Renesas RA Family

Exception Handling

Introduction

This application note explains how to handle exceptions on Renesas RA family MCUs with Arm® Cortex®-M cores for user applications using the Flexible Software Package (FSP). By properly managing the factors, location, and history of exception events that occur, you can increase debugging efficiency in the application development process and produce a robust system against exception events. This application note also describes a unique bus error monitoring system on Renesas RA family devices with Arm® Cortex®-M33 and Arm® Cortex®-M85 cores that support Arm® TrustZone® technology. The application project uses the Flexible Software Package (FSP) of the RA family, the GNU GCC compiler, and the integrated development environment e^2 studio IDE to demonstrate an exception handling flow for multiple possible faults.

Prerequisites

- Experience using Renesas e^2 studio IDE
- Experience using Flexible Software Package (FSP) for the RA Family.

Note: For developers who are new to the Renesas RA family of MCUs, we strongly recommend to start with Tutorial: Your First RA MCU Project – Blinky prior to trying out this application.

Required Resources

Target Hardware:

- Renesas RA Kit EK-RA6M3 (for RA6M3 with Arm® Cortex®-M4)
- Renesas RA Kit EK-RA2E1 (for RA2E1 with Arm® Cortex®-M23)
- Renesas RA Kit EK-RA6M5 (for RA6M5 with Arm® Cortex®-M33)
- Renesas RA Kit EK-RA8M1 (for RA8M1 with Arm® Cortex®-M85)

Note: When applying this application note to other MCUs, be sure to change the settings according to the MCU specifications and evaluate it carefully.

Development Tools and Software:

- e^2 studio IDE version 2023-10 (23.10.0) or later
- Renesas Flexible Software Package (FSP) version 5.1.0 or later
- GCC ARM Embedded Toolchain version 13.2.1.arm-13-7 or later
- Segger J-Link RTT Viewer version 7.92o or later

Using this Application Note

Section 1 covers the general overview of the exception model on Arm® Cortex®-M processor core.

Section 2 covers the Bus Error Monitoring System for Renesas implementation.

Section 3 covers the implementing method for user-defined exception handler on FSP-based projects.

Section 4 covers the debugging method of the occurred exception event.

Section 5 covers the demonstration of the application attached to this application note.

Notes on this document

In this document, the following names are used for Renesas devices equipped with ARM Cortex®-M core.

- **RA-CM4 devices**: RA family devices with a single Arm® Cortex®-M4 processor core based on Armv7-M architecture profile.
  Examples of target devices: RA4M1, RA6M3
- **RA-CM23 devices**: RA family devices with a single Arm® Cortex®-M23 processor core based on Armv8-M architecture profile (without Security Extension)
  Examples of target devices: RA2A1, RA2E1
- **RA-CM33 devices**: RA family devices with a single Arm® Cortex®-M33 processor core based on Armv8-M architecture profile (with Security Extension)
  Examples of target devices: RA4M3, RA6M5, RA6E1, RA6T2
- **RA-CM85 devices**: RA family devices with a single Arm® Cortex®-M85 processor core based on Armv8.1-M architecture profile (with Security Extension)
  Examples of target devices: RA8M1, RA8D1, RA8T1

Contents

1. Exception Model on Arm® Cortex®-M Processor Core

The Arm® Cortex®-M processor core supports various fault detection and user notification methods. When the processor detects faults, exceptions, and interrupts, the processor transitions the processor mode. Available processor modes on Arm® Cortex®-M processor core are as follows (Figure 1).

- Thread mode
  Executes application software.
  The processor enters Thread mode on Reset, or as a result of an exception return.
- Handler mode
  Handles exceptions.
  The processor returns to Thread mode when it has finished all exception processing.

A processor with the Security Extension supports both Non-secure and Secure states, which are orthogonal to traditional thread and handler modes. The four processor modes of operation are:

- Non-secure Thread mode
- Non-secure Handler mode
- Secure Thread mode
- Secure Handler mode

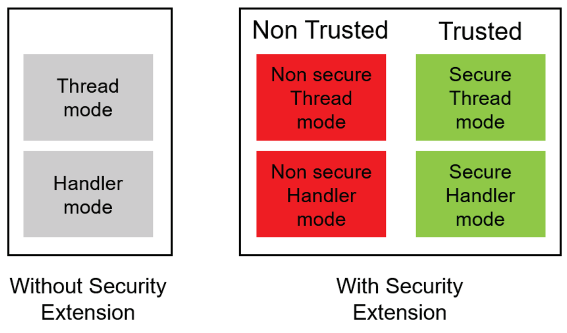


Figure 1. Processor States

When the processor takes an exception, the processor pushes information onto the current stack, and stores the information to status registers prior to exception entry.

Table 1 shows the available exception types and handlers, and fault status registers on each processor core. Please refer to Section: Exception model in the processor core's generic user guide for more details. The link of processor core's generic user guide is provided in the section 6 – reference.

1.1 Exception Types and Handlers

Table 1 shows the available exception types and handler types on each processor core.

Table 1. Available Exception Types and Handler Types

Handler types	Exception types		
	Arm® Cortex®-M4	**Arm® Cortex®-M23**	**Arm® Cortex®-M33** Note5 **Arm® Cortex®-M85** Note5
Interrupt Service Routines (ISRs) Note1	IRQ0 – IRQ239	IRQ0 – IRQ239	IRQ0 – IRQ479
Fault handler Note4	HardFault MemManage Fault BusFault UsageFault	HardFault	HardFault Note2 MemManage Fault BusFault Note2 UsageFault SecureFault Note3
System handler	NMI PendSV SVCall SysTick	NMI PendSV SVCall SysTick	NMI Note2 PendSV SVCall SysTick

Note 1. Usable IRQ number depends on MCU device specification. Please refer to hardware user's manual of MCUs.

Note 2. The entry target state for HardFault, BusFault, NMI is controlled by a BFHFNMINS bit in AIRCR (Application Interrupt and Reset Control) register.

Note 3. SecureFault always targets Secure State.

Note 4. All faults can be treated as HardFault according to a fault enable bit in SHCSR (System Handler Control and State) register. This bit is set to 0 (Disabled) by default.

Note 5. With Security Extension.

1.2 Fault Status Registers

Table 2 shows the fault status registers on each fault. The processor core stores the information of fault status to these related registers when the fault occurs. By verifying these registers, we can recognize the factor and the location of faults that occurred. Please refer to processor core's generic user guide for more register details.

Table 2. Fault status registers

Fault type	**Status and Address register name** Note2		
	Arm® Cortex®-M4	**Arm® Cortex®-M23**	**Arm® Cortex®-M33** **Arm® Cortex®-M85**
HardFault	SCB.HFSR	-	SCB.HFSR
MemManage Fault	SCB.MMFSR Note1 SCB.MMFAR	-	SCB.MMFSR Note1 SCB.MMFAR
BusFault	SCB.BFSR Note1 SCB.BFAR	-	SCB.BFSR Note1 SCB.BFAR
UsageFault	SCB.UFSR Note1	-	SCB.UFSR Note1
SecureFault	-	-	SAU.SFSR SAU.SFAR

Note 1. A subregister of the CFSR (Configurable Fault Status Register).

Note 2. Abbreviations used in this table are shown as follows.
SCB: System Control Block registers
SAU: Security Attribution Unit registers
HFSR: HardFault Status Register
MMFSR: MemManage Fault Status Register
MMFAR: MemManage Fault Address Register
BFSR: BusFault Status Register
BFAR: BusFault Address Register
UFSR: UsageFault Status Register
SFSR: Secure Fault Status Register
SFAR: Secure Fault Address Register

2. Bus Error Monitoring on RA Devices

The RA Family MCUs have bus error monitoring section which monitors each individual area, and whenever it detects an error, it returns the error to the requesting master IP. Also, the bus access status (access address, read/write information, error occurrence location) is stored in the bus error register.

Please refer to the Buses section in MCU's hardware user's manual for more details.

The subsequent section describes the faults specific to Arm® Cortex®-M33 and M85 core with Arm® TrustZone® and additional error detection capabilities associated with them.

2.1 Implementation on RA-CM33 Devices

In addition to the faults specific to Arm® Cortex®-M33 CPU core, RA-CM33 devices provide some additional error detection capability supported by the bus error monitoring system. This chapter describes how to handle additional error information.

Note on Section 2.1: This chapter describes the bus error monitoring system in case of RA-CM33 device RA6M5 MCU. Please refer to the MCU's hardware user's manual for other MCUs.

2.1.1 Overview of Bus Error

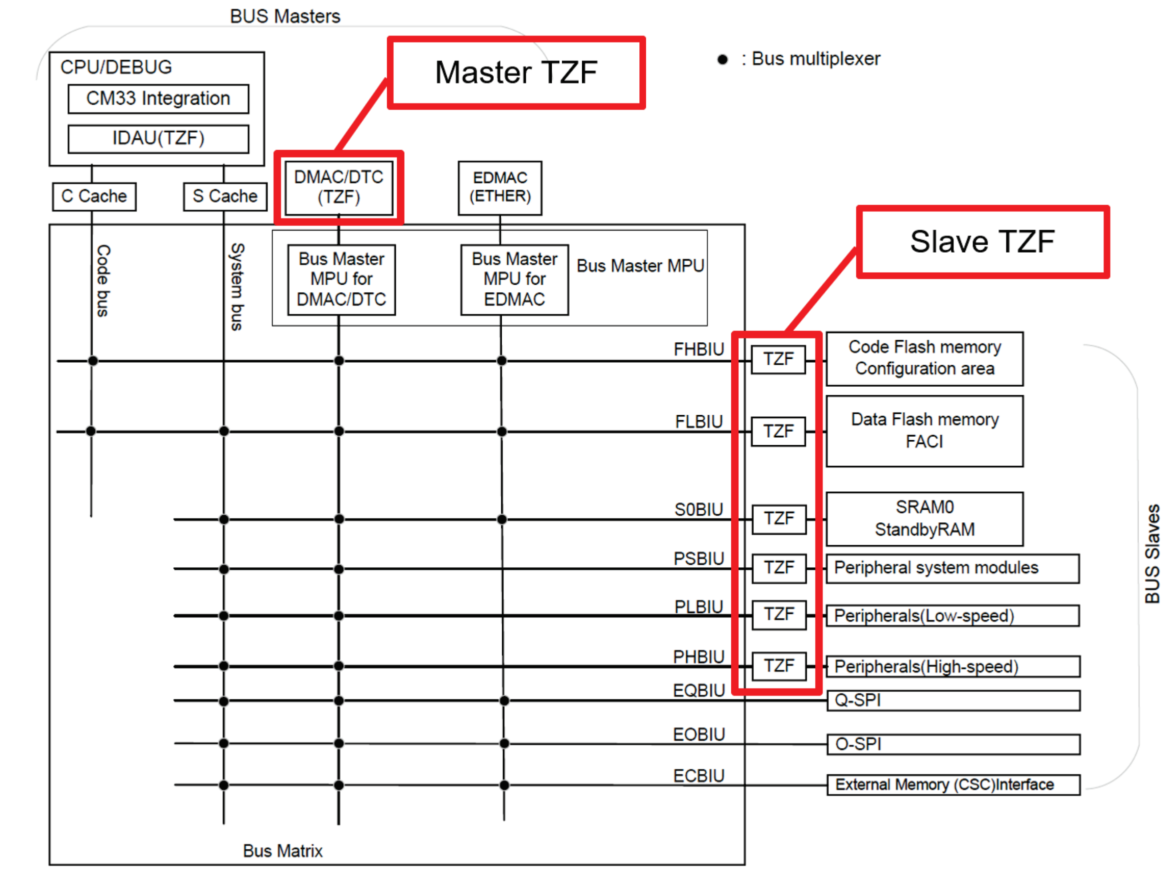
The bus error monitoring system can detect the following types of bus errors:

- Illegal address access
- Bus master MPU error
- TrustZone Filter error
- Bus error transmitted from each slave IP

The TrustZone Filter (TZF) is special on RA-CM33 devices and can detect security access errors caused by peripheral modules below that are out of CPU core scope.

- Master TZF: For bus master (DMAC, DTC, etc.)
- Slave TZF: For bus slave (Memory, peripherals)

See Figure 2 for the TZF implementation on RA-CM33 devices.



Note: TZF is TrustZone Filter. C cache is instruction cache. S cache is data cache.
For FLBIU, the code bus accesses the data flash memory and configuration areas (code region) through FLBIU. The system bus accesses FACI and SCDS (peripheral region) through FLBIU.

Figure 2. TZF on RA-CM33 Devices

When TZF detects the access errors, the TZF error occurs. Table 3 shows the system behavior when the TZF error occurred. The behavior varies depending on the master or slave area to be accessed.

Table 3. Behavior When the TZF Error Ocurred

Access to	Access from		
	CPU	DMAC/DTC	EDMAC
FHBIU (Code Flash) FLBIU (Data Flash) S0BIU (SRAM)	• SecureFault exception occurs	• Transfer does not start • NMI or reset occurs Note1 • Interrupt occurs (DMA_TRANSERR)	• NMI or reset occurs Note1 • Interrupt occurs (ETHER_EINT0) Note4
PSBIU (System peripherals) PHBIU (High-speed peripherals)	• BusFault exception occurs Note2 • NMI or reset occurs Note1	• Stop transfer Note3 • NMI or reset occurs Note1 Note3	(Out of accessible area)

Access to	Access from		
	CPU	DMAC/DTC	EDMAC
PLBIU (Low-speed peripherals)		• interrupt occurs Note3 (DMA_TRANSERR)	

Note 1: NMI or reset is selected with OAD (Operation after detection) bit in TZFOAD (TZF Operation After Detection) register.

Note 2: These error behaviors do not occur for write access to the PHBIU or PLBIU address space which memory attribute is set to "Early Write Acknowledgment" by the ARM® MPU.

Note 3: These error behaviors do not occur for write access from DMAC to the PHBIU or PLBIU address space when the bufferable write is enabled by DMBWR.BWE.

Note 4: Address error flag in EESR.ADE bit is set. The interrupt occurs when interrupt is enabled in EESIPR.ADEIP bit.

For information on the master TZF and others, please refer to the Bus Error Monitoring Section in the MCU hardware user’s manual. The next section shows the error operations and the state clearing flow of the slave TZF.

(1) Slave TrustZone® Filter

The Slave TZF is applied to FHBIU (code flash), FLBIU (data flash), S0BIU (SRAM), PSBIU (System peripherals), PHBIU (high-speed peripherals) and PLBIU (low-speed peripherals).

When the slave TZF error is detected, the following steps are processed:

1. Store the address of the error in BTZFnERRADD (Bus Error Address) register.
2. Store the read/write information of the error in BTZFnERRRW (Bus Error Read Write) register.
3. Set 1 to STERRSTAT (Slave TZF Error Status) bit of BUSnERRSTAT (Bus Error Status Register) register.

Then an NMI request or reset request is generated depending on the OAD (Operation after Detection) bit setting in the TZFOAD (TZF Operation After Detection) register as shown in Figure 3. Also Bus Fault exception handler will be triggered.

Users can recognize the error status by verifying BTZFnERRADD, BTZFnERRRW and BUSnERRSTAT register in the NMI handler or after reset. These registers are kept until reset other than MPU- and TZF-related resets or being cleared by BUSnERRCLR (BUS Error Clear) register.

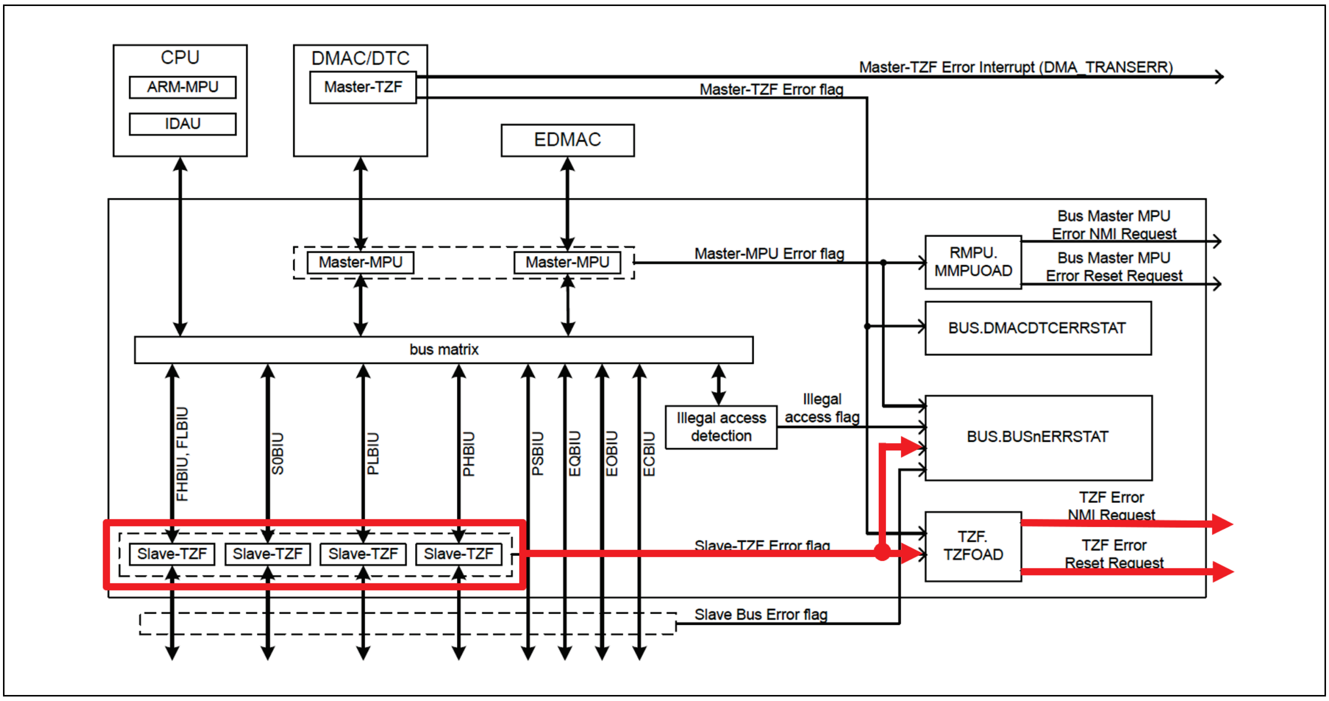


Figure 3. Slave TZF Error Signal Flow

2.1.2 Bus Error Processing

In order to verify and clear the error status flag without unexpected operation, additional operations below should be added to the BusFault exception handler routine.

1. Verify the error information in the corresponding register.
2. Clear the data in cache for the error address if cache is enabled.
3. Clear the Error Status register in the bus module.

Figure 4. shows the recommended BusFault exception handling flowchart. For the RA-CM33 devices, one consideration is needed for the Slave TZF error if the TZFOAD register is configured to generate an NMI. In this case, NMI will occur first then the BusFault exception occurs next. The NMI handler should only clear the NMI status, and the bus error status should be cleared in the BusFault handler. Figure 5. shows the NMI handling flowchart.

