HOW TO DEBUG A FAULT EXCEPTION ON ARM CORTEX-M MCU

Alvin Liu

2022-08-15

CONFIDENTIAL AND PROPRIETARY

SECURE CONNECTIONS
FOR A SMARTER WORLD

Agenda

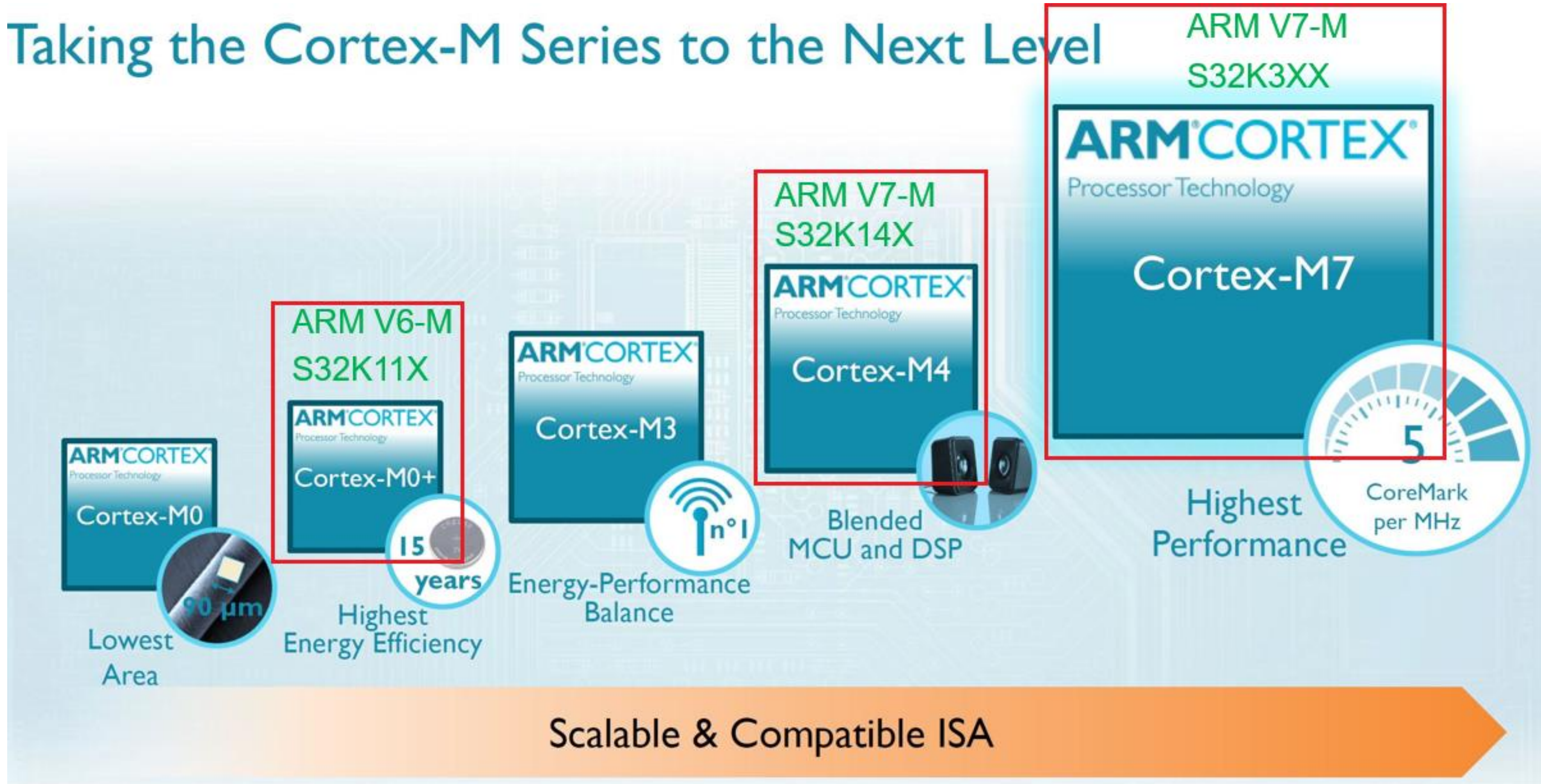
How To Debug A Fault Exception On ARM Cortex-M(V7M) MCU(S32K3X)

- Fault Exception Model Overview
- Registers Used To Control And Status Fault Exceptions
- Practice On Fault Exception Debugging
- Fault Exception Handling

CONFIDENTIAL AND PROPRIETARY

FAULT EXCEPTION MODEL OVERVIEW

Cortex-M Series Productions



CONFIDENTIAL AND PROPRIETARY

Exception Model

Exception Type	ARMv6-M (Cortex-M0/M0+/M1)	ARMv7-M (Cortex-M3/M4/M7)	ARMv8-M Baseline (Cortex-M23)	ARMv8-M Mainline (Cortex-M33)	Vector Table	Vector address offset (initial)
495		Not supported in Cortex-M3/M4/M7	Not supported in Cortex-M23	Device Specific Interrupts	Interrupt#479 vector	0x000007BC
256						
255		Device Specific Interrupts	Device Specific Interrupts		Interrupt#239 vector	0x000003FC
31	Device Specific Interrupts				Interrupt#31 vector	0x000000BC
17					Interrupt#1 vector	0x00000044
16					Interrupt#0 vector	0x00000040
15	SysTick	SysTick	SysTick	SysTick	SysTick vector	0x0000003C
14	PendSV	PendSV	PendSV	PendSV	PendSV vector	0x00000038
13	Not used	Not used	Not used	Not used	Not used	0x00000034
12		Debug Monitor		Debug Monitor	Debug Monitor vector	0x00000030
11	SVC	SVC	SVC	SVC	SVC vector	0x0000002C
10	Not used	Not used	Not used	Not used	Not used	0x00000028
9					Not used	0x00000024
8					Not used	0x00000020
7				SecureFault	SecureFault (ARMv8-M Mainline)	0x0000001C
6		Usage Fault		Usage Fault	Usage Fault vector	0x00000018
5		Bus Fault		Bus Fault	Bus Fault vector	0x00000014
4		MemManage (fault)		MemManage (fault)	MemManage vector	0x00000010
3	HardFault	HardFault	HardFault	HardFault	HardFault vector	0x0000000C
2	NMI	NMI	NMI	NMI	NMI vector	0x00000008
1					Reset vector	0x00000004
0					MSP initial value	0x00000000

Fault Exception Model

	ARMv6-M (Cortex-M0, Cortex-M0+, Cortex-M1) and ARMv8-M Baseline (Cortex-M23)	ARMv7-M (Cortex-M3, Cortex-M4, Cortex-M7) and ARMv8-M Mainline (Cortex-M33)
HardFault	Y	Y
MemManage	-	Y
Usage Fault	-	Y
Bus Fault	-	Y
SecureFault	-	ARMv8-M Mainline only
Fault Status Registers	- (one Debug FSR for debug only)	Y
Fault Address Register	-	Y

CONFIDENTIAL AND PROPRIETARY

Fault Exception Model

- **HardFault:** is the default exception and can be triggered because of an error during exception processing, or because an exception cannot be managed by any other exception mechanism.
- **MemManage Fault:** detects memory access violations to regions that are defined in the Memory Management Unit (MPU)
- **BusFault:** detects memory access errors on instruction fetch, data read/write, interrupt vector fetch, and register stacking (save/restore) on interrupt (entry/exit).
- **UsageFault:** detects execution of undefined instructions, unaligned memory access for load/store multiple. When enabled, divide-by-zero and other unaligned memory accesses are detected.

Exception	Exception Number	Priority	IRQ Number	Activation
HardFault	3	-1	-13	-
MemManage fault	4	Configurable	-12	Synchronous
BusFault	5	Configurable	-11	Synchronous when precise, asynchronous when imprecise.
UsageFault	6	Configurable	-10	Synchronous

CONFIDENTIAL AND PROPRIETARY

Fault Escalation

Faults escalated to HardFault :

- A fault handler causes the same kind of fault as the one it is servicing. This escalation to HardFault occurs because a handler cannot preempt itself (it must have the same priority as the current priority level).
- A fault handler causes a fault with the same or lower priority as the fault it is servicing. This is because the handler for the new fault cannot preempt the currently executing fault handler.
- An exception handler causes a fault for which the priority is the same as or lower than the currently executing exception.
- A fault occurs and the handler for that fault is not enabled.

Only Reset and NMI can preempt the fixed priority HardFault. A HardFault can preempt any exception.

