TEXAS INSTRUMENTS

20450 Century Boulevard
Germantown, MD 20874
Fax: (301) 515-7954

100_CSLC6498

CSL legacy API change document

Revision D

QRSA 00013320

12 November 2009

Revision Record	
Document Title:	**Statement of Work**

Revision	Description of Change
A	Initial Release: covers legacy API changes for EDMA, SEM, CACHE, VCP2, IDMA, CHIP/TSC modules. Describes new CPINTC module (no changes to INTC module)
B	GPIO API changes vs. Legacy
C	QRSA update (footer changed to TI Confidential-NDA restrictions) for customer release
D	RAC API update

Note: Be sure the Revision of this document matches the QRSA record Revision letter. The revision letter increments only upon approval via the Quality Record System.

TABLE OF CONTENTS

1 Scope

This document defines the statement of work involved in the design and development of the CSL Functional layer for the following IP modules:-

1. CPINTC (new). Old INTC module didn't' change.
2. EDMA (API change vs. Legacy)
3. CACHE (API change vs. Legacy)
4. SEM (API change vs. Legacy)
5. VCP2 (API change vs. Legacy)
6. IDMA (API change vs. Legacy)
7. CHIP / TSC (API change vs. Legacy)
8. GPIO (API change vs. Legacy)
9. RAC (API change vs. Legacy)

CPINTC is new and we present here the basic function of this module and its mapping to the hardware. For the rest of the modules listed above we **map API changes vs. legacy code.** We explain in this document the change and the reasoning behind it.

For the rest of the modules (all new) included in the CSL release the complete API set is provided as doxygen-chm file.

2 References

The following references are related to the feature described in this document and shall be consulted as necessary.

No	Referenced Document	Control Number	Description
1	100_CSLC6498 PRD	QRSA00012825	Product Requirements

Table 1. Referenced Materials

3 Definitions

Acronym	Description
API	Application Programming Interface
DSP	Digital Signal Processor

Table 2. Definitions

4 Requirements

The goal of the Chip Support Library (CSL) is to abstract peripheral on-chip programming to the upper software-layers.

- Shorten the development time of an application using the CSL.
- Provide a consistent interface to peripheral registers, for portability of applications across devices.

The Chip Support library provides this abstraction by using the following components:-
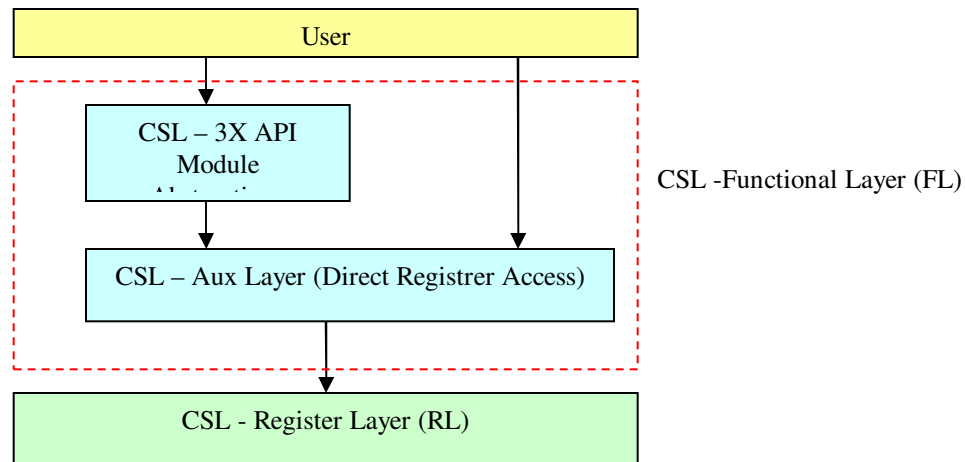
1. Register Layer
   The Register layer provided by the CSL incorporates a register overlay structure and field definition macros. These are auto-generated (using Autogen and / or Asha tools) based on the design XML/TCL MMR representation for a given IP module but may be subject to change by CSL Development team for reasons of code efficiency / performance. The data structures and definitions exposed here are used by the CSL functional layer.

2. Functional Layer
   The Functional layer provides a C API interface on top of the Register layer defined above. The functional layer abstracts from the users the following key concerns:-
   - *Multiple Instances*
     Certain IP blocks like the CPINTC, EDMA etc have multiple instances and this is abstracted by the CSL functional layer.
   - *Device Specific Mapping*
     Each IP block is mapped to a different memory mapped region. This mapping is device specific. The CSL functional layer abstracts the mapping from the users; the API definitions do not change thus ensuring that user applications remain consistent as they move from one device to another.
   - *IP Module Code*
     Each IP block can be internally broken up into two key components;
     IP module specific CSL code and device specific code as explained above. The CSL functional layer keeps this demarcation to maximize code reuse among different devices which share the same IP block.

The following figure showcases the top level architecture figure of the CSL Module

```
┌─────────────────────────────────────────────────┐
│                      User                         │
└─────────────────────────────────────────────────┘
        ┌─────────────────────────────┐
        │  ┌──────────────────┐        │   CSL -Functional Layer (FL)
        │  │  CSL – 3X API     │        │
        │  │  Module           │        │
        │  │                   │        │
        │  └──────────────────┘        │
        │                              │
        │  ┌──────────────────────────┐│
        │  │CSL – Aux Layer (Direct   ││
        │  │Registrer Access)         ││
        │  └──────────────────────────┘│
        └─────────────────────────────┘
        ┌─────────────────────────────────┐
        │   CSL - Register Layer (RL)       │
        └─────────────────────────────────┘
```

The CSL3X is the legacy API model which was used in previous releases. This API model enforces a set of well defined API which need to be implemented per API module such as `Open`, `Close`, `HwControl`, `HwStatus` etc. This model had the following issues:-

1. *Performance Impact*
   Since there was a single well defined API entry point for all operations with the registers there was a big switch statement in the code which would decode the command and then call the appropriate the register handling API.
2. *Programming Difficulty*
   Users of the CSL are familiar with the IP registers and IP module functionality; but since the CSL 3x style introduced another level of abstraction; users of the CSL would need to first read the CSL documentation before they could use the module. *For example*: Assume there is a register X which carries the module IP version information. For a user to retrieve this; they will first need to read the CSL documentation; determine that there is a command GET_PID and then pass that with the correct arguments to `HwStatus` API

The AUX Style on the other hand provides atomic register level operations and provides multiple API to be able to get/set different parameters in all the registers. All the functions in the AUX header files are provided as *inline* functions. The AUX style is also the backend which is also used by all the CSL3X style API. The CSL release will document all the API's in the AUX header files and users are encouraged to use these API.
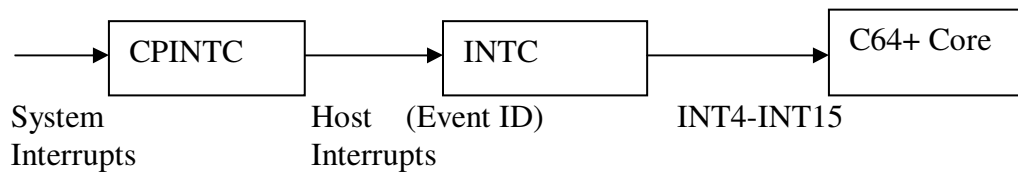
5   Overview of Work

5.1   CPINTC

The CPINTC module controls the system interrupt mapping to the host interrupt interface. System interrupts are generated by the IP modules. The CPINTC receives the system interrupts and maps them to internal channels. The channels are used to group interrupts together and to

prioritize them. These channels are then mapped onto the host interface that is typically a smaller number of host interrupts or a vector input.

This is a new IP module which was added only in the TCI649x devices,

The following is the architectural block diagram which showcases the CPINTC system architecture:-

```
            ┌──────────┐        ┌──────────┐        ┌──────────┐
  ────────► │  CPINTC  │ ─────► │   INTC   │ ─────► │ C64+ Core│
            └──────────┘        └──────────┘        └──────────┘
System              Host    (Event ID)       INT4-INT15
Interrupts          Interrupts
```

The CSL Functional layer for CPINTC had to account for the following:-
1. TCI649x devices support multiple instances of the CPINTC IP module
2. Parameters in the each instance of the CPINTC IP module are different. _For example_:- Number of System Interrupts supported on Instance 0 is 200 while on Instance 1 there are only 124.

The CPINTC is designed to conform to the AUX style described above.

5.2 EDMA

The Enhanced Direct Memory Access (EDMA3) controller's primary purpose is to service user-programmed data transfers between two memory-mapped slave endpoints on the device. The EDMA3 module is a legacy IP module which was present in previous CSL releases.

However for the TCI649x devices there have been some modifications which affect the CSL Functional layer:-
1. Multiple EDMA3 CC Instances are supported
2. Each EDMA3 CC IP Instance has a different set of configuration parameters. _For example:_ Number of DMA channels for Instance 0 is 16 but the DMA Channels for Instance 1 and 2 are 64.

The EDMA3 module in legacy CSL modules was developed according to the 3X style API but was limited to operate only for a single instance (thus only 1 set of configuration parameters). The new EDMA3 CSL Functional layer has to be extended to operate for multiple instances and differing set of configuration parameters.

Currently the EDMA3 CSL Functional layer assumes that if the channel number is greater than the maximum number of DMA channels then it is a QDMA channel. This will have to be replaced with a run time check which stores configuration parameters of each instance.

Since the EDMA3 CSL Functional layer is used extensively by various customers; it was decided to retain the 3X style API. However the AUX style backend is enhanced and many new API's have been added. These modifications will be abstracted from users since the use the 3X Style API which have not been modified.

5.3 SEMAPHORE

In any multi-tasking environment where system resources must be shared, the problem of controlling simultaneous accesses to a resource must be solved. In order to ensure proper system operation, it is necessary to allow that a resource is accessed by one, and only one, C64x+ core at a time. That is, it is necessary to provide mutual exclusion for the shared resource.

The CSL SEM Functional Layer is based on the CSL3x API model. While the CSL3x API model abstracts the Auxiliary header files and thus Auxiliary API modifications are not directly visible to the end users, after review of the internal Auxiliary header files and there are some discrepancies which were observed.

Semaphores can be requested by a master in the following ways:-
1. **Direct Access**
    In this access a semaphore if available is immediately allocated to the master; else the semaphore request fails.
2. **Indirect Access**
    In this access a semaphore request is placed into a queue and when the request can be serviced an interrupt is generated.
3. **Combined Access**
    This access is a combination of the above 2.

