MCAL User Manual for Hssl

32-bit TriCore™ AURIX™ TC3xx microcontroller

About this document

Scope and purpose

This User Manual is intended to enable users to integrate the Microcontroller Abstraction Layer (MCAL) software for the TriCore™ AURIX™ family of 32-bit microcontrollers.

This document describes responsibilities of integrator in-charge of integrating MCAL software with the basic software (BSW) stack. This document also provides detailed information on safety, configuration and functions along with examples of usage of significant features.

Note: Detailed information about package installation, safety and other generic information that are common across all modules are provided in MCAL User Manual General.

Intended audience

This document is intended for anyone using the Hssl module of the TC3xx MCAL software.

Document conventions

Table 1 Conventions

Convention	Explanation
Bold	Emphasizes heading levels, column headings, table and figure captions, screen names, windows, dialog boxes, menus, sub-menus
Italics	Denotes variable(s) and reference(s)

`Courier`
 	
`New`
	Denotes APIs, functions, interrupt handlers, events, data types, error handlers, file/folder names, directories, command line inputs, code snippets

>	Indicates that a cascading sub-menu opens when you select a menu item
[cover parentID=<alpha numeric value>]	Used for traceability completeness. Reader should ignore these.

Reference documents

This User Manual should be read in conjunction with the following documents:

- AURIX™ TC3xx MCAL User Manual General

Table of contents

About this document . . . 1
Table of contents . . . 2
1 HSSL driver . . . 5
1.1 User information . . . 5
1.1.1 Description . . . 5
1.1.2 Hardware-software mapping . . . 5
1.1.2.1 HSSL . . . 6
1.1.2.2 SCU: dependent Hardware peripheral . . . 7
1.1.2.3 SRC . . . 7
1.1.2.4 Port . . . 7
1.1.2.5 DMA . . . 8
1.1.3 File structure . . . 8
1.1.3.1 C file structure . . . 8
1.1.3.2 Code generator plugin files . . . 10
1.1.4 Integration hints . . . 11
1.1.4.1 Integration with AUTOSAR stack . . . 11
1.1.4.2 Multicore and resource manager . . . 15
1.1.4.3 MCU support . . . 15
1.1.4.4 PORT support . . . 16
1.1.4.5 DMA support . . . 20
1.1.4.6 Interrupt connections . . . 26
1.1.4.7 Example usage . . . 29
1.1.5 Key architectural considerations . . . 34
1.2 Assumptions of Use (AoU) . . . 34
1.3 Reference information . . . 34
1.3.1 Configuration interfaces . . . 34
1.3.1.1 Container: HsslGeneral . . . 35
1.3.1.1.1 HsslDevErrorDetect . . . 35
1.3.1.1.2 HsslVersionInfoApi . . . 36
1.3.1.1.3 HsslInitApimode . . . 36
1.3.1.1.4 HsslMultiSlaveMode . . . 36
1.3.1.1.5 Hsslclockpredivider . . . 37
1.3.1.1.6 HsslInterfaceMode . . . 37
1.3.1.1.7 HsslOperatingMode . . . 38
1.3.1.2 Container: HsslConfig . . . 38
1.3.1.2.1 HsslInstanceID . . . 38
1.3.1.2.2 HsslCh2Mode . . . 39
1.3.1.2.3 HsslStreamingModeTx . . . 39
1.3.1.2.4 HsslStreamingModeRx . . . 40

Table of contents

1.3.1.2.5 HsslTargetIDAddr 40
1.3.1.2.6 HsslEXICallbackFunction 40
1.3.1.2.7 HsslDmaMultiWriteChannelRef 41
1.3.1.2.8 HsslDmaMultiWriteCallback 41
1.3.1.2.9 HsslDmaMultiReadTxChannelRef 42
1.3.1.2.10 HsslDmaMultiReadRxChannelRef 42
1.3.1.2.11 HsslDmaMultiReadCallback 43
1.3.1.2.12 HsslAcessWindowStartAddrx 43
1.3.1.2.13 HsslAcessWindowEndAddrx 43
1.3.1.2.14 HsslAcessRuleWindowx 44
1.3.1.2.15 HsslReferenceClock 44
1.3.1.2.16 HsslSystemClockDivider 45
1.3.1.2.17 HsslMasterTxSpeed 45
1.3.1.2.18 HsslMasterRxSpeed 46
1.3.1.2.19 Container: HsslChannelCallbackConfig 46
1.3.1.2.20 HsslChxCOKCallbackFunction 46
1.3.1.2.21 HsslChxRDICallbackFunction 47
1.3.1.2.22 HsslChxTRGCallbackFunction 47
1.3.1.2.23 HsslChxERRCallbackFunction 47
1.3.1.3 Container: CommonPublishedInformation 48
1.3.1.3.1 ArMajorVersion 48
1.3.1.3.2 ArMinorVersion 48
1.3.1.3.3 ArPatchVersion 49
1.3.1.3.4 SwMajorVersion 49
1.3.1.3.5 SwMinorVersion 50
1.3.1.3.6 SwPatchVersion 50
1.3.1.3.7 ModuleId 50
1.3.1.3.8 VendorId 51
1.3.1.3.9 Release 51
1.3.2 Functions – Type definitions 52
1.3.2.1 Hssl_DataTemplate 52
1.3.2.2 Hssl_channel 52
1.3.2.3 Hssl_ReadDataTemplate 52
1.3.2.4 Hssl_InstanceID 53
1.3.2.5 Hssl_SlaveStatusType 53
1.3.2.6 Hssl_EventType 53
1.3.3 Functions - APIs 54
1.3.3.1 Hssl_Init 54
1.3.3.2 Hssl_InitChannel 55
1.3.3.3 Hssl_SetMode 55
1.3.3.4 Hssl_Reset 56
1.3.3.5 Hssl_Write 56

Table of contents

1.3.3.6 Hssl_WriteAck 57
1.3.3.7 Hssl_Read 58
1.3.3.8 Hssl_ReadRply 59
1.3.3.9 Hssl_Id 60
1.3.3.10 Hssl_Trigger 60
1.3.3.11 Hssl_StartStream 61
1.3.3.12 Hssl_StopStream 62
1.3.3.13 Hssl_MultiWrite 63
1.3.3.14 Hssl_MultiRead 64
1.3.3.15 Hssl_ActivateSlave 65
1.3.3.16 Hssl_DeactivateSlave 65
1.3.3.17 Hssl_SelectSlave 66
1.3.3.18 Hssl_GetGlobalError 67
1.3.3.19 Hssl_GetChannelError 67
1.3.3.20 Hssl_GetVersionInfo 68
1.3.4 Notifications and callbacks 68
1.3.4.1 Hssl_DmaCallout 69
1.3.4.2 Hssl_DmaErrCallout 69
1.3.5 Scheduled functions 70
1.3.6 Interrupt service routines 70
1.3.6.1 Hssl_IsrCOK 70
1.3.6.2 Hssl_IsrRDI 71
1.3.6.3 Hssl_IsrError 71
1.3.6.4 Hssl_IsrTrg 72
1.3.6.5 Hssl_IsrEXI 72
1.3.7 Callout 73
1.3.8 Error Handling 73
1.3.9 Deviations and limitations 73
1.3.9.1 Deviations 73
1.3.9.1.1 Software specification deviations 73
1.3.9.1.2 AMDC violations 74
1.3.9.1.3 VSMD violations 74
1.3.9.2 Limitations 74
Revision history 74
Disclaimer 75

1 HSSL driver

1.1 User information

1.1.1 Description

The HSSL driver provides the necessary configuration parameters and APIs for the point-to-point communication of single data value and of large data blocks called streams. The HSSL driver initializes the HSCT module. The HSSL driver is implemented as a pre-compile variant. The HSSL driver does not support the read stream.

The features of HSSL are:

- Point-to-point communication between two devices
- Each kernel provides four channels to transfer data to/from target
- Each channels support direct writing of 8/16/32 bit data from initiator to the target register
- For transferring large data blocks channel 2 contains FIFO
- HSSL module implements the transport layer tasks
- HSCT module implements data link layer and physical layer services

1.1.2 Hardware-software mapping

This section describes the system view of the HSSL driver and peripherals administered by it.

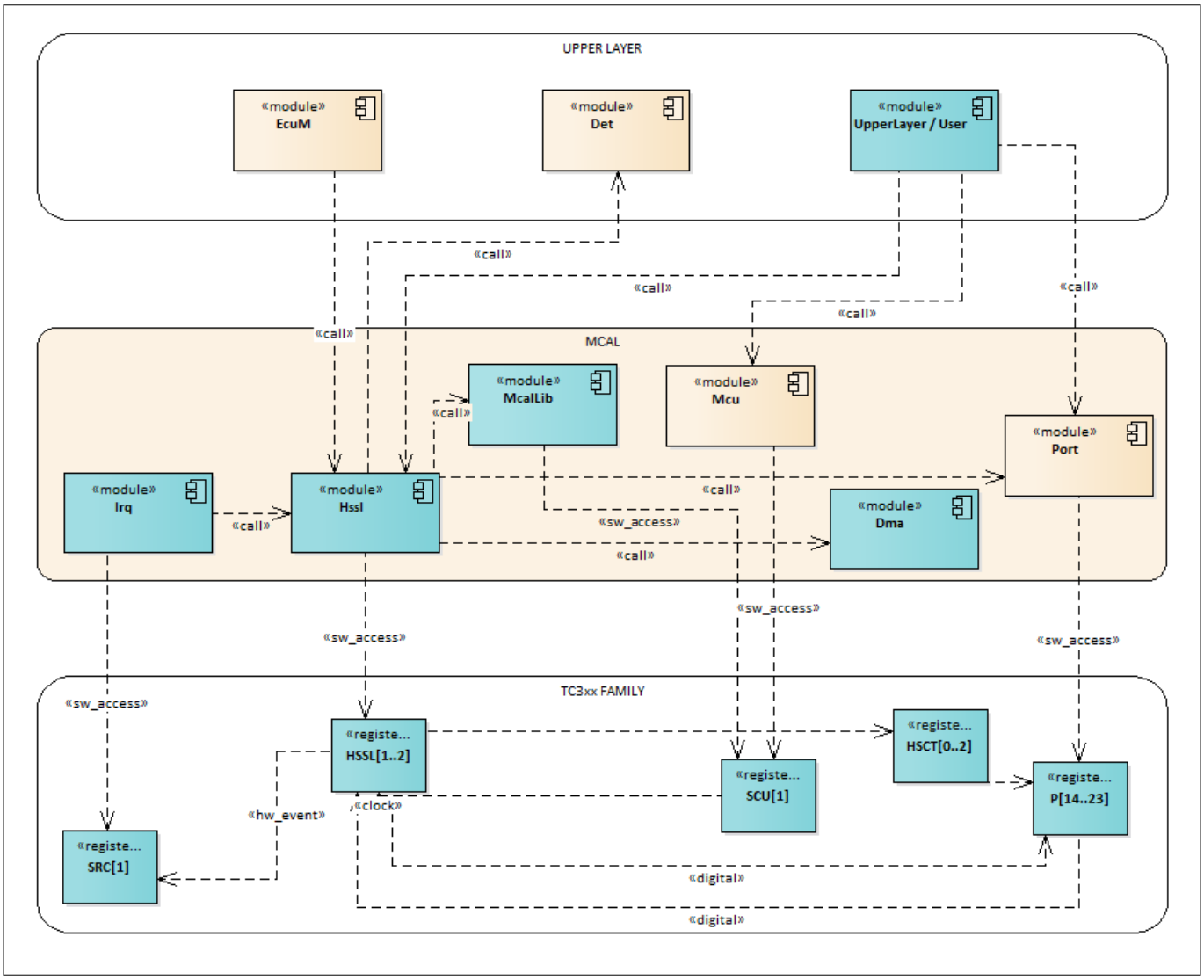


Figure 1 Mapping of hardware-software interfaces

1.1.2.1 HSSL

HSSL is a serial communication protocol driver, which enables two devices to communicate with each other.

Hardware functional features

The key HSSL features used by the HSSL driver are:

- Writing a single 8/16/32 bit data value into the register of a target device
- Reading single data from an 8/16/32 bit register of a target device
- Support of 32-bit address range
- Transfers protected by CRC16
- Programmable time outs for detection of blocked answer transfers
- Automatic frame transfer ID generation for detection of dropped frames
- Support of DMA driven multiple register write / read transfers efficient transmission and reception of large data blocks/streams
- Acknowledge for command and stream frames to reduce latency of error detection
- Two stage FIFOs for transmitting and receiving streaming data

- Automatic FIFO flush when entering the run mode, for error handling
- Write protection by an external memory protection unit
- Remote trigger of event / interrupt in the target device by the initiator
- Identification of the target by the JTAG ID number
- Feature set identification of the HSSL module possible by using the JTAG ID number

Users of the hardware

The HSSL driver uses the HSSL and HSCT peripheral of the AURIX™ platform to realize the functionality. The HSSL driver provides APIs to initialize the complete HSSL and HSCT hardware unit to be able to read/write register, block transfer and streaming of data.

Hardware diagnostic features

None.

Hardware events

The HSSL module generates events for the following conditions.

COK – On successful receiving acknowledgment

RDI – On receiving data

ERR - On channel specific error (NACK, Transaction tag (TTE), TIMEOUT and UNEXPECTED)

TRG – On receiving the trigger frame

EXI – On receiving global errors like CRC, SRI/SPB bus access and PHY Inconsistency Error.

1.1.2.2 SCU: dependent Hardware peripheral

Hardware functional features

The HSSL driver depends on the SCU for the clock functionality.

Users of the hardware

The SCU module provides the clock for all the peripherals. It is only the MCU driver, however, that is responsible for the configuration of the clock tree.

Hardware diagnostic features

The SMU alarms configured for the SCU are not monitored by the HSSL driver.

Hardware events

None.

1.1.2.3 SRC

Hardware functional features

The SRC registers are not updated by the HSSL driver. The application should enable the SRC as per the mode HSSL configuration.

Users of the hardware

None.

Hardware diagnostic features

None.

Hardware events

None.

1.1.2.4 Port

Hardware functional features

The HSSL driver depends on the PORT driver for configuring the LVDS port pins.

Users of the hardware

The port settings are exclusively set by the PORT driver. The HSSL driver is indirectly depended on the port functionality.

Hardware diagnostic features

None.

Hardware events

None.

1.1.2.5 DMA

Hardware functional features

The HSSL driver uses the DMA for the transmission and reception of data in the multiread and multiwrite functions. The HSSL driver uses the interface APIs provided by the DMA driver to use the DMA functionality.

Users of the hardware

The DMA channels are exclusively owned by the DMA user, but the functionality is shared by the MCAL drivers. The DMA driver is triggered for every element transmitted or received on the HSSL interface.

Hardware diagnostic features

Move engine (ME) error is enabled during the data transmission.

Hardware events

If any ME error is encountered during the data transfer, the DMA raises an error which is handled by the driver module.

1.1.3 File structure

1.1.3.1 C file structure

This section provides details of the C files of the HSSL driver.

