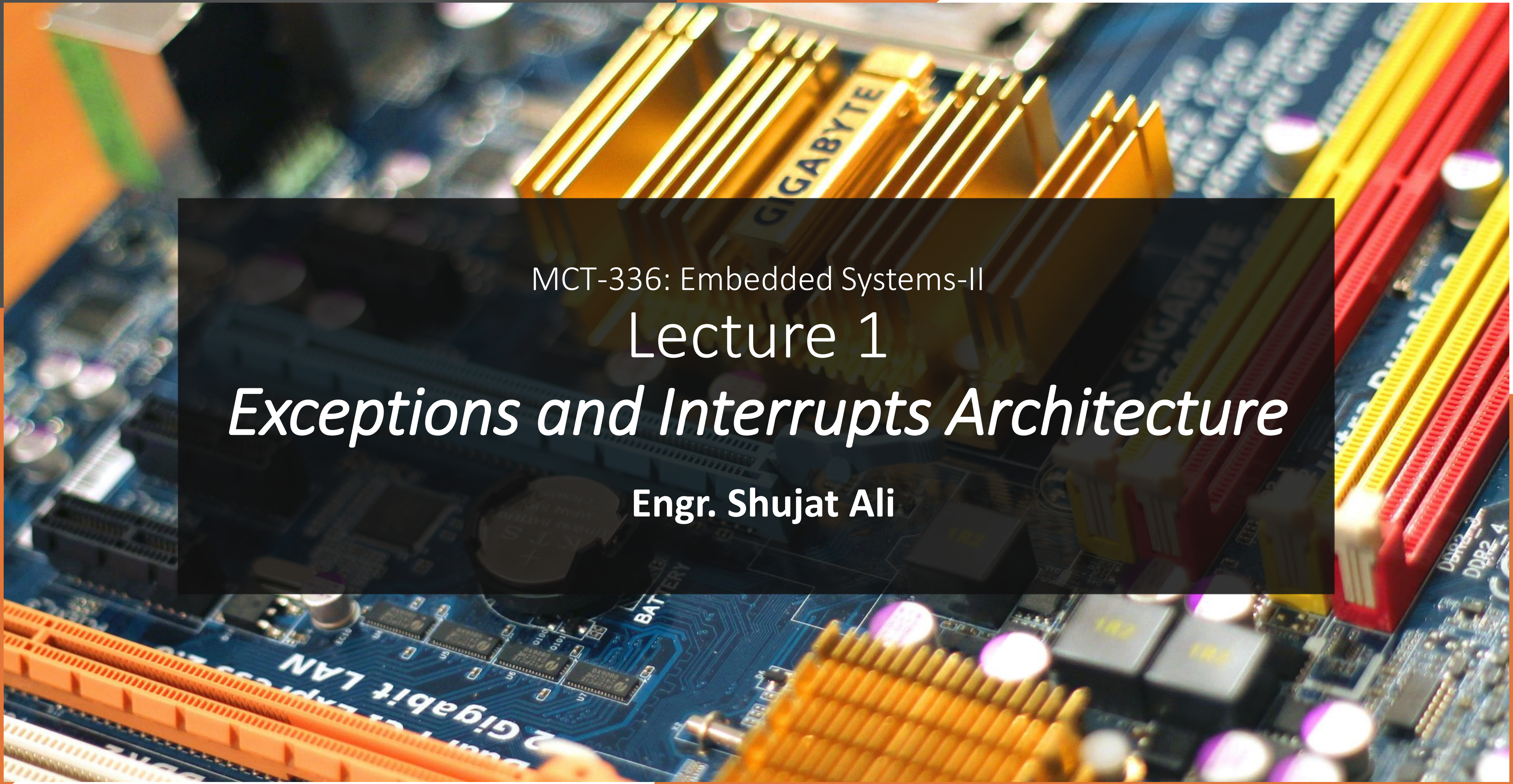