Indirect Access Semaphore can be acquired by writing a value of 0 to any of the following 3 types of registers:-
a) Semaphore Direct Registers (SEM in the Register Overlay)
b) Semaphore Indirect Registers (ISEM in the Register Overlay)
c) Semaphore Query Register (QSEM in the Register Overlay)

Now if we consider the current CSL API:-

```
static inline void CSL_semAcquireDirect (CSL_SemHandle hSem)
```

This API unlike what the name suggests is actually used to acquire an indirect access semaphore **((2) above)** request. The name DIRECT in the API implies that the semaphore resource is allocated using the SEMAPHORE DIRECT Registers **((a) above)**.

Similarly the current CSL API:-

```
static inline void CSL_semAcquireIndirect(CSL_SemHandle hSem)
```

This API also acquires an indirect access semaphore request but the INDIRECT in the name implies that the semaphore resource is allocated using the SEMAPHORE Indirect Registers **((b) above)**.

Taking into account these discrepancies and also the 3X style cons mentioned above; the CSL Functional layer for the semaphore module will be implemented as per the AUX style.

The following table summarizes the changes to the CSL SEM API:-

Old API	New API	Comments
CSL_semHwControl Cmd= CSL_SEM_CMD_FREE_DIRECT	CSL_semReleaseSemaphore	Release a previously acquired semaphore.
CSL_semHwControl Cmd= CSL_SEM_CMD_WRITE_POST_DIRECT	CSL_semAcquireIndirect	Acquire a semaphore for Indirect Access request using (a) above
CSL_semHwControl Cmd= CSL_SEM_CMD_WRITE_POST_INDIRECT	CSL_semAcquireIndirect	Acquires a semaphore for Indirect Access using (b) above
CSL_semHwControl Cmd= CSL_SEM_CMD_WRITE_POST_QUERY	CSL_semAcquireIndirect	Acquires a semaphore for Indirect Access using (c) above
CSL_semHwControl Cmd= CSL_SEM_CMD_EOI_WRITE	CSL_semSetEoi	
CSL_semHwControl Cmd= CSL_SEM_CMD_CLEAR_FLAGS	CSL_semClearFlags	
CSL_semHwControl Cmd= CSL_SEM_CMD_CLEAR_ERR	CSL_semClearError	
CSL_semGetHwStatus Cmd= CSL_SEM_QUERY_ERROR	CSL_semGetErrorCode CSL_semGetErrorSemaphoreNumber CSL_semGetErrorFaultID	
CSL_semGetHwStatus Cmd= CSL_SEM_QUERY_FLAGS	CSL_semGetFlags	
CSL_semGetHwStatus Cmd=CSL_SEM_QUERY_STATUS	CSL_semIsFree	
CSL_semGetHwStatus Cmd=CSL_SEM_QUERY_DIRECT	CSL_semAcquireDirect	This gets a semaphore using direct access. The API is confusing because to acquire indirect semaphore the HwControl API was used.
CSL_semGetHwStatus Cmd= CSL_SEM_QUERY_INDIRECT	CSL_semAcquireCombined	The old API is confusing because the API gets a semaphore for combined

		access but this is indicated neither in the command name nor API name.

5.4 CACHE

The CSL CACHE Functional Layer from the legacy code was analyzed and the following issues were seen:-

1. **Global Variables.**
   This is an issue since a global variable (*For Example*: `_CSL_cachebusyState`) implies that the CSL API now need to be protected through critical sections. *For example:* If the CSL API is being used by a TASK and also by an ISR; then the system developers need to protect the CSL API through critical sections; else this global variable is overwritten and could result in incorrect behavior.

2. **Enable and Disable Interrupts:**
   The CSL API enables/disables interrupts in the system. This could have an impact on system interrupt latency. Developers can determine if interrupts need to be disabled and if required do so before calling the API.

In order to solve these issues the CSL functional layer was redesigned and implemented as per the new AUX style described above.

The following table summarizes the changes to the CSL API's:-

Old API	New API	Comments
CACHE_L1_Freeze CACHE_freezeL1d(void)	void CACHE_freezeL1D(void)	Old API returned the previous state of the L1D cache. The new one simply sets the FREEZE Mode. Instead use the CACHE_getPrevL1DMode () API to get the previous mode if so required.
N/A	Uint8 CACHE_getPrevL1DMode(void)	Newer API which returns the previous state of the L1D Cache (POPER)
CACHE_L1_Freeze CACHE_unfreezeL1d (void)	void CACHE_unfreezeL1D(void)	Old API returned the previous state of the L1D cache. Instead use the CACHE_getPrevL1DMode () API to get the previous

		mode if so required.
CACHE_L1_Freeze CACHE_freezeL1(**void**)	N/A	The API freezes the L1P and L1D Cache. The same can be achieved by calling CACHE_freezeL1d and CACHE_freezeL1p.

CACHE_wait	CACHE_invAllL1dWait
	CACHE_wbInvL1dWait
	CACHE_invAllL1pWait
	CACHE_invL1dWait
	CACHE_invL1pWait
	CACHE_wbAllL1dWait
	CACHE_wbInvAllL1dWait
	CACHE_wbInvL1dWait
	CACHE_wbL1dWait
		The OLD API used to maintain state in the global variable. Once a CACHE Invalidate operation is done. The global variable would be set and then when the CACHE_wait API was called; depending on the global variable the function would "Loop" around on the correct register waiting for the operation to complete.
		
		This intelligence has been removed and instead the WAIT API have been exposed on a per CACHE operation. Developers will need to call the correct API to wait for a specific operation to complete. For example:-
		
		    CACHE_wbInvAllL1d (CACHE_NOWAIT);
		    ...
		
		CACHE_wbInvAllL1dWait();
		
		On the other hand if an API is invoked as follows:-
		
		    CACHE_wbInvAllL1d (CACHE_WAIT);
		
		Then the WAIT API does not need to be invoked; as previously.

N/A	Uint8 CACHE_getPrevL1PMode(void)	Newer API which returns the previous state of the L1P Cache(POPER)

5.5   VCP2

The VCP2 module CSL Functional Layer (FL) APIs were originally designed and written such that only one instance of VCP2 CSL-FL module could be used at any time in the system. This

approach was best suited for single core scenarios. For a multi core device like TCI6498, these legacy APIs wouldn't scale well and hence the VCP2 CSL-FL APIs have been extended to support multiple instances of the module.

Multiple instance support in VCP2 has been added two fold:

1. Definition of new data structures to hold VCP2 instance handle related information.

2. API definitions and implementations have been changed to use the VCP2 instance handle to configure, retrieve VCP2 peripheral information for the corresponding VCP2 peripheral instance.

Also, a new API and data structure have been added to support retrieval of peripheral identifier and revision information associated with a VCP2 instance.

The following new data structure definitions have been added in VCP2 towards multiple instance support:

```
typedef volatile CSL_Vcp2EdmaRegs        *CSL_Vcp2RegsOvly;
typedef volatile CSL_Vcp2ConfigRegs      *CSL_Vcp2CfgRegsOvly;


/**
 * This will have the base-address information for the peripheral instance
 */
typedef struct {
   /** Base-address of the VCP2 registers accessible through Data bus */
   CSL_Vcp2RegsOvly   regs;
   /** Base-address of the Configuration registers of VCP2  */
   CSL_Vcp2CfgRegsOvly cfgregs;
} VCP2BaseAddress;


/**
 * This structure/object holds the context of the instance of VCP2
 * opened using VCP2_init() function.
 *
 * Pointer to this object is passed as VCP2 Handle to all VCP2 CSL APIs.
 * VCP2_init() function intializes this structure based on the parameters
 * passed
 */
typedef struct VCP2Obj {
   /** Pointer to the register overlay structure of the Tcp2 */
   CSL_Vcp2RegsOvly   regs;
```

```
   /** Pointer to the Configuration registers overlay structure of the Tcp2 */
   CSL_Vcp2CfgRegsOvly cfgregs;
   /** Instance of TCP2 being referred by this object  */
   CSL_InstNum        perNum;
} VCP2Obj;


/** This is a pointer to Vcp2Obj and is passed as the first
 *  parameter to all VCP2 CSL APIs
 */
typedef struct VCP2Obj *VCP2Handle;
```

The following data structure definition has been added to support the retrieval of Peripheral Id information:

```
/** VCP Peripheral ID structure */
typedef struct
{
   /** Peripheral type */
   Uint8 type;
   /** Peripheral class */
   Uint8 pid_class;
   /** Peripheral revision */
   Uint8 rev;
} VCP2_PID;
```

Following are the API changes in the new VCP2 CSL-FL:

Old API	New API	Comments
N/A	VCP2Handle VCP2_init(VCP2Obj* pVcp2Obj, int instNum, int* pStatus);	New API. This API MUST be called prior to using any other VCP2 APIs to modify/retrieve VCP2 device instance. A valid instance number and VCP2 Object handle must be passed to this API. This API initializes the device instance corresponding to the instance number specified and reserves it for further use. On successful initialization, this API returns a VCP2 instance handle that MUST be used where necessary with VCP2 APIs to indicate

		the VCP2 device instance info.
N/A	CSL_Status VCP2_Close (VCP2Handle hVcp2);	New API. This API MUST be called to unreserve the VCP2 device instance specified by the handle passed. A VCP2_init must have been called prior to calling this API to obtain a valid VCP2 instance handle.

N/A	CSL_IDEF_INLINE
	void VCP2_getPeripheralID
	(
	    VCP2Handle hVcp2,
	    VCP2_PID* hPid
	);
		New API. This API retrieves the VCP2 Peripheral ID and Revision Number for the VCP2 peripheral instance specified.

