CSE 3103: Microprocessor and Microcontroller

Professor Upama Kabir

Computer Science and Engineering, University of Dhaka,

Lecture: Fault Exception

April 03, 2024

Table of Contents

Reasons of Fault

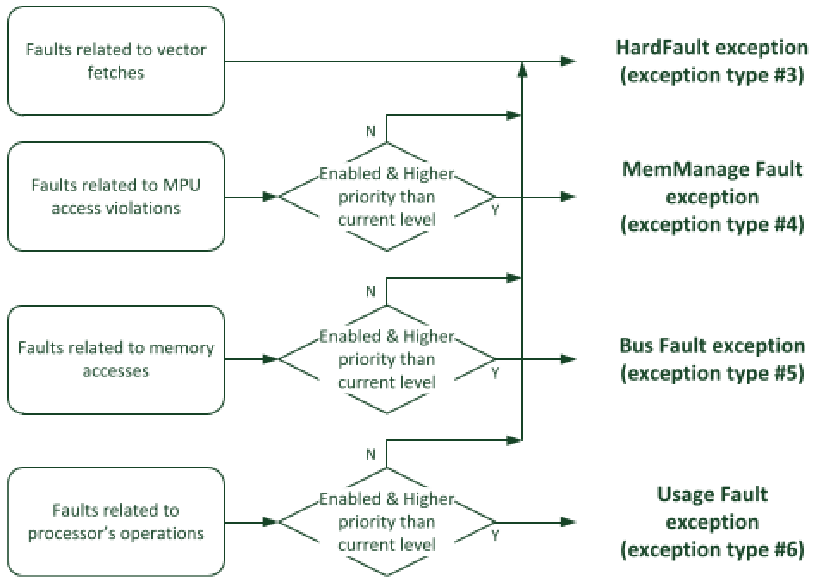
- Bug in Software
- External factors
  - Unstable power supply
  - Electrical noise (e.g., noise from power lines)
  - Electromagnetic interference (EMI)
  - Electrostatic discharge
  - Extreme operation environment (e.g., temperature, mechanical vibrations)
  - Usage issues (e.g., end users did not read the manual) or invalid external data input

Fault handling

In order to allow problems to be detected as early as possible, the Cortex-M processors have a fault exception mechanism included. If a fault is detected, a fault exception is triggered and one of the fault exception handlers is executed.

- By default, all the faults trigger the HardFault exception (exception type number 3).
- Cortex-M4 processors have three additional configurable fault exception handlers:
  - MemManage (Memory Management) Fault (exception type 4)
  - Bus Fault (exception type 5)
  - Usage Fault (exception type 6)
- These exceptions are triggered if they are enabled, and if their priority is higher than the current exception priority level.
- These exceptions are called configurable fault exceptions, and have programmable exception priority levels.

Fault handling



Faults related to vector fetches
Faults related to MPU access violations
Faults related to memory accesses
Faults related to processor's operations
Enabled & Higher priority than current level
N
Y
HardFault exception (exception type #3)
MemManage Fault exception (exception type #4)
Bus Fault exception (exception type #5)
Usage Fault exception (exception type #6)

FIGURE 12.1

Fault exceptions available in ARMv7-M architecture

Figure 1

Memory management (MemManage) faults

- MemManage faults can be caused by violation of access rules defined by the MPU configurations.
    - Unprivileged tasks trying to access a memory region that is privileged access only
    - Access to a memory location that is not defined by any defined MPU regions (except the Private Peripheral Bus (PPB), which is always accessible by privi- leged code)
    - Writing to a memory location that is defined as read-only by the MPU
- The accesses could be data accesses during program execution, program fetches, or stack operations during execution sequences.
- For a MemManage fault triggered by stack operation during exception sequence:
    - If the MemManage fault occurred during stack pushing in the exception entrance sequence, it is called a stacking error.
    - If the MemManage fault occurrs during stack popping in the exception exit sequence, it is called an unstacking error.
- The MemManage fault can also be triggered when trying to execute program code in eXecute Never (XN) regions such as the PERIPHERAL region, DEVICE region, or SYSTEM region

