailed Description
This structure describes the divider enable state and its ratio.

Field Documentation

CSL_PllcDivState CSL_PllcDivEnable::divEnable
Divider State (Enabled/Disabled)

Uint32 CSL_PllcDivEnable::pllDivRatio
Divider ratio

9.3.7 CSL_PllcDivCntrl

Detailed Description
This structure describes the divider, its state and divider ratio.
It can be either enabled/disabled.

Field Documentation

CSL_PllcDivNum CSL_PllcDivCntrl::divNum
Divider number (DIV0..DIV3, OD1)

9.3.8 CSL_PllcSysClkStatus

Detailed Description
This structure describes the sysclk and its corresponding status of the clock.

Field Documentation

CSL_PllcSysClk CSL_PllcClkStatus::sysClock
SYSCLK1 to SSYCLK8

CSL_PllcClkStatus CSL_PllcClkStatus::pllClkState
Shows the status Disabled/Enabled

9.3.9 CSL_PllcBaseAddress

Detailed Description
This structure contains the base-address information for the peripheral instance of the PLLC.

Field Documentation

CSL_PllcRegsOvly CSL_PllcBaseAddress::regs
Base-address of the configuration registers of the peripheral

9.4 Enumerations

This section lists the enumerations available in the PLLC module.

9.4.1 CSL_PllcMode

enum CSL_PllcMode
This enum describes the modes of the pllc.
It can either be in bypass/pll mode.

Enumeration values:

CSL_PLLC_BYPASS	The pllc is in bypass mode.
CSL_PLLC_PLL_MODE	The pllc is in pll mode.

9.4.2 CSL_PllcResetState

enum CSL_PllcResetState
This enum describes the reset states of the pllc.
It can be either released/asserted.

Enumeration values:

CSL_PLLC_RELEASE	The pllc reset is released.
CSL_PLLC_ASSERT	The pllc reset is asserted.

9.4.3 CSL_PllcState

enum CSL_PllcState
This enum describes the states of the pllc.
It can be either in operational/power down state.

Enumeration values:

CSL_PLLC_OPERATIONAL	The pllc is in operational state.
CSL_PLLC_PWRDN	The pllc is in power down state.

9.4.4 CSL_PllcOscStableState

enum CSL_PllcOscStableState
This enum describes the states of the Oscillator power down.
It can be either in operational/power down state.

Enumeration values:

CSL_PLLC_CLK_STABLE	The pllc is in operational state.
CSL_PLLC_CLK_NOT_STABLE	The pllc is in power down state.

9.4.5 CSL_PllcGoStatus

enum CSL_PllcGoStatus
This enum describes the GO Operation Enabled state.

It can be in either Enabled or disabled.

Enumeration values:

CSL_PLLC_GO_OPRN_DISABLE	The GO Operation is disabled.
CSL_PLLC_GO_OPRN_IN_PROGRESS	The GO Operation is in progress.

9.4.6 CSL_PllcClkState

enum CSL_PllcClkState
This enum describes the various clocks' enable status.
It can be either enabled/disabled.

Enumeration values:

CSL_PLLC_CLK_ENABLE	Clock is enabled.
CSL_PLLC_CLK_DISABLE	Clock is disabled.

9.4.7 CSL_PllcClockStatus

enum CSL_PllcClockStatus
This enum describes the clocks' enable status.
It can be either ON/Gated.

Enumeration values:

CSL_PLLC_CLK_ON	Clock is On.
CSL_PLLC_CLK_GATED	Clock is Gated.

9.4.8 CSL_PllcDivNum

enum CSL_PllcDivNum
This enum describes the various dividers

Enumeration values:

CSL_PLLC_DIV0	Divider 0 selected
CSL_PLLC_DIV1	Divider 1 selected
CSL_PLLC_DIV2	Divider 2 selected
CSL_PLLC_DIV3	Divider 3 selected

9.4.9 CSL_PllcSysClk

enum CSL_PllcSysClk
This enum describes all the possible SYSCLKs.

Enumeration values:

CSL_PLLC_SYSCLK1	SYSCLK1
CSL_PLLC_SYSCLK2	SYSCLK2
CSL_PLLC_SYSCLK3	SYSCLK3

9.4.10 CSL_PllcDivState

Enum CSL_PllcDivState
This enum describes the states of the pllc dividers.
It can be either enabled/disabled.

Enumeration values:

CSL_PLLC_PLLDIV_ENABLE	The pllc divider is enabled.
CSL_PLLC_PLLDIV_DISABLE	The pllc divider 0 is disabled.

9.4.11 CSL_PllcHwControlCmd

enum CSL_PllcHwControlCmd
This enum describes control commands passed to CSL_pllcHwControl().
This is the set of commands that are passed to CSL_pllcHwControl() with an optional argument type-casted to a void* . The arguments to be passed with each enumeration (if any) are specified next to the enumeration.

Enumeration values:

CSL_PLLC_CMD_SET_MODE	Set PLL mode: Either bypass/ pll mode Set PLL state: Either power down/ operational state.
CSL_PLLC_CMD_SET_PLL_STATE	Set Oscillator state: Either power down/ operational state.
CSL_PLLC_CMD_SET_OSC_STATE	Assert reset to pll: Either reset released/ asserted.
CSL_PLLC_CMD_RESET_CONTROL	Set GO Operation: Either Enable/Disable GO Operation.
CSL_PLLC_CMD_SET_GO_OPN	Controls aligning with the corresponding SYSCLK.
CSL_PLLC_CMD_ALIGN_CONTROL	Set PLLM multiplier factor.
CSL_PLLC_CMD_SET_PLLM_MULFACTOR	Controls the given divider by enabling/disabling the divider and/or setting the divider ratio of the given divider.
CSL_PLLC_CMD_DIV_CONTROL	Controls which clock is output to OBSCLK.
CSL_PLLC_CMD_AUCXLK_CONTROL	Controls the AuxCLK enable/disable

9.4.12 CSL_PllcHwStatusQuery

enum CSL_PllcHwStatusQuery
This enum describes queries passed to CSL_PllcGetHwStatus().
This is used to get the status of different operations. The arguments to be passed with each enumeration if any are specified next to the enumeration.

Enumeration values:

CSL_PLLC_QUERY_MODE	Queries PLL Controller Mode (bypass/ pll) Queries PLL Controller State (operational/ powerdown). **Parameters:**

	(CSL_PllcState)*
CSL_PLLC_QUERY_STATE	Queries Oscillator Powerdown State (operational/ powerdown). **Parameters:** *(CSL_PllcState*)*
CSL_PLLC_QUERY_OSC_STATE	Queries PLL Controller Reset (reset released/ asserted). **Parameters:** *(CSL_PllcResetState*)*
CSL_PLLC_QUERY_RESET	Queries PLL Controller DISABLE (Disable released/ asserted). **Parameters:** *(CSL_PllcDisableState*)*
CSL_PLLC_QUERY_DISABLE	Queries Oscillator Input stable state (stable/ Not stable). **Parameters:** *(CSL_PllcOscStableState*)*
CSL_PLLC_QUERY_OSC_STABLE_STATE	Queries PLL Core Lock Status (LOCKED / UNLOCKED). **Parameters:** *(CSL_PllcLockState*)*
CSL_PLLC_QUERY_LOCK_STATE	Queries Go Operation Enabled Status (Enabled/Disabled). **Parameters:** *(CSL_PllcGoState*)*
CSL_PLLC_QUERY_GOEN_STATE	Queries Go transition Status (Enabled/Disabled). **Parameters:** *(CSL_PllcGoStatus*)*
CSL_PLLC_QUERY_GOSTAT	Queries PLL Controller Multiplier multiplication factor. **Parameters:** *(CSL_BitMask16*)*
CSL_PLLC_QUERY_PLLM_MUL_FACTOR	Queries PLL Controller Divider state (enabled/ disabled) and its divider ratio. **Parameters:** *(CSL_PllcDivCntrl*)*
CSL_PLLC_QUERY_DIVIDER_STATE	Queries OBSCLK Selected state. **Parameters:** *(CSL_PllcObsclkSelState*)*
CSL_PLLC_QUERY_CLK_STATE	Queries the given SYSCLK clock state (ON/Gated). **Parameters:** *(CSL_PllcSysClkStatus*)*

9.5 Macros

#define CSL_PLLC_CONFIG_DEFAULTS
Value:

```
{  \
        CSL_PLLC_PLLCSR_RESETVAL,   \
        CSL_PLLC_PLLM_RESETVAL,     \
        CSL_PLLC_PLLDIV0_RESETVAL,  \
        CSL_PLLC_PLLDIV1_RESETVAL,  \
        CSL_PLLC_PLLDIV2_RESETVAL,  \
        CSL_PLLC_PLLDIV3_RESETVAL,  \
        CSL_PLLC_PLLCMD_RESETVAL,   \
        CSL_PLLC_ALNCTL_RESETVAL,   \
        CSL_PLLC_CKEN_RESETVAL      \
}
```

Default values for config structure.

#define CSL_PLLC_HWSETUP_DEFAULTS
Value:

```
{  \
        CSL_PLLC_BYPASS, \
            {CSL_PLLC_PLLDIV_ENABLE,0x0},\
            {CSL_PLLC_PLLDIV_ENABLE,0x0},\
            {CSL_PLLC_PLLDIV_ENABLE,0x1},\
            {CSL_PLLC_PLLDIV_ENABLE,0x2},\
        CSL_PLLC_PLLM_RESETVAL,     \
        NULL                           \
}
```

Default hardware setup parameters.

#define DELAY_200MS   6000
pllc global macro declarations

Chapter 10
RTI Module

Topics

10.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within RTI module.

The Real Time Interrupt Module (RTI) provides timer functionality for operating systems and for benchmarking code. The Module incorporates several counters, which define the time bases needed for scheduling in the operating system.

The timers also give the ability to benchmark certain areas of code by reading
the counter contents at the beginning and the end of the desired code range and calculating the difference between the values.

The following features are supported on :

- Clock Free Running Counters from RTICLK = SYSCLK2
- Four Output Compares for Event Generation
- Two Input Captures to Measure Input Sample Rates With Respect to SYSCLK2
- Digital Watchdog {disabled by default, must be enabled by software}
- Interrupt and DMA Event Generation

10.2 Functions

This section lists the functions available in the RTI module.

10.2.1 CSL_rtiInit

CSL_Status CSL_rtiInit (CSL_RtiContext * *pContext*)

Description
This is the initialization function for the real time interrupt CSL. This function needs to be called before any RTI CSL functions are to be called. This function is idem-potent. Currently, this function does not perform anything.

Arguments

```
pContext    Pointer to module-context. As real time interrupt
            doesn't have any context based information user is
            expected to pass NULL.
```

Return Value
CSL_Status

- CSL_SOK - Always returns

Pre Condition
CSL_sysInit() must be called.

Post Condition
None

Modifies
None

Example

```
CSL_Status       status;
...

status = CSL_rtiInit(NULL);
...
```

10.2.2 CSL_rtiOpen

CSL_RtiHandle CSL_rtiOpen
(CSL_RtiObj * *pRtiObj*,
CSL_InstNum *rtiNum*,
CSL_RtiParam * *pRtiParam*,
CSL_Status * *pStatus*
)

Description
This function returns the handle to the Real time interrupt controller instance. This handle is passed to all other CSL APIs.

Arguments

```
pRtiObj     Pointer to Real time interrupt object.

rtiNum      Instance of Real time interrupt CSL to be opened.
            There is only one instance of the Real time interrupt
            available. Therefore, the value for this parameter
            will be CSL_RTI always.

pRtiParam   Module specific parameters.

pStatus     Status of the function call
```

Return Value
CSL_RtiHandle
Valid Real time interrupt handle will be returned if status value is equal to CSL_SOK.

Pre Condition
The RTI must be successfully initialized via CSL_rti*Init*() before calling this function

Post Condition
1. The status is returned in the status variable. If status returned is

- CSL_SOK Valid - Real time interrupt handle is returned
- CSL_ESYS_FAIL - The Real time interrupt instance is invalid

2. Real time interrupt object structure is populated.

Modifies
1. The status variable
2. Real time interrupt object structure

Example

```
CSL_Status              status;
CSL_RtiObj              rtiObj;
CSL_RtiHandle           hRti;
...
hRti = CSL_rtiOpen (&rtiObj, CSL_RTI, NULL, &status);
...
```

10.2.3 CSL_rtiClose

CSL_Status CSL_rtiClose (CSL_RtiHandle *hRti*)

Description
This function marks that CSL for the real time interrupt instance is closed. CSL for the real time interrupt instance need to be reopened before using any real time interrupt CSL API.