CSL_IDEF_INLINE Uint32 VCP2_getBmEndian (
    void
);
	CSL_IDEF_INLINE
	Uint32 VCP2_getBmEndian (
	    VCP2Handle hVcp2
	);
		API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE
void VCP2_getIcConfig
(
    VCP2_ConfigIc
*configIc
);
	CSL_IDEF_INLINE
	void VCP2_getIcConfig (
	    VCP2Handle hVcp2,
	    VCP2_ConfigIc*
	configIc
	);
		API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE
Uint32
VCP2_getNumInFifo (
    void
);
	CSL_IDEF_INLINE
	Uint32 VCP2_getNumInFifo (
	    VCP2Handle hVcp2
	);
		API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE
Uint32
VCP2_getNumOutFifo (
    void
);
	CSL_IDEF_INLINE
	Uint32 VCP2_getNumOutFifo
	(
	    VCP2Handle hVcp2
	);
		API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE
Uint32
VCP2_getSdEndian (
    void
);
	CSL_IDEF_INLINE
	Uint32 VCP2_getSdEndian (
	    VCP2Handle hVcp2
	);
		API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE
Uint8
VCP2_getStateIndex (
    void
);
	CSL_IDEF_INLINE
	Uint8 VCP2_getStateIndex (
	    VCP2Handle hVcp2
	);
		API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE
Uint8 VCP2_getYamBit (
    void
);
	CSL_IDEF_INLINE
	Uint8 VCP2_getYamBit (
	    VCP2Handle hVcp2
	);
		API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE
Int16 VCP2_getMaxSm (
    void
	CSL_IDEF_INLINE
	Int16 VCP2_getMaxSm (
	    VCP2Handle hVcp2
		API has been modified to take an argument to indicate the VCP2 device

| ```
);
``` | ```
);
``` | instance on which API needs to act. |
| ```
CSL_IDEF_INLINE
Int16 VCP2_getMinSm (
    void
);
``` | ```
CSL_IDEF_INLINE
Int16 VCP2_getMinSm (
    VCP2Handle    hVcp2
);
``` | API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act. |
| ```
CSL_IDEF_INLINE
void VCP2_icConfig (
    VCP2_ConfigIc
*vcpConfigIc
);
``` | ```
CSL_IDEF_INLINE
void VCP2_icConfig (
    VCP2Handle    hVcp2,
    VCP2_ConfigIc
*vcpConfigIc
);
``` | API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act. |
| ```
CSL_IDEF_INLINE
void VCP2_icConfigArgs
(
    Uint32     ic0,
    Uint32     ic1,
    Uint32     ic2,
    Uint32     ic3,
    Uint32     ic4,
    Uint32     ic5
);
``` | ```
CSL_IDEF_INLINE
void VCP2_icConfigArgs (
    VCP2Handle  hVcp2,
    Uint32      ic0,
    Uint32      ic1,
    Uint32      ic2,
    Uint32      ic3,
    Uint32      ic4,
    Uint32      ic5
);
``` | API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act. |
| ```
CSL_IDEF_INLINE
void VCP2_setBmEndian
(
    Uint32       bmEnd
);
``` | ```
CSL_IDEF_INLINE
void VCP2_setBmEndian (
    VCP2Handle  hVcp2,
    Uint32      bmEnd
);
``` | API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act. |
| ```
CSL_IDEF_INLINE
void
VCP2_setNativeEndian (
    void
);
``` | ```
CSL_IDEF_INLINE
void VCP2_setNativeEndian
(
    VCP2Handle   hVcp2
);
``` | API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act. |
| ```
CSL_IDEF_INLINE
void
VCP2_setPacked32Endian
(
    void
);
``` | ```
CSL_IDEF_INLINE
void
VCP2_setPacked32Endian (
    VCP2Handle   hVcp2
);
``` | API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act. |
| ```
CSL_IDEF_INLINE
void VCP2_setSdEndian
(
    Uint32       sdEnd
);
``` | ```
CSL_IDEF_INLINE
void VCP2_setSdEndian (
    VCP2Handle  hVcp2,
    Uint32      sdEnd
);
``` | API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act. |
| ```
CSL_IDEF_INLINE
Bool VCP2_statError (
    void
);
``` | ```
CSL_IDEF_INLINE
Bool VCP2_statError (
    VCP2Handle   hVcp2
);
``` | API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act. |
| ```
CSL_IDEF_INLINE
Uint32 VCP2_statInFifo
(
    void
);
``` | ```
CSL_IDEF_INLINE
Uint32 VCP2_statInFifo (
    VCP2Handle   hVcp2
);
``` | API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act. |
| ```
CSL_IDEF_INLINE
Uint32
``` | ```
CSL_IDEF_INLINE
Uint32 VCP2_statOutFifo (
``` | API has been modified to take an argument to |

VCP2_statOutFifo (void);	VCP2Handle hVcp2);	indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE
Uint32 VCP2_statPause
(
    void
);
	CSL_IDEF_INLINE
	Uint32 VCP2_statPause (
	    VCP2Handle hVcp2
	);
		API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE
Uint32 VCP2_statRun (
    void
);
	CSL_IDEF_INLINE
	Uint32 VCP2_statRun (
	    VCP2Handle hVcp2
	);
		API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE
Uint32
VCP2_statSymProc (
    void
);
	CSL_IDEF_INLINE
	Uint32 VCP2_statSymProc (
	    VCP2Handle hVcp2
	);
		API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE
Uint32 VCP2_statWaitIc
(
    void
);
	CSL_IDEF_INLINE
	Uint32 VCP2_statWaitIc (
	    VCP2Handle hVcp2
	);
		API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE
void VCP2_start (
    void
);
	CSL_IDEF_INLINE
	void VCP2_start (
	    VCP2Handle hVcp2
	);
		API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE
void VCP2_pause (
    void
);
	CSL_IDEF_INLINE
	void VCP2_pause (
	    VCP2Handle hVcp2
	);
		API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE
void VCP2_unpause (
    void
);
	CSL_IDEF_INLINE
	void VCP2_unpause (
	    VCP2Handle hVcp2
	);
		API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE
void
VCP2_stepTraceback (
    void
);
	CSL_IDEF_INLINE
	void VCP2_stepTraceback (
	    VCP2Handle hVcp2
	);
		API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE
void VCP2_reset (
    void
);
	CSL_IDEF_INLINE
	void VCP2_reset (
	    VCP2Handle hVcp2
	);
		API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE
void VCP2_getErrors (
    VCP2_Errors
*pVcpErr
);
	CSL_IDEF_INLINE
	void VCP2_getErrors (
	    VCP2Handle hVcp2,
	    VCP2_Errors *pVcpErr
	);
		API has been modified to take an argument to indicate the VCP2 device instance on which API needs to act.

CSL_IDEF_INLINE Uint32	CSL_IDEF_INLINE Uint32 VCP2_statEmuHalt (	API has been modified to take an argument to

| VCP2_statEmuHalt (
    void
); | VCP2Handle hVcp2
); | indicate the VCP2 device
instance on which API
needs to act. |
| CSL_IDEF_INLINE
void VCP2_emuDisable (
    void
); | CSL_IDEF_INLINE
void VCP2_emuDisable (
    VCP2Handle hVcp2
); | API has been modified to
take an argument to
indicate the VCP2 device
instance on which API
needs to act. |

5.6 IDMA

The CSL IDMA Functional Layer from the legacy code was analyzed and the following issues were seen:-

1. **Global Variables.**
   The IDMA CSL previously remembered various data transfer settings such as priority of transfer, interrupt enable settings on a transfer and would use in the corresponding transfer APIs. This although simplified the actual transfer APIs by not having to specify these settings per transfer, introduced a critical section around writes to these global variables in the FL. If the CSL FL API were to be called from an Application Task and ISR context, the global variables now would have to be protected using some sort of synchronization technique which could introduce additional latency. To keep the APIs simple and to avoid any additional latency in the data path due to the same, these global variables have been eliminated.

2. **Enable and Disable Interrupts:**
   The IDMA CSL API used system wide interrupt enables/disables to protect data transfers. This could have an impact on system interrupt latency. Developers can determine if interrupts need to be disabled and if required do so before calling the API.

3. **Compile Time Flags:**
   The IDMA CSL FL from earlier releases used a compile time flag "PEND_SUPPORT" that when enabled would make the IDMA channel 0, 1 APIs wait till any pending data transfers on that channel to complete before a new transfer was initiated. This now has been eliminated and instead a new argument to the appropriate APIs has been added to add the same support at run time.

Based on the issues discussed above, the IDMA CSL FL APIs have been modified as follows:

Old API	New API	Comments

Int IDMA1_init (
    IDMA_priSet
priority,
    IDMA_intEn interr
);
	N/A	This API has been eliminated
		from the current drop. This
		API was earlier used to
		configure some global
		variables that would remember

		the IDMA Channel 1 transfer settings like priority, and interrupt enable settings. For reasons discussed above, this API has been eliminated and instead these settings would now have to be specified per transfer.
Int IDMA1_setPriority (IDMA_priSet priority);	N/A	This API has been eliminated from the current drop. This API was earlier used to configure a global variable that would remember the IDMA Channel 1 transfer priority. For reasons discussed above, this API has been eliminated and instead this setting would now have to be specified per transfer.
Int IDMA1_setInt (IDMA_intEn interr);	N/A	This API has been eliminated from the current drop. This API was earlier used to configure a global variable that would remember the IDMA Channel 1 transfer interrupt settings. For reasons discussed above, this API has been eliminated and instead this setting would now have to be specified per transfer.

Int IDMA1_copy (
    Uint32   *src,
    Uint32   *dst,
    Uint32
byteCnt
);
	CSL_IDEF_INLINE void CSL_IDMA_chan1TransferData (
	    CSL_IDMA_IDMA1CONFIG*
	idmaChan1Config,
	    Uint32
	bWaitToCompletion
	);
		The new API definition now takes two parameters. The first parameter expected is the IDMA1 Configuration structure with the source, destination, byte count, priority, interrupt settings filled. The second parameter indicates if this API should wait for any pending/active transfer to be completed before it starts the transfer and also wait for the transfer to be completed before it returns. Also, as mentioned earlier now the application would have to disable/enable interrupts if required before calling this API.

