SJSU SAN JOSÉ STATE UNIVERSITY

Charles W. Davidson College of Engineering

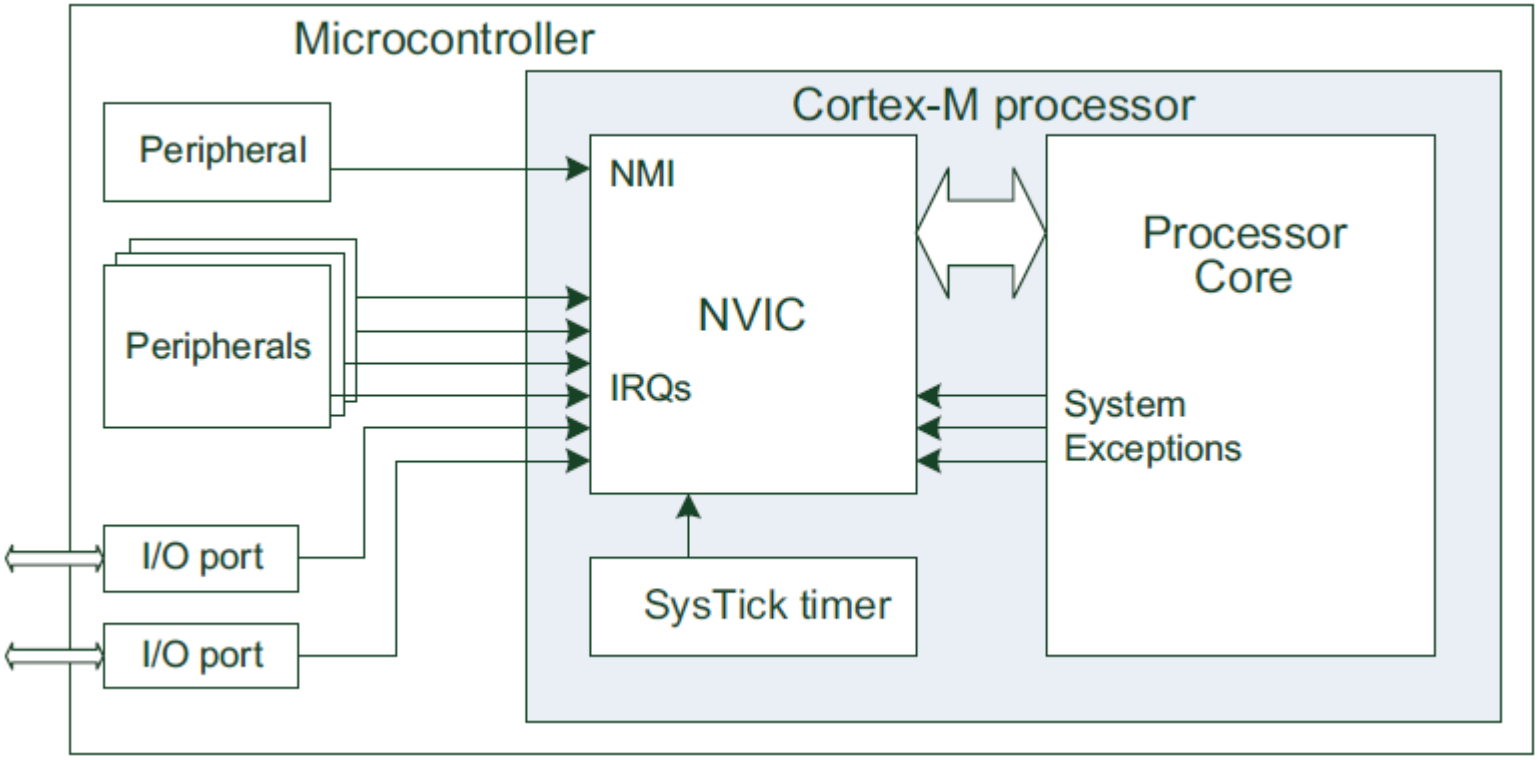
Department of Computer Engineering

Real-Time Embedded System
Co-Design
CMPE 146 Section 1
Fall 2024

MCU Control Units

Interrupt Controller

- ARM Cortex-M processors: Nested Vectored Interrupt Controller (NVIC)
  - Tightly integrated with processor core for lower interrupt latency
- Interrupt sources managed by NVIC in a typical MCU:



- Can handle large number of interrupts
  - Up to 256 interrupts

Diagram source: Joseph Yiu, The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors, third edition, Elsevier, Inc., 2014.