REGISTERS USED TO CONTROL AND STATUS FAULT EXCEPTIONS

Fault Types

Fault type	Handler	Status Register	Bit Name
Bus error on a vector read error	HardFault	HFSR	VECTTBL
Fault that is escalated to a hard fault			FORCED
Fault on breakpoint escalation			DEBUGEVT
Fault on instruction access	MemManage	MMFSR	IACCVIOL
Fault on direct data access			DACCVIOL
Context stacking, because of an MPU access violation			MSTKERR
Context unstacking, because of an MPU access violation			MUNSTKERR
During lazy floating-point state preservation			MLSPERR
During exception stacking	BusFault	BFSR	STKERR
During exception unstacking			UNSTKERR
During instruction prefetching, precise			IBUSERR
During lazy floating-point state preservation			LSPERR
Precise data access error, precise			PRECISERR
Imprecise data access error, imprecise			IMPRECISERR
Undefined instruction	UsageFault	UFSR	UNDEFINSTR
Attempt to enter an invalid instruction set state			INVSTATE
Failed integrity check on exception return			INVPC
Attempt to access a non-existing coprocessor			NOCPC
Illegal unaligned load or store			UNALIGNED
Stack overflow			STKOF
Divide By 0			DIVBYZERO

Fault Exception Relevant Status Registers

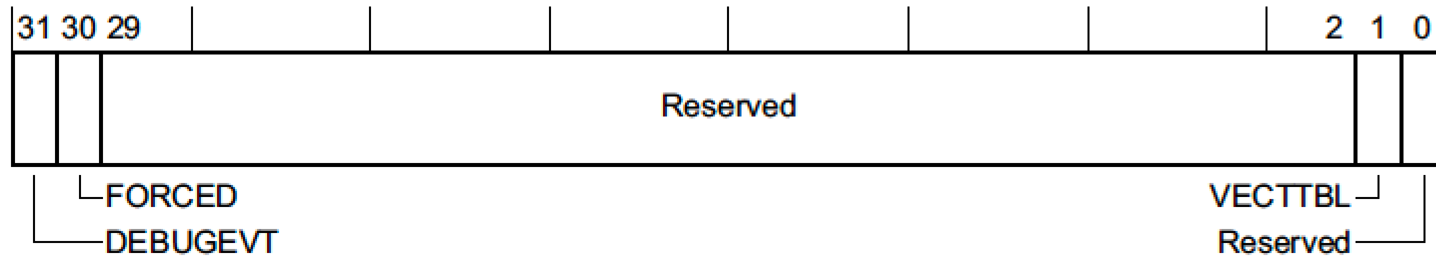
Handler	Status Register	Address Register	Address	Description
HardFault	HFSR		0xE000ED2C	HardFault Status Register
MemManage	MMFSR 	 MMFAR	0xE000ED28 0xE000ED34	MemManage Fault Status Register MemManage Fault Address Register
BusFault	BFSR 	 BFAR	0xE000ED29 0xE000ED38	BusFault Status Register BusFault Address Register
UsageFault	UFSR		0xE000ED2A	UsageFault Status Register
	AFSR		0xE000ED3C	Auxiliary Fault Status Register. Implementation defined content
	ABFSR		0xE000EFA8	Auxiliary BusFault Status Register. Only for Cortex-M7

CONFIDENTIAL AND PROPRIETARY

Fault Exception Relevant Status Registers - HFSR

- HardFault Status Register (HFSR) - 0xE000ED2C

This registers explains the reason a HardFault exception was triggered



DEBUGEVT - Indicates that a debug event occurred while the debug subsystem was not enabled

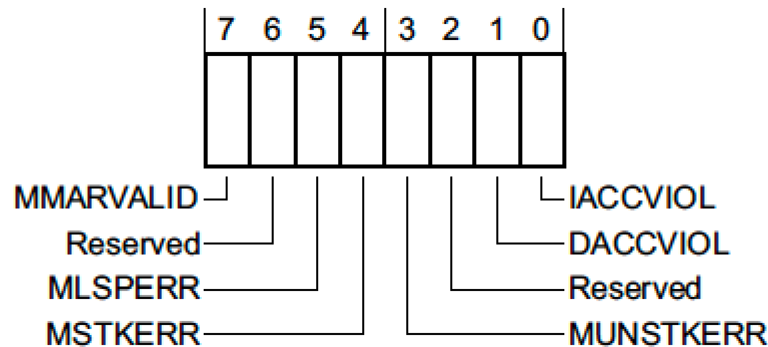
FORCED - This means a configurable fault was escalated to a HardFault, either because the configurable fault handler was not enabled or a fault occurred within the handler.

VECTTBL - Indicates a fault occurred because of an issue reading from an address in the vector table.
This is pretty a typical but could happen if there is a bad address in the vector table and an unexpected interrupt fires.

Fault Relevant Status Registers - MMFSR

- MemManage Status Register (MMFSR) - 0xE000ED28

The MemManage fault status register (MMFSR) indicates a memory access violation detected by the Memory Protection Unit (MPU). Privileged access permitted only. Unprivileged accesses generate a BusFault.



MMARVALID - Indicates that the *MemManage Fault Address Register* (**MMFAR**), a 32 bit register located at 0xE000ED34, holds the address which triggered the MemManage fault.

MLSPERR & MSTKERR - Indicates that a MemManage fault occurred during lazy state preservation or exception entry, respectively.

MUNSTKERR - Indicates that a fault occurred while returning from an exception

DACCVIOL - Indicates that a data access triggered the MemManage fault.

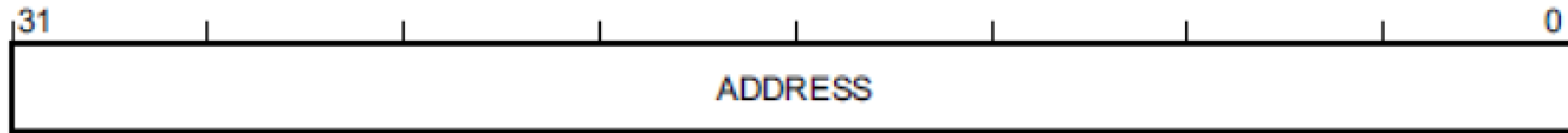
IACCVIOL - Indicates that an attempt to execute an instruction triggered an MPU or Execute Never (XN) fault.

CONFIDENTIAL AND PROPRIETARY

Fault Relevant Status Registers - MMFAR

- MemManage Fault Address Register (**MMFAR**)- 0xE000ED34

The BFAR address is associated with a precise data access BusFault. Privileged access permitted only. Unprivileged accesses generate a BusFault.

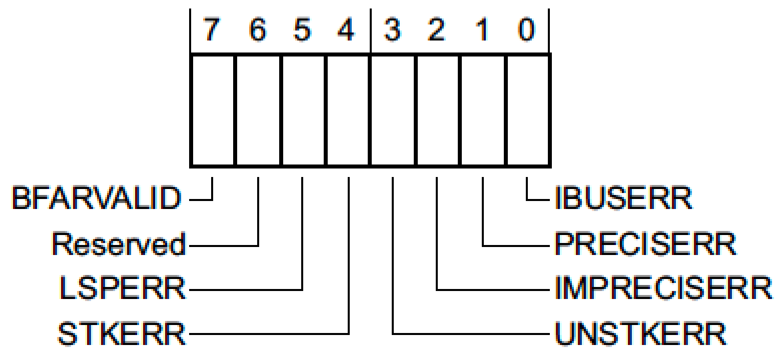


Data address for a MemManage fault. This register is updated with the address of a location that produced a MemManage fault. The MMFSR shows the cause of the fault. This field is valid only when MMFSR.MMARVALID is set. In implementations without unique BFAR and MMFAR registers, the value of this register is UNKNOWN if BFSR.BFARVALID is set.

Fault Relevant Status Registers - BFSR

- BusFault Status Register (BFSR) - 0xE000ED29

The BusFault Status Register shows the status of bus errors resulting from instruction fetches and data accesses and indicates memory access faults detected during a bus operation. Only privileged access is permitted. Unprivileged access will generate a BusFault.



BFARVALID - Indicates that the *Bus Fault Address Register* (**BFAR**), a 32 bit register located at 0xE000ED38, holds the address which triggered the fault.

LSPERR & STKERR - Indicates that a fault occurred during lazy state preservation or during exception entry, respectively. Both are situations where the hardware is automatically saving state on the stack**.** One way this error may occur is if the stack in use overflows off the valid RAM address range while trying to service an exception

UNSTKERR - Indicates that a fault occurred trying to return from an exception. This typically arises if the stack was corrupted while the exception was running or the stack pointer was changed and its contents were not initialized correctly

IMPRECISERR - This flag is *very* important. It tells us whether or not the hardware was able to determine the exact location of the fault

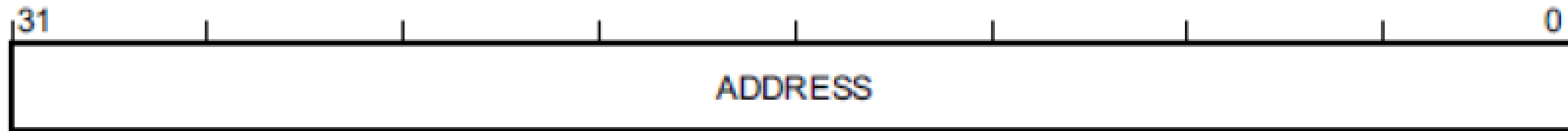
PRECISERR - Indicates that the instruction which was executing prior to exception entry triggered the fault.

 CONFIDENTIAL AND PROPRIETARY

Fault Relevant Status Registers - BFAR

- BusFault Address Register (BFAR) - 0xE000ED38

The BFAR address is associated with a precise data access BusFault. Privileged access permitted only. Unprivileged accesses generate a BusFault.