Int IDMA1_fill (
    Uint32   *dst,
    Uint32
byteCnt,
    Uint32
fill_value
);
	CSL_IDEF_INLINE void CSL_IDMA_chan1FillData (
	    CSL_IDMA_IDMA1CONFIG*
	idmaChan1Config,
	    Uint32
	bWaitToCompletion
	);
		The new API definition now takes two parameters. The first parameter expected is the IDMA1 Configuration structure with the source, destination, byte count, priority, interrupt settings

		filled. The second parameter indicates if this API should wait for any pending/active transfer to be completed before it starts the transfer and also wait for the transfer to be completed before it returns. Also, as mentioned earlier now the application would have to disable/enable interrupts if required before calling this API.
Uint32 IDMA1_getStatus(void);	CSL_IDEF_INLINE void CSL_IDMA_chan1GetStatus (CSL_IDMA_STATUS* idmaChanStatus);	The function definition has been changed from the earlier releases. Previously, the status bits were masked and returned as a single 32-bit integer. The application would then have to read the 2 LSB bits to determine the status. The new API has been redefined to simplify things for the application. The new API definition now takes a valid pointer to IDMA Status structure. The API returns the status as individual fields in the data structure pointer passed to this function.
void IDMA1_wait (void);	CSL_IDEF_INLINE void CSL_IDMA_chan1Wait (void);	The function has been renamed to keep up with the CSL standard naming conventions. The functionality remains the same.
Int IDMA0_init (IDMA_intEn interr);	N/A	This API has been eliminated from the current drop. This API was earlier used to configure a global variable that would remember the IDMA Channel 0 transfer interrupt enable settings. For reasons discussed earlier, this API has been eliminated and instead this setting would now have to be specified per transfer.
void IDMA0_config (IDMA0_Config *config);	CSL_IDEF_INLINE void CSL_IDMA_chan0TransferData (CSL_IDMA_IDMA0CONFIG* idmaChan0Config, Uint32 bWaitToCompletion);	The old API has been renamed and definition has been changed to take two new parameters. The first parameter expected is the IDMA0 Configuration structure with the source, destination, mask, window count, interrupt settings filled. The second

Texas Instruments Incorporated   Statement of Work
QRSA # 13320
Revision D   100_CSLC6498

		parameter indicates if this API should wait for any pending/active transfer to be completed before it starts the transfer and also wait for the transfer to be completed before it returns. Also, as mentioned earlier now the application would have to disable/enable interrupts if required before calling this API.

`void IDMA0_configArgs (`
`    Uint32      mask,`
`    Uint32      *src,`
`    Uint32      *dst,`
`    Uint32      count`
`);`
	N/A	This API has been eliminated. The same functionality has been captured in the previous API, CSL_IDMA_chan0TransferData();

`Uint32`
`IDMA0_getStatus(void);`
	`CSL_IDEF_INLINE void`
	`CSL_IDMA_chan0GetStatus (`
	`        CSL_IDMA_STATUS*`
	`idmaChanStatus`
	`);`
		The function definition has been changed from the earlier releases. Previously, the status bits were masked and returned as a single 32-bit integer. The application would then have to read the 2 LSB bits to determine the status. The new API has been redefined to simplify things for the application. The new API definition now takes a valid pointer to IDMA Status structure. The API returns the status as individual fields in the data structure pointer passed to this function.

`void IDMA0_wait(void);`	`CSL_IDEF_INLINE void`
	`CSL_IDMA_chan0Wait (`
	`        void`
	`);`
		The function has been renamed to keep up with the CSL standard naming conventions. The functionality remains the same.

`Int IDMA0_setInt (`
`    IDMA_intEn interr`
`);`
	N/A	This API has been eliminated from the current drop. This API was earlier used to configure a global variable that would remember the IDMA Channel 0 transfer interrupt settings. For reasons discussed earlier, this API has been eliminated and instead this setting would now have to be specified per transfer.

The following new data structure definitions have been added to enable IDMA transfer configuration:

```
/** @brief     This structure holds the information required
 *          to initiate a IDMA Channel 1 Block Fill/Transfer
 *          request in the GEM.
 */
typedef struct {

  /** @brief  IDMA channel 1 Source Address.
   *
   *  @Description
   *  The source address must point to a word-aligned
   *  memory location local to GEM. When performing a
   *  block fill, all 32 bits of the address specified
   *  are considered; While, for block transfers, the
   *  2 LSBs are ignored and the higher order 30 bits
   *  are read as the valid Source Address for transfer.
   */
  Uint32*     source;

  /** @brief  IDMA channel 1 Destination Address.
   *
   *  @Description
   *  The destination address must point to a 32 bit
   *  word-aligned memory location local to GEM.
   *  This address must be local to GEM, either in L1P,
   *  L1D, L2 or CFG and also must be different port
   *  than the source address to obtain full throughput.
   */
  Uint32*     destn;

  /** @brief  Number of bytes to be transfered
   *
   *  @Description
   *  The count signifies the number of bytes to be
   *  transferred using IDMA channel 1. This must be
   *  a multiple of 4 bytes. A count of zero will not
   *  transfer any data, but will generate an interrupt
   *  if requested.
   */
  Uint16     count;

  /** @brief  Transfer Priority.
```

```
 *
 * @Description
 * The transfer priority is used for arbitration between
 * the CPU and DMA accesses when there are conflicts.
 * Valid values for the priority range between 0 and 7.
 */
Uint32     priority:3;

/** @brief  Boolean Flag to enable/disable CPU interrupt.
 *
 * @Description
 * When this interrupt flag is set, a CPU Interrupt IDMA_INT1
 * is raised on completion of the block transfer/fill request.
 */
Uint32     intEnable:1;

}CSL_IDMA_IDMA1CONFIG;


/** @brief     This structure holds the information required
 *          to initiate a IDMA Channel 0 Configuration(CFG) space
 *          Transfer request from the GEM.
 */
typedef struct {
  /** @brief  IDMA channel 0 Mask.
   *
   * @Description
   * The mask allows unwanted registers within the window
   * to be blocked from access, facilitating multiple read/write
   * transactions to be completed with a single transfer command
   * by the CPU. Each of the 32 bits of the mask correspond to
   * a single register in the CFG space identified by the source/
   * destination address registers.
   */
  Uint32     mask;

  /** @brief  IDMA channel 0 Source Address.
   *
   * @Description
   * The source address must point to a 32-byte-aligned
   * memory location local to GEM or to a valid configuration
   * register space.
   */
  Uint32*     source;
```

```
    /** @brief  IDMA channel 0 Destination Address.
     *
     * @Description
     * The destination address must point to a 32-byte
     * -aligned memory location local to GEM or to a valid
     * configuration register space.
     */
    Uint32*     destn;


    /** @brief  Number of 32-word windows to be transfered
     *
     * @Description
     * The count signifies the number of windows to be accessed
     * during data transfer. Upto 16 contiguous 32-word regions
     * can be specified using this field.
     */
    Uint32      count:4;


    /** @brief  Boolean Flag to enable/disable CPU interrupt.
     *
     * @Description
     * When this interrupt flag is set, a CPU Interrupt IDMA_INT0
     * is raised on completion of the block transfer/fill request.
     */
    Uint32      intEnable:1;


}CSL_IDMA_IDMA0CONFIG;




/** @brief     This structure holds the information required
 *          to interpret the IDMA Channel 0/1 Transfer
 *          Status.
 */
typedef struct {
    /** @brief  Boolean Flag that indicates if any pending transfers
     *          exist on the iDMA channel 1/channel 0.
     *
     * @Description
     * Set when control registers are written to by the CPU and there
     * is already an active transfer in progress (ACTV=1) and cleared
     * when the pending transfer becomes active.
```

```
    */
  Uint32     isPending:1;

  /** @brief  Boolean Flag that indicates if any active transfers
   *          exist on the iDMA channel 1/channel 0.
   *
   *  @Description
   *  Set when channel 0/1 begins reading data from the source address
   *  and cleared following the write to the destination address.
   */
  Uint32     isActive:1;
}CSL_IDMA_STATUS;
```

5.7   CHIP / TSC

The Chip Module in CSL package provides programmatic access to the C64x+ CPU control registers. The Time Stamp Counter (TSC) module in CSL on the other hand provides APIs to just enable and read the C64x+ CPU's TSC registers. The legacy code for these modules was written in assembly language which had the following problems:

1. Portability:

Code written generally in assembly language is not easily portable across different platforms or achitectures.

2.   **Ease of Use:**

Code written in high level languages like "C" are easily understood, and debugged.

3.   **Optimizations:**

The compiler takes care of translating the "C" code to the most appropriate series of assembly instructions without any errors as per the compiler optimization options provided for the application. However, when code is written in plain assembly language, the onus is now on the programmer to understand all the nitty-gritty's of optimization and appropriately write the code such that there are no errors eventually when parsed by the assembler. This is quite difficult and could lead to portability issues again, since now you'll have to write different versions of the same program for different level of optimizations.

There are three possible solutions to circumvent the issues discussed above:

1. **Turn off Optimizations/Compression**:

The easiest solution would be to use assembler / compiler directives to turn off any optimizations that could lead to portability issues.

For example, the C64x+ processor typically processes 32-bit instructions. However, to achieve size optimization, the compiler can "*compress*" code into using 16-bit instructions. When done so, the compiler generated optimized assembly code could have some additional instructions that could mess up any branches in the original code. This can be turned off in assembly code using ".*nocmp*" assembler directive or by using "—no_compress" option with the compiler keeping the code sane.

However, turning off such optimizations at application level might not be suitable for the application from performance perspective and hence is not the best way to address the issue.

2. **Use Inline assembly from C**

The complex assembly code in these modules with the "*branch*" instructions, offset calculations etc could be replaced with a C function which in turn uses the "*asm*" directive to read/write from the registers.

However, the disadvantage of this approach is that the compiler doesn't check the contents of the "*asm*" statements and hence any A-side / B-side registers used for accessing arguments to the function or setting return values from this function could be overwritten by the compiler generated code leading to inconsistent code.

3. **Use C code**

The C6000 C/C++ Compiler provides a "*cregister*" keyword and definitions for all the C64x+ CPU control registers to access them from C functions.

Example:

```
extern cregister volatile unsigned int AMR;
extern cregister volatile unsigned int CSR;
.
.
.
```

The CSL-FL code for Chip and TSC modules have been rewritten to use these definitions provided by the compiler. This simplifies the code and provides the much needed portability to the CSL-FL.

Based on the above discussion, the CSL-FL APIs for **Chip** module have been modified as following:

Old API	New API	Comments

Uint32
CSL_chipReadReg(
   CSL_ChipReg
reg
);
	Uint32 CSL_chipReadReg(
	   CSL_ChipReg reg
	);
		The API prototype / arguments have not been changed. However, this function was earlier written in pure assembly code. This API implementation now has been modified to be written in pure C language and to use C6000 C/C++ compiler's control register definitions to read the registers directly.
		
		However, this function uses a "switch" statement to branch to the appropriate portion of code to read the register specified as argument to this function. Hence, this API might not be suitable to applications that might invoke it from their data-path. If so, alternate direct "Read" function for each of the registers that are Readable have been provided (as listed below in this table) that can be called without taking a performance hit.

Uint32 CSL_chipWriteReg (
   CSL_ChipReg
reg,
   CSL_Reg32
val
	Uint32 CSL_chipWriteReg (
	   CSL_ChipReg reg,
	   CSL_Reg32 val
	);
		The API prototype / arguments have not been changed. However, this function was earlier written in pure assembly code. This API

|); | | implementation now has been modified to be written in pure C language and to use C6000 C/C++ compiler's control register definitions to write to the registers directly.

