Application Report

Peter Li, Joe Shen, and Karan Saxena

ABSTRACT

This application report focuses on Texas Instruments' System Control Interface (TISCI) server integration in Vector AUTOSAR for the Jacinto™ 7 family of devices. The content covered is applicable for systems running AUTOSAR on MCU R5F with SDK7.1 or later.

Table of Contents

Trademarks

Jacinto™ is a trademark of Texas Instruments.
Arm® and Cortex® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
All trademarks are the property of their respective owners.

1 Introduction

1.1 SYSFW

Jacinto 7 devices introduce the concept of a centralized Power, Resource and Security Management to allow mitigating the challenges of the traditional approach of distributed system control.

SYSFW is a collective term used to describe the TI Foundational Security (TIFS) and Resource Management (RM)/Power Management (PM) services. For more information on this, see the publicly available documentation.

1.2 TISCI

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

TISCI protocol is used to talk to the SYSFW. For more information on this, see the TISCI user's guide.

1.3 TISCI Client or SciClient

The SciClient is an interface to the TISCI protocol for RTOS and non-OS based applications. It exposes the core message details, valid module/clock IDs to the higher-level software and abstracts the communication with SYSFW based on the TISCI protocol.

For more details on this, see the SciClient documentation in the SDK.

1.4 TISCI Server or SciServer

The Sciserver is a new module that enables MCU R5F to service Device Management requests from all non-secure entities on the SoC. DMSC only services requests by secure entities.

1.5 Acronyms Used in This Document

Acronym	Description
J7	Jacinto 7
SYSFW	System Firmware
TISCI	Texas Instruments System Controller Interface
TIFS	Texas Instruments Foundational Security
DM	Device Manager
RM	Resource Management
PM	Power Management
HSM	Hardware Security Modules
DMSC	Device Management and Security Controller
MCU	Micro Controller Unit
MCU R5F	Arm Cortex-R5F in the MCU domain
CSL	Chip Support Library
PDK	Platform Development Kit
SDK	Software Development Kit
TCM	Tightly-Coupled Memory
M3	Arm Cortex-M3
R5F	Arm Cortex-R5F

2 Target Audience

SYSFW executes on the Security Manager and Device Manager Core. On J7 devices, the DMSC subsystem is the Security Manager Core (Arm® Cortex®-M3) in the SoC. The Device Manager (DM) core however can be on the DMSC itself or the MCU R5F. The choice of whether this runs on the DMSC or the MCU R5F is based on the kind of applications the device is targeting.

For devices that do not need a dedicated security island or HSM, the device manager runs on the DMSC. If the device is targeting applications, where there is a need for a dedicated HSM, the Device manager core is independent of the DMSC (which is exclusively kept for Security functions).

Table 2-1. Missing Title

Category	Device	TIFS Core	Dedicated Security Island or HSM?	3rd Party HSM/SHE Stack Core	RM/PM (DM) core
1	DRA829/TDA4VM	DMSC (M3)	NO	NA	DMSC (M3)
2	DRA829/TDA4VM	DMSC (M3)	YES	DMSC (Cortex-M3)	Library on MCU R5F

The intention of this application report is to cater to the second category mentioned above.

3 MCU1_0 Role in Jacinto7 SDK V7.1+

For TI Jacinto7 SDK V7.1 and onwards, the MCU1_0 acts as the Resource Manager and Power Manager by running DM for the platform.

MCU1_0 uses sciclient_direct.aer5f for TISCI function calls. Other MCUs or DSPs uses sciclient_indirect libraries for TISCI functionalities related to RM/PM.

Sciserver or TISCI server needs to be hosted on MCU1_0 only to provide RM/PM functionalities to other cores in Jacinto 7 SOC.

4 TISCI Client and Server on TI-RTOS

TI has implemented the TISCI server on MCU1_0 in TI Jacinto7 TI-RTOS SDK. This can be taken as a reference for integrating the TISCI server functionalities to other RTOS, such as AUTOSAR.

For details on the implementation, see the SciClient documentation in the SDK.

4.1 J7 SDK Download

Platform	Device	Download Link
J721E	DRA829	https://www.ti.com/tool/download/PROCESSOR-SDK-RTOS-J721E#downloads
	TDA4VM	
J7200	DRA821	https://www.ti.com/tool/download/PROCESSOR-SDK-RTOS-J7200#downloads

4.2 SciClient Driver Location

Table 4-1 shows the driver code and public repo for SciClient:

Table 4-1. Driver Code and Public Repo for SciClient

Driver code	`$J7SDK/ti-processor-sdk-rtos-j721e-evm-xx_xx_xx_xx/pdk_jacinto_xx_xx_xx_xx/packages/ti/drv/sciclient`
Repo	PDK's public git

4.3 TISCI Server Initialization Example

The code snippet gives an example implementation of the SciClient and SciServer initialization.

```
sint32 SetupSciServer(void)
{
#if (defined (BUILD_MCU1_0) && (defined (SOC_J721E) || defined (SOC_J7200)))
    Sciserver_TirtosCfgPrms_t appPrms;
    Sciclient_ConfigPrms_t clientPrms;
    sint32 ret = CSL_PASS;

    appPrms.taskPriority[SCISERVER_TASK_USER_LO] = 1;
    appPrms.taskPriority[SCISERVER_TASK_USER_HI] = 4;

    /* Sciclient needs to be initialized before Sciserver. Sciserver depends on
     * Sciclient API to execute message forwarding */
    ret = Sciclient_configPrmsInit(&clientPrms);
    if (ret == CSL_PASS)
    {
        ret = Sciclient_init(&clientPrms);
    }

    if (ret == CSL_PASS)
    {
        ret = Sciserver_tirtosInit(&appPrms);
    }

    if (ret == CSL_PASS)
    {
        AppUtils_Printf(MSG_NORMAL, "Starting Sciserver..... PASSED\n");
    }
    else
    {
        AppUtils_Printf(MSG_NORMAL, "Starting Sciserver..... FAILED\n");
    }

#endif
    return ret;
}
```

Example code for **Sciserver_tirtosInit**() can be found in **$J7SDK/ti-processor-sdk-rtos-j721e-evm-xx_xx_xx_xx/pdk_jacinto_xx_xx_xx_xx/packages/ti/drv/sciclient/src/sciserver/sciserver_tirtos.c** in PDK's public GIT here.

4.4 Integration Guide

Due to the difference between AUTOSAR and TI-RTOS, there are several implementations which need to be ported.

In cases where the MCU1_0 is not running TI-RTOS but an OS like AUTOSAR, the application developer should use the below library and also create, build and link an equivalent file to sciserver_tirtos.c.

For the example, as shown in Section 4.3, the function `ret = Sciserver_tirtosInit(&appPrms);` should be implemented with the target OS provided APIs. Customers can re-name the Sciserver_tirtosInit() function to a more appropriate name as per their OS.

The OS components mentioned below are needed in the implementation of **Sciserver_tirtosInit**(). In the SDK the target OS is TI RTOS, hence, the below discussions are based on TI-RTOS. In subsequent sections we will see how to port this to AUTOSAR.

4.4.1 Semaphore

In TI-RTOS, semaphores are used for:

- Triggering the SciServer User tasks execution
- Synchronizing between the two SciServer User tasks

The code for initializing the semaphores can be found at **$J7SDK/ti-processor-sdk-rtos-j721e-evm-xx_xx_xx_xx/pdk_jacinto_xx_xx_xx_xx/packages/ti/drv/sciclient/src/sciserver/sciserver_tirtos.c** function**Sciserver_tirtosInitSemaphores**() and PDK's public GIT here.

4.4.2 Interrupts Registration

There are four interrupts used for SciServer User tasks processing.

The following table provides details of the interrupts. these four interrupts should be reserved for SciServer on MCU R5F.

	IRQ	Description
1	70	Request from MCU domain navigation system with high priority
2	71	Request from Main domain navigation system with high priority
3	72	Request from MCU domain navigation system with normal priority
4	73	Request from Main domain navigation system with normal priority

The code for initializing the Hwis can be found at **$J7SDK/ti-processor-sdk-rtos-j721e-evm-xx_xx_xx_xx/pdk_jacinto_xx_xx_xx_xx/packages/ti/drv/sciclient/src/sciserver/sciserver_tirtos.c** function **Sciserver_tirtosInitHwis ()** and PDK's public GIT here.

4.4.3 Interrupts Handling

The code for interrupt handler for SciServer interrupts can be found at **$J7SDK/ti-processor-sdk-rtos-j721e-evm-xx_xx_xx_xx/pdk_jacinto_xx_xx_xx_xx/packages/ti/drv/sciclient/src/sciserver/sciserver_tirtos.c** function **Sciserver_tirtosUserMsgHwiFxn ()** and PDK's public GIT here.