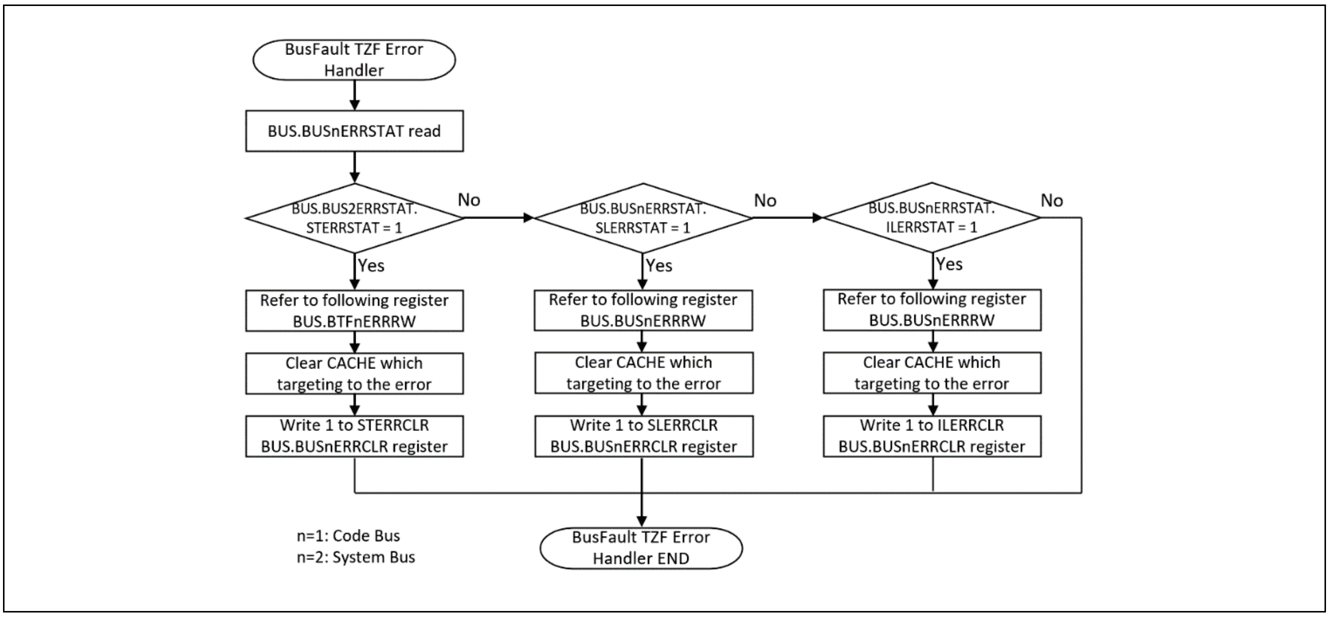

BusFault TZF Error Handler
BUS.BUSnERRSTAT read
BUS.BUS2ERRSTAT. STERRSTAT = 1
No
Yes
Refer to following register BUS.BTFnERRRW
Clear CACHE which targeting to the error
Write 1 to STERRCLR BUS.BUSnERRCLR register
BUS.BUSnERRSTAT. SLERRSTAT = 1
No
Yes
Refer to following register BUS.BUSnERRRW
Clear CACHE which targeting to the error
Write 1 to SLERRCLR BUS.BUSnERRCLR register
BUS.BUSnERRSTAT. ILERRSTAT = 1
No
Yes
Refer to following register BUS.BUSnERRRW
Clear CACHE which targeting to the error
Write 1 to ILERRCLR BUS.BUSnERRCLR register
n=1: Code Bus
n=2: System Bus
BusFault TZF Error Handler END

Figure 4. BusFault Interrupt Handling Flowchart

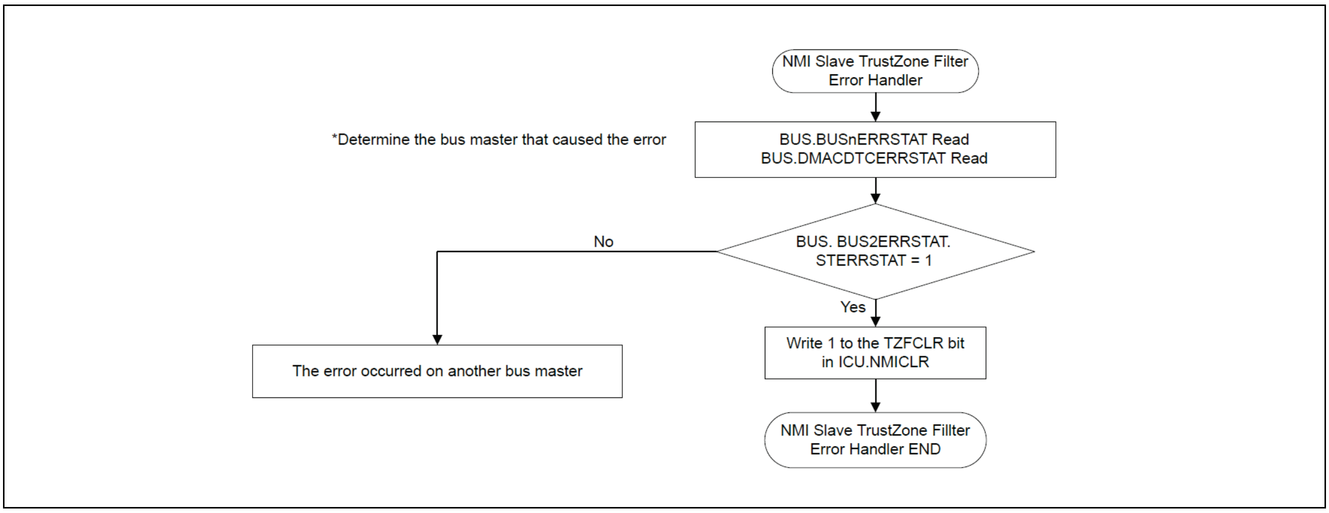

NMI Slave TrustZone Filter Error Handler
*Determine the bus master that caused the error
BUS.BUSnERRSTAT Read
BUS.DMACDTCERRSTAT Read
BUS. BUS2ERRSTAT. STERRSTAT = 1
No
Yes
The error occurred on another bus master
Write 1 to the TZFCLR bit in ICU.NMICLR
NMI Slave TrustZone Fillter Error Handler END

Figure 5. NMI Handling Flowchart

2.2 Implementation on RA-CM85 Devices

In addition to the faults specific to Arm® Cortex®-M85 CPU core, RA-CM85 devices provide some additional error detection capability in the bus error monitoring system.

Note: This section describes the bus error monitoring system in case of RA-CM85 device RA8D1 MCU. Please refer to the MCU's hardware user's manual for other MCUs.

2.2.1 Overview of Bus Error

The bus error monitoring system can detect the following types of bus errors:

- Master Security Attribution Unit error
- Illegal address access
- Bus master MPU error
- Slave TrustZone® Filter error
- Bus error transmitted from each slave IP

The Master Security Attribution Unit (MSAU) and the TrustZone Filter (TZF) is special on RA-CM85 devices and can detect security access errors caused by peripheral modules below that are out of CPU core scope.

- MSAU: For bus master (DMAC, DTC, etc.)
- Slave TZF: For bus slave (Memory, peripherals)

When the access errors are detected, the TZF error occurs. Table 4 shows the system behavior when the TZF access error occurs. The behavior varies depending on the master or slave area to be accessed.

Table 4. Behavior when the TZF Error Ocurred

CPU	DMAC/DTC	EDMAC	DRW/MIPI/GLCDC/CEU
• When IDAU/SAU detects, SecureFault exception occurs. • When slave TrustZone® filter detects, BusFault exception occurs. Also, a reset can be issued.	• Stop transfer. • NMI or reset occurs Note1 • Interrupt occurs (DMA0_TRANSERR)	• NMI or reset occurs Note1 • Interrupt occurs (ETHER_EINT0) Note2	• NMI or reset occurs Note1 • Interrupt occurs (DRW: DRW_IRQ Note3, MIPI: DSI_SEQ0Note4/ DSI_SEQ1Note4 / DSI_RCV Note5)

Note: If an access error occurs in bufferable write accesses (can be issued by CPU, DMA, and CEU), an error response may not be returned to the requesting master IP. In this case, the response can be given from the midpoint of bus architecture, and NMI or reset occurs according to relevant OAD (Operation After Detection) register setting. For more details, please refer to the Operations When a Bus Error Occurs section in MCU hardware user's manual.

Note 1: NMI or reset is selected with SRERROAD (Operation after detection) bit in BUSOAD (TZF Operation After Detection) register.

Note 2: Address Error flag (ADE bit in EESR (ETHERC/EDMAC status) register) is set. The interrupt occurs when it is enabled in ADEIP (Address error interrupt request enable) bit in EESIPR (ETHERC/EDMAC status interrupt enable) register.

Note 3: Access error flag (BUSIRQ (Bus error interrupt triggered) bit in STATUS register) is set. The interrupt occurs when it is enabled in BUSIRQEN (BUSIRA interrupt mask enable) bit in IRQCTL (Interrupt control) register.

Note 4: Access error flag (TXIBERR (Tx internal bus error interrupt) bit in SQCH0SR (Sequence channel 0 status) register, TXIBERR bit in SQCH1SR (Sequence channel 1 status) register) is set. The interrupt occurs when it is enabled in TXIBERR bit in SQCH0IER (Sequence channel 0 interrupt enable) register, or TXIBERR bit in SQCH1IER (Sequence channel 1 interrupt enable) register.

Note 5: Access error flag (IBERR (Internal bus error interrupt) bit in RXSR (Receive status) register) is set. The interrupt occurs when it is enabled in IBERR bit in RXIER (Receive interrupt enable) register.

For information on the master TZF (MSAU) and others, please refer to the Bus Error Monitoring Section in MCU hardware user's manual.

2.2.2 Bus Error Processing

In order to verify and clear the error status flag without unexpected operation, operations below should be added in your application. For the information of corresponding registers and operation based on different bus error types, please refer to the Operations When a Bus Error Occurs Section in MCU hardware user's manual.

1. Verify the error information in the corresponding register.
2. Clear the data in cache for the error address if cache is enabled.
3. Clear the Error Status register in the bus module.

3. User-Defined Exception Handler

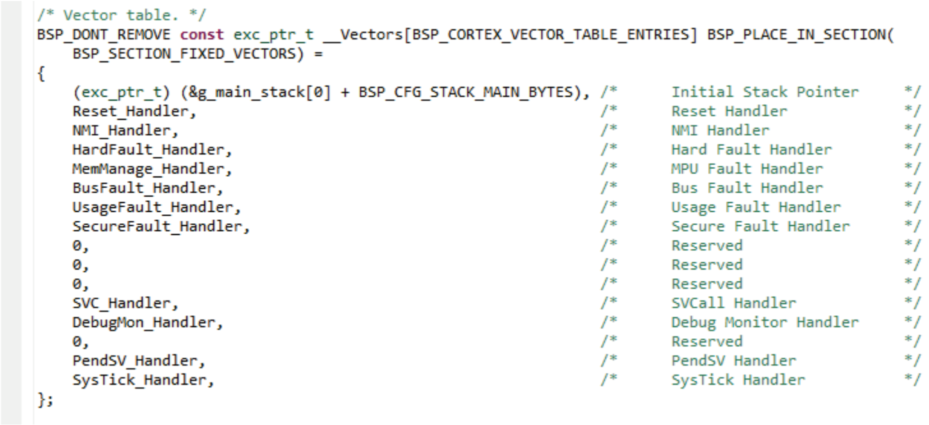
This chapter describes how to implement user-defined exception handlers in Flexible Software Package (FSP)-based projects.

3.1 Default Implementation

The FSP of RA family MCUs provides Board Support Packages (BSP) to support startups, exception handlers, and so on that depend on the specifications of each device. BSP implements exception handlers in the following source files. These files are independent of the CPU core type and are applied to any devices in the RA family.

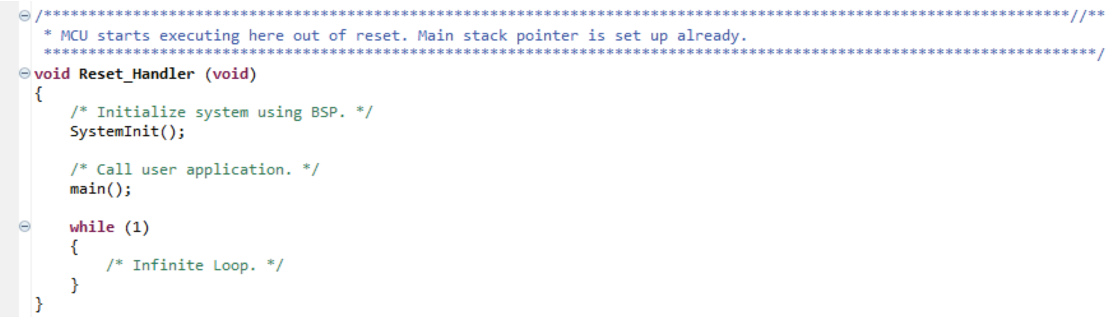
In `ra/fsp/src/bsp/cases/Device/RENESAS/Source/startup.c`:

- Vector table (Figure 6)
- Reset handler function (Figure 7)
- Prototype declaration of handler functions (Figure 8)
- Default handler function (Figure 9)



```
/* Vector table. */
BSP_DONT_REMOVE const exc_ptr_t __Vectors[BSP_CORTEX_VECTOR_TABLE_ENTRIES] BSP_PLACE_IN_SECTION(
    BSP_SECTION_FIXED_VECTORS) =
{
    (exc_ptr_t) (&g_main_stack[0] + BSP_CFG_STACK_MAIN_BYTES), /*      Initial Stack Pointer     */
    Reset_Handler,                                             /*      Reset Handler             */
    NMI_Handler,                                               /*      NMI Handler               */
    HardFault_Handler,                                         /*      Hard Fault Handler        */
    MemManage_Handler,                                         /*      MPU Fault Handler         */
    BusFault_Handler,                                          /*      Bus Fault Handler         */
    UsageFault_Handler,                                        /*      Usage Fault Handler       */
    SecureFault_Handler,                                       /*      Secure Fault Handler      */
    0,                                                         /*      Reserved                  */
    0,                                                         /*      Reserved                  */
    0,                                                         /*      Reserved                  */
    SVC_Handler,                                               /*      SVCall Handler            */
    DebugMon_Handler,                                          /*      Debug Monitor Handler     */
    0,                                                         /*      Reserved                  */
    PendSV_Handler,                                            /*      PendSV Handler            */
    SysTick_Handler,                                           /*      SysTick Handler           */
};
```

Figure 6. Vector Table



```
/*******************************************************************************************************************//**
 * MCU starts executing here out of reset. Main stack pointer is set up already.
 **********************************************************************************************************************/
void Reset_Handler (void)
{
    /* Initialize system using BSP. */
    SystemInit();

    /* Call user application. */
    main();

    while (1)
    {
        /* Infinite Loop. */
    }
}
```

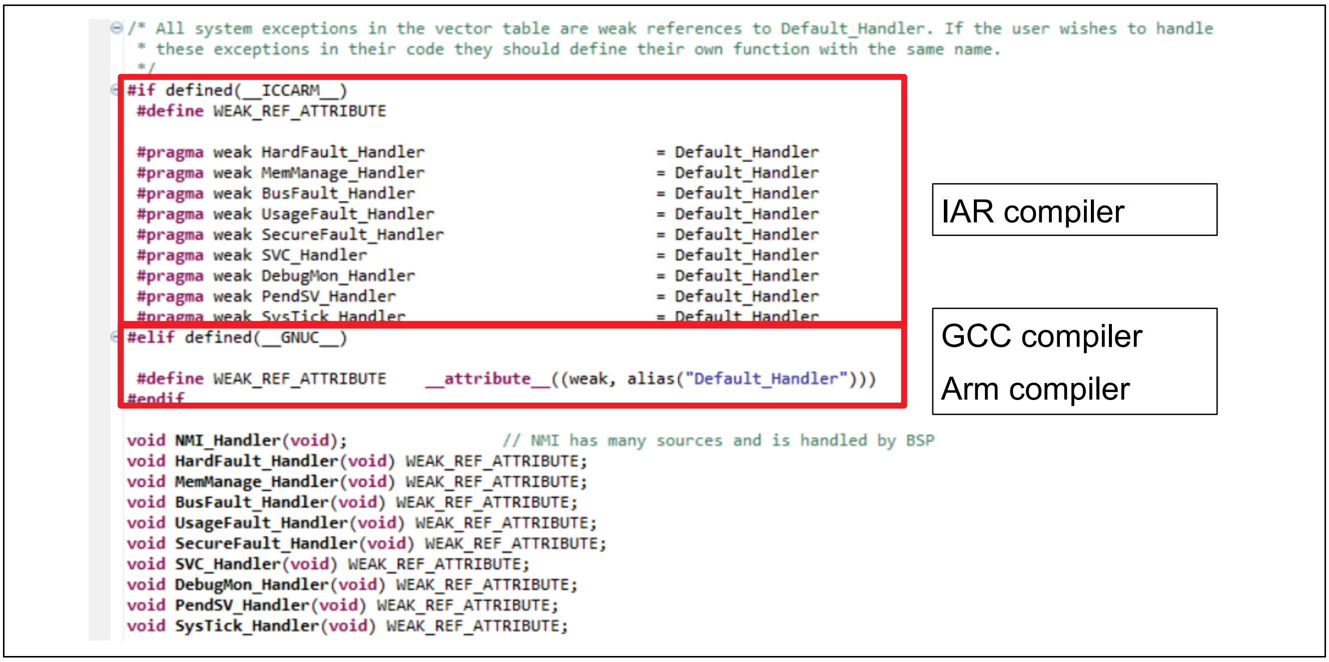
Figure 7. Reset Handler Function

In `ra/fsp/src/bsp/mcu/all/bsp_group_irq.c`:

- NMI handler function (Figure 10)

3.1.1 Exception Handler Definition

FSP applies the default handler function to various exception handler functions to ensure they exist even if a user did not define each exception handler function. The method of implementation depends on the compiler (Figure 8.).



```
/* All system exceptions in the vector table are weak references to Default_Handler. If the user wishes to handle
 * these exceptions in their code they should define their own function with the same name.
 */
#if defined(__ICCARM__)
 #define WEAK_REF_ATTRIBUTE

 #pragma weak HardFault_Handler                         = Default_Handler
 #pragma weak MemManage_Handler                         = Default_Handler
 #pragma weak BusFault_Handler                          = Default_Handler
 #pragma weak UsageFault_Handler                        = Default_Handler
 #pragma weak SecureFault_Handler                       = Default_Handler
 #pragma weak SVC_Handler                               = Default_Handler
 #pragma weak DebugMon_Handler                          = Default_Handler
 #pragma weak PendSV_Handler                            = Default_Handler
 #pragma weak SysTick_Handler                           = Default_Handler
#elif defined(__GNUC__)

 #define WEAK_REF_ATTRIBUTE    __attribute__((weak, alias("Default_Handler")))
#endif

void NMI_Handler(void);                 // NMI has many sources and is handled by BSP
void HardFault_Handler(void) WEAK_REF_ATTRIBUTE;
void MemManage_Handler(void) WEAK_REF_ATTRIBUTE;
void BusFault_Handler(void) WEAK_REF_ATTRIBUTE;
void UsageFault_Handler(void) WEAK_REF_ATTRIBUTE;
void SecureFault_Handler(void) WEAK_REF_ATTRIBUTE;
void SVC_Handler(void) WEAK_REF_ATTRIBUTE;
void DebugMon_Handler(void) WEAK_REF_ATTRIBUTE;
void PendSV_Handler(void) WEAK_REF_ATTRIBUTE;
void SysTick_Handler(void) WEAK_REF_ATTRIBUTE;
```

Figure 8. Definition of Exception Handlers

3.1.1.1 GCC compiler and Arm® compiler

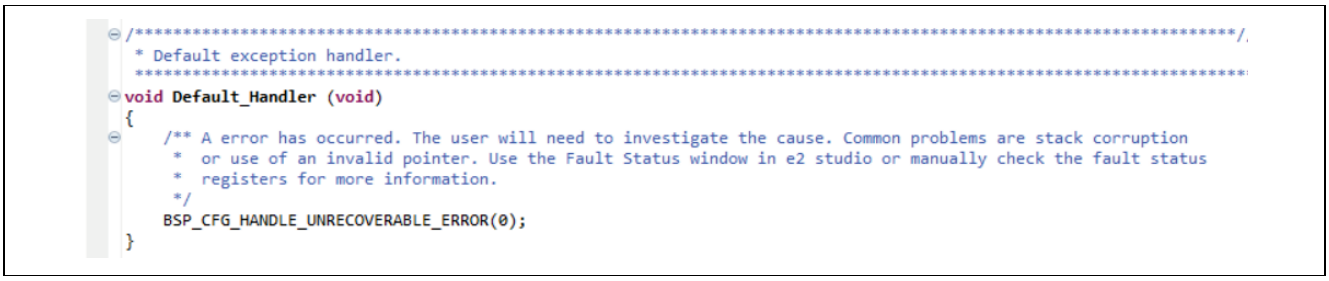
The GCC and Arm® compiler use function attributes (weak symbols, alias functions) to apply the default handler function to exception handers, other than the reset handler. The user can overwrite these handlers to user-defined functions if the function is defined using the same name.

3.1.1.2 IAR compiler

The IAR compiler uses pragma (weak symbols) to apply the default handler function to exception handlers, other than the reset handler. The user can overwrite these handlers to user-defined functions if the function is defined using the same name.

3.1.2 Default Handler

The BSP of FSP defines the default handler function (`Default_Handler`) and the function applies identically to exception handlers other than NMI handlers. The default handler function executes a breakpoint instruction to stop the program.

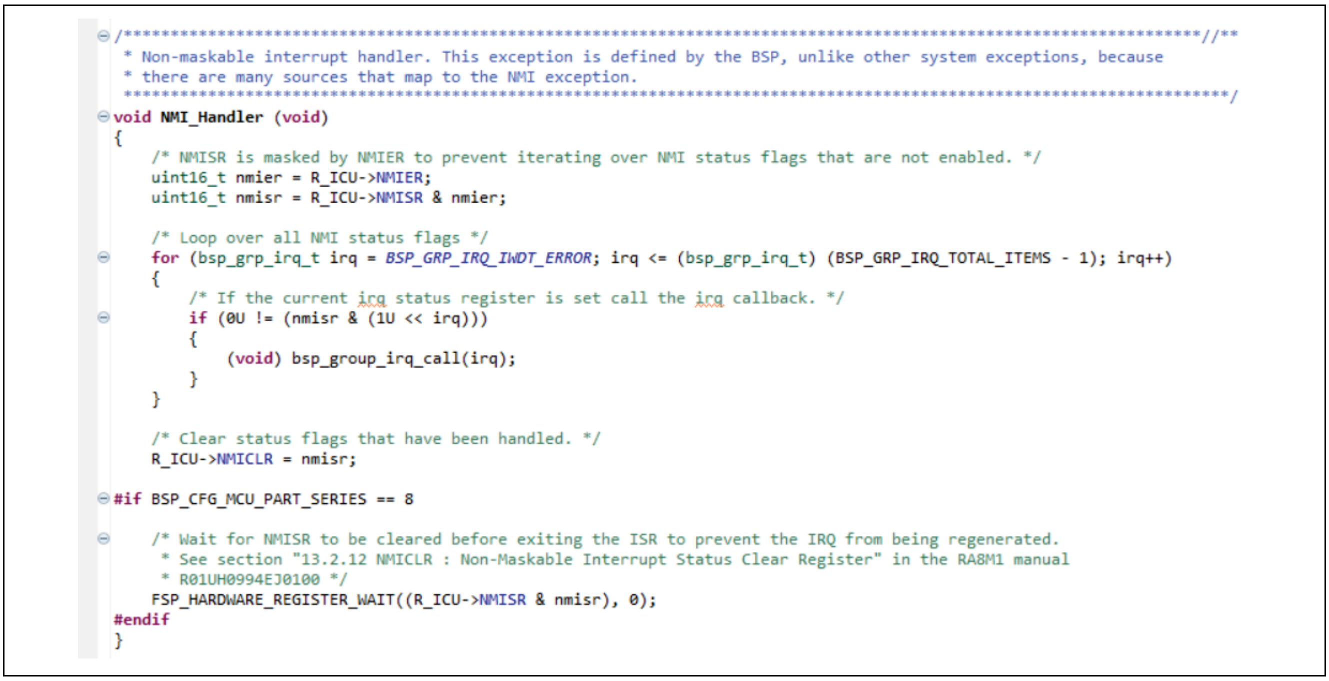


```
/*******************************************************************************************************************//**
 * Default exception handler.
 **********************************************************************************************************************/
void Default_Handler (void)
{
    /** A error has occurred. The user will need to investigate the cause. Common problems are stack corruption
     *  or use of an invalid pointer. Use the Fault Status window in e2 studio or manually check the fault status
     *  registers for more information.
     */
    BSP_CFG_HANDLE_UNRECOVERABLE_ERROR(0);
}
```

Figure 9. Default Handler Function

3.1.3 NMI Handler

The NMI handler function is defined in `bsp_group_irq.c`. When an NMI event occurs, the NMI handler calls a callback function that can be preset by the `R_BSP_GroupIrqWrite()` API. It then clears the status flag.



```
/************************************************************************************************************//**
 * Non-maskable interrupt handler. This exception is defined by the BSP, unlike other system exceptions, because
 * there are many sources that map to the NMI exception.
 **************************************************************************************************************/
void NMI_Handler (void)
{
    /* NMISR is masked by NMIER to prevent iterating over NMI status flags that are not enabled. */
    uint16_t nmier = R_ICU->NMIER;
    uint16_t nmisr = R_ICU->NMISR & nmier;

    /* Loop over all NMI status flags */
    for (bsp_grp_irq_t irq = BSP_GRP_IRQ_IWDT_ERROR; irq <= (bsp_grp_irq_t) (BSP_GRP_IRQ_TOTAL_ITEMS - 1); irq++)
    {
        /* If the current irq status register is set call the irq callback. */
        if (0U != (nmisr & (1U << irq)))
        {
            (void) bsp_group_irq_call(irq);
        }
    }

    /* Clear status flags that have been handled. */
    R_ICU->NMICLR = nmisr;

#if BSP_CFG_MCU_PART_SERIES == 8

    /* Wait for NMISR to be cleared before exiting the ISR to prevent the IRQ from being regenerated.
     * See section "13.2.12 NMICLR : Non-Maskable Interrupt Status Clear Register" in the RA8M1 manual
     * R01UH0994EJ0100 */
    FSP_HARDWARE_REGISTER_WAIT((R_ICU->NMISR & nmisr), 0);
#endif
}
```

Figure 10. NMI Handler Function

3.2 How to Add User-defined Exception Handler

3.2.1 How to Configure NMI Handler and Faults

3.2.1.1 Enable the Additional Exceptions

The additional exceptions can be enabled by the following bits. These bits are set 0 (Disable) after reset.