Arguments

```
hRti          Handle to the real time interrupt instance
```

Return Value
CSL_Status

- CSL_SOK - Real Time Interrupt is closed successfully.
- CSL_ESYS_BADHANDLE - The handle passed is invalid

Pre Condition
Both CSL_rtiInit() and CSL_rtiOpen() must be called successfully in order before calling this function.

Post Condition
The real time interrupt CSL APIs can not be called until the real time interrupt CSL is reopened again using CSL_rtiOpen()

Modifies
None

Example

```
CSL_RtiHandle   hRti;
CSL_Status      status;
...
status = CSL_rtiClose (hRti);
...
```

10.2.4 CSL_rtiHwSetup

CSL_Status CSL_rtiHwSetup (CSL_RtiHandle *hRti*, CSL_RtiHwSetup * *hwSetup*)

Description
It configures the Real time interrupt registers as per the values passed in the hardware setup structure.

Arguments

```
hRti            Handle to the  Real time interrupt

hwSetup         Pointer to hardware setup structure
```

Return Value
CSL_Status

- CSL_SOK - Hardware setup successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Hardware structure is not properly initialized

Pre Condition
Both CSL_rtiInit() and CSL_rtiOpen() must be called successfully in order before calling this function.

Post Condition
Real time interrupt registers are configured according to the hardware setup parameters.

Modifies
Real time interrupt registers

Example

```
CSL_RtiHandle           hRti;
CSL_RtiObj              rtiObj;
CSL_Status              status;
CSL_RtiHwSetup          hwSetup;
...
hRti = CSL_rtiOpen (&rtiObj, CSL_RTI, NULL, &status);

hwSetup.contOnSuspend = CSL_RTI_COUNTERS_RUN;
hwSetup.blk0ExtnCntl = CSL_RTI_CAPTURE_EVENT_SOURCE0;
hwSetup.blk1ExtnCntl = CSL_RTI_CAPTURE_EVENT_SOURCE0;
hwSetup.compare0Cntl = CSL_RTI_FRC0_COMPARE_ENABLE;
hwSetup.compare1Cntl = CSL_RTI_FRC0_COMPARE_ENABLE;
hwSetup.compare2Cntl = CSL_RTI_FRC0_COMPARE_ENABLE;
hwSetup.compare3Cntl = CSL_RTI_FRC0_COMPARE_ENABLE;
hwSetup.counters.frc0Counter = FRC0_TEST_VALUE;
hwSetup.counters.uc0Counter = UC0_TEST_VALUE;
hwSetup.counters.frc1Counter = FRC1_TEST_VALUE;
hwSetup.counters.uc1Counter = UC1_TEST_VALUE;
hwSetup.compareUpCntrs.compareUpCntr0 = UC0_COMP_VALUE;
hwSetup.compareUpCntrs.compareUpCntr1 = UC1_COMP_VALUE;
hwSetup.compVal.comp0Val = COMPARE0_VALUE;
hwSetup.compVal.comp1Val = COMPARE1_VALUE;
hwSetup.compVal.comp2Val = COMPARE2_VALUE;
hwSetup.compVal.comp3Val = COMPARE3_VALUE;
hwSetup.updateCompVal.updateComp0Val = UPDATE_COMPARE0_VALUE;
hwSetup.updateCompVal.updateComp1Val = UPDATE_COMPARE1_VALUE;
hwSetup.updateCompVal.updateComp2Val = UPDATE_COMPARE2_VALUE;
hwSetup.updateCompVal.updateComp3Val = UPDATE_COMPARE3_VALUE;
hwSetup.intEnable.compIntr0En = FALSE;
hwSetup.intEnable.compIntr1En = FALSE;
hwSetup.intEnable.compIntr2En = FALSE;
hwSetup.intEnable.compIntr3En = FALSE;
hwSetup.intEnable.ovlInt0En = FALSE;
hwSetup.intEnable.ovlInt0En = FALSE;
hwSetup.dmaReq.dmareq0En = FALSE;
hwSetup.dmaReq.dmareq1En = FALSE;
hwSetup.dmaReq.dmareq2En = FALSE;
hwSetup.dmaReq.dmareq3En = FALSE;
hwSetup.preLoadWatchdog = 0x1FFF;
```

```
status = CSL_rtiHwsetup (hRti, &hwSetup);
...
```

10.2.5 CSL_rtiHwSetupRaw

CSL_Status CSL_rtiHwSetupRaw (CSL_RtiHandle ***hRti***,
CSL_RtiConfig * ***config***
)

Description
This function initializes the device registers with the register-values provided through the Config Data structure.

Arguments

```
hRti        Handle to the RTI instance

config      Pointer to config structure
```

Return Value
CSL_Status

- CSL_SOK - Configuration successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Configuration is not properly initialized

Pre Condition
Both CSL_rtiInit() and CSL_rtiOpen() must be called successfully in order before calling this function.

Post Condition
The registers of the specified RTI instance will be setup according to value passed.

Modifies
Hardware registers of the specified RTI instance

Example

```
CSL_RtiHandle       hRti;
CSL_RtiConfig       config = CSL_RTI_CONFIG_DEFAULTS;
CSL_Status          status;
...
status = CSL_rtiHwSetupRaw (hRti, &config);
...
```

10.2.6 CSL_rtiGetHwSetup

CSL_Status CSL_rtiGetHwSetup (CSL_RtiHandle ***hRti***,

CSL_RtiHwSetup * *hwSetup*

)

Description
It retrieves the hardware setup parameters of the real time interrupt module specified by the given handle.

Arguments

```
hRti          Handle to the real time interrupt

hwSetup       Pointer to the hardware setup structure
```

Return Value
CSL_Status

- CSL_SOK - Retrieving the hardware setup parameters is successful
- CSL_ESYS_BADHANDLE - The handle is passed is invalid

Pre Condition
Both CSL_rtiInit() and CSL_rtiOpen() must be called successfully in order before calling this function.

Post Condition
The hardware setup structure is populated with the hardware setup parameters.

Modifies
hwSetup variable

Example

```
CSL_RtiHandle   hRti;
CSL_RtiHwSetup  hwSetup;
...
status = CSL_rtiGetHwSetup(hRti, &hwSetup);
...
```

10.2.7 CSL_rtiHwControl

CSL_Status CSL_rtiHwControl (CSL_RtiHandle *hRti*,
CSL_RtiHwControlCmd *cmd*,
void * *cmdArg*

)

Description
This function performs various control operations on the real time interrupt, based on the command passed.

Arguments

```
hRti          Handle to the real time interrupt

cmd           Operation to be performed on the real time interrupt

cmdArg        Argument specific to the command
```

Return Value
CSL_Status

- CSL_SOK - Command execution successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command

Pre Condition
Both CSL_rtiInit() and CSL_rtiOpen() must be called successfully in order before calling this function.

Post Condition
Real time interrupt registers are configured according to the command and the command arguments. The command determines which registers are modified.

Modifies
Real time interrupt registers determined by the command

Example

```
CSL_RtiHandle   hRti;
Uint32 dwdCounterDis = 0x5312ACED;
...
status = CSL_rtiHwControl(hRti, CSL_RTI_CMD_DWD_ENABLE,
                          &dwdCounterDis);
...
```

10.2.8 CSL_rtiGetHwStatus

CSL_Status CSL_rtiGetHwStatus **(** CSL_RtiHandle ***hRti***,
CSL_RtiHwStatusQuery ***query***,
void * ***response***
)

Description
This function is used to get the value of various parameters of the real time interrupt. The value returned depends on the query passed.

Arguments

```
hRti            Handle to the real time interrupt

query           Query to be performed.
```

```
response        Pointer to buffer to return the data requested by
                the query passed
```

Return Value
CSL_Status

- CSL_SOK - Successful completion of the query
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Query command not supported

Pre Condition
Both CSL_rtiInit() and CSL_rtiOpen() must be called successfully in order before calling this function.

Post Condition
Data requested by the query is returned through the variable "response".

Modifies
The input argument "response" is modified.

Example

```
Uint32       count = 0;
CSL_Status   status;
...
status = CSL_rtiGetHwStatus(hRti,
                            CSL_RTI_QUERY_CUR_FRC0_CNT,
                            &count);
...
```

10.2.9 CSL_rtiGetBaseAddress

CSL_Status CSL_rtiGetBaseAddress (CSL_InstNum *rtiNum*,
CSL_RtiParam * *pRtiParam*,
CSL_RtiBaseAddress * *pBaseAddress*
)

Description
Function to get the base address of the peripheral instance. This function is used for getting the base address of the peripheral instance. This function will be called inside the CSL_rtiOpen() function call. This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral MMRs go to an alternate location.

Arguments

```
rtiNum          Specifies the instance of the RTI to be opened.

pRtiParam       Module specific parameters.
```

```
pBaseAddress    Pointer to baseaddress structure containing base
                address details.
```

Return Value
CSL_Status

- CSL_SOK - Open call is successful
- CSL_ESYS_FAIL - The instance number is invalid.

Pre Condition
None

Post Condition
Base Address structure is populated.

Modifies
1. The status variable
2. Base address structure is modified.

Example

```
CSL_Status          status;
CSL_RtiBaseAddress  baseAddress;

...
status = CSL_rtiGetBaseAddress(CSL_RTI, NULL, &baseAddress);
...
```

10.3 Data Structures

This section lists the data structures available in the RTI module.

10.3.1 CSL_RtiObj

Detailed Description
Real Time Interrupt object structure.

Field Documentation

CSL_InstNum CSL_RtiObj::perNum
Specifies a particular instance of real time interrupt

CSL_RtiRegsOvly CSL_RtiObj::regs
Pointer to the register overlay structure for the peripheral

10.3.2 CSL_RtiConfig

Detailed Description
Configuration structure. This is used to configure real time interrupt instance using CSL_rtiHwSetupRaw function.

Field Documentation

volatile Uint32 CSL_RtiConfig::RTICAPCTRL
RTI Capture Control Register

volatile Uint32 CSL_RtiConfig::RTICLEARINT
RTI Clear/Status Interrupt Register

volatile Uint32 CSL_RtiConfig::RTICOMP0
RTI Compare 0 Register

volatile Uint32 CSL_RtiConfig::RTICOMP1
RTI Compare 1 Register

volatile Uint32 CSL_RtiConfig::RTICOMP2
RTI Compare 2 Register

volatile Uint32 CSL_RtiConfig::RTICOMP3
RTI Compare 3 Register

volatile Uint32 CSL_RtiConfig::RTICOMPCTRL
RTI Compare Control Register

volatile Uint32 CSL_RtiConfig::RTICPUC0
RTI Compare Up Counter 0 Register

volatile Uint32 CSL_RtiConfig::RTICPUC1
RTI Compare Up Counter 1 Register

volatile Uint32 CSL_RtiConfig::RTIDWDCTRL

Digital Watchdog Control Register

volatile Uint32 CSL_RtiConfig::RTIDWDPRLD
Digital Watchdog Preload Register

volatile Uint32 CSL_RtiConfig::RTIFRC0
RTI Free Running Counter 0 Register

volatile Uint32 CSL_RtiConfig::RTIFRC1
RTI Free Running Counter 1 Register

volatile Uint32 CSL_RtiConfig::RTIGCTRL
RTI Global Control Register

volatile Uint32 CSL_RtiConfig::RTISETINT
RTI Set/Status Interrupt Register

volatile Uint32 CSL_RtiConfig::RTIUC0
RTI Up Counter 0 Register

volatile Uint32 CSL_RtiConfig::RTIUC1
RTI Up Counter 1 Register

volatile Uint32 CSL_RtiConfig::RTIUDCP0
RTI Update Compare 0 Register

volatile Uint32 CSL_RtiConfig::RTIUDCP1
RTI Update Compare 1 Register

volatile Uint32 CSL_RtiConfig::RTIUDCP2
RTI Update Compare 2 Register

volatile Uint32 CSL_RtiConfig::RTIUDCP3
RTI Update Compare 3 Register

volatile Uint32 CSL_RtiConfig::RTIWDKEY
Watchdog Key Register

10.3.3 CSL_RtiParam

Detailed Description
Module specific parameters. Present implementation doesn't have any module specific parameters.