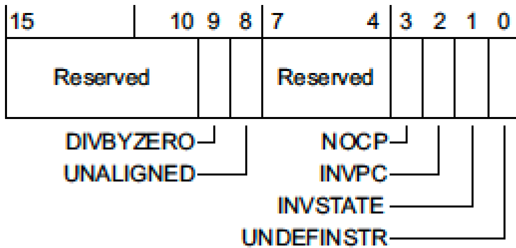


Data address for a precise BusFault. This register is updated with the address of a location that produced a BusFault. The BFSR shows the reason for the fault. This field is valid only when BFSR.BFARVALID is set. In implementations without unique BFAR and MMFAR registers, the value of this register is UNKNOWN if MMFSR.MMARVALID is set.

Fault Relevant Status Registers - UFSR

- UsageFault Status Register (UFSR) - 0xE000ED2A

The UsageFault Status Register UFSR contains the status for some instruction execution faults, and for data access. Privileged access permitted only. Unprivileged accesses generate a BusFault.



DIVBYZERO - Indicates a divide instruction was executed where the denominator was zero. This fault is configurable**.**

UNALIGNED - Indicates an unaligned access operation occurred. Unaligned multiple word accesses.

NOCP - Indicates that a Cortex-M coprocessor instruction was issued but the coprocessor was disabled or not present.

INVPC - Indicates an integrity check failure on EXC_RETURN.

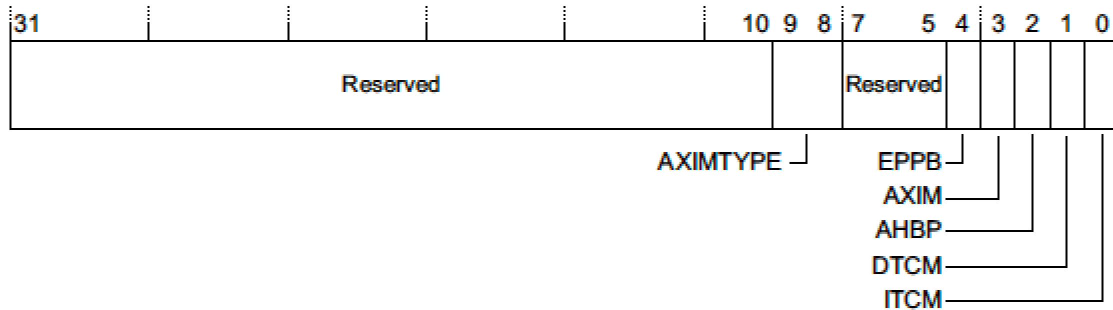
INVSTATE - Indicates the processor has tried to execute an instruction with an invalid *Execution Program Status Register* (**EPSR**) value.

UNDEFINSTR - Indicates an undefined instruction was executed. This can happen on exception exit if the stack got corrupted.

CONFIDENTIAL AND PROPRIETARY

Fault Relevant Status Registers - ABFSR

- Auxiliary Bus Fault Status Register (ABFSR Cortex-M7 only) - 0xE000EFA8



When an IMPRECISE error occurs it will at least give us an indication of what memory bus the fault occurred on

AXIMTYPE: Indicates the type of fault on the AXIM interface. The values are valid only when AXIM=1.
0b00 = OKAY
0b01 = EXOKAY
0b10 = SLVERR
0b11= DECERR
EPPB: Asynchronous fault on EPPB interface
AXIM: Asynchronous fault on AXIM interface
AHBP: Asynchronous fault on AHBP interface
DTCM: Asynchronous fault on DTCM interface
ITCM: Asynchronous fault on ITCM interface

CONFIDENTIAL AND PROPRIETARY

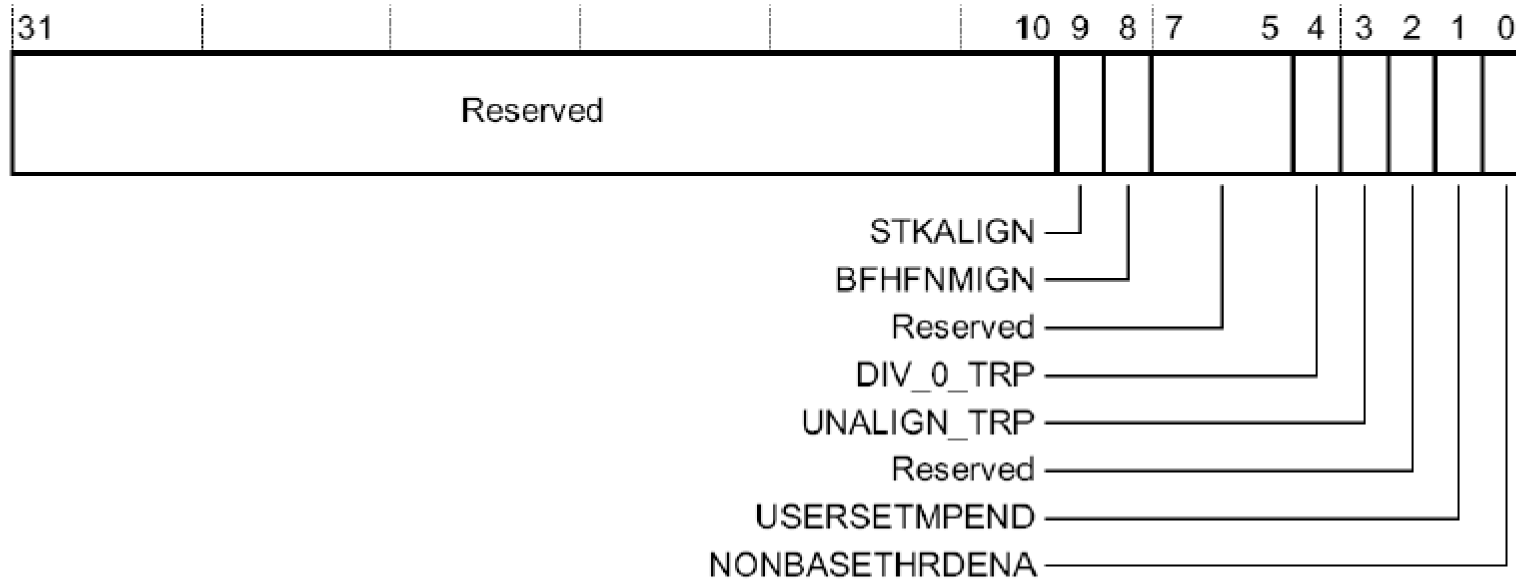
Fault Exception Relevant Control Registers

Address / Access	Register	Reset Value	Description
0xE000ED14 RW privileged	CCR	0x00000000	Configuration and Control Register: contains enable bits for trapping of divide-by-zero and unaligned accesses with the UsageFault.
0xE000ED18 RW privileged	SHP[12]	0x00	System Handler Priority registers: control the priority of exception handlers.
0xE000ED24 RW privileged	SHCSR	0x00000000	HardFault Status Register: contains bits that indicate the reason for HardFault

 CONFIDENTIAL AND PROPRIETARY

Fault Relevant Control Registers - CCR

- *Configuration and Control Register* (**CCR**)**- 0xE000ED14**



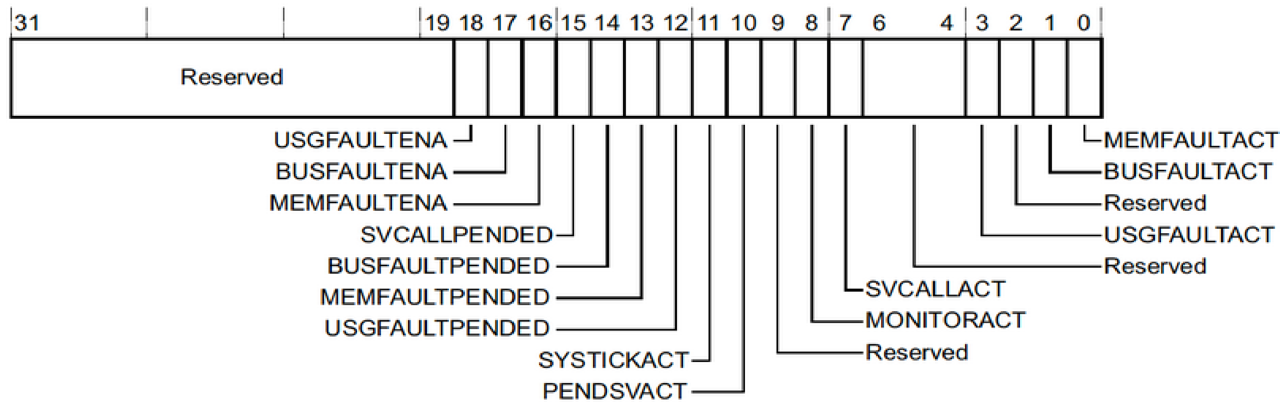
DIV_0_TRP - Controls whether or not divide by zeros will trigger a fault.

UNALIGN_TRP - Controls whether or not unaligned accesses will always generate a fault.