4.4.4 User Tasks Registration

There are two user tasks with different priorities: one is to handle high priority TISCI requests and the other is to handle normal priority TISCI requests.

The code for registration of these user tasks can be found at **$J7SDK/ti-processor-sdk-rtos-j721e-evm-xx_xx_xx_xx/pdk_jacinto_xx_xx_xx_xx/packages/ti/drv/sciclient/src/sciserver/sciserver_tirtos.c** function **Sciserver_tirtosInitUserTasks ()** and PDK's public GIT here.

4.4.5 User Tasks Processing

The user task with higher priority processes MCU domain navigation system high priority requests, and also Main domain navigation system high priority requests. While the task with normal priority processes MCU domain navigation system normal priority requests, and also Main domain navigation system normal priority requests.

The code for processing of these user tasks can be found at **$J7SDK/ti-processor-sdk-rtos-j721e-evm-xx_xx_xx_xx/pdk_jacinto_xx_xx_xx_xx/packages/ti/drv/sciclient/src/sciserver/sciserver_tirtos.c** function **Sciserver_tirtosUserMsgTask ()** and PDK's public GIT here.

5 Configurations in DaVinci

5.1 DaVinci Developer

DaVinci Developer is a tool for designing the architecture of software components (SWCs) for AUTOSAR ECUs.

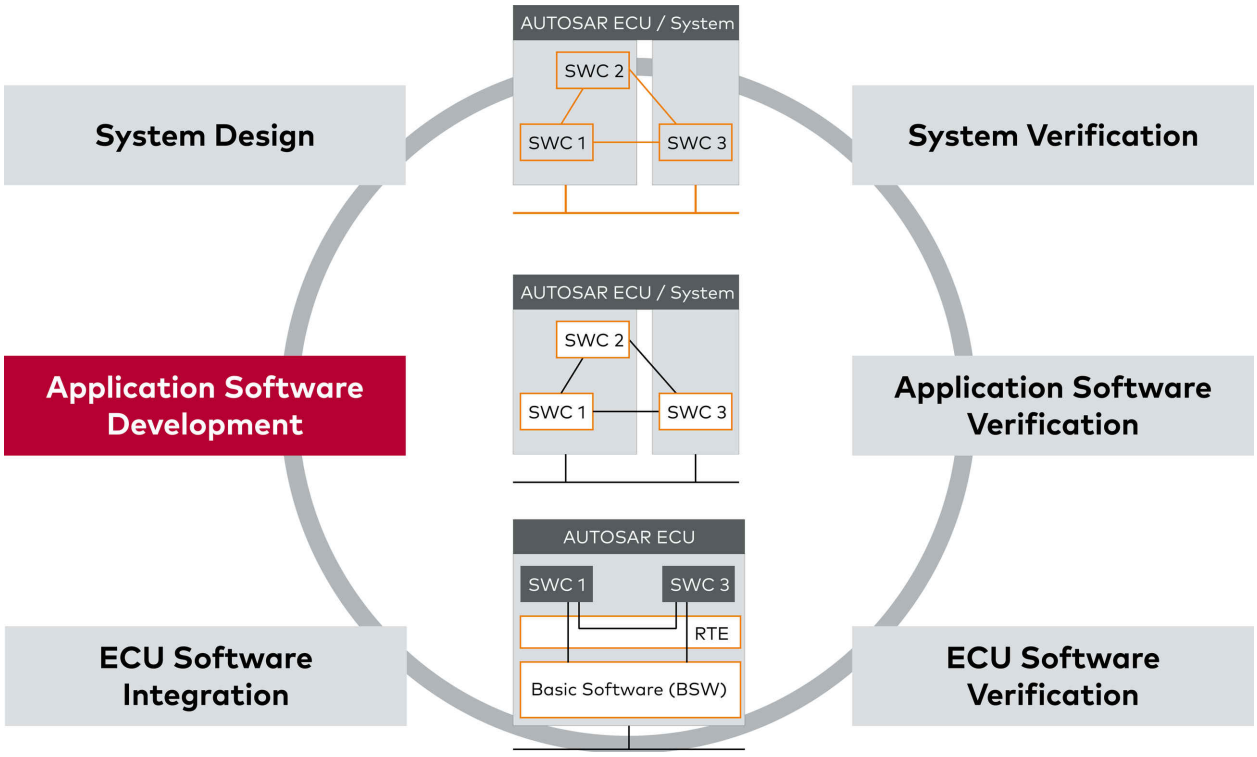

Figure 5-1. Circle Model of Typical AUTOSAR Projects

The DaVinci Developer is used in the phase Application Software Development.

5.2 DaVinci Configurator Pro

DaVinci Configurator Pro is the central tool for configuring, validating and generating the basic software (BSW) and the runtime environment (RTE) of an AUTOSAR ECU.

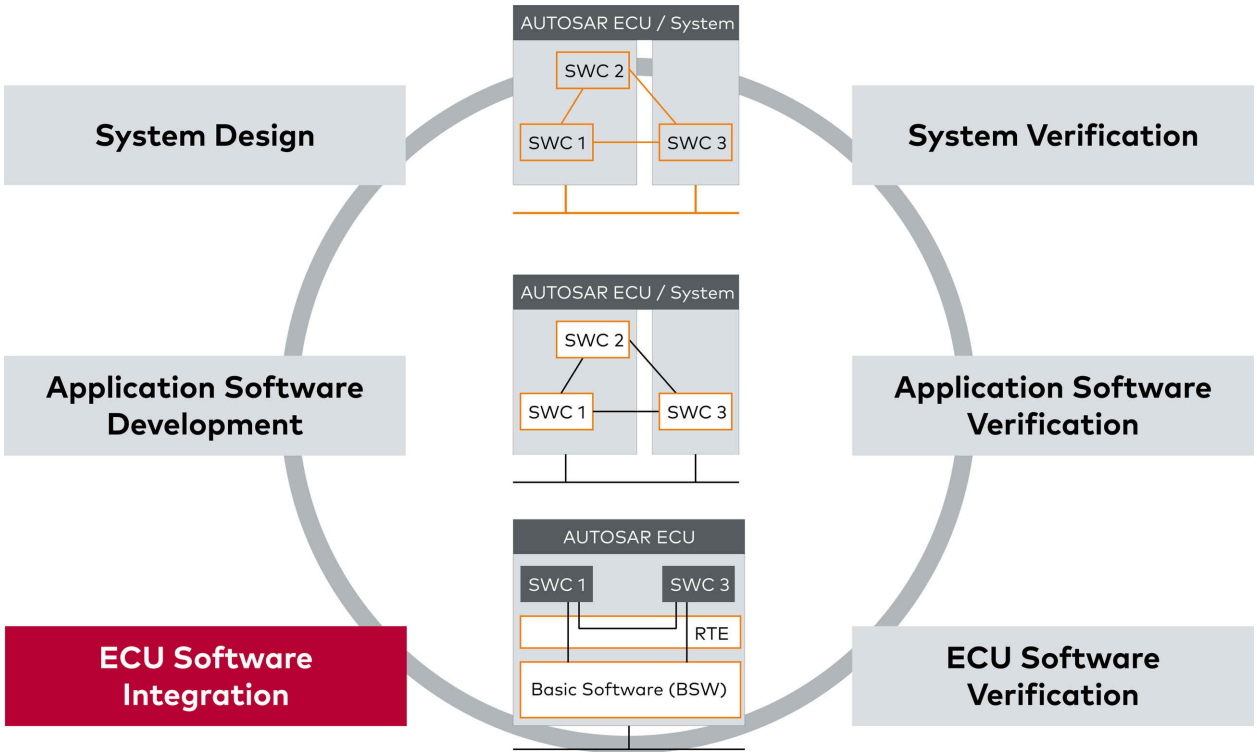

Figure 5-2. Circle Model of Typical AUTOSAR Projects

The DaVinci Configurator Pro is used in the phase of ECU software integration.

More information can be found at:

https://www.vector.com/int/en/products/products-a-z/software/davinci-configurator-pro/

5.3 Resource

An OsResource object is used to co-ordinate the concurrent access by tasks and ISRs to a shared resource, for example, the scheduler, any program sequence, memory or any hardware area. More information can be found from AUTOSAR website.

This resource is used to sync between the two SciServer user tasks.