- SCB.SHCSR (System Handler Control and State Register)
  MEMFAULTENA: MemManage enable bit
  BUSFAULTENA: BusFault enable bit
  USGFAULTENA: UsageFault enable bit
  SECUREFAULTENA: SecureFault enable bit Note1

Note 1: Available for a processor with Security Extension

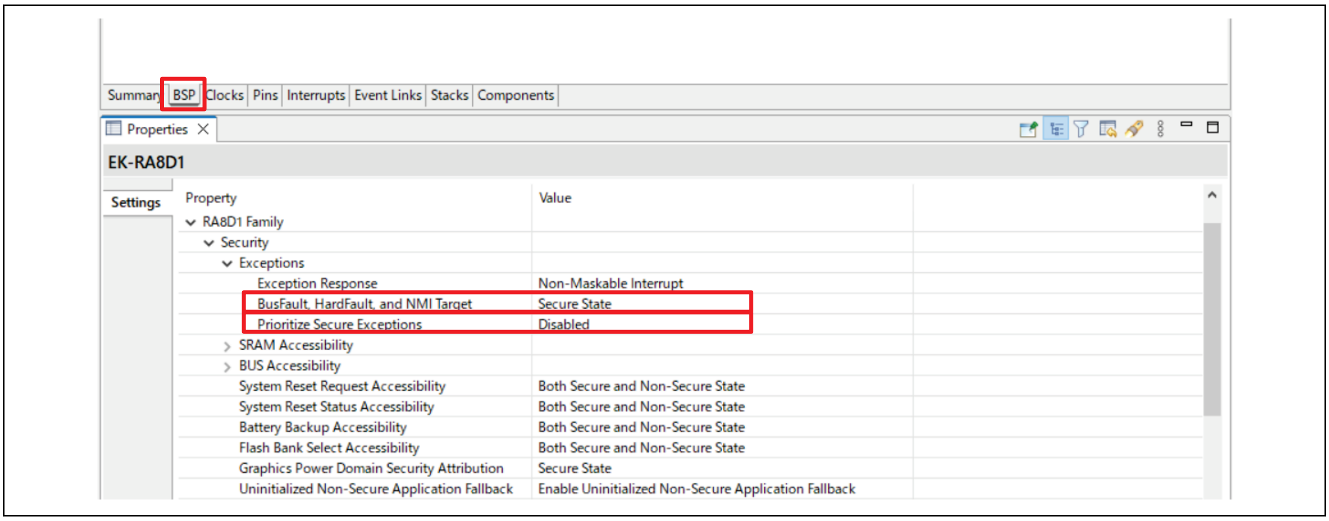
3.2.1.2 Configure the Exception for RA-CM33/RA-CM85 Devices

The target of HardFault, BusFault, and NMI are controlled by the BFHFNMINS (BusFault, HardFault, and NMI Non-secure) bit in AIRCR (Application Interrupt and Reset Control) register. This register can be configured using the FSP configurator (Figure).

- **Properties of BSP tab > RAxxx Family > Security > Exceptions > BusFault, HardFault, and NMI Target**

Also, the secure exception prioritization can be controlled by the PRIS (Prioritize Secure exceptions) bit in AIRCR. If this bit is set to 1 (Enabled), Non-secure exceptions are de-prioritized. In the FSP configurator, the configuration for this bit is:

- **Properties of BSP tab > RAxxx Family > Security > Exceptions > Prioritize Secure Exceptions**



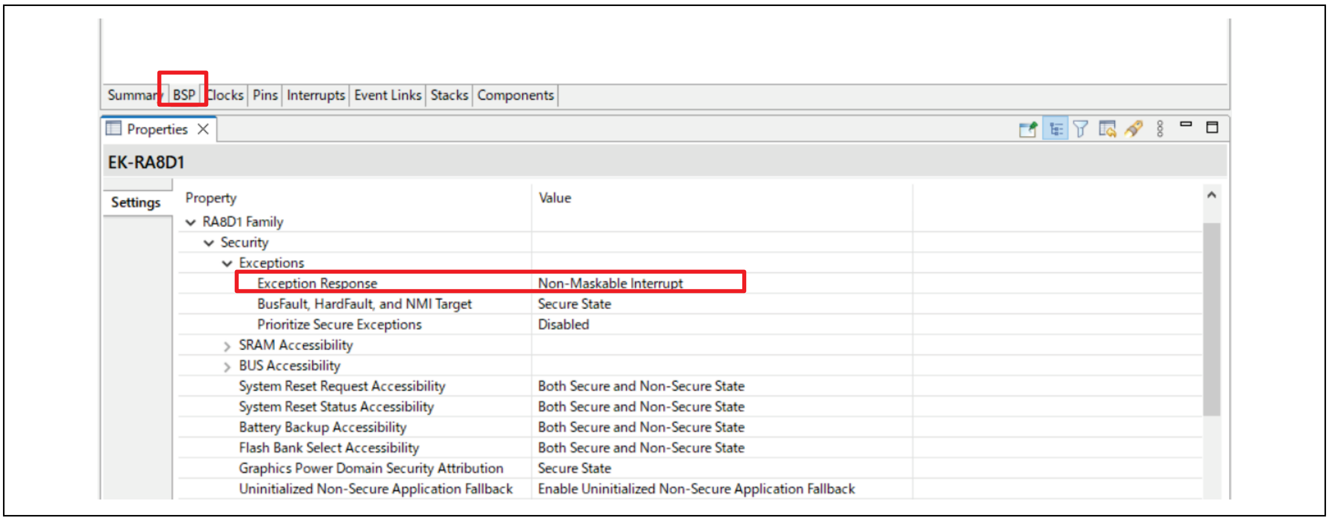
Property	Value
RA8D1 Family	
Security	
Exceptions	
Exception Response	Non-Maskable Interrupt
BusFault, HardFault, and NMI Target	Secure State
Prioritize Secure Exceptions	Disabled
SRAM Accessibility	
BUS Accessibility	
System Reset Request Accessibility	Both Secure and Non-Secure State
System Reset Status Accessibility	Both Secure and Non-Secure State
Battery Backup Accessibility	Both Secure and Non-Secure State
Flash Bank Select Accessibility	Both Secure and Non-Secure State
Graphics Power Domain Security Attribution	Secure State
Uninitialized Non-Secure Application Fallback	Enable Uninitialized Non-Secure Application Fallback

Figure 11. Settings of Exception

3.2.1.3 Configure the Reset Interrupt Request for Peripherals

TZF and some peripherals like WDT have options to cause Non-Maskable Interrupt request or Reset output when an exception occurs. For peripherals, most options can be configured in peripheral module driver properties in the **Stacks** tab. For TZF, this can be configured by the following settings in the **BSP** tab (Figure 12).

- **Properties of BSP tab > RAxxx Family > Security > Exceptions > Exception Response**



Property	Value
RA8D1 Family	
Security	
Exceptions	
Exception Response	Non-Maskable Interrupt
BusFault, HardFault, and NMI Target	Secure State
Prioritize Secure Exceptions	Disabled
SRAM Accessibility	
BUS Accessibility	
System Reset Request Accessibility	Both Secure and Non-Secure State
System Reset Status Accessibility	Both Secure and Non-Secure State
Battery Backup Accessibility	Both Secure and Non-Secure State
Flash Bank Select Accessibility	Both Secure and Non-Secure State
Graphics Power Domain Security Attribution	Secure State
Uninitialized Non-Secure Application Fallback	Enable Uninitialized Non-Secure Application Fallback

Figure 12. Settings of Exception Response for TZF

3.2.2 How to Implement A User Exception Handler

As shown in Section 3.1.1, FSP's BSP applies the `Default_Handler()` function to each exception handler, other than the NMI handler, by using weak symbols. These exception handlers can be overwritten to a user-defined exception handler by non-weak definition.

The following shows the example of a definition for user HardFault handler.

```
void HardFault_Handler(void);
void HardFault_Handler(void){
    …
}
```

3.3 Notes

RA-CM33 and RA-CM85 devices implement some security features such as TrustZone® technology (Armv8-M Security Extension) and DLM (Device Lifecycle Management). These features enable a more secure application and allow for a more secure software development. When an exception occurs, the processor may call a secure exception handler depending on the security settings, but a Non-secure application may not be able to detect the state. A Secure application may want to provide a user notification for a Non-secure application as needed.

3.3.1 TrustZone® Technology

Arm® TrustZone® technology divides the system and the application into Secure and Non-secure domains. Secure applications can access both Secure and Non-secure memory and resources. Non-secure code can access Non-secure memory and resources, as well as Secure resources through a set of so-called veneers located in the Non-secure Callable (NSC) region. When developing applications using DLM, you need to be aware of accessible areas by TrustZone® technology.

Note: For the application project with the FSP’s Flat Project type, the entire project will be in the Secure domain except for the EDMAC RAM buffers.

3.3.2 Device Lifecycle Management

RA-CM33 and RA-CM85 devices provide Device Lifecycle Management (DLM) function that allows users to limit the area that can be accessed from the debug interface and serial programming interfaces. By changing the level of this feature to the next stage, application developers can restrict program code reads from debug interface and serial programming interfaces. When developing an application using DLM, users need to be aware of how DLM works and the current status (Table 5). Please refer to MCU’s Hardware User’s Manual for more information.

Table 5. DLM State in Software Development Stage

Lifecycle	Debug level	Serial programming
RA-CM33: SSD state (Secure Software Development) RA-CM85: OEM-AL2 state (Original Equipment Manufacturer – Authentication Level 2)	Allows the debug connection. Can access all memories and peripherals.	Available Can program/erase/read all code/data flash area
RA-CM33: NSECSD state (Non-secure Software Development) RA-CM85: OEM-AL1 state (Original Equipment Manufacturer – Authentication Level 1)	Allows the debug connection Can access only Non-secure memory regions and peripherals	Available Can program/erase/read only Non-secure code/data flash area

4. Debugging Exception Events

This chapter provides debugging methods for exception events that have occurred. RA family devices provide a variety of debugging environments such as Renesas e^2 studio IDE and partner tools. You can use these debugging environments to improve debugging efficiency.

When an exception occurs, it is important to determine the factor, location, and history of the exception event to recognize that state. These can be acquired by checking the related registers and tracing the program execution.

- Confirm the fault related registers
  Determine the factors by checking the related fault state register, and the location of the root cause by checking the CPU general registers that have been retracted into current stack memory.
- Tracing the program execution
  Determine the history of the faults by tracing their execution.

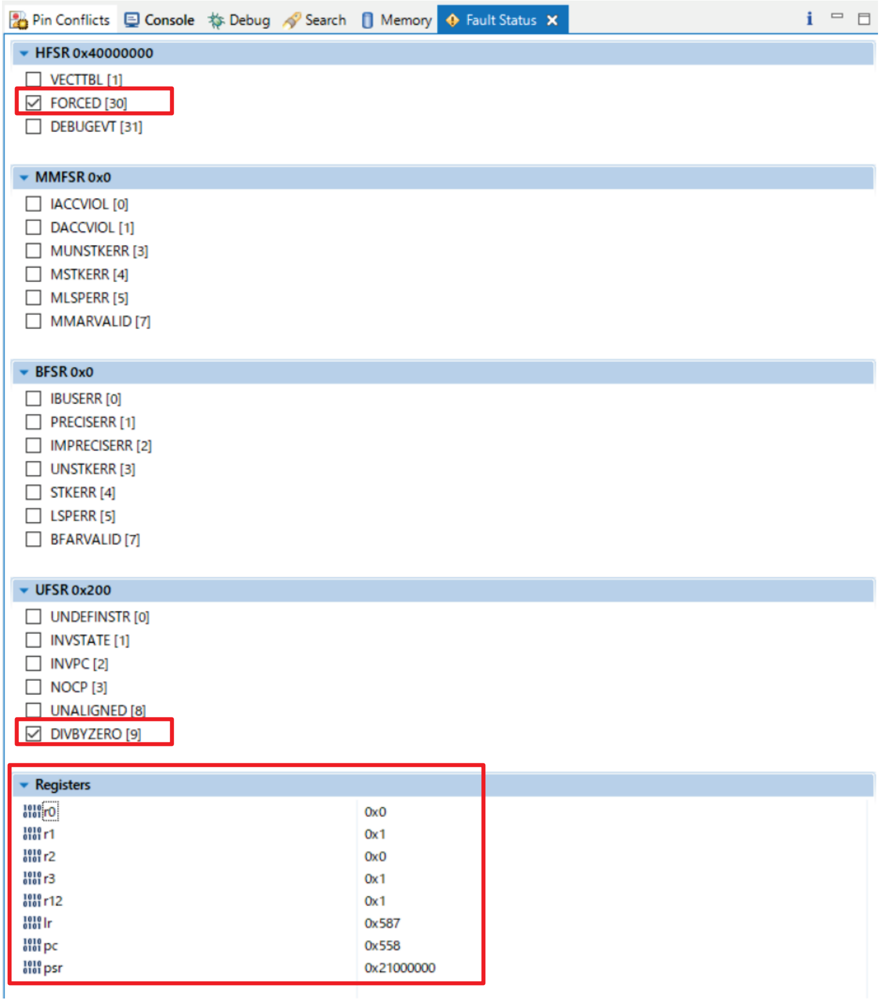
4.1 Confirming the Fault Status

Recognize the fault status using the e^2 studio view functions or partner tools.

4.1.1 View Function on Renesas e^2 studio IDE

The e^2 studio IDE provides various view functions to visualize the state of the MCU. Refer to *Renesas e^2 studio 2022-07 or higher User's Manual: Quick Start Guide* (R20UT5210) for more details and usage instructions.

- **Fault Status** view
  The **Fault Status** view displays the related registers with the fault condition and the CPU general registers that have been retracted into stack memory (Figure 13).
- **Registers** view
  The **Registers** view displays information about the general registers in the CPU core.
- **Memory** view
  The **Memory** view allows users to display and edit the memory. That can be used to check stack memory.
- **Expressions** view
  The **Expressions** view allows users to monitor the value of global variables, static variables, or local variables that stored in the memory. That can be used to check the current value of the user buffer.



HFSR 0x40000000
□ VECTTBL [1]
☑ FORCED [30]
□ DEBUGEVT [31]

MMFSR 0x0
□ IACCVIOL [0]
□ DACCVIOL [1]
□ MUNSTKERR [3]
□ MSTKERR [4]
□ MLSPERR [5]
□ MMARVALID [7]

BFSR 0x0
□ IBUSERR [0]
□ PRECISERR [1]
□ IMPRECISERR [2]
□ UNSTKERR [3]
□ STKERR [4]
□ LSPERR [5]
□ BFARVALID [7]

UFSR 0x200
□ UNDEFINSTR [0]
□ INVSTATE [1]
□ INVPC [2]
□ NOCP [3]
□ UNALIGNED [8]
☑ DIVBYZERO [9]

Registers

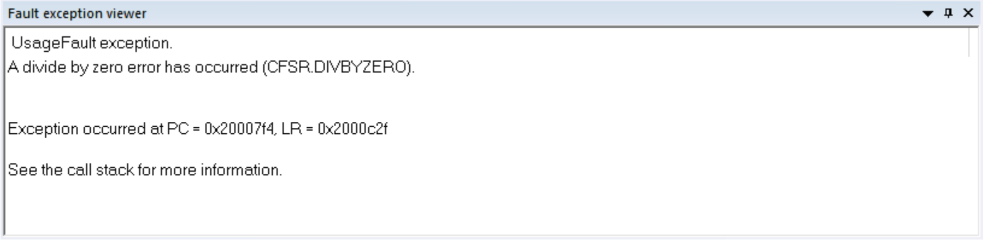
r0	0x0
r1	0x1
r2	0x0
r3	0x1
r12	0x1
lr	0x587
pc	0x558
psr	0x21000000

Note: This figure shows the fault status when a division-by-zero error occurs and the UsageFault is disabled.

Figure 13. Fault Status view on e^2 studio IDE

4.1.2 Partners Tools

IAR Embedded Workbench for ARM and Arm® Keil® MDK provide visibility into fault-related status information for RA devices. Figure 14 and Figure 19 show an example of the **Fault exception viewer** window that displays information on the most recent fault exception on the IAR EW for ARM and Arm® Keil® MDK. Please refer to partner tool documentation for more information.



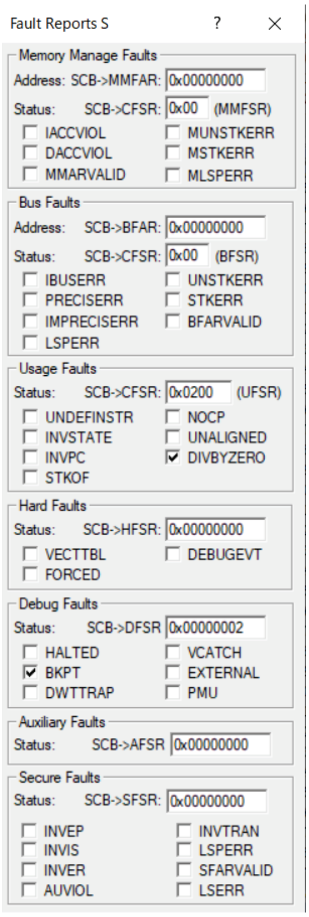
UsageFault exception.
A divide by zero error has occurred (CFSR.DIVBYZERO).

Exception occurred at PC = 0x20007f4, LR = 0x2000c2f

See the call stack for more information.

Note: This figure shows the fault status when a dividion-by-zero error occurs.

Figure 14. Fault exception viewer window on the IAR EW for ARM



Memory Manage Faults
Address: SCB->MMFAR: 0x00000000
Status: SCB->CFSR: 0x00 (MMFSR)
□ IACCVIOL □ MUNSTKERR
□ DACCVIOL □ MSTKERR
□ MMARVALID □ MLSPERR

Bus Faults
Address: SCB->BFAR: 0x00000000
Status: SCB->CFSR: 0x00 (BFSR)
□ IBUSERR □ UNSTKERR
□ PRECISERR □ STKERR
□ IMPRECISERR □ BFARVALID
□ LSPERR

Usage Faults
Status: SCB->CFSR: 0x0200 (UFSR)
□ UNDEFINSTR □ NOCP
□ INVSTATE □ UNALIGNED
□ INVPC ☑ DIVBYZERO
□ STKOF

Hard Faults
Status: SCB->HFSR: 0x00000000
□ VECTTBL □ DEBUGEVT
□ FORCED

Debug Faults
Status: SCB->DFSR 0x00000002
□ HALTED □ VCATCH
☑ BKPT □ EXTERNAL
□ DWTTRAP □ PMU

Auxiliary Faults
Status: SCB->AFSR 0x00000000

Secure Faults
Status: SCB->SFSR: 0x00000000
□ INVEP □ INVTRAN
□ INVIS □ LSPERR
□ INVER □ SFARVALID
□ AUVIOL □ LSERR

Note: This figure shows the fault status when a dividion-by-zero error occurs.

Figure 15. Fault Reports window on the Arm® Keil® MDK

4.2 Tracing the Execution

By tracing the program execution, the history of faults can be recognized.

- Tracing the program execution with Arm® CoreSight trace technology (ETM/SWV trace)
- Tracing the program execution without Processor Core Trace feature

4.2.1 Use of Arm® CoreSight Trace

Arm® CoreSight technology provides hardware tracing features such as ETM (Embedded Trace Macrocell) and SWV (Serial Wire Viewer). Using these tracing functions, we can understand the operation of a processor without affecting user program execution. These tracing functions are available for RA-CM4 and RA-CM33 devices.