However, this function uses a "switch" statement to branch to the appropriate portion of code to modify the register specified as argument to this function. Hence, this API might not be suitable to applications that might invoke it from their data-path. If so, alternate direct "Write" function for each of the registers that are Writeable have been provided (as listed below in this table) that can be called without taking a performance hit. |
|---|---|---|
| N/A | Uint32 CSL_chipReadAMR() | This API reads the Addressing Mode (AMR) control register directly (using C6000 C/C++ compiler's control register definitions) and returns its contents. |
| N/A | Uint32 CSL_chipReadCSR() | This API reads the Control Status (CSR) register (using C6000 C/C++ compiler's control register definitions) and returns its contents. |
| N/A | Uint32 CSL_chipReadIFR() | This API reads the Interrrupt Flag register |

		(IFR) (using C6000 C/C++ compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipReadIER()	This API reads the Interrrupt Enable register (IER) (using C6000 C/C++ compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipReadISTP()	This API reads the Interrrupt Service Table Pointer register (ISTP) (using C6000 C/C++ compiler's control register definitions) and returns its contents
N/A	Uint32 CSL_chipReadIRP()	This API reads the Interrrupt Return Pointer register (IRP) (using C6000 C/C++ compiler's control register definitions) and returns its contents
N/A	Uint32 CSL_chipReadNRP()	This API reads the Nonmaskable Interrupt Return Pointer register (NRP) (using C6000 C/C++ compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipReadERP()	This API reads the Exception Return Pointer register (ERP) (using C6000 C/C++ compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipReadTSCL()	This API reads the Time Stamp Counter Lower Order 32-bits (TSCL) (using C6000 C/C++ compiler's control register

		definitions) register and returns its contents
N/A	Uint32 CSL_chipReadTSCH()	This API reads the Time Stamp Counter Higher Order 32-bits (TSCH) (using C6000 C/C++ compiler's control register definitions) register and returns its contents.
N/A	Uint32 CSL_chipReadARP()	This API reads the Analysis Return Pointer (ARP) (using C6000 C/C++ compiler's control register definitions) register and returns its contents.
N/A	Uint32 CSL_chipReadILC()	This API reads the Inner Loop SPL Buffer Count (ILC) register (using C6000 C/C++ compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipReadRILC()	This API reads the Reload Inner Loop SPL Buffer Count (RILC) register (using C6000 C/C++ compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipReadREP()	This API reads the Restricted Entry Point Address (REP) register (using C6000 C/C++ compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipReadPCE1()	This API reads the Program Counter, E1 Phase (PCE1) register (using C6000 C/C++ compiler's control register definitions) and returns its

		contents.
N/A	Uint32 CSL_chipReadDNUM()	This API reads the DSP Core Number (DNUM) register (using C6000 C/C++ compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipReadSSR()	This API reads the Saturation Status Register (SSR) (using C6000 C/C++ compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipReadGPLYA()	This API reads the GMPY A-side polynomial Register (GPLYA) (using C6000 C/C++ compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipReadGPLYB()	This API reads the GMPY B-side polynomial Register (GPLYB) (using C6000 C/C++ compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipReadGFPGFR()	This API reads the Galios Field Multiply Control Register (GFPGFR) (using C6000 C/C++ compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipReadDIER()	This API reads the Debug Interrupt Enable Register (DIER) (using C6000 C/C++ compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipReadTSR()	This API reads the Task State Register (TSR) (using C6000 C/C++

		compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipReadITSR()	This API reads the Interrupt Task State Register (ITSR) (using C6000 C/C++ compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipReadNTSR()	This API reads the NMI/Exception Task State Register (NTSR) (using C6000 C/C++ compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipReadEFR()	This API reads the Exception Flag Register (EFR) (using C6000 C/C++ compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipReadIERR()	This API reads the Internal Exception Report Register (IERR) (using C6000 C/C++ compiler's control register definitions) and returns its contents.
N/A	Uint32 CSL_chipWriteAMR(CSL_Reg32 val)	This API configures the Addressing Mode Register (AMR) (using C6000 C/C++ compiler's control register definitions) with the specified new value 'val'. This API returns the old AMR value on return.
N/A	Uint32 CSL_chipWriteCSR(CSL_Reg32 val)	This API configures the Control Status Register (CSR) (using C6000 C/C++ compiler's control register definitions) with

		the specified new value 'val'. This API returns the old CSR value on return.
N/A	Uint32 CSL_chipWriteISR(CSL_Reg32 val)	This API configures the Interrupt Set Register (ISR) (using C6000 C/C++ compiler's control register definitions) with the specified new value 'val'. This API returns the old ISR value on return.
N/A	Uint32 CSL_chipWriteICR(CSL_Reg32 val)	This API writes to the Interrupt Clear Register (ICR) (using C6000 C/C++ compiler's control register definitions) with the specified new value 'val' to clear the maskable interrupts configured in the IFR register. This API returns the old interrupt state from IFR value on return.
N/A	Uint32 CSL_chipWriteIER(CSL_Reg32 val)	This API writes to the Interrupt Enable Register (IER) (using C6000 C/C++ compiler's control register definitions) with the specified new value 'val' to enable/disable individual interrupts in the system. This API returns the old interrupt enable state from IER value on return.
N/A	Uint32 CSL_chipWriteISTP(CSL_Reg32 val)	This API writes to the Interrupt Service Table Pointer Register (ISTP) (using C6000 C/C++ compiler's control register definitions) with the specified value 'val' to setup a ISR base address

		for a given interrupt in the ISTB. This API returns the old ISTP value on return.
N/A	Uint32 CSL_chipWriteIRP(CSL_Reg32 val)	This API writes to the Interrupt Return Pointer Register (IRP) (using C6000 C/C++ compiler's control register definitions) with the specified value 'val' to setup a return point in the program to continue execution after a maskable interrupt is executed. This API returns the old IRP value on return.
N/A	Uint32 CSL_chipWriteNRP(CSL_Reg32 val)	This API writes to the NMI Return Pointer Register (NRP) (using C6000 C/C++ compiler's control register definitions) with the specified value 'val' to setup a return point in the program to continue execution after an NMI has occured. This API returns the old NRP value on return.
N/A	Uint32 CSL_chipWriteERP(CSL_Reg32 val)	This API writes to the Exception Return Pointer Register (ERP) (using C6000 C/C++ compiler's control register definitions) with the specified value 'val' to setup a return point in the program to continue execution after an exception has occured. This API returns the old ERP value on return.

N/A	Uint32 CSL_chipWriteTSCL(CSL_Reg32 val)	This API writes to the Time-Stamp Counter (low order 32 bits) Register (TSCL) (using C6000 C/C++ compiler's control register definitions) with the specified value 'val' to enable the TSC. This API returns the old TSCL value on return. The actual value written "val" to TSCL has no effect on the Time stamp counter. The value is simply ignored by the hardware, and a write to TSCL is used only to enable the Time Stamp Counter.
N/A	Uint32 CSL_chipWriteARP(CSL_Reg32 val)	This API writes to the Analysis Return Pointer (ARP) (using C6000 C/C++ compiler's control register definitions) with the specified value 'val'. This API returns the old ARP value on return.
N/A	Uint32 CSL_chipWriteILC(CSL_Reg32 val)	This API writes to the SPLOOP Inner Loop Count Register (ILC) (using C6000 C/C++ compiler's control register definitions) with the specified value 'val'. This API returns the old ILC value on return.
N/A	Uint32 CSL_chipWriteRILC(CSL_Reg32 val)	This API writes to the SPLOOP Reload Inner Loop Count Register (RILC) (using C6000 C/C++ compiler's control register definitions) with the specified value 'val'. This API returns the old

		RILC value on return.
N/A	Uint32 CSL_chipWriteREP(CSL_Reg32 val)	This API writes to the Restricted Entry-Point Register (REP) (using C6000 C/C++ compiler's control register definitions) with the specified value 'val'. This API returns the old REP value on return.
N/A	Uint32 CSL_chipWriteSSR(CSL_Reg32 val)	This API writes to the Saturation Status Register (SSR) (using C6000 C/C++ compiler's control register definitions) with the specified value 'val'. This API returns the old SSR value on return.
N/A	Uint32 CSL_chipWriteGPLYA(CSL_Reg32 val)	This API writes to the GMPY Polynomial-A side Register (GPLYA) (using C6000 C/C++ compiler's control register definitions) with the specified value 'val'. This API returns the old GPLYA value on return.
N/A	Uint32 CSL_chipWriteGPLYB(CSL_Reg32 val)	This API writes to the GMPY Polynomial-B side Register (GPLYB) (using C6000 C/C++ compiler's control register definitions) with the specified value 'val'. This API returns the old GPLYB value on return.
N/A	Uint32 CSL_chipWriteGFPGFR(CSL_Reg32 val)	This API writes to the Galios Field Polynomial Generator Function Register (GFPGFR) (using C6000 C/C++ compiler's control register definitions) with the

		specified value 'val'. This API returns the old GFPGFR value on return.
N/A	Uint32 CSL_chipWriteDIER(CSL_Reg32 val)	This API writes to the Debug Interrupt Enable Register (DIER) (using C6000 C/C++ compiler's control register definitions) with the specified value 'val'. This API returns the old DIER value on return.
N/A	Uint32 CSL_chipWriteTSR(CSL_Reg32 val)	This API writes to the Task State Register (TSR) (using C6000 C/C++ compiler's control register definitions) with the specified value 'val'. This API returns the old TSR value on return.
N/A	Uint32 CSL_chipWriteITSR(CSL_Reg32 val)	This API writes to the Interrupt Task State Register (ITSR) (using C6000 C/C++ compiler's control register definitions) with the specified value 'val'. This API returns the old ITSR value on return.
N/A	Uint32 CSL_chipWriteNTSR(CSL_Reg32 val)	This API writes to the NMI/Exception Task State Register (NTSR) (using C6000 C/C++ compiler's control register definitions) with the specified value 'val'. This API returns the old NTSR value on return.
N/A	Uint32 CSL_chipWriteECR(CSL_Reg32 val)	This API writes to the Exception Clear Register (ECR) (using C6000 C/C++ compiler's control register definitions) with

		the specified value 'val' to clear the corresponding bits in the EFR. This API returns the old EFR value on return.
N/A	Uint32 CSL_chipWriteIERR(CSL_Reg32 val)	This API writes to the Internal Exception Report Register (IERR) (using C6000 C/C++ compiler's control register definitions) with the specified value 'val'. This API returns the old IERR value on return.