Interrupt Sources

- External interrupts
  - Asynchronous events generated by external hardware devices
    - Examples: UART, ADC, Timers, I/O ports, etc.
- Internal interrupts
  - Synchronous events generated by the processor itself during instruction execution
  - Conditional, some examples:
    - Divide-by-zero error
    - Memory access violation
    - Invalid instruction
- Software interrupts
  - Synchronous events generated by special instructions in a program
  - Unconditional; instruction always generates an interrupt.

Functionality of Interrupt Controller

- Masking
  - Programmable to ignore interrupt inputs
  - Some highly critical ones cannot be ignored
    - Reset, hardware fault, watchdog timer
- Prioritizing
  - An interrupt source can be assigned with a priority level
    - Higher-priority ones can pre-empt lower-priority ones during interrupt handling
  - ARM Cortex-M3/M4 processors support up to 256 levels
- Managing level and edge triggering
  - Level triggering
    - Input interrupt signal reaches certain level, high or low
  - Edge triggering
    - There is a change in the input interrupt signal, high-to-low or low-to-high

Interrupt Service Routine

Interrupt Service Routine (ISR) is a special program segment to handle an interrupt. The general process of handling:

1. Interrupt controller provides processor a vector number associated with the interrupt input
2. Processor finishes the instruction(s) currently being executed
3. Based on the number, processor picks up a vector from a table in memory
   - A vector points to the starting address of the ISR
4. Processor invokes the ISR
5. ISR saves registers to be affected onto the stack
6. ISR executes instructions to handle the requesting interrupt
7. ISR restores registers that were saved
8. ISR returns
9. Processor resumes the interrupted program execution

ARM Interrupt Vector Table

- For ARM processors, interrupts are referred to as exceptions
- Cortex-M processor interrupt vector table:

Memory Address		Exception Number
	1	
	1	
0x0000004C	Interrupt#3 vector 1	19
0x00000048	Interrupt#2 vector 1	18
0x00000044	Interrupt#1 vector 1	17
0x00000040	Interrupt#0 vector 1	16
0x0000003C	SysTick vector 1	15
0x00000038	PendSV vector 1	14
0x00000034	Not used	13
0x00000030	Debug Monitor vector 1	12
0x0000002C	SVC vector 1	11
0x00000028	Not used	10
0x00000024	Not used	9
0x00000020	Not used	8
0x0000001C	Not used	7
0x00000018	Usage Fault vector 1	6
0x00000014	Bus Fault vector 1	5
0x00000010	MemManage vector 1	4
0x0000000C	HardFault vector 1	3
0x00000008	NMI vector 1	2
0x00000004	Reset vector 1	1
0x00000000	MSP initial value	0

Interrupt Nesting

- Interrupt of higher priority can interrupt the active ISR
- Normal program execution is interrupted by x, which is further interrupted by y and z
- Both ends of ISR cannot be interrupted
  - Processor state are being saved or restored