Field Documentation

CSL_BitMask16 CSL_RtiParam::flags
Bit mask to be used for module specific parameters. The declaration is just a placeholder for future implementation.

10.3.4 CSL_RtiContext

Detailed Description

Module specific context information. Present implementation doesn't have any Context information.

Field Documentation

Uint16 CSL_RtiContext::contextInfo
Context information of Real Time Interrupt. The declaration is just a placeholder for future implementation.

10.3.5 CSL_RtiHwSetup

Detailed Description
Hardware setup structure.

Field Documentation

CSL_RtiExtnControl CSL_RtiHwSetup::blk0ExtnCntl
Select capture event source0/capture event source1 for block0

CSL_RtiExtnControl CSL_RtiHwSetup::blk1ExtnCntl
Select capture event source0/capture event source1 for block1

CSL_RtiCompareCntl CSL_RtiHwSetup::compare0Cntl
Select compare counter FRC0/FRC1

CSL_RtiCompareCntl CSL_RtiHwSetup::compare1Cntl
Select compare counter FRC0/FRC1

CSL_RtiCompareCntl CSL_RtiHwSetup::compare2Cntl
Select compare counter FRC0/FRC1

CSL_RtiCompareCntl CSL_RtiHwSetup::compare3Cntl
Select compare counter FRC0/FRC1

CSL_RtiCompUpCounter CSL_RtiHwSetup::compareUpCntrs
Set values for Compare up counters

CSL_RtiCompareVal CSL_RtiHwSetup::compVal
Set compare register values

CSL_RtiContOnSuspend CSL_RtiHwSetup::contOnSuspend
Stop/continue counters

CSL_RtiCounters CSL_RtiHwSetup::counters
Set value for counters

CSL_RtiDmaReq CSL_RtiHwSetup::dmaReqEn
Configuration to enable dma Request

CSL_RtiIntrConfig CSL_RtiHwSetup::intConfig
Configuration to enable the interrupt

Uint16 CSL_RtiHwSetup::preLoadWatchdog
Set preload value for digital watchdog

CSL_RtiUpdateCompVal CSL_RtiHwSetup::updateCompVal
Set update compare register values

10.3.6 CSL_RtiBaseAddress

Detailed Description
This structure contains the base-address information for the peripheral instance.

Field Documentation

CSL_RtiRegsOvly CSL_RtiBaseAddress::regs
Base-address of the Configuration registers of the peripheral

10.3.7 CSL_RtiCompUpCounter

Detailed Description
This structure is used to set compare up counter values.

Field Documentation

Uint32 CSL_RtiCompUpCounter::compareUpCntr0
Set Compare Up counter0 value

Uint32 CSL_RtiCompUpCounter::compareUpCntr1
Set Compare Up counter1 value

10.3.8 CSL_RtiCounters

Detailed Description
This structure is used to set Up counters and Free running counters values.

Field Documentation

Uint32 CSL_RtiCounters::frc0Counter
Set Free Running Counter0 value

Uint32 CSL_RtiCounters::frc1Counter
Set Free Running Counter1 value

Uint32 CSL_RtiCounters::uc0Counter
Set Up Counter0 value

Uint32 CSL_RtiCounters::uc1Counter
Set Up Counter1 value

10.3.9 CSL_RtiCompareVal

Detailed Description
This structure is used to set Compare register values.

Field Documentation

Uint32 CSL_RtiCompareVal::comp0Val
Set Compare0 value

Uint32 CSL_RtiCompareVal::comp1Val
Set Compare1 value

Uint32 CSL_RtiCompareVal::comp2Val
Set Compare2 value

Uint32 CSL_RtiCompareVal::comp3Val
Set Compare3 value

10.3.10 CSL_RtiUpdateCompVal

Detailed Description
This structure is used to set Update Compare register values.

Field Documentation

Uint32 CSL_RtiUpdateCompVal::updateComp0Val
Set Update Compare0 value

Uint32 CSL_RtiUpdateCompVal::updateComp1Val
Set Update Compare1 value

Uint32 CSL_RtiUpdateCompVal::updateComp2Val
Set Update Compare2 value

Uint32 CSL_RtiUpdateCompVal::updateComp3Val
Set Update Compare3 value

10.3.11 CSL_RtiDmaReq

Detailed Description
This structure is used enable/disable the DMA requests.

Field Documentation

Bool CSL_RtiDmaReq::dmaReq0En
Enable/Disable DMA request0

Bool CSL_RtiDmaReq::dmaReq1En
Enable/Disable DMA request1

Bool CSL_RtiDmaReq::dmaReq2En
Enable/Disable DMA request2

Bool CSL_RtiDmaReq::dmaReq3En
Enable/Disable DMA request3

10.3.12 CSL_RtiIntrConfig

Detailed Description
This structure is used enable/disable the Interrupts.

Field Documentation

Bool CSL_RtiIntrConfig::compIntr0En
Enable/Disable Interrupt0

Bool CSL_RtiIntrConfig::compIntr1En
Enable/Disable Interrupt1

Bool CSL_RtiIntrConfig::compIntr2En
Enable/Disable Interrupt2

Bool CSL_RtiIntrConfig::compIntr3En
Enable/Disable Interrupt3

Bool CSL_RtiIntrConfig::ovlIntr0En
Enable/Disable Overflow Interrupt0

Bool CSL_RtiIntrConfig::ovlIntr1En
Enable/Disable Overflow Interrupt1

10.4 Enumerations

This section lists the enumerations available in the RTI module.

10.4.1 CSL_RtiContOnSuspend

enum CSL_RtiContOnSuspend
Enumeration for Start/Continue Counters when device is in debug mode.

Enumeration values:

CSL_RTI_COUNTERS_STOP	Stop the Counters in debug mode
CSL_RTI_COUNTERS_RUN	Continue the counters in debug mode

10.4.2 CSL_RtiExtnControl

enum CSL_RtiExtnControl
Enumeration for Configure external interrupt source for both UC0 and FRC0 and configure external interrupt source for both UC1 and FRC1.

Enumeration values:

CSL_RTI_CAPTURE_EVENT_SOURCE0	Enable capture event triggered by Capture Event Source 0
CSL_RTI_CAPTURE_EVENT_SOURCE1	Enable capture event triggered by Capture Event Source 1

10.4.3 CSL_RtiCompareCntl

enum CSL_RtiCompareCntl
Enumeration for free running counters with which compare registers value is compared.

Enumeration values:

CSL_RTI_FRC0_COMPARE_ENABLE	Enable compare with FRC0
CSL_RTI_FRC1_COMPARE_ENABLE	Enable compare with FRC1

10.4.4 CSL_RtiHwControlCmd

enum CSL_RtiHwControlCmd
Enumeration for hardware control commands.

Enumeration values:

CSL_RTI_CMD_START_BLOCK0	Start the Block0 Counters **Parameters:** *(None)*
CSL_RTI_CMD_STOP_BLOCK0	Stop the Block0 Counters. **Parameters:** *(None)*
CSL_RTI_CMD_START_BLOCK1	Start the Block1 Counters. **Parameters:** *(None)*

CSL_RTI_CMD_STOP_BLOCK1	Stop the Block1 Counters. **Parameters:** *(None)*
CSL_RTI_CMD_CLEAR_INT0	Clear the Compare Interrupt0. **Parameters:** *(None)*
CSL_RTI_CMD_CLEAR_INT1	Clear the Compare Interrupt1. **Parameters:** *(None)*
CSL_RTI_CMD_CLEAR_INT2	Clear the Compare Interrupt2. **Parameters:** *(None)*
CSL_RTI_CMD_CLEAR_INT3	Clear the Compare Interrupt3. **Parameters:** *(None)*
CSL_RTI_CMD_CLEAR_OVFINT0	Clear the Overflow Interrupt0. **Parameters:** *(None)*
CSL_RTI_CMD_CLEAR_OVFINT1	Clear the Overflow Interrupt1. **Parameters:** *(None)*
CSL_RTI_CMD_DWD_ENABLE	Enable Digital Watchdog. **Parameters:** *(None)*
CSL_RTI_CMD_DWD_DISABLE	Disable Digital Watchdog. **Parameters:** *(None)*
CSL_RTI_CMD_CLEAR_DWD	Clear Digital Watchdog status. **Parameters:** *(None)*
CSL_RTI_CMD_WDKEY	Set the Digital Watchdog Key **Parameters:** *(None)*

10.4.5 CSL_RtiHwStatusQuery

enum CSL_RtiHwStatusQuery
Enumeration for hardware status query commands.

Enumeration values:

CSL_RTI_QUERY_CUR_FRC0_CNT	Query the current value of free running counter0 Query the current value of up counter0. **Parameters:** *(Uint32 *)*
CSL_RTI_QUERY_CUR_UC0_CNT	Query the current value of free running counter1. **Parameters:** *(Uint32 *)*
CSL_RTI_QUERY_CUR_FRC1_CNT	Query the current value of up counter1.

	Parameters: *(Uint32 *)*
CSL_RTI_QUERY_CUR_UC1_CNT	Query the captured value of up counter0. **Parameters:** *(Uint32 *)*
CSL_RTI_QUERY_UC0_CAPTURE_VAL	Query the captured value of free running counter0. **Parameters:** *(Uint32 *)*
CSL_RTI_QUERY_FRC0_CAPTURE_VAL	Query the captured value of up counter1. **Parameters:** *(Uint32 *)*
CSL_RTI_QUERY_UC1_CAPTURE_VAL	Query the captured value of free running counter1. **Parameters:** *(Uint32 *)*
CSL_RTI_QUERY_FRC1_CAPTURE_VAL	Query the value of compare0 register. **Parameters:** *(Uint32 *)*
CSL_RTI_QUERY_COMP0_CNT	Query the value of compare1 register. **Parameters:** *(Uint32 *)*
CSL_RTI_QUERY_COMP1_CNT	Query the value of compare2 register. **Parameters:** *(Uint32 *)*
CSL_RTI_QUERY_COMP2_CNT	Query the value of compare3 register. **Parameters:** *(Uint32 *)*
CSL_RTI_QUERY_COMP3_CNT	Query the Status of overflow interrupt0. **Parameters:** *(Bool *)*
CSL_RTI_QUERY_OVLINT0_STATUS	Query the Status of overflow interrupt1. **Parameters:** *(Bool *)*
CSL_RTI_QUERY_OVLINT1_STATUS	Query the Status of compare interrupt0. **Parameters:** *(Bool *)*
CSL_RTI_QUERY_INT0_STATUS	Query the Status of compare interrupt1. **Parameters:** *(Bool *)*
CSL_RTI_QUERY_INT1_STATUS	Query the Status of compare interrupt2. **Parameters:** *(Bool *)*
CSL_RTI_QUERY_INT2_STATUS	Query the Status of compare interrupt3. **Parameters:** *(Bool *)*
CSL_RTI_QUERY_INT3_STATUS	Query the Status of digital watchdog. **Parameters:**

	*(Bool *)*
CSL_RTI_QUERY_WATCHDOG_STATUS	Query the Status of digital watchdog **Parameters:** *(Bool *)*
CSL_RTI_QUERY_WATCHDOG_DWNCTR	Query the digital watchdog Counter value **Parameters:** *(Uint32 *)*

10.5 Macros

#define CSL_RTI_COMP_UPCOUNTERS_SETUP_DEFAULTS
Value:

```
{ \
        0,    \
        0     \
}
```

Default compare upcounters setup parameters

#define CSL_RTI_COMPARE_SETUP_DEFAULTS
Value:

```
{ \
        0,    \
        0,    \
        0,    \
        0     \
}
```

Default compare register setup parameters

#define CSL_RTI_CONFIG_DEFAULTS
Value:

```
{\
    CSL_RTI_RTIGCTRL_RESETVAL,\
    CSL_RTI_RTICAPCTRL_RESETVAL,\
    CSL_RTI_RTICOMPCTRL_RESETVAL,\
    CSL_RTI_RTIFRC0_RESETVAL,\
    CSL_RTI_RTIUC0_RESETVAL,\
    CSL_RTI_RTICPUC0_RESETVAL,\
    CSL_RTI_RTIFRC1_RESETVAL,\
    CSL_RTI_RTIUC1_RESETVAL,\
    CSL_RTI_RTICPUC1_RESETVAL,\
    CSL_RTI_RTICOMP0_RESETVAL,\
    CSL_RTI_RTIUDCP0_RESETVAL,\
    CSL_RTI_RTICOMP1_RESETVAL,\
    CSL_RTI_RTIUDCP1_RESETVAL,\
    CSL_RTI_RTICOMP2_RESETVAL,\
    CSL_RTI_RTIUDCP2_RESETVAL,\
    CSL_RTI_RTICOMP3_RESETVAL,\
    CSL_RTI_RTIUDCP3_RESETVAL,\
    CSL_RTI_RTISETINT_RESETVAL,\
    CSL_RTI_RTICLEARINT_RESETVAL,\
    CSL_RTI_RTIDWDCTRL_RESETVAL,\
    CSL_RTI_RTIDWDPRLD_RESETVAL,\
    CSL_RTI_RTIWDKEY_RESETVAL\
}
```

Default Values for Config structure.