Similarly the **TSC** module APIs have been modified as following:

Old API	New API	Comments
void CSL_tscEnable(void)	void CSL_tscEnable(void)	No API prototype change. The implementation of this API however has been changed from assembly to C to make it portable. This API now simply writes "0" to TSCL register (using C6000 C/C++ compiler's control register definitions) to initialize the C64x+ CPU timer.
CSL_Uint64 CSL_tscRead(void)	CSL_Uint64 CSL_tscRead(void)	No API prototype change. The implementation of this API however has been changed from assembly to C to make it portable. This API now reads the TSCL and TSCH registers (using C6000 C/C++ compiler's control register definitions) and concatenates the two 32-bit vales obtained from them into a 64-bit unsigned value and returns it.

5.8 GPIO

The CSL GPIO Functional Layer from the legacy code was analyzed and the following issues were seen:

1. **Scalability**

   The legacy implementation wasn't scalable to support multiple GPIO banks. It supported only 2 GPIO banks.

2. **Overhead**

   In the legacy implementation there was only one API was provided to set the directionality and rising and falling edge triggers of a pin. This introduces an overhead where checks are done to analyze the trigger value in order to figure out which registers have to be configured.

   Also it did not allow a user to set directionality only and not change the triggers. The new APIs are atomic in nature.

   When reading output data on the GPIO pins, the legacy APIs returned the entire 32 bit value corresponding to all GPIO pins. The user would then have to compute the bit-value of the pin required. The new API returns only the output drive state of GPIO pin specified as the input parameter.

Taking into account the 3X style cons mentioned above; the CSL Functional layer for the GPIO module will be implemented as per the AUX style.

The following table summarizes the changes to the CSL GPIO API:

Old API	New API	Comments
CSL_gpioInit	N/A	Obsolete
CSL_gpioOpen	CSL_GPIO_open	This API opens the GPIO instance. This must be the first call to the CSL GPIO Functional Layer. The return handle should be passed as an input parameter to all subsequent CSL API's
CSL_gpioHwControl Cmd= CSL_GPIO_CMD_BANK_INT_ENABLE	CSL_GPIO_bankInterruptEnable	Enables the GPIO per bank interrupt.
CSL_gpioHwControl Cmd= CSL_GPIO_CMD_BANK_INT_DISABLE	CSL_GPIO_bankInterruptDisable	Disables the GPIO per bank interrupt.
CSL_gpioHwControl Cmd= CSL_GPIO_CMD_CONFIG_BIT	CSL_GPIO_setPinDirOutput CSL_GPIO_setPinDirInput CSL_GPIO_setRisingEdgeDetect CSL_GPIO_clearRisingEdgeDetect CSL_GPIO_setFallingEdgeDetect CSL_GPIO_clearFallingEdgeDetect	The old API sets the directionality, rising and falling edge triggers. The new APIs are atomic.
CSL_gpioHwControl Cmd= CSL_GPIO_CMD_SET_BIT	CSL_GPIO_setOutputData	Sets the output drive state of GPIO pin.
CSL_gpioHwControl Cmd= CSL_GPIO_CMD_CLEAR_BIT	CSL_GPIO_clearOutputData	Clears the output drive state of GPIO pin.
CSL_gpioHwControl Cmd= CSL_GPIO_CMD_GET_INPUTBIT	CSL_GPIO_getInputData	The old API returned the entire 32 bit value. The new API returns only the input bit data on GPIO pin specified in the input parameter.
CSL_gpioHwControl Cmd= CSL_GPIO_CMD_GET_OUTDRVSTATE	CSL_GPIO_getOutputData	The old API returned the entire 32 bit value. The new API returns only the output drive state of GPIO pin specified in the input parameter.
CSL_gpioHwControl Cmd= CSL_GPIO_CMD_GET_BIT	CSL_GPIO_getInputData	
CSL_gpioHwControl Cmd=CSL_GPIO_CMD_SET_OUT_BIT	CSL_GPIO_setOutputData	Sets the output drive state of GPIO pin.
CSL_gpioGetHwStatus Cmd= CSL_GPIO_QUERY_BINTEN_STAT	CSL_GPIO_isBankInterruptEnabled	Returns the status of GPIO per bank interrupt. Each bank supports 16 GPIO signals.
N/A	CSL_GPIO_getPID	Returns the

		peripheral ID register which identifies the scheme of PID encoding, function, rtl id, major id, custom id and minor id.
N/A	CSL_GPIO_getPCR	Returns the peripheral Control register value which identifies the emulation mode.
N/A	CSL_GPIO_getPinDirection	Returns the direction configuration of GPIO pin specified in the input parameter.
N/A	CSL_GPIO_isRisingEdgeDetect	This function checks if the interrupt detection for GPIO pin is set to rising edge or not.
N/A	CSL_GPIO_isFallingEdgeDetect	This function checks if the interrupt detection for GPIO pin is set to falling edge or not.
N/A	CSL_GPIO_getInterruptStatus	This function returns the GPIO pin interrupt status
N/A	CSL_GPIO_clearInterruptStatus	This function clears the GPIO pin interrupt status.
CSL_gpioHwSetupRaw	N/A	New APIs listed above replace the legacy API.
CSL_gpioGetHwSetup	N/A	Legacy API was an empty function call.
CSL_gpioHwSetup	N/A	Legacy API was an empty function call.
CSL_gpioClose	N/A	Obsolete

5.9 RAC

The CSL RAC Functional Layer APIs have been extended to support multiple RAC peripheral instance configuration.

The RAC CSL-FL APIs have been modified as shown below:

Old API	New API	Comments
void CSL_RAC_BETI_enable ();	void CSL_RAC_BETI_enable (CSL_RAC_Handle hRac);	Added a new argument RAC

		instance handle to indicate which RAC instance to use for the operation.
`void CSL_RAC_BETI_disable ();`	`void CSL_RAC_BETI_disable ( CSL_RAC_Handle        hRac );`	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
`CSL_RAC_BETI_statusBit CSL_RAC_BETI_getStatus ();`	`CSL_RAC_BETI_statusBit CSL_RAC_BETI_getStatus ( CSL_RAC_Handle        hRac );`	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
`void CSL_RAC_BETI_setWatchDog ( Uint16 nbCyclesToReload );`	`void CSL_RAC_BETI_setWatchDog ( CSL_RAC_Handle        hRac, Uint16 nbCyclesToReload );`	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
`Uint32 CSL_RAC_BETI_getWatchDogStatus ( Uint8 gccpId ;`	`Uint32 CSL_RAC_BETI_getWatchDogStatus ( CSL_RAC_Handle        hRac, Uint8                 gccpId );`	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
`CSL_RAC_BETI_wdInterruptStatus CSL_RAC_BETI_getWatchDogInterruptStatus ( Uint8 gccpId );`	`CSL_RAC_BETI_wdInterruptStatus CSL_RAC_BETI_getWatchDogInterruptStatus ( CSL_RAC_Handle        hRac, Uint8                 gccpId );`	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
`CSL_RAC_BETI_odbtStatusBit CSL_RAC_BETI_getOdbtStatus ( Uint8 odbtEntryId );`	`CSL_RAC_BETI_odbtStatusBit CSL_RAC_BETI_getOdbtStatus ( CSL_RAC_Handle        hRac, Uint8                 odbtEntryId );`	Added a new argument RAC instance handle to indicate

		which RAC instance to use for the operation.

`void CSL_RAC_BEII_enable ();`	`void CSL_RAC_BEII_enable (`
	`CSL_RAC_Handle          hRac`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`void CSL_RAC_BEII_disable ();`	`void CSL_RAC_BEII_disable (`
	`CSL_RAC_Handle          hRac`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`void CSL_RAC_BEII_setMasterMask (`
`Uint8              cpuId,`
`CSL_RAC_BEII_interrupt`
`cpuIntContrlEnable`
`);`
	`void CSL_RAC_BEII_setMasterMask (`
	`CSL_RAC_Handle          hRac,`
	`Uint8              cpuId,`
	`CSL_RAC_BEII_interrupt`
	`cpuIntContrlEnable`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`void CSL_RAC_BEII_getInterruptStatus (`
`Uint8              cpuId,`
`CSL_RAC_BEII_interruptStatus *`
`cpuIntStatus`
`);`
	`void CSL_RAC_BEII_getInterruptStatus (`
	`CSL_RAC_Handle          hRac,`
	`Uint8              cpuId,`
	`CSL_RAC_BEII_interruptStatus`
	`*cpuIntStatus`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`void CSL_RAC_BEII_setCycleOverflowMask`
`(`
`Uint8              cpuId,`
`Uint8              gccpId,`
`CSL_RAC_BEII_interrupt interruptEnable`
`);`
	`void CSL_RAC_BEII_setCycleOverflowMask`
	`(`
	`CSL_RAC_Handle          hRac,`
	`Uint8              cpuId,`
	`Uint8              gccpId,`
	`CSL_RAC_BEII_interrupt`
	`interruptEnable`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`void CSL_RAC_BEII_setFifoOverflowMask`
`(`
`Uint8              cpuId,`
`Uint8              gccpId,`
`CSL_RAC_BEII_interrupt interruptEnable`
`);`
	`void CSL_RAC_BEII_setFifoOverflowMask`
	`(`
	`CSL_RAC_Handle          hRac,`
	`Uint8              cpuId,`
	`Uint8              gccpId,`
	`CSL_RAC_BEII_interrupt`
	`interruptEnable`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the