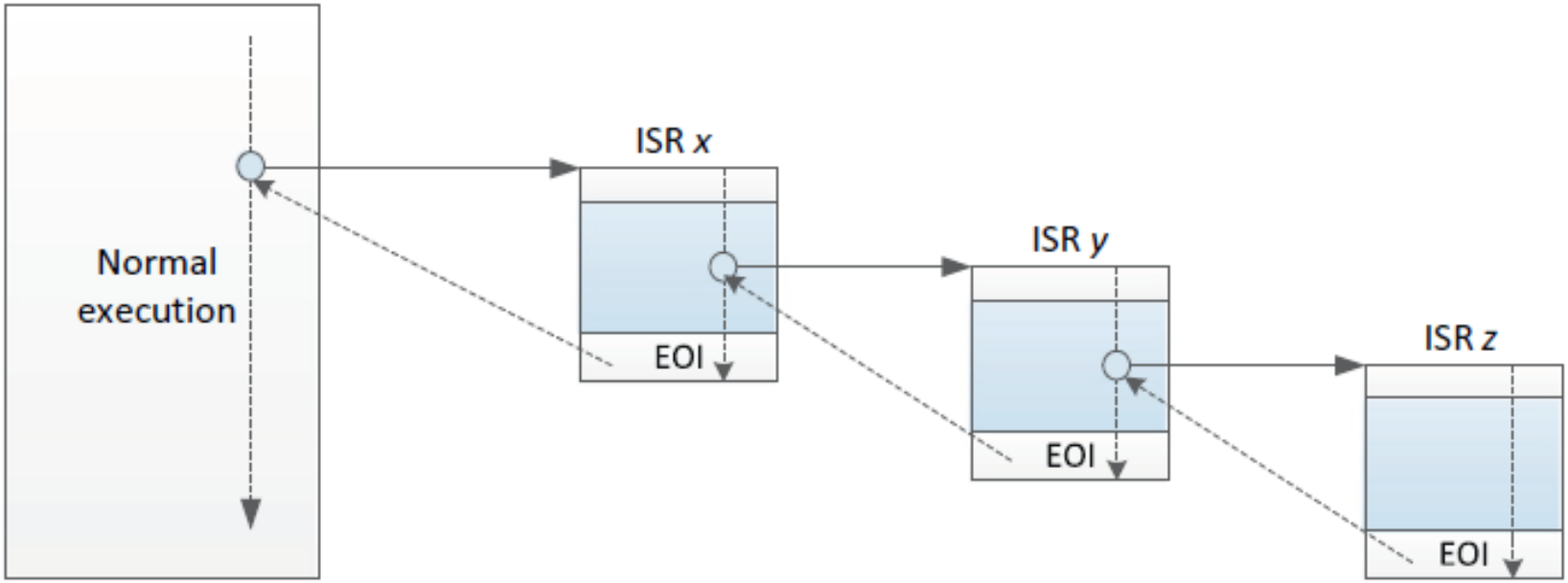


Diagram source: Xiaocong Fan, Real-Time Embedded Systems: Design Principles and Engineering Practices, Elsevier, Inc., 2015.

Interrupt Latency

- Interrupt latency is critical for real-time applications
- Defined as the time interval between the interrupt request and the first ISR instruction being executed
- Major impacts on interrupt latency can be produced by
  - Higher-priority ISR being active
  - Interrupts being temporarily disabled by user application or OS
- Other impacting factors
  - MCU architecture design
    - Tight integration of interrupt controller helps
  - Interrupt controller design
    - Higher number of priority levels increases latency (and power, too)
  - Sharing interrupt lines increases handling overhead
  - Memory system performance
    - Interrupt vector table is stored in memory; slower memory means longer latency
  - Pipeline length
    - Processor needs to finish instruction(s) being executed
- ARM Cortex-M3/M4 processors: 12 cycles with zero wait state memory

Data Sharing with ISRs

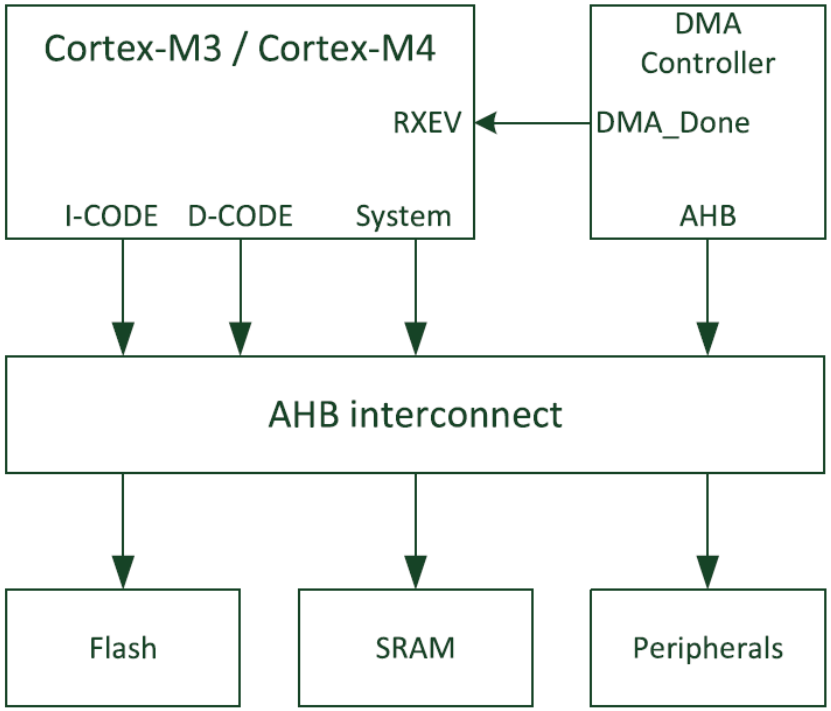
- Application tasks and ISRs typically communicate through shared data variables in memory
  - Quite often, there is a higher-level application task to do further processing based on the set of data/signals collected by ISR
- If communication protocol is not designed properly, race conditions may occur
  - Creates intermittent error or even system failure
    - Problem could be extremely hard to debug
- One obvious solution: Disable interrupts before accessing data
  - May increase latency for other interrupts
- Better solution: Elimination of simultaneous accesses
  - No race condition happens
  - No need to disable interrupt
  - For example, use a queue data structure for communication
    - ISR adds data records to the queue
    - Application task retrieves records from the queue at a later time