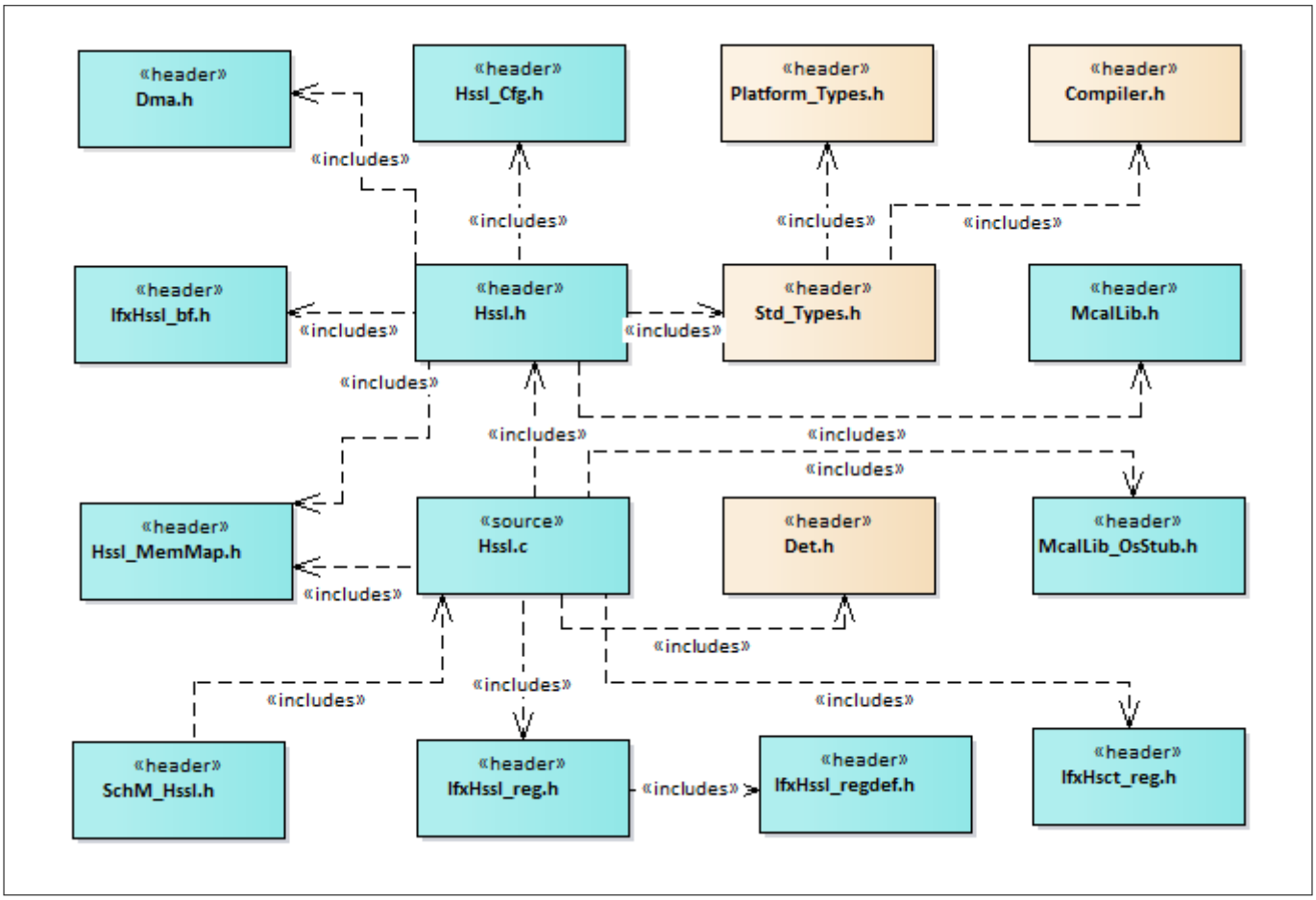


Figure 2 C file structure

Table 2 C file structure

File name	Description
`Compiler.h`	Provides abstraction from compiler-specific keywords
`Std_Types.h`	Standard data types to be used are declared here
`Hssl_Cfg.h`	Generated header file containing macros and configuration data of interrupt priority and interrupt service providers
`Platform_Types.h`	Platform-specific type declaration file as defined by AUTOSAR
`Det.h`	Provides the exported interface of DET
`Hssl.c`	File (static) containing implementation of APIs
`Hssl.h`	Header file (static) defining prototypes of data structures and APIs
`Hssl_MemMap.h`	Memmap file is used to define the section of memory to which variables or constants will be placed
`Dma.h`	Header file (static) defining prototypes of data structure and APIs
`McalLib.h`	Static header file defining prototypes of data structure and APIs exported by the MCALLIB
`IfxHssl_reg.h`	SFR header file for HSSL
`IfxHssl_bf.h`	Provides the Bit Mask, Length and Offset Macro definition for HSSL registers

Table 2 C file structure (continued)

File name	Description
IfxHssl_regdef.h	Includes the register definition file for HSSL
IfxHsct_reg.h	SFR header file for HSCT
IfxHsct_bf.h	Provides the Bit Mask, Length and Offset Macro definition for HSCT registers
IfxHsct_regdef.h	Includes the register definition file for HSCT
IfxDma_reg.h	SFR header file for the DMA
IfxDma_bf.h	SFR header file for the DMA
IfxPort_reg.h	SFR header file for the PORT
IfxPort_bf.h	SFR header file for the PORT
SchM_Hssl.h	Export Header for Schm functions of HSSL driver. Functions to protect critical sections
Hssl_Irq.h	IRQ file for handling all the HSSL interrupts
McalLib_OsStub.h	McalLib_OsStub.h provides macros to support user mode of TriCore™. This shall be included by other drivers to call OS APIs.

1.1.3.2 Code generator plugin files

This section provides details of the code generator plugin files of the HSSL driver.

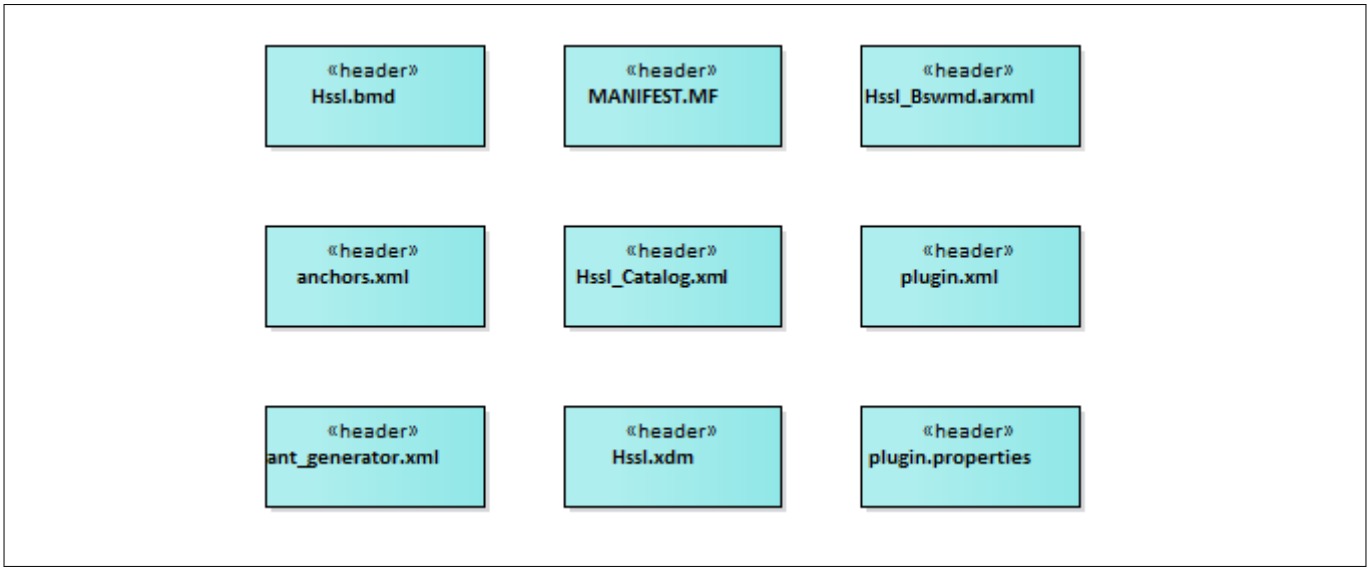


Figure 3 Code generator plugin files

Table 3 Code generator plugin files

File name	Description
anchors.xml	Tresos anchors support file for the HSSL driver
Hssl.xdm	Tresos format XML data model schema file
Hssl.bmd	AUTOSAR format XML data model schema file (for each device)

Table 3 Code generator plugin files (continued)

File name	Description
`MANIFEST.MF`	Tresos plugin support file containing the metadata for the HSSL driver
`plugin.proprties`	Tresos plugin support file for the HSSL driver
`plugin.xml`	Tresos plugin support file for the HSSL driver
`Hssl_Bswmd.arxml`	AUTOSAR format module description file
`Hssl_Catalog.xml`	AUTOSAR format catalog file

1.1.4 Integration hints

This section lists the key points that an integrator or user of the HSSL driver must consider.

1.1.4.1 Integration with AUTOSAR stack

This sections lists the module that are not part of the MCAL, but are required to integrate the HSSL driver.

- **EcuM**

  The ECU Manager module is a part of the AUTOSAR stack that manages common aspects of ECU. Specifically, in the context of MCAL, EcuM is used for initialization of the software drivers. The EcuM module provided in the MCAL package is a stub code and needs to be replaced with a complete EcuM module during the integration phase.

- **Memory mapping**

  Memory mapping is a concept from AUTOSAR that allows relocation of text, variables, constants and configuration data to user-specific memory regions. To achieve this, all the relocatable elements of the driver are encapsulated in different memory-section macros. These macros are defined in the `Hssl_MemMap.h` file. The `Hssl_MemMap.h` file is provided in the MCAL package as a stub code. The integrator must place the appropriate compiler pragmas within the memory-section macros. The pragmas ensure

that the elements are relocated to the correct memory region. A sample implementation listing the memory-section macros is as follows.

```
/*To be used for all global or static variables.*/
#if defined HSSL_START_SEC_VAR_CLEARED_QM_LOCAL_8
    /* User Pragma here */
    #undef HSSL_START_SEC_VAR_CLEARED_QM_LOCAL_8
    #undef MEMMAP_ERROR

#elif defined HSSL_STOP_SEC_VAR_CLEARED_QM_LOCAL_8
    /* User Pragma here */
    #undef HSSL_STOP_SEC_VAR_CLEARED_QM_LOCAL_8
    #undef MEMMAP_ERROR

#elif defined HSSL_START_SEC_VAR_INIT_QM_LOCAL_8
    /* User Pragma here */
    #undef HSSL_START_SEC_VAR_INIT_QM_LOCAL_8
    #undef MEMMAP_ERROR

#elif defined HSSL_STOP_SEC_VAR_INIT_QM_LOCAL_8
    /* User Pragma here */
    #undef HSSL_STOP_SEC_VAR_INIT_QM_LOCAL_8
    #undef MEMMAP_ERROR
#elif defined HSSL_START_SEC_VAR_INIT_QM_LOCAL_32
    /* User Pragma here */
    #undef HSSL_START_SEC_VAR_INIT_QM_LOCAL_32
    #undef MEMMAP_ERROR

#elif defined HSSL_STOP_SEC_VAR_INIT_QM_LOCAL_32
    /* User Pragma here */
    #undef HSSL_STOP_SEC_VAR_INIT_QM_LOCAL_32
    #undef MEMMAP_ERROR

#elif defined HSSL_START_SEC_VAR_CLEARED_QM_LOCAL_32
    /* User Pragma here */
    #undef HSSL_START_SEC_VAR_CLEARED_QM_LOCAL_32
    #undef MEMMAP_ERROR

#elif defined HSSL_STOP_SEC_VAR_CLEARED_QM_LOCAL_32
    /* User Pragma here */
    #undef HSSL_STOP_SEC_VAR_CLEARED_QM_LOCAL_32
    #undef MEMMAP_ERROR

#elif defined HSSL_START_SEC_CONST_QM_LOCAL_32
    /* User Pragma here */
    #undef HSSL_START_SEC_CONST_QM_LOCAL_32
    #undef MEMMAP_ERROR

#elif defined HSSL_STOP_SEC_CONST_QM_LOCAL_32
    /* User Pragma here */
    #undef HSSL_STOP_SEC_CONST_QM_LOCAL_32
    #undef MEMMAP_ERROR
```

```
/* Code Section */
#elif defined HSSL_START_SEC_CODE_QM_LOCAL
    /* User Pragma here */
    #undef HSSL_START_SEC_CODE_QM_LOCAL
    #undef MEMMAP_ERROR

#elif defined HSSL_STOP_SEC_CODE_QM_LOCAL
     /* User Pragma here */
    #undef HSSL_STOP_SEC_CODE_QM_LOCAL
    #undef MEMMAP_ERROR
#endif
#if defined MEMMAP_ERROR
#error "Hssl_MemMap.h, wrong pragma command"
#endif
```

- **DET**

  The DET module is a part of the AUTOSAR stack that handles all the development and runtime errors reported by the BSW modules. The HSSL driver reports all the development errors to the DET module through the `Det_ReportError()` API. The user of the HSSL driver must process all the errors reported to the DET module through the `Det_ReportError()` API.

  The `Det.h` and `Det.c` files are provided in the MCAL package as a stub code and needs to be replaced with a complete DET module during the integration phase.

- **DEM**

  The HSSL driver does not report production errors.

- **SchM**

  The SchM module is a part of the RTE that manages the BSW scheduler. The HSSL driver uses the exclusive areas defined in SchM_Hssl.h to protect the SFRs and variables from concurrent accesses from different threads. The SchMs identified for the HSSL driver are:

  - Channel status lock
  - Activate slave
  - Deactivate slave
  - DMA operated command queues

  The `SchM_Hssl.h` and `SchM_Hssl.c` files are provided in the MCAL package as an example code and needs to updated by the integrator. The user must implement the SchM functions defined by the HSSL driver as suspend/resume of interrupts for the CPU on which the API is invoked. A sample implementation of the SchM functions is as follows:

```
/* sample implementation of SchM_Hssl.c */
void SchM_Enter_Hssl_ChannelStatusLock(void)
{
    /* Start of Critical Section */
    SuspendAllInterrupts();/* Suspend CPU core interrupt */
}

void SchM_Exit_Hssl_ChannelStatusLock(void)
{
    /* Start of Critical Section */
    ResumeAllInterrupts();/* Suspend CPU core interrupt */
}

void SchM_Enter_Hssl_ActivateSlave(void)
{
    /* Start of Critical Section */
    SuspendAllInterrupts();/* Suspend CPU core interrupt */
}

void SchM_Exit_Hssl_ActivateSlave(void)
{
    /* Start of Critical Section */
    ResumeAllInterrupts();/* Suspend CPU core interrupt */
}

void SchM_Enter_Hssl_DeactivateSlave(void)
{
    /* Start of Critical Section */
    SuspendAllInterrupts();/* Suspend CPU core interrupt */
}

void SchM_Exit_Hssl_DeactivateSlave(void)
{
    /* Start of Critical Section */
    ResumeAllInterrupts();/* Suspend CPU core interrupt */
}

void SchM_Enter_Hssl_DmaOperatedCmdQueue(void)
{
    SuspendAllInterrupts();/* Suspend CPU core interrupt */
}

void SchM_Exit_Hssl_DmaOperatedCmdQueue(void)
{
    ResumeAllInterrupts();/* Suspend CPU core interrupt */
}
```

- **Safety error**

  The HSSL driver does not report any safety error.
- **Notifications and callbacks**

The HSSL driver implements the Hssl_UserNotify and Hssl_DMAUserNotify notification functions for ISR_COK, ISR_RDI, ISR_TRG, ISR_ERR, Hssl_DmaCallout and Hssl_DmaErrCallout, respectively. These notification functions can be configured by the user in the EB tresos for each ISRs separately.

- **Operating system**

  The OS or application must ensure correct type of service and interrupt priority is configured in the SR register. Enabling and disabling of interrupts must also be managed by the OS or application. The OS files provided by the MCAL package are only an example code and must be updated by the integrator with the actual OS files for the desired function. The HSSL driver does not program any Service Request (SR) register.

1.1.4.2 Multicore and resource manager

The driver does not support execution on multiple cores in parallel.

1.1.4.3 MCU support

The HSSL driver is dependent on the MCU driver for the clock configuration. The initialization of HSSL driver must be started only after completion of the MCU initialization. The following must be considered while configuring the MCU driver in the EB tresos:

In the MCU configuration, the following fields are to be configured in McuClockRefrencePoint