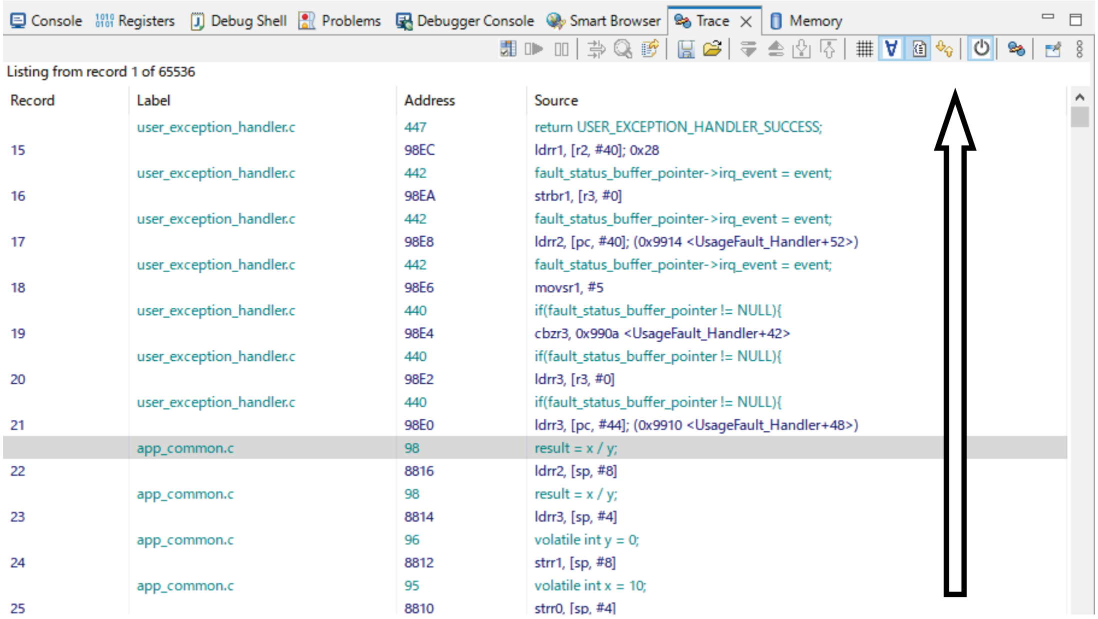
- ETM
  ETM is a real-time trace module providing all the executed instructions and data tracing of a processor. It requires 4 wires and is an event-driven trace.
- SWV
  SWV provides trace capabilities such as display of reads, writes, exceptions, PC Samples, and printf. It is single-wire and is a periodic polling trace.

Note: Renesas kits implement the ETM trace interface, but the on-board debugger does not support ETM trace. Additional hardware such as Segger J-Trace are required.

4.2.2 Use of e^2 studio Trace View

The e^2 studio IDE provides the program tracing function **Trace** view that can trace the program execution flow through the debugger. The function can be used without additional hardware and software. Therefore, the **Trace** view can be an effective tool for discovering root causes.

Refer to “R*enesas e^2 studio 2021-04 or higher User’s Manual: Quick Start Guide* (R20UT4989) for more details and usage instructions.



Listing from record 1 of 65536

Record	Label	Address	Source
	user_exception_handler.c	447	return USER_EXCEPTION_HANDLER_SUCCESS;
15		98EC	ldrr1, [r2, #40]; 0x28
	user_exception_handler.c	442	fault_status_buffer_pointer->irq_event = event;
16		98EA	strbr1, [r3, #0]
	user_exception_handler.c	442	fault_status_buffer_pointer->irq_event = event;
17		98E8	ldrr2, [pc, #40]; (0x9914 <UsageFault_Handler+52>)
	user_exception_handler.c	442	fault_status_buffer_pointer->irq_event = event;
18		98E6	movsr1, #5
	user_exception_handler.c	440	if(fault_status_buffer_pointer != NULL){
19		98E4	cbzr3, 0x990a <UsageFault_Handler+42>
	user_exception_handler.c	440	if(fault_status_buffer_pointer != NULL){
20		98E2	ldrr3, [r3, #0]
	user_exception_handler.c	440	if(fault_status_buffer_pointer != NULL){
21		98E0	ldrr3, [pc, #44]; (0x9910 <UsageFault_Handler+48>)
	app_common.c	98	result = x / y;
22		8816	ldrr2, [sp, #8]
	app_common.c	98	result = x / y;
23		8814	ldrr3, [sp, #4]
	app_common.c	96	volatile int y = 0;
24		8812	strr1, [sp, #8]
	app_common.c	95	volatile int x = 10;
25		8810	strr0, [sp, #4]

Note: This figure shows the fault status when a dividion-by-zero error occurs.

Figure 16. Trace View on e^2 studio IDE

5. Demonstration

This chapter explains the sample application that demonstrates an exception handling flow for multiple possible faults.

5.1 Functional Specifications

The sample application implements the following functions.

- **Illegal operation attempt function**
  This function provides some options for attempting illegal operation. Table 6. shows the available options. After the MCU starts, users can select the options by inputting the value on RTT Viewer.

- **Example exception handling function**
  This function implements the processing flow to identify and clear the slave TZF error as shown in section 2.1.2. When faults occur, this function **saves the fault status event**, **clears the status flag**, and **acquires the last execution program counter**. The fault status register values saved in this function are as shown in Table 2. The saved fault status will be output to RTT Viewer after system reset.
  Note: The software reset can be enabled by defining the ENABLED_SOFTWARE_RESET macro. If the macro is defined, the system executes software reset after exception operation, otherwise stop the program by breakpoint instruction.
  Note: For EK-RA6M5 projects that use Secure/Non-secure project types, by default, the acquired fault status events are stored in a Non-secure region memory buffer for both Secure and Non-secure exceptions.
- **LED Blinky function (For confirming the CPU is running under no-fault)**
  This function toggles a LED for a test purpose. When LED blinks, that means the MCU is running fine. When it stopped, exception(s) have occurred.

Table 6. Available Input Options on RTT Viewer

RTT Viewer Input	**Action**	**Available exceptions on target MCU kits**			
		EK-RA6M3 (RA-CM4)	**EK-RA2E1 (RA-CM23)**	**EK-RA6M5 (RA-CM33), EK-RA8M1 (RA-CM85)**	**EK-RA6M5 (RA-CM33), EK-RA8M1 (RA-CM85)**
		Flat project	**Flat project**	**Flat project**	**Secure/Non-secure project**
“1”	WDT underflow	✓ NMI	✓ NMI	✓ NMI	✓ NMI
“2”	Stack overflow	✓ NMI	✓ NMI	✓ UsageFault	✓ UsageFault
“3”	Execute instruction from illegal region	✓ MemManage	✓ HardFault	✓ MemManage	✓ MemManage
“4”	Divide by zero	✓ UsageFault	-	✓ UsageFault	✓ UsageFault
“5”	Access Secure attribute memory from Non-secure code	-	-	–	✓ SecureFault
“6”	Access Secure attribute peripheral register from Non-secure code	-	-	–	✓ NMI, BusFault
“7”	Access Secure attribute peripheral register from Non-secure code via NSC	-	-	–	✓ (No fault)
“10”	Clear the fault status buffer	✓	✓	✓	✓

Note. “√” ：Available, “-”: Not available.

5.2 Description of the Application Project

Table 7 describes the sample applications in the application project (zip file). The projects can be downloaded from the Renesas web site.

Table 7. Sample Projects

Project Name	Description
Exception_Handling_Example_EK_RA6M3_Flat	The sample application with Flat project type for EK-RA6M3 kit
Exception_Handling_Example_EK_RA2E1_Flat	The sample application with Flat project type for EK-RA2E1 kit
Exception_Handling_Example_EK_RA6M5_Flat	The sample application with Flat project type for EK-RA6M5 kit
Exception_Handling_Example_EK_RA6M5_NS Exception_Handling_Example_EK_RA6M5_S	The sample applications with Secure/Non-secure project type for EK-RA6M5 kit
Exception_Handling_Example_EK_RA8M1_Flat	The sample application with Flat project type for EK-RA8M1 kit
Exception_Handling_Example_EK_RA8M1_NS Exception_Handling_Example_EK_RA8M1_S	The sample applications with Secure/Non-secure project type for EK-RA8M1 kit

The files in Table 8 from this application project serve as a reference.

Table 8. Files used in application project

File name	Purpose
`src/hal_entry.c`	Contains main application call
`src/app_main.c`	Contains data structures and functions used in main application
`src/app_common.c`	Contains functions used for the LED Blinky function and Illegal operation attempting function
`src/app_common.h`	Accompanying header for exposing functionality provided by app_common.c
`src/app_message_data.h`	Contains macros to output messages and register descriptions to RTT Viewer
`src/user_exception_handler.c`	Contains data structures and functions used in example exception handling function
`src/user_exception_handler.h`	Accompanying header for exposing functionality provided by user_exception_handler.c
`src/user_nmi_handler.c`	Contains functions used in exception handling function
`src/user_nmi_handler.h`	Accompanying header for exposing functionality provided by user_nmi_handler.c
`src/SEGGER_RTT/*`	Implementation of SEGGER real-time transfer (RTT) which allows real-time communication on targets which support debugger memory accesses while the CPU is running
`src/rtt_common_utils.h`	Contains macros, data structures, and functions commonly used across the project

5.3 Used Peripheral Modules

This application uses the following peripheral modules.

- GPT (General PWM Timer) channel 0: Generate periodic interrupt for the LED toggling. For Secure/ Non-secure projects, the GPT0's security attribute is set to Secure.
- WDT (WatchDog Timer): Generate NMI interrupt when underflow event has occurred.

5.4 User Interface

Table 9 and Figure 17 show the pins used for user interfaces.

Table 9. Used pins and connectors

User Interface	EK-RA6M3 Kit	EK-RA2E1 Kit	EK-RA6M5 Kit	EK-RA8M1 Kit
LED1	P403	P915	P006	P600
PC connector	J10	J10	J10	J10

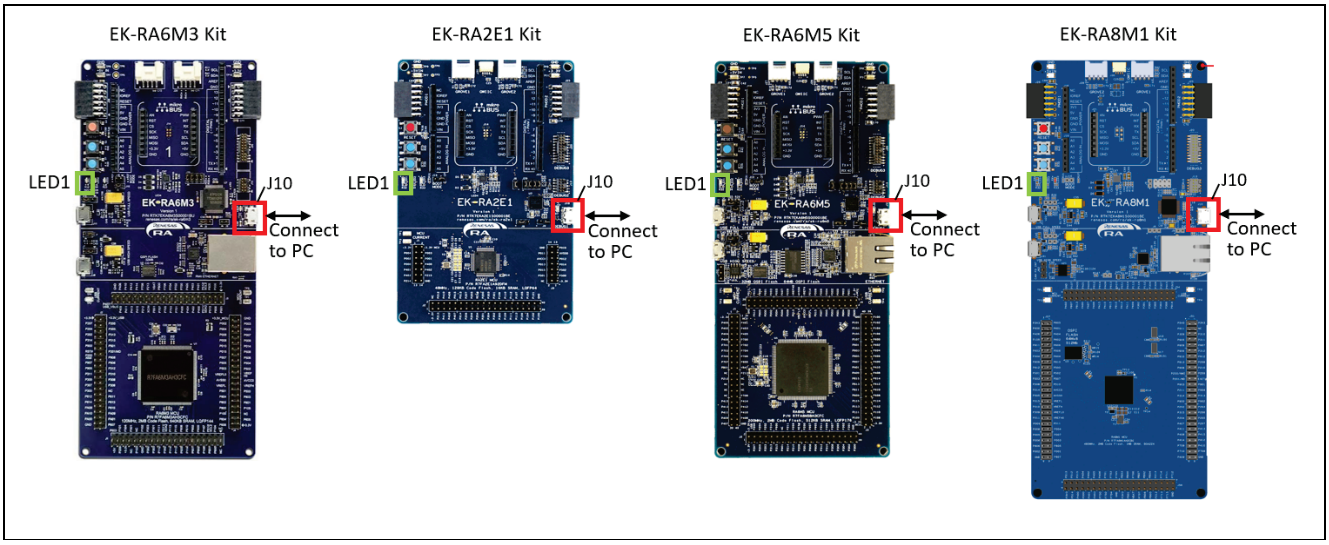


Figure 17. Position of Used LED and Connector

5.5 Overall Algorithms

Figure 18 and Figure 19 shows the overall algorithms of the sample application.

For EK-RA6M3 Flat project, EK-RA2E1 Flat project, EK-RA6M5 Flat project and EK-RA8M1 Flat project, please refer to Figure 18.

For EK-RA6M5 Secure/Non-secure project and EK-RA8M1 Secure/Non-secure project, please refer to Figure 19.

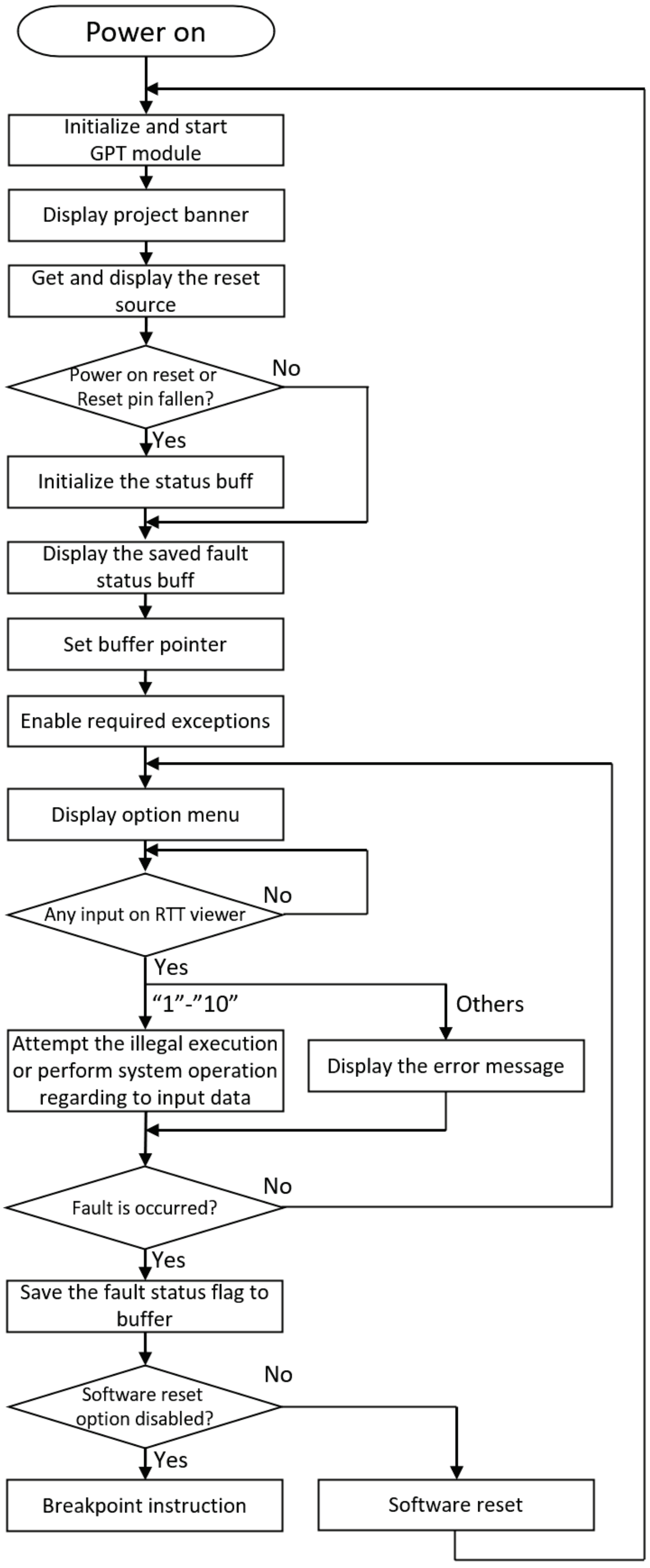

Power on
Initialize and start GPT module
Display project banner
Get and display the reset source
Power on reset or Reset pin fallen?
No
Yes
Initialize the status buff
Display the saved fault status buff
Set buffer pointer
Enable required exceptions
Display option menu
Any input on RTT viewer
No
Yes
“1”-”10”
Others
Attempt the illegal execution or perform system operation regarding to input data
Display the error message
Fault is occurred?
No
Yes
Save the fault status flag to buffer
Software reset option disabled?
No
Yes
Breakpoint instruction
Software reset

Figure 18. Overall algorithm of sample application (EK-RA6M3 Flat project, EK-RA2E1 Flat project, EK-RA6M5 Flat project and EK-RA8M1 Flat project)

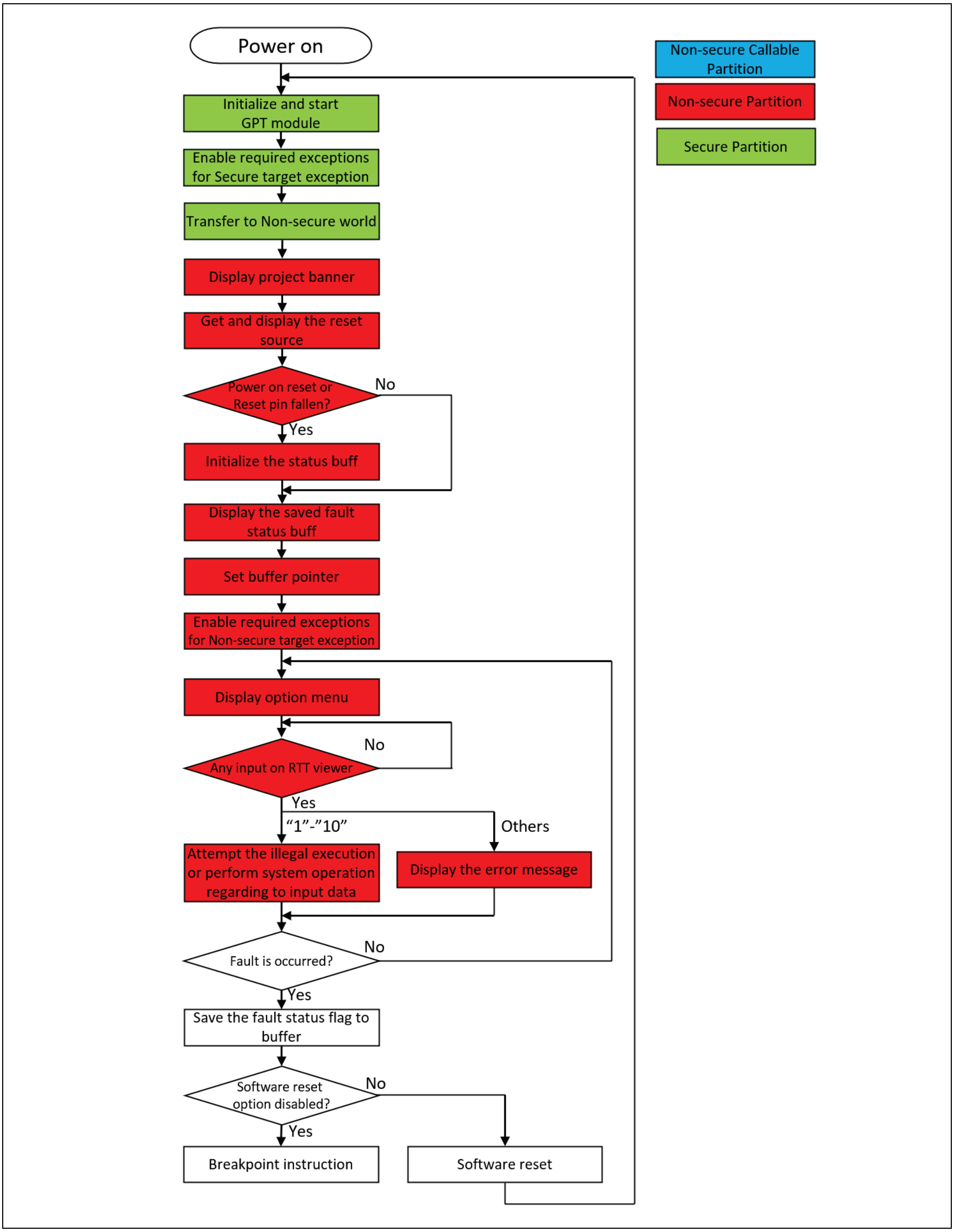

Power on
Initialize and start GPT module
Enable required exceptions for Secure target exception
Transfer to Non-secure world
Display project banner
Get and display the reset source
Power on reset or Reset pin fallen?
No
Yes
Initialize the status buff
Display the saved fault status buff
Set buffer pointer
Enable required exceptions for Non-secure target exception
Display option menu
Any input on RTT viewer
No
Yes
"1"-"10"
Others
Attempt the illegal execution or perform system operation regarding to input data
Display the error message
Fault is occurred?
No
Yes
Save the fault status flag to buffer
Software reset option disabled?
No
Yes
Breakpoint instruction
Software reset
Non-secure Callable Partition
Non-secure Partition
Secure Partition

Figure 19. Overall algorithm of sample application (EK-RA6M5 Secure/Non-secure project and EK-RA8M1 Secure/Non-secure project)

5.6 Checking Procedure

The following steps show the checking procedure of the sample application.

1. Import and build a project. See section 5.6.1.
2. Download program and start debugging. See section 5.6.2.
3. Start the program tracing. See section 5.6.3
4. Press the **Resume** button twice in e^2 studio IDE.
5. Connect to the J-Link RTT Viewer. See section 5.6.4.
6. Enter the user selection to J-Link RTT Viewer and Check the behavior. See section 5.8 for more information on this demonstration.

5.6.1 Import and Build A Project

To build an application project with e^2 studio ISDE, proceed as follows:

1. Launch e^2 studio IDE.
2. Select any workspace in **Workspace launcher**.
3. Close **Welcome** window.
4. Select a **File** > **Import**.
5. Select **Existing Projects into Workspace** from the Import dialog box.
6. Select archive file.
7. Select the project you want to import and click **Finish**.