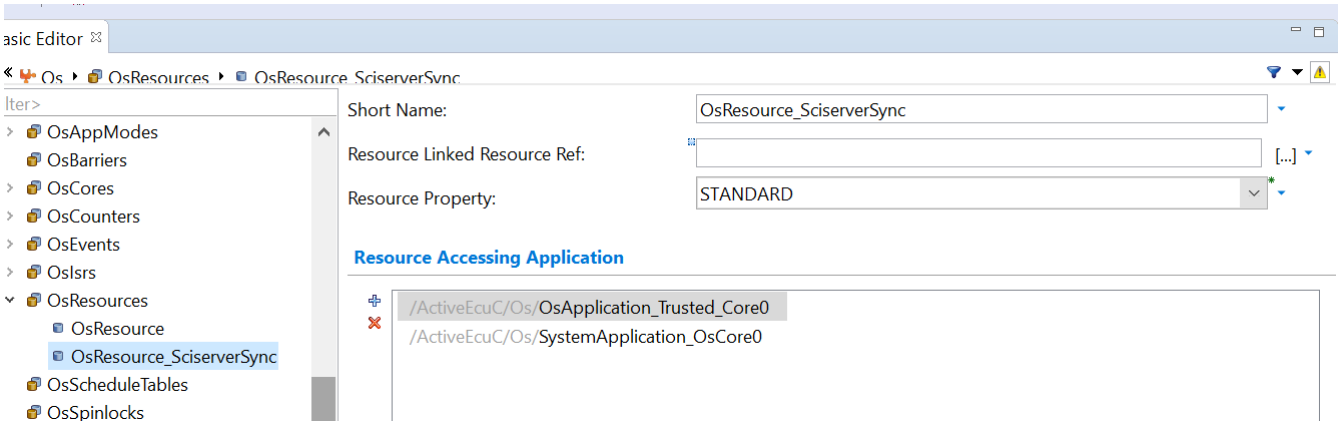

Figure 5-3. Resource Configuration

5.4 Events

Since there is not a semaphore in AUTOSAR, Events are used to trigger SciServer user tasks execution from hardware interrupts.

5.4.1 Event for High Priority Requests

This event triggers the execution of the SciServer user task that processes high priority requests.

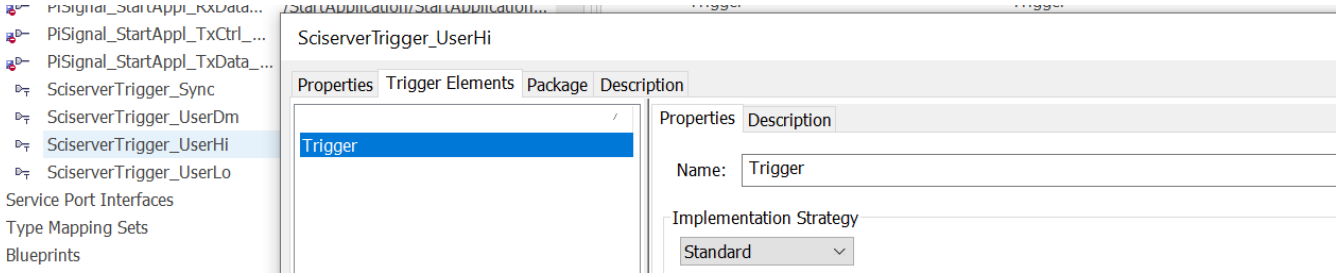

Figure 5-4. Event Configuration for High Priority Requests

5.4.2 Event for Normal Priority Requests

This event triggers the execution of the SciServer user task that processes normal priority requests.

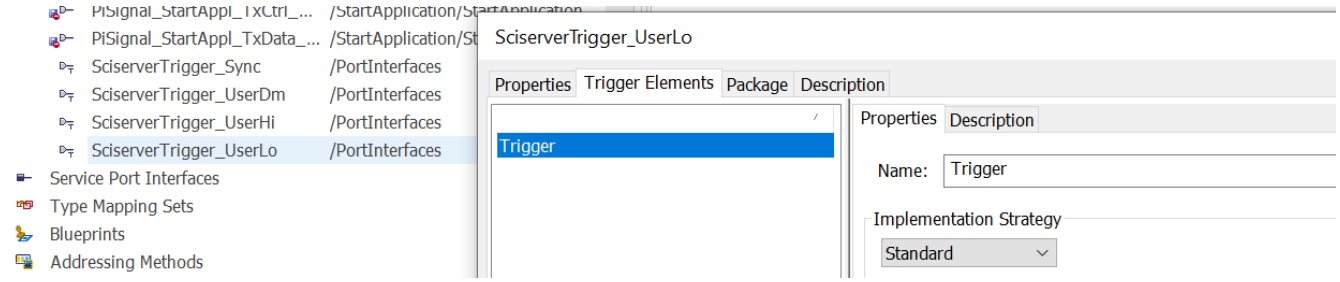

Figure 5-5. Event Configuration for Normal Priority Requests

5.5 SciServer User Tasks

5.5.1 High Priority User Task

This task processes the high priority requests from the MCU domain navigation system and the Main domain navigation system.

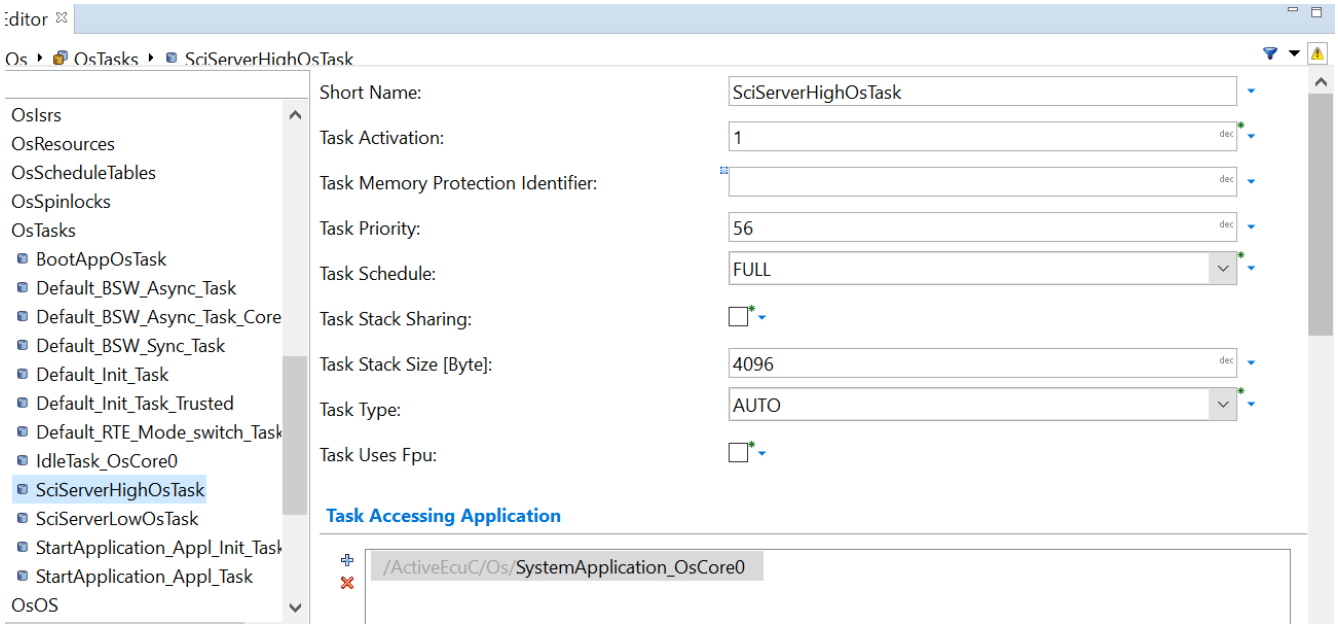

Figure 5-6. High Priority User Task Configuration

As described in Section 5.4.1, the Event SciserverTrigger_UserHi is used to trigger this task execution.

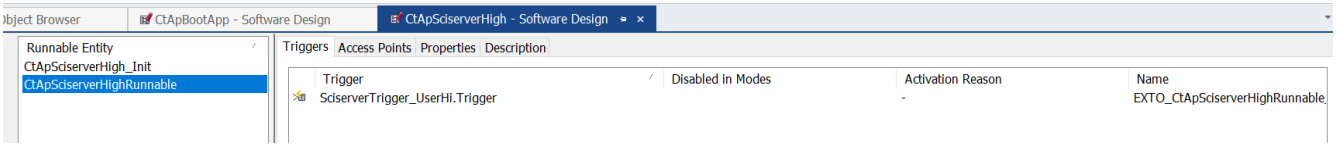

Figure 5-7. High Priority User Task Trigger

5.5.2 Normal Priority UserTask

This task processes the normal priority requests from MCU domain navigation system and Main domain navigation system.

