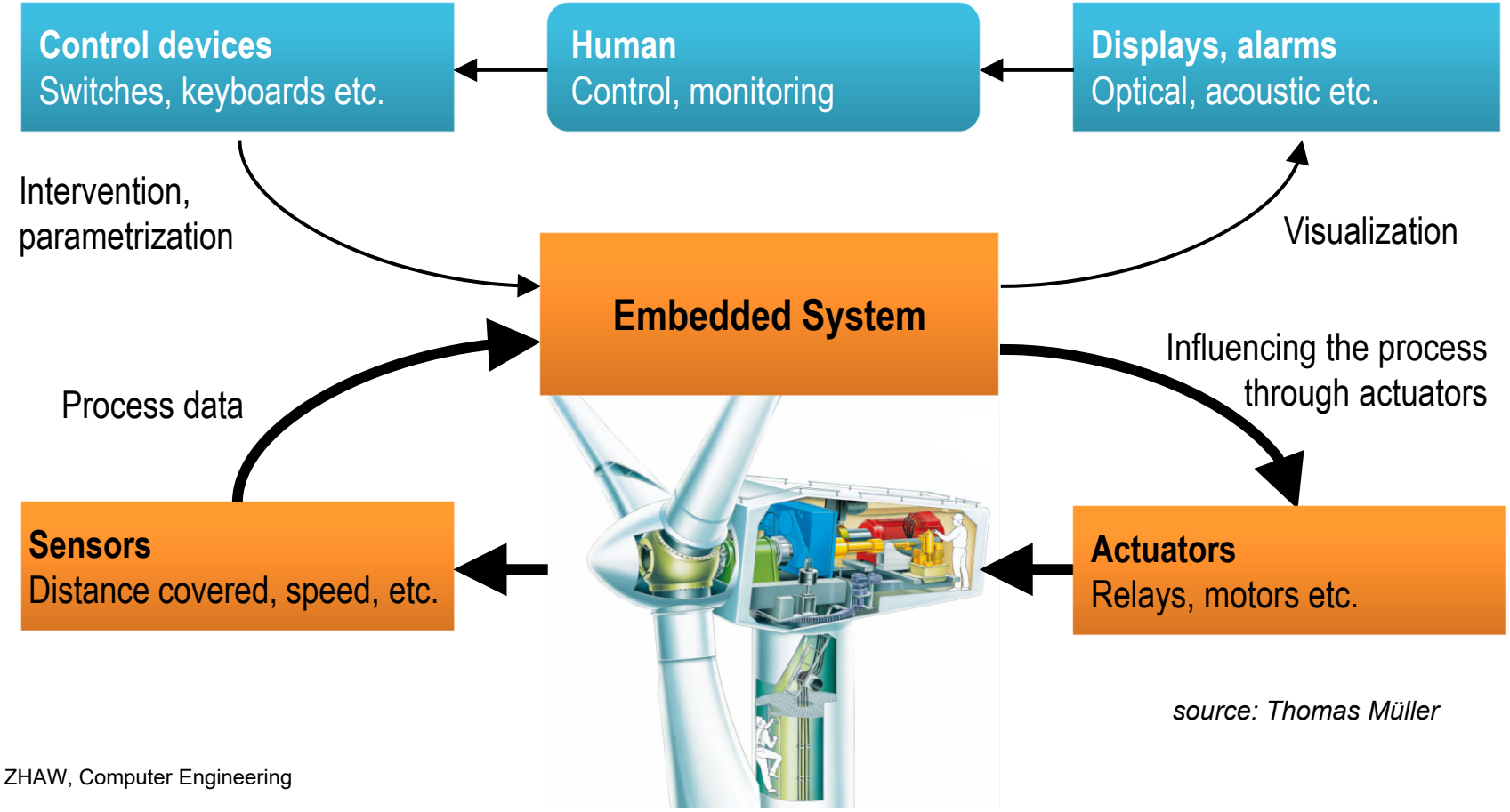
Detecting Events – Interrupt Performance

Computer Engineering 2

Motivation

- **How do you recognize events?**
  - Cyclic queries or interrupt-driven?