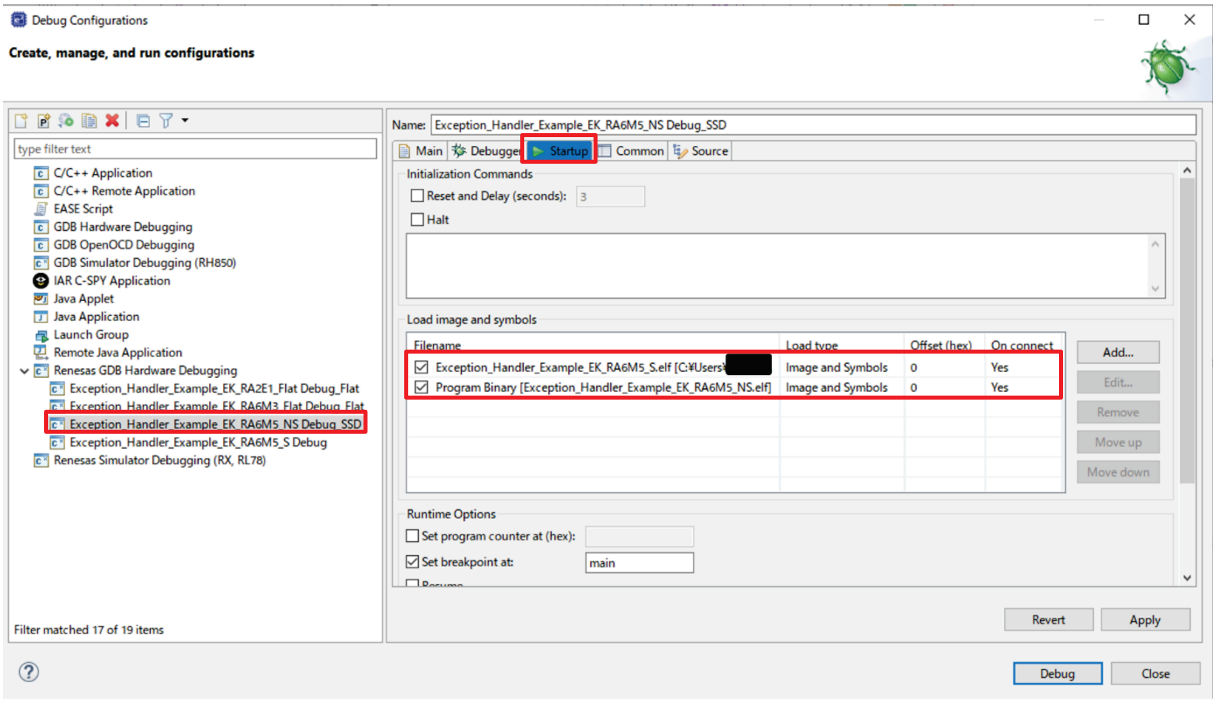
   Note: For the EK-RA6M5 Secure/Non-secure projects and EK-RA8M1 Secure/Non-secure projects, please import both the Non-secure (NS) project and the Secure (S) project.
8. Clicking **Generate Project Content** in the Configurator window.
9. Select **Project** > **Build Project**.

   Note: For the EK-RA6M5 Secure/Non-secure projects and EK-RA8M1 Secure/Non-secure projects, the Secure project (`Exception_Handling_Example_EK_RAxxx_S`) must be built at first. Then build the Non-secure project (`Exception_Handling_Example_EK_RAxxx_NS`).

5.6.2 Download Program and Debug

To download an application project and start debugging, proceed as follows.

1. Connect the J10 connector on each kit and PC with a micro-USB cable.
2. Right-click the project and open **Debug Configuration…** from **Debug As**
   Note: For EK-RA6M5 Secure/Non-secure projects and EK-RA8M1 Secure/Non-secure projects, open the **Debug Configuration** of the Non-secure project (`Exception_Handling_Example_EK_RAxxx_NS`) and make sure that both the secure project program and the Non-secure project program are set to be downloaded (Figure 20).



Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Remote Application
- EASE Script
- GDB Hardware Debugging
- GDB OpenOCD Debugging
- GDB Simulator Debugging (RH850)
- IAR C-SPY Application
- Java Applet
- Java Application
- Launch Group
- Remote Java Application
- Renesas GDB Hardware Debugging
  - Exception_Handler_Example_EK_RA2E1_Flat Debug_Flat
  - Exception_Handler_Example_EK_RA6M3_Flat Debug_Flat
  - Exception_Handler_Example_EK_RA6M5_NS Debug_SSD
  - Exception_Handler_Example_EK_RA6M5_S Debug
- Renesas Simulator Debugging (RX, RL78)

Filter matched 17 of 19 items

Name: Exception_Handler_Example_EK_RA6M5_NS Debug_SSD

Initialization Commands

☐ Reset and Delay (seconds): 3

☐ Halt

Load image and symbols

Filename	Load type	Offset (hex)	On connect
☑ Exception_Handler_Example_EK_RA6M5_S.elf [C:¥Users¥[redacted]	Image and Symbols	0	Yes
☑ Program Binary [Exception_Handler_Example_EK_RA6M5_NS.elf]	Image and Symbols	0	Yes

Runtime Options

☐ Set program counter at (hex):

☑ Set breakpoint at: main

☐ Resume

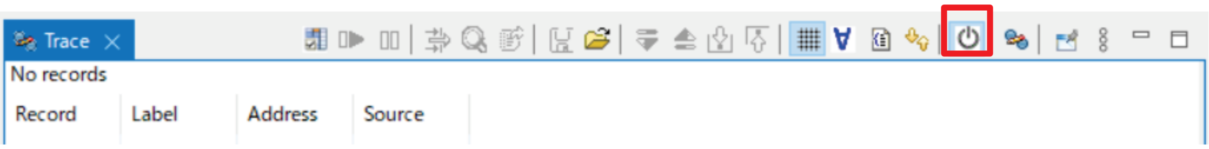
Figure 20. Debug Configuration on EK-RA6M5 Secure/Non-secure projects

3. Click **Debug**.

5.6.3 Start the Program Tracing

To trace the program execution, this demonstration uses the **Trace** view function of e^2 studio IDE. The **Trace** view is turned off by default. It must be turned on before the target system starts. The **Trace** view can be turned on as follows.

1. Open the **Trace** view window.
2. Press the **Turn Trace On/Off** button at the right-top in the window.



No records

Record	Label	Address	Source

Figure 21. Trace view window

5.6.4 Connect to J-Link RTT Viewer

The sample application uses the Segger J-Link RTT Viewer for inputting/outputting data. The J-Link RTT Viewer can be connected as follows.

1. Launch J-Link RTT Viewer.
2. Configure the settings in Configuration page as follows.
   a. EK-RA6M3 Flat project:
      Specify Target Device：R7FA6M3AH
      RTT Control Block：Auto Detection
   b. EK-RA2E1 Flat project:
      Specify Target Device：R7FA2E1A9
      RTT Control Block：Auto Detection
   c. EK-RA6M5 Flat project and Secure/Non-secure projects:
      Specify Target Device：R7FA6M5BH
      RTT Control Block：Search Range (Enter “0x20000000 0x10000” in input box)
   d. EK-RA8M1 Flat project
      Specify Target Device：R7FA8M1AH
      RTT Control Block：Search Range (Enter “0x22000000 0x10000” in input box)
   e. EK-RA8M1 Secure/Non-secure projects:
      Specify Target Device：R7FA8M1AH
      RTT Control Block：Search Range (Enter “0x32000000 0x10000” in input box)

   Note: If an IDE tool such as e^2 studio connects a debug session, **Existing Session** should be selected in Connection to J-Link setting.
3. Select **OK**.
4. Configure the Sending settings to **Send on Enter**.

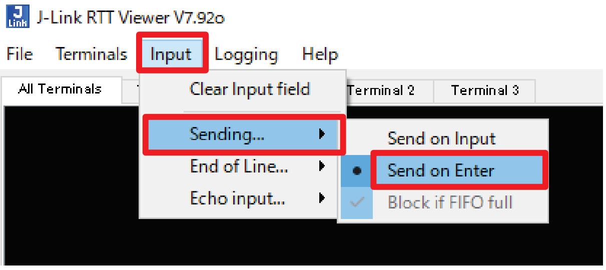
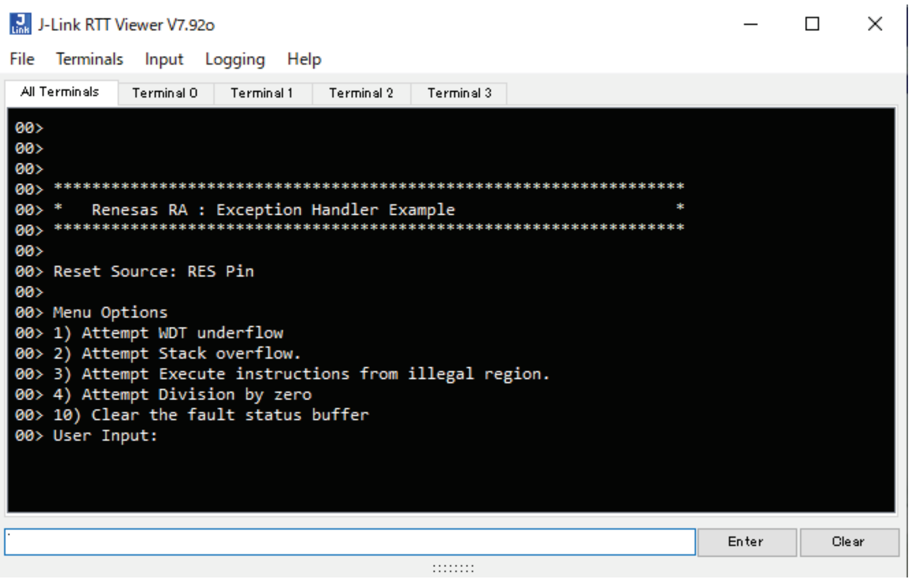


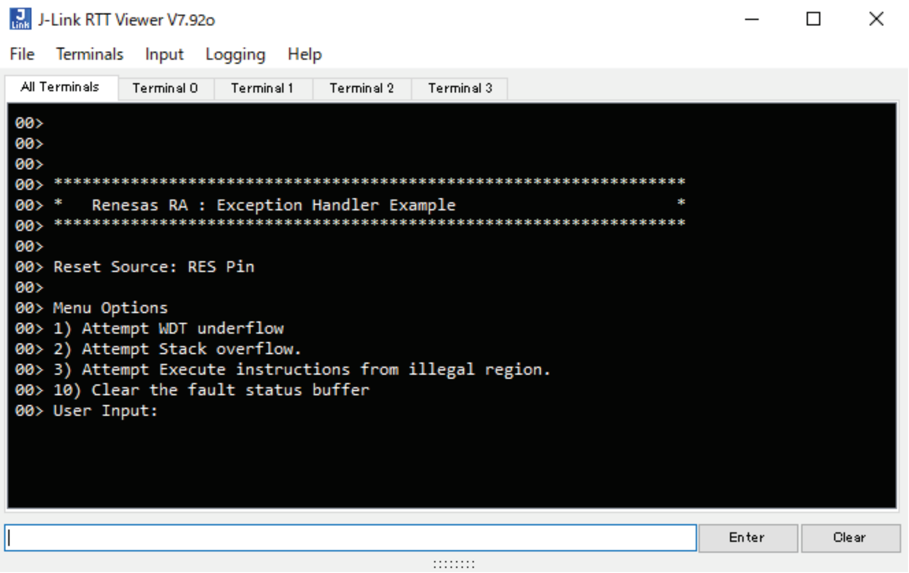
Figure 22. The Sending function on RTT Viewer

After the program starts, if the connection is successful, a startup message (project banner) can be shown as following Figure 23 - Figure 25. .



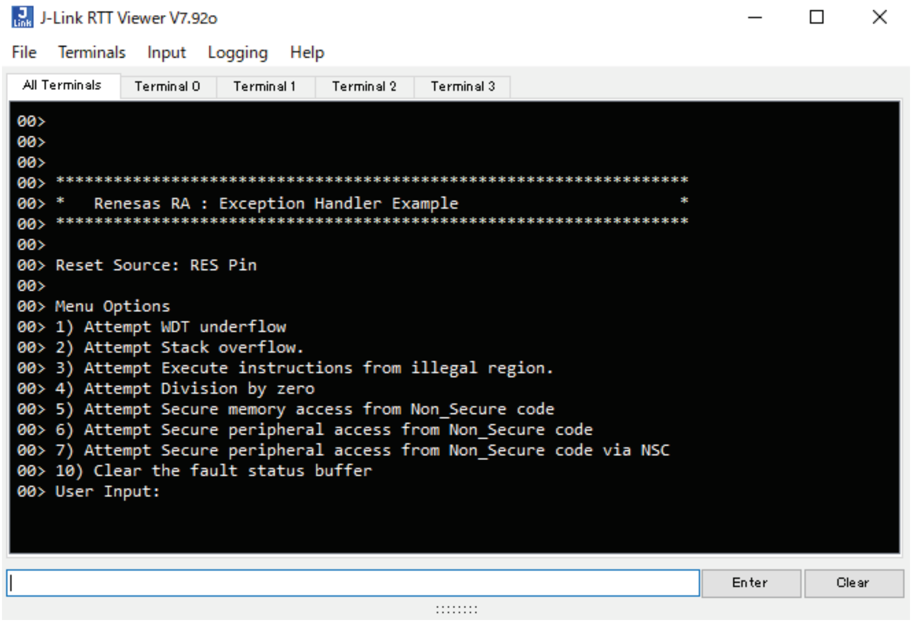
```
00>
00>
00>
00> *****************************************************************
00> *   Renesas RA : Exception Handler Example                      *
00> *****************************************************************
00>
00> Reset Source: RES Pin
00>
00> Menu Options
00> 1) Attempt WDT underflow
00> 2) Attempt Stack overflow.
00> 3) Attempt Execute instructions from illegal region.
00> 4) Attempt Division by zero
00> 10) Clear the fault status buffer
00> User Input:
```

Figure 23. Startup message on J-Link RTT Viewer (EK-RA6M3 Flat project, EK-RA6M5 Flat project and EK-RA8M1 Flat project)



```
00>
00>
00>
00> *****************************************************************
00> *   Renesas RA : Exception Handler Example                      *
00> *****************************************************************
00>
00> Reset Source: RES Pin
00>
00> Menu Options
00> 1) Attempt WDT underflow
00> 2) Attempt Stack overflow.
00> 3) Attempt Execute instructions from illegal region.
00> 10) Clear the fault status buffer
00> User Input:
```

Figure 24. Startup message on J-Link RTT Viewer (EK-RA2E1 Flat project)



J-Link RTT Viewer V7.92o
File Terminals Input Logging Help
All Terminals Terminal 0 Terminal 1 Terminal 2 Terminal 3
Enter
Clear

```
00>
00>
00>
00> ******************************************************************
00> *   Renesas RA : Exception Handler Example                       *
00> ******************************************************************
00>
00> Reset Source: RES Pin
00>
00> Menu Options
00> 1) Attempt WDT underflow
00> 2) Attempt Stack overflow.
00> 3) Attempt Execute instructions from illegal region.
00> 4) Attempt Division by zero
00> 5) Attempt Secure memory access from Non_Secure code
00> 6) Attempt Secure peripheral access from Non_Secure code
00> 7) Attempt Secure peripheral access from Non_Secure code via NSC
00> 10) Clear the fault status buffer
00> User Input:
```

Figure 25. Startup Message on J-Link RTT Viewer (EK-RA6M5 Secure/Non-secure Project and EK-RA8M1 Secure/Non-secure Project)

5.7 Expected Results

This section shows the expected results of actions taken by user input. The results shown in this section are examples of running the application project attached to this application note.

For EK-RA6M3 Flat project, please refer to Table 10.

For EK-RA2E1 Flat project, please refer to Table 11.

For EK-RA6M5 Flat project, please refer to Table 12.

For EK-RA6M5 Secure/Non-secure projects, please refer to Table 13.

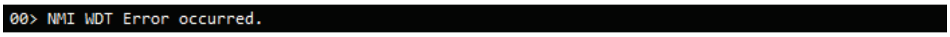
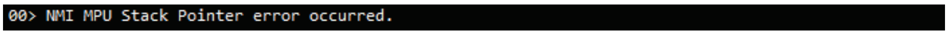
For EK-RA8M1 Flat project, please refer to Table 14.

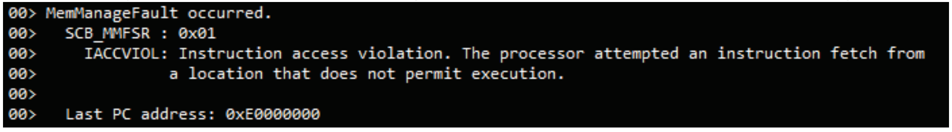
For EK-RA8M1 Secure/Non-secure projects, please refer to Table 15.

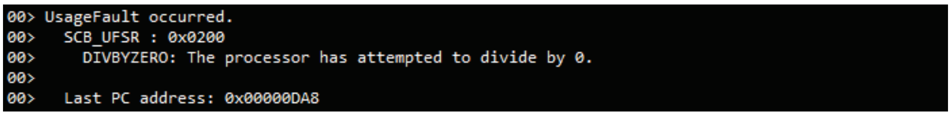
Note: The program address in results will be changed if you change the configurations

Note: The results in RTT Viewer are displayed after the target system restarts, except for successful messages.

Table 10. Expected Result on the EK-RA6M3 Flat Project

RTT Viewer Input and Action	Expected Result
“1” WDT underflow	• NMI has occurred. • An exception occurred in `bsp_prv_software_delay_loop()` in `bsp_delay.c` • The address that the exception occurs in is not recognized from the buffer data because this sample application acquires the last program counter only when the fault occurs. • The results in RTT Viewer are as follows.  `00> NMI WDT Error occurred.`
“2” Stack overflow	• NMI has occurred. • An exception occurred at the address 0xDC0 (Line 164 in `common.c`) • The address that the exception occurs in is not recognized from the buffer data because this sample application acquires the last program counter only when the fault occurs. • The results in RTT Viewer are as follows.  `00> NMI MPU Stack Pointer error occurred.`

“3”
Execute instruction from illegal region
	• MemManage exception has occurred.
	• An exception occurred at the address 0xDB8 (Line 152 in `app_common.c`)
	• The address that the exception occurs in is not recognized from the stack memory that stores the last program counter because this is the program
	
	`00> MemManageFault occurred.`
	`00>   SCB_MMFSR : 0x01`
	`00>     IACCVIOL: Instruction access violation. The processor attempted an instruction fetch from`
	`00>             a location that does not permit execution.`
	`00>`
	`00>   Last PC address: 0xE0000000`

“4”
Divide by zero
	• UsageFault exception has occurred.
	• An exception occurred at the address 0xDA8 (Line 137 in app_common.c)
	• The results in RTT Viewer are as follows.
	
	`00> UsageFault occurred.`
	`00>   SCB_UFSR : 0x0200`
	`00>     DIVBYZERO: The processor has attempted to divide by 0.`
	`00>`
	`00>   Last PC address: 0x00000DA8`

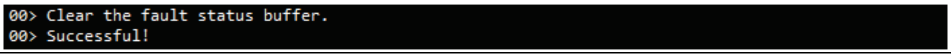
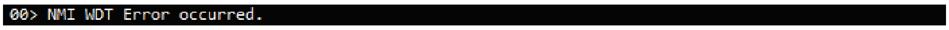
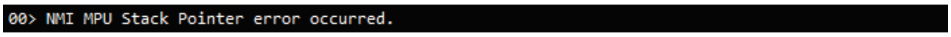
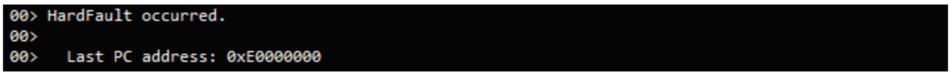
“10”
Clear the fault status buffer
	• No exception occurred.
	• The results in RTT Viewer are as follows.
	
	`00> Clear the fault status buffer.`
	`00> Successful!`

Table 11. Expected Result on the EK-RA2E1 Flat Project

RTT Viewer Input and Action	Expected Result
“1” WDT underflow	• NMI has occurred. • An exception occurred in `bsp_prv_software_delay_loop()` in `bsp_delay.c` • The address that the exception occurs in is not recognized from the buffer data because this sample application acquires the last program counter only when the fault occurs. • The results in RTT Viewer are as follows.  `00> NMI WDT Error occurred.`
“2” Stack overflow	• NMI has occurred. • An exception occurred at the address 0xDB8 (Line 164 in `common.c`) • The address that the exception occurs in is not recognized from the buffer data because this sample application acquires the last program counter only when the fault occurs. • The results in RTT Viewer are as follows.  `00> NMI MPU Stack Pointer error occurred.`

“3”
Execute instruction from illegal region
	• HardFault exception has occurred.
	• An exception occurred at the address 0xDAC (Line 152 in `app_common.c`)
	• The address that the exception occurs in is not recognized from the stack memory that stores the last program counter because this is the program execution error. The above address can be found from the results of the program tracing.
	• The results in RTT Viewer are as follows.
	
	`00> HardFault occurred.`
	`00>`
	`00>   Last PC address: 0xE0000000`

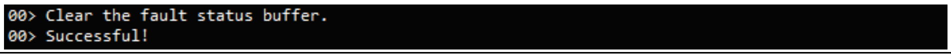
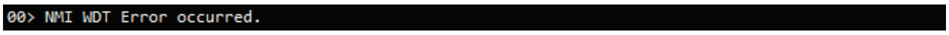
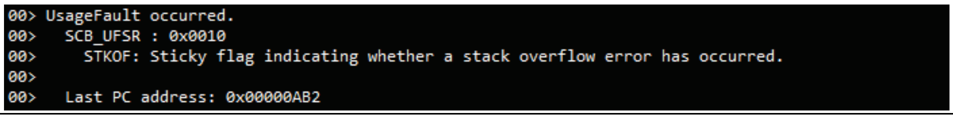
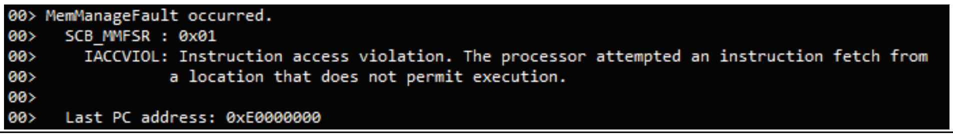
“10”
Clear the fault status buffer
	• No exception occurred.
	• The results in RTT Viewer are as follows.
	