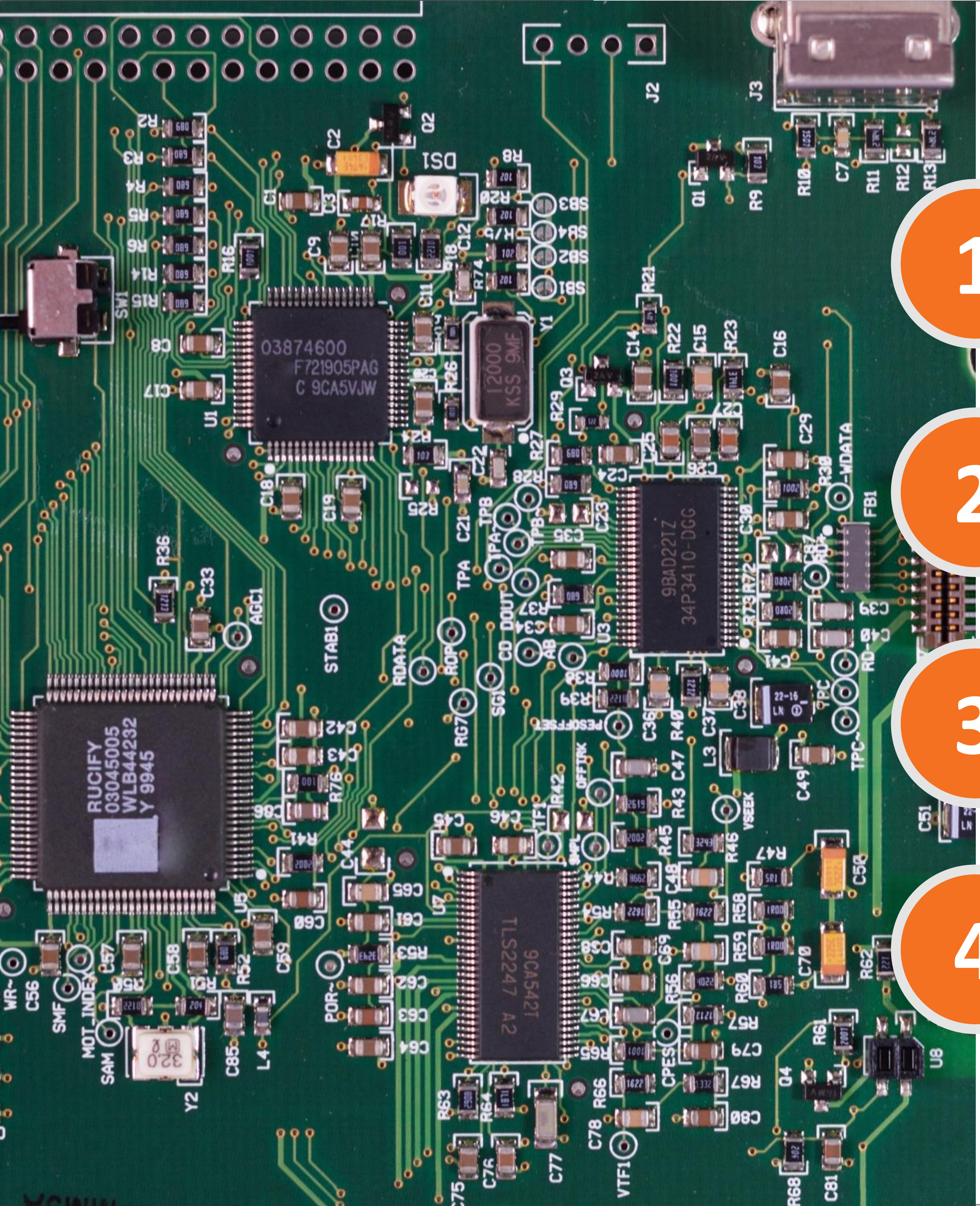