Fault Relevant Control Registers - SHCSR

- System Handler Control and State Register (SHCSR) - 0xE000ED24

This register lets you view the status of or enable various built in exception handlers:



MEMFAULTACT: Memory Management Fault exception active bit, reads as 1 if exception is active.

BUSFAULTACT: BusFault exception active bit, reads as 1 if exception is active.

USGFAULTACT: UsageFault exception active bit, reads as 1 if exception is active.

USGFAULTPENDED: UsageFault exception pending bit, reads as 1 if exception is pending.

MEMFAULTPENDED: Memory Management Fault exception pending bit, reads as 1 if exception is pending.

BUSFAULTPENDED: BusFault exception pending bit, reads as 1 if exception is pending.

MEMFAULTENA: Memory Management Fault exception enable bit, set to 1 to enable; set to 0 to disable.

BUSFAULTENA: BusFault exception enable bit, set to 1 to enable; set to 0 to disable.

USGFAULTENA: UsageFault exception enable bit, set to 1 to enable; set to 0 to disable.

CONFIDENTIAL AND PROPRIETARY

PRACTICE ON FAULT EXCEPTION DEBUGGING

Debug Fault Exception & Fault Type

- When a Fault Exception occurred, we need know which Fault Exception triggered and Fault Type that leading the Fault Exception.
- To debug the Fault Exception and Fault Types, we can check relevant status registers as descripted in previous slides.
- To debug the Fault on S32K3XX with S32DS V3.4, more efficient way is to use Exception Catching Feature

Debug Configurations
Create, manage, and run configurations
Plugin has not been registered. Some functionality may not be available.

SAF_100_S32K344_M7_0_Release_FLASH_PNE
SAF_100_S32K344_M7_0_Release_RAM_PNE
SecureBootApp_Debug_FLASH_PNE
SecureBootApp_Debug_FLASH_PNE
SecureBootApp_Debug_RAM_PNE
SecureBootApp_Debug_RAM_PNE
Siul2_Port_Ip_Example_S32K344_Debug_FLASH_PNE
Siul2_Port_Ip_Example_S32K344_Debug_RAM_PNE
swo_printf_Debug_FLASH_PNE
swo_printf_Debug_RAM_PNE
swo_printf_Release_FLASH_PNE
swo_printf_Release_RAM_PNE
temp324_oneElf_M7_0_Debug_FLASH_PNE
temp324_oneElf_M7_0_Debug_RAM_PNE
temp324_oneElf_M7_0_Release_FLASH_PNE
temp324_oneElf_M7_0_Release_RAM_PNE
temp324_oneElf_M7_1_Debug_FLASH_PNE
tempsense_test_S32K314_flash_pemicro
tempsense_test_S32K324_flash_pemicro
tempsense_test_S32K344_flash_pemicro
test_k344
GDB SEGGER J-Link Debugging
IAR C-SPY Application
Launch Group
Launch Group (Deprecated)
Launch Group for S32 Debugger
Lauterbach TRACE32 Debugger
S32 Debugger

Name: Siul2_Port_Ip_Example_S32K344_Debug_FLASH_PNE
Main | PEmicro Debugger | Startup | Source | Common
Delay after reset and before communicating to target for
GDB Server Settings
Launch Server Locally GDBMI Port Number: 6224
Hostname or IP: localhost Server Port Number: 7224
Server Parameters:
GDB Client Settings
Executable: ${S32DS_GDB_ARM32_EXE}
Other options:
Commands: set mem inaccessible-by-default off
set tcp auto-retry on
set tcp connect-timeout 240
SWO/Power Measurement Settings
Enable Streaming More Info...
Streaming Server Port: 10224
Exception Catching Settings (Valid for CortexM cores only)
Enable HardFault Catch
Enable Exception Entry/Return Catch
Enable BusFault Catch
Enable State Information Error Catch
Enable Checking Error Catch
Enable No-Coprocessor Catch
Enable MemManage Catch
Enable Reset Vector Catch

Debug Fault Exception & Fault Type – MemManage Fault

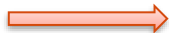
```
int main(void)
{
    uint8 count = 0U;
    /* Initialize all pins using the Port driver */
    Siul2_Port_Ip_Init(NUM_OF_CONFIGURED_PINS0, g_pin_mux_InitConfigArr0);
    illegal_instruction_execution();

    while (1)
    {
        Siul2_Dio_Ip_WritePin(LED_Q172_PORT, LED_Q172_PIN, 1U);
        //Siul2_Dio_Ip_WritePin(LED_Q257_PORT, LED_Q257_PIN, 1U);
        Siul2_Dio_Ip_WritePin(TEST_POINT_PORT, TEST_POINT_PIN, 1U);
        TestDelay(4800000);

        Siul2_Dio_Ip_WritePin(LED_Q172_PORT, LED_Q172_PIN, 0U);
        //Siul2_Dio_Ip_WritePin(LED_Q257_PORT, LED_Q257_PIN, 0U);
        Siul2_Dio_Ip_WritePin(TEST_POINT_PORT, TEST_POINT_PIN, 0U);
        TestDelay(4800000);
    }

    Exit_Example(TRUE);
    return (0U);
}
```

```
95 int illegal_instruction_execution(void) {
96 int (*bad_instruction)(void) = (void *)0xE0000000;
97 return bad_instruction();
98 }
```



```
84 void HardFault_Handler(void)
85 {
86     while(TRUE){};
87 }
```

```
Initializing.
Soft reset failed!
Target has been RESET and is active.
Soft reset failed!
MemManage: The processor attempted an instruction fetch from a location that does not permit execution.
HardFault: A fault has been escalated to a hard fault.
```

Fault Type

Fault Exception

As MemManage Fault did no enabled, the Fault escalated to Hard Fault

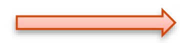
Variables | Breakpoints | Expressions | Peripheral Regi... | Arm System Re... | EmbSys Regist...

Expression	Type	Value
(*((volatile uint32_t*)0xE0001000))	volatile uint32_t	0x40000000 (Hex)
(*((volatile uint32_t*)0xE0001004))	volatile uint32_t	0
*(uint8_t *)0xE000ED28	uint8_t	1 '\001'
Add new expression		

Debug Fault Exception & Fault Type – MemManage Fault

```
123 int main(void)
124 {
125     uint8 count = 0U;
126     /* Initialize all pins using the Port driver */
127     Siul2_Port_Ip_Init(NUM_OF_CONFIGURED_PINS0, g_pin_mux_InitConfigArr0);
128     //illegal_instruction_execution();
129     read_from_bad_address();
130
131     while (1)
132     {
133         Siul2_Dio_Ip_WritePin(LED_Q172_PORT, LED_Q172_PIN, 1U);
134         //Siul2_Dio_Ip_WritePin(LED_Q257_PORT, LED_Q257_PIN, 1U);
135         Siul2_Dio_Ip_WritePin(TEST_POINT_PORT, TEST_POINT_PIN, 1U);
136         TestDelay(4800000);
137
138         Siul2_Dio_Ip_WritePin(LED_Q172_PORT, LED_Q172_PIN, 0U);
139         //Siul2_Dio_Ip_WritePin(LED_Q257_PORT, LED_Q257_PIN, 0U);
140         Siul2_Dio_Ip_WritePin(TEST_POINT_PORT, TEST_POINT_PIN, 0U);
141         TestDelay(4800000);
142     }
143
144     Exit_Example(TRUE);
145     return (0U);
146 }
```

```
100 uint32_t read_from_bad_address(void) {
101     return *(volatile uint32_t *)0xbadcafe;
102 }
103
```



```
84 void HardFault_Handler(void)
85 {
86     while(TRUE){};
87 }
```

Console | Registers | Progress | Problems | Executables | Debug Shell | Watch registers | Debugger Console | Me

Siul2_Port_Ip_Example_S32K344_Debug_FLASH_PNE [GDB PEMicro Interface Debugging] C:\NXP\S32DS.3.4\eclipse\plugins\com.pemicro.

```
CMD>VC
Command is inactive for this .ARP file.
VC is not implemented, falling back to VM

CMD>VM

Verifying.
Verified.

CMD>RE

Initializing.
Soft reset failed!
Target has been RESET and is active.
Soft reset failed!
MemManage: The processor attempted a load or store at a location that does not permit the operation.
Possible MemManage fault location: 0x0BADCAFE.
HardFault: A fault has been escalated to a hard fault.
```

Variables | Breakpoints | Expressions | Peripheral Regist... | Arm System Regi... | EmbSy

Expression	Type	Value
*(uint16_t *)0xE000ED2A	uint16_t	0
*(uint8_t *)0xE000ED29	uint8_t	0 '\0'
*(uint8_t *)0xE000ED28	uint8_t	0x82 (Hex)
Add new expression		