- McuClockRefSelection
- McuClockRefrencePointFrequency

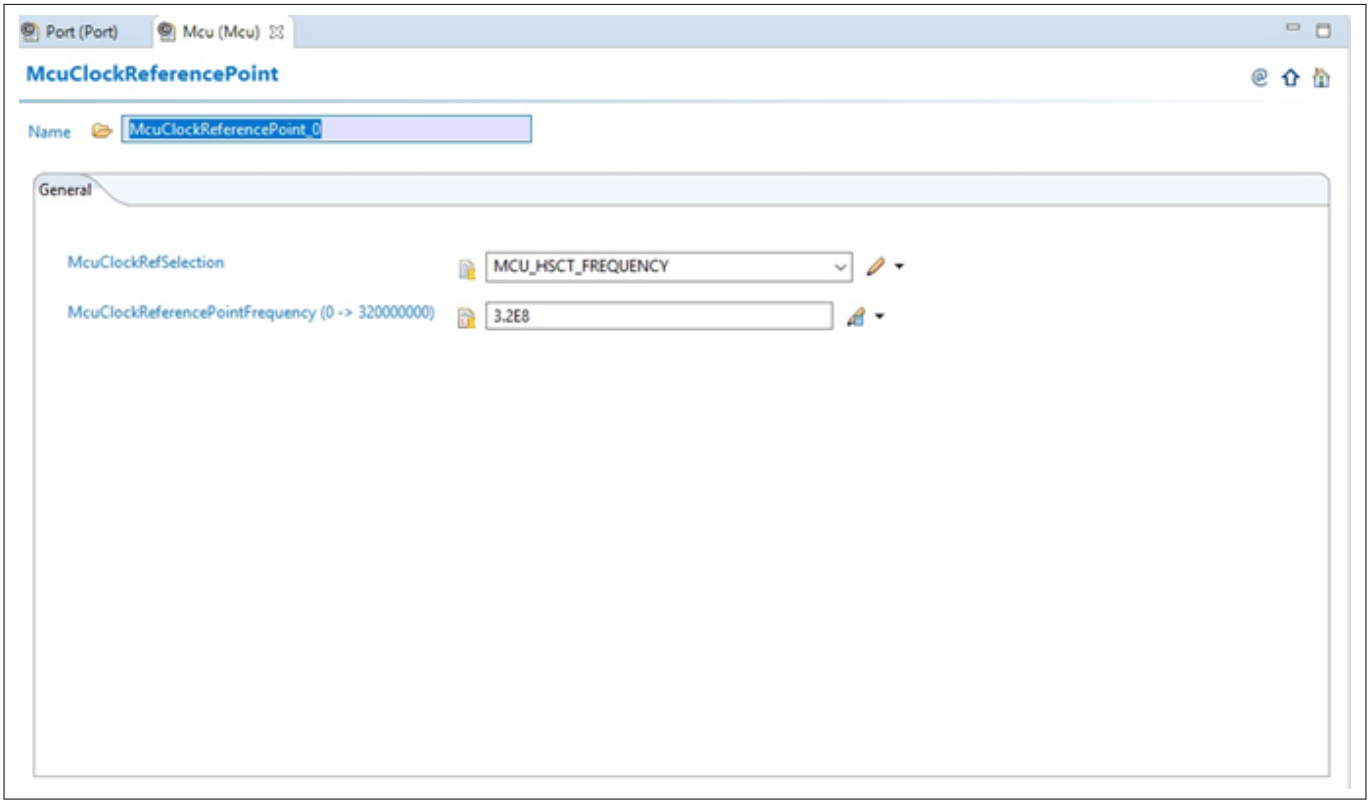


Figure 4 HSSL MCU configuration

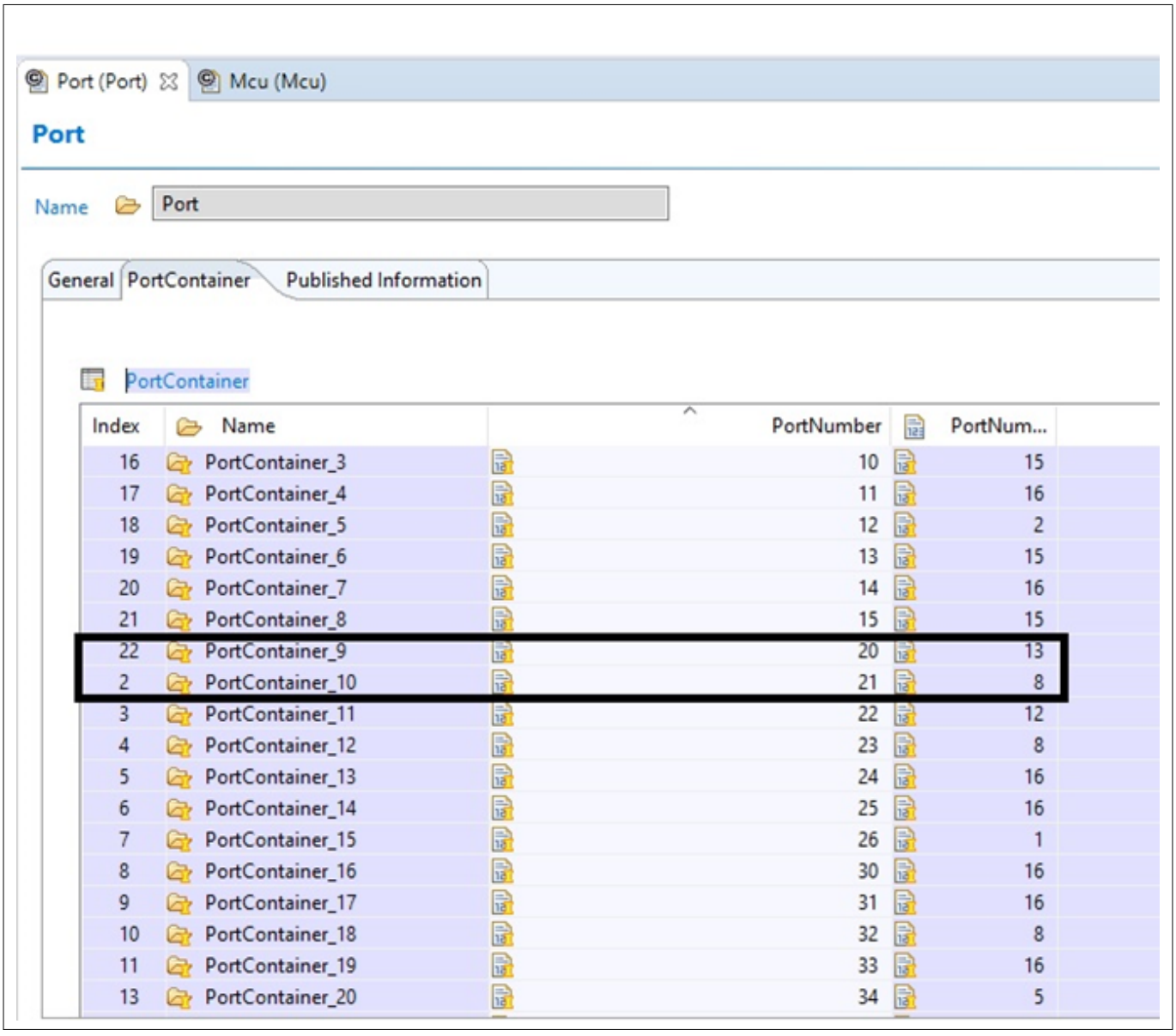
1.1.4.4 PORT support

The PORT driver configures the port pins of the entire microcontroller. The user must configure port pins used by the PORT driver through the port configuration and initialize the port LVDS pins prior to invoking the HSSL initialization.

Following port pins for the HSSL are to be configured as per the configuration:

- HSCT_SYSCLK_OUT - system clock output
- HSCT_RXDN – Rx data negative pin
- HSCT_RXDP - RX data positive pin
- HSCT_TXDN - TX data negative pin
- HSCT_TXDP _ TX data positive pin

The following images shows the example configuration of the PORT pins for HSCT for instance (HSSL0) similarly:



Index	Name	PortNumber	PortNum...
16	PortContainer_3	10	15
17	PortContainer_4	11	16
18	PortContainer_5	12	2
19	PortContainer_6	13	15
20	PortContainer_7	14	16
21	PortContainer_8	15	15
22	PortContainer_9	20	13
2	PortContainer_10	21	8
3	PortContainer_11	22	12
4	PortContainer_12	23	8
5	PortContainer_13	24	16
6	PortContainer_14	25	16
7	PortContainer_15	26	1
8	PortContainer_16	30	16
9	PortContainer_17	31	16
10	PortContainer_18	32	8
11	PortContainer_19	33	16
13	PortContainer_20	34	5

Figure 5 Port number used for HSSL0 module

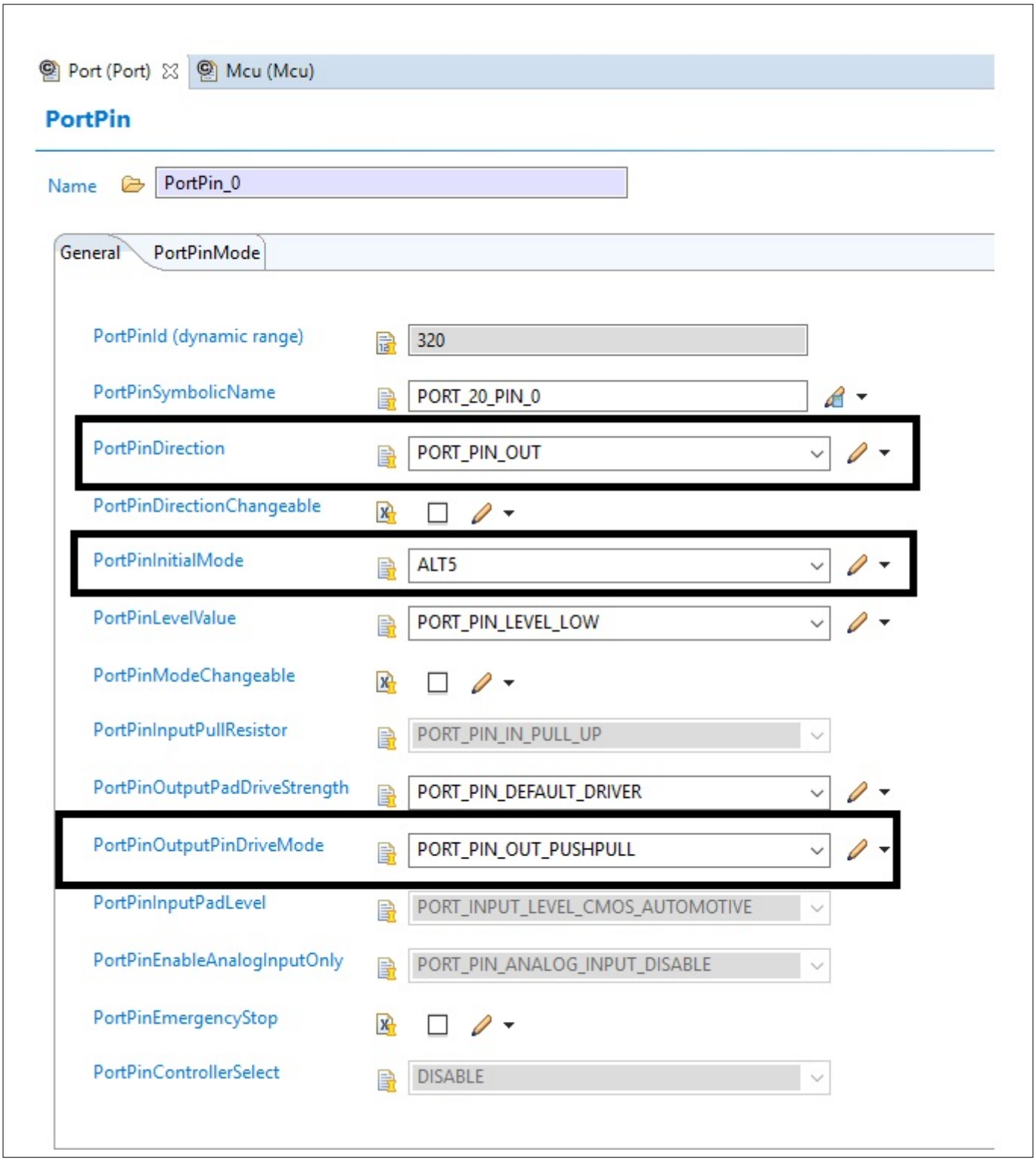


Figure 6 HSSL system clock port output pin configuration

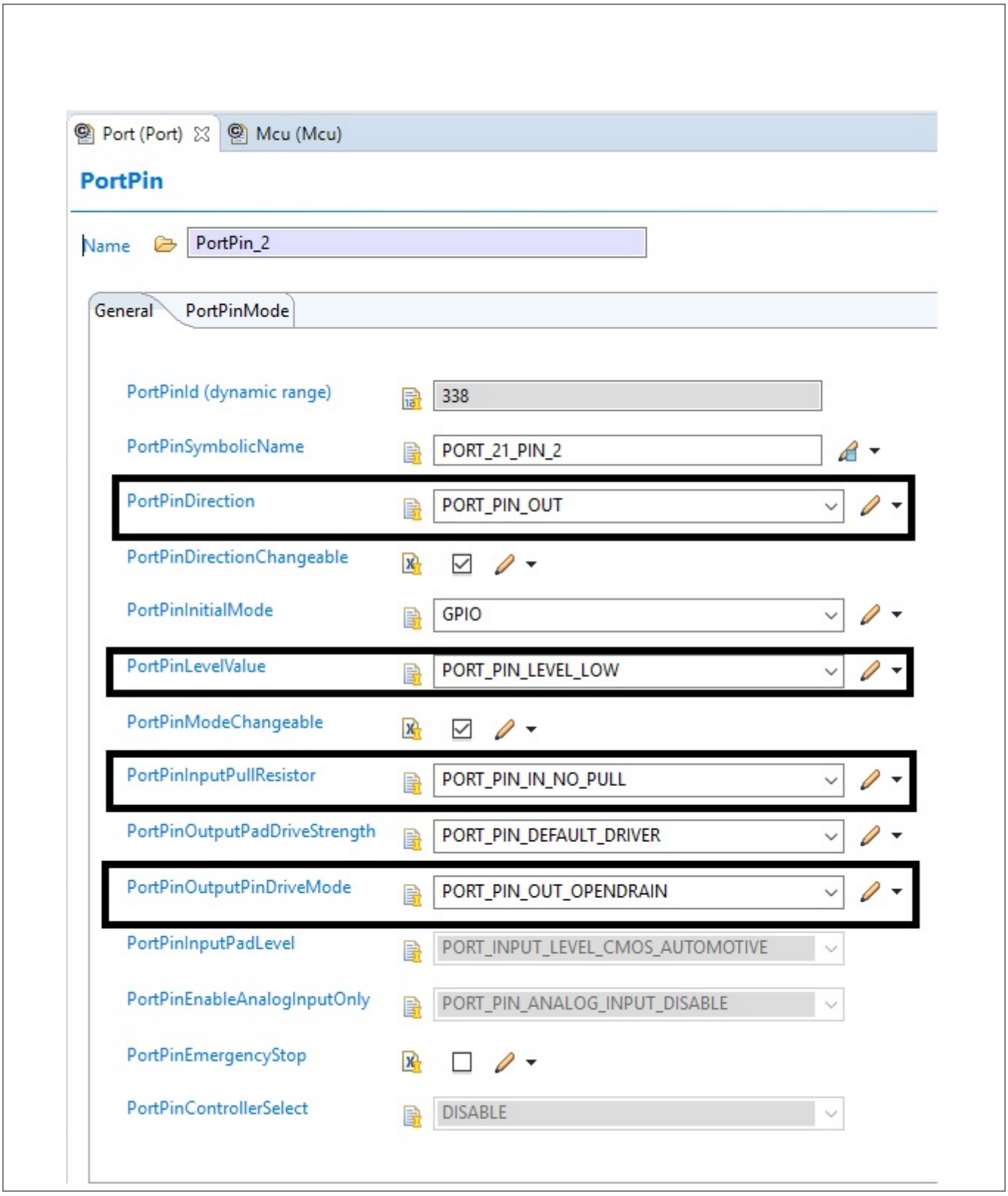


Figure 7 HSSL port pins configuration for RXDN and RXDP

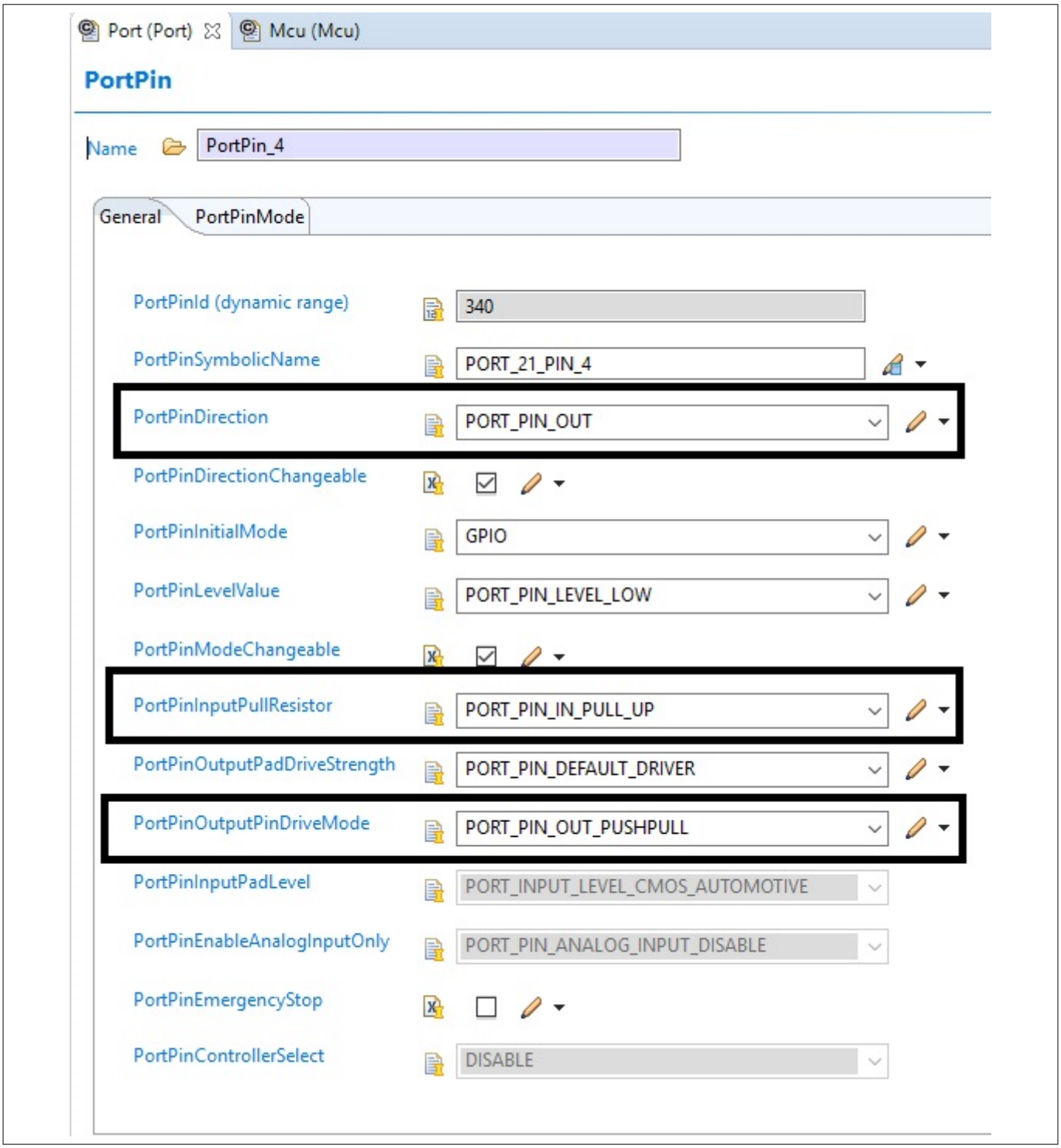


Figure 8 HSSL port configuration for TXDN and TXDP

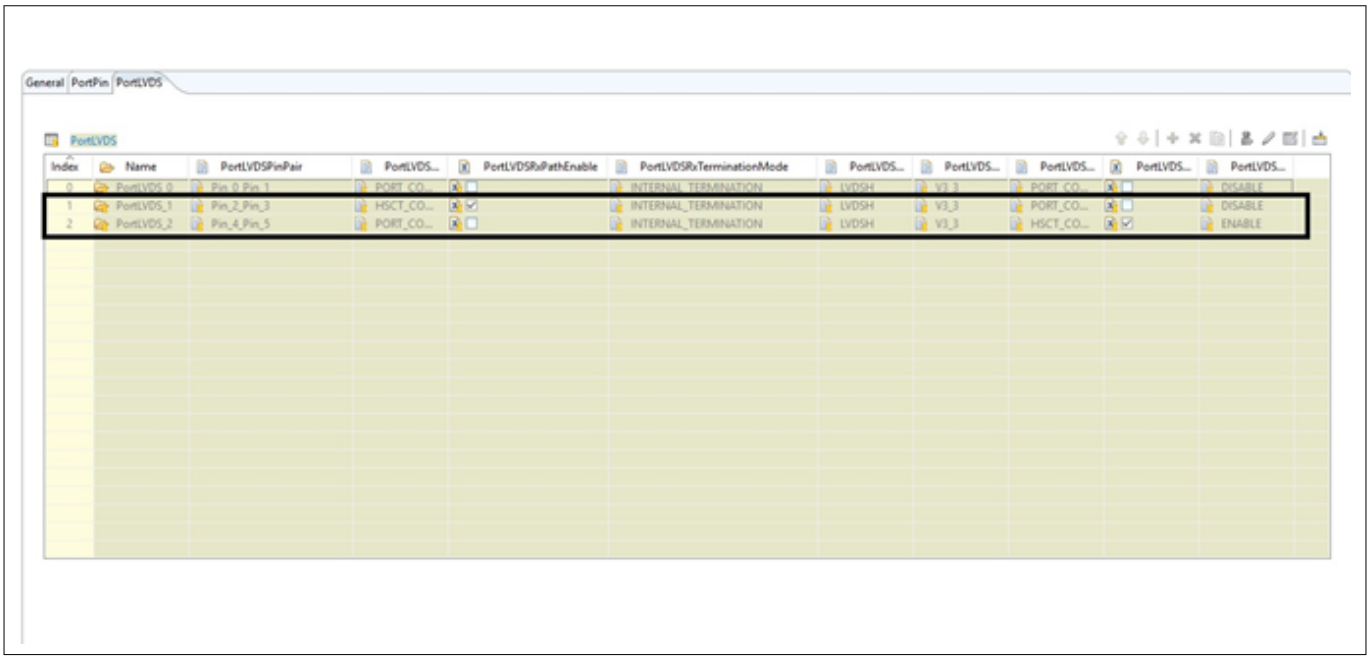


Figure 9 LVDS port configuration for HSCT

1.1.4.5 DMA support

The DMA channels should be configured to use the HSSL `Multi_Read` and `Multi_Write` APIs.
The following figures shows the general configuration of DMA.