#define CSL_RTI_COUNTERS_SETUP_DEFAULTS
Value:

```
{ \
        0,  \
```

```
        0,  \
        0,   \
        0    \
}
```

Default counters setup parameters

#define CSL_RTI_DMA_SETUP_DEFAULTS
Value:

```
{ \
        FALSE,   \
        FALSE,   \
        FALSE,   \
        FALSE    \
}
```

Default dma request setup parameters

#define CSL_RTI_HWSETUP_DEFAULTS
Value:

```
{\
        CSL_RTI_COUNTERS_STOP,  \
        CSL_RTI_CAPTURE_EVENT_SOURCE0,  \
        CSL_RTI_CAPTURE_EVENT_SOURCE0,  \
        CSL_RTI_FRC0_COMPARE_ENABLE,    \
        CSL_RTI_FRC0_COMPARE_ENABLE,    \
        CSL_RTI_FRC0_COMPARE_ENABLE,    \
        CSL_RTI_FRC0_COMPARE_ENABLE,    \
        CSL_RTI_COUNTERS_SETUP_DEFAULTS, \
        CSL_RTI_COMP_UPCOUNTERS_SETUP_DEFAULTS, \
        CSL_RTI_COMPARE_SETUP_DEFAULTS, \
        CSL_RTI_UPDATE_COMPARE_SETUP_DEFAULTS,  \
        CSL_RTI_INT_SETUP_DEFAULTS,     \
        CSL_RTI_DMA_SETUP_DEFAULTS,     \
        0x1FFF  \
}
```

Default hardware setup parameters.

#define CSL_RTI_INT_SETUP_DEFAULTS
Value:

```
{ \
        FALSE,   \
        FALSE,   \
        FALSE,   \
        FALSE,   \
        FALSE,   \
        FALSE    \
}
```

Default interrupt config setup parameters.

#define CSL_RTI_UPDATE_COMPARE_SETUP_DEFAULTS
Value:

```
{ \
        0,   \
        0,   \
        0,   \
        0    \
```

```
}
```

Default update compare setup parameters.

Chapter 11
SPI Module

Topics

11.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within SPI module.

SPI is a high speed serial input and output port that allows a serial bit stream of programmed length. It is used by a micro controller to communicate with peripheral devices or other controller.

The SPI / MibSPI has the following attributes:

- SPI Kernel Only
- Three, Four and Five Pin SPI Options
- Master and Slave Options
- Interrupt and DMA requests
- GIO Control of I/O pins

11.2 Functions

This section lists the functions available in the SPI module.

11.2.1 CSL_spiInit

CSL_Status CSL_spiInit (CSL_SpiContext * *pContext*)

Description
This is the initialization function for the SPI. This function is idempotent in that calling it many times is same as calling it once. This function does not modify any registers or check status. It returns status CSL_SOK. It has been kept for future use.

Arguments

```
pContext   Context information for the instance. Should be NULL
```

Return Value
CSL_Status

- CSL_SOK - Always returns

Pre Condition
CSL_sysInit() must be called.

Post Condition
None

Modifies
None

Example

```
CSL_Status       status;
...
status = CSL_spiInit(NULL)
...
```

11.2.2 CSL_spiOpen

CSL_SpiHandle CSL_spiOpen
(**CSL_SpiObj *** ***pSpiObj*****,**
CSL_InstNum ***spiNum*****,**
CSL_SpiParam * ***pSpiParam*****,**
CSL_Status * ***pStatus***
)

Description
Opens, if possible, the instance of SPI requested

This function populates the peripheral data object for the instance and returns a handle to the instance. The open call sets up the data structures for the particular instance of SPI device. Reserves the specified spi for use. The device can be re-opened anytime after it has been normally closed if so required. The handle returned by this call is input as an essential argument for rest of the APIs described for this module. The CSL system as well as SPI must be successfully initialized via CSL_sysInit() and CSL_spiInit() before calling this function. Memory for the CSL_SpiObj must be allocated outside this call.

Arguments

```
pSpiObj     Pointer to the SPI instance object

spiNum      Instance of the SPI to be opened.

pSpiParam   Pointer to module specific parameters

pStatus     Pointer for returning status of the function call
```

Returns
CSL_SpiHandle
Valid SPI instance handle will be returned if status value is equal to CSL_SOK.

Pre Condition
The SPI must be successfully initialized via CSL_spi*Init*() before calling this function

Post Condition
1. The status is returned in the status variable. If status returned is

- `CSL_SOK` - Valid SPI handle is returned
- `CSL_ESYS_FAIL` - The SPI instance is invalid
- `CSL_ESYS_INVPARAMS` - Invalid parameter

2. SPI object structure is populated.

Modifies
1. The status variable
2. SPI object structure

Example:

```
CSL_SpiObj     spiObj;
CSL_SpiHwSetup spiSetup;
CSL_Status     status;
...
hSpi = CSL_spiOpen(&spiobj,
                   CSL_SPI_0,
                   NULL,
                   &status);
...
```

11.2.3 CSL_spiClose

CSL_Status CSL_spiClose (CSL_SpiHandle *hSpi*)

Description
This function closes the specified instance of SPI.

Arguments

```
hSpi            Handle to the SPI instance
```

Return Value
CSL_Status

- CSL_SOK - Close successful
- CSL_ESYS_BADHANDLE - Invalid handle

Pre Condition
Both CSL_spiInit() and CSL_spiOpen() must be called successfully in order before calling CSL_spiClose().

Post Condition
The SPI CSL APIs can not be called until the SPI CSL is reopened again using CSL_spiOpen().

Modifies
SPI Handle.

Example

```
CSL_SpiHandle   hSpi;
CSL_Status      status;
...
status = CSL_spiClose(hSpi);
...
```

11.2.4 CSL_spiHwSetup

CSL_Status CSL_spiHwSetup (CSL_SpiHandle *hSpi*, CSL_SpiHwSetup * *hwSetup*)

Description
It configures the SPI registers as per the values passed in the hardware setup structure. Programs the SPI to a useful state as specified

Arguments

```
hSpi            Handle to the Spi instance
```

```
hwSetup         Pointer to hardware setup structure
```

Return Value
CSL_Status

- CSL_SOK - Hardware setup successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Hardware structure is not properly initialized

Pre Condition
Both CSL_spiInit() and CSL_spiOpen() must be called successfully in order before calling this function.

Post Condition
SPI controller registers are configured according to the hardware setup parameters.

Modifies
SPI controller registers

Example

```
CSL_SpiHandle   hSpi;
CSL_SpiHwSetup  hwSetup;
CSL_Status      status;
...
status = CSL_spiHwSetup(hSpi, &hwSetup);
...
```

11.2.5 CSL_spiGetHwSetup

CSL_Status CSL_spiGetHwSetup (CSL_SpiHandle *hSpi*, CSL_SpiHwSetup * *hwSetup*)

Description
It retrieves the hardware setup parameters of the SPI instance specified by the given handle.

Arguments

```
hSpi        Handle to the SPI instance

hwSetup     Pointer to the hardware setup structure
```

Return Value
CSL_Status

- CSL_SOK - Retrieving the hardware setup parameters is successful
- CSL_ESYS_BADHANDLE - The handle is passed is invalid
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition
Both CSL_spiInit() and CSL_spiOpen() must be called successfully in order before calling this function.

Post Condition
Hardware setup parameters are returned in the hwSetup variable

Modifies
hwSetup variables

Example

```
CSL_SpiHandle   hSpi;
CSL_SpiHwSetup  hwSetup;
CSL_Status       status;
...
status = CSL_SpiGetHwSetup(hSpi, &hwSetup);
...
```

11.2.6 CSL_spiHwControl

CSL_Status CSL_spiHwControl (CSL_SpiHandle ***hSpi***,
CSL_SpiHwControlCmd ***cmd***,
void * ***arg***
)

Description
Controls the different operations that can be performed by SPI Takes a command of SPI with an optional argument and implements it.

Arguments

```
hSpi        Handle to the SPI instance

cmd         The command to this API indicates the action to be
            taken on SPI.

arg         An optional argument.
```

Return Value
CSL_Status

- CSL_SOK - Status info return successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command
- CSL_ESYS_INVPARAMS - Invalid parameter
- CSL_ESYS_NOTSUPPORTED - Action Not Supported

Pre Condition

Both CSL_spiInit() and CSL_spiOpen() must be called successfully in order before calling this function.

Post Condition
SPI registers are configured according to the command passed.

Modifies
The hardware registers of SPI.

Example

```
CSL_SpiHandle         hSpi;
CSL_SpiHwControlCmd   cmd=CSL_SPI_CMD_CPT_DMA_ENABLE;
void                  arg;
CSL_Status       status;
...
status = CSL_spiHwControl (hSpi, cmd, &arg);
...
```

11.2.7 CSL_spiHwSetupRaw

CSL_Status CSL_spiHwSetupRaw (CSL_SpiHandle *hSpi*, CSL_SpiConfig * *config*)

Description
This function initializes the device registers with the register-values provided through the Config Data structure. This configures registers based on a structure of register values, as compared to HwSetup, which configures registers based on structure of bit field values and may perform other functions (delays, etc.)

Arguments

```
hSpi            Handle to the SPI

config          Pointer to config structure
```

Return Value
CSL_Status

- CSL_SOK - Configuration successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Configuration is not properly initialized

Pre Condition
Both CSL_spiInit() and CSL_spiOpen() must be called successfully in order before calling this function.

Post Condition
The registers of the specified SPI instance will be setup according to value passed.

Modifies
Hardware registers of the specified SPI instance.

Example

```
CSL_SpiHandle       hSpi;
CSL_SpiConfig       config = CSL_SPI_CONFIG_DEFAULTS;
CSL_Status          status;
...
status = CSL_spiHwSetupRaw (hSpi, &config);
...
```

11.2.8 CSL_spiGetHwStatus

CSL_Status CSL_spiGetHwStatus **(** CSL_SpiHandle ***hSpi***,
CSL_SpiHwStatusQuery ***query***,
void * ***response***
)

Description
Gets the status of the different settings of SPI instance.
Returns the status of different operations or the current setup of SPI.

Arguments

```
hSpi      Handle to the SPI instance

query     The query to this API of SPI that indicates the
          status to be returned.

response  Placeholder to return the status.
```

Return Value
CSL_Status

- CSL_SOK - Status info return successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Invalid query command
- CSL_ESYS_INVPARAMS - Invalid parameter
- CSL_ESYS_NOTSUPPORTED- Action Not Supported

Pre Condition
Both CSL_spiInit() and CSL_spiOpen() must be called successfully in order before calling this function.

Post Condition
None

Modifies
Third parameter response value

Example

```
CSL_SpiHandle          hSpi;
CSL_SpiHwStatusQuery   query=CSL_SPI_QUERY_INT_VECTOR0;
CSL_SpiIntVec          response;
...
status = CSL_spiGetHwStatus (hSpi, query, &response);
...
```

11.2.9 CSL_spiGetBaseAddress

CSL_Status CSL_spiGetBaseAddress (CSL_InstNum *spiNum*,
CSL_SpiParam * *pSpiParam*,
CSL_SpiBaseAddress * *pBaseAddress*
)

Description
This function is used for getting the base address of the peripheral instance. This function is called inside the CSL_spiOpen() function call. This function is open for re-implementing. if the user wants to modify the base address of the peripheral object to point to a different location and thereby allow CSL initiated write/reads into peripheral MMRs go to an alternate location.