Fault Type

Fault Exception

Faut escalated to Hard Fault

Debug Fault Exception & Fault Type – Usage Fault

```
int main(void)
{
    uint8 count = 0U;
    /* Initialize all pins using the Port driver */
    Siul2_Port_Ip_Init(NUM_OF_CONFIGURED_PINS0, g_pin_mux_InitConfig
    //illegal_instruction_execution();
    //read_from_bad_address();
    //read_from_no_int_sram_address();
    //stack_overflow();
    //NULL_address_write();
    *(uint8_t *)0xE000ED14 = 0x10;
    count = count / 0;
    while (1)
    {
        Siul2_Dio_Ip_WritePin(LED_Q172_PORT, LED_Q172_PIN, 1U);
        //Siul2_Dio_Ip_WritePin(LED_Q257_PORT, LED_Q257_PIN, 1U);
        Siul2_Dio_Ip_WritePin(TEST_POINT_PORT, TEST_POINT_PIN, 1U);
        TestDelay(4800000);

        Siul2_Dio_Ip_WritePin(LED_Q172_PORT, LED_Q172_PIN, 0U);
        //Siul2_Dio_Ip_WritePin(LED_Q257_PORT, LED_Q257_PIN, 0U);
        Siul2_Dio_Ip_WritePin(TEST_POINT_PORT, TEST_POINT_PIN, 0U);
        TestDelay(4800000);
    }

    Exit_Example(TRUE);
    return (0U);
}
```

```
84 void HardFault_Handler(void)
85 {
86     while(TRUE){};
87 }
```

Console | Registers | Progress | Problems | Executables | Debug Shell

Siul2_Port_Ip_Example_S32K344_Debug_FLASH_PNE [GDB PEMicro Interface Debugging] C:\N

```
.
Programmed.
CMD>VC
Command is inactive for this .ARP file.
VC is not implemented, falling back to VM

CMD>VM

Verifying.
Verified.


CMD>RE

Initializing.
Soft reset failed!
Target has been RESET and is active.
Soft reset failed!
UsageFault: A divide by zero error has occurred.
HardFault: A fault has been escalated to a hard fault.
```

Variables | Breakpoints | Expressions | Peripheral Regi... | Arm System Re... | EmbSys Reg

Expression	Type	Value
*(uint16_t *)0xE000ED2A	uint16_t	0x200 (Hex)
*(uint8_t *)0xE000ED29	uint8_t	0 '\0'
*(uint8_t *)0xE000ED28	uint8_t	0x0 (Hex)
*(uint8_t *)0xE000ED14	uint8_t	0x10 (Hex)
Add new expression		

Fault Type

Fault Exception

Fault escalted to Hard fault

 CONFIDENTIAL AND PROPRIETARY

Debug Fault Exception & Fault Type – Bus Fault

```
144 int main(void)
145 {
146     uint8 count = 0U;
147     /* Initialize all pins using the Port driver */
148     Siul2_Port_Ip_Init(NUM_OF_CONFIGURED_PINS0, g_pin_mux_InitConfigArr0);
149     //illegal_instruction_execution();
150     //read_from_bad_address();
151     read_from_no_int_sram_address();
152     //stack_overflow();
153     //NULL_address_write();
154     //*(uint8_t *)0xE000ED14 = 0x10;
155     //count = count / 0;
156     while (1)
157     {
158         Siul2_Dio_Ip_WritePin(LED_Q172_PORT, LED_Q172_PIN, 1U);
159         //Siul2_Dio_Ip_WritePin(LED_Q257_PORT, LED_Q257_PIN, 1U);
160         Siul2_Dio_Ip_WritePin(TEST_POINT_PORT, TEST_POINT_PIN, 1U);
161         TestDelay(4800000);
162
163         Siul2_Dio_Ip_WritePin(LED_Q172_PORT, LED_Q172_PIN, 0U);
164         //Siul2_Dio_Ip_WritePin(LED_Q257_PORT, LED_Q257_PIN, 0U);
165         Siul2_Dio_Ip_WritePin(TEST_POINT_PORT, TEST_POINT_PIN, 0U);
166         TestDelay(4800000);
167     }
168
169     Exit_Example(TRUE);
170     return (0U);
171 }
```

```
103
104 uint32_t read_from_no_int_sram_address(void) {
105     return *(volatile uint32_t *)0x20443FF0;
106 }
107
```

```
84 void HardFault_Handler(void)
85 {
86     while(TRUE){};
87 }
```

```
Interrupt command received. Halting execution.
BusFault: A precise (synchronous) data access error has occurred.
Possible BusFault location: 0x20443FF0.
HardFault: A fault has been escalated to a hard fault.
```

Fault Type

BFAR

Fault escalted to Hard Fault

Fault Exception

Debug Fault Exception & Fault Type – Bus Fault

workspaceS32DS.3.4 - Lin_Master_S32K344_Example_DS/RTD/src/Clock_Ip_Divider.c - S32 Design Studio for S32 Platform

File Edit Source Refactor Navigate Search Project ConfigTools Run PEMicro FreeRTOS Window Help

Debug | Project Explorer

- Lin_Master_S32K344_Example_DS_Debug_FLASH
  - Lin_Master_S32K344_Example_DS.elf
    - Thread #1 (Suspended : Signal : SIGTRAP:Tr
      - HardFault_Handler() at exceptions.c:86 (
      - <signal handler called>() at 0xfffffff9
      - Clock_Ip_InitClock() at Clock_Ip.c:754 0x
      - Mcu_Ipw_InitClock() at Mcu_Ipw.c:427 0x
      - Mcu_InitClock() at Mcu.c:857 0x4088fc
      - MCU_Setup() at main.c:70 0x401c3a
      - main() at main.c:248 0x4015ee
  - C:\NXP\S32DS.3.4\eclipse\plugins\com.pemic
  - arm-none-eabi-gdb
  - <terminated, exit value: 0>Semihosting Consc

Clock_Ip_Di... | Clock_Ip_Di... | main.c | exceptions.c | Clock_Ip.c

```
292            WfiStatus = (IP_MC_ME->PRTN0_CORE2_STAT & MC_ME_PRTN0_CORE2_STAT_WFI_M
293            TimeoutOccurred = Clock_Ip_TimeoutExpired(&StartTime, &ElapsedTime, Ti
294        }
295        while ((CLOCK_IP_WFI_EXECUTED != WfiStatus) && (FALSE == TimeoutOccurred))
296
297        if (FALSE == TimeoutOccurred)
298        {
299            RegValue = IP_CONFIGURATION_GPR->CONFIG_REG_GPR;
300            //if ((RegValue & 0xA0000000) == 0xA0000000)
301            {
302                RegValue &= ~CONFIGURATION_GPR_CONFIG_REG_GPR_FIRC_DIV_SEL_MASK;
303                RegValue |= CONFIGURATION_GPR_CONFIG_REG_GPR_FIRC_DIV_SEL(DividerV
304                IP_CONFIGURATION_GPR->CONFIG_REG_GPR = RegValue;
305            }
306        }
307    }
308 }
309 #endif
310
311 /* Clock stop section code */
312 #define MCU_STOP_SEC_CODE
```

RTD 2.0, If HSE_FW installed, then configure the FIRC divider, will lead bus fault

Variables | Breakpoi... | Expressio... | Periphera... | Arm Syst... | EmbSys

Expression	Type	Value
*(uint16_t *)0xE000ED2A	uint16_t	0x0 (Hex)
*(uint8_t *)0xE000ED29	uint8_t	0x4 (Hex)
*(uint8_t *)0xE000ED28	uint8_t	0x0 (Hex)
*(uint8_t *)0xE000ED14	uint8_t	0x0 (Hex)
Add new expression		

Console | Registers | Progress | Problems | Executables | Debug Shell | Watch registers | Debugger Console | Memory | Real Time Expressions | SWO Trace Capture

Lin_Master_S32K344_Example_DS_Debug_FLASH_PNE [GDB PEMicro Interface Debugging] C:\NXP\S32DS.3.4\eclipse\plugins\com.pemicro.debug.gdbjtag.pne_5.1.7.202112141853\win32\pegdbserver_console

```
VC is not implemented, falling back to VM

CMD>VM

Verifying.
Verified.


CMD>RE

Initializing.
Soft reset failed!
Target has been RESET and is active.
Soft reset failed!
BusFault: An imprecise (asynchronous) data access error has occurred.
HardFault: A fault has been escalated to a hard fault.
```