	`00> Clear the fault status buffer.`
	`00> Successful!`

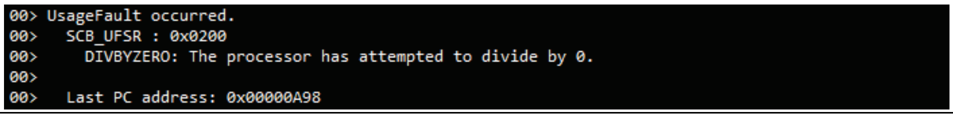
Table 12. Expected Result on the EK-RA6M5 Flat Project

RTT Viewer Input and Action	Expected Result
“1” WDT underflow	• NMI has occurred. • An exception occurred in `bsp_prv_software_delay_loop()` in `bsp_delay.c` • The address that the exception occurs in is not recognized from the buffer data because this sample application acquires the last program counter only when the fault occurs. • The results in RTT Viewer are as follows.  `00> NMI WDT Error occurred.`

“2”
Stack overflow
	• UsageFault exception has occurred.
	• An exception occurred at the address 0xAB2 (Line 164 in `app_common.c`)
	• The results in RTT Viewer are as follows.
	
	`00> UsageFault occurred.`
	`00>   SCB_UFSR : 0x0010`
	`00>     STKOF: Sticky flag indicating whether a stack overflow error has occurred.`
	`00>`
	`00>   Last PC address: 0x00000AB2`

RTT Viewer Input and Action	Expected Result

“3”
Execute instruction from illegal region
	• MemManage exception has occurred.
	• An exception occurred at the address 0xAA8 (Line 152 in `app_common.c`)
	• The address that the exception occurs in is not recognized from the stack memory that stores the last program counter because this is the program execution error. The above address can be found from the results of the program tracing.
	• The results in RTT Viewer are as follows.
	
	`00> MemManageFault occurred.`
	`00>    SCB_MMFSR : 0x01`
	`00>      IACCVIOL: Instruction access violation. The processor attempted an instruction fetch from`
	`00>               a location that does not permit execution.`
	`00>`
	`00>    Last PC address: 0xE0000000`

“4”
Divide by zero
	• UsageFault exception has occurred.
	• An exception occurred at the address 0xA98 (Line 137 in `app_common.c`)
	• The results in RTT Viewer are as follows.
	
	`00> UsageFault occurred.`
	`00>    SCB_UFSR : 0x0200`
	`00>      DIVBYZERO: The processor has attempted to divide by 0.`
	`00>`
	`00>    Last PC address: 0x00000A98`

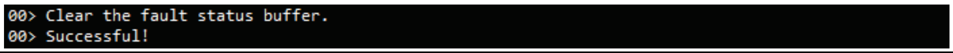
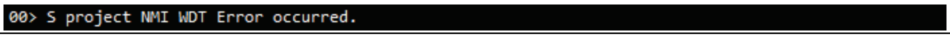
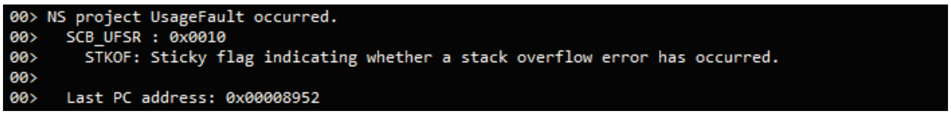
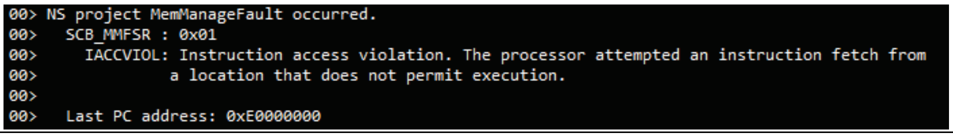
“10”
Clear the fault status buffer
	• No exception occurred.
	• The results in RTT Viewer are as follows.
	
	`00> Clear the fault status buffer.`
	`00> Successful!`

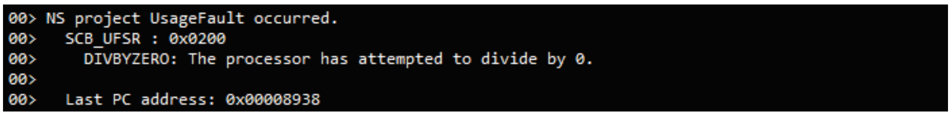
Table 13. Expected Result on the EK-RA6M5 Secure/Non-secure Project

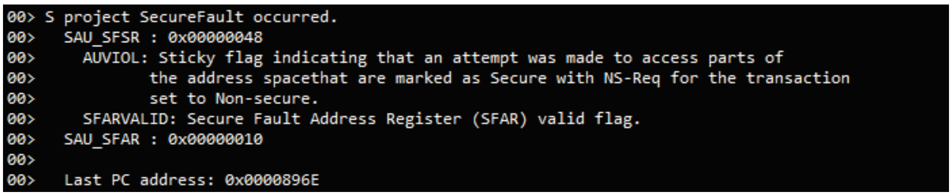
RTT Viewer Input and Action	Expected Result
“1” WDT underflow	• Secure world NMI has occurred. • An exception occurred in `bsp_prv_software_delay_loop()` in `bsp_delay.c` • The address that the exception occurs in is not recognized from the buffer data because this sample application acquires the last program counter only when the fault occurs. • The results in RTT Viewer are as follows.  `00> S project NMI WDT Error occurred.`

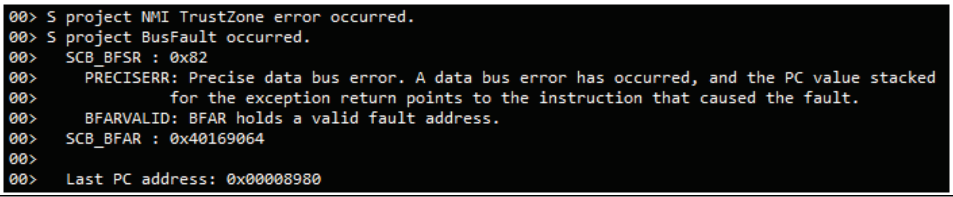
“2”
Stack overflow
	• Non-secure world UsageFault exception has occurred.
	• An exception occurred at the address 0x8952 (Line 164 in `app_common.c`)
	• The results in RTT Viewer are as follows.
	
	`00> NS project UsageFault occurred.`
	`00>    SCB_UFSR : 0x0010`
	`00>      STKOF: Sticky flag indicating whether a stack overflow error has occurred.`
	`00>`
	`00>    Last PC address: 0x00008952`

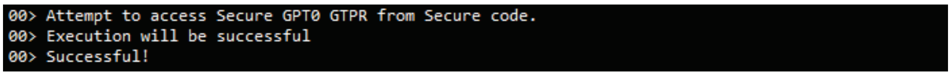
“3”
Execute instruction from illegal region
	• Non-secure world MemManage exception has occurred.
	• An exception occurred at the address 0x8948 (Line 152 in `app_common.c`)
	• The address that the exception occurs in is not recognized from the stack memory that stores the last program counter because this is the program execution error. The above address can be found from the results of the program tracing.
	• The results in RTT Viewer are as follows.
	
	`00> NS project MemManageFault occurred.`
	`00>    SCB_MMFSR : 0x01`
	`00>      IACCVIOL: Instruction access violation. The processor attempted an instruction fetch from`
	`00>               a location that does not permit execution.`
	`00>`
	`00>    Last PC address: 0xE0000000`

RTT Viewer Input and Action	Expected Result

“4”
Divide by zero
	• Non-secure world UsageFault exception has occurred.
	• An exception occurred at the address 0x8938 (Line 137 in `app_common.c`)
	• The results in RTT Viewer are as follows.
	
	`00> NS project UsageFault occurred.`
	`00>   SCB_UFSR : 0x0200`
	`00>     DIVBYZERO: The processor has attempted to divide by 0.`
	`00>`
	`00>   Last PC address: 0x00008938`

“5”
Access Secure attribute memory from Non-secure code
	• Secure world SecureFault exception has occurred.
	• An exception occurred at the address 0x896E (Line 179 in `app_common.c`)
	• The results in RTT Viewer are as follows.
	
	`00> S project SecureFault occurred.`
	`00>   SAU_SFSR : 0x00000048`
	`00>     AUVIOL: Sticky flag indicating that an attempt was made to access parts of`
	`00>            the address spacethat are marked as Secure with NS-Req for the transaction`
	`00>            set to Non-secure.`
	`00>     SFARVALID: Secure Fault Address Register (SFAR) valid flag.`
	`00>   SAU_SFAR : 0x00000010`
	`00>`
	`00>   Last PC address: 0x0000896E`

“6”
Access Secure attribute peripheral register from Non-secure code
	• Secure world NMI and BudFault exception has occurred.
	• An exception occurred at the address 0x8980 (Line 197 in `app_common.c`)
	• The results in RTT Viewer are as follows.
	
	`00> S project NMI TrustZone error occurred.`
	`00> S project BusFault occurred.`
	`00>   SCB_BFSR : 0x82`
	`00>     PRECISERR: Precise data bus error. A data bus error has occurred, and the PC value stacked`
	`00>              for the exception return points to the instruction that caused the fault.`
	`00>     BFARVALID: BFAR holds a valid fault address.`
	`00>   SCB_BFAR : 0x40169064`
	`00>`
	`00>   Last PC address: 0x00008980`

“7”
Access Secure attribute peripheral register from Non-secure code via NSC
	• No exception occurred.
	• The results in RTT Viewer are as follows.
	
	`00> Attempt to access Secure GPT0 GTPR from Secure code.`
	`00> Execution will be successful`
	`00> Successful!`

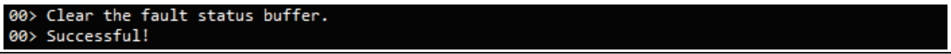
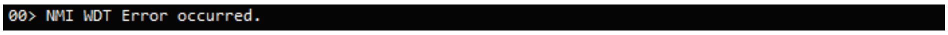
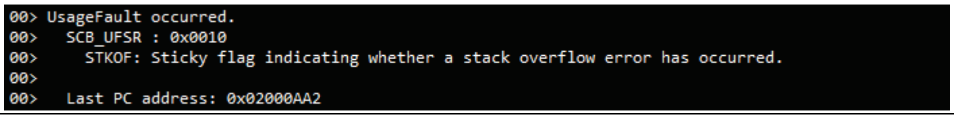
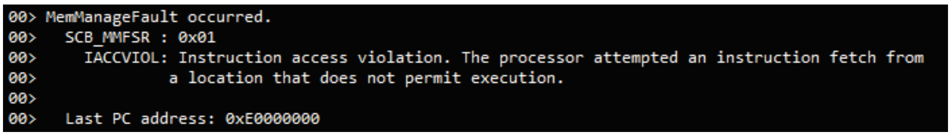
“10”
Clear the fault status buffer
	• No exception occurred.
	• The results in RTT Viewer are as follows.
	
	`00> Clear the fault status buffer.`
	`00> Successful!`

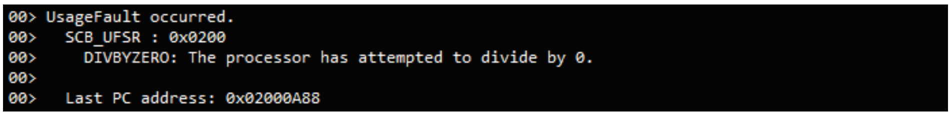
Table 14. Expected Result on the EK-RA8M1 Flat Project

RTT Viewer Input and Action	Expected Result
“1” WDT underflow	• NMI has occurred. • An exception occurred in `bsp_prv_software_delay_loop()` in `bsp_delay.c` • The address that the exception occurs in is not recognized from the buffer data because this sample application acquires the last program counter only when the fault occurs. • The results in RTT Viewer are as follows.  `00> NMI WDT Error occurred.`

“2”
Stack overflow
	• UsageFault exception has occurred.
	• An exception occurred at the address 0x2000AA2 (Line 164 in `app_common.c`)
	• The results in RTT Viewer are as follows.
	
	`00> UsageFault occurred.`
	`00>   SCB_UFSR : 0x0010`
	`00>     STKOF: Sticky flag indicating whether a stack overflow error has occurred.`
	`00>`
	`00>   Last PC address: 0x02000AA2`

RTT Viewer Input and Action	Expected Result

"3"
Execute instruction from illegal region
	- MemManage exception has occurred.
	- An exception occurred at the address 0x2000A98 (Line 152 in `app_common.c`)
	- The address that the exception occurs in is not recognized from the stack memory that stores the last program counter because this is the program execution error. The above address can be found from the results of the program tracing.
	- The results in RTT Viewer are as follows.
	
	`00> MemManageFault occurred.`
	`00>   SCB_MMFSR : 0x01`
	`00>     IACCVIOL: Instruction access violation. The processor attempted an instruction fetch from`
	`00>               a location that does not permit execution.`
	`00>`
	`00>   Last PC address: 0xE0000000`

"4"
Divide by zero
	- UsageFault exception has occurred.
	- An exception occurred at the address 0x2000A88 (Line 137 in `app_common.c`)
	- The results in RTT Viewer are as follows.
	
	`00> UsageFault occurred.`
	`00>   SCB_UFSR : 0x0200`
	`00>     DIVBYZERO: The processor has attempted to divide by 0.`
	`00>`
	`00>   Last PC address: 0x02000A88`

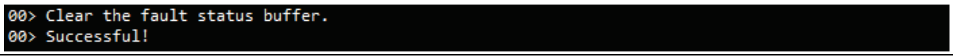
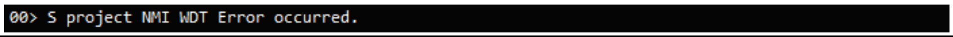
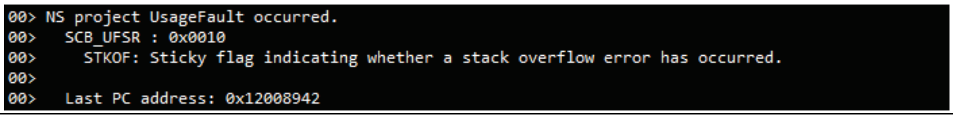
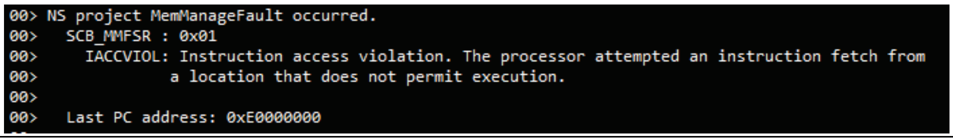
"10"
Clear the fault status buffer
	- No exception occurred.
	- The results in RTT Viewer are as follows.
	
	`00> Clear the fault status buffer.`
	`00> Successful!`

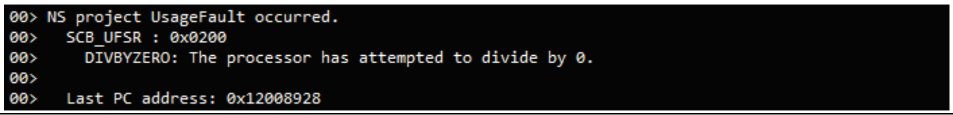
Table 15. Expected Result on the EK-RA8M1 Secure/Non-secure Project

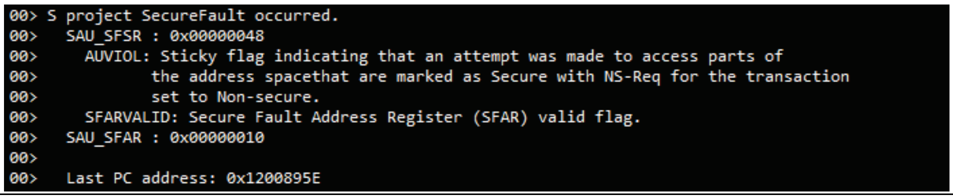
RTT Viewer Input and Action	Expected Result
"1" WDT underflow	- Secure world NMI has occurred. - An exception occurred in `bsp_prv_software_delay_loop()` in `bsp_delay.c` - The address that the exception occurs in is not recognized from the buffer data because this sample application acquires the last program counter only when the fault occurs. - The results in RTT Viewer are as follows.  `00> S project NMI WDT Error occurred.`

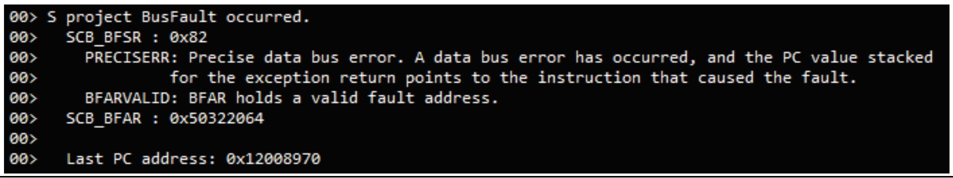
"2"
Stack overflow
	- Non-secure world UsageFault exception has occurred.
	- An exception occurred at the address 0x12008942 (Line 164 in `app_common.c`)
	- The results in RTT Viewer are as follows.
	
	`00> NS project UsageFault occurred.`
	`00>   SCB_UFSR : 0x0010`
	`00>     STKOF: Sticky flag indicating whether a stack overflow error has occurred.`
	`00>`
	`00>   Last PC address: 0x12008942`

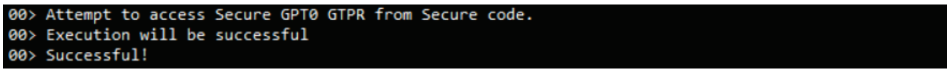
RTT Viewer Input and Action	Expected Result

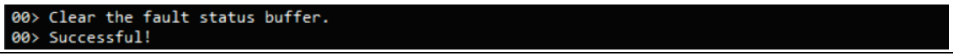
“3”
Execute instruction from illegal region
	• Non-secure world MemManage exception has occurred.
	• An exception occurred at the address 0x12008938 (Line 152 in `app_common.c`)
	• The address that the exception occurs in is not recognized from the stack memory that stores the last program counter because this is the program execution error. The above address can be found from the results of the program tracing.
	• The results in RTT Viewer are as follows.
	
	`00> NS project MemManageFault occurred.`
	`00>   SCB_MMFSR : 0x01`
	`00>     IACCVIOL: Instruction access violation. The processor attempted an instruction fetch from`
	`00>              a location that does not permit execution.`
	`00>`
	`00>   Last PC address: 0xE0000000`

“4”
Divide by zero
	• Non-secure world UsageFault exception has occurred.
	• An exception occurred at the address 0x12008928 (Line 137 in `app_common.c`)
	• The results in RTT Viewer are as follows.
	
	`00> NS project UsageFault occurred.`
	`00>   SCB_UFSR : 0x0200`
	`00>     DIVBYZERO: The processor has attempted to divide by 0.`
	`00>`
	`00>   Last PC address: 0x12008928`

“5”
Access Secure attribute memory from Non-secure code
	• Secure world SecureFault exception has occurred.
	• An exception occurred at the address 0x1200895E (Line 179 in `app_common.c`)
	• The results in RTT Viewer are as follows.
	
	`00> S project SecureFault occurred.`
	`00>   SAU_SFSR : 0x00000048`
	`00>     AUVIOL: Sticky flag indicating that an attempt was made to access parts of`
	`00>            the address spacethat are marked as Secure with NS-Req for the transaction`
	`00>            set to Non-secure.`
	`00>     SFARVALID: Secure Fault Address Register (SFAR) valid flag.`
	`00>   SAU_SFAR : 0x00000010`
	`00>`
	`00>   Last PC address: 0x1200895E`

“6”
Access Secure attribute peripheral register from Non-secure code
	• Secure world BusFault exception has occurred.
	• An exception occurred at the address 0x12008970 (Line 197 in `app_common.c`)
	• The results in RTT Viewer are as follows.
	
	`00> S project BusFault occurred.`
	`00>   SCB_BFSR : 0x82`
	`00>     PRECISERR: Precise data bus error. A data bus error has occurred, and the PC value stacked`
	`00>              for the exception return points to the instruction that caused the fault.`
	`00>     BFARVALID: BFAR holds a valid fault address.`
	`00>   SCB_BFAR : 0x50322064`
	`00>`
	`00>   Last PC address: 0x12008970`

“7”
Access Secure attribute peripheral register from Non-secure code via NSC
	• No exception occurred.
	• The results in RTT Viewer are as follows.
	
	`00> Attempt to access Secure GPT0 GTPR from Secure code.`
	`00> Execution will be successful`
	`00> Successful!`

“10”
Clear the fault status buffer
	• No exception occurred.
	• The results in RTT Viewer are as follows.
	
	`00> Clear the fault status buffer.`
	`00> Successful!`

5.8 Demonstration

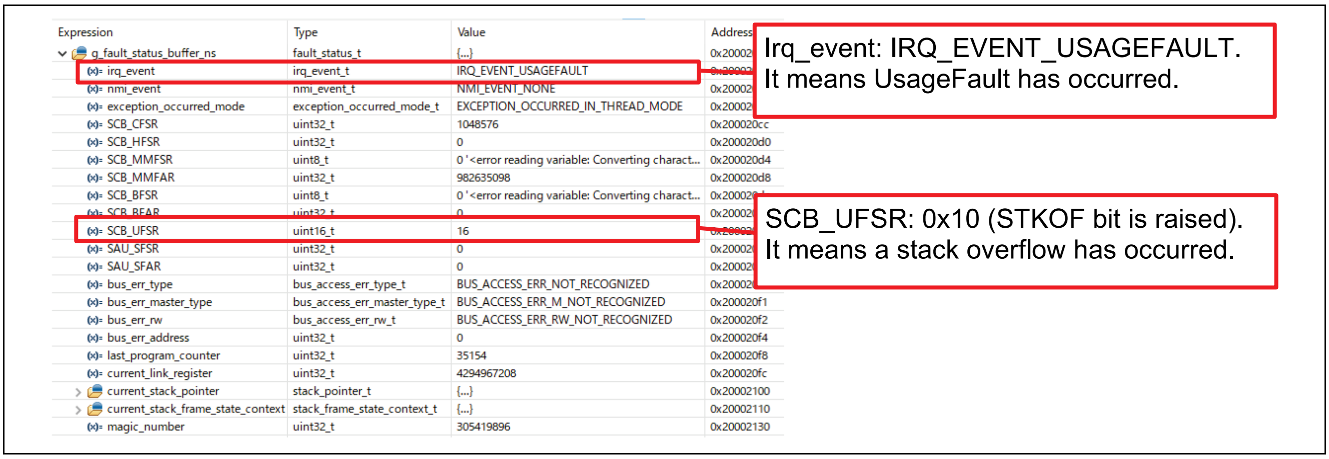
This section demonstrates how to determine the root cause from the related registers and the program tracing results, for the EK-RA6M5 Secure/Non-secure projects and the following configurations.