source: Thomas Müller

Agenda

School of Engineering
InES Institute of Embedded Systems

- **Polling**
- **Interrupt Driven I/O**
- **Interrupt Performance**
- **Interrupt Latency**
- **Managing Latency**
- **Interrupt Driven FSM**

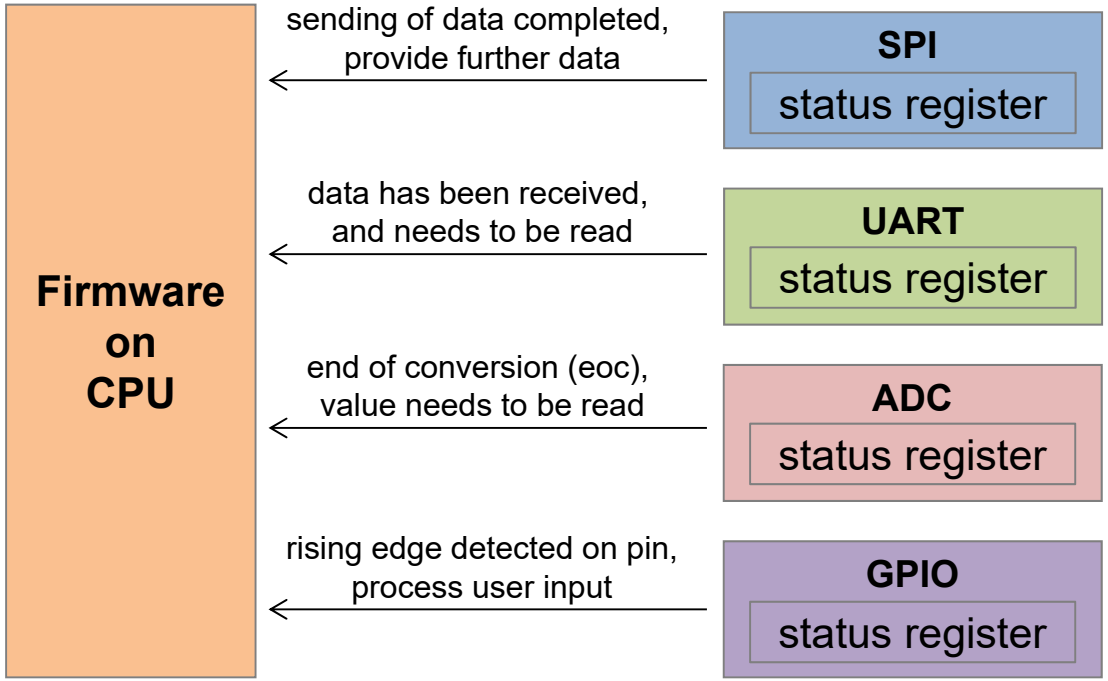
Learning Objectives

At the end of this lesson you will be able

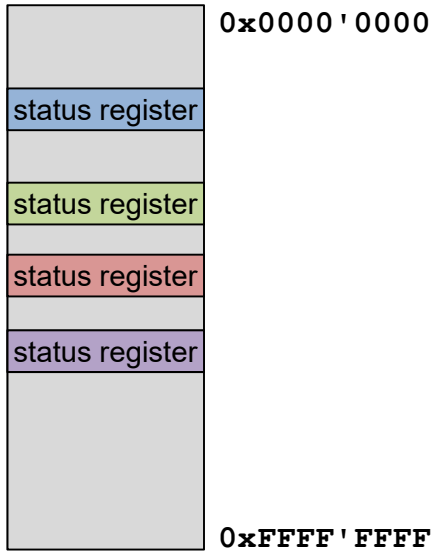
- to explain the methods "Polling" and "Interrupt Driven I/O"
- to name important factors for the two methods
- to enumerate advantages and disadvantages of the methods
- to evaluate for a given situation whether polling or interrupt driven I/O is appropriate
- to quantify timing aspects for interrupt driven I/O
- to comprehend the term "Interrupt Latency" and name potential sources
- to explain the term ‘pre-emption’
- to understand approaches to manage interrupt latency
- to name the components and explain the structure of an interrupt driven FSM