Fault Type

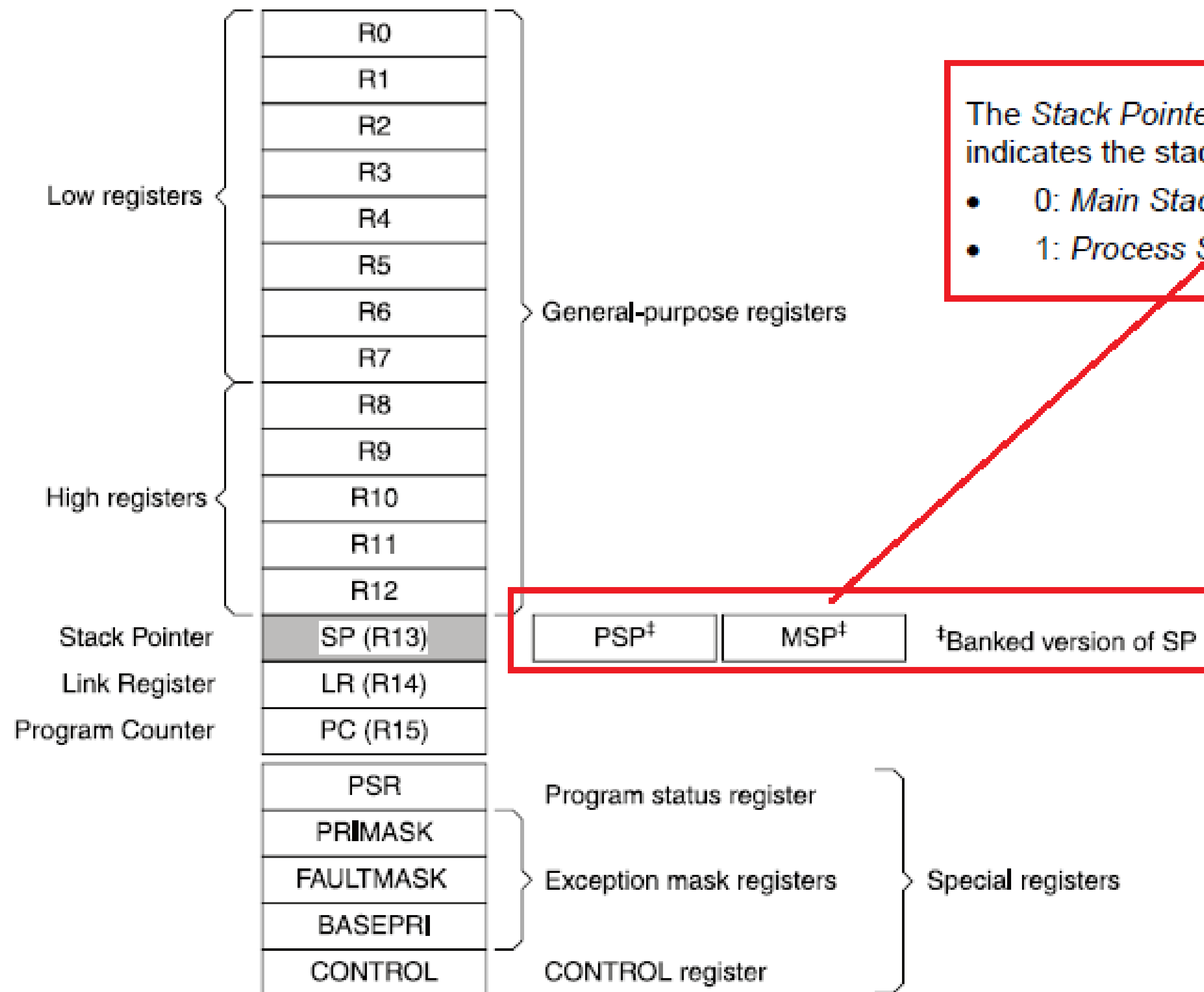
Fault Exception

Dashboard

- Project Creation
  - S32DS Application Project
  - S32DS Library Project
- Build/Debug
  - Build (All)
  - Clean (All)
  - Debug

CONFIDENTIAL AND PROPRIETARY

Debug Fault Address

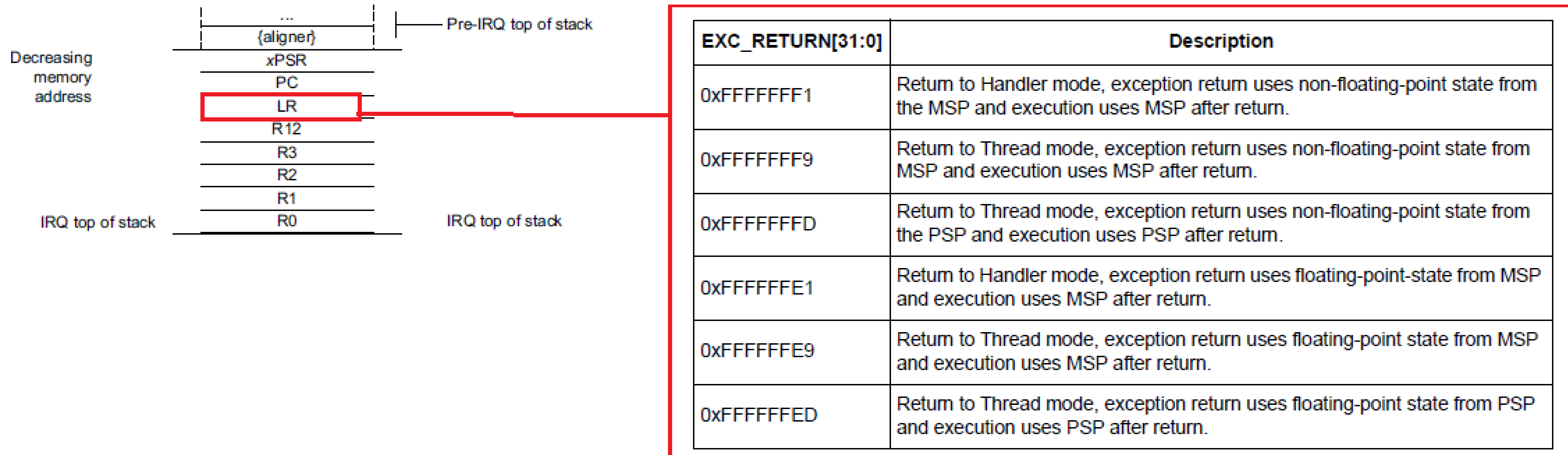


The *Stack Pointer* (SP) is register R13. In Thread mode, bit[1] of the CONTROL register indicates the stack pointer to use:

- 0: *Main Stack Pointer* (MSP). This is the reset value.
- 1: *Process Stack Pointer* (PSP).

Debug Fault Address

- At exception entry, the processor saves R0-R3, R12, LR, PC and PSR on the stack, and LR is updated with EXC_RETURN, Bit 2 of EXC_RETURN indicate the MSP or PSP used. If Bit 2 is 1, PSP used, if Bit 2 is 0, MSP used.



EXC_RETURN[31:0]	Description
0xFFFFFFF1	Return to Handler mode, exception return uses non-floating-point state from the MSP and execution uses MSP after return.
0xFFFFFFF9	Return to Thread mode, exception return uses non-floating-point state from MSP and execution uses MSP after return.
0xFFFFFFFD	Return to Thread mode, exception return uses non-floating-point state from the PSP and execution uses PSP after return.
0xFFFFFFE1	Return to Handler mode, exception return uses floating-point-state from MSP and execution uses MSP after return.
0xFFFFFFE9	Return to Thread mode, exception return uses floating-point state from MSP and execution uses MSP after return.
0xFFFFFFED	Return to Thread mode, exception return uses floating-point state from PSP and execution uses PSP after return.

- So, we can locate the Fault Address by stack Backtrace.

CONFIDENTIAL AND PROPRIETARY

Debug Fault Address – Stack Backtrace Manaully

```
79
80 void NMI_Handler(void)
81 {
82     while(TRUE){};
```

Name	Value
r8	0
r9	0
r10	0
r11	0
r12	0
sp	0x2042efc0
lr	0xfffffff9
pc	0x401a9a <HardFault_Handler>
xpsr	1627389955
d0	0
d1	0

Disassembly 0x0040196b

```
00401958:   lsls     r4, r3, #7
0040195a:   movs     r0, #64 ; 0x40
illegal_instruction_execution:
0040195c:   push     {lr}
0040195e:   sub      sp, #12
00401960:   mov.w    r3, #3758096384 ; 0xe0000000
00401964:   str      r3, [sp, #4]
00401966:   ldr      r3, [sp, #4]
00401968:   blx      r3
0040196a:   mov      r3, r0
0040196c:   mov      r0, r3
0040196e:   add      sp, #12
00401970:   ldr.w    pc, [sp], #4
read_from_bad_address:
00401974:   ldr      r3, [pc, #4]     ; (0x40197c <rea
read_from_bad_address:
00401975:   ldr      r3, [pc, #4]     ; (0x40197c <rea
```

Memory — Monitors: 0x00400000, 0x2042efc0, 0xE0001000, 0xE000EDFC, 0x20400000

0x2042efc0 : 0x2042EFC0 <Hex Integer>

Address	0 - 3 (R0 / R12)	4 - 7 (R1 / LR)	8 - B (R2 / PC)	C - F (R3 / PSR)
2042EFC0	00000000	00280000	00000003	E0000000
2042EFD0	00000000	0040196B	E0000000	60000000
2042EFE0	00408FB4	E0000000	00408FB4	00401A53
2042EFF0	00000001	000000F0	000008B4	00401573
2042F000	5AA55AA5	00000001	5AA55AA5	00000001
2042F010	5AA55AA5	00000001	5AA55AA5	00000001
2042F020	5AA55AA5	00000001	5AA55AA5	00000001

Exception return Address

Fault Address

CONFIDENTIAL AND PROPRIETARY

Debug Fault Address – Stack Backtrace Automation

- .gdbinit script

When GDB launching, it will look for the .gdbinit script, if found, GDB will conduct the CMD from list.