MCT-336: Embedded Systems-II

# Lecture 1
# *Exceptions and Interrupts Architecture*

**Engr. Shujat Ali**



1 WHAT IS AN INTERRUPT?

2 CORTEX-M EXCEPTIONS AND INTERRUPTS

3 INTERRUPT CONFIGURATION

4 EXCEPTION/INTERRUPT HANDLING

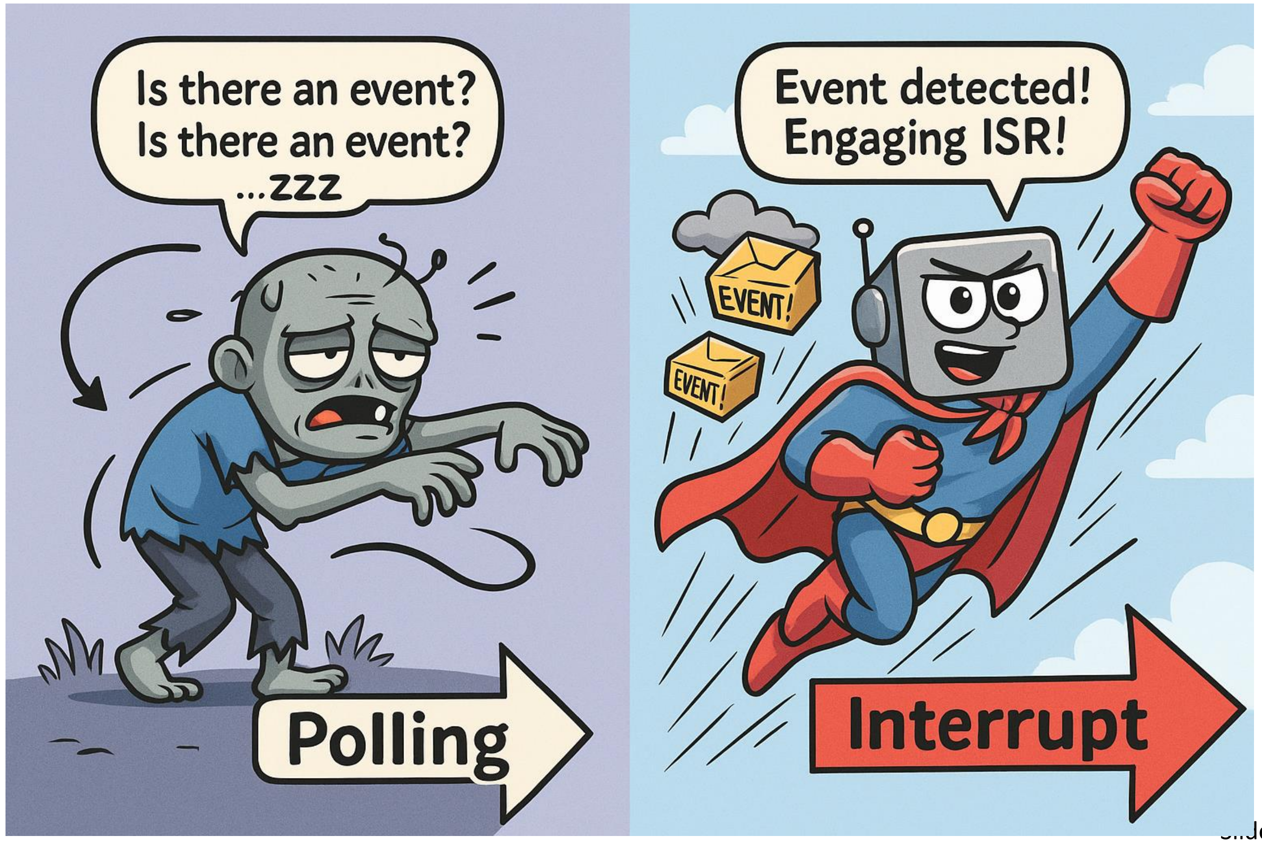
## WHAT IS AN INTERRUPT?

Polling vs Interrupt, ISR, Interrupt Routine Handling

# Polling vs. Interrupt

## WHAT IS AN INTERRUPT?

- **Example**: A program might continuously check if data is available from a sensor.
- It wastes processing time, as it is always checking, even if no event occurs.
- **Polling** is basically a protocol in which the CPU services the I/O devices by continuously checking for events



- An **interrupt** is a mechanism that allows a processor to temporarily halt its current execution to attend to an event that requires immediate attention.
- Unlike polling, interrupts allow the processor to respond to events only when they occur, making them more efficient in real-time systems.
- **Example**: A sensor triggers an interrupt to notify the processor when new data is available, rather than the processor constantly checking.
- **What happens when an interrupt occurs?**

## Interrupt Service Routine (ISR)

### WHAT IS AN INTERRUPT?

- When an event or exception (or interrupt) occurs, the processor responds by executing a function call known as the **Interrupt Service Routine** (ISR) to handle the corresponding peripheral or hardware.
- Most microcontrollers today integrate interrupt capability, ranging from simple interrupts to multi-level prioritized interrupts.
- Interrupts can be generated by both hardware (e.g., external inputs or peripherals) and software events.