Arguments

```
spiNum        Specifies the instance of the SPI to be opened.

pSpiParam     Module specific parameters.

pBaseAddress  Pointer to baseaddress structure containing base
              address details.
```

Return Value
CSL_Status

- CSL_SOK Open call is successful
- CSL_ESYS_FAIL The instance number is invalid.

Pre Condition
None

Post Condition
Base Address structure is populated.

Modifies
1. The status variable
2. Base address structure is modified.

Example

```
CSL_Status          status;
CSL_SpiBaseAddress  baseAddress;
...
status = CSL_spiGetBaseAddress(CSL_SPI_0, NULL,
                                &baseAddress);
...
```

11.3 Data Structures

This section lists the data structures available in the SPI module.

11.3.1 CSL_SpiObj

Detailed Description
This object contains the reference to the instance of SPI opened using the *CSL_spiOpen()*
An object related to this structure is passed to all SPI CSL APIs as the first argument

Field Documentation

CSL_InstNum CSL_SpiObj::perNum
This is the instance of SPI being referred to by this object

CSL_SpiRegsOvly CSL_SpiObj::regs
This is a pointer to the registers of the instance of SPI referred to by this object

11.3.2 CSL_SpiParam

Detailed Description
SPI specific parameters. Present implementation doesn't have any specific parameters.

Field Documentation

CSL_BitMask16 CSL_SpiParam::flags
Bit mask to be used for module specific parameters. The declaration is just a placeholder for future implementation.

11.3.3 CSL_SpiContext

Detailed Description
SPI specific context information. Present implementation doesn't have any Context information.

Field Documentation

Uint16 CSL_SpiContext::contextInfo
Context information of SPI. The declaration is just a placeholder for future implementation.

11.3.4 CSL_SpiConfig

Detailed Description
This structure is used configure the whole of SPI.

Field Documentation

Uint32 CSL_SpiConfig::SPIGCR0
Holds the value for Global Control Register0

Uint32 CSL_SpiConfig::SPIGCR1
Holds the value for Global Control Register1

Uint32 CSL_SpiConfig::SPIINT0
Holds the value for Interrupt Register

Uint32 CSL_SpiConfig::SPILVL
Holds the value to map interrupts to different interrupt lines

Uint32 CSL_SpiConfig::SPIFLG
Holds the value Flag Status Register

Uint32 CSL_SpiConfig::SPIPC0
Holds the value for Pin Control Register

Uint32 CSL_SpiConfig::SPIPC1
Holds the value for Pin Control Register 1

Uint32 CSL_SpiConfig::SPIPC2
Holds the value for Pin Control Register 2

Uint32 CSL_SpiConfig::SPIPC3
Holds the value for Pin Control Register 3

Uint32 CSL_SpiConfig::SPIPC4
Holds the value for Pin Control Register 4

Uint32 CSL_SpiConfig::SPIPC5
Holds the value for Pin Control Register 5

Uint32 CSL_SpiConfig::SPIDAT0
Holds the value for SPI Data Register0

Uint32 CSL_SpiConfig::SPIDAT1
Holds the value for SPI Data Register1

Uint32 CSL_SpiConfig::SPIBUF
Holds the value for Buffer Register

Uint32 CSL_SpiConfig::SPIDELAY
Holds the value for delay Register

Uint32 CSL_SpiConfig::SPIDEF
Holds the value for default chip select Register

Uint32 CSL_SpiConfig::SPIFMT
Holds the value for Data Format Register

Uint32 CSL_SpiConfig::TGINTVEC
SPI Interrupt Vector Register Transfer Group Interrupt Vector Register

11.3.5 CSL_SpiHwSetup

Detailed Description
Main structure that is used to setup the SPI

This structure is used to setup or obtain the existing setup of SPI using *CSL_spiHwSetup()* and *CSL_spiGetHwStatus()* functions respectively. If a particular member pointer is null, then these functions do not setup or get the setup of the corresponding part of SPI.

Field Documentation

CSL_SpiHwSetupCpt* CSL_SpiHwSetup::cptPtr
This pointer to *CSL_SpiHwSetupSdio* is used to hold information on the SDIO part of the SPI setup

CSL_SpiHwSetupGen* CSL_SpiHwSetup::genPtr
This pointer to *CSL_SpiHwSetupCommon* is used to hold information on the common part of the SPI setup

CSL_SpiHwSetupPins* CSL_SpiHwSetup::pinsPtr
This pointer to *CSL_SpiHwSetupSdio* is used to hold information on the SPI part of the SPI setup

CSL_SpiHwSetupPri* CSL_SpiHwSetup::priPtr
This pointer to *CSL_SpiHwSetupFifo* is used to hold information on the data FIFO part of the SPI setup

11.3.6 CSL_SpiBaseAddress

Detailed Description
This will have the base-address information for the peripheral instance.

Field Documentation

CSL_SpiRegsOvly CSL_SpiBaseAddress::regs
Base-address of the Configuration registers of SPI

11.3.7 CSL_SpiHwSetupPins

Detailed Description
Sets up the properties if the pins of Spi
This object is used to setup or get the setup of the pins in Spi

Field Documentation

Uint16 CSL_SpiHwSetupPins::dir
If GPIO, decides the directions of the pins

Uint16 CSL_SpiHwSetupPins::func
Decides if the pins will be Spi or GPIO

11.3.8 CSL_SpiHwSetupFmtCtrl

Detailed Description
Sets up the format selection for a set of transfers
This object is used to setup or get the setup of the format selection for a set of transfers

Field Documentation

Uint8 CSL_SpiHwSetupFmtCtrl::cSel
Defines the chip select that will be activated for the transfer

CSL_SpiCsHold CSL_SpiHwSetupFmtCtrl::csHold
Decides if chip select is to be held active between transfers

CSL_SpiFmtSel CSL_SpiHwSetupFmtCtrl::fmtSel
Decides which format to select

CSL_SpiWDelayEn CSL_SpiHwSetupFmtCtrl::wDel
Decides if delay specified in the selected format must be allowed between 2 consecutive transfers

11.3.9 CSL_SpiHwSetupCpt

Detailed Description
Sets up the Spi for compatibility mode
This structure is used to setup or get the setup of Spi in comaptibility mode

Field Documentation

CSL_SpiHwSetupFmtCtrl* CSL_SpiHwSetupCpt::fmtCtrlPtr
Selects the format & associated controls

Uint32* CSL_SpiHwSetupCpt::lvl
Selects if interrupts should go to lines INT0 or INT1

11.3.10 CSL_SpiHwSetupPriFmt

Detailed Description
Sets up the a formatting for an outgoing word
This object is used to set up or get the setup of the format registers in Spi

Field Documentation

Uint8 CSL_SpiHwSetupPriFmt::charLen
The length of the word to be transmitted and/or received

CSL_SpiParity CSL_SpiHwSetupPriFmt::parity
Whether parity should be enabled; if enabled then even or odd

CSL_SpiClkPhase CSL_SpiHwSetupPriFmt::phase
Whether data should be in phase of 1/2 cycle ahead of the clock

CSL_SpiClkPolarity CSL_SpiHwSetupPriFmt::polarity
Whether clock should be high or low when inactive

Uint8 CSL_SpiHwSetupPriFmt::preScale
The factor with which to multiply functional clock in order to get the serial clock

CSL_SpiShDir CSL_SpiHwSetupPriFmt::shiftDir
Whether LSB or MSB should be shifted first

CSL_SpiWaitEn **CSL_SpiHwSetupPriFmt::waitEna**
If in master mode; whether Spi should wait for ENA from slave

Uint8 CSL_SpiHwSetupPriFmt::wDelay
Delay between two consecutive words

11.3.11 CSL_SpiHwSetupPri

Detailed Description
Sets up the parameters to be setup in priority mode
This object is used to setup or get the setup of the parameters to be setup in priority mode

Field Documentation

CSL_SpiHwSetupPriFmt* **CSL_SpiHwSetupPri::fmt[4]**
Array of pointers to structures of formats of an outgoing word

11.3.12 CSL_SpiHwSetupGen

Detailed Description
Sets up the parameters that are needed by multi-buffer as well as compatibility modes
This objetc is used to set up or get the setup of parameters that are needed by multi-buffer as well as compatibility modes

Field Documentation

Uint8 CSL_SpiHwSetupGen::c2eTmout
Chip-select-active-to-ENA-signal-active-time-out

Uint8 CSL_SpiHwSetupGen::c2tDelay
Chip-select-active-to-transmit-start-delay

Uint8 CSL_SpiHwSetupGen::csDefault
The default value on Chip select when inactive

CSL_SpiEnaHiZ **CSL_SpiHwSetupGen::enaHiZ**
Whether ENA signal should be tristated when inactive or if it should bear a value

CSL_SpiOpMod **CSL_SpiHwSetupGen::opMode**
Master or slave mode

Uint8 CSL_SpiHwSetupGen::t2cDelay
Transmit-end-to-chip-select-inactive-delay

Uint8 CSL_SpiHwSetupGen::t2eTmout
Transmit-data-finished-to-ENA-pin-inactive-time-out

11.3.13 CSL_SpiIntVec

Detailed Description
Gets the information for interrupt vectors

Field Documentation

Uint8 CSL_SpiIntVec::intVal
Interrupt vector number

11.3.14 CSL_SpiBufStat

Detailed Description
Gets the status of buffer after a transfer

Field Documentation

Uint8 CSL_SpiBufStat::cSel
Status of the chip select during last transfer

11.3.15 CSL_SpiCptData

Detailed Description
This structure is used to get data and its status

Field Documentation

Uint8 CSL_SpiCptData::cSel
Place to hold the data

Uint8 CSL_SpiCptData::status
Place to hold the status

11.4 Enumerations

This section lists the enumerations available in the SPI module.

11.4.1 CSL_SpiHwControlCmd

enum CSL_SpiHwControlCmd
Enumeration for control commands passed to *CSL_spiHwControl()*.
This is the set of commands that are passed to the *CSL_spiHwControl()* with an optional argument type-casted to *void** . The arguments to be passed with each enumeration, if any, are specified next to the enumeration.

Enumeration values:

CSL_SPI_CMD_PRI_RESET	Reset the SPI **Parameters:** *(None)*
CSL_SPI_CMD_CPT_DMA_ENABLE	Enable DMA transaction capability for DMA in compatibility mode **Parameters:** *(None)*
CSL_SPI_CMD_CPT_DMA_DISABLE	Enable DMA transaction capability for DMA in compatibility mode **Parameters:** *(None)*
CSL_SPI_CMD_CPT_WRITE0	Write data in argument to SPIDAT0 register for transmitting out **Parameters:** *Uint16**
CSL_SPI_CMD_CPT_WRITE1	Write data in argument to SPIDAT1 register for transmitting out **Parameters:** *Uint16**
CSL_SPI_CMD_CPT_READ	Read the data in SPIBUF register to the argument **Parameters:** *CSL_SpiCptData**
CSL_SPI_CMD_INT_ENABLE	Enable the interrupts in the bit-vector argument **Parameters:** *Uint16**
CSL_SPI_CMD_INT_DISABLE	Disable the interrupts in the bit-vector argument **Parameters:** *Uint16**
CSL_SPI_CMD_PINS_VALUE	Set value passed in argument over pins configured as GPIO **Parameters:** *Uint16**
CSL_SPI_CMD_PINS_SET	Set pins passed in bit-vector argument if configured as GPIO **Parameters:** *Uint16**

CSL_SPI_CMD_PINS_CLEAR	Clear pins passed in bit-vector argument if configured as GPIO **Parameters:** *Uint16**
CSL_SPI_CMD_XFER_ENABLE	Enable the data-transfer section of the SPI **Parameters:** *(None)*
CSL_SPI_CMD_XFER_DISABLE	Disable the data-transfer section of the SPI **Parameters:** *(None)*

11.4.2 CSL_SpiHwStatusQuery

enum CSL_SpiHwStatusQuery
Enumeration for queries passed to *CSL_spiGetHwStatus()*.
This is used to get the status of different operations or to get the existing setup of SPI. The arguments to be passed with each enumeration, if any, are specified next to the enumeration.