Stand CMD can be found from:

C:\NXP\S32DS.3.4\S32DS\tools\gdb-arm\arm32-eabi\arm-none-eabi\share\docs\pdf\GDB.pdf

```
help
List of classes of commands:

aliases -- Aliases of other commands.
breakpoints -- Making program stop at certain points.
data -- Examining data.
files -- Specifying and examining files.
internals -- Maintenance commands.
obscure -- Obscure features.
running -- Running the program.
stack -- Examining the stack.
status -- Status inquiries.
support -- Support facilities.
tracepoints -- Tracing of program execution without stopping the program.
user-defined -- User-defined commands.

Type "help" followed by a class name for a list of commands in that class.
```

Debug Fault Address – Stack Backtrace Automation

- Define Stack Backstrace CMD in .gdbinit script

Apart from stand GDB CMD, we also can define ourself CMD in .gdbint, it can be recognized by GDB, then later we can use this CMD just stand CMD anywhere.

Here we defined the CMD “armex” to do Stack Backtrace.

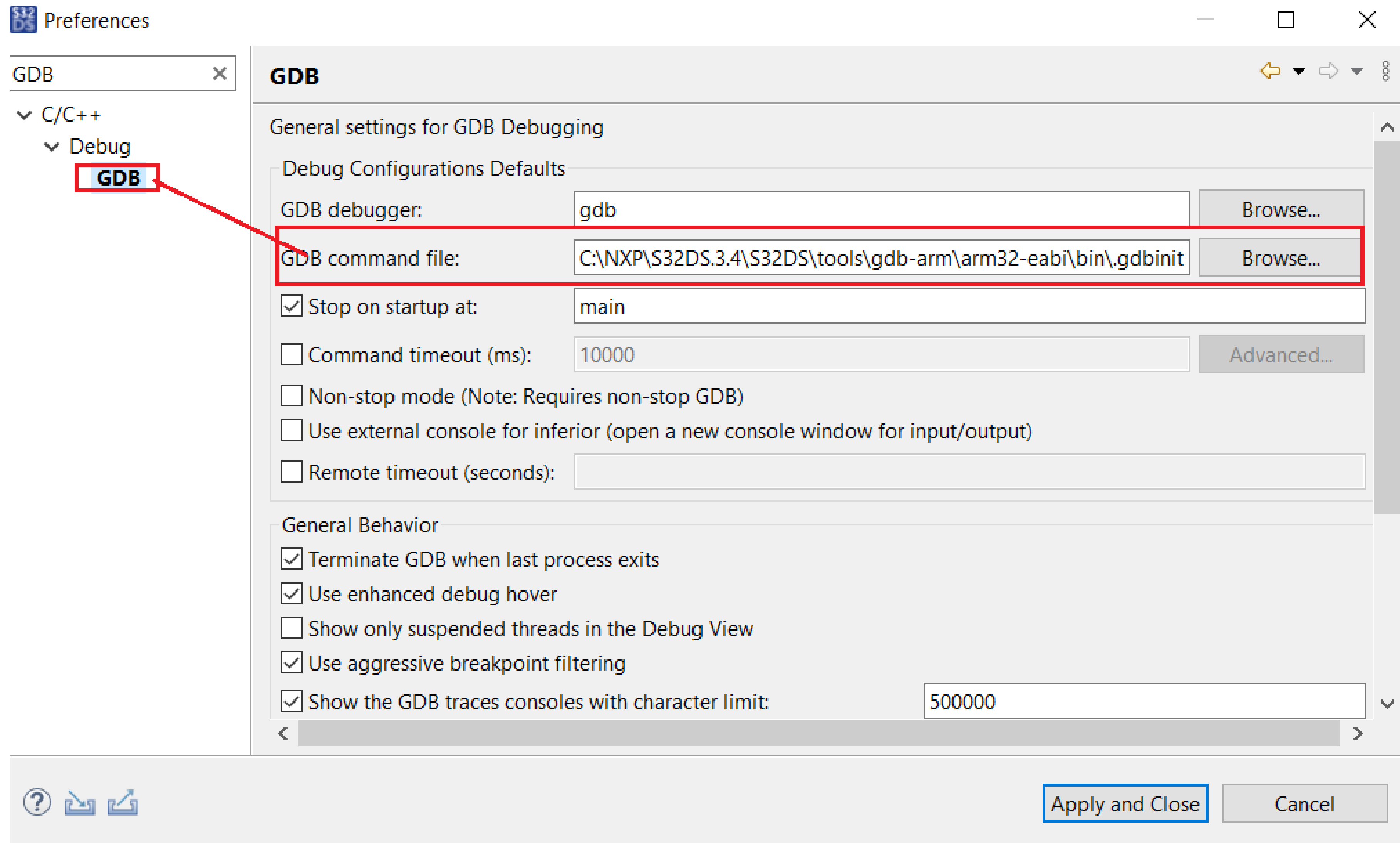
```
set language c
define armex
  printf "EXEC_RETURN (LR):\n",
  info registers $lr
    if (((unsigned int)$lr & 0x04) == 0x4)
      printf "Uses PSP 0x%x return.\n", $psp
      set $armex_base = (unsigned int)$psp
    else
      printf "Uses MSP 0x%x return.\n", $msp
      set $armex_base = (unsigned int)$msp
    end

    printf "xPSR            0x%x\n", *($armex_base+28)
    printf "ReturnAddress   0x%x\n", *($armex_base+24)
    printf "LR (R14)        0x%x\n", *($armex_base+20)
    printf "R12             0x%x\n", *($armex_base+16)
    printf "R3              0x%x\n", *($armex_base+12)
    printf "R2              0x%x\n", *($armex_base+8)
    printf "R1              0x%x\n", *($armex_base+4)
    printf "R0              0x%x\n", *($armex_base)
    printf "Return instruction:\n"
    x/i *($armex_base+24)
    printf "LR instruction:\n"
    x/i *($armex_base+20)
end

document armex
ARMv7 Exception entry behavior.
xPSR, ReturnAddress, LR (R14), R12, R3, R2, R1, and R0
end
```

Debug Fault Address – Stack Backtrace Automation

- Configure .gdbinit PATH in S32DS V3.4



Debug Fault Address – Stack Backtrace Automation

- Run the Stack Backtrace CMD to locate the Fault Address

ect ConfigTools Run PEMicro FreeRTOS Window Help

main.c | bootloader_m... | exceptions.c | Ota.c | main.c | startup_cm7.s | »93

```c
79
80 void NMI_Handler(void)
81 {
82     while(TRUE){};
83 }
84 void HardFault_Handler(void)
85 {
86     while(TRUE){};
87 }
88 void MemManage_Handler(void)
89 {
90     while(TRUE){};
91 }
92 void BusFault_Handler(void)
93 {
94     while(TRUE){};
95 }
96 void UsageFault_Handler(void)
97 {
98     while(TRUE){};
```

Varial | Disassembly | 0x0040196b

Expressi

```
00401958:   lsls    r4, r3, #7
0040195a:   movs    r0, #64 ; 0x40
          illegal_instruction_execution:
0040195c:   push    {lr}
0040195e:   sub     sp, #12
00401960:   mov.w   r3, #3758096384 ; 0xe0000000
00401964:   str     r3, [sp, #4]
00401966:   ldr     r3, [sp, #4]
00401968:   blx     r3
0040196a:   mov     r3, r0
0040196c:   mov     r0, r3
0040196e:   add     sp, #12
00401970:   ldr.w   pc, [sp], #4
          read_from_bad_address:
00401974:   ldr     r3, [pc, #4]    ; (0x40197
          read_from_bad_address:
00401975:   ldr     r3, [pc, #4]    ; (0x40197
```

Console | Registers | Progress | Problems | Executables | Debug Shell | Watch registers | Debugger Console | Memory | Real Time Expressions | SWO Trace Capture

Siul2_Port_Ip_Example_S32K344_Debug_FLASH_PNE [GDB PEMicro Interface Debugging] C:\NXP\S32DS.3.4\S32DS\tools\gdb-arm\arm32-eabi\bin\arm-none-eabi-gdb (9.2)

```
armex
EXEC_RETURN (LR):
lr             0xfffffff9          0xfffffff9
Uses MSP 0x2042efc0 return.
xPSR            0x60000000
ReturnAddress   0xe0000000
LR (R14)        0x40196b
R12             0x0
R3              0xe0000000
R2              0x3
R1              0x280000
R0              0x0
Return instruction:
   0xe0000000:  movs    r1, r0
LR instruction:
   0x40196b <illegal_instruction_execution+14>: mov     r3, r0
```

FAULT EXCEPTION HANDLING

Fault Exception Handling - Common

- For a final application, a fault handler may be implemented that performs

System Reset: by setting bit 2 (SYSRESETREQ) in AIRCR (Application Interrupt and Reset Control Register). This will reset most parts of the system apart from the debug logic. If you do not want to reset the whole system, just set the bit 0 (VECTRESET) in AIRCR which causes only a processor reset.

Recovery: in some cases, it might be possible to resolve the problem that caused the fault exception. For example, in case of a coprocessor instruction, the handler may emulate the instruction in software.

Task termination: for systems running a real-time operating system (RTOS), the task that created the fault may be terminated and restarted if needed.