## Interrupt Routine Handling

### WHAT IS AN INTERRUPT?

Key steps involved during interrupt routine handling:

1. One of the interrupt or exception sources generates a request.
2. In response to the interrupt, the processor suspends the currently executing task.
3. The processor executes an interrupt service routine to generate the response and service the source of the interrupt.
4. Finally, the processor resumes the execution of the previously suspended task from the same state.

# CORTEX-M EXCEPTIONS AND INTERRUPTS

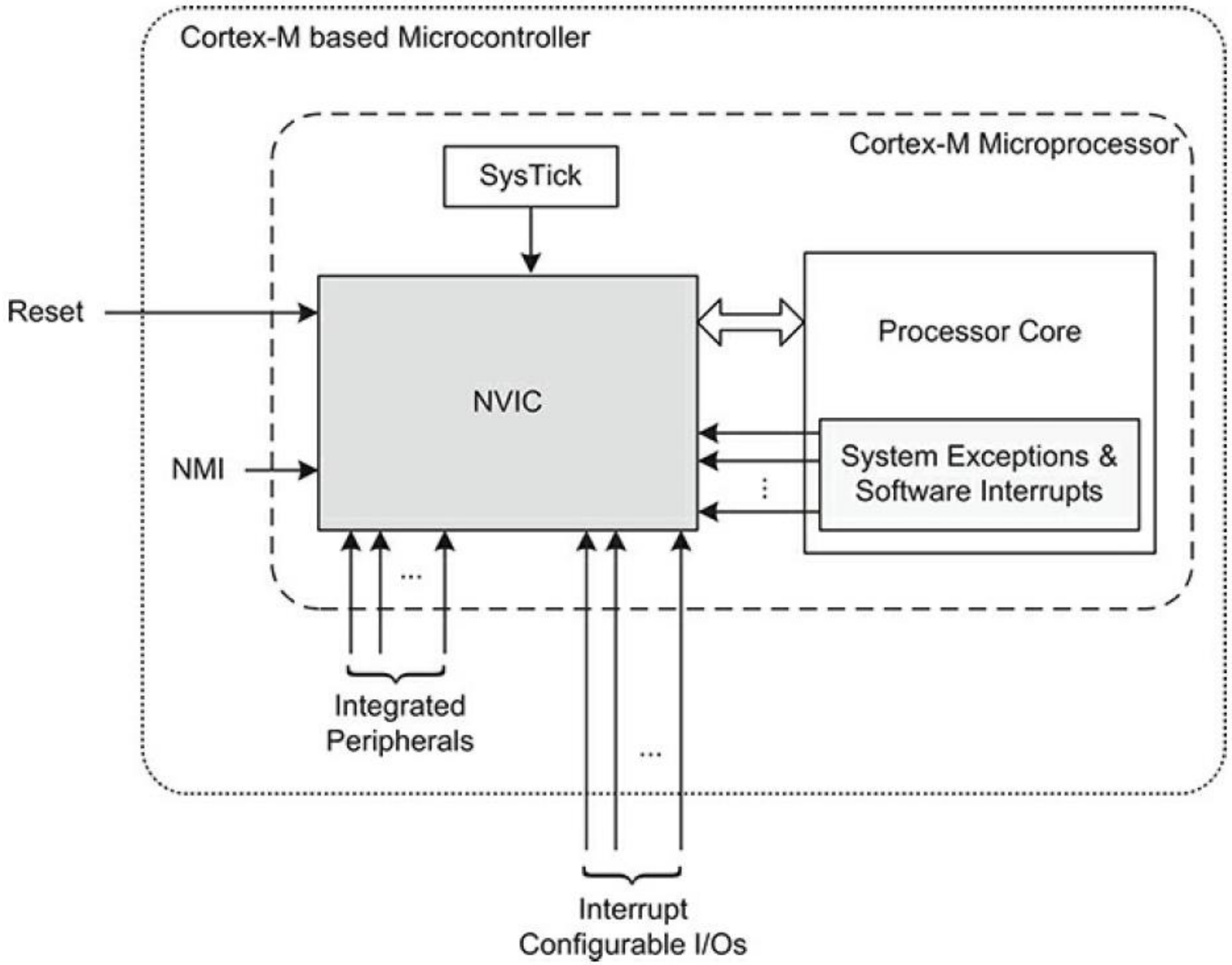
NVIC, System Exceptions and Interrupts and their Priorities & Sates