Enumeration values:

CSL_SPI_QUERY_PINS_VALUE	Get the value present on the pins as a bit-vector **Parameters:** *Uint16**
CSL_SPI_QUERY_INT_VECTOR0	Get value of highest priority int that has occurred on INT0 line **Parameters:** *CSL_SpiMbfInt**
CSL_SPI_QUERY_INT_VECTOR1	Get value of highest priority int that has occurred on INT1 line **Parameters:** *CSL_SpiMbfInt**
CSL_SPI_QUERY_EVT_STATUS	Get value of Flag status register **Parameters:** Uint8*
CSL_SPI_QUERY_INT_ENABLED	Get the bit-vector of interrupts that have been enabled **Parameters:** *Uint16**
CSL_SPI_QUERY_CPT_DMA_ENABLED	Get the status of whether DMA is enabled in compatibility mode **Parameters:** *CSL_SpiCptDma**

11.4.3 CSL_SpiInt

enum CSL_SpiInt
Enumeration for Spi general interrupts.

Enumeration values:

CSL_SPI_INT_RX	Interrupt on successful receive

CSL_SPI_INT_OVRN	Interrupt on receiver overrun
CSL_SPI_INT_BITERR	Interrupt on bit error
CSL_SPI_INT_DESYNC	Interrupt on loss of synchronization between master & slave
CSL_SPI_INT_TIMEOUT	Interrupt on timeout error

11.4.4 CSL_SpiBufStatus

enum CSL_SpiBufStatus
Enumeration for Spi status bits.

Enumeration values:

CSL_SPI_BUFSTATUS_RXINT	Indicates that a word has been received in SPIBUF
CSL_SPI_BUFSTATUS_RXEPTY	Indicates that receive BUF is empty
CSL_SPI_BUFSTATUS_RXOVRN	Indicates that overrun occurred/not
CSL_SPI_BUFSTATUS_TXFULL	Indicates that transmit BUF is full
CSL_SPI_BUFSTATUS_BITERR	Indicates that a bit error occurred during transaction
CSL_SPI_BUFSTATUS_DESYNC	Indicates that desynchronization with slave detected
CSL_SPI_BUFSTATUS_TMOUT	Indicates timeout

11.4.5 CSL_SpiCsHold

enum CSL_SpiCsHold
Enumeration to hold the chip select active between 2 transfers.

Enumeration values:

CSL_SPI_CSHOLD_YES	Hold chip select active between consecutive transfers
CSL_SPI_CSHOLD_NO	Chip select to be inactivated after the each transfer

11.4.6 CSL_SpiWDelayEn

enum CSL_SpiWDelayEn
Enumeration to control format delay between two consecutive transfers.

Enumeration values:

CSL_SPI_WDELAYEN_YES	Enable format delay between 2 consecutive transfers
CSL_SPI_WDELAYEN_NO	Disable format delay between 2 consecutive transfers

11.4.7 CSL_SpiFmtSel

enum CSL_SpiFmtSel
Enumeration to select the required data transfer format.

Enumeration values:

CSL_SPI_FMTSEL_0	Select format 0
CSL_SPI_FMTSEL_1	Select format 1
CSL_SPI_FMTSEL_2	Select format 2
CSL_SPI_FMTSEL_3	Select format 3

11.4.8 CSL_SpiWaitEn

enum CSL_SpiWaitEn
Enumeration to control the dependence of transfer in ENA signal in master mode.

Enumeration values:

CSL_SPI_WAITEN_YES	If in master mode; wait for ENA signal from slave
CSL_SPI_WAITEN_NO	Do not wait for ENA signal from slave

11.4.9 CSL_SpiParity

enum CSL_SpiParity
Enumeration to control the parity setting in the data format.

Enumeration values:

CSL_SPI_PARITY_EVEN	Enable even parity
CSL_SPI_PARITY_ODD	Enable odd parity
CSL_SPI_PARITY_DISABLE	Disable parity

11.4.10 CSL_SpiClkPolarity

enum CSL_SpiClkPolarity
Enumeration to control the polarity of serial clock.

Enumeration values:

CSL_SPI_POLARITY_INACTIVELO	Clock is low when inactive
CSL_SPI_POLARITY_INACTIVEHI	Clock is high when inactive

11.4.11 CSL_SpiClkPhase

enum CSL_SpiClkPhase
Enumeration to control phase relationship between data and clock.

Enumeration values:

CSL_SPI_PHASE_IN	Data and clock in phase
CSL_SPI_PHASE_OUT	Data 1/2 cycle before clock

11.4.12 CSL_SpiShDir

enum CSL_SpiShDir
Enumeration to control direction of the word during transfer.

Enumeration values:

CSL_SPI_SHDIR_MSBFIRST	Transfer MSB first
CSL_SPI_SHDIR_LSBFIRST	Transfer LSB first

11.4.13 CSL_SpiOpMod

enum CSL_SpiOpMod
Enumeration to control the operating mode of Spi

Enumeration values:

CSL_SPI_OPMOD_MASTER	Operate as master
CSL_SPI_OPMOD_SLAVE	Operate as slave

11.4.14 CSL_SpiEnaHiZ

enum CSL_SpiEnaHiZ
Enumeration to control the SPIENA status when inactive

Enumeration values:

CSL_SPI_ENAHIZ_YES	Force SPIENA signal high-z when inactive
CSL_SPI_ENAHIZ_NO	Keep SPIENA signal a value when inactive

11.4.15 CSL_SpiGpioType

enum CSL_SpiGpioType
Enumeration to define status of SPI

Enumeration values:

CSL_SPI_GPIOTYPE_FUNC	SPI pins acts as GPIO
CSL_SPI_GPIOTYPE_DIR	Defines the direction of GPIO
CSL_SPI_GPIOTYPE_OPNDRAIN	Defines the open drain of GPIO

11.4.16 CSL_SpiPinType

enum CSL_SpiPinType
Enumeration for Spi serial communication pins

Enumeration values:

CSL_SPI_PINTYPE_SOMI	SOMI pin
CSL_SPI_PINTYPE_SIMO	SIMO pin
CSL_SPI_PINTYPE_CLK	CLK pin
CSL_SPI_PINTYPE_ENA	ENA pin
CSL_SPI_PINTYPE_SCS	SCS pin

11.4.17 CSL_SpiCptDma

enum CSL_SpiCptDma
Enumeration to control DMA enabling in compatibility mode

Enumeration values:

CSL_SPI_CPTDMA_ENABLE	Enable dma servicing in compatibility mode
CSL_SPI_CPTDMA_DISABLE	Disable dma servicing in compatibility mode

11.4.18 CSL_SpiXferEn

enum CSL_SpiXferEn
Enumeration to control reset of transfer mechanism of Spi

Enumeration values:

CSL_SPI_XFEREN_DISABLE	Enable spi to begin transfers
CSL_SPI_XFEREN_ENABLE	Hold spi transfer mechanism in reset

11.5 Macros

#define CSL_SPI_CONFIG_DEFAULTS
Value:

```
{    \
        CSL_SPI_SPIGCR0_RESETVAL     \
        CSL_SPI_SPIGCR1_RESETVAL     \
        CSL_SPI_SPIINT0_RESETVAL     \
        CSL_SPI_SPILVL_RESETVAL      \
        CSL_SPI_SPIFLG_RESETVAL      \
        CSL_SPI_SPIPC0_RESETVAL      \
        CSL_SPI_SPIPC1_RESETVAL      \
        CSL_SPI_SPIPC2_RESETVAL      \
        CSL_SPI_SPIPC3_RESETVAL      \
        CSL_SPI_SPIPC4_RESETVAL      \
        CSL_SPI_SPIPC5_RESETVAL      \
        CSL_SPI_SPIDAT0_RESETVAL     \
        CSL_SPI_SPIDAT1_RESETVAL     \
        CSL_SPI_SPIBUF_RESETVAL      \
        CSL_SPI_SPIDELAY_RESETVAL    \
        CSL_SPI_SPIDEF_RESETVAL      \
        CSL_SPI_SPIFMT_RESETVAL      \
        CSL_SPI_TGINTVEC_RESETVAL    \
}
```

Default Values for Config structure

#define CSL_SPI_HWSETUP_CPT_DEFAULTS
Value:

```
{ \
    NULL,                                   \
    NULL                                    \
    }
```

Default setting for *CSL_SpiHwSetupCpt*

#define CSL_SPI_HWSETUP_DEFAULTS
Value:

```
{ \
    NULL,                                \
    NULL,                                \
    NULL,                                \
    NULL                                 \
    }
```

Default setting for *CSL_SpiHwSetup*

#define CSL_SPI_HWSETUP_FMTCTRL_DEFAULTS
Value:

```
{ \
    CSL_SPI_CSHOLD_NO,                      \
    CSL_SPI_WDELAYEN_NO,                    \
    CSL_SPI_FMTSEL_0,                       \
    0,                                      \
    }
```

Default setting for *CSL_SpiHwSetupFmtCtrl*

#define CSL_SPI_HWSETUP_GEN_DEFAULTS
Value:

```
{ \
    CSL_SPI_OPMOD_SLAVE,                    \
    CSL_SPI_ENAHIZ_NO,                      \
    0,                                      \
    0,                                      \
    0,                                      \
    0,                                      \
    0                                       \
    }
```

Default setting for *CSL_SpiHwSetupGen*

#define CSL_SPI_HWSETUP_PINS_DEFAULTS
Value:

```
{ \
    NULL,                                    \
    NULL,                                    \
    NULL,                                    \
    NULL                                     \
    }
```

Default setting for *CSL_SpiHwSetupPins*

#define CSL_SPI_HWSETUP_PRI_DEFAULTS
Value:

```
{
        NULL,                               \
        NULL,                               \
        NULL,                               \
        NULL                                \
    }
```

Default setting for *CSL_SpiHwSetupPri*

#define CSL_SPI_HWSETUP_PRI_FMT_DEFAULTS
Value:

```
{ \
    0,                                       \
    0,                                       \
    0,                                       \
    CSL_SPI_WAITEN_NO,                       \
    CSL_SPI_PARITY_DISABLE,                  \
    CSL_SPI_PHASE_IN,                        \
    CSL_SPI_SHDIR_LSBFIRST                   \
    }
```

Default setting for *CSL_SpiHwSetupPriFmt*

Chapter 12
UHPI Module

Topics

12.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within UHPI module.

Universal Host Port Interface (UHPI) provides a parallel port through which an external host processor can access a CPU's memory space. The UHPI enables a host device and CPU to exchange information via internal or external memory. Connectivity to the CPU's memory space is provided through the UHPI's Vbus master interface. The Vbus master initiates CPU memory accesses through the DMA. Dedicated address and Data registers (HPIA and HPID) within the UHPI provide the data path between the external host interface and the Vbus master interface. A UHPI control register (HPIC) is available to the host and the CPU for various configuration and interrupt functions.

Different modes of operation may be performed by the UHPI peripheral. The desired operating modes are determined by the width of the UHPI data bus and the width of the CPU memory that the UHPI will be accessing.

- 32-bit Multiplexed mode (32 bit data bus/32 bit memory width)
- 32-bit Non-multiplexed mode (32 bit data bus/32 bit memory width)

Multiplexed mode supports hosts with multiplexed address and data lines. In this mode, the HPIA register is used to store the CPU memory address to be accessed by the UHPI.

The non-multiplexed mode utilizes a dedicated address bus and does not require an HPIA register. The HPIC control register is still accessible in this mode.

12.2 Functions

This section lists the functions available in the UHPI module.

12.2.1 CSL_uhpiInit

CSL_Status CSL_uhpiInit (CSL_UhpiContext * *pContext*)

Description
This is the initialization function for the uhpi CSL. The function must be called before calling any other API from this CSL. This function reset the SYNC bridge and also configure the HPI HINT as output which is muxed with McASP. It returns status CSL_SOK.

Arguments

```
pContext   Pointer to module-context. As uhpi doesn't have any
           context based information user is expected to pass
           NULL.
```

Return Value
CSL_Status

- CSL_SOK - Always returns

Pre Condition
CSL_sysInit() must be called.

Post Condition
The CSL for uhpi is initialized.