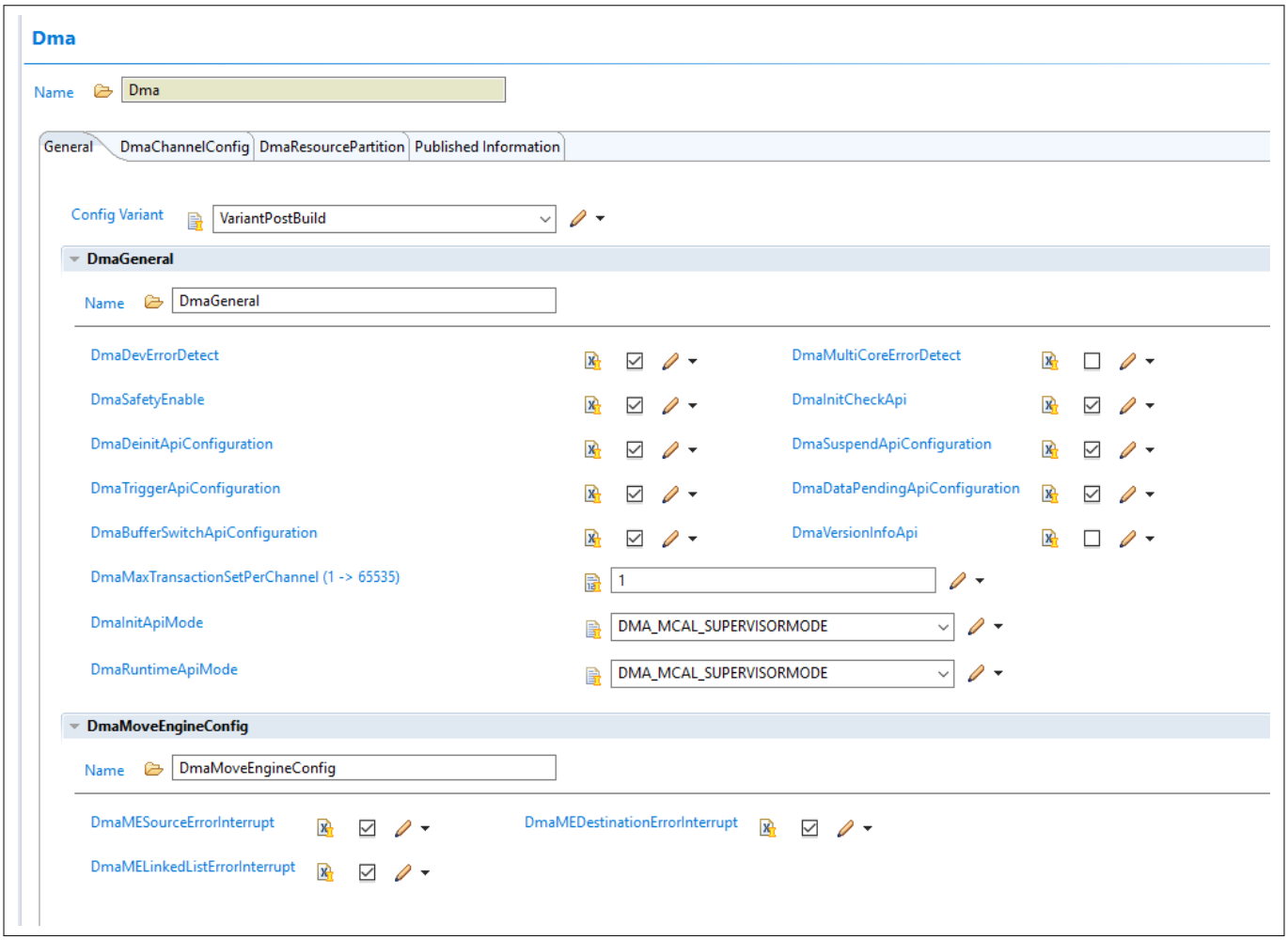


Figure 10 DMA general configuration

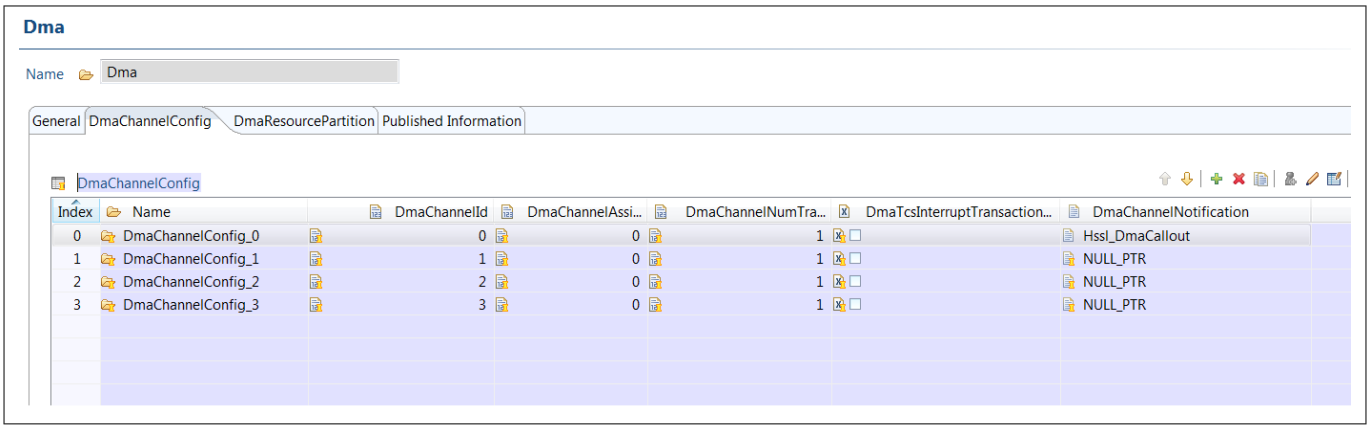


Figure 11 Configure the number of DMA channels to be used

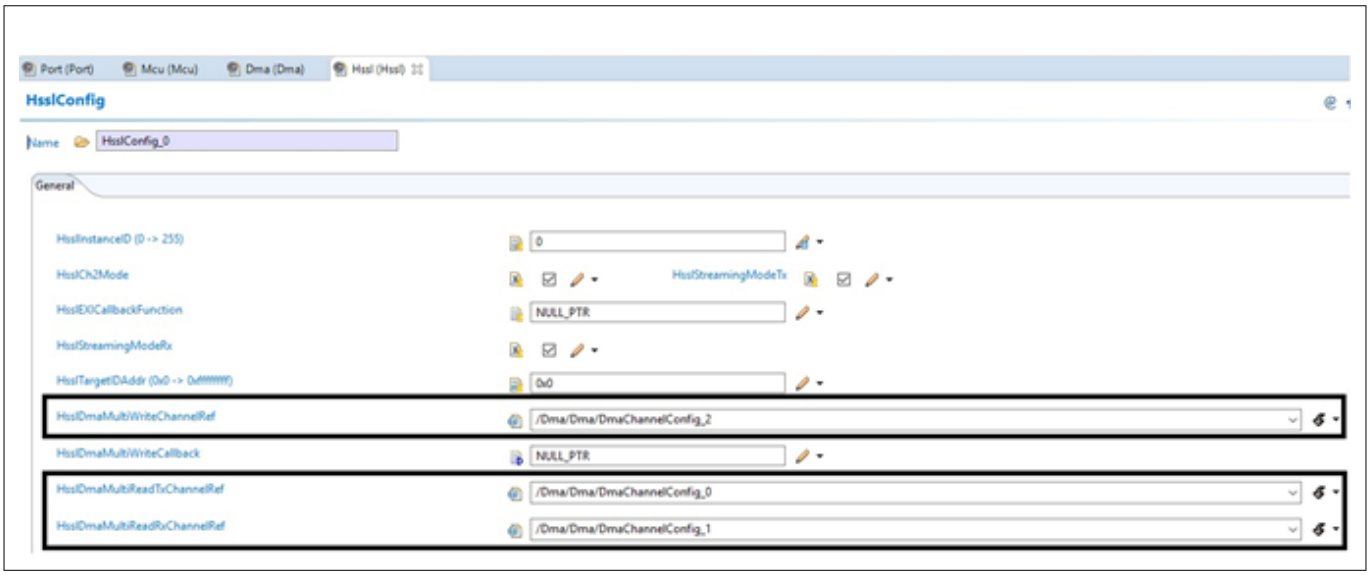


Figure 12 Selection of DMA channels in HSSL configuration

Hssl_MultiRead requires two DMA channels, one for transmission and another for reception.
The following figure shows the configuration to be used for transmit channel for multi-read operation

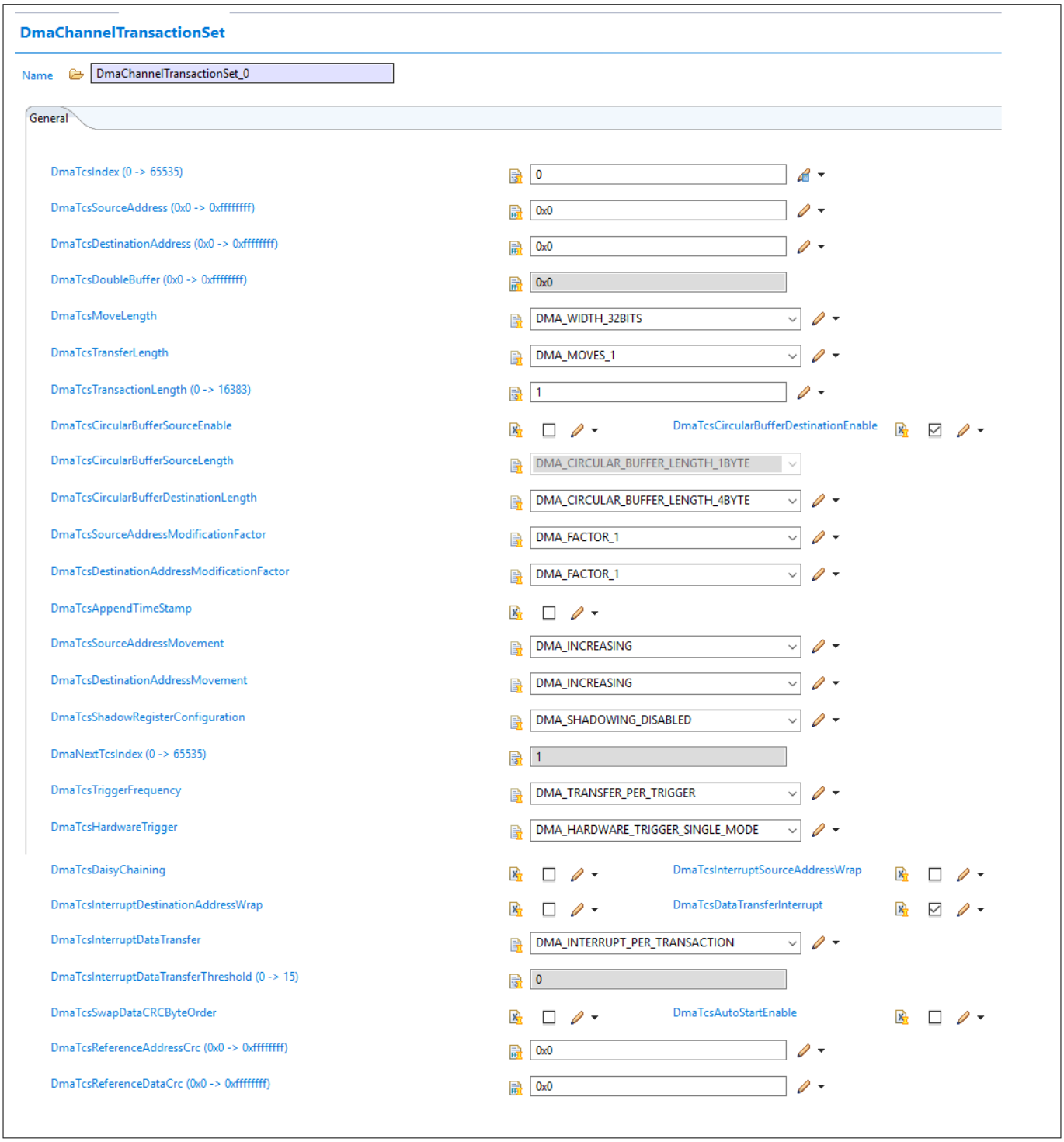


Figure 13 DMA channel transaction configuration for transmit multi-read

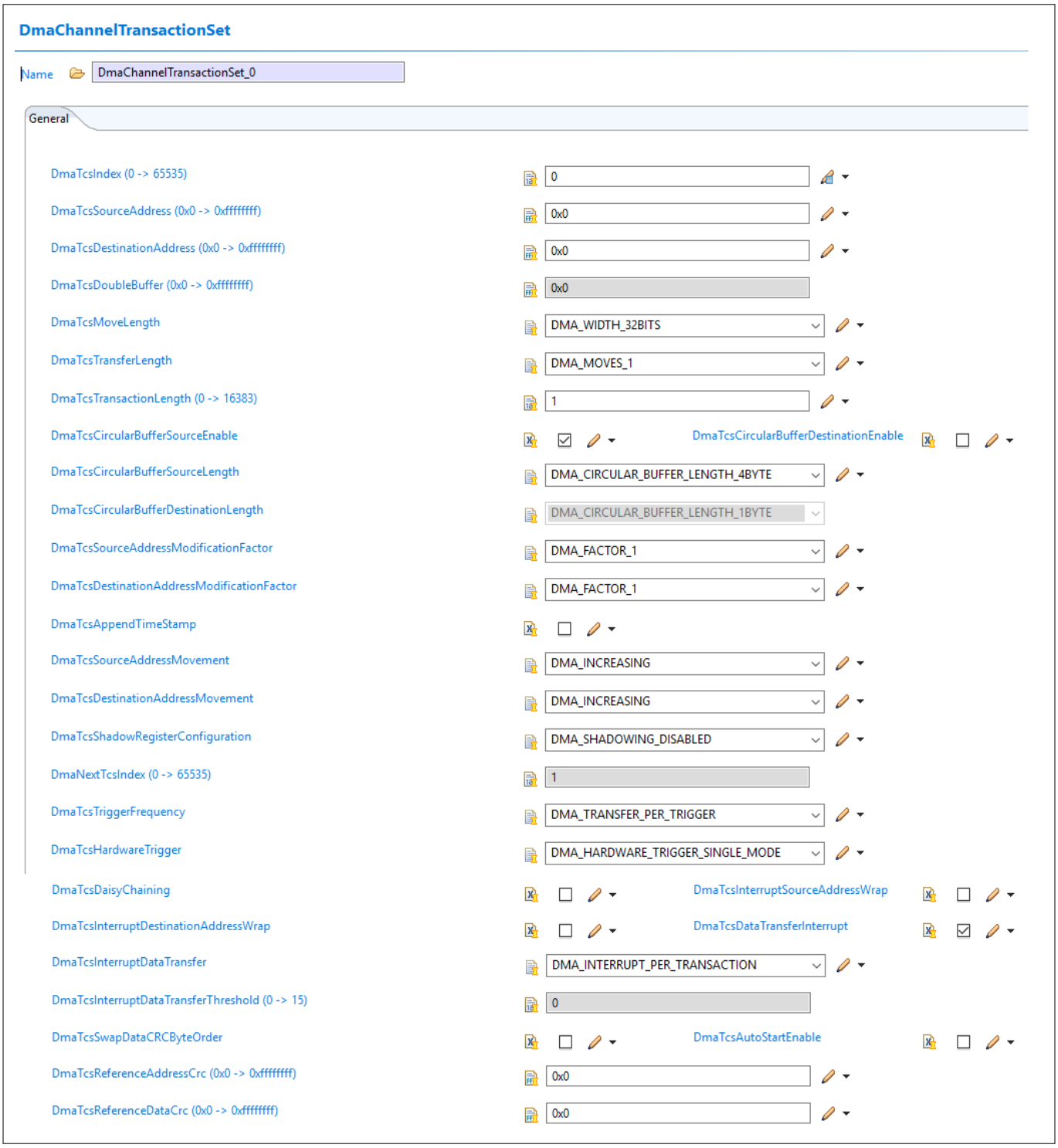


Figure 14 DMA channel transaction for receive multi-read

Hssl_MultiWrite requires one DMA channel for transmission. The following figure shows the configuration to be used for transmit channel for multi-write operation.