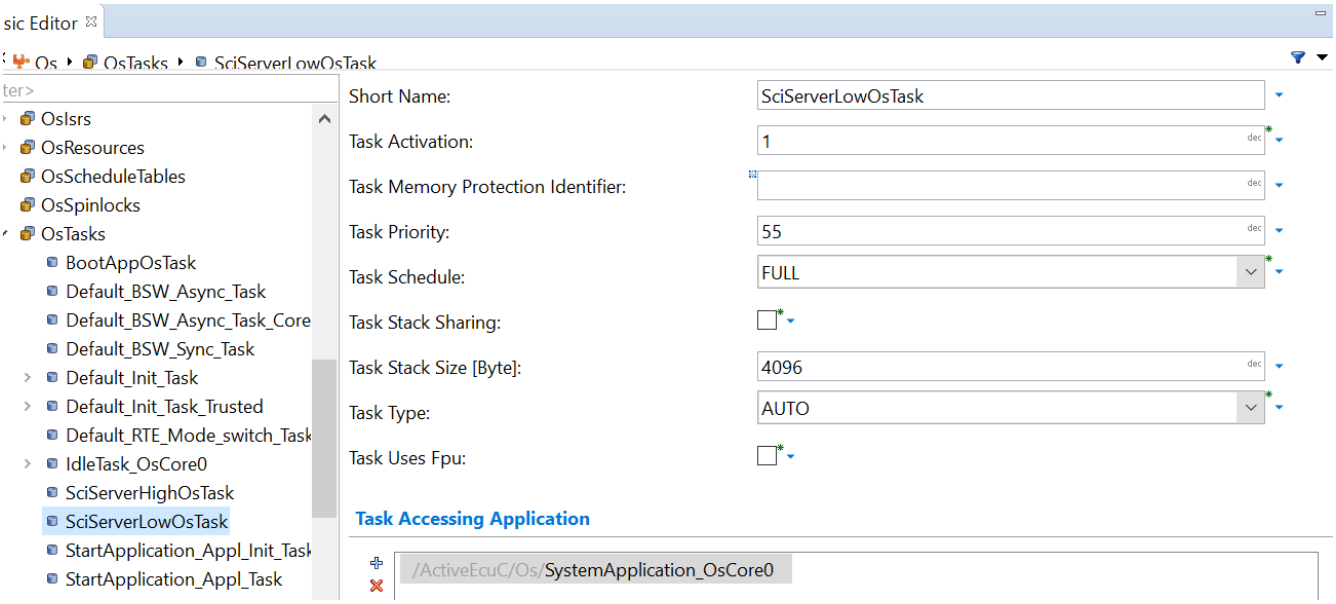

Figure 5-8. Normal Priority User Task Configuration

As described in Section 5.4.2, the Event SciserverTrigger_UserLo is used to trigger this task execution.

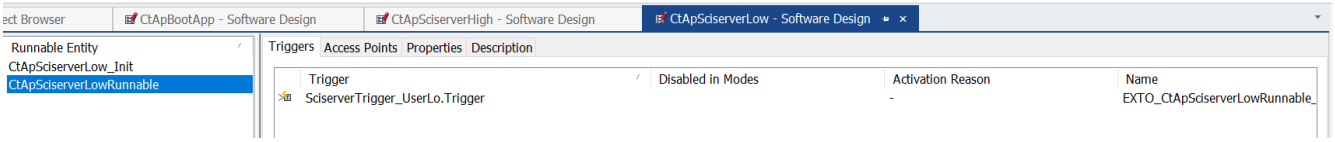

Figure 5-9. Normal Priority User Task Trigger

5.6 Synchronization Between Sciserver User Tasks

OSResource is used for the synchronization between the two Sciserver user tasks.

5.6.1 Configure the Resource for High Priority User Task

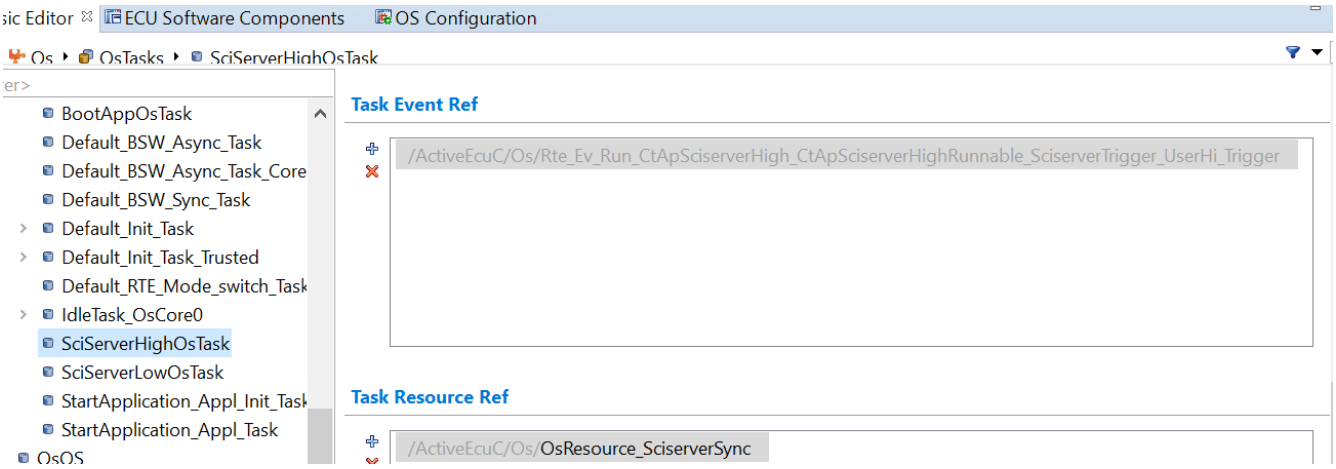

Figure 5-10. Resource Configuration for High Priority User Task

5.6.2 Configure the Resource for Normal Priority User Task

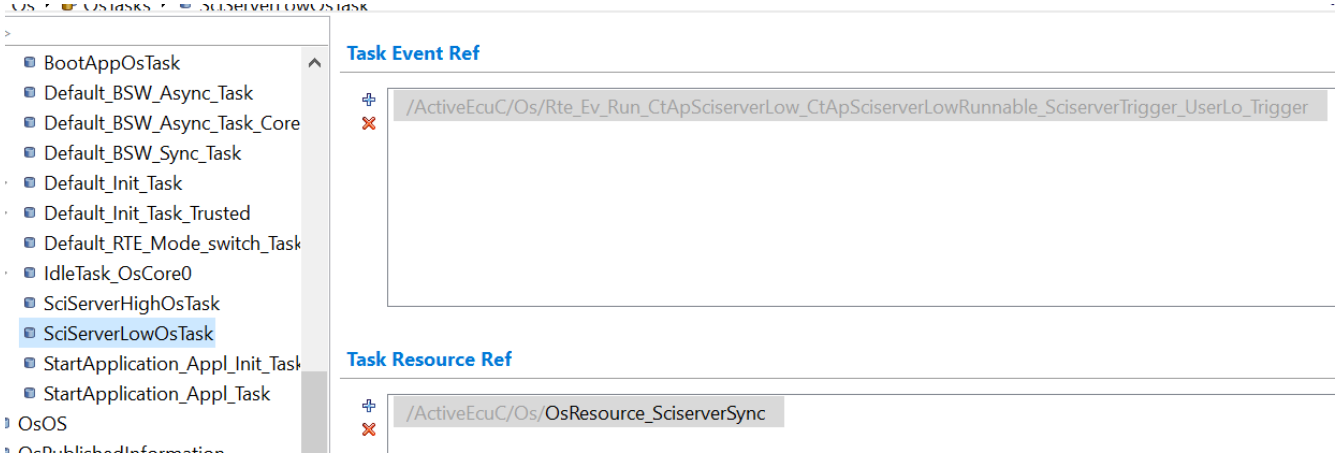

Figure 5-11. Resource Configuration for Normal Priority User Task

5.7 Sciserver Interrupts

As described in Section 5.4.2, there are four interrupts needed by Sciserver.

Below are the examples of registering these four interrupts in AUTOSAR.