Modifies
None

Example

```
CSL_Status       status;
...
status = CSL_uhpiInit(NULL);
...
```

12.2.2 CSL_uhpiOpen

CSL_UhpiHandle CSL_uhpiOpen

(CSL_UhpiObj * *pUhpiObj*,
CSL_InstNum *uhpiNum*,
CSL_UhpiParam * *pUhpiParam*,
CSL_Status * *pStatus*
)

Description
This function returns the handle to the UHPI controller instance. This handle is passed to all other CSL APIs.

Arguments

```
uhpiObj     Pointer to uhpi object.

uhpiNum     Instance of DSP UHPI CSL to be opened.
            There is only one instance of the uhpi available.
            So, the value for this parameter will be
            CSL_UHPI always.

pUhpiParam  Module specific parameters.

pStatus     Status of the function call
```

Return Value
CSL_UhpiHandle
Valid uhpi handle will be returned if status value is equal to CSL_SOK.

Pre Condition
The HPI must be successfully initialized via CSL_*uhpiInit*() before calling this function

Post Condition
1. The status is returned in the status variable. If status returned is

- CSL_SOK Valid - uhpi handle is returned
- CSL_ESYS_FAIL - The uhpi instance is invalid
- CSL_ESYS_INVPARAMS - Invalid parameter

2. UHPI object structure is populated.

Modifies
1. The status variable
2. UHPI object structure

Example

```
CSL_Status          status;
CSL_UhpiObj         uhpiObj;
CSL_UhpiHandle      hUhpi;
...
hUhpi = CSL_uhpiOpen(
                    &uhpiObj,
                    CSL_UHPI,
                    NULL,
                    &status);
...
```

12.2.3 CSL_uhpiClose

CSL_Status CSL_uhpiClose (CSL_UhpiHandle *hUhpi*)

Description
This function closes the specified instance of UHPI.

Arguments

```
hUhpi            Handle to the UHPI
```

Return Value
CSL_Status

- CSL_SOK - Close successful
- CSL_ESYS_BADHANDLE - Invalid handle

Pre Condition
Both CSL_uhpiInit() and CSL_uhpiOpen() must be called successfully in order before calling CSL_uhpiClose().

Post Condition
The UHPI CSL APIs can not be called until the UHPI CSL is reopened again using CSL_uhpiOpen().

Modifies
Obj structure values

Example

```
CSL_UhpiHandle      hUhpi;
CSL_Status          status;
...
status = CSL_uhpiClose(hUhpi);
...
```

12.2.4 CSL_uhpiHwSetup

CSL_Status CSL_uhpiHwSetup (CSL_UhpiHandle *hUhpi*, CSL_UhpiHwSetup * *hwSetup*)

Description
It configures the uhpi registers as per the values passed in the hardware setup structure.

Arguments

```
hUhpi               Handle to the uhpi

hwSetup             Pointer to hardware setup structure
```

Return Value
CSL_Status

- CSL_SOK - Hardware setup successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Hardware structure is not properly initialized

Pre Condition
Both CSL_uhpiInit() and CSL_uhpiOpen() must be called successfully in order before calling this function.

Post Condition
DSP UHPI controller registers are configured according to the hardware setup parameters.

Modifies
DSP UHPI controller registers

Example

```
CSL_Status          status;
CSL_UhpiHwSetup     myHwSetup;
CSL_UhpiHandle      hUhpi;

myHwSetup.gpIntCtrl.gpIntEnable = CSL_UHPI_GPINT_EN0_ENABLE;
myHwSetup.gpIntCtrl.gpIntInvEnable = CSL_UHPI_GPINT_INV_ENABLE;
...
status = CSL_uhpiHwSetup(hUhpi, &hwSetup);
...
```

12.2.5 CSL_uhpiHwControl

CSL_Status CSL_uhpiHwControl **(** CSL_UhpiHandle ***hUhpi***, CSL_UhpiHwControlCmd ***cmd***, **void *** ***arg*** **)**

Description
Takes a command of UHPI with an optional argument and implements it.

Arguments

hUhpi	Handle to the UHPI instance
cmd	The command to this API indicates the action to be taken on UHPI.
arg	An optional argument.

Return Value
CSL_Status

- CSL_SOK - Status info return successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition
CSL_uhpiInit() and CSL_uhpiOpen() must be called successfully in order before calling CSL_uhpiHwControl().

Post Condition
UHPI registers are configured according to the command passed.

Modifies
The hardware registers of UHPI.

Example

```
CSL_UhpiHandle        hUhpi;
CSL_UhpiHwControlCmd  cmd;
void                  arg;
CSL_Status            status;
...
status = CSL_uhpiHwControl (hUhpi, cmd, &arg);
...
```

12.2.6 CSL_uhpiGetHwStatus

CSL_Status CSL_uhpiGetHwStatus (CSL_UhpiHandle ***hUhpi***,
CSL_UhpiHwStatusQuery ***query***,
void * ***response***
)

Description
Gets the status of the different operations of UHPI.

Arguments

```
hUhpi         Handle to the UHPI instance

query         The query to this API of UHPI that indicates the
              status to be returned.

response      Placeholder to return the status.
```

Return Value
CSL_Status

- CSL_SOK - Status info return successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Invalid query command
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition
CSL_uhpiInit() and CSL_uhpiOpen() must be called successfully in order before calling CSL_uhpiGetHwStatus().

Post Condition
None

Modifies
Third parameter response value

Example

```
CSL_UhpiHandle          hUhpi;
CSL_UhpiHwStatusQuery   query;
void                    response;
CSL_Status              status;
...
status = CSL_uhpiGetHwStatus (hUhpi, query, &response);
...
```

12.2.7 CSL_uhpiHwSetupRaw

CSL_Status CSL_uhpiHwSetupRaw (CSL_UhpiHandle ***hUhpi***,
CSL_UhpiConfig * ***config***
)

Description
This function initializes the device registers with the register-values provided through the Config Data structure. This configures registers based on a structure of register values, as compared to HwSetup, which configures registers based on structure of bit field values and may perform other functions (delays, etc.)

Arguments

```
hUhpi       Handle to the UHPI instance

config      Pointer to config structure
```

Return Value
CSL_Status

- CSL_SOK - Configuration successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Configuration is not properly initialized

Pre Condition

CSL_uhpiInit() and CSL_uhpiOpen() must be called successfully in order before calling CSL_uhpiHwSetupRaw().

Post Condition

The registers of the specified UHPI instance will be setup according to input configuration structure values.

Modifies
Hardware registers of the specified UHPI instance.

Example

```
CSL_UhpiHandle       hUhpi;
CSL_UhpiConfig       config = CSL_UHPI_CONFIG_DEFAULTS;
CSL_Status           status;
...
status = CSL_uhpiHwSetupRaw (hUhpi, &config);
...
```

12.2.8 CSL_uhpiGetHwSetup

CSL_Status CSL_uhpiGetHwSetup (CSL_UhpiHandle *hUhpi*, CSL_UhpiHwSetup * *hwSetup*)

Description
It retrieves the hardware setup parameters of the uhpi specified by the given handle.

Arguments

hUhpi	Handle to the uhpi
hwSetup	Pointer to the hardware setup structure

Return Value
CSL_Status

- CSL_SOK - Retrieving the hardware setup parameters is successful
- CSL_ESYS_BADHANDLE - The handle is passed is invalid
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition
CSL_uhpiInit() and CSL_uhpiOpen() must be called successfully in order before calling CSL_uhpiGetHwSetup().

Post Condition
The hardware setup structure is populated with the hardware setup parameters.

Modifies
hwSetup variable

Example

```
CSL_UhpiHandle   hUhpi;
CSL_UhpiHwSetup  hwSetup;
CSL_Status       status;
...
status = CSL_uhpiGetHwSetup(hUhpi, &hwSetup);
...
```

12.2.9 CSL_uhpiGetBaseAddress

CSL_Status CSL_uhpiGetBaseAddress (CSL_InstNum *uhpiNum*,
CSL_UhpiParam * *pUhpiParam*,
CSL_UhpiBaseAddress * *pBaseAddress*
)

Description
This function is used for getting the base address of the peripheral instance. This function is called inside the CSL_uhpiOpen() function call. This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and thereby allowing CSL initiated write/reads into peripheral MMRs go to an alternate location.

Arguments

```
uhpiNum         Specifies the instance of the uhpi to be opened

pUhpiParam      Module specific parameters.

pBaseAddress    Pointer to base address structure containing
                base address details.
```

Return Value
CSL_Status

- CSL_SOK Open call is successful
- CSL_ESYS_FAIL The instance number is invalid.
- CSL_ESYS_INVPARAMS Invalid parameter

Pre Condition
None

Post Condition
Base address structure is populated.

Modifies
1. The status variable
2. Base address structure is modified.

Example

```
CSL_Status              status;
CSL_UhpiBaseAddress  baseAddress;
```

```
...
status = CSL_uhpiGetBaseAddress(CSL_UHPI_0, NULL,
                                &baseAddress);
...
```

12.3 Data Structures

This section lists the data structures available in the UHPI module.

12.3.1 CSL_UhpiObj

Detailed Description
UHPI object structure.

Field Documentation

CSL_UhpiRegsOvly CSL_UhpiObj::regs
Pointer to the register overlay structure of the uhpi

CSL_InstNum CSL_UhpiObj::uhpiNum
Instance of uhpi being referred by this object

12.3.2 CSL_UhpiConfig

Detailed Description
This structure is used to setup the configuration structure.

Field Documentation

volatile Uint32 CSL_UhpiConfig::GPINT_CTRL
General Purpose Interrupt Control Register

volatile Uint32 CSL_UhpiConfig::GPIO_DAT1
GPIO Data 1 Register

volatile Uint32 CSL_UhpiConfig::GPIO_DAT2
GPIO Data 2 Register

volatile Uint32 CSL_UhpiConfig::GPIO_DAT3
GPIO Data 3 Register

volatile Uint32 CSL_UhpiConfig::GPIO_DIR1
GPIO Direction 1 Register

volatile Uint32 CSL_UhpiConfig::GPIO_DIR2
GPIO Direction 2 Register

volatile Uint32 CSL_UhpiConfig::GPIO_DIR3
GPIO Direction 3 Register

volatile Uint32 CSL_UhpiConfig::GPIO_EN
GPIO Enable Register

volatile Uint32 CSL_UhpiConfig::HPIAR
Host Port Interface Read Address Register

volatile Uint32 CSL_UhpiConfig::HPIAW

Host Port Interface Write Address Register

12.3.3 CSL_UhpiContext

Detailed Description
Module specific context information. Present implementation of uhpi CSL doesn't have any context information.

Field Documentation

Uint32 CSL_UhpiContext::contextInfo
Context information of uhpi CSL. The declaration is just a placeholder for future implementation.

12.3.4 CSL_UhpiParam

Detailed Description
Module specific parameters. Present implementation of uhpi CSL doesn't have any module specific parameters.

Field Documentation

CSL_BitMask32 CSL_UhpiParam::flags
Bit mask to be used for module specific parameters. The declaration is just a placeholder for future implementation.

12.3.5 CSL_UhpiHwSetup

Detailed Description
Structure to configure Hardware Setup.

Field Documentation

CSL_UhpiGpIntCtrl CSL_UhpiHwSetup::gpIntCtrl
General Purpose Interrrupt Control Register

CSL_UhpiGpioData CSL_UhpiHwSetup::gpioData
General Purpose I/O Data Register

CSL_UhpiGpioDir CSL_UhpiHwSetup::gpioDir
General Purpose I/O Direction Register

CSL_UhpiGpioEnable CSL_UhpiHwSetup::gpioEnable
General Purpose I/O Enable Register

CSL_UhpiAddrCfg CSL_UhpiHwSetup::hpiAddr
Host port Interface Read & Write Address Register

CSL_UhpiCtrl CSL_UhpiHwSetup::hpiCtrl
Host port Interface control Register

12.3.6 CSL_UhpiBaseAddress

Detailed Description
This structure contains the base-address information for the peripheral instance of the UHPI.

Field Documentation

CSL_UhpiRegsOvly CSL_UhpiBaseAddress::regs
Base-address of the configuration registers of the peripheral

12.3.7 CSL_UhpiAddrCfg

Detailed Description
Structure configures Host Port Interface Write & Read Address.

Field Documentation

Uint32 CSL_UhpiAddrCfg::hpiaReadAddr
Host Port Interface Read Address

Uint32 CSL_UhpiAddrCfg::hpiaWrtAddr
Host Port Interface Write Address

12.3.8 CSL_UhpiGpioData

Detailed Description
This structure is used to configure the GPIO Data.