Configurations	Selected value
Software reset option in `user_exception_handler.h`	Not enabled
RA6M5 Family → Security → Exceptions → Exception Response	Non-Maskable Interrupt
RA6M5 Family → Security → Exceptions →BusFault, HardFault, and NMI Target	Secure State
DLM Stage	SSD (Secure Software Development)

5.8.1 Demo 1: Attempt Stack Overflow

Demo 1 shows the demonstration result when attempting a stack overflow.

1. Run the MCU. LED1 blinks.
2. Enter **2** to the input box of the RTT Viewer.
3. System attempts to cause the stack overflow. Then, the system will stop in the NS project `UsageFault_handler()`.
4. Confirm the saved fault status in the buffer to determine the fault factor.



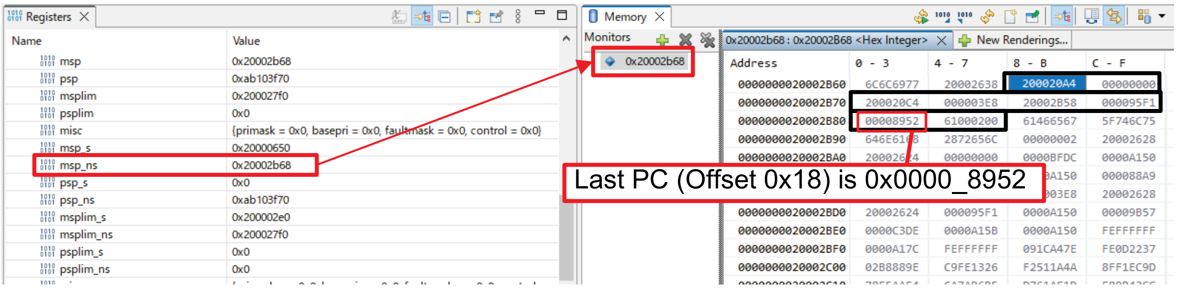
Expression	Type	Value	Address
g_fault_status_buffer_ns	fault_status_t	{...}	0x20002
irq_event	irq_event_t	IRQ_EVENT_USAGEFAULT	
nmi_event	nmi_event_t	NMI_EVENT_NONE	0x20002
exception_occurred_mode	exception_occurred_mode_t	EXCEPTION_OCCURRED_IN_THREAD_MODE	0x20002
SCB_CFSR	uint32_t	1048576	0x200020cc
SCB_HFSR	uint32_t	0	0x200020d0
SCB_MMFSR	uint8_t	0 '<error reading variable: Converting charact...	0x200020d4
SCB_MMFAR	uint32_t	982635098	0x200020d8
SCB_BFSR	uint8_t	0 '<error reading variable: Converting charact...	0x20002
SCB_BFAR	uint32_t	0	0x20002
SCB_UFSR	uint16_t	16	
SAU_SFSR	uint32_t	0	0x20002
SAU_SFAR	uint32_t	0	0x20002
bus_err_type	bus_access_err_type_t	BUS_ACCESS_ERR_NOT_RECOGNIZED	0x20002
bus_err_master_type	bus_access_err_master_type_t	BUS_ACCESS_ERR_M_NOT_RECOGNIZED	0x200020f1
bus_err_rw	bus_access_err_rw_t	BUS_ACCESS_ERR_RW_NOT_RECOGNIZED	0x200020f2
bus_err_address	uint32_t	0	0x200020f4
last_program_counter	uint32_t	35154	0x200020f8
current_link_register	uint32_t	4294967208	0x200020fc
current_stack_pointer	stack_pointer_t	{...}	0x20002100
current_stack_frame_state_context	stack_frame_state_context_t	{...}	0x20002110
magic_number	uint32_t	305419896	0x20002130

Irq_event: IRQ_EVENT_USAGEFAULT.
It means UsageFault has occurred.

SCB_UFSR: 0x10 (STKOF bit is raised).
It means a stack overflow has occurred.

Figure 26. Saved Fault Status

5. Confirm the stack pointer to determine the location of the root cause.
   For RA-CM4 and RA-CM23 project, it can be confirmed from the value “ms” (main stack pointer) or “psp” (process stack pointer) on **Registers** view. For RA-CM33 and RA-CM85 Secure/Non-secure projects, the stack pointer separates between Secure and Non-secure state. Therefore, we should check four stack memories pointed to by msp_s (main stack pointer in secure world), the msp_ns (main stack pointer in Non-secure world), psp_s (process stack pointer in Secure world) and psp_ns (process stack pointer in Non-secure world). For an RA-CM33 and RA-CM85 Flat project, the Non-secure stack pointer can be ignored since the program is always running in Secure state.
   The stack pointer containing the last program counter can be recognized by the EXC_RETURN value in current link register. Please refer to the section “Exception return” in the processor core’s generic user guide for more details.
   The following figure shows the demonstration that finds the last PC from the stack memory msp_ns points to using **Memory** view.



Name	Value
msp	0x20002b68
psp	0xab103f70
msplim	0x200027f0
psplim	0x0
misc	{primask = 0x0, basepri = 0x0, faultmask = 0x0, control = 0x0}
msp_s	0x20000650
msp_ns	0x20002b68
psp_s	0x0
psp_ns	0xab103f70
msplim_s	0x200002e0
msplim_ns	0x200027f0
psplim_s	0x0
psplim_ns	0x0

Monitors: 0x20002b68

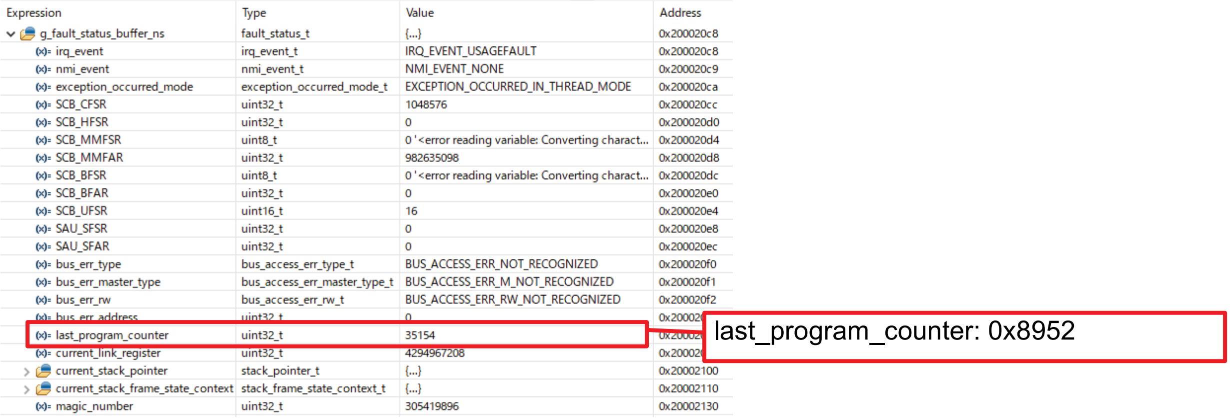
0x20002b68 : 0x20002B68 <Hex Integer>

```
Address            0 - 3     4 - 7     8 - B     C - F
0000000020002B60   6C6C6977  20002638  200020A4  00000000
0000000020002B70   200020C4  000003E8  20002B58  000095F1
0000000020002B80   00008952  61000200  61466567  5F746C75
0000000020002B90   646E6108  2872656C  00000002  20002628
0000000020002BA0   20002624  00000000  0000BFDC  0000A150
                                       0A150     000088A9
                                       03E8      20002628
0000000020002BD0   20002624  000095F1  0000A150  00009B57
0000000020002BE0   0000C3DE  0000A15B  0000A150  FEFFFFFF
0000000020002BF0   0000A17C  FEFFFFFF  091CA47E  FE0D2237
0000000020002C00   02B8889E  C9FE1326  F2511A4A  8FF1EC9D
```

Last PC (Offset 0x18) is 0x0000_8952

Figure 27. Finding the Last PC from msp_ns

The sample application also implements the last program counter acquisition. So, we can find that result in the buffer as follows.



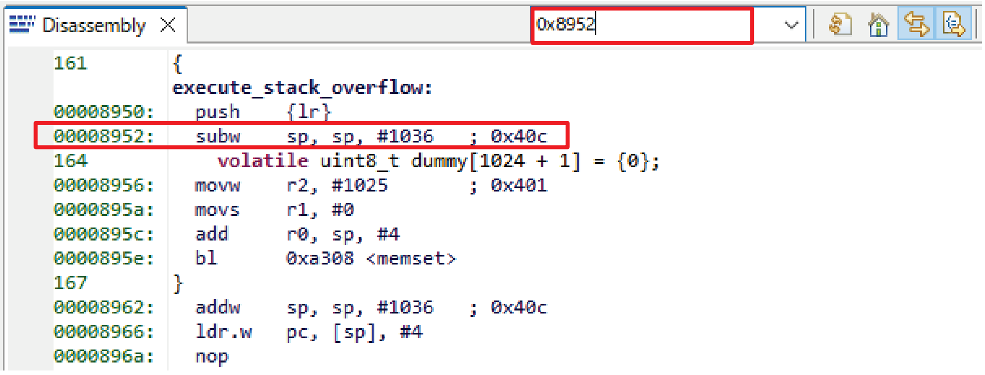
Expression	Type	Value	Address
g_fault_status_buffer_ns	fault_status_t	{...}	0x200020c8
irq_event	irq_event_t	IRQ_EVENT_USAGEFAULT	0x200020c8
nmi_event	nmi_event_t	NMI_EVENT_NONE	0x200020c9
exception_occurred_mode	exception_occurred_mode_t	EXCEPTION_OCCURRED_IN_THREAD_MODE	0x200020ca
SCB_CFSR	uint32_t	1048576	0x200020cc
SCB_HFSR	uint32_t	0	0x200020d0
SCB_MMFSR	uint8_t	0 '<error reading variable: Converting charact...	0x200020d4
SCB_MMFAR	uint32_t	982635098	0x200020d8
SCB_BFSR	uint8_t	0 '<error reading variable: Converting charact...	0x200020dc
SCB_BFAR	uint32_t	0	0x200020e0
SCB_UFSR	uint16_t	16	0x200020e4
SAU_SFSR	uint32_t	0	0x200020e8
SAU_SFAR	uint32_t	0	0x200020ec
bus_err_type	bus_access_err_type_t	BUS_ACCESS_ERR_NOT_RECOGNIZED	0x200020f0
bus_err_master_type	bus_access_err_master_type_t	BUS_ACCESS_ERR_M_NOT_RECOGNIZED	0x200020f1
bus_err_rw	bus_access_err_rw_t	BUS_ACCESS_ERR_RW_NOT_RECOGNIZED	0x200020f2
bus_err_address	uint32_t	0	0x20002
last_program_counter	uint32_t	35154	0x20002
current_link_register	uint32_t	4294967208	0x20002
current_stack_pointer	stack_pointer_t	{...}	0x20002100
current_stack_frame_state_context	stack_frame_state_context_t	{...}	0x20002110
magic_number	uint32_t	305419896	0x20002130

last_program_counter: 0x8952

Figure 28. Finding the Last PC from Buffer

From the above results, we found the last executed program address (0x0000_8952).

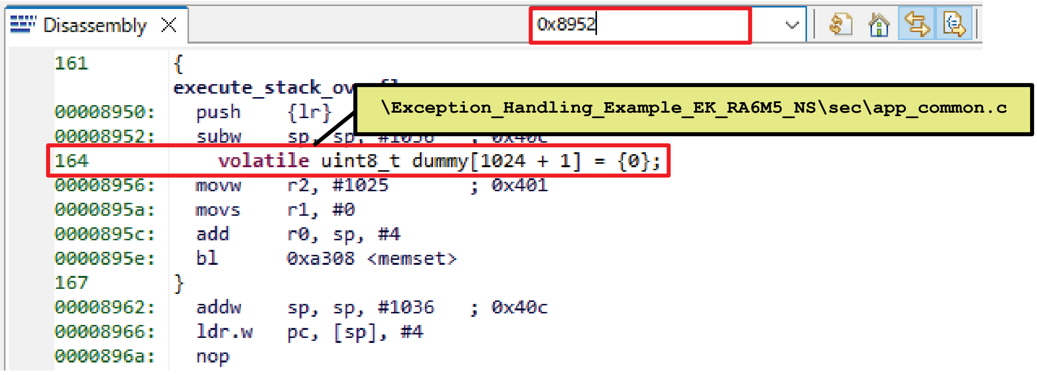
6. Determine the source code from the last executed program address.
   Open the **Disassembly** view and enter the last program counter address **0x8952** to the input box in top of this view. The view will disassemble the program and highlight the related code as follows.



```
161         {
            execute_stack_overflow:
00008950:     push      {lr}
00008952:     subw      sp, sp, #1036    ; 0x40c
164             volatile uint8_t dummy[1024 + 1] = {0};
00008956:     movw      r2, #1025        ; 0x401
0000895a:     movs      r1, #0
0000895c:     add       r0, sp, #4
0000895e:     bl        0xa308 <memset>
167         }
00008962:     addw      sp, sp, #1036    ; 0x40c
00008966:     ldr.w     pc, [sp], #4
0000896a:     nop
```

Figure 29. Disassembling Code

You can find the source code and that line number by scrolling up/down a bit from the highlighted line. Also, when you hover the mouse cursor on the line number of the source code, you can find the file name such as `\Exception_Handling_Example_EK_RA6M5_NS\sec\app_common.c`.



```
161          {
             execute_stack_ov
00008950:      push     {lr}
00008952:      subw     sp, sp, #1036     ; 0x40c
164              volatile uint8_t dummy[1024 + 1] = {0};
00008956:      movw     r2, #1025         ; 0x401
0000895a:      movs     r1, #0
0000895c:      add      r0, sp, #4
0000895e:      bl       0xa308 <memset>
167          }
00008962:      addw     sp, sp, #1036     ; 0x40c
00008966:      ldr.w    pc, [sp], #4
0000896a:      nop
```

\Exception_Handling_Example_EK_RA6M5_NS\sec\app_common.c

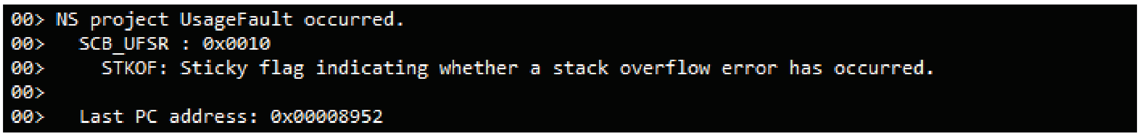
Figure 30. Finding the File Name

Note: If you cannot see the source code, press the **Show Source** button in right-top in this view.

From above, we can find the root cause of the fault that occurred.

- Factor: a stack pointer overflow
- Location: the program address 0x0000_8952 (Line 164 in `app_common.c` file)

Also, after the system is restarted by pressing the **Reset** button and **Resume** button on e^2 studio IDE, the details of the saved fault status are displayed on RTT viewer as follows.



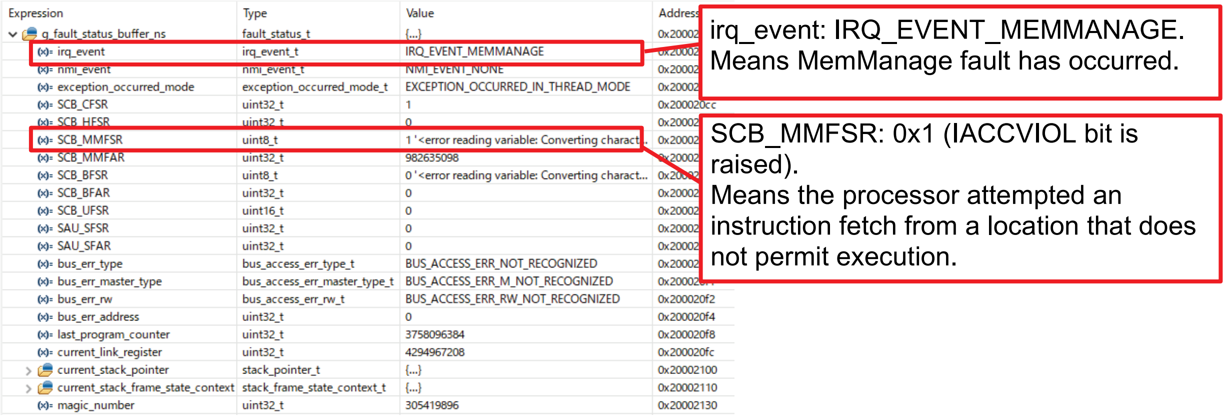
```
00> NS project UsageFault occurred.
00>   SCB_UFSR : 0x0010
00>     STKOF: Sticky flag indicating whether a stack overflow error has occurred.
00>
00>   Last PC address: 0x00008952
```

Figure 31. Saved Fault Status after Restart

5.8.2 Demo 2: Attempt to Execute Instruction from Illegal Region

Demo 2 shows the demonstration result when attempting an execution from an illegal region.

1. Run the MCU. LED1 blinks.
2. Enter **3** to the input box of the RTT Viewer.
3. System attempts to execute instruction from an illegal region (0xE0000000). Then the system will stop in the NS project's `MemManage_handler()`.
4. Confirm the saved fault status in the buffer to determine the fault factor.



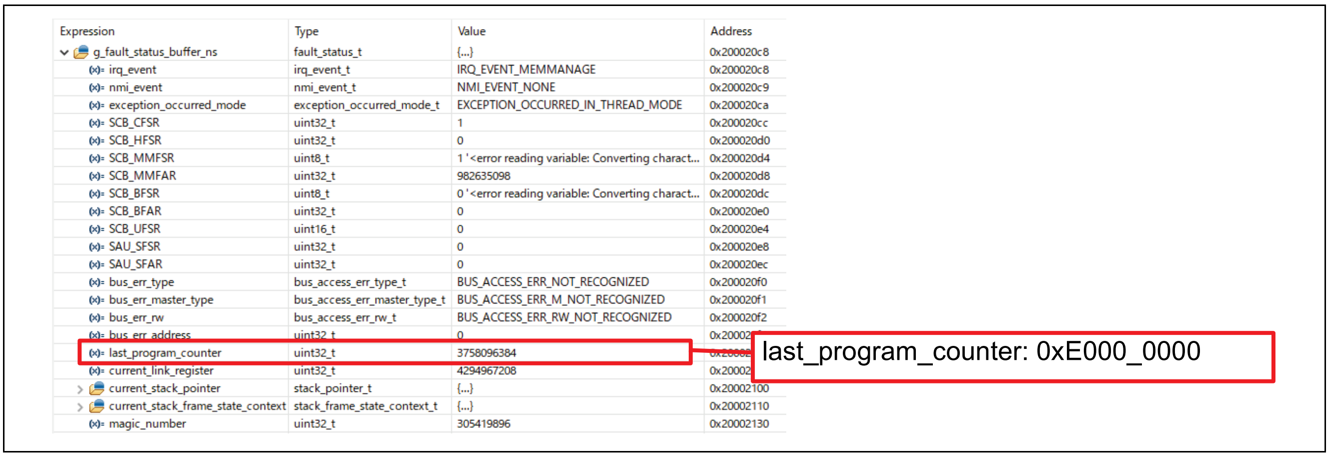
Expression	Type	Value	Address
g_fault_status_buffer_ns	fault_status_t	{...}	0x20002
irq_event	irq_event_t	IRQ_EVENT_MEMMANAGE	0x20002
nmi_event	nmi_event_t	NMI_EVENT_NONE	0x20002
exception_occurred_mode	exception_occurred_mode_t	EXCEPTION_OCCURRED_IN_THREAD_MODE	0x20002
SCB_CFSR	uint32_t	1	0x200020cc
SCB_HFSR	uint32_t	0	0x20002
SCB_MMFSR	uint8_t	1 '<error reading variable: Converting charact...	0x20002
SCB_MMFAR	uint32_t	982635098	0x20002
SCB_BFSR	uint8_t	0 '<error reading variable: Converting charact...	0x20002
SCB_BFAR	uint32_t	0	0x20002
SCB_UFSR	uint16_t	0	0x20002
SAU_SFSR	uint32_t	0	0x20002
SAU_SFAR	uint32_t	0	0x20002
bus_err_type	bus_access_err_type_t	BUS_ACCESS_ERR_NOT_RECOGNIZED	0x20002
bus_err_master_type	bus_access_err_master_type_t	BUS_ACCESS_ERR_M_NOT_RECOGNIZED	0x20002
bus_err_rw	bus_access_err_rw_t	BUS_ACCESS_ERR_RW_NOT_RECOGNIZED	0x200020f2
bus_err_address	uint32_t	0	0x200020f4
last_program_counter	uint32_t	3758096384	0x200020f8
current_link_register	uint32_t	4294967208	0x200020fc
current_stack_pointer	stack_pointer_t	{...}	0x20002100
current_stack_frame_state_context	stack_frame_state_context_t	{...}	0x20002110
magic_number	uint32_t	305419896	0x20002130

irq_event: IRQ_EVENT_MEMMANAGE. Means MemManage fault has occurred.

SCB_MMFSR: 0x1 (IACCVIOL bit is raised). Means the processor attempted an instruction fetch from a location that does not permit execution.

Figure 32. Determine Fault Factor

5. Confirm the stack pointer to determine the location of root cause
   From the following results, we found the last executed program address (0xE000_0000) from the buffer. But its address is out of the flash area. Therefore, we cannot get the information from the stack memory.