Bus Faults

- The bus faults can be triggered by error responses received from the processor bus interface during a memory access
- The bus fault can also occur during stacking and unstacking of the exception handling sequence
  - tIf the bus error occurred during stack pushing in the exception entrance sequence, it is called a stacking error.
  - If the bus error occurred during stack popping in the exception exit sequence, it is called an unstacking error.
  - Writing to a memory location that is defined as read-only by the MPU
- If a bus error is returned at vector fetch, the HardFault exception would be activated even when Bus Fault exception is enabled.
- A memory system can return error responses if
  - The processor attempts to access an invalid memory location. In this case, the transfer is sent to a module in the bus system called default slave. The default slave returns an error response and triggers the bus fault exception in the processor.
  - The device is not ready to accept a transfer (e.g., trying to access DRAM without initializing the DRAM controller might trigger the bus error. This behavior is device-specific.)
  - The bus slave receiving the transfer request returns an error response. For example, it might happen if the transfer type/size is not supported by the bus slave, or if the peripherals determined that the operation carried out is not allowed.
  - Unprivileged access to the Private Peripheral Bus (PPB) that violates the default memory access permission

Usage Faults

The Usage Fault exception can be caused by a wide range of factors:

- Execution of an undefined instruction (including trying to execute floating point instructions when the floating point unit is disabled).
- Execution of Co-processor instructions: Cortex-M4 processors do not support Co-processor access instructions, but it is possible to use the usage fault mechanism to emulate co-processor instruction support.
- Trying to switch to ARM state: Cortex-M4 processors only support Thumb ISA.
- Invalid EXC_RETURN code during exception-return sequence . For example, trying to return to Thread level with exceptions still active (apart from the current serving exception).
- Execution of SVC when the priority level of the SVC is the same or lower than current level.
- Also possible, by setting up the Configuration Control Register (CCR) : Divide by zero

Hard Fault

- The HardFault exception can be triggered by escalation of configurable fault exceptions.
- Bus error received during a vector fetch
- Execution of breakpoint instruction (BKPT) with a debugger attached
- For example, when reaching a “printf” operation, the processor executes a BKPT instruction and halt, and the debugger can detect the halt and enabling fault handlers check the register status and the immediate value in the BKPT instruction. Then the debugger can display the message or character form the message in the printf statement. If the debugger is not attached, such operation results in HardFault and executes the HardFault exception handler.