Field Documentation

Uint32 CSL_UhpiGpioData::gpioData1
GPIO Direction - 1 Register

CSL_UhpiGpioDat2 CSL_UhpiGpioData::gpioData2
GPIO Direction - 2 Register

Uint32 CSL_UhpiGpioData::gpioData3
GPIO Direction - 3 Register

12.3.9 CSL_UhpiGpioDir

Detailed Description
This structure is used to configure the GPIO Direction.

Field Documentation

Uint32 CSL_UhpiGpioDir::gpioDir1
GPIO Direction - 1 Register

CSL_UhpiGpioDir2 CSL_UhpiGpioDir::gpioDir2
GPIO Direction - 2 Register

Uint32 CSL_UhpiGpioDir::gpioDir3
GPIO Direction - 3 Register

12.3.10 CSL_UhpiGpIntCtrl

Detailed Description
Structure to configure General Purpose Interrupt Control.

Field Documentation

CSL_UhpiGpIntEnable CSL_UhpiGpIntCtrl::gpIntEnable
GPINT control Interrupt Enable

CSL_UhpiGpIntInvEnable CSL_UhpiGpIntCtrl::gpIntInvEnable
GPINT control Interrupt Invert Enable

12.4 Enumerations

This section lists the enumerations available in the UHPI module.

12.4.1 CSL_UhpiHwStatusQuery

enum CSL_UhpiHwStatusQuery
Enumeration for queries passed to *CSL_uhpiGetHwStatus()*.
Enumeration for hardware status query commands

Enumeration values:

CSL_UHPI_QUERY_PID_REV — Query the current value of Peripheral Revision Query the current value of Peripheral Class.
Parameters:
*(Uint32 *)*

CSL_UHPI_QUERY_PID_CLASS — Query the current value of Peripheral Type.
Parameters:
*(Uint32 *)*

CSL_UHPI_QUERY_PID_TYPE — Query the current value of GP Interrupt Enable-0.
Parameters:
*(Uint32 *)*

CSL_UHPI_QUERY_GPINT_EN0 — Query the current value of GP Interrupt Enable 1.
Parameters:
*(Uint32 *)*

CSL_UHPI_QUERY_GPINT_EN1 — Query the current value of GP Interrupt Enable 2.
Parameters:
*(Uint32 *)*

CSL_UHPI_QUERY_GPINT_EN2 — Query the current value of GPIO Direction 1.
Parameters:
*(Uint32 *)*

CSL_UHPI_QUERY_GPIO_DIR1 — Query the current value of GPIO Direction 2.
Parameters:
*(Uint32 *)*

CSL_UHPI_QUERY_GPIO_DIR2 — Query the current value of GPIO Direction 3.
Parameters:
*(Uint32 *)*

CSL_UHPI_QUERY_GPIO_DIR3 — Query the current value of Host Ready.
Parameters:
*(Uint32 *)*

CSL_UHPI_QUERY_HRDY — Query the current value of Host Fetch.
Parameters:
*(Uint32 *)*

CSL_UHPI_QUERY_FETCH — Query the current value of HPI Reset.
Parameters:
*(Uint32 *)*

CSL_UHPI_QUERY_HPI_RST — Query the current value of Half-word ordering status.
Parameters:
*(Uint32 *)*

12.4.2 CSL_UhpiHwControlCmd

enum CSL_UhpiHwControlCmd
Enumeration for queries passed to *CSL_uhpiHwControl()*.
Enumeration for hardware set commands

Enumeration values:

CSL_UHPI_CMD_GPINT_SET	Set the GP Interrupt set the GP Interrupt Invert **Parameters:** *Uint32*
CSL_UHPI_CMD_SET_GPINT_INV	Set the GPIO Enable **Parameters:** *Uint32*
CSL_UHPI_CMD_SET_GPIO_EN	Set the GPIO Direction-1 **Parameters:** *Uint32*
CSL_UHPI_CMD_SET_GPIO_DIR1	Set the GPIO Direction-2 **Parameters:** *Uint16*
CSL_UHPI_CMD_SET_GPIO_DIR2	Set the GPIO Direction-3 **Parameters:** *Uint32*
CSL_UHPI_CMD_SET_GPIO_DIR3	Set the HPIC Host-to-DSP Interrupt. **Parameters:** *(None)*
CSL_UHPI_CMD_SET_DSP_INT	Reset the HPIC Host-to-DSP Interrupt. **Parameters:** *(None)*
CSL_UHPI_CMD_RESET_DSP_INT	Set the HPIC DSP-to-Host Interrupt. **Parameters:** *(None)*
CSL_UHPI_CMD_SET_HINT	Reset the HPIC DSP-to-Host Interrupt. **Parameters:** *(None)*

12.4.3 CSL_UhpiCtrl

enum CSL_UhpiCtrl
Enumeration for Host Port Interface Control.

Enumeration values:

CSL_UHPI_HWOB	Half-word Ordering Bit
CSL_UHPI_DSP_INT	Host-to-DSP Interrupt
CSL_UHPI_HINT	DSP-to-Host Interrupt
CSL_UHPI_HRDY	Host Ready
CSL_UHPI_FETCH	Host Fetch
CSL_UHPI_XHPIA	UHPI Extended Address
CSL_UHPI_HPI_RST	HPI Reset

CSL_UHPI_HWOB_STAT	Half-word ordering bit status
CSL_UHPI_DUAL_HPIA	Dual HPIA mode configuration bit
CSL_UHPI_HPIA_RW_SEL	HPIA register select bit

12.4.4 CSL_UhpiGpioDat2

enum CSL_UhpiGpioDat2
Enumeration for General Purpose I/O Data.

Enumeration values:

CSL_UHPI_GPIO_DAT0	This DAT bit corresponds to the /HAS pin
CSL_UHPI_GPIO_DAT1	Controls/ states the level of the /HCS pin
CSL_UHPI_GPIO_DAT2	Controls/states the level of the /HDS1 pin
CSL_UHPI_GPIO_DAT3	Controls/states the level of the /HDS1 pin
CSL_UHPI_GPIO_DAT4	Controls/states the level of the HR/W pin
CSL_UHPI_GPIO_DAT5	Controls/states the level of the HHWIL pin
CSL_UHPI_GPIO_DAT6	Controls/states the level of the HCNTLB pin
CSL_UHPI_GPIO_DAT7	Controls/states the level of the HCNTLA pin
CSL_UHPI_GPIO_DAT8	Controls/states the level of the /HINT pin
CSL_UHPI_GPIO_DAT9	Control/states the level of the HRDY pin
CSL_UHPI_GPIO_DAT10	Control/states the level of the /HRDY pin
CSL_UHPI_GPIO_DAT11	Control/states the level of the /HBE0 pin
CSL_UHPI_GPIO_DAT12	Control/states the level of the /HBE1 pin
CSL_UHPI_GPIO_DAT13	Control/states the level of the /HBE2 pin
CSL_UHPI_GPIO_DAT14	Control/states the level of the /HBE3 pin

12.4.5 CSL_UhpiGpioDir2

enum CSL_UhpiGpioDir2
Enumeration for General Purpose I/O Direction

Enumeration values:

CSL_UHPI_GPIO_DIR0	Control the direction of the /HAS pin
CSL_UHPI_GPIO_DIR1	Control the direction of the /HCS pin
CSL_UHPI_GPIO_DIR2	Control the direction of the /HDS1 pin
CSL_UHPI_GPIO_DIR3	Control the direction of the /HDS2 pin
CSL_UHPI_GPIO_DIR4	Control the direction of the HR/W pin
CSL_UHPI_GPIO_DIR5	Control the direction of the HHWIL pin
CSL_UHPI_GPIO_DIR6	Control the direction of the HCNTLB pin
CSL_UHPI_GPIO_DIR7	Control the direction of the HCNTLA pin
CSL_UHPI_GPIO_DIR8	Control the direction of the /HINT pin
CSL_UHPI_GPIO_DIR9	Control the direction of the HRDY pin
CSL_UHPI_GPIO_DIR10	Control the direction of the /HRDY pin
CSL_UHPI_GPIO_DIR11	Control the direction of the /HBE0 pin
CSL_UHPI_GPIO_DIR12	Control the direction of the /HBE1 pin
CSL_UHPI_GPIO_DIR13	Control the direction of the /HBE2 pin
CSL_UHPI_GPIO_DIR14	Control the direction of the /HBE3 pin

12.4.6 CSL_UhpiGpioEnable

enum CSL_UhpiGpioEnable
Enumeration for GPIO Enable.

Enumeration values:

CSL_UHPI_GPIO_EN0	GPIO_EN0 bank includes the /HCS /HDS1 /HDS2 pins
CSL_UHPI_GPIO_EN1	GPIO_EN1 bank includes HCNTL[B:A] pins
CSL_UHPI_GPIO_EN2	GPIO_EN2 bank includes the /HAS pin
CSL_UHPI_GPIO_EN3	GPIO_EN3 bank includes the /HBE[3:0]
CSL_UHPI_GPIO_EN4	GPIO_EN4 bank includes the /HHWIL pin
CSL_UHPI_GPIO_EN5	GPIO_EN5 bank includes the /HRDY pin
CSL_UHPI_GPIO_EN6	GPIO_EN6 bank includes the /HINT pin
CSL_UHPI_GPIO_EN7	GPIO_EN7 bank includes the HD[7:0] pin
CSL_UHPI_GPIO_EN8	GPIO_EN8 bank includes the HD[15:8] pin
CSL_UHPI_GPIO_EN9	GPIO_EN9 bank includes the HD[23:16] pin
CSL_UHPI_GPIO_EN10	GPIO_EN10 bank includes the HD[31:24] pin
CSL_UHPI_GPIO_EN11	GPIO_EN10 bank includes the HA[0:7] pin
CSL_UHPI_GPIO_EN12	GPIO_EN10 bank includes the HA[15:8] pin
CSL_UHPI_GPIO_EN13	GPIO_EN10 bank includes the HA[19:16] pin
CSL_UHPI_GPIO_EN14	GPIO_EN10 bank includes the HA[23:20] pin
CSL_UHPI_GPIO_EN15	GPIO_EN10 bank includes the HA[27:24] pin
CSL_UHPI_GPIO_EN16	GPIO_EN10 bank includes the HA[31:28] pin

12.4.7 CSL_UhpiGpIntInvEnable

enum CSL_UhpiGpIntInvEnable
Enumeration for General Purpose Interrupt Invert.

Enumeration values:

CSL_UHPI_GPINT_INV0	GPINT_INV0 inverts the corresponding GPINT_EN0
CSL_UHPI_GPINT_INV1	GPINT_INV1 corresponds with the HCNTLA GPINT
CSL_UHPI_GPINT_INV2	GPINT_INV2 corresponds with the /HAS GPINT

12.4.8 CSL_UhpiGpIntEnable

enum CSL_UhpiGpIntEnable
Enumeration for General Purpose Interrupt Enable.

Enumeration values:

CSL_UHPI_GPINT_EN0	GPINT_EN0 enables HCNTLB as a GPINT sourcing DMA_WRITE_EVENT
CSL_UHPI_GPINT_EN1	GPINT_EN1 enables HCNTLA as a GPINT sourcing DMA_READ_EVENT
CSL_UHPI_GPINT_EN2	GPINT_EN2 enables /HAS as a GPINT sourcing the CPU Interrupt

12.5 Macros

#define CSL_UHPI_CONFIG_DEFAULTS
Value:

```
{\
    CSL_HPI_GPIO_EN_RESETVAL,\
    CSL_RTI_RTICOMPCTRL_RESETVAL,\
    CSL_HPI_GPIO_DIR1_RESETVAL,\
    CSL_HPI_GPIO_DAT1_RESETVAL,\
    CSL_HPI_GPIO_DIR2_RESETVAL,\
    CSL_HPI_GPIO_DAT2_RESETVAL,\
    CSL_HPI_GPIO_DIR3_RESETVAL,\
    CSL_HPI_GPIO_DAT3_RESETVAL,\
    CSL_HPI_HPIC_RESETVAL,\
    CSL_HPI_HPIAW_RESETVAL,\
    CSL_HPI_HPIAR_RESETVAL\

}
```

Default Values for Config structure.