Firmware Has to Act on Events

- **Peripherals Signal Events to Firmware**
  - Something happened that requires servicing in firmware



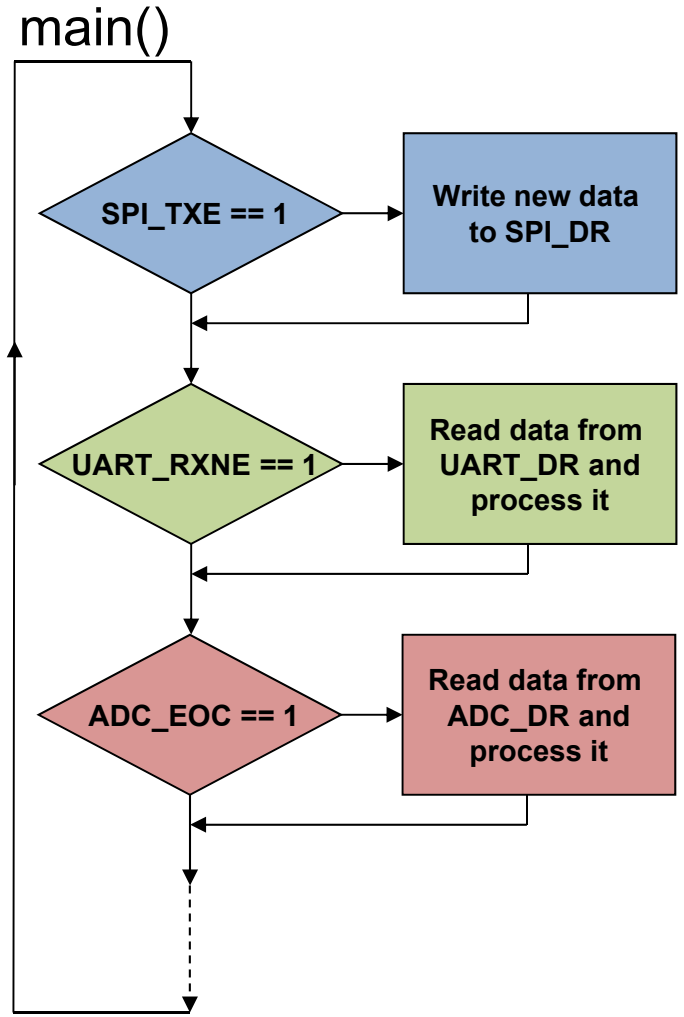
status registers can be addressed and read by firmware



SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmitter
GPIO	General Purpose Input Output
ADC	Analog-Digital Converter

Polling

- **Periodic Query of Status Information**
  - Synchronous with main program
  - Advantages
    - Simple and straightforward
    - Implicit synchronization
    - Deterministic
    - No additional interrupt logic required
  - Disadvantages
    - Busy wait → wastes CPU time
    - Reduced throughput
    - Long reaction times
      in case of many I/O devices or if the CPU is working on other tasks