Fault status registers and fault address registers

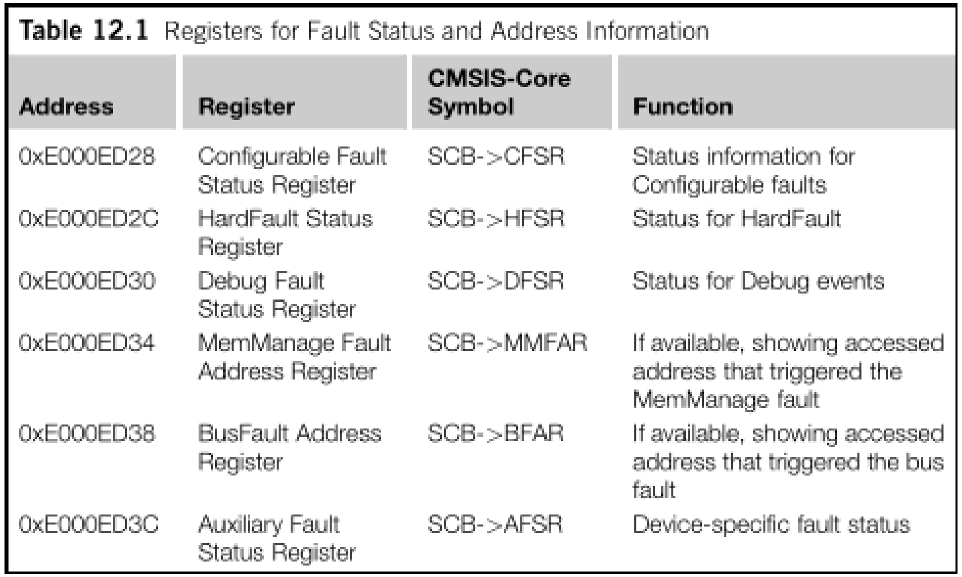


Table 12.1 Registers for Fault Status and Address Information

Address	Register	CMSIS-Core Symbol	Function
0xE000ED28	Configurable Fault Status Register	SCB->CFSR	Status information for Configurable faults
0xE000ED2C	HardFault Status Register	SCB->HFSR	Status for HardFault
0xE000ED30	Debug Fault Status Register	SCB->DFSR	Status for Debug events
0xE000ED34	MemManage Fault Address Register	SCB->MMFAR	If available, showing accessed address that triggered the MemManage fault
0xE000ED38	BusFault Address Register	SCB->BFAR	If available, showing accessed address that triggered the bus fault
0xE000ED3C	Auxiliary Fault Status Register	SCB->AFSR	Device-specific fault status

Figure 2

Fault status registers and fault address registers

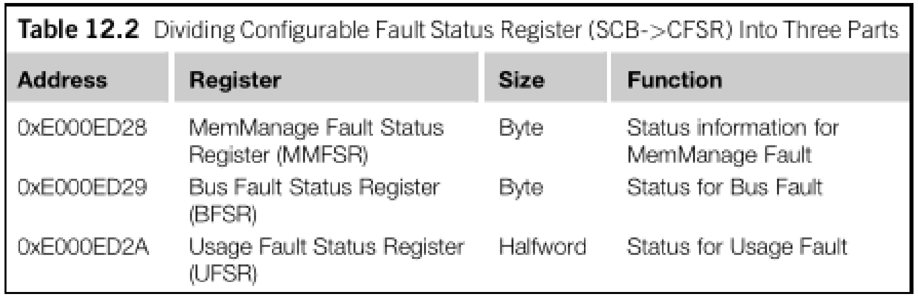


Table 12.2 Dividing Configurable Fault Status Register (SCB->CFSR) Into Three Parts

Address	Register	Size	Function
0xE000ED28	MemManage Fault Status Register (MMFSR)	Byte	Status information for MemManage Fault
0xE000ED29	Bus Fault Status Register (BFSR)	Byte	Status for Bus Fault
0xE000ED2A	Usage Fault Status Register (UFSR)	Halfword	Status for Usage Fault

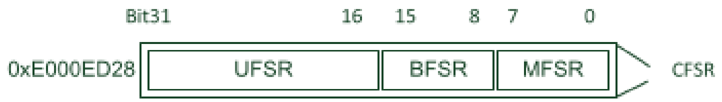


FIGURE 12.2

Configurable Fault Status Register partitioning

Figure 3

Tail Chaining

When an exception takes place but the processor is handling another exception of the same or higher priority: the exception will enter the pending state. (Figure 8.12). In this way, the timing gap between the two exception handlers is considerably reduced. For a memory system with no-wait state, the tail-chain latency is only six clock cycles.

- The new exception will enter pending state
- When the processor finishes executing the current exception handler, it can then proceed to process the pending exception/interrupt request.
- Instead of restoring the registers back from the stack (unstacking) and then pushing them on to the stack again (stacking), the processor skips the unstacking and stacking steps and enters the exception handler of the pended exception as soon as possible
- For a memory system with no-wait state, the tail-chain latency is only six clock cycles.