		operation.
| ```
 void CSL_RAC_BEII_setSequencerIdleMask
(
Uint8                    cpuId,
Uint8                    gccpId,
CSL_RAC_BEII_interrupt interruptEnable
);
``` | ```
 void CSL_RAC_BEII_setSequencerIdleMask
(
CSL_RAC_Handle              hRac,
Uint8                       cpuId,
Uint8                       gccpId,
CSL_RAC_BEII_interrupt
interruptEnable
);
``` | Added a new argument RAC instance handle to indicate which RAC instance to use for the operation. |
| ```
 void CSL_RAC_BEII_setBeWatchDogMask (
Uint8                    cpuId,
Uint8                    gccpId,
CSL_RAC_BEII_interrupt interruptEnable
);
``` | ```
 void CSL_RAC_BEII_setBeWatchDogMask (
CSL_RAC_Handle              hRac,
Uint8                       cpuId,
Uint8                       gccpId,
CSL_RAC_BEII_interrupt
interruptEnable
);
``` | Added a new argument RAC instance handle to indicate which RAC instance to use for the operation. |
| ```
 void CSL_RAC_BEII_setBetiEotMask (
Uint8                    cpuId,
CSL_RAC_BEII_interrupt interruptEnable
);
``` | ```
 void CSL_RAC_BEII_setBetiEotMask (
CSL_RAC_Handle              hRac,
Uint8                       cpuId,
CSL_RAC_BEII_interrupt
interruptEnable
);
``` | Added a new argument RAC instance handle to indicate which RAC instance to use for the operation. |
| ```
 void
CSL_RAC_BEII_setBetiObbtRdCrossingMask
(
Uint8                    cpuId,
CSL_RAC_BEII_interrupt interruptEnable
);
``` | ```
 void
CSL_RAC_BEII_setBetiObbtRdCrossingMask
(
CSL_RAC_Handle              hRac,
Uint8                       cpuId,
CSL_RAC_BEII_interrupt
interruptEnable
);
``` | Added a new argument RAC instance handle to indicate which RAC instance to use for the operation. |
| ```
 void
CSL_RAC_BEII_setBetiOdbtRdCrossingMask
(
Uint8                    cpuId,
CSL_RAC_BEII_interrupt interruptEnable
);
``` | ```
 void
CSL_RAC_BEII_setBetiOdbtRdCrossingMask
(
CSL_RAC_Handle              hRac,
Uint8                       cpuId,
CSL_RAC_BEII_interrupt
interruptEnable
);
``` | Added a new argument RAC instance handle to indicate which RAC instance to use for the operation. |
| ```
 void CSL_RAC_BEII_setFeWatchDogMask (
Uint8                    cpuId,
CSL_RAC_BEII_interrupt interruptEnable
);
``` | ```
 void CSL_RAC_BEII_setFeWatchDogMask (
CSL_RAC_Handle              hRac,
Uint8                       cpuId,
CSL_RAC_BEII_interrupt
interruptEnable
);
``` | Added a new argument RAC instance handle to indicate which RAC instance to use for the operation. |
| ```
 Uint32
CSL_RAC_BETI_getObbtRdCrossingStatus
``` | ```
 Uint32
CSL_RAC_BETI_getObbtRdCrossingStatus (
``` | Added a new argument |

	CSL_RAC_Handle hRac);	RAC instance handle to indicate which RAC instance to use for the operation.
Uint32 CSL_RAC_BETI_getOdbtRdCrossingStatus ();	Uint32 CSL_RAC_BETI_getOdbtRdCrossingStatus (CSL_RAC_Handle hRac);	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
Uint32 CSL_RAC_BETI_getEotInterruptStatus (Uint8 cpuId);	Uint32 CSL_RAC_BETI_getEotInterruptStatus (CSL_RAC_Handle hRac, Uint8 cpuId);	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
Uint32 CSL_RAC_Stats_getCfgTotalAccess ();	Uint32 CSL_RAC_Stats_getCfgTotalAccess (CSL_RAC_Handle hRac);	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
Uint32 CSL_RAC_Stats_getCfgWriteAccess ();	Uint32 CSL_RAC_Stats_getCfgWriteAccess (CSL_RAC_Handle hRac);	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
Uint32 CSL_RAC_Stats_getCfgReadAccess ();	Uint32 CSL_RAC_Stats_getCfgReadAccess (CSL_RAC_Handle hRac);	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
Uint32 CSL_RAC_Stats_getSlaveTotalAccess ();	Uint32 CSL_RAC_Stats_getSlaveTotalAccess (CSL_RAC_Handle hRac);	Added a new argument RAC instance handle to

		indicate which RAC instance to use for the operation.
`Uint32 CSL_RAC_Stats_getSlaveWriteAccess ();`	`Uint32 CSL_RAC_Stats_getSlaveWriteAccess ( CSL_RAC_Handle          hRac );`	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
`Uint32 CSL_RAC_Stats_getSlaveReadAccess ();`	`Uint32 CSL_RAC_Stats_getSlaveReadAccess ( CSL_RAC_Handle          hRac );`	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
`Uint32 CSL_RAC_Stats_getMasterLowPrioAccess ();`	`Uint32 CSL_RAC_Stats_getMasterLowPrioAccess ( CSL_RAC_Handle          hRac );`	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
`Uint32 CSL_RAC_Stats_getMasterHighPrioAccess ();`	`Uint32 CSL_RAC_Stats_getMasterHighPrioAccess ( CSL_RAC_Handle          hRac );`	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
`void CSL_RAC_HP_FD_CTL_setDataPriority ( Uint8 priorityVal );`	`void CSL_RAC_HP_FD_CTL_setDataPriority ( CSL_RAC_Handle          hRac, Uint8                  priorityVal );`	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
`void CSL_RAC_HP_FD_CTL_setObdPriority ( Uint8 priorityVal );`	`void CSL_RAC_HP_FD_CTL_setObdPriority ( CSL_RAC_Handle          hRac, Uint8                  priorityVal );`	Added a new argument RAC instance handle to indicate which RAC instance to

		use for the operation.
void CSL_RAC_HP_FPE_setDataPriority (Uint8 priorityVal);	void CSL_RAC_HP_FPE_setDataPriority (CSL_RAC_Handle hRac, Uint8 priorityVal);	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
void CSL_RAC_HP_FPE_setObdPriority (Uint8 priorityVal);	void CSL_RAC_HP_FPE_setObdPriority (CSL_RAC_Handle hRac, Uint8 priorityVal);	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
void CSL_RAC_LP_FD_CTL_setDataPriority (Uint8 priorityVal);	void CSL_RAC_LP_FD_CTL_setDataPriority (CSL_RAC_Handle hRac, Uint8 priorityVal);	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
void CSL_RAC_LP_FD_CTL_setObdPriority (Uint8 priorityVal);	void CSL_RAC_LP_FD_CTL_setObdPriority (CSL_RAC_Handle hRac, Uint8 priorityVal);	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
void CSL_RAC_LP_FD_DATA_setDataPriority (Uint8 priorityVal);	void CSL_RAC_LP_FD_DATA_setDataPriority (CSL_RAC_Handle hRac, Uint8 priorityVal);	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
void CSL_RAC_LP_FD_DATA_setObdPriority (Uint8 priorityVal);	void CSL_RAC_LP_FD_DATA_setObdPriority (CSL_RAC_Handle hRac, Uint8 priorityVal);	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
void CSL_RAC_LP_FPE_setDataPriority (	void CSL_RAC_LP_FPE_setDataPriority (	Added a new

`Uint8 priorityVal`
`);`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   priorityVal`
	`);`
		argument RAC instance handle to indicate which RAC instance to use for the operation.

` void CSL_RAC_LP_FPE_setObdPriority (`
`Uint8 priorityVal`
`);`
	` void CSL_RAC_LP_FPE_setObdPriority (`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   priorityVal`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

` void CSL_RAC_LP_FT_setDataPriority (`
`Uint8 priorityVal`
`);`
	` void CSL_RAC_LP_FT_setDataPriority (`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   priorityVal`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

` void CSL_RAC_LP_FT_setObdPriority (`
`Uint8 priorityVal`
`);`
	` void CSL_RAC_LP_FT_setObdPriority (`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   priorityVal`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

` void CSL_RAC_LP_PM_setDataPriority (`
`Uint8 priorityVal`
`);`
	` void CSL_RAC_LP_PM_setDataPriority (`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   priorityVal`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

` void CSL_RAC_LP_PM_setObdPriority (`
`Uint8 priorityVal`
`);`
	` void CSL_RAC_LP_PM_setObdPriority (`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   priorityVal`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

` void CSL_RAC_LP_PD_setDataPriority (`
`Uint8 priorityVal`
`);`
	` void CSL_RAC_LP_PD_setDataPriority (`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   priorityVal`
	`);`
		Added a new argument RAC instance

		handle to indicate which RAC instance to use for the operation.
| ```
 void CSL_RAC_LP_PD_setObdPriority (
Uint8 priorityVal
);
``` | ```
 void CSL_RAC_LP_PD_setObdPriority (
CSL_RAC_Handle            hRac,
Uint8                 priorityVal
);
``` | Added a new argument RAC instance handle to indicate which RAC instance to use for the operation. |
| ```
 void CSL_RAC_LP_SPE_setDataPriority (
Uint8 priorityVal
);
``` | ```
 void CSL_RAC_LP_SPE_setDataPriority (
CSL_RAC_Handle            hRac,
Uint8                 priorityVal
);
``` | Added a new argument RAC instance handle to indicate which RAC instance to use for the operation. |
| ```
 void CSL_RAC_LP_SPE_setObdPriority (
Uint8 priorityVal
);
``` | ```
 void CSL_RAC_LP_SPE_setObdPriority (
CSL_RAC_Handle            hRac,
Uint8                 priorityVal
);
``` | Added a new argument RAC instance handle to indicate which RAC instance to use for the operation. |
| ```
 void CSL_RAC_LP_SIP_setDataPriority (
Uint8 priorityVal
);
``` | ```
 void CSL_RAC_LP_SIP_setDataPriority (
CSL_RAC_Handle            hRac,
Uint8                 priorityVal
);
``` | Added a new argument RAC instance handle to indicate which RAC instance to use for the operation. |
| ```
 void CSL_RAC_LP_SIP_setObdPriority (
Uint8 priorityVal
);
``` | ```
 void CSL_RAC_LP_SIP_setObdPriority (
CSL_RAC_Handle            hRac,
Uint8                 priorityVal
);
``` | Added a new argument RAC instance handle to indicate which RAC instance to use for the operation. |
| ```
 void CSL_RAC_FE_enable ();
``` | ```
 void CSL_RAC_FE_enable (
CSL_RAC_Handle            hRac
);
``` | Added a new argument RAC instance handle to indicate which RAC |

		instance to use for the operation.