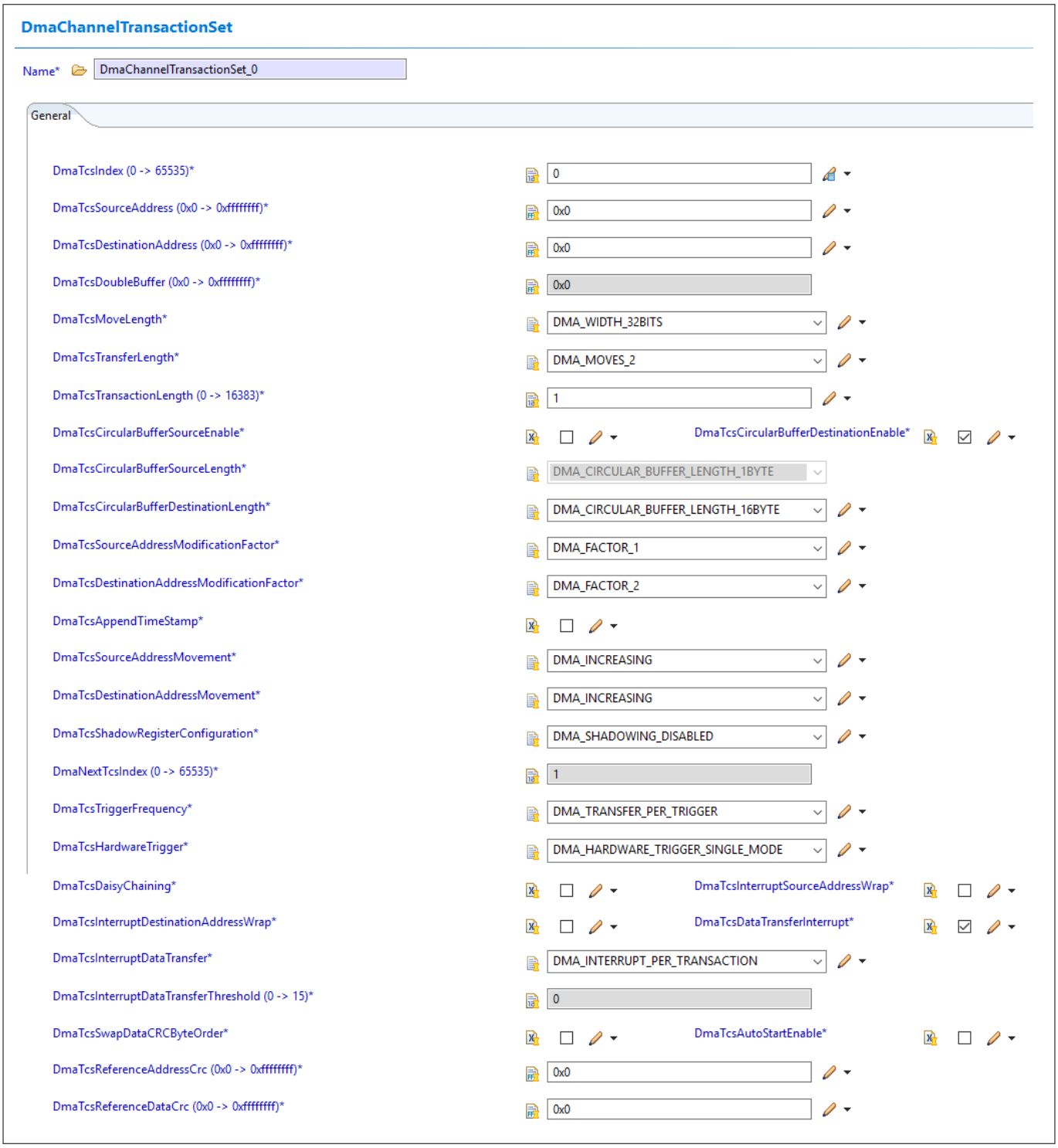


Figure 15 DMA channel transaction for multi-write

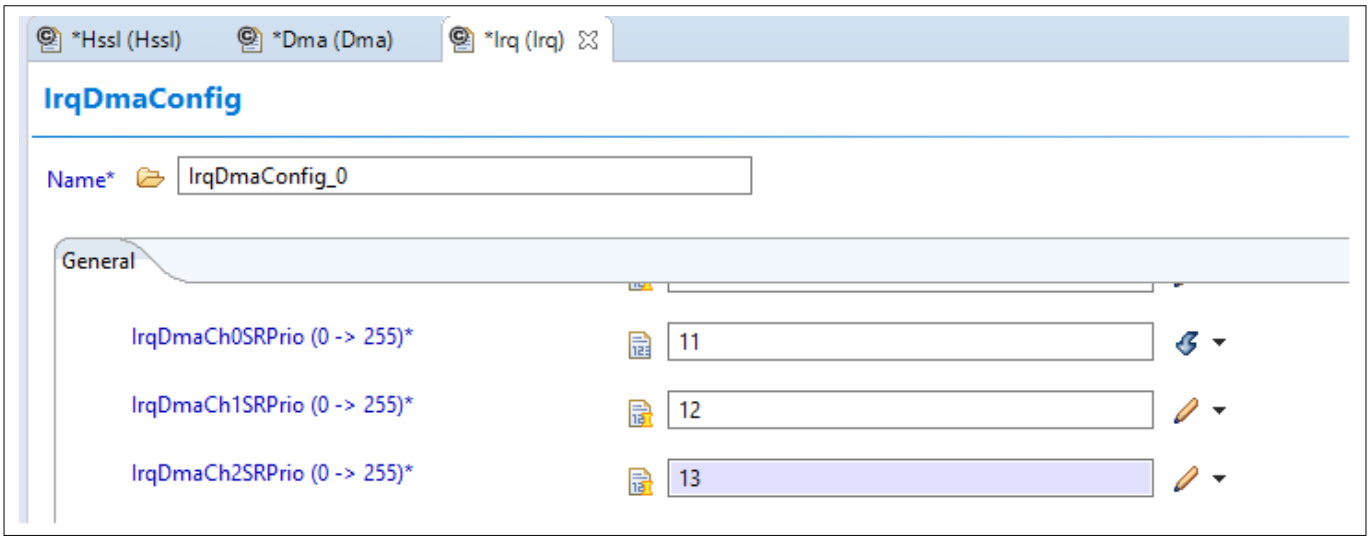


Figure 16 DMA IRQ configuration for multi-read and multi-write

A callout function Hssl_DmaCallout is registered for the each DMA channel which is invoked after every DMA channel transfer completion.

Note: Separate DMA channels to be configured and used for Hssl_MultiRead and Hssl_MultiWrite as mentioned in the above figures. Same DMA channel cannot be used for Hssl_MultiRead and Hssl_MultiWrite.

Note: For Each HSSL kernel requires three separate DMA channels (two for Hssl_MultiRead and one for Hssl_MultiWrite). DMA channels cannot be shared across the HSSL kernels.

Note: In one HSSL kernel, if a channel is performing Hssl_MultiWrite operation then other channels of the same kernel are not allowed to perform the same operation until the channel finishes the HSSL DMA multi write operation. Same applicable for Hssl_MultiRead.

Note: If DMA channel used by the HSSL driver encounters an error, then HSSL driver provides a notification for User. If error occurred during DMA transfer, user application need to do the following:

- *Stop the DMA channel*
- *Deinitialize the DMA channel*
- *Reinitialize the DMA channel and reinitiate the transmission request*

1.1.4.6 Interrupt connections

The HSSL driver has seventeen interrupt lines. The names of the interrupt signals are COK_INT,RDI_INT,ERR_INT and TRG_INT.

Command OK interrupt COK

The arrival of error-free response frame triggers the COK interrupt at the initiator side. This contains four interrupt lines for four channels.

Read data interrupt RDI

The arrival of read response frame triggers RDI interrupt in addition to the COK interrupt. This contains four interrupt lines for four channels.

Error interrupt ERR

When the erroneous response frame (NACK frame) is received at the initiator side, it triggers the ERR interrupt. After an ERR interrupt, normal transmission must be resumed by the software because an optional DMA would remain not triggered and would wait for COK indefinitely. This contains four interrupt lines for four channels.

The HSSL protocol defines four types of errors:

- Time Out Error

A time out error is detected at the initiator side, if the expected ACK frame was not received within the expected time window. This can occur if a frame had been sent by the initiator, and the target detected a CRC error and did not answer with an acknowledge, or the connection between the initiator and the target is physically damaged in one or the other direction.

- Transaction tag error

A transaction tag error occurs at the initiator side, if instead the expected ACK frame with the expected TAG number, an acknowledge with an unexpected transaction tag was received. This would indicate a missing frame or missing acknowledge. Transaction tag errors generate frames that pass the CRC checking stage.

- Target error

A Target Error can occur at the target side, when accessing the target memory a bus error or memory protection error occurs. In such a case, the target responds with an NACK frame indicating the error.

Trigger interrupt TRG

The arrival of a trigger frame at the target side triggers a TRG interrupt at target side. This contains four interrupt lines for four channels.

Exception interrupt EXI

If the receive stage of the HSSL driver detects a CRC error or any inconsistency in the received data, the global EXI Interrupt is activated, which is not channel specific.

- CRC error

A CRC Error can occur:

a. at the target side, in which case:

- the CRC error flag is set
- the received command frame with a CRC error is discarded
- no acknowledge frame is sent and
- a channel unspecific EXI error interrupt is generated, if enabled.

b. at the initiator side, in which case:

- the CRC error flag is set
- the received response frame with a CRC error is discarded
- a channel unspecific EXI error interrupt is generated, if enabled

Both scenarios lead to a time out at the initiator side. In both cases the CRC error flag is set at the side receiving the erroneous frame (either initiator or target) and an interrupt is generated, if enabled.

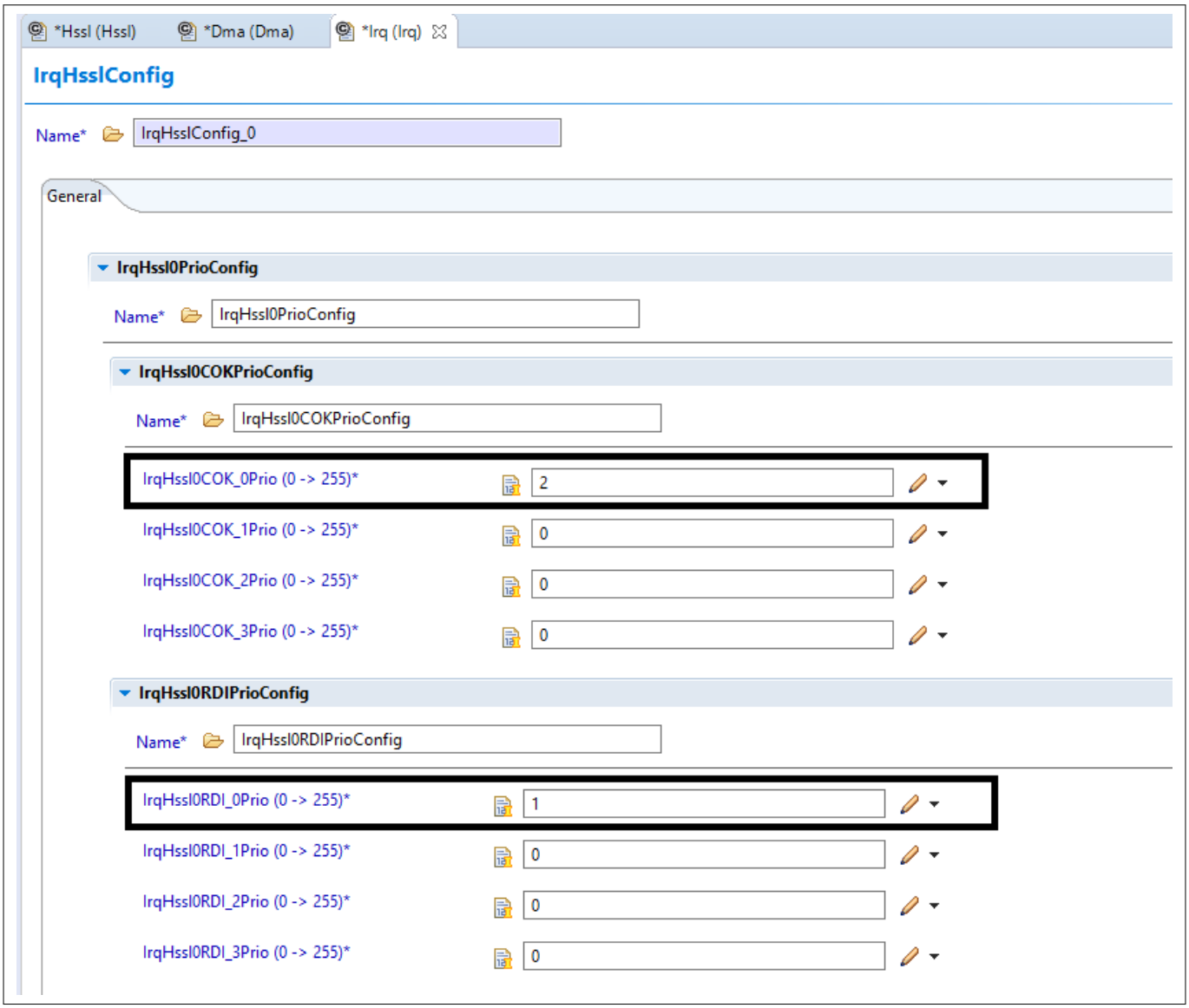


Figure 17 HSSL IRQ priority configuration for multi-read and multi-write

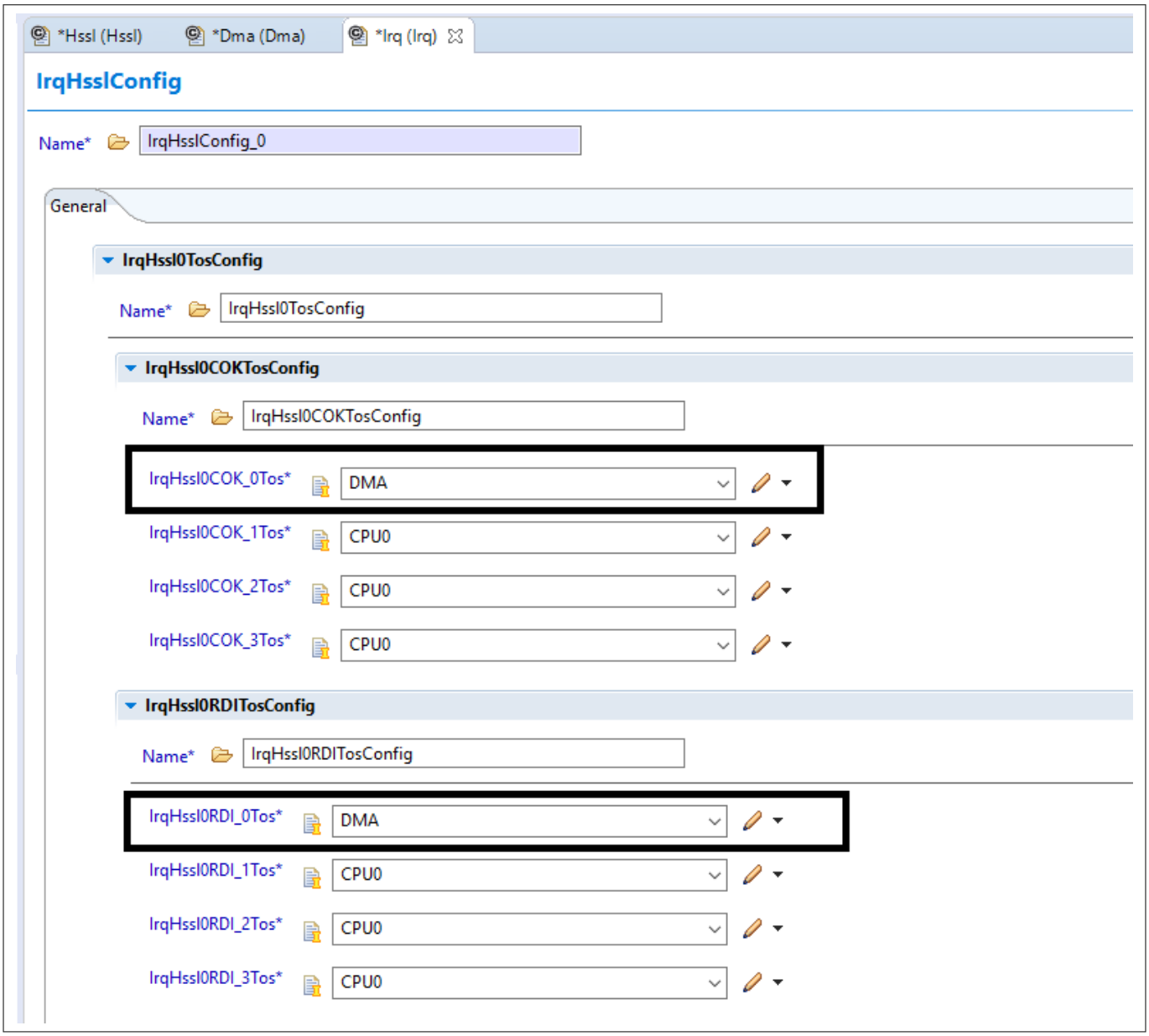


Figure 18 HSSL IRQ configuration for multi-read and multi-write

1.1.4.7 Example usage

The following are the pre-requisite for the HSSL initialization:

Note: Global that needs to be defined in the application code:

- *Mcu_ConfigType Mcu_Config*
- *Port_ConfigType Port_Config*
- *Dma_ConfigType Dma_Config*

Refer to the *Integration hints* and add all dependent modules from the catalog. Follow the below sequence in the application code:

1. Initialize the MCU and clock Mcu_Init API.
2. Initialize the PORT driver using the Port_Init API.
3. Initialize the DMA driver using the Dma_init API.