# CORTEX-M EXCEPTIONS

## CORTEX-M EXCEPTIONS AND INTERRUPTS

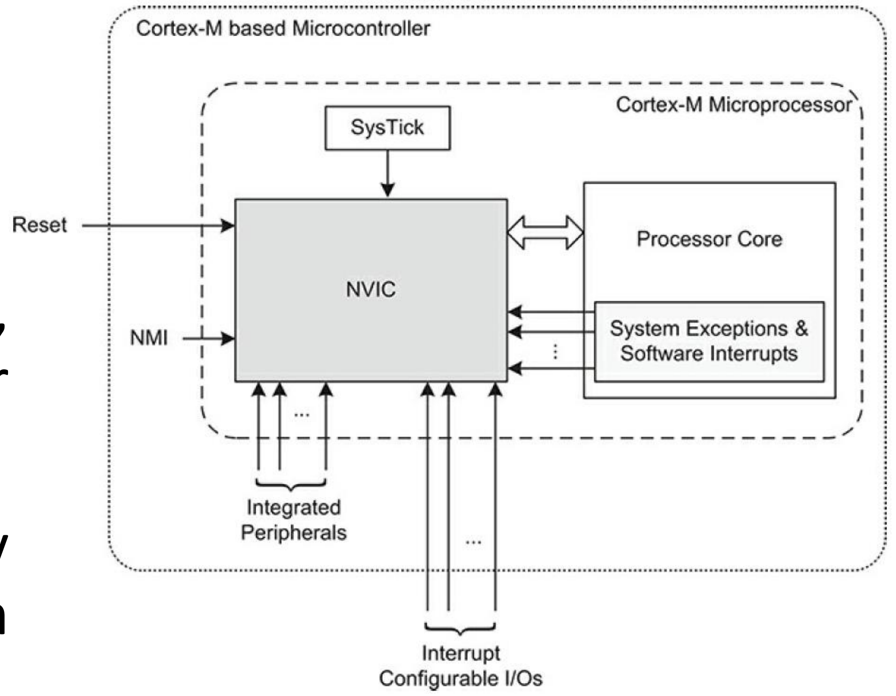
- All Cortex-M processors have a NVIC (Nested Vector Interrupt Controller) that is responsible for handling exceptions and interrupts
- Exceptions are numbered 1 to 255 and according to ARM nomenclature:
  - Exceptions numbered 1 to 15 are called **System Exceptions** or simply **Exceptions**
  - Exceptions numbered 16 to 255 are called **Interrupts**

Block diagram showing NVIC connectivity with ARM Core and different interrupt sources


Cortex-M based Microcontroller
Cortex-M Microprocessor
SysTick
Reset
NVIC
Processor Core
NMI
System Exceptions & Software Interrupts
...
Integrated Peripherals
...
Interrupt Configurable I/Os

## Nested Vector interrupt Controller (NVIC)

### CORTEX-M EXCEPTIONS AND INTERRUPTS



Cortex-M based Microcontroller
Cortex-M Microprocessor
SysTick
Reset
NVIC
Processor Core
NMI
System Exceptions & Software Interrupts
Integrated Peripherals
Interrupt Configurable I/Os

- In ARM Cortex-M based µProcessor Architecture, NVIC is tightly integrated with Cortex-M processor core
- NVIC can be configured for the desired functionality using its memory-mapped control registers, mostly in privileged mode
- NVIC also contains a timing module called SYSTICK timer that is responsible for generating a timing reference used by system software to manage and schedule its activities
- NVIC supports 1-240 peripheral interrupts (correspondingly exceptions 16-255), which are also known as **Interrupt Requests** (IRQs)
- Almost all the Cortex-M based microcontrollers support exceptions 1-15, however, the actual number of interrupts that are supported by the microcontroller is determined by the hardware manufacturers