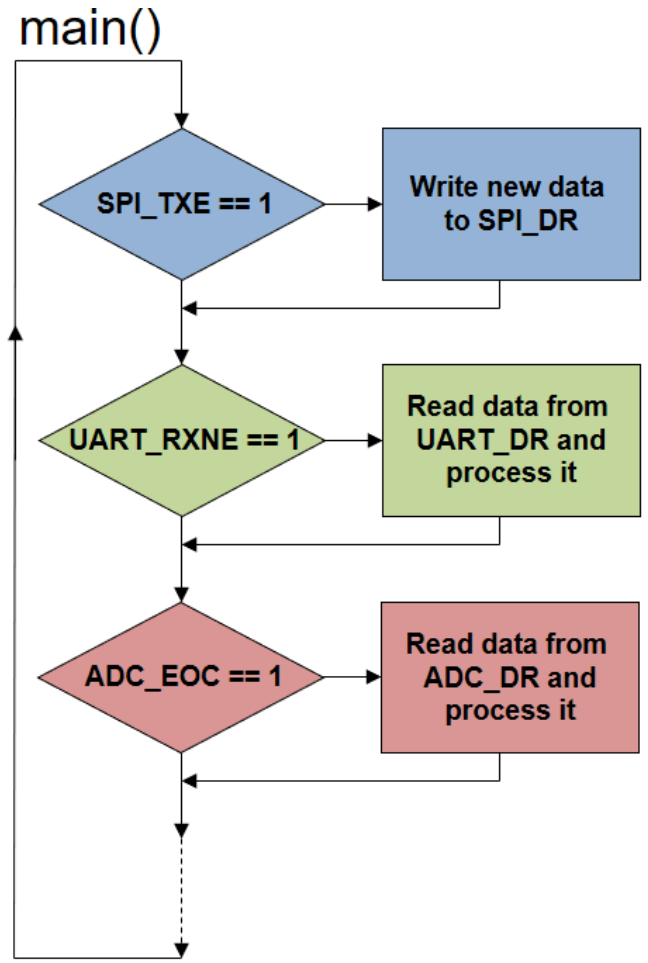


[1] to poll → abfragen

Polling

- **Implementation**

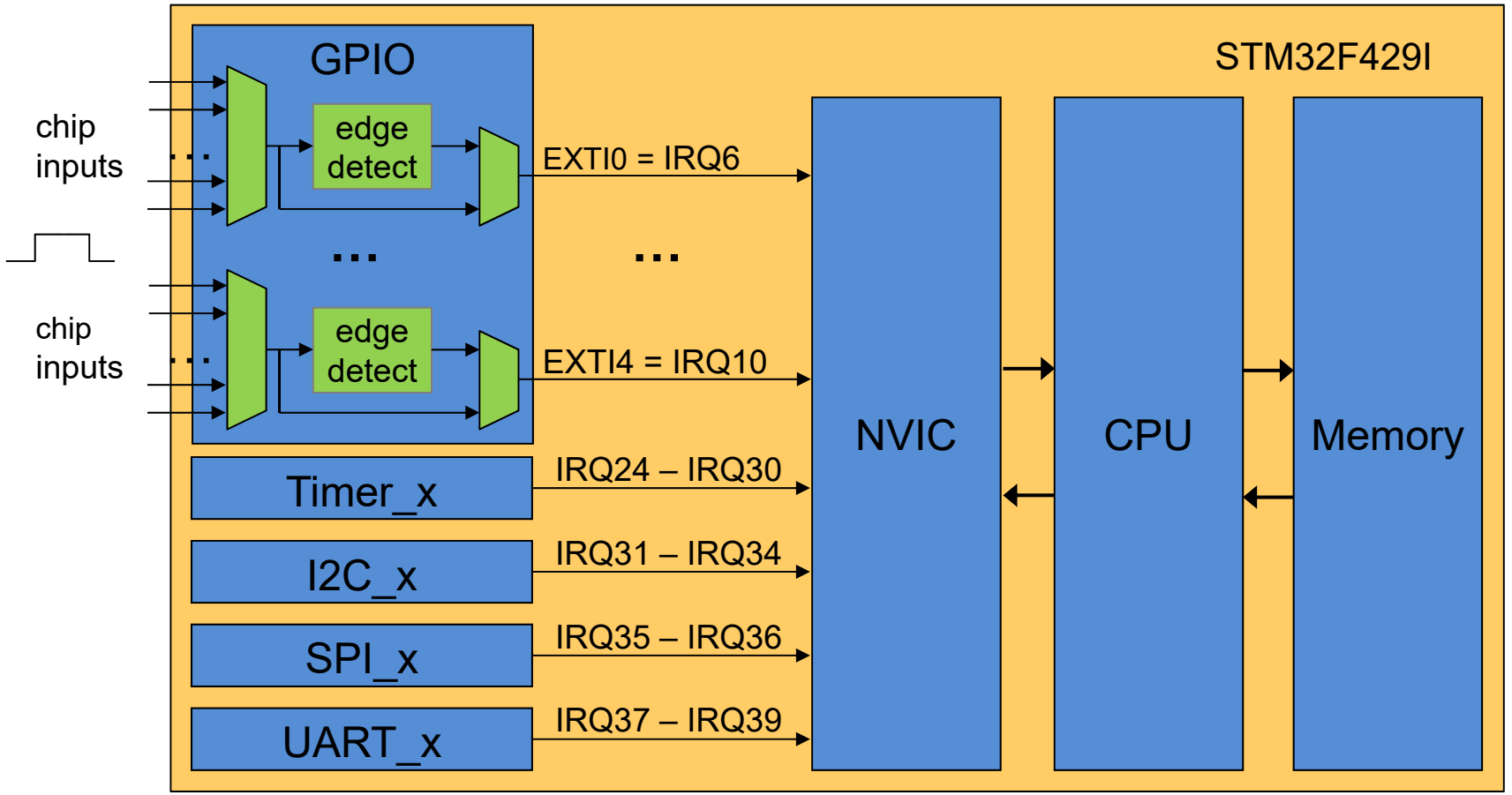
```
while(1){
    if (spi_is_txe_set()) {
        ...
        spi_write_data(...);
    }

    if (uart_is_rxne_set()) {
       uart_data = uart_read_data();
       ...
    }

    if (adc_is_eoc()) {
       adc_data = adc_read_data();
       ...
    }

    ...
}
```