DMA Controller

- Direct Memory Access (DMA) controller provides data transfer operations without processor intervention
- Copying large amount of data with processor is inefficient
  - Copying loop repeatedly executes the same instructions
    - Read data from memory to register
    - Write data from register to memory
  - Takes up memory bandwidth if instructions and data are in the same memory region
    - Can consume more than a few cycles per transfer
  - Uses more energy
- DMA Controller is designed just for data transfer
  - No program loop to execute
  - Takes fewer cycles per transfer than processor
  - Reduces processor's workload
  - Provides multiple channels with priorities for doing multiple transfers in parallel

System Configuration

- DMA controller is one of the system bus masters
- One typical system configuration:



- Transfer types:
  - Memory-to-memory
  - Memory-to-peripheral
  - Peripheral-to-memory

General Working Process

1. Processor provides DMA controller
   - Source memory (or memory mapped I/O) address
   - Destination memory (or memory mapped I/O) address
   - Transfer size
   - Operating Mode
2. Initiates the transfer
   - Can be triggered by an external event (e.g., program instruction, hardware timer)
3. DMA controller transfers the data
4. Processor waits for transfer completion; two methods:
   - Polls register status
     - Wastes bus bandwidth
   - Notified by interrupt

Main Features

- Address increment
  - Source address can increment after read
  - Destination address can increment after write
  - Increment can be 0, 1 (byte transfer), 2 (word transfer), 4 (long word transfer)
- DMA Cycle
  - A sub-transaction of the entire process
    - Defined by the trigger
  - Can transfer single or a block of data
  - Arbitration for other channels (if necessary) after each cycle

Operation Modes

Common modes of operation for embedded applications

- Basic DMA transfer
  - Each trigger produces a transfer of a single unit of data
  - Completion occurs when the required number of transfers are done
  - Suitable for peripheral-to-memory transfer
- Auto-request
  - Once triggered, complete all the required number of transfers
  - Suitable for memory-to-memory transfer
- Ping-pong
  - Two data regions are used (e.g., using double-buffering scheme)
  - After one DMA cycle, switch to the other region
  - Processor is notified after each cycle is completed so that it can start processing the received data immediately
  - Suitable for high-speed data acquisition where block of data is processed one at a time