4. Initialize the IRQ for dependent modules using the IrqDma_Init and IrqHssl_Init APIs.
5. Initialize the HSSL driver using the Hssl_Init API.

Initialization of the driver

The code sequence for initializing the HSSL driver is as follows.

```
#include "Hssl.h"
#include "Mcu.h"
#include "Port.h"
#include "Dma.h"
#include "Irq.h"

extern const Mcu_ConfigType Mcu_Config;
extern const Dma_ConfigType Dma_Config;
extern const Port_ConfigType Port_Config;

void core0_main (void)
{
    Hssl_ConfigType *cfg = NULL_PTR;

    /* Initialize all dependent modules */
    Mcu_Init(&Mcu_Config);
    Mcu_InitClock( 0 );
    while(Mcu_GetPllStatus() != MCU_PLL_LOCKED);
    Mcu_DistributePllClock();

    IrqDma_Init();
    IrqHssl_Init();
    Dma_Init(&Dma_Config);
    Port_Init(&Port_Config);

   /* Enable service request for all the configured interrupts */
    SRC_DMACH1.U |= 0x400U;
    SRC_DMACH2.U |= 0x400U;
    SRC_HSSL0COK0.B.SRE = 0x1;
    SRC_HSSL0RDI0.B.SRE = 0x1;

    Hssl_Init (cfg);
    Hssl_SetMode((Hssl_InstanceID) 0U,HSSL_MODE_INIT) ;
    Hssl_SetMode((Hssl_InstanceID) 0U,HSSL_MODE_RUN) ;
}
```

Sample code for single write command and single read command

The code sequence for performing single write and single read operation between the master and slave is as follows.

```
Hssl_DataTemplateType WriteData;
    uint32 DataBuffer = 0x33333333U;
    uint32 DataAddr;
    Std_ReturnType RetVal;
    Hssl_ChannelType Hssl_channel;

    WriteData.Data = &DataBuffer;
    DataAddr =  0x70003420U;
    WriteData.Address = &DataAddr;
    Hssl_channel.Number = 0U;
    Hssl_channel.Timeout=0xFFFFFFFFU;

   /* Trigger the Write command */
   RetVal = Hssl_Write ((Hssl_InstanceID)0U,&WriteData, HSSL_DATA_SIZE_32BIT, &Hssl_channel
,0U);
   if (RetVal == E_OK)
   {
       RetVal = Hssl_Read ((Hssl_InstanceID)0U,&WriteData,HSSL_DATA_SIZE_32BIT, &Hssl_channel
,0U);
   }
```

Sample code for multi-write operation

The code sequence for performing the multi-write operation on the slave using the DMA is shown as follows (refer to the DMA support and interrupt connection configuration).

```
#include "Hssl.h"
#include "Mcu.h"
#include "Port.h"
#include "Dma.h"
#include "Irq.h"

extern const Mcu_ConfigType Mcu_Config;
extern const Dma_ConfigType Dma_Config;
extern const Port_ConfigType Port_Config;

Hssl_DataTemplateType WriteDataDMA[8U];
uint32 DataBufferDMA[8U];
uint8 RetVal;

/* User callback function is invoked post DMA transmission completes */
void Hssl0_Dma_Write_User_Fn(Dma_ChEventType Event)
{
    while(1);
}

void core0_main (void)
{
    Hssl_ConfigType *cfg = NULL_PTR;

    Hssl_ChannelType Hssl_channel;

      Mcu_Init(&Mcu_Config);
      Mcu_InitClock( 0 );
      while(Mcu_GetPllStatus() != MCU_PLL_LOCKED);
    Mcu_DistributePllClock();

    IrqDma_Init();
    IrqHssl_Init();
    Port_Init(&Port_Config);
       Dma_Init(&Dma_Config);

       SRC_DMACH1.U |= 0x400U;
    SRC_DMACH2.U |= 0x400U;
    SRC_HSSL0COK0.B.SRE = 0x1;
    SRC_HSSL0RDI0.B.SRE = 0x1;

    Hssl_Init (cfg);
    Hssl_SetMode((Hssl_InstanceID) 0U,HSSL_MODE_INIT) ;
    Hssl_SetMode((Hssl_InstanceID) 0U,HSSL_MODE_RUN) ;


      Hssl_channel.Number = 0U;
      Hssl_channel.Timeout=0xFFFFFFFFU;
```

```
      DataBufferDMA[0U] = 0xAAAAAAAAU;
      DataBufferDMA[1U] = 0x70003420U;
      DataBufferDMA[2U] = 0xBBBBBBBBU;
      DataBufferDMA[3U] = 0x70003430U;
      DataBufferDMA[4U] = 0xCCCCCCCCU;
      DataBufferDMA[5U] = 0x70003440U;
      DataBufferDMA[6U] = 0xDDDDDDDDU;
      DataBufferDMA[7U] = 0x70003450U;

    WriteDataDMA[0U].Data = &DataBufferDMA[0U];
    WriteDataDMA[1U].Address = &DataBufferDMA[1];
    WriteDataDMA[2U].Data = &DataBufferDMA[2U] ;
    WriteDataDMA[3U].Address = &DataBufferDMA[3U];
    WriteDataDMA[4U].Data = &DataBufferDMA[4U] ;
    WriteDataDMA[5].Address = &DataBufferDMA[5U];
    WriteDataDMA[6].Data = &DataBufferDMA[6U] ;
   WriteDataDMA[7].Address = &DataBufferDMA[7U];

   RetVal = Hssl_MultiWrite(0U,(Hssl_DataTemplateType *)WriteDataDMA,
HSSL_DATA_SIZE_32BIT,4U,&Hssl_channel,0U);
}
```

Sample code for multi-read operation

The code sequence for performing multi-read operation from the slave using the DMA is shown as folows (refer to the DMA support and interrupt connection configuration).

```
/* User callback function is invoked post DMA transmission completes */
void Hssl0_Dma_Read_User_Fn(Dma_ChEventType Event)
{
    while(1);
}

/*Global variable declarations*/
Hssl_ReadDataTemplateType ReadDataDMA[8U];
uint32 ReaddataBuffer[4];
uint32 DataBufferDMA[8U];
uint8 RetVal;

/*Address buffer from which the data has to read*/
DataBufferDMA[0U] = 0x70003420U;
DataBufferDMA[1U] = 0x70003430U;
DataBufferDMA[2U] = 0x70003440U;
DataBufferDMA[3U] = 0x70003450U;

ReadDataDMA[0U].Address = &DataBufferDMA[0U] ;
ReadDataDMA[1U].Address = &DataBufferDMA[1U];
ReadDataDMA[2U].Address = &DataBufferDMA[2U] ;
ReadDataDMA[3U].Address = &DataBufferDMA[3U];

RetVal = Hssl_MultiRead(0U,
(Hssl_ReadDataTemplateType*)ReadDataDMA,ReadDataDMA,HSSL_DATA_SIZE_32BIT,4U,&Hssl_channel,0U);
```

Sample code for streaming operation

The code sequence for performing streaming operation.

```
uint32 DataAddress[32];
uint32 *DestinationAddressStart = &Dst_Addr;
uint8 retVal;
/*Source buffer data to be transmitted*/
for(Index = 0U; Index < 32U; Index++)
{
  databuf[Index] = 0x22222222;
}
/* Start stream operation */
retVal  = Hssl_StartStream ((Hssl_InstanceID)0U,(&DataAddress[0]),
*DestinationAddressStart,HSSL_DATA_SIZE_32BIT,0U);
```

Sample code for multi-slave operation

The code sequence for performing multi-slave operation.

```
/*Sequence for the multi slave mode*/
uint8 Slaveid = 1U;
uint8 retVal;

/* Slave must be selected before activating a slave */
retVal = Hssl_SelectSlave((Hssl_InstanceID)0U, Slaveid);
if(retVal == E_OK)
{
  retVal = Hssl_ActivateSlave((Hssl_InstanceID)0U,Slaveid,Hssl_SlaveStatusType
*Hssl_SlaveStatus);
}

/*Perform read or write or stream operation*/

/* Deactivating slave */
retVal = Hssl_DeactivateSlave((Hssl_InstanceID)0U,Slaveid,Hssl_SlaveStatusType
*Hssl_SlaveStatus)
```

1.1.5 Key architectural considerations

There are no key architectural considerations for the HSSL driver.

1.2 Assumptions of Use (AoU)

There are no AoU for the HSSL driver.

1.3 Reference information

1.3.1 Configuration interfaces

The HSSL driver is delivered as a Variant Pre-Compile.

The following diagram depicts the hierarchy along with the extensions provided for HSSL module.

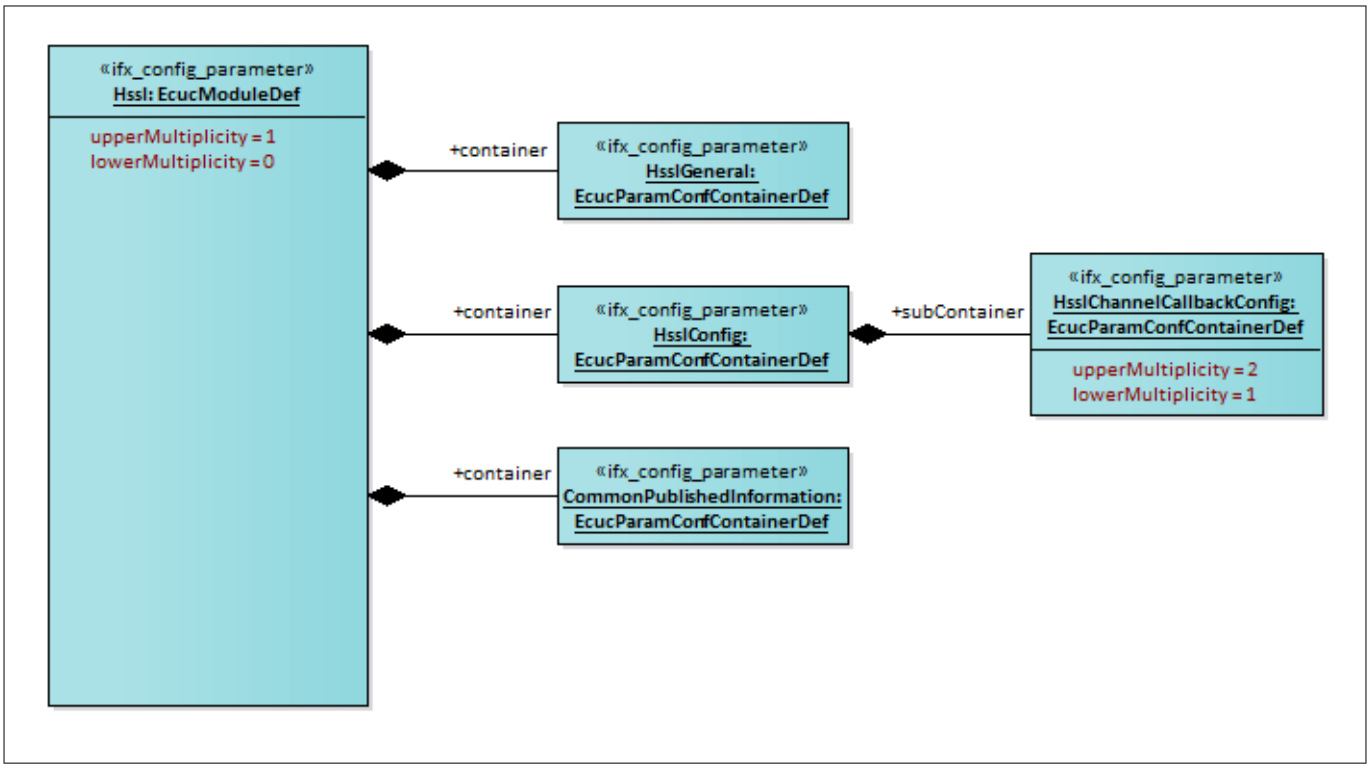


Figure 19 HSSL module configuration

1.3.1.1 Container: HsslGeneral

This container contains the general configuration parameters of the HSSL driver

1.3.1.1.1 HsslDevErrorDetect

Table 4 Specification for HsslDevErrorDetect

Name	HsslDevErrorDetect		
Description	Enables or disables the development error detection		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE: Enabled FALSE: Disabled		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.1.2 HsslVersionInfoApi

Table 5 Specification for HsslVersionInfoApi

Name	HsslVersionInfoApi		
Description	Enables or disables the Hssl_GetVersionInfo function		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE: Enabled FALSE: Disabled		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.1.3 HsslInitApimode

Table 6 Specification for HsslInitApiMode

Name	HsslInitApiMode		
Description	This configuration parameter defines the mode in which the HSSLInit API will be used		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	HSSL_MCAL_SUPERVISOR HSSL_MCAL_USER		
Default value	HSSL_MCAL_SUPERVISOR		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.1.4 HsslMultiSlaveMode

Table 7 Specification for HsslMultiSlaveMode

Name	HsslMultiSlaveMode		
Description	Enables or disables the multi slave mode		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE: Enabled		

Table 7 Specification for HsslMultiSlaveMode (continued)

	FALSE: Disabled		
Default value	FALSE		
Post-build variant value	FALSE	Post-build variant multiplicity	
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.1.5 Hsslclockpredivider

Table 8 Specification for Hsslclockpredivider

Name	Hsslclockpredivider		
Description	This configuration parameter is used to set the clock predivider value		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0x0000 0x3FFF		
Default value	0xFF		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.1.6 HsslInterfaceMode

Table 9 Specification for HsslInterfaceMode

Name	HsslInterfaceMode		
Description	This configuration parameter is used to select the master or slave interface		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	HSSL_MASTER HSSL_SLAVE		
Default value	HSSL_MASTER		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-

Table 9 Specification for HsslInterfaceMode (continued)

Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.1.7 HsslOperatingMode

Table 10 Specification for HsslOperatingMode

Name	HsslOperatingMode		
Description	This configuration parameter is used to select the operating mode in polling or interrupt mode		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	HSSL_POLLING_MODE HSSL_INTERUPT_MODE		
Default value	HSSL_POLLING_MODE		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2 Container: HsslConfig

This container contains the module kernel specific configuration parameters.