Interrupt Driven I/O

School of Engineering
InES Institute of Embedded Systems

- **Interrupt System STM32F429I**



Interrupt Driven I/O

School of Engineering
InES Institute of Embedded Systems

- **Vector Table and ISR**

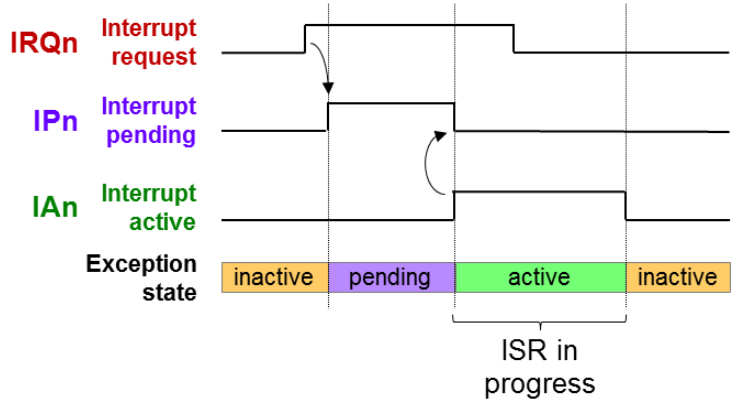
```
; Vector Table Mapped to Address 0 at Reset
                AREA    RESET, DATA, READONLY
                EXPORT  __Vectors
                EXPORT  __Vectors_End
                EXPORT  __Vectors_Size


__Vectors       DCD     __initial_sp              ; Top of Stack
                DCD     Reset_Handler             ; Reset Handler
                DCD     NMI_Handler               ; NMI Handler
                DCD     HardFault_Handler         ; Hard Fault Handler
                DCD     MemManage_Handler         ; MPU Fault Handler
                DCD     BusFault_Handler          ; Bus Fault Handler
                DCD     UsageFault_Handler        ; Usage Fault Handler
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     SVC_Handler               ; SVCall Handler
                DCD     DebugMon_Handler          ; Debug Monitor Handler
                DCD     0                         ; Reserved
                DCD     PendSV_Handler            ; PendSV Handler
                DCD     SysTick_Handler           ; SysTick Handler

                ; External Interrupts
                DCD     WWDG_IRQHandler                 ; Window WatchDog
                DCD     PVD_IRQHandler                  ; PVD through EXTI Line detection
                DCD     TAMP_STAMP_IRQHandler           ; Tamper and TimeStamps through the EXTI line
                DCD     RTC_WKUP_IRQHandler             ; RTC Wakeup through the EXTI line
                DCD     FLASH_IRQHandler                ; FLASH
                DCD     RCC_IRQHandler                  ; RCC
                DCD     EXTI0_IRQHandler                ; EXTI Line0
                DCD     EXTI1_IRQHandler                ; EXTI Line1
                DCD     EXTI2_IRQHandler                ; EXTI Line2
                DCD     EXTI3_IRQHandler                ; EXTI Line3
```