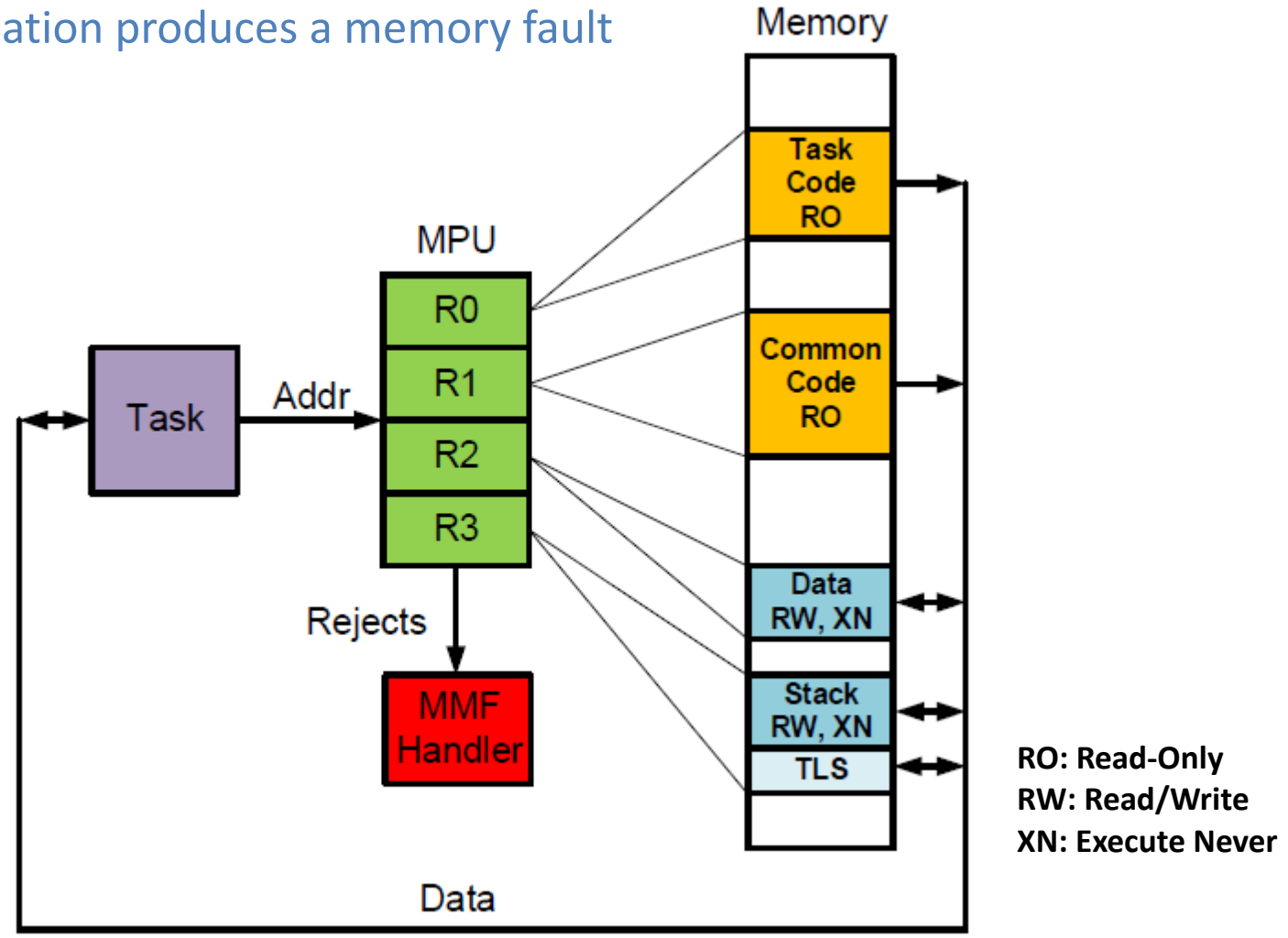
Memory Management Unit

Memory Management Unit (MMU)

- General-purpose microprocessors have been using MMU for decades for
  - System security
    - Separate user programs and OS
    - Isolate processes
  - Ease of programming with virtual memory
- Many medium or large MCUs have integrated MMU security features
  - On ARM processors, the controller is called Memory Protection Unit (MPU)
  - Makes systems more reliable
    - One malfunctioning application task will not bring down the system
  - Makes systems more secure against attacks
    - Can prevent instruction execution from data memory (injection attack), for example
- Tightly coupled with the processor core
  - No impact on memory access performance
- Most simple applications do not require an MMU
  - Can be disabled entirely when not needed

Memory Partitioning

- MMU/MPU divides memory map into regions for efficient management
  - Regions are assigned attributes
- Access violation produces a memory fault



RO: Read-Only
RW: Read/Write
XN: Execute Never

Diagram source: http://www.smxrtos.com/articles/mpusecurity.htm

Memory Regions

For each region, MPU maintains a number of parameters

- Starting address and size
  - Regions can overlap
    - Tasks or processes can share memory for communication
- Enable/Disable
- Main access attributes
  - Read
  - Write
  - eXecute Never (XN)
    - Prevents injection attack
- Types
  - Shareable
    - For multiple processor cores to maintain cache coherency
  - Cacheable
    - Memory mapped I/Os should not be cached

Configurations

- Static configuration without RTOS (Real-Time Operating System)
  - A simpler approach
    - MMU configuration remains unchanged after initial setup
  - Protects read-only data in SRAM
  - Detects stack overflow
  - XN on SRAM to prevent injection attack
- Dynamic configuration with tasks running under RTOS
  - Have the same protection and detection benefits of the static configuration
  - Tasks can be isolated in the code, data and stack spaces
  - Tasks can be restricted to have access to certain I/O ports