Note: Availability of modules is based on the release notes

1.3.1.2.1 HsslInstanceID

Table 11 Specification for HsslInstanceID

Name	HsslInstanceID		
Description	This configuration parameter is used to select HSSL instance		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 : HSSL0 1: HSSL1		
Default value	0: HSSL0		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-

Table 11 Specification for HsslInstanceID (continued)

Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.2 HsslCh2Mode

Table 12 Specification for HsslCh2Mode

Name	HsslCh2Mode		
Description	This configuration parameter is used to select channel 2 mode in streaming or command mode		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE : HSSL_CH2_STREAMING FALSE : HSSL_CH2_COMMAND		
Default value	FALSE : HSSL_CH2_COMMAND		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.3 HsslStreamingModeTx

Table 13 Specification for HsslStreamingModeTx

Name	HsslStreamingModeTx		
Description	Defines the Streaming Mode for Transmitter to be either Continuous or Streaming.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	TRUE : HSSL_STREAMING_MODE_SINGLE FALSE: HSSL_STREAMING_MODE_CONTINOUS		
Default value	FALSE: HSSL_STREAMING_MODE_CONTINOUS		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.4 HsslStreamingModeRx

Table 14 Specification for HsslStreamingModeRx

Name	HsslStreamingModeRx		
Description	Defines the Streaming Mode for Receiver to be either Continuous or Streaming.		
Multiplicity	1..1	Type	EcucBooleanParamDef
Range	TRUE : HSSL_STREAMING_MODE_SINGLE FALSE: HSSL_STREAMING_MODE_CONTINOUS		
Default value	FALSE: HSSL_STREAMING_MODE_CONTINOUS		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.5 HsslTargetIDAddr

Table 15 Specification for HsslTargetIDAddr

Name	HsslTargetIDAddr		
Description	Defines the Address pointer containing the address of the memory location containing the unique ID data		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 65535		
Default value	0x0000		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.6 HsslEXICallbackFunction

Table 16 Specification for HsslEXICallbackFunction

Name	HsslEXICallbackFunction		
Description	This configuration parameter is used to define the function name for the user function for global interrupt.		
Multiplicity	1..1	Type	EcucStringParamDef

Table 16 Specification for HsslEXICallbackFunction (continued)

Range	NULL_PTR		
Default value	NULL_PTR		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.7 HsslDmaMultiWriteChannelRef

Table 17 Specification for HsslDmaMultiWriteChannelRef

Name	HsslDmaMultiWriteChannelRef		
Description	This configuration parameter refers to the DmaConfiguration of DMA channel used by HSSL Multi write shall be provided as reference.		
Multiplicity	1..1	Type	EcucRefrenceParamDef
Range	Reference to parameter of type DmaChannel		
Default value	None		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.8 HsslDmaMultiWriteCallback

Table 18 Specification for HsslDmaMultiWriteCallback

Name	HsslDmaMultiWriteCallback		
Description	This configuration parameter is used to define the function name for the user function for multi write function.		
Multiplicity	1..1	Type	EcucStringParamDef
Range	NULL_PTR		
Default value	NULL_PTR		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-

Table 18 Specification for HsslDmaMultiWriteCallback (continued)

Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.9 HsslDmaMultiReadTxChannelRef

Table 19 Specification for HsslDmaMultiReadTxChannelRef

Name	HsslDmaMultiReadTxChannelRef		
Description	This configuration parameter refers to the DmaConfiguration of DMA channel used by HSSL Multi read for TX channel shall be provided as reference.		
Multiplicity	1..1	Type	EcucRefrenceParamDef
Range	Reference to parameter of type DmaChannel		
Default value	None		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.10 HsslDmaMultiReadRxChannelRef

Table 20 Specification for HsslDmaMultiReadRxChannelRef

Name	HsslDmaMultiReadRxChannelRef		
Description	This configuration parameter refers to the DmaConfiguration of DMA channel used by HSSL Multi read for Rx channel shall be provided as reference.		
Multiplicity	1..1	Type	EcucRefrenceParamDef
Range	Reference to parameter of type DmaChannel		
Default value	None		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.11 HsslDmaMultiReadCallback

Table 21 Specification for HsslDmaMultiReadCallback

Name	HsslDmaMultiReadCallback		
Description	This configuration parameter is used to define the function name for the user function for multi read function.		
Multiplicity	1..1	Type	EcucStringParamDef
Range	NULL_PTR		
Default value	NULL_PTR		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.12 HsslAcessWindowStartAddrx

Table 22 Specification for HsslAcessWindowStartAddrx

Name	HsslAcessWindowStartAddrx (x = 0-3)		
Description	Defines the upper 24 bits of the start address of the corresponding access window		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 16384		
Default value	0x0000		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.13 HsslAcessWindowEndAddrx

Table 23 Specification for HsslAcessWindowEndAddrx

Name	HsslAcessWindowEndAddrx (x = 0-3)		
Description	Defines the upper 24 bits of the End address of the corresponding access window		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 - 16384		
Default value	0x0000		

Table 23 Specification for HsslAcessWindowEndAddrx (continued)

Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.14 HsslAcessRuleWindowx

Table 24 Specification for HsslAcessRuleWindowx

Name	HsslAcessRuleWindowx (x = 0-3)		
Description	This configuration parameter represents the Access Rules for Window(x)		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	0 - HSSL_NO_ACCESS 1 - HSSL_READ_ACCESS 2 - HSSL_WRITE_ACCESS 3 - HSSL_READ_WRITE		
Default value	0 - HSSL_NO_ACCESS		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.15 HsslReferenceClock

Table 25 Specification for HsslReferenceClock

Name	HsslReferenceClock		
Description	This configuration parameter is used to select the reference clock frequency		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	0 – HSSL_10MHZ 1 – HSSL_20MHZ 2 – HSSL_40MHZ		
Default value	1 – HSSL_20MHZ		
Post-build variant value	FALSE	Post-build variant multiplicity	-

Table 25 Specification for HsslReferenceClock (continued)

Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.16 HsslSystemClockDivider

Table 26 Specification for HsslSystemClockDivider

Name	HsslSystemClockDivider		
Description	This configuration parameter represents the system clock frequency divider		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	HSSL_SYSCLK_DIV_1 HSSL_SYSCLK_DIV_2 HSSL_SYSCLK_DIV_4		
Default value	HSSL_SYSCLK_DIV_1		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.17 HsslMasterTxSpeed

Table 27 Specification for HsslMasterTxSpeed

Name	HsslMasterTxSpeed		
Description	This configuration parameter is used to select HSSL master transmitter speed		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	HSSL_TX_LOW_SPEED HSSL_TX_HIGH_SPEED		
Default value	HSSL_TX_LOW_SPEED		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.18 HsslMasterRxSpeed

Table 28 Specification for HsslMasterRxSpeed

Name	HsslMasterRxSpeed		
Description	This configuration parameter is used to select HSSL master receiver speed		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	HSSL_RX_LOW_SPEED HSSL_RX_MEDIUM_SPEED HSSL_RX_HIGH_SPEED		
Default value	HSSL_RX_LOW_SPEED		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.19 Container: HsslChannelCallbackConfig

This container contains callback user notification functions for the channel specific interrupts.

1.3.1.2.20 HsslChxCOKCallbackFunction

Table 29 Specification for HsslChxCOKCallbackFunction

Name	HsslChxCOKCallbackFunction (x = 0-3)		
Description	This configuration parameter is used to define the function name for the user call back notification function for command ok interrupt, where x represents the channel number.		
Multiplicity	1..1	Type	EcucStringParamDef
Range	NULL_PTR		
Default value	NULL_PTR		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.21 HsslChxRDICallbackFunction

Table 30 Specification for HsslChxRDICallbackFunction

Name	HsslChxRDICallbackFunction (x = 0-3)		
Description	This configuration parameter is used to define the function name for the user call back notification function for read data interrupt, where x represents the channel number.		
Multiplicity	1..1	Type	EcucStringParamDef
Range	NULL_PTR		
Default value	NULL_PTR		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.22 HsslChxTRGCallbackFunction

Table 31 Specification for HsslChxTRGCallbackFunction

Name	HsslChxTRGCallbackFunction (x = 0-3)		
Description	This configuration parameter is used to define the function name for the user call back notification function for trigger interrupt, triggered by the trigger command frame, where x represents the channel number.		
Multiplicity	1..1	Type	EcucStringParamDef
Range	NULL_PTR		
Default value	NULL_PTR		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.2.23 HsslChxERRCallbackFunction

Table 32 Specification for HsslChxERRCallbackFunction

Name	HsslChxERRCallbackFunction (x = 0-3)
Description	This configuration parameter is used to define the function name for the user call back notification function for error interrupt, where x represents the channel number.

Table 32 Specification for HsslChxERRCallbackFunction (continued)

Multiplicity	1..1	Type	EcucStringParamDef
Range	NULL_PTR		
Default value	NULL_PTR		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3 Container: CommonPublishedInformation

This container contains published information about vendor and versions.

1.3.1.3.1 ArMajorVersion

Table 33 Specification for ArMajorVersion

Name	ArMajorVersion		
Description	This parameter specifies AUTOSAR major release version.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 to 255		
Default value	4		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3.2 ArMinorVersion

Table 34 Specification for ArMinorVersion

Name	ArMinorVersion		
Description	This parameter specifies AUTOSAR minor release version.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 to 255		
Default value	As per AUTOSAR minor version.		

Table 34 Specification for ArMinorVersion (continued)

Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3.3 ArPatchVersion

Table 35 Specification for ArPatchVersion

Name	ArPatchVersion		
Description	This parameter specifies AUTOSAR patch release version.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 to 255		
Default value	As per AUTOSAR patch version.		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3.4 SwMajorVersion

Table 36 Specification for SwMajorVersion

Name	SwMajorVersion		
Description	This parameter specifies software major release version.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 to 255		
Default value	As per driver		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3.5 SwMinorVersion

Table 37 Specification for SwMinorVersion

Name	SwMinorVersion		
Description	This parameter specifies software minor release version.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 to 255		
Default value	As per driver		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3.6 SwPatchVersion

Table 38 Specification for SwPatchVersion

Name	ArPatchVersion		
Description	This parameter specifies software patch release version.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 to 255		
Default value	As per driver		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3.7 ModuleId

Table 39 Specification for ModuleId

Name	ModuleId		
Description	This parameter specifies module identification number.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 to 65535		
Default value	255		

Table 39 Specification for ModuleId (continued)

Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3.8 VendorId

Table 40 Specification for VendorId

Name	VendorId		
Description	This parameter specifies vendor identification number.		
Multiplicity	1..1	Type	EcucIntegerParamDef
Range	0 to 65535		
Default value	17		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.1.3.9 Release

Table 41 Specification for Release

Name	Release		
Description	This parameter indicates the TC3xx dice derivative used for implementation		
Multiplicity	1..1	Type	EcucStringParamDef
Range	String		
Default value	As per hardware derivative		
Post-build variant value	FALSE	Post-build variant multiplicity	-
Value configuration class	Published information	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.2 Functions – Type definitions

This section describes all the type definitions that are used by APIs.

1.3.2.1 Hssl_DataTemplate

Table 42 Hssl_ DataTemplate

Name	`Hssl_ DataTemplate`	
Type	Structure	
File	Hssl.h	
Range	uint16* Data	Pointer to the data Range: [0x0..0xFFFFFFFF]
	uint32* Address	Pointer to the address Range: [0x0..0xFFFFFFFF]
Description	This Type definition is used to hold the address of read data buffer	

1.3.2.2 Hssl_channel

Table 43 Specification for Hssl_channel

Name	`Hssl_ChannelType`	
Type	Structure	
File	Hssl.h	
Range	uint8 Number	Channel number Range: [0..3]
	uint32 Timeout	Variable holding the timeout value of the channel Range: [0x0..0xFFFFFFFF]
Description	This type definition is used to hold the channel number and timeout of the channel	

1.3.2.3 Hssl_ReadDataTemplate

Table 44 Hssl_ Specification for Hssl_ReadDataTemplate

Name	`Hssl_ReadDataTemplateType`	
Type	Structure	
File	Hssl.h	
Range	uint32*	Address Range: [0x0..0xFFFFFFFF]
Description	This Type definition is used to hold the address of read data buffer	

1.3.2.4 Hssl_InstanceID

Table 45 Hssl_ InstanceID

Name	`Hssl_InstanceID`	
Type	Enumeration	
File	Hssl.h	
Range	HSSL0	HSSL instance id is 0
	HSSL1	HSSL instance id is 1
Description	This type definition is used to select the HSSL instance.	

1.3.2.5 Hssl_SlaveStatusType

Table 46 Hssl_ SlaveStatusType

Name	`Hssl_SlaveStatusType`	
Type	Enumeration	
File	Hssl.h	
Range	HSSL_SLAVE_ACTIVATED	Slave is activated
	HSSL_SLAVE_DEACTIVATED	Slave is deactivated
	HSSL_SLAVE_NOT_RESPONDING	Slave is not responding
	HSSL_SLAVE_NOT_SELECTED	Slave is not selected
Description	This type definition is used to select the status of the slave in multislave mode.	

1.3.2.6 Hssl_EventType

Table 47 Hssl_ EventType

Name	`Hssl_EventType`	
Type	Enumeration	
File	Hssl.h	
Range	HSSL_NO_EVENT	0U
	HSSL_WRITE_COMMAND_COMPLETED	0x2U
	HSSL_READ_COMMAND_COMPLETED	0x4U
	HSSL_TRIGGER_COMMAND_COMPLETED	0x8U
	HSSL_ERROR_NACK	0x10U
	HSSL_ERROR_TRANSACTION_TAG	0x20U
	HSSL_ERROR_TIMEOUT	0x40U
	HSSL_ERROR_UNEXPECTED	0x80U

Table 47 Hssl_ EventType (continued)

	HSSL_STREAM_BLOCK_TRANSMITTED	0x100U
	HSSL_STREAM_BLOCK_ERROR_OCCURED	0x200U
	HSSL_STREAM_BLOCK_RECEIVED	0x400U
	HSSL_SRI_BUS_ERROR	0x800U
	HSSL_PIE1_CHANNEL_NUMBER_CODE_ERROR	0x1000U
	HSSL_PIE2_DATA_LENGHT_ERROR	0x2000U
	HSSL_CRC_ERROR	0x4000U
Description	This type definition is used to indicate the event for notification functions.	

1.3.3 Functions - APIs

This section lists the APIs provided by HSSL driver along with a short description of the functionality.

1.3.3.1 Hssl_Init

Table 48 Specification for Hssl_Init API

Syntax	Std_ReturnType Hssl_Init (const Hssl_ConfigType* const Address)	
Service ID	0x3C	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Address	May be null pointer since it is pre compile module
Parameters (out)	None	
Parameters (in-out)	None	
Return	Std ReturnType Returns 'E_OK' if successful, 'E_NOT_OK' otherwise	
Description	Initializes the HSCT and HSSL module ,setting the Access window start and end address , access mode, target address registers and channel 2 mode	
Source	IFX	
Error handling	HSSL_E_INV_POINTER	
Configuration dependencies	-	

1.3.3.2 Hssl_InitChannel

Table 49 Specification for Hssl_Initchannel API

Syntax	Std_ReturnType Hssl_InitChannel (const Hssl_InstanceID id, const Hssl_ChannelType *const Channel, const uint8 TimeoutErr, const uint8 TransID, const uint8 AckErr)	
Service ID	0x3D	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	channel	HSSL Channel
	TimeoutErr	Enable/Disable Timeout Error interrupt
	TransID	Enable/Disable Transaction ID Error interrupt
	AckErr	Enable/Disable Acknowledge Error interrupt
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	This API initializes the HSSL channels and also sets the interrupt.	
Source	IFX	
Error handling	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_NOT_INITIALIZED, HSSL_E_INV_PARAM	
Configuration dependencies	-	

1.3.3.3 Hssl_SetMode

Table 50 Specification for Hssl_SetMode API

Syntax	Std_ReturnType Hssl_SetMode (const Hssl_InstanceID id, const uint8 Mode)
Service ID	0x3A
Sync/Async	Synchronous
Reentrancy	Non-reentrant

Table 50 Specification for Hssl_SetMode API (continued)

Parameters (in)	id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Mode	1 = Initialize, 2 =Run
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	This API sets the mode of the HSSL module to the required mode.	
Source	IFX	
Error handling	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_NOT_INITIALIZED, HSSL_E_INV_PARAM	
Configuration dependencies	-	

1.3.3.4 Hssl_Reset

Table 51 Specification for Hssl_Reset API

Syntax	Std_ReturnType Hssl_Reset (const Hssl_InstanceID id)	
Service ID	0x3B	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	This API resets the HSCT and HSSL kernel and clears the status and error resisters.	
Source	IFX	
Error handling	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_NOT_INITIALIZED	
Configuration dependencies	-	

1.3.3.5 Hssl_Write

Table 52 Specification for Hssl_Write API

Syntax	Std_ReturnType Hssl_Write (

Table 52 Specification for Hssl_Write API (continued)

	const Hssl_InstanceID id, const Hssl_DataTemplateType *WriteData, const uint16 DataSize, const Hssl_ChannelType *const Channel, const uint16 InjectedError)	
Service ID	0x3E	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	writedata	Pointer to Hssl_Datatemplatetype structure which includes write address and data to be written
	Datasize	Size of the data to be written
	Channel	HSSL channel to use
	InjectedError	Error injected if needed
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Triggers the write command. In case of polling mode, Hssl_WriteAck API must be called to wait for acknowledgement. In case of interrupt mode, a notification is given to user after successful reception of acknowledgement	
Source	IFX	
Error handling	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_NOT_INITIALIZED, HSSL_E_INV_PARAM, HSSL_E_INV_MODE, HSSL_E_INV_POINTER	
Configuration dependencies	-	

1.3.3.6 Hssl_WriteAck

Table 53 Specification for Hssl_WriteAck API

Syntax	Std_ReturnType Hssl_WriteAck (const Hssl_InstanceID id, const Hssl_ChannelType *const Channel)
Service ID	0x3F

Table 53 Specification for Hssl_WriteAck API (continued)

Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	channel	HSSL channel to use
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Waits for acknowledgment. The rationale behind adding separate polling function to poll for the acknowledgment: This reduces the blocking time of Hssl_Write API for back to back triggers for other HSSL channels.	
Source	IFX	
Error handling	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_NOT_INITIALIZED, HSSL_E_INV_PARAM, HSSL_E_INV_MODE	
Configuration dependencies	-	

1.3.3.7 Hssl_Read

Table 54 Specification for Hssl_Read API

Syntax	Std_ReturnType Hssl_Read (const Hssl_InstanceID id, const Hssl_DataTemplateType *DataAddress, const uint16 DataSize, const Hssl_ChannelType *const Channel, const uint16 InjectedError)	
Service ID	0x40	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	dataAddress	Pointer to Hssl_DataTemplateType structure which includes read address
	dataSize	Size of data to be read
	Channel	HSSL channel to use
	InjectedError	Error injected if needed
Parameters (out)	None	

Table 54 Specification for Hssl_Read API (continued)