# System Exceptions and Interrupts

## CORTEX-M EXCEPTIONS AND INTERRUPTS

Exception #, their Labels and Descriptions

| # | Label | Description |
|---|---|---|
| 1 | Reset | System Reset Exception |
| 2 | NMI | Non-maskable Interrupt. The use of this system exception is defined by the user |
| 3 | Hard Fault | This exception is caused by the Bus Fault, Memory Management Fault, or Usage Fault. The Usage Fault occurs if the corresponding interrupt handler cannot be executed. |
| 4 | Memory Management Fault | This fault detects memory access violations to regions that are defined in the Memory Protection Unit (MPU). One possibility can be code execution from a memory region with read/write access only. |
| 5 | Bus Fault | This system exception occurs when memory access errors are detected when performing instruction fetch, data read or write, interrupt vector fetch or register stacking. |
| 6 | Usage Fault | Can occur due to execution of undefined instruction, unaligned memory access (in case of multiple data word load/store instructions). When this exception is enabled, it can detect, divide-by-zero as well as unaligned memory access. |

# System Exceptions and Interrupts

## CORTEX-M EXCEPTIONS AND INTERRUPTS

Exception #, their Labels and Descriptions

| # | Label | Description |
|---|---|---|
| 7-10 | Reserved | - |
| 11 | SVC | SuperVisor Call used by operating system. |
| 12 | Debug Monitor | Debug exception due to events including breakpoints, watchpoints, etc. |
| 13 | Reserved | - |
| 14 | PendSV | An OS based software exception for scenarios like context switching. |
| 15 | SysTick | Exception generates by System Tick Timer. This timer can be used by the OS for system timing reference generation. |
| 16 | Interrupt 0 | Peripheral Interrupt 0 also called IRQ0. Can be connected to on-chip peripherals or interrupt I/O lines. This argument is valid for all IRQs. |
| 17 | Interrupt 1 | Peripheral Interrupt 1 also called IRQ1. |
| … | … | … |
| 255 | Interrupt 239 | Peripheral Interrupt 239 also called IRQ239. |

# Exception and Interrupt Priority

## CORTEX-M EXCEPTIONS AND INTERRUPTS

- *What if multiple exceptions or interrupt occur at the same time?*
- *Since a microcontroller can perform a single task at a time, then how will it manage to perform these multiple tasks?*

- Exceptions or interrupts can be assigned priority level and microcontroller performs these tasks based on the assigned priority
- In ARM Cortex-M architecture, a higher priority corresponds to a smaller number assigned for priority level
- When the exception priorities are enabled, a higher priority exception can preempt a lower priority (correspondingly a larger value in priority level) exception.

## Exception and Interrupt Priority

### CORTEX-M EXCEPTIONS AND INTERRUPTS

- ARM Cortex-M based microcontroller support 3 fixed highest-priority levels and up to 128 programmable priorities levels
- Since higher priority corresponds to lower priority value, Reset interrupt is the highest priority interrupt
- Number of programmable priority levels depends on the microcontroller chip manufacturer
- For ARM Cortex-M4 microcontroller on TI TIVA LaunchPad, there are only 8 programmable priority levels for interrupts
- Interrupt-priority level configuration registers are used to assign the required priority level to a specific interrupt

Priority Assignment of different Exceptions

| Exception # | Label | Priority |
|---|---|---|
| 1 | Reset | -3 |
| 2 | NMI | -2 |
| 3 | Hard Fault | -1 |
| 4 | Memory Management Fault | Programmable |
| 5 | Bus Fault | Programmable |
| 6 | Usage Fault | Programmable |
| … | … | … |
| 16 | Interrupt 0 | Programmable |
| 17 | Interrupt 1 | Programmable |
| … | … | … |
| 255 | Interrupt 239 | Programmable |

## Interrupt States

CORTEX-M EXCEPTIONS AND INTERRUPTS

- Due to multiple priority levels of different interrupts, an interrupt can have one of the following operating states.

1. **Active State**
   - The interrupt is in active state when it is being serviced by the processor, but the servicing has not been completed yet
   - An exception handler of higher priority can interrupt the execution of another lower priority exception handler. In this case, both exceptions are in the active state.

2. **Inactive State**
   - An inactive state of an interrupt corresponds to the situation when no interrupt condition has been generated from the corresponding interrupt source
   - The interrupt is neither active nor pending in this state

3. **Pending State**
   - The interrupt is waiting to be serviced by the processor as it is busy in servicing a high priority interrupt
   - The pending state changes to active state when the servicing corresponding to that interrupt starts.

4. **Active and Pending State**
   - An interrupt is being serviced by the processor and there is a pending interrupt from the same source