5.7.1 MCU Domain Navigation System High Priority Interrupts

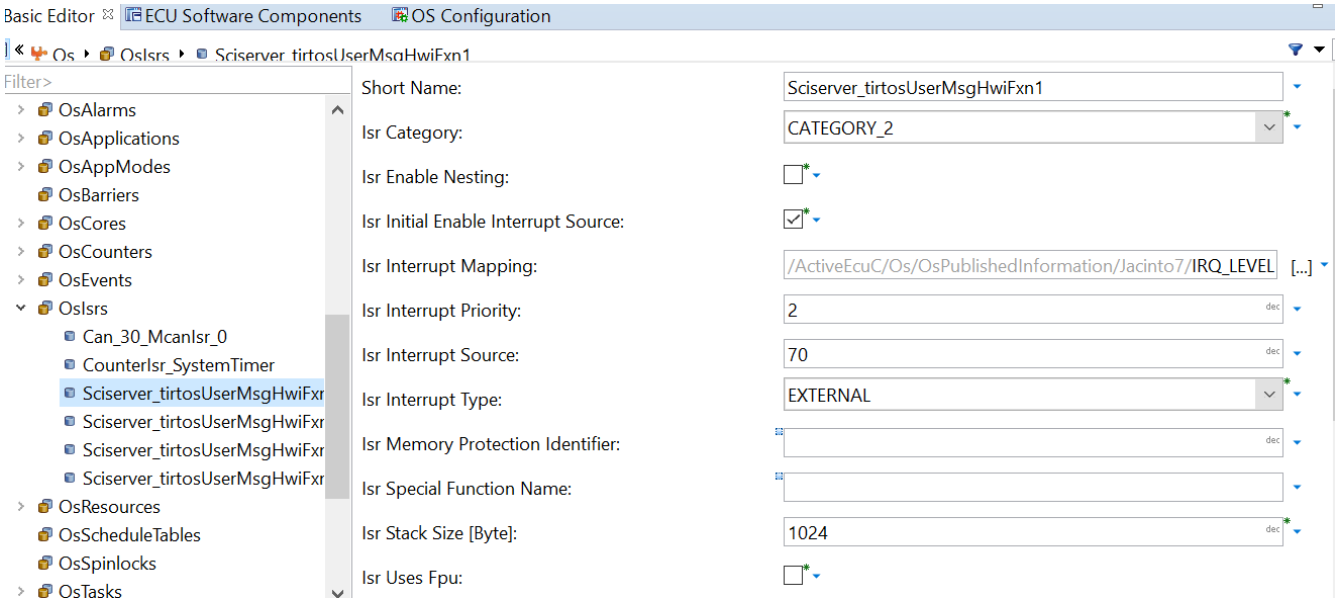

Figure 5-12. Interrupt Configuration for MCU Domain High Priority Interrupts

5.7.2 Main Domain Navigation System High Priority Interrupts

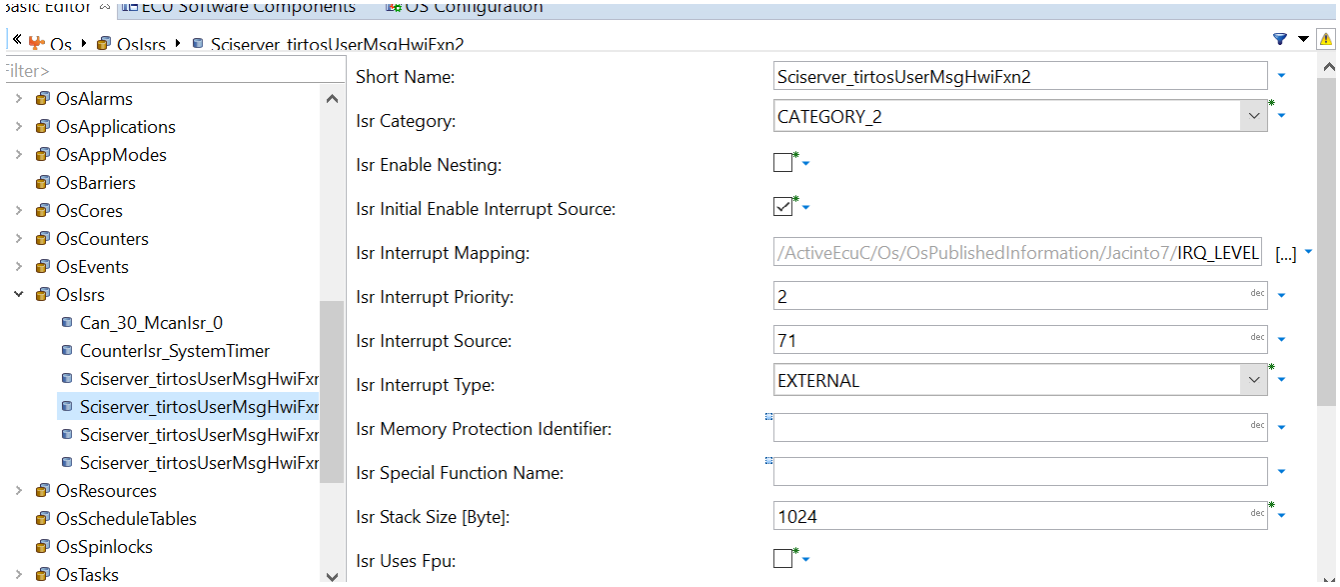

Figure 5-13. Interrupt Configuration for Main Domain High Priority Interrupts

5.7.3 MCU Domain Navigation System Normal Priority Interrupts

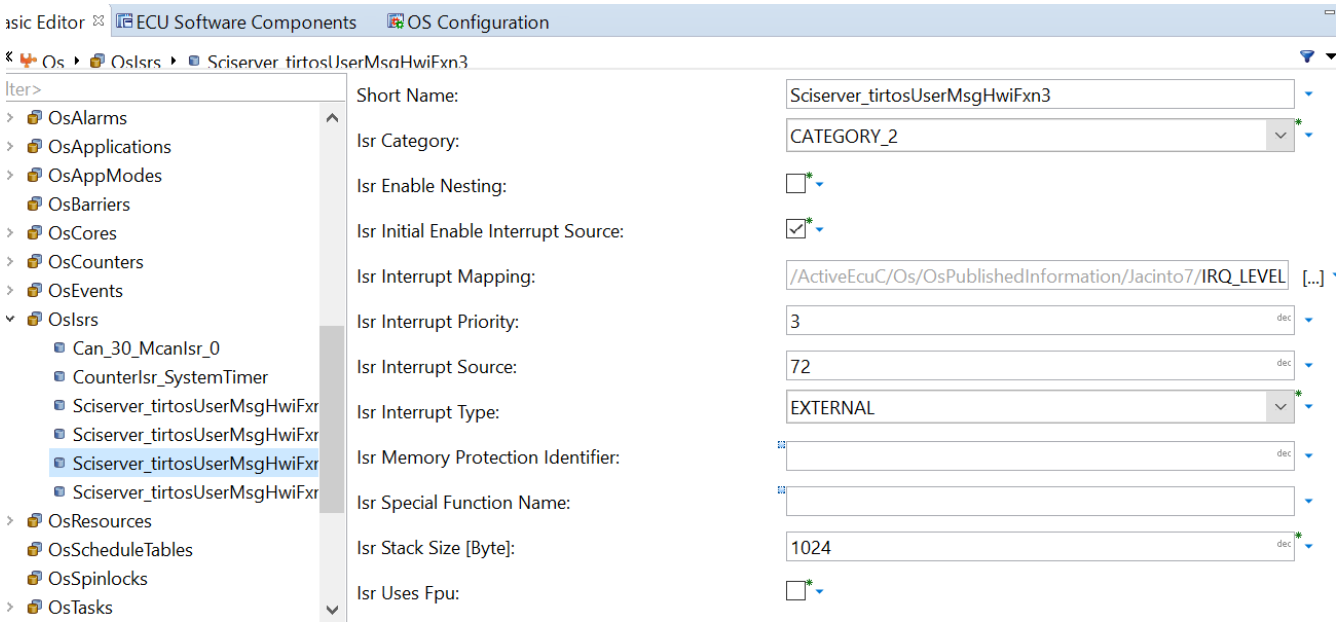

Figure 5-14. Interrupt Configuration for MCU Domain Normal Priority Interrupts

5.7.4 Main Domain Navigation System Normal Priority Interrupts

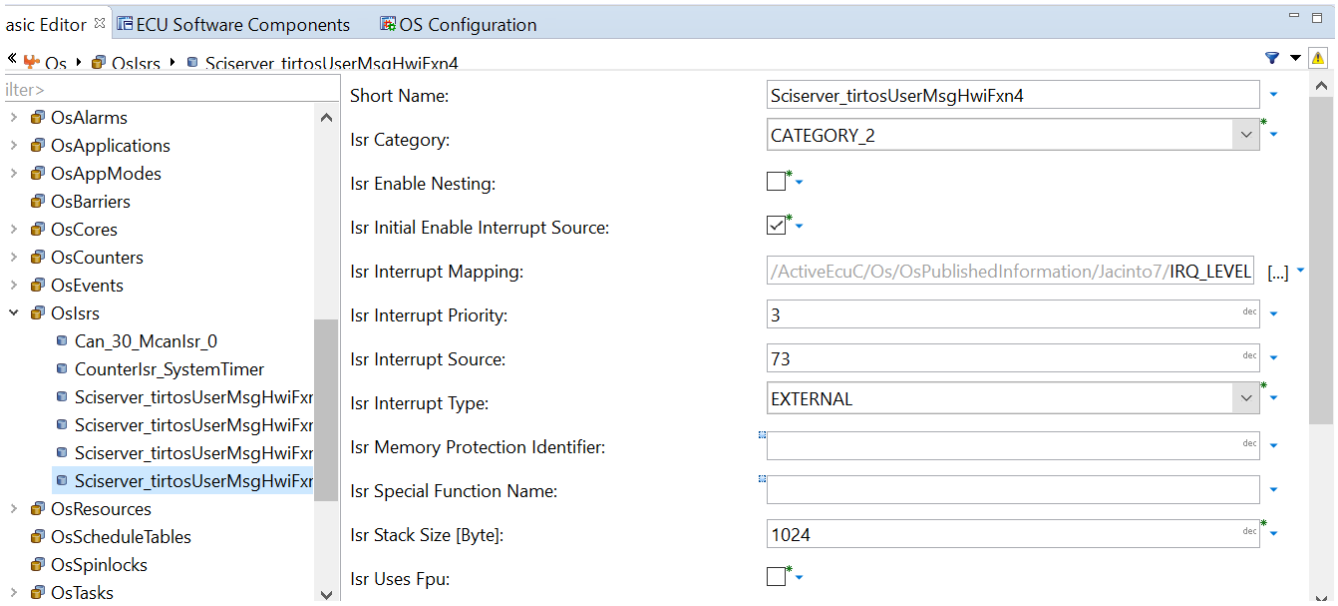

Figure 5-15. Interrupt Configuration for Main Domain Normal Priority Interrupts

6 AUTOSAR TISCI Client

As described in Section 1, each core in J7 SoC needs to be registered as SCI client to DMSC, so as to get SCI services from DMSC.

6.1 TISCI Client Registration in AUTOSAR

Below is the example of where we put the TISCI client registration (Board_sysInit()) in AUTOSAR.

```
void Brs_PreMainStartup(void)
{
  uint32 coreID;

  /* Relocate Vectors to ATCM, Please refer to Other Topics in this doc */
  memcpy((void *)0, (void *)_OS_EXCVEC_CORE0_CODE_START, _OS_EXCVEC_CORE0_CODE_LIMIT);

  /* some code are not shown here */

  Board_sysInit();

  main();
}
```

Where **Board_sysInit()** can be defined by taking reference from the implementation at **$J7SDK/ti-processor-sdk-rtos-j721e-evm-xx_xx_xx_xx/pdk_jacinto_xx_xx_xx_xx/packages/ti/board/src/j721e_evm/board_init.c** function**Board_sysInit()** and PDK's public GIT here.

```
static int Board_sysInit(void)
{
    int status = 0;
    int ret;
    Sciclient_ConfigPrms_t config;

    if(gBoardSysInitDone == 0)
    {
        Sciclient_configPrmsInit(&config);

        ret = Sciclient_init(&config);
        if(ret != 0)
        {
            status = -1;
        }

        if(status == 0)
        {
            gBoardSysInitDone = 1;
        }
    }

    return status;
}
```

7 AUTOSAR TISCI Interrupts Handling

The direct register access shown below should be replaced with AUTOSAR specific equivalent interrupt handling APIs.

Example implementation in TI-RTOS is shown in comment lines for comparison and better understanding.

7.1 MCU Domain Navigation System High Priority Interrupts

The registration of this interrupt is shown in Section 5.7.1.

```
/* user_mcu_nav_high_priority */
ISR(Sciserver_tirtosUserMsgHwiFxn1)
{
    Sciserver_hwiData *uhd =  &sciserver_hwi_list[USER_MCU_NAV_HIGH];
    int32_t ret = CSL_PASS;
    bool soft_error = false;

    /* TI RTOS: Osal_DisableInterrupt(0, (int32_t) uhd->irq_num); */
    *(volatile unsigned int *)(0x40F80000 + 0x400 +
        (CSLR_MCU_R5FSS0_CORE0_INTR_MCU_NAVSS0_INTR_ROUTER_0_OUTL_INTR_6/32)*0x20 + 0x0C) = 0x40;

    ret = Sciserver_interruptHandler(uhd, &soft_error);

    if ((ret != CSL_PASS) && (soft_error == true))
    {
        /* TI RTOS: Osal_EnableInterrupt(0, (int32_t) uhd->irq_num); */
        *(volatile unsigned int *)(0x40F80000 + 0x400 +
            (CSLR_MCU_R5FSS0_CORE0_INTR_MCU_NAVSS0_INTR_ROUTER_0_OUTL_INTR_6/32)*0x20 + 0x08) =
0x40;
    }
    else
    {