- Suggest to enable MemFault, BusFault and UsageFault, to handle them separately

 CONFIDENTIAL AND PROPRIETARY

Fault Exception Handling - Baremetal

- Handling for MemFault

```
54 /* Access violation occurs will trigger MM Fault and will enter this function */
55 void userMMFault_Handler(void){
56     statMMFault = S32_SCB->CFSR & 0x000000FF;
57     /* if the bus fault address is valid, save the error address */
58     if(S32_SCB->CFSR & S32_SCB_CFSR_MMFSR_MMARVALID_MASK){
59         addrMMFault = S32_SCB->MMFAR;
60     }
61     printf("MM Fault stats = 0x%02X, fault address = 0x%08X.\r\n", (unsigned int)statMMFault, (unsigned int)addrMMFault);
62     Power_Ip_PerformReset(&Power_Ip_HwIPsConfigPB);
63     return;
64 }
```

Fault Exception Handling - Baremetal

- Handling for MemFault

```
106⊖void userBusFault_Handler(void){
107     if(IP_EIM->EICHEN > 0){
108         IP_EIM->EICHEN = 0;
109         IP_EIM->EIMCR = 0UL;
110     }
111 
112     statBusFault = S32_SCB->CFSR & 0x0000FF00;
113     /* if the bus fault address is valid, save the error address */
114     if(S32_SCB->CFSR & S32_SCB_CFSR_BFSR_BFARVALID_MASK){
115         addrBusFault = S32_SCB->BFAR;
116     }
117 
118     /* if FCCU Alarm interrupt is active, leave bus fault handler and let FCCU Alarm ISR to handle the fault */
119     if(IP_FCCU->IRQ_STAT & FCCU_IRQ_STAT_ALRM_STAT_MASK){
120         return;
121     }
122 
123⊖    /* if it's Flash ECC error, need call the API to read fault and clear fault.
124      * The fault is also reported to FCCU. So after bus fault handler, the FCCU Alarm ISR will handle the fault flag
125     if(IP_FLASH->MCRS & 0xFFFF0000){
126         printf("Flash failure address = 0x%08X.\r\n", (unsigned int)addrBusFault);
127     }
128 
129     /* if it's RAM ECC error, leave bus fault handler, and let FCCU Alarm ISR to handle the fault */
130     if(IP_ERM->SR0){
131         eMcem_GetMemErrInfo( EMCEM_ERM_SRAM0, &memErrInfo );
132         return;
133     } else if(IP_ERM->SR1 | IP_ERM->SR2){
134         eMcem_GetMemErrInfo( EMCEM_ERM_SRAM1, &memErrInfo );
135         return;
136     }
137 
138     printf("Bus fault occurred without FCCU Alarm interrupt.\r\n");
139     printf("Bus fault address is 0x%08X.\r\n", (unsigned int)addrBusFault);
140     printf("Trigger functional reset.\r\n");
141     Power_Ip_PerformReset(&Power_Ip_HwIPsConfigPB);
142     return;
```

Fault Exception Handling - Baremetal

- Handling for UsageFault

```
109 void UsageFault_Handler(void)
110 {
111     while(TRUE){};
112 }
113
```


NXP

Fault Exception Handling - FreeRTOS

- Do not implement Exception Handling in OS level, leave the handling to user. Just Example Code.

```
/* The prototype shows it is a naked function - in effect this is just an
assembly function. */
static void HardFault_Handler( void ) __attribute__( ( naked ) );

/* The fault handler implementation calls a function called
prvGetRegistersFromStack(). */
static void HardFault_Handler(void)
{
    __asm volatile
    (
        " tst lr, #4                                        \n"
        " ite eq                                            \n"
        " mrseq r0, msp                                     \n"
        " mrsne r0, psp                                     \n"
        " ldr r1, [r0, #24]                                 \n"
        " ldr r2, handler2_address_const                    \n"
        " bx r2                                             \n"
        " handler2_address_const: .word prvGetRegistersFromStack    \n"
    );
}
```

```
void prvGetRegistersFromStack( uint32_t *pulFaultStackAddress )
{
/* These are volatile to try and prevent the compiler/linker optimising them
away as the variables never actually get used.  If the debugger won't show the
values of the variables, make them global my moving their declaration outside
of this function. */
volatile uint32_t r0;
volatile uint32_t r1;
volatile uint32_t r2;
volatile uint32_t r3;
volatile uint32_t r12;
volatile uint32_t lr; /* Link register. */
volatile uint32_t pc; /* Program counter. */
volatile uint32_t psr;/* Program status register. */

    r0 = pulFaultStackAddress[ 0 ];
    r1 = pulFaultStackAddress[ 1 ];
    r2 = pulFaultStackAddress[ 2 ];
    r3 = pulFaultStackAddress[ 3 ];

    r12 = pulFaultStackAddress[ 4 ];
    lr = pulFaultStackAddress[ 5 ];
    pc = pulFaultStackAddress[ 6 ];
    psr = pulFaultStackAddress[ 7 ];

    /* When the following line is hit, the variables contain the register values. */
    for( ;; );
}
```


CONFIDENTIAL AND PROPRIETARY

Fault Exception Handling – AutoSAR OS

```
/*
 * OS_ExceptionBusFaultHandler
 *
 * This is a handler for the bus fault exception.
 * The BusFault fault handles memory-related faults, other than those handled by the
 * MemManage fault, for both instruction and data memory transactions.
 * Typically these faults arise from errors detected on the system buses.
 * The architecture permits an implementation to report synchronous or asynchronous
 * BusFaults according to the circumstances that trigger the exceptions.
 * Software can disable this fault. If it does, a BusFault escalates to HardFault.
 * BusFault has a configurable priority.
*/
    OS_ASM_FUNC(OS_ExceptionBusFaultHandler)
OS_ASM_LABEL(OS_ExceptionBusFaultHandler)
    cpsid   f                                 /* Lock all interrupts */
    OS_DISABLE_MPU_GET_STATE    r0, r2, r1
    mov     r0, #OS_CORTEXM_EX_BUS_FAULT
    ldr     r12, CFSR_Addr                    /* configurable fault status register */
    ldr     r2, [r12]
    ldr     r3, BFAR_Addr                     /* bus fault address register */
    ldr     r3, [r3]
    ldr     r12, OS_CFSR_DerivedFault
    tst     r2, r12
    bne     OS_NoContext
    b       OS_ExceptionHandler               /* branch to the generic handler */
    OS_ASM_END_FUNC(OS_ExceptionBusFaultHandler)
```

```
 */
os_uint32_t OS_Exception(os_uint32_t id, os_uint32_t pc, os_uint32_t status, os_
{
    os_uint8_t inKernel;
    OS_PH_PARAMETERACCESS_DECL

    inKernel = OS_GetKernelData()->inKernel;
    OS_GetKernelData()->inKernel = 1;

    OS_PH_SAVE_PARAMETER_N(0,pc);
    OS_PH_SAVE_PARAMETER_N(1,status);
    OS_PH_SAVE_PARAMETER_N(2,addr);

    switch (id)
    {
    case OS_CORTEXM_EX_NMI:
        /* can't propagate the return value of OS_ERROR -> ignore it */
        (void) OS_ERROR(OS_ERROR_NonMaskableInterrupt, OS_PH_GET_PARAMETER_VAR());
        break;

    case OS_CORTEXM_EX_HARD_FAULT:
        /* can't propagate the return value of OS_ERROR -> ignore it */
        (void) OS_ERROR(OS_ERROR_HardFault, OS_PH_GET_PARAMETER_VAR());
        break;

    case OS_CORTEXM_EX_MEM_MANAGE:
        /* can't propagate the return value of OS_ERROR -> ignore it */
        (void) OS_ERROR(OS_ERROR_MemoryManagement, OS_PH_GET_PARAMETER_VAR());
        break;

    case OS_CORTEXM_EX_BUS_FAULT:
        /* can't propagate the return value of OS_ERROR -> ignore it */
        (void) OS_ERROR(OS_ERROR_BusFault, OS_PH_GET_PARAMETER_VAR());
        break;

    case OS_CORTEXM_EX_USAGE_FAULT:
        /* can't propagate the return value of OS_ERROR -> ignore it */
        (void) OS_ERROR(OS_ERROR_UsageFault, OS_PH_GET_PARAMETER_VAR());
        break;

    case OS_CORTEXM_EX_RESERVED_1:
        /* can't propagate the return value of OS_ERROR -> ignore it */
        (void) OS_ERROR(OS_ERROR_ReservedException_1, OS_PH_GET_PARAMETER_VAR())
        break;

    case OS_CORTEXM_EX_RESERVED_2:
        /* can't propagate the return value of OS_ERROR -> ignore it */
        (void) OS_ERROR(OS_ERROR_ReservedException_2, OS_PH_GET_PARAMETER_VAR())
        break;

    case OS_CORTEXM_EX_RESERVED_3:
        /* can't propagate the return value of OS_ERROR -> ignore it */
        (void) OS_ERROR(OS_ERROR_ReservedException_3, OS_PH_GET_PARAMETER_VAR())
        break;
```

SECURE CONNECTIONS
FOR A SMARTER WORLD