Tail Chaining

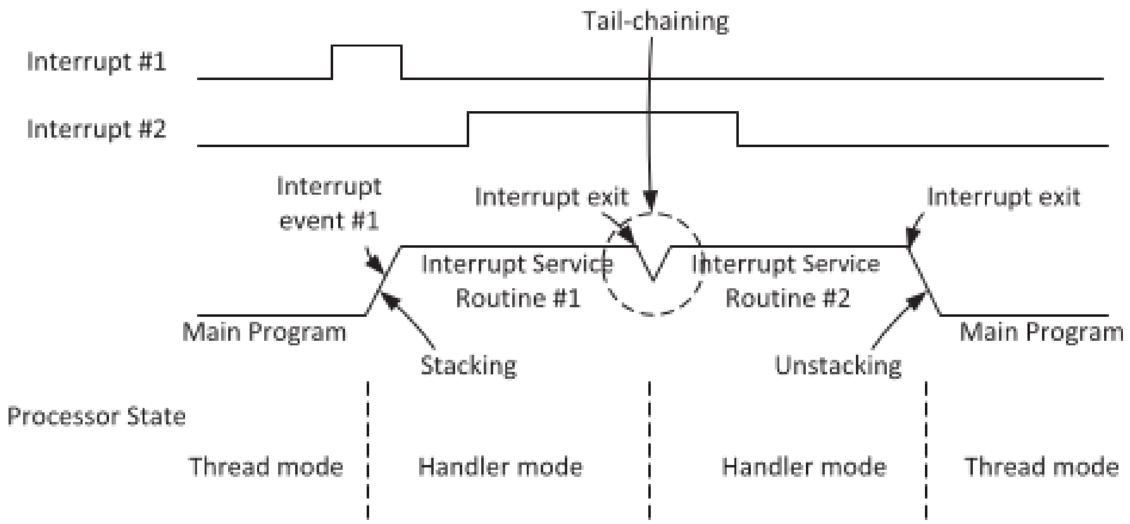
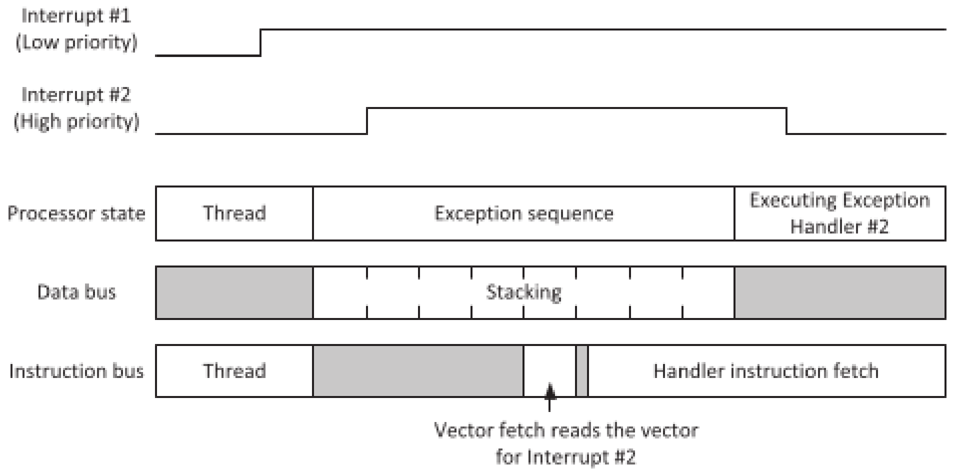


Figure 4: Tail Chaining

Late Arrival

- When an exception takes place, the processor accepts the exception request and starts the stacking operation. If during this stacking operation another exception of higher priority takes place, the higher priority late arrival exception will be serviced first.
- For example, if Exception #1 (lower priority) takes place a few cycles before Exception #2 (higher priority), the processor will behave as shown in following figure, such that Handler #2 is executed as soon as the stacking completes.


Interrupt #1
(Low priority)
Interrupt #2
(High priority)
Processor state
Thread
Exception sequence
Executing Exception Handler #2
Data bus
Stacking
Instruction bus
Thread
Handler instruction fetch
Vector fetch reads the vector for Interrupt #2