         /* TI RTOS: (void) SemaphoreP_post(gSciserverUserSemHandles[uhd->semaphore_id]); */
        (void)SetEvent(SciServerHighOsTask,
            Rte_Ev_Run_CtApSciserverHigh_CtApSciserverHighRunnable_SciserverTrigger_UserHi_Trigger);
    }

    /* TI RTOS: Osal_ClearInterrupt(0, (int32_t) uhd->irq_num); */
    *(volatile unsigned int *)(0x40F80000 + 0x400 +
        (CSLR_MCU_R5FSS0_CORE0_INTR_MCU_NAVSS0_INTR_ROUTER_0_OUTL_INTR_6/32)*0x20 + 0x04) = 0x40;

}
```

7.2 Main Domain Navigation System High Priority Interrupts

The registration of this interrupt is shown in Section 5.7.2.

```
/* user_main_nav_high_priority */
ISR(Sciserver_tirtosUserMsgHwiFxn2)
{
    Sciserver_hwiData *uhd = NULL;
    int32_t ret = CSL_PASS;
    bool soft_error = false;

    uhd =   &sciserver_hwi_list[USER_MAIN_NAV_HIGH];

    /* TI RTOS: Osal_DisableInterrupt(0, (int32_t) uhd->irq_num); */
    *(volatile unsigned int *)(0x40F80000 + 0x400 +
        (CSLR_MCU_R5FSS0_CORE0_INTR_MCU_NAVSS0_INTR_ROUTER_0_OUTL_INTR_7/32)*0x20 + 0x0C) = 0x80;

    ret = Sciserver_interruptHandler(uhd, &soft_error);

    if ((ret != CSL_PASS) && (soft_error == true))
    {
        /* TI RTOS: Osal_EnableInterrupt(0, (int32_t) uhd->irq_num); */
        *(volatile unsigned int *)(0x40F80000 + 0x400 +
            (CSLR_MCU_R5FSS0_CORE0_INTR_MCU_NAVSS0_INTR_ROUTER_0_OUTL_INTR_7/32)*0x20 + 0x08) =
0x80;
    }
    else
    {
        /* TI RTOS: (void) SemaphoreP_post(gSciserverUserSemHandles[uhd->semaphore_id]); */
        (void)SetEvent(SciServerHighOsTask,
            Rte_Ev_Run_CtApSciserverHigh_CtApSciserverHighRunnable_SciserverTrigger_UserHi_Trigger);
    }

    /* TI RTOS: Osal_ClearInterrupt(0, (int32_t) uhd->irq_num); */
    *(volatile unsigned int *)(0x40F80000 + 0x400 +
        (CSLR_MCU_R5FSS0_CORE0_INTR_MCU_NAVSS0_INTR_ROUTER_0_OUTL_INTR_7/32)*0x20 + 0x04) = 0x80;

}
```

7.3 MCU Domain Navigation System Normal Priority Interrupts

The registration of this interrupt is shown in Section 5.7.3.

```
/* user_mcu_nav_low_priority */
ISR(Sciserver_tirtosUserMsgHwiFxn3)
{
    Sciserver_hwiData *uhd =  &sciserver_hwi_list[USER_MCU_NAV_LOW];
    int32_t ret = CSL_PASS;
    bool soft_error = false;

    /* TI RTOS: Osal_DisableInterrupt(0, (int32_t) uhd->irq_num); */
    *(volatile unsigned int *)(0x40F80000 + 0x400 +
        (CSLR_MCU_R5FSS0_CORE0_INTR_MCU_NAVSS0_INTR_ROUTER_0_OUTL_INTR_8/32)*0x20 + 0x0C) = 0x100;

    ret = Sciserver_interruptHandler(uhd, &soft_error);

    if ((ret != CSL_PASS) && (soft_error == true))
    {
        /* TI RTOS: Osal_EnableInterrupt(0, (int32_t) uhd->irq_num); */
        *(volatile unsigned int *)(0x40F80000 + 0x400 +
            (CSLR_MCU_R5FSS0_CORE0_INTR_MCU_NAVSS0_INTR_ROUTER_0_OUTL_INTR_8/32)*0x20 + 0x08) =
0x100;
    }
    else
    {
        /* TI RTOS: (void) SemaphoreP_post(gSciserverUserSemHandles[uhd->semaphore_id]); */
        (void)SetEvent(SciServerLowOsTask,
            Rte_Ev_Run_CtApSciserverLow_CtApSciserverLowRunnable_SciserverTrigger_UserLo_Trigger);
    }

    /* TI RTOS: Osal_ClearInterrupt(0, (int32_t) uhd->irq_num); */
    *(volatile unsigned int *)(0x40F80000 + 0x400 +
        (CSLR_MCU_R5FSS0_CORE0_INTR_MCU_NAVSS0_INTR_ROUTER_0_OUTL_INTR_8/32)*0x20 + 0x04) = 0x100;

}
```