Parameters (in-out)	None
Return	Returns 'E_OK' upon successful triggering of read command,otherwise 'E_NOT_OK' if unsuccessful.
Description	Triggers the read command. In case of polling mode, The response for the read command can be obtained by calling Hssl_ReadRply API. In case of interrupt mode, a notification is given to user after successful read response is received.
Source	IFX
Error handling	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_NOT_INITIALIZED, HSSL_E_INV_PARAM,HSSL_E_INV_MODE, HSSL_E_INV_POINTER
Configuration dependencies	-

1.3.3.8 Hssl_ReadRply

Table 55 Specification for Hssl_ReadRply API

Syntax	Std_ReturnType Hssl_ReadRply (const Hssl_InstanceID id, const Hssl_ChannelType *const Channel)	
Service ID	0x41	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	channel	HSSL Channel to use
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK', if response is received. E_NOT_OK, any error occurred.	
Description	Reads the response (data) for the read command triggered and updates the data buffer which is passed in Hssl_Read API. The rationale behind adding separate polling function to poll for the response: This reduces the blocking time of Hssl_Write API for back to back triggers for other HSSL channels.	
Source	IFX	
Error handling	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_NOT_INITIALIZED, HSSL_E_INV_PARAM,HSSL_E_INV_MODE	

Table 55 Specification for Hssl_ReadRply API (continued)

Configuration dependencies	-

1.3.3.9 Hssl_Id

Table 56 Specification for Hssl_Id API

Syntax	Std_ReturnType Hssl_Id (const Hssl_InstanceID id, uint32 *const StoreAddress, const Hssl_ChannelType *const Channel)	
Service ID	0x42	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	StoreAddress	Pointer to the Address location/variable to store the ID received from target
	Channel	HSSL channel to use
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Sends ID Request Frame to target device. The received data (JTAG_ID) is used to identify the device capabilities.	
Source	IFX	
Error handling	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_NOT_INITIALIZED, HSSL_E_INV_PARAM, HSSL_E_INV_MODE, HSSL_E_INV_POINTER	
Configuration dependencies	-	

1.3.3.10 Hssl_Trigger

Table 57 Specification for Hssl_Trigger API

Syntax	Std_ReturnType Hssl_Trigger (const Hssl_InstanceID id, const Hssl_ChannelType *const Channel)
Service ID	0x4D

Table 57 Specification for Hssl_Trigger API (continued)

Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Channel	HSSL channel to use
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	This API triggers the Trigger interrupt at the Target side	
Source	IFX	
Error handling	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_NOT_INITIALIZED, HSSL_E_INV_PARAM,HSSL_E_INV_MODE	
Configuration dependencies	-	

1.3.3.11 Hssl_StartStream

Table 58 Specification for Hssl_StartStream API

Syntax	Std_ReturnType Hssl_StartStream (const Hssl_InstanceID id, const uint32 *const SourceAddressStart, const uint32 *const DestinationAddressStart, const uint16 DataSize, const uint16 InjectedError)	
Service ID	0x43	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	id	Hssl Instance Id (0:HSSL0 and 1: HSSL1)
	SourceAddressstart	Pointer to address containing start of data to be streamed. *Note: The source address must be aligned to 256 bit.*
	DestinationAddressstart	Pointer to address containing Destination start address of target.

Table 58 Specification for Hssl_StartStream API (continued)

		Note: The source address must be aligned to 256 bit.
	dataSize	Indicates the number of stream frames to transmit. *Note: Each frame length is 256 bit.*
	InjectedError	Error injected if needed
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK', on successful stream operation trigger. E_NOT_OK, in case of any error.	
Description	Perform write stream operation. Read stream is not possible due to hardware limitation. Polling mode for stream operation is not supported. User must enable the interrupts to get the notification about the streaming completion.	
Source	IFX	
Error handling	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_NOT_INITIALIZED, HSSL_E_INV_PARAM, HSSL_E_INV_MODE, HSSL_E_INV_POINTER	
Configuration dependencies	-	

1.3.3.12 Hssl_StopStream

Table 59 Specification for Hssl_StopStream API

Syntax	Std_ReturnType Hssl_StopStream (const Hssl_InstanceID id)	
Service ID	0x44	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Stops the ongoing streaming	
Source	IFX	

Table 59 Specification for Hssl_StopStream API (continued)

Error handling	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_INV_MODE
Configuration dependencies	-

1.3.3.13 Hssl_MultiWrite

Table 60 Specification for Hssl_MultiWrite API

Syntax	Std_ReturnType Hssl_MultiWrite (const Hssl_InstanceID id, const Hssl_DataTemplateType *WriteArray, const uint16 DataSize, const uint16 NumCmd, const Hssl_ChannelType *const Channel, const uint16 InjectedError)	
Service ID	0x45	
Sync/Async	Asynchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	WriteArray	Hssl_DataTemplateType structure which includes array containing write Address and Data to be written for each array record
	DataSIze	Size of data to be written
	NumCmd	Number of address / data pair to be transmitted. *Note: The maximum size must not be greater than 2048.*
	Channel	HSSL Channel to use
	InjectedError	Error injected if needed
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Performs the Multi Write transfer using DMA. User must configure the notification function for configuration parameter “HsslDmaMultiWriteCallback” in order to get notified.	
Source	IFX	
Error handling		

Table 60 Specification for Hssl_MultiWrite API (continued)

	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_NOT_INITIALIZED, HSSL_E_INV_PARAM, HSSL_E_INV_MODE, HSSL_E_INV_POINTER
Configuration dependencies	-

1.3.3.14 Hssl_MultiRead

Table 61 Specification for Hssl_MultiRead API

Syntax	Std_ReturnType Hssl_MultiRead (const Hssl_InstanceID id, const Hssl_ReadDataTemplateType *ReadArray, const uint32 *Buffer, const uint16 DataSize, const uint16 NumCmd, const Hssl_ChannelType *const Channel, const uint16 InjectedError)	
Service ID	0x46	
Sync/Async	Asynchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	Hssl Instance Id (0:HSSL0 and 1: HSSL1)
	ReadArray	Pointer to Hssl_ReadDataTemplateType structure which includes read Address
	Buffer	Store address
	dataSize	Size of data to be written
	NumCmd	Number of address / data pair to be transmitted. *Note: The maximum size must not be greater than 2048.*
	Channel	HSSL Channel to use
	InjectedError	Error injected if needed
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Performs the Multi read transfer using DMA. User must configure the notification function for configuration parameter “HsslDmaMultiReadCallback” in order to get notified.	

Table 61 Specification for Hssl_MultiRead API (continued)

Source	IFX
Error handling	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_NOT_INITIALIZED, HSSL_E_INV_PARAM, HSSL_E_INV_MODE, HSSL_E_INV_POINTER
Configuration dependencies	-

1.3.3.15 Hssl_ActivateSlave

Table 62 Specification for Hssl_ActivateSlave API

Syntax	Std_ReturnType Hssl_ActivateSlave (const Hssl_InstanceID id, const uint8 Hssl_SlaveID, Hssl_SlaveStatusType *const Hssl_SlaveStatus)	
Service ID	0x49	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Hssl_SlaveID	Select the slave based on the slave id in multi-slave mode
	Hssl_SlaveStatus	Status of the slave
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Activates the slave in multi slave mode. API Hssl_SelectSlave must be called before calling this API to select the slave. Once slave is selected, this API is necessary to call to activate the slave. Once slave is activated, any other operation can be performed.	
Source	IFX	
Error handling	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_INV_MODE ,HSSL_E_INV_POINTER	
Configuration dependencies	-	

1.3.3.16 Hssl_DeactivateSlave

Table 63 Specification for Hssl_DeactivateSlave API

Syntax	Std_ReturnType Hssl_DeactivateSlave (

Table 63 Specification for Hssl_DeactivateSlave API (continued)

	const Hssl_InstanceID id, const uint8 Hssl_SlaveID, Hssl_SlaveStatusType *const Hssl_SlaveStatus)	
Service ID	0x4a	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Hssl_SlaveID	Select the slave based on the slave id in multi-slave mode
	Hssl_SlaveStatus	Status of the slave
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Deactivates the slave in multi slave mode. Once the slave is deactivated, It is must to call Hssl_SelectSlave and Hssl_ActivateSlave respectively before calling any other API.	
Source	IFX	
Error handling	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_INV_MODE, HSSL_E_INV_POINTER	
Configuration dependencies	-	

1.3.3.17 Hssl_SelectSlave

Table 64 Specification for Hssl_SelectSlave API

Syntax	Std_ReturnType Hssl_SelectSlave (const Hssl_InstanceID id, uint8 Hssl_SlaveID)	
Service ID	0x4B	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Hssl_SlaveID	Select the slave based on the slave id in multi-slave mode
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	

Table 64 Specification for Hssl_SelectSlave API (continued)

Description	Selects the slave in multi slave mode. This API must be called before calling Hssl_ActivateSlave.
Source	IFX
Error handling	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_INV_MODE, HSSL_E_INV_PARAM
Configuration dependencies	-

1.3.3.18 Hssl_GetGlobalError

Table 65 Specification for Hssl_GetGlobalError API

Syntax	Std_ReturnType Hssl_GetGlobalError (const Hssl_InstanceID id, uint32 *const Hssl_GlobalErrFlg)	
Service ID	0X47	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Hssl_GlobalErrFlg	Pointer to store Hssl Global error flags value
Parameters (out)	None	
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Reads the global error.	
Source	IFX	
Error handling	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_INV_MODE, HSSL_E_INV_POINTER	
Configuration dependencies	-	

1.3.3.19 Hssl_GetChannelError

Table 66 Specification for Hssl_GetChannelError API

Syntax	Std_ReturnType Hssl_GetChannelError (const Hssl_InstanceID id, const Hssl_ChannelType *const Channel, Hssl_ChannelErrorType *const ChannelError)

Table 66 Specification for Hssl_GetChannelError API (continued)

Service ID	0X4C	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
Parameters (out)	Channel	HSSL channel number
Parameters (in-out)	None	
Return	Returns 'E_OK' if successful, otherwise 'E_NOT_OK' if unsuccessful	
Description	Returns the channel error occurred for a specific channel.	
Source	IFX	
Error handling	HSSL_E_INSTANCE_NOT_CONFIGURED, HSSL_E_INV_MODE, HSSL_E_INV_PARAM	
Configuration dependencies	-	

1.3.3.20 Hssl_GetVersionInfo

Table 67 Specification for Hssl_GetVersionInfo API

Syntax	void Hssl_GetVersionInfo (Std_VersionInfoType *const versioninfo)	
Service ID	0X48	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	Versioninfo	Pointer to store the version information of this module
Parameters (out)	None	
Parameters (in-out)	None	
Return	None	
Description	This service returns the version information of module.	
Source	IFX	
Error handling	HSSL_E_INV_POINTER	
Configuration dependencies	-	

1.3.4 Notifications and callbacks

This section lists all the notifications and callbacks of the HSSL driver.

1.3.4.1 Hssl_DmaCallout

Table 68 Specification for Hssl_DmaCallout

Syntax	void Hssl_DmaCallout (const uint8 Channel, const uint32 Event)	
Service ID	None	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Channel	DMA channel number
	Event	DMA channel event
Parameters (out)	None	
Parameters (in-out)	None	
Return	None	
Description	Dma callback is called after the successful transmission.	
Source	IFX	
Error handling	None	
Configuration dependencies	-	

1.3.4.2 Hssl_DmaErrCallout

Table 69 Specification for Hssl_DmaCallout

Syntax	void Hssl_DmaErrCallout (const uint8 Channel, const uint32 Event)	
Service ID	None	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Channel	DMA channel number
	Event	DMA channel event
Parameters (out)	None	
Parameters (in-out)	None	
Return	None	

Table 69 Specification for Hssl_DmaCallout (continued)

Description	This function is called when the error is occurred during DMA transaction.
Source	IFX
Error handling	None
Configuration dependencies	-

1.3.5 Scheduled functions

The HSSL driver does not provide any scheduled functions.

1.3.6 Interrupt service routines

This section lists all the interrupt handlers of the HSSL driver.

1.3.6.1 Hssl_IsrCOK

Table 70 Specification for Hssl_IsrCOK

Syntax	void Hssl_IsrCOK (const Hssl_InstanceID id, const uint8 Channel)	
Service ID	None	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Channel	HSSL channel to use
Parameters (out)	None	
Parameters (in-out)	None	
Return	None	
Description	The error free response frame triggers the COK interrupt	
Source	IFX	
Error handling	-	
Configuration dependencies	HsslChxRDICallbackFunction (x = 0-3) Where x represents the channel number	

1.3.6.2 Hssl_IsrRDI

Table 71 Specification for Hssl_IsrRDI

Syntax	void Hssl_IsrRDI (const Hssl_InstanceID id, const uint8 Channel)	
Service ID	None	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Channel	HSSL channel to use
Parameters (out)	None	
Parameters (in-out)	None	
Return	None	
Description	The read frame triggers the RDI interrupt	
Source	IFX	
Error handling	-	
Configuration dependencies	HsslChxRDICallbackFunction (x = 0-3) Where x represents the channel number	

1.3.6.3 Hssl_IsrError

Table 72 Specification for Hssl_IsrError

Syntax	void Hssl_IsrError (const Hssl_InstanceID id, const uint8 Channel)	
Service ID	None	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Channel	HSSL channel to use
Parameters (out)	None	
Parameters (in-out)	None	
Return	None	
Description	The ISR gets called when the error interrupt is triggered	

Table 72 Specification for Hssl_IsrError (continued)

Source	IFX
Error handling	-
Configuration dependencies	HsslChxERRCallbackFunction (x = 0-3) Where x represents the channel number

1.3.6.4 Hssl_IsrTrg

Table 73 Specification for Hssl_IsrTrg

Syntax	void Hssl_IsrTrg (const Hssl_InstanceID id, const uint8 Channel)	
Service ID	None	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	Id	HSSL Instance Id (0:HSSL0 and 1: HSSL1)
	Channel	HSSL channel to use
Parameters (out)	None	
Parameters (in-out)	None	
Return	None	
Description	ISR get called at target when trigger frame is arrived	
Source	IFX	
Error handling	-	
Configuration dependencies	HsslChxTRGCallbackFunction (x = 0-3) Where x represents the channel number	

1.3.6.5 Hssl_IsrEXI

Table 74 Specification for Hssl_IsrEXI

Syntax	void Hssl_IsrEXI (const Hssl_InstanceID id)
Service ID	None
Sync/Async	Asynchronous
Reentrancy	Reentrant

Table 74 Specification for Hssl_IsrEXI (continued)

Parameters (in)	Id	Hssl Instance Id (0:HSSL0 and 1: HSSL1)
Parameters (out)	None	
Parameters (in-out)	None	
Return	None	
Description	ISR gets called when the global interrupt is triggered	
Source	IFX	
Error handling	-	
Configuration dependencies	HsslEXICallbackFunction	

1.3.7 Callout

The HSSL driver does not provide any callout.

1.3.8 Error Handling

This section describes the various errors reported by the HSSL driver.

Error Name: Description	Source	Error ID (AS422)	Type (AS422)	Error ID (AS440)	Type (AS440)
HSSL_E_NOT_INITIALIZED:API service is called before initialization API Hssl_Init.	IFX	0x01	DET	0x01	DET
HSSL_E_INV_POINTER: Service is called with NULL or Invalid pointer.	IFX	0x02	DET	0x02	DET
HSSL_E_INV_PARAM: Service is called with invalid parameter.	IFX	0x03	DET	0x03	DET
HSSL_E_INV_MODE: Service is called in an Invalid driver mode.	IFX	0x04	DET	0x04	DET
HSSL_E_INSTANCE_NOT_CONFIGURED: Service is called with unconfigured Hssl Instance.	IFX	0x05	DET	0x05	DET

1.3.9 Deviations and limitations

This section describes the deviations and limitations of the HSSL driver.

1.3.9.1 Deviations

This section describes the deviations of the HSSL driver.

1.3.9.1.1 Software specification deviations

The HSSL driver does not have any deviations.

1.3.9.1.2 AMDC violations

The HSSL driver does not have any AMDC violations.

1.3.9.1.3 VSMD violations

The HSSL driver does not have any VSMD violations.

1.3.9.2 Limitations

The section describes the limitations of the HSSL driver.

Table 75 Known limitations

Reference	Limitation
Handling OS calls invoked by HSSL Interrupt service routine in CAT1 context	If the runtime API mode (HsslRuntimeApiMode) is configured to HSSL_MCAL_USER1, the HSSL interrupt handler uses OS service to access supervisor privileged SFRs. Due to this, if the HSSL interrupt handlers are invoked in CAT1 context, the application software must handle the OS service call invoked from HSSL handler.
Due to unreliability of the wake-up functionality, sleep mode for the HSCT is no longer supported.	Use HSSL_SetMode API to set HSSL module to only INIT or RUN mode.

Revision history

Major changes since the last revision.

Date	Version	Description
2021-03-03	3.0	Document is released
2021-02-26	2.1	• Updated limitations section • Updated specification for Hssl_SetMode API table
2020-11-27	2.0	Document is released
2020-11-26	1.1	• Error handling format of all the APIs updated in Functions - APIs section • Error handling section format updated
2020-08-13	1.0	Document is released
2020-08-10	0.1	• Initial version • HSSL driver chapter moved from TC3xx_SW_MCAL_UM_DEMO to this document. • Updated Development Errors table. • Added hints for DMA error handling.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2021-03-03

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-twu1596784670487

IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffenheitsgarantie").

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.