`void CSL_RAC_FE_disable ();`	`void CSL_RAC_FE_disable (`
	`CSL_RAC_Handle          hRac`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`CSL_RAC_FE_transferState`
`CSL_RAC_FE_getStatus ();`
	`CSL_RAC_FE_transferState`
	`CSL_RAC_FE_getStatus (`
	`CSL_RAC_Handle          hRac`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`CSL_RAC_FE_gccpStatus`
`CSL_RAC_FE_getGccpStatus (`
`Uint8 gccpId`
`);`
	`CSL_RAC_FE_gccpStatus`
	`CSL_RAC_FE_getGccpStatus (`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   gccpId`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`void CSL_RAC_FE_setInputBufferDepth (`
`Uint8 ibDepth`
`);`
	`void CSL_RAC_FE_setInputBufferDepth (`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   ibDepth`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`void`
`CSL_RAC_FE_setMaxCyclesPerIteration (`
`Uint16 maxCyclesNb`
`);`
	`void`
	`CSL_RAC_FE_setMaxCyclesPerIteration (`
	`CSL_RAC_Handle          hRac,`
	`Uint16                  maxCyclesNb`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`Uint16 CSL_RAC_FE_getWatchDogStatus`
`();`
	`Uint16 CSL_RAC_FE_getWatchDogStatus (`
	`CSL_RAC_Handle          hRac`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

CSL_RAC_FE_wdInterruptStatus CSL_RAC_FE_getWatchDogInterruptStatus ();	CSL_RAC_FE_wdInterruptStatus CSL_RAC_FE_getWatchDogInterruptStatus (CSL_RAC_Handle hRac);	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
void CSL_RAC_FE_setTimestamp (CSL_RAC_FE_Timestamp_req * timestamp);	void CSL_RAC_FE_setTimestamp (CSL_RAC_Handle hRac, CSL_RAC_FE_Timestamp_req *timestamp);	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
void CSL_RAC_FE_getTimestamp (CSL_RAC_FE_Timestamp_req * timestamp);	void CSL_RAC_FE_getTimestamp (CSL_RAC_Handle hRac, CSL_RAC_FE_Timestamp_req *timestamp);	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
void CSL_RAC_FE_softwareReset ();	void CSL_RAC_FE_softwareReset (CSL_RAC_Handle hRac);	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
CSL_RAC_FE_resetStatus CSL_RAC_FE_getResetStatus ();	CSL_RAC_FE_resetStatus CSL_RAC_FE_getResetStatus (CSL_RAC_Handle hRac);	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
void CSL_RAC_FE_startSoftwareIteration ();	void CSL_RAC_FE_startSoftwareIteration (CSL_RAC_Handle hRac);	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
void CSL_RAC_GCCP_enable (Uint8 gccpId);	void CSL_RAC_GCCP_enable (CSL_RAC_Handle hRac, Uint8 gccpId	Added a new argument RAC

	`);`	instance handle to indicate which RAC instance to use for the operation.

`void CSL_RAC_GCCP_disable (`
`Uint8 gccpId`
`);`
	`void CSL_RAC_GCCP_disable (`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   gccpId`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`Uint16 CSL_RAC_GCCP_getActiveCycles (`
`Uint8 gccpId`
`);`
	`Uint16 CSL_RAC_GCCP_getActiveCycles (`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   gccpId`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`Uint16 CSL_RAC_GCCP_getSequencerCycles`
`(`
`Uint8 gccpId`
`);`
	`Uint16 CSL_RAC_GCCP_getSequencerCycles`
	`(`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   gccpId`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`void`
`CSL_RAC_GCCP_resetHighPriorityQueue (`
`Uint8 gccpId`
`);`
	`void`
	`CSL_RAC_GCCP_resetHighPriorityQueue (`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   gccpId`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`void`
`CSL_RAC_GCCP_resetLowPriorityQueue (`
`Uint8 gccpId`
`);`
	`void`
	`CSL_RAC_GCCP_resetLowPriorityQueue (`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   gccpId`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`void CSL_RAC_GCCP_getReadTime (`
`CSL_RAC_FE_Timestamp_req * timestamp`
`);`
	`void CSL_RAC_GCCP_getReadTime (`
	`CSL_RAC_Handle          hRac,`
	`CSL_RAC_FE_Timestamp_req   *timestamp`
	`);`
		Added a new argument RAC instance handle to indicate

		which RAC instance to use for the operation.

`Uint32`
`CSL_RAC_GCCP_getLowPrioDataLevel (`
`Uint8 gccpId`
`);`
	`Uint32`
	`CSL_RAC_GCCP_getLowPrioDataLevel (`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   gccpId`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`Uint32`
`CSL_RAC_GCCP_getLowPrioDataWatermark (`
`Uint8 gccpId`
`);`
	`Uint32`
	`CSL_RAC_GCCP_getLowPrioDataWatermark (`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   gccpId`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`Uint32`
`CSL_RAC_GCCP_getHighPrioDataLevel (`
`Uint8 gccpId`
`);`
	`Uint32`
	`CSL_RAC_GCCP_getHighPrioDataLevel (`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   gccpId`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`Uint32`
`CSL_RAC_GCCP_getHighPrioDataWatermark (`
`Uint8 gccpId`
`);`
	`Uint32`
	`CSL_RAC_GCCP_getHighPrioDataWatermark (`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   gccpId`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`Uint32`
`CSL_RAC_GCCP_getLowPrioControlLevel (`
`Uint8 gccpId`
`);`
	`Uint32`
	`CSL_RAC_GCCP_getLowPrioControlLevel (`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   gccpId`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

`Uint32`
`CSL_RAC_GCCP_getLowPrioControlWatermark`
`(`
`Uint8 gccpId`
`);`
	`Uint32`
	`CSL_RAC_GCCP_getLowPrioControlWatermark`
	`(`
	`CSL_RAC_Handle          hRac,`
	`Uint8                   gccpId`
	`);`
		Added a new argument RAC instance handle to indicate which RAC instance to use for the

		operation.
```Uint32 CSL_RAC_GCCP_getHighPrioControlLevel (Uint8 gccpId);```	```Uint32 CSL_RAC_GCCP_getHighPrioControlLevel (CSL_RAC_Handle hRac, Uint8 gccpId);```	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
```Uint32 CSL_RAC_GCCP_getHighPrioControlWatermark (Uint8 gccpId);```	```Uint32 CSL_RAC_GCCP_getHighPrioControlWatermark (CSL_RAC_Handle hRac, Uint8 gccpId);```	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
```void CSL_RAC_GCCP_setIctEntry (Uint8 gccpId, Uint8 entryId, Int8 coeff0, Int8 coeff1, Int8 coeff2, Int8 coeff3);```	```void CSL_RAC_GCCP_setIctEntry (CSL_RAC_Handle hRac, Uint8 gccpId, Uint8 entryId, Int8 coeff0, Int8 coeff1, Int8 coeff2, Int8 coeff3);```	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
```void CSL_RAC_GCCP_setCgtEntry (Uint8 gccpId, Uint8 entryId, Uint32 partY);```	```void CSL_RAC_GCCP_setCgtEntry (CSL_RAC_Handle hRac, Uint8 gccpId, Uint8 entryId, Uint32 partY);```	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
```Uint32 CSL_RAC_GCCP_getCgtEntry (Uint8 gccpId, Uint8 entryId);```	```Uint32 CSL_RAC_GCCP_getCgtEntry (CSL_RAC_Handle hRac, Uint8 gccpId, Uint8 entryId);```	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
```void CSL_RAC_GCCP_setPrtEntry (Uint8 gccpId, Uint8 entryId, Uint8 wordId, Uint32 format);```	```void CSL_RAC_GCCP_setPrtEntry (CSL_RAC_Handle hRac, Uint8 gccpId, Uint8 entryId, Uint8 wordId, Uint32 format);```	Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.
```Uint32 CSL_RAC_GCCP_getPrtEntry (Uint8 gccpId,```	```Uint32 CSL_RAC_GCCP_getPrtEntry (CSL_RAC_Handle hRac,```	Added a new argument

Texas Instruments Incorporated   Statement of Work
QRSA # 13320
Revision D   100_CSLC6498

```Uint8   entryId,```
```Uint8   wordId```
```);```
	```Uint8 gccpId,```
	```Uint8 entryId,```
	```Uint8 wordId```
	```);```
		RAC instance handle to indicate which RAC instance to use for the operation.

``` void```
```CSL_RAC_GCCP_getCycleOverflowStatus (```
```Uint8```
```gccpId,```
```CSL_RAC_GCCP_cycleOverflowStatus *```
```cycleOverflowStatus```
```);```
	``` void```
	```CSL_RAC_GCCP_getCycleOverflowStatus (```
	```CSL_RAC_Handle```
	```hRac,```
	```Uint8```
	```gccpId,```
	```CSL_RAC_GCCP_cycleOverflowStatus```
	```*cycleOverflowStatus```
	```);```
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

``` void```
```CSL_RAC_GCCP_getFifoOverflowStatus (```
```Uint8 gccpId,```
```CSL_RAC_GCCP_fifoOverflowStatus *```
```fifoOverflowStatus```
```);```
	``` void```
	```CSL_RAC_GCCP_getFifoOverflowStatus (```
	```CSL_RAC_Handle```
	```hRac,```
	```Uint8```
	```gccpId,```
	```CSL_RAC_GCCP_fifoOverflowStatus *```
	```fifoOverflowStatus```
	```);```
		Added a new argument RAC instance handle to indicate which RAC instance to use for the operation.

``` CSL_RAC_Handle CSL_RAC_open (```
```CSL_RAC_Obj   *pRac2Obj,```
```int   instNum,```
```int   *pStatus```
```);```
	–	New API. MUST be called to open a specific RAC peripheral instance for use by the application.

``` CSL_Status CSL_RAC_close (```
```CSL_RAC_Handle hRac2```
```);```
	–	New API. MUST be called to unreserved the RAC instance.

6 Testing Considerations

6.1 Engineering Unit Test Plan

The functional layer for each CSL IP module was tested according to either of the following approaches:-

- **Functional Test**
  In this approach a functional test case for the IP block was written using the CSL FL API's. This would not only test the CSL FL API but would also ensure that the IP module

Texas Instruments Incorporated   Statement of Work
QRSA # 13320
Revision D   100_CSLC6498

was operational and working. These test cases or example code is also released to the customer as a part of the CSL package.

- **API Test**
  In certain cases it is not possible to have a functional test case; *for example:* Limitations in the Simulator etc. In such cases the CSL Functional layer was tested using the API test schema. *For example*: Assume there is a function to set a specific parameter in a register; there should reside a corresponding API to get the specific parameter from that register too. Thus the API test case for this case would to be to use the SET API to configure the parameter to a pre-defined value and then to use the corresponding GET API to retrieve the value and ensure that it matches the pre-defined value.