7.4 Main Domain Navigation System Normal Priority Interrupts

The registration of this interrupt is shown in Section 5.7.4.

```
/* user_main_nav_low_priority */
ISR(Sciserver_tirtosUserMsgHwiFxn4)
{
    Sciserver_hwiData *uhd =  &sciserver_hwi_list[USER_MAIN_NAV_LOW];
    int32_t ret = CSL_PASS;
    bool soft_error = false;

    /* TI RTOS: Osal_DisableInterrupt(0, (int32_t) uhd->irq_num); */
    *(volatile unsigned int *)(0x40F80000 + 0x400 +
        (CSLR_MCU_R5FSS0_CORE0_INTR_MCU_NAVSS0_INTR_ROUTER_0_OUTL_INTR_9/32)*0x20 + 0x0C) = 0x200;

    ret = Sciserver_interruptHandler(uhd, &soft_error);

    if ((ret != CSL_PASS) && (soft_error == true))
    {
        /* TI RTOS: Osal_EnableInterrupt(0, (int32_t) uhd->irq_num); */
        *(volatile unsigned int *)(0x40F80000 + 0x400 +
            (CSLR_MCU_R5FSS0_CORE0_INTR_MCU_NAVSS0_INTR_ROUTER_0_OUTL_INTR_9/32)*0x20 + 0x08) =
0x200;
    }
    else
    {
        /* TI RTOS: (void) SemaphoreP_post(gSciserverUserSemHandles[uhd->semaphore_id]); */
        (void)SetEvent(SciServerLowOsTask,
            Rte_Ev_Run_CtApSciserverLow_CtApSciserverLowRunnable_SciserverTrigger_UserLo_Trigger);
    }

    /* TI RTOS: Osal_ClearInterrupt(0, (int32_t) uhd->irq_num); */
    *(volatile unsigned int *)(0x40F80000 + 0x400 +
        (CSLR_MCU_R5FSS0_CORE0_INTR_MCU_NAVSS0_INTR_ROUTER_0_OUTL_INTR_9/32)*0x20 + 0x04) = 0x200;

}
```

8 AUTOSAR TISCI User Tasks Processing

The direct register access shown below should be replaced with AUTOSAR specific equivalent interrupt handling APIs.

Example implementation in TI-RTOS is shown in comment lines for comparison and better understanding.

8.1 High Priority User Task Initialization

The creation of this task is shown in Section 5.5.1.

```
static Sciserver_taskData *utdHigh = NULL;
static volatile int highIsrEnableVal = 0;

FUNC(void, CtApSciserverHigh_CODE) CtApSciserverHigh_Init(void)
{
    utdHigh = &gSciserverTaskList[SCISERVER_TASK_USER_HI];

    /* Set the pending State first */
    utdHigh->state->state = SCISERVER_TASK_PENDING;
}
```

8.2 High Priority User Task Runnable

```
FUNC(void, CtApSciserverHigh_CODE) CtApSciserverHighRunnable(void)
{
    sint32 ret;

    GetResource(OsResource_SciserverSync);

    ret = Sciserver_processtask(utdHigh);
    if (ret != CSL_PASS)
    {
        /* Failed to process message and failed to send nak response */
        /* TI-RTOS: BIOS_exit(0); */
        ReleaseResource(OsResource_SciserverSync);
       (void)TerminateTask();
    }
    else
    {
        /* TI-RTOS:
        Osal_EnableInterrupt(0, sciserver_hwi_list[2U * utd->task_id +
               utd->state->current_buffer_idx].irq_num);  */
        highIsrEnableVal = 1 << ((sciserver_hwi_list[2U * utdHigh->task_id + utdHigh->state-
>current_buffer_idx].irq_num) % 32);
        *(volatile unsigned int *)(0x40F80000 + 0x400 +
            (CSLR_MCU_R5FSS0_CORE0_INTR_MCU_NAVSS0_INTR_ROUTER_0_OUTL_INTR_7/32)*0x20 + 0x08) =
highIsrEnableVal;
    }

    ReleaseResource(OsResource_SciserverSync);
}
```

8.3 Normal Priority User Task Initialization

The creation of this task is shown in Section 5.5.2.

```
static Sciserver_taskData *utdLow = NULL;
static volatile int lowIsrEnableVal = 0;

FUNC(void, CtApSciserverLow_CODE) CtApSciserverLow_Init(void)
{
    utdLow = &gSciserverTaskList[SCISERVER_TASK_USER_LO];

    /* Set the pending State first */
    utdLow->state->state = SCISERVER_TASK_PENDING;
}
```

8.4 Normal Priority User Task Runnable

```
FUNC(void, CtApSciserverLow_CODE) CtApSciserverLowRunnable(void)
{
    sint32 ret;

    GetResource(OsResource_SciserverSync);

    ret = Sciserver_processtask(utdLow);
    if (ret != CSL_PASS)
    {
        /* Failed to process message and failed to send nak response */
        /* TI-RTOS: BIOS_exit(0); */
        ReleaseResource(OsResource_SciserverSync);
       (void)TerminateTask();
    }
    else
    {
        /* TI-RTOS:
           Osal_EnableInterrupt(0, sciserver_hwi_list[2U * utd->task_id +
                utd->state->current_buffer_idx].irq_num);  */
        lowIsrEnableVal = 1 << ((sciserver_hwi_list[2U * utdLow->task_id + utdLow->state-
>current_buffer_idx].irq_num) % 32);
        *(volatile unsigned int *)(0x40F80000 + 0x400 +
            (CSLR_MCU_R5FSS0_CORE0_INTR_MCU_NAVSS0_INTR_ROUTER_0_OUTL_INTR_8/32)*0x20 + 0x08) =
lowIsrEnableVal;
    }
```

```
    ReleaseResource(OsResource_SciserverSync);
}
```

9 TISCI Server Validation in AUTOSAR

9.1 Boot App

Boot App is an application used by MCU1_0 to boot other cores in the SoC.

Documentation can be found in the Jacinto7 SDK: MCUSW boot app.

If TISCI client and TISCI server are implemented correctly in the AUTOSAR OS, MCU1_0 should be able to successfully load other core images and boot them.

9.2 Boot Task Configuration

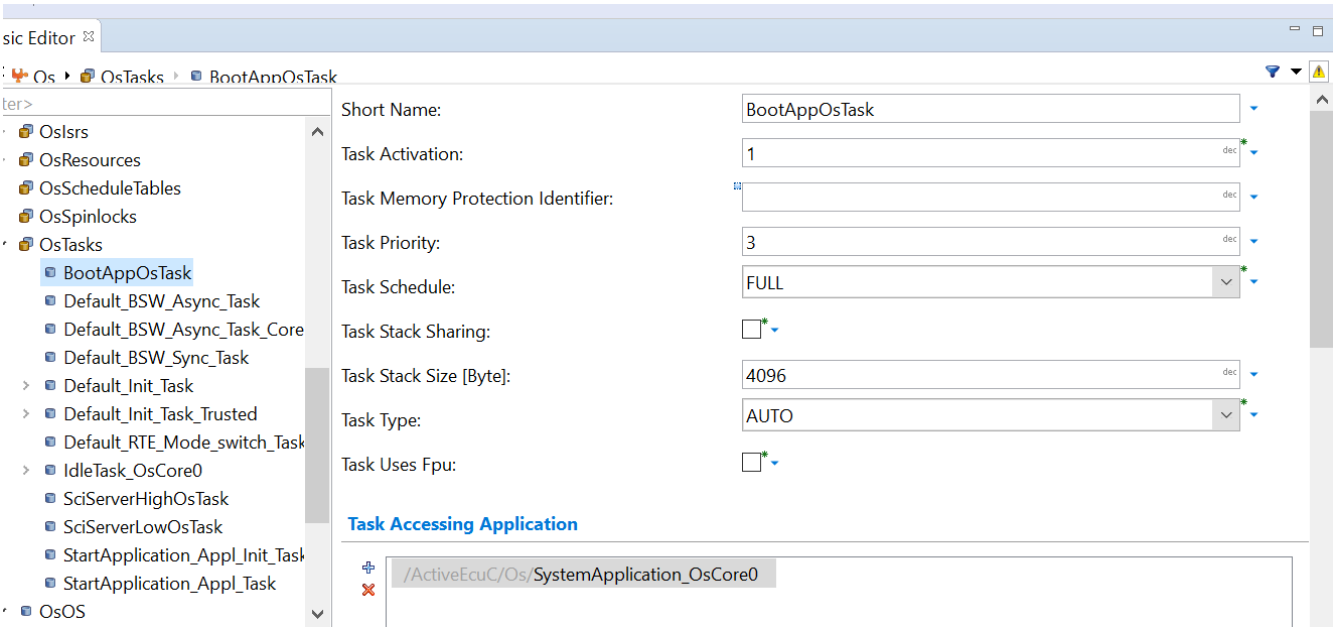

Figure 9-1. Boot Task Configuration

Boot task runnable is not needed.