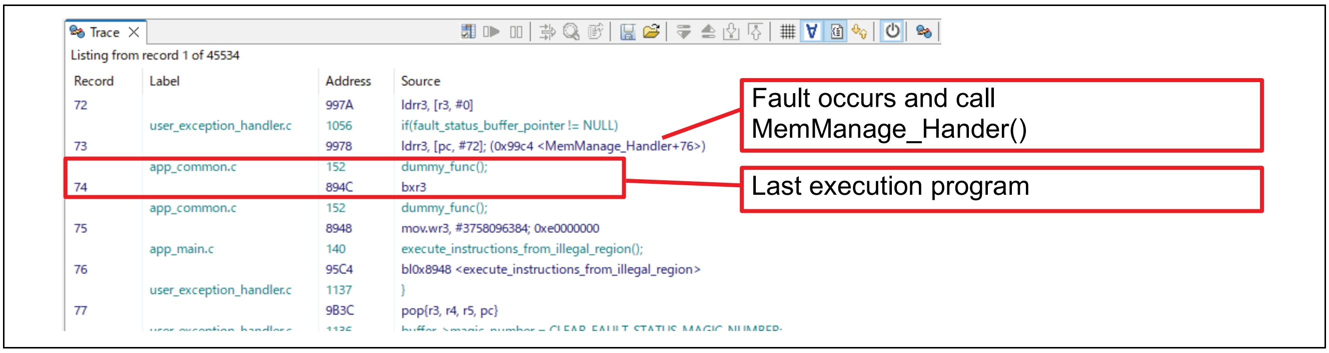


Expression	Type	Value	Address
g_fault_status_buffer_ns	fault_status_t	{...}	0x200020c8
irq_event	irq_event_t	IRQ_EVENT_MEMMANAGE	0x200020c8
nmi_event	nmi_event_t	NMI_EVENT_NONE	0x200020c9
exception_occurred_mode	exception_occurred_mode_t	EXCEPTION_OCCURRED_IN_THREAD_MODE	0x200020ca
SCB_CFSR	uint32_t	1	0x200020cc
SCB_HFSR	uint32_t	0	0x200020d0
SCB_MMFSR	uint8_t	1 '<error reading variable: Converting charact...	0x200020d4
SCB_MMFAR	uint32_t	982635098	0x200020d8
SCB_BFSR	uint8_t	0 '<error reading variable: Converting charact...	0x200020dc
SCB_BFAR	uint32_t	0	0x200020e0
SCB_UFSR	uint16_t	0	0x200020e4
SAU_SFSR	uint32_t	0	0x200020e8
SAU_SFAR	uint32_t	0	0x200020ec
bus_err_type	bus_access_err_type_t	BUS_ACCESS_ERR_NOT_RECOGNIZED	0x200020f0
bus_err_master_type	bus_access_err_master_type_t	BUS_ACCESS_ERR_M_NOT_RECOGNIZED	0x200020f1
bus_err_rw	bus_access_err_rw_t	BUS_ACCESS_ERR_RW_NOT_RECOGNIZED	0x200020f2
bus_err_address	uint32_t	0	0x20002
last_program_counter	uint32_t	3758096384	[illegible]
current_link_register	uint32_t	4294967208	0x20002
current_stack_pointer	stack_pointer_t	{...}	0x20002100
current_stack_frame_state_context	stack_frame_state_context_t	{...}	0x20002110
magic_number	uint32_t	305419896	0x20002130

last_program_counter: 0xE000_0000

Figure 33. Program Counter Outside Flash Area

6. Check the tracing result.
   When the MCU stops, the **Trace** view will show the tracing result in order of the execution history.
   By default, only the executed program address is displayed. Selecting the **Source** button and **Disassembly** button is recommended to easily understand the related program.
   We can find the exception entry point by scrolling the line. In the below example, the fault has occurred after the record number 74. Also, from the displayed results, we find the related source code, the file name (app_common.c), and line number (Line 152).



Listing from record 1 of 45534

Record	Label	Address	Source
72		997A	ldrr3, [r3, #0]
	user_exception_handler.c	1056	if(fault_status_buffer_pointer != NULL)
73		9978	ldrr3, [pc, #72]; (0x99c4 <MemManage_Handler+76>)
	app_common.c	152	dummy_func();
74		894C	bxr3
	app_common.c	152	dummy_func();
75		8948	mov.wr3, #3758096384; 0xe0000000
	app_main.c	140	execute_instructions_from_illegal_region();
76		95C4	bl0x8948 <execute_instructions_from_illegal_region>
	user_exception_handler.c	1137	}
77		9B3C	pop{r3, r4, r5, pc}
	user_exception_handler.c	1136	buffer->magic_number = CLEAR_FAULT_STATUS_MAGIC_NUMBER;

Fault occurs and call MemManage_Hander()

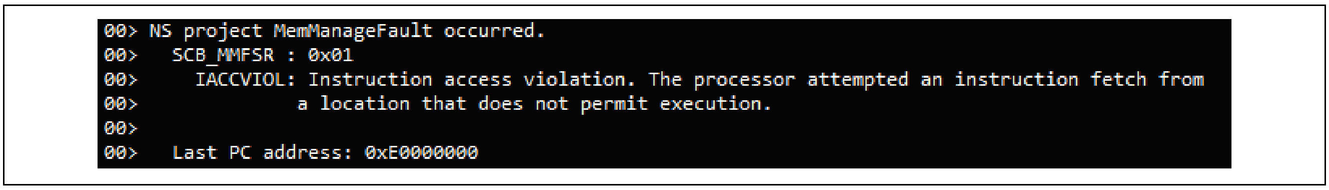
Last execution program

Figure 34. Checking Trace Results

From the above, we can find the root cause of the fault that occurred.

- Factor: a processor attempted an execution from an unpermitted location
- Location: the program address 0x0000_894C (Line.152 in app_common.c file)

Also, after pressing the **Reset** button and **Resume** button on e^2 studio IDE has restarted the system, the details of the saved fault status are displayed on RTT viewer as follows.



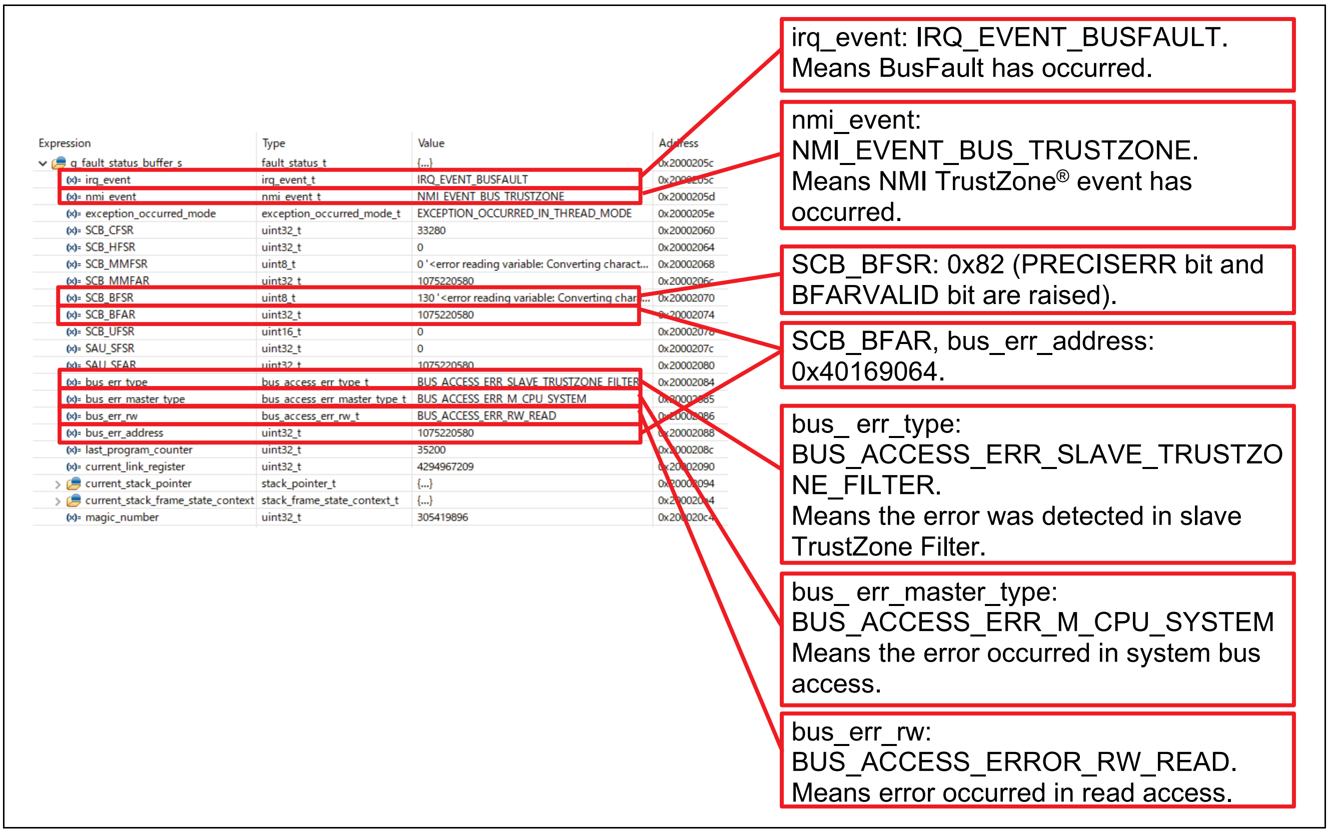
```
00> NS project MemManageFault occurred.
00>   SCB_MMFSR : 0x01
00>     IACCVIOL: Instruction access violation. The processor attempted an instruction fetch from
00>             a location that does not permit execution.
00>
00>   Last PC address: 0xE0000000
```

Figure 35. Saved Fault Status Displayed on RTT Viewer

5.8.3 Demo 3: Attempt Secure Peripheral Access from Non-secure Code

The Demo 3 shows the demonstration result when attempting a secure peripheral access from Non-secure code without NSC call.

1. Run the MCU. LED1 blinks.
2. Enter **6** to the input box of the RTT Viewer.
3. System attempts a Secure peripheral access from Non-secure code. Then the system will stop in the S project's `BusFault_handler()`.
4. Confirm the status of the fault status buffer.



Expression	Type	Value	Address
g_fault_status_buffer_s	fault_status_t	{...}	0x2000205c
irq_event	irq_event_t	IRQ_EVENT_BUSFAULT	0x2000205c
nmi_event	nmi_event_t	NMI_EVENT_BUS_TRUSTZONE	0x2000205d
exception_occurred_mode	exception_occurred_mode_t	EXCEPTION_OCCURRED_IN_THREAD_MODE	0x2000205e
SCB_CFSR	uint32_t	33280	0x20002060
SCB_HFSR	uint32_t	0	0x20002064
SCB_MMFSR	uint8_t	0 '<error reading variable: Converting charact...	0x20002068
SCB_MMFAR	uint32_t	1075220580	0x2000206c
SCB_BFSR	uint8_t	130 '<error reading variable: Converting char...	0x20002070
SCB_BFAR	uint32_t	1075220580	0x20002074
SCB_UFSR	uint16_t	0	0x20002078
SAU_SFSR	uint32_t	0	0x2000207c
SAU_SFAR	uint32_t	1075220580	0x20002080
bus_err_type	bus_access_err_type_t	BUS_ACCESS_ERR_SLAVE_TRUSTZONE_FILTER	0x20002084
bus_err_master_type	bus_access_err_master_type_t	BUS_ACCESS_ERR_M_CPU_SYSTEM	0x20002085
bus_err_rw	bus_access_err_rw_t	BUS_ACCESS_ERR_RW_READ	0x20002086
bus_err_address	uint32_t	1075220580	0x20002088
last_program_counter	uint32_t	35200	0x2000208c
current_link_register	uint32_t	4294967209	0x20002090
current_stack_pointer	stack_pointer_t	{...}	0x20002094
current_stack_frame_state_context	stack_frame_state_context_t	{...}	0x200020a4
magic_number	uint32_t	305419896	0x200020c4

irq_event: IRQ_EVENT_BUSFAULT. Means BusFault has occurred.

nmi_event: NMI_EVENT_BUS_TRUSTZONE. Means NMI TrustZone® event has occurred.

SCB_BFSR: 0x82 (PRECISERR bit and BFARVALID bit are raised).

SCB_BFAR, bus_err_address: 0x40169064.

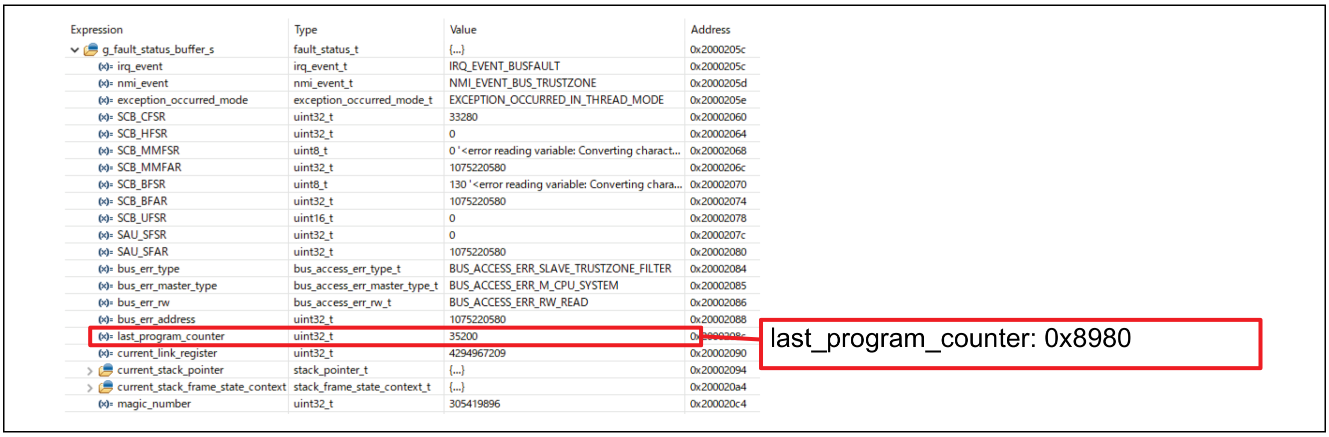
bus_ err_type: BUS_ACCESS_ERR_SLAVE_TRUSTZONE_FILTER. Means the error was detected in slave TrustZone Filter.

bus_ err_master_type: BUS_ACCESS_ERR_M_CPU_SYSTEM Means the error occurred in system bus access.

bus_err_rw: BUS_ACCESS_ERROR_RW_READ. Means error occurred in read access.

Figure 36. Status of the Fault Status Buffer

5. Confirm the stack pointer to determine the location of root cause.
   From the following results, we found the last executed program address (0x0000_8980) from the buffer.

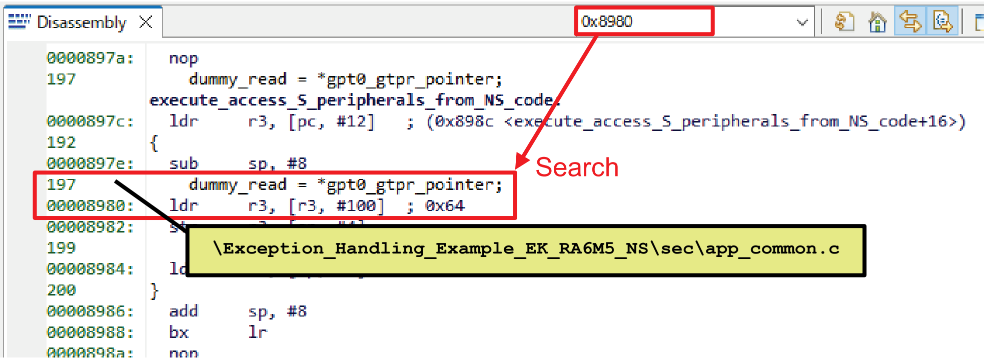


Expression	Type	Value	Address
g_fault_status_buffer_s	fault_status_t	{...}	0x2000205c
irq_event	irq_event_t	IRQ_EVENT_BUSFAULT	0x2000205c
nmi_event	nmi_event_t	NMI_EVENT_BUS_TRUSTZONE	0x2000205d
exception_occurred_mode	exception_occurred_mode_t	EXCEPTION_OCCURRED_IN_THREAD_MODE	0x2000205e
SCB_CFSR	uint32_t	33280	0x20002060
SCB_HFSR	uint32_t	0	0x20002064
SCB_MMFSR	uint8_t	0 '<error reading variable: Converting charact...	0x20002068
SCB_MMFAR	uint32_t	1075220580	0x2000206c
SCB_BFSR	uint8_t	130 '<error reading variable: Converting chara...	0x20002070
SCB_BFAR	uint32_t	1075220580	0x20002074
SCB_UFSR	uint16_t	0	0x20002078
SAU_SFSR	uint32_t	0	0x2000207c
SAU_SFAR	uint32_t	1075220580	0x20002080
bus_err_type	bus_access_err_type_t	BUS_ACCESS_ERR_SLAVE_TRUSTZONE_FILTER	0x20002084
bus_err_master_type	bus_access_err_master_type_t	BUS_ACCESS_ERR_M_CPU_SYSTEM	0x20002085
bus_err_rw	bus_access_err_rw_t	BUS_ACCESS_ERR_RW_READ	0x20002086
bus_err_address	uint32_t	1075220580	0x20002088
last_program_counter	uint32_t	35200	0x2000208c
current_link_register	uint32_t	4294967209	0x20002090
current_stack_pointer	stack_pointer_t	{...}	0x20002094
current_stack_frame_state_context	stack_frame_state_context_t	{...}	0x200020a4
magic_number	uint32_t	305419896	0x200020c4

last_program_counter: 0x8980

Figure 37. Finding the Last PC from Buffer

6. Determine the source code from the last program counter address.
   From the following results, we found the source code (Line 197 in `app_common.c`).



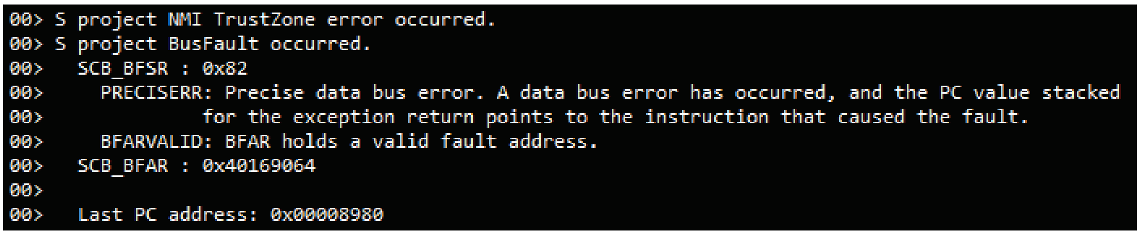
```
0000897a:   nop
197           dummy_read = *gpt0_gtpr_pointer;
          execute_access_S_peripherals_from_NS_code:
0000897c:   ldr     r3, [pc, #12]   ; (0x898c <execute_access_S_peripherals_from_NS_code+16>)
192       {
0000897e:   sub     sp, #8
197           dummy_read = *gpt0_gtpr_pointer;
00008980:   ldr     r3, [r3, #100]  ; 0x64
00008982:   s
199
00008984:   ld
200       }
00008986:   add     sp, #8
00008988:   bx      lr
0000898a:   nop
```

Figure 38. Determine the Source File

From the above, we can find the root cause of the fault that occurred.

- Factor: illegal access from the Non-secure world to a Secure attributed peripheral register (0x40169064)
- Location: the program address 0x0000_8980 (Line 197 in `app_common.c` file)

Also, after pressing the **Reset** button and **Resume** button on the e^2 studio IDE has restarted the system , the details of the saved fault status are displayed on RTT viewer as follows.



```
00> S project NMI TrustZone error occurred.
00> S project BusFault occurred.
00>   SCB_BFSR : 0x82
00>     PRECISERR: Precise data bus error. A data bus error has occurred, and the PC value stacked
00>              for the exception return points to the instruction that caused the fault.
00>     BFARVALID: BFAR holds a valid fault address.
00>   SCB_BFAR : 0x40169064
00>
00>   Last PC address: 0x00008980
```

Figure 39. Saved Fault Status Displayed on RTT Viewer

5.9 Using Example Exception Handler in Your Projects

This application note provides an example exception handler. You can apply it to your projects in the following manner.

1. Copy the following files to your project.
   — `user_exception_handler.c`
   — `user_exception_hanlder.h`
2. Add the following code to your project to enable the example exception handler.

```
#include "user_exception_handler.h"  ← Include directive
fault_status_t   <user-defined buffer name>; ← Definition of user buffer
…
void hal_entry{
f_clear_fault_status_buffer(&<user-defined buffer name>);  ← Initialize user buffer
f_set_fault_status_buffer(&<user-defined buffer name>);  ← Set user buffer pointer to program
f_enable_additional_faults(); ← Enable additional faults
…
}
```

6. References

• Renesas FSP User's Manual	renesas.github.io/fsp
• Renesas RA MCU datasheet	Select the relevant MCU from www.renesas.com/ra
• Example Projects	github.com/renesas/ra-fsp-examples
• Arm® Cortex®-M4 Devices Generic User Guide:	https://developer.arm.com/documentation/dui0553
• Arm® Cortex®-M23 Devices Generic User Guide:	https://developer.arm.com/documentation/dui1095
• Arm® Cortex®-M33 Devices Generic User Guide:	https://developer.arm.com/documentation/100235
• Arm® Cortex®-M85 Devices Generic User Guide:	https://developer.arm.com/documentation/101928

Website and Support

Visit the following URLs to learn about key elements of the RA family, download components and related documentation, and get support.

RA Product Information	renesas.com/ra
RA Product Support Forum	renesas.com/ra/forum
RA Flexible Software Package	renesas.com/FSP
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Mar.23.22	—	First release document
1.10	Jan.31.24	—	Add RA8x1 series support Update to FSPv5.1.0

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.