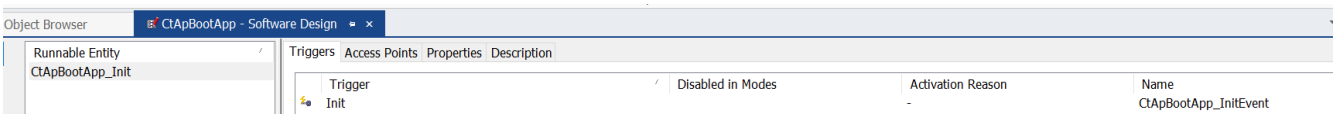

Figure 9-2. Boot Task Trigger

9.3 Boot App in AUTOSAR

9.3.1 Boot App Launch

```
FUNC(void, CtApBootApp_CODE) CtApBootApp_Init(void)
{
    Boot_App();
}
```

9.3.2 Boot App Implementation

The code for "**Boot_App**()" can be found at **$J7SDK/ti-processor-sdk-rtos-j721e-evm-xx_xx_xx_xx/mcusw/mcuss_demos/boot_app_mcu_rtos/boot.c** function **Boot_App ()**.

Below are code snippets for the Boot_App function.

```
/* Main Boot task */
static sint32 Boot_App()
{
    sint32          retVal;
    cpu_core_id_t   core_id, *boot_array;
    uint8           i, j, num_cores_to_boot, num_booted_cores = 0;

    MainDomainBootSetup();
    SBL_SPI_init();
    SBL_ospiInit(&boardHandle);
    /* Initialize the entry point array to 0. */
    for (core_id = MPU1_CPU0_ID; core_id < NUM_CORES; core_id ++) {
        (&k3xx_evmEntry)->CpuEntryPoint[core_id] = SBL_INVALID_ENTRY_ADDR;
    }
    for (j = 0; j < NUM_BOOT_STAGES; j++) {
        retVal = RequestStageCores(j);
        if (retVal != CSL_PASS) {
            ReleaseStageCores(j);
        } else {
            retVal = OSPIBootStageImage(&k3xx_evmEntry, ospi_main_domain_flash_rtos_images[j]);
            if (retVal != CSL_PASS){
            } else {
                retVal = ReleaseStageCores(j);
                if (retVal != CSL_PASS) {
                }
            }
        } /* if (retVal != CSL_PASS) */
        if (retVal == CSL_PASS) {
            /* Start the individual cores for the boot stage */
            num_cores_to_boot = num_cores_per_boot_stage[j];
            boot_array        = boot_array_stage[j];

            for (i = 0; i < num_cores_to_boot; i++) {
                core_id = boot_array[i];
                /* Try booting all cores other than the cluster running the SBL */
                if ((k3xx_evmEntry.CpuEntryPoint[core_id] != SBL_INVALID_ENTRY_ADDR) &&
                    ((core_id != MCU1_CPU1_ID) && (core_id != MCU1_CPU0_ID))) {
                    SBL_SlaveCoreBoot(core_id, NULL, &k3xx_evmEntry,SBL_REQUEST_CORE);
                    num_booted_cores++;
                }
            }
        } /* if (retVal == CSL_PASS) */
    } /* for (j = 0; j < NUM_BOOT_STAGES; j++) */

    SBL_ospiClose(&boardHandle);

    return (retVal);
}
```

10 PDK Libraries Used in AUTOSAR

Below are the library binaries used in AUTOSAR, which are from the PDK of TI J7 SDK.

```
PDK Path:
        $J7SDK/ti-processor-sdk-rtos-j721e-evm-xx_xx_xx_xx/pdk_jacinto_xx_xx_xx_xx/packages
```

- rm_pm_hal.aer5f
- sbl_lib_cust.aer5f
- sciclient_direct.aer5f
- sciserver_baremetal.aer5f
- ti.board.aer5f
- ti.csl.aer5f
- ti.csl.init.aer5f
- ti.drv.spi.aer5f
- ti.osal.aer5f
- ipc_baremetal.aer5f

11 R5F Configurations Needed for AUTOSAR

11.1 Memory Layout for Cortex-R5F

Table 11-1 shows the ATCM and BTCM in the Cortex-R5F.

Table 11-1. ATCM and BTCM in the Cortex-R5F

Region Name	Start Address	End Address	Size
ARMSS_ATCM	0x0000 0000	0x0000 7FFF	32KB
ARMSS_BTCM	0x4101 0000	0x4101 7FFF	32KB

There are several check points for running AUTOSAR on Cortex-R5F.

- **ATCM usage:** R5 SPL does not enable MCU1_0's ATCM by default and, hence, if the AutoSAR application is being booted on MCU1_0 core from R5 SPL then ATCM shouldn't be used in the memory layout.
  - Startup_Code should be placed to BTCM in that case.
- **Startup_Code alignment:** should be 256 bytes aligned.

Example is shown below:

```
MEMORY
{
  OCMCRAM_Common : ORIGIN = 0x41C50000 , LENGTH = 0x00004000 /* 48 KiB */
  OCMCRAM_Common_NonCache : ORIGIN = 0x41C54000 , LENGTH = 0x00000400 /* 1024 Byte */
  OCMCRAM_Core0 : ORIGIN = 0x41C54400 , LENGTH = 0x00000800 /* 2048 Byte */
  OCMCRAM_Core1 : ORIGIN = 0x41C54C00 , LENGTH = 0x00000400 /* 1024 Byte */
  OCMCRAM_Core2 : ORIGIN = 0x41C55000 , LENGTH = 0x00000400 /* 1024 Byte */
  OCMCRAM_Core3 : ORIGIN = 0x41C55400 , LENGTH = 0x00000400 /* 1024 Byte */
  OCMCRAM_Core4 : ORIGIN = 0x41C55800 , LENGTH = 0x00000400 /* 1024 Byte */
  OCMCRAM_Core5 : ORIGIN = 0x41010000 , LENGTH = 0x00000400 /* 1024 Byte */
  DDR0 : ORIGIN = 0x41C55C00 , LENGTH = 0xA5000 /* 16 MiB */
}

  .Startup_Code : ALIGN(256)
  {
    _Startup_Code_START = .;
    *(.brsStartup)
    . = ALIGN(256);
    _Startup_Code_END = . - 1;
    _Startup_Code_LIMIT = .;
  } > OCMCRAM_Core5
```

11.2 R5F Cache Configuration

There are three places that have R5F cache configurations. The configuration in sbl_main.c is the appropriate one for AUTOSAR. This configuration is widely used in J7 SDK.

- SBL Cache configuration
  - **$SDK_Path/ti-processor-sdk-rtos-$platform-evm-$psdkra_ver/pdk_jacinto_$pdk_ver/packages/ti/boot/sbl/board/k3/sbl_main.c** as a part of the structure named **gCslR5MpuCfg**
- Boot app Cache configuration
  - **$SDK_Path/ti-processor-sdk-rtos-$platform-evm-$psdkra_ver/mcusw/boot_app_mcu_rtos_overrides/j721e/r5_mpu.xs.** This configuration gets included in the TI RTOS configuration file at **$SDK_Path/ti-processor-sdk-rtos-$platform-evm-$psdkra_ver/mcusw/boot_app_mcu_rtos_overrides/j721e/sysbios_r5f.cfg**.
- CSL Cache configuration (gets overridden by baremetal apps)
  - **$SDK_Path/ti-processor-sdk-rtos-$platform-evm-$psdkra_ver/pdk_jacinto_$pdk_ver/packages/ti/csl/arch/r5/src/startup/startup.c.**

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

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