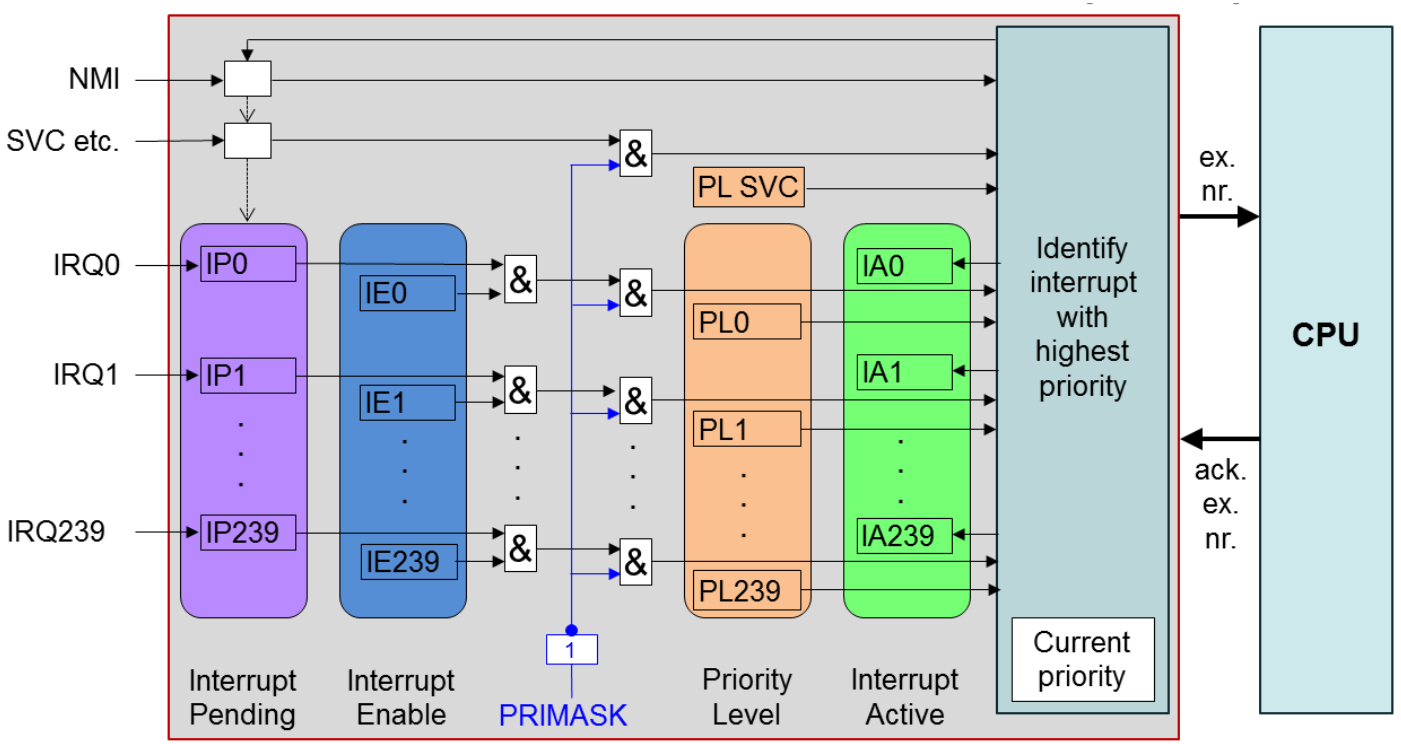
```
/*
 * Interrupt service routines
 * ----------------------------------------------------

void EXTI0_IRQHandler(void)
{
    // clear flag in status register of peripheral

    // service interrupt
}
```

Interrupt Driven I/O

School of Engineering
InES Institute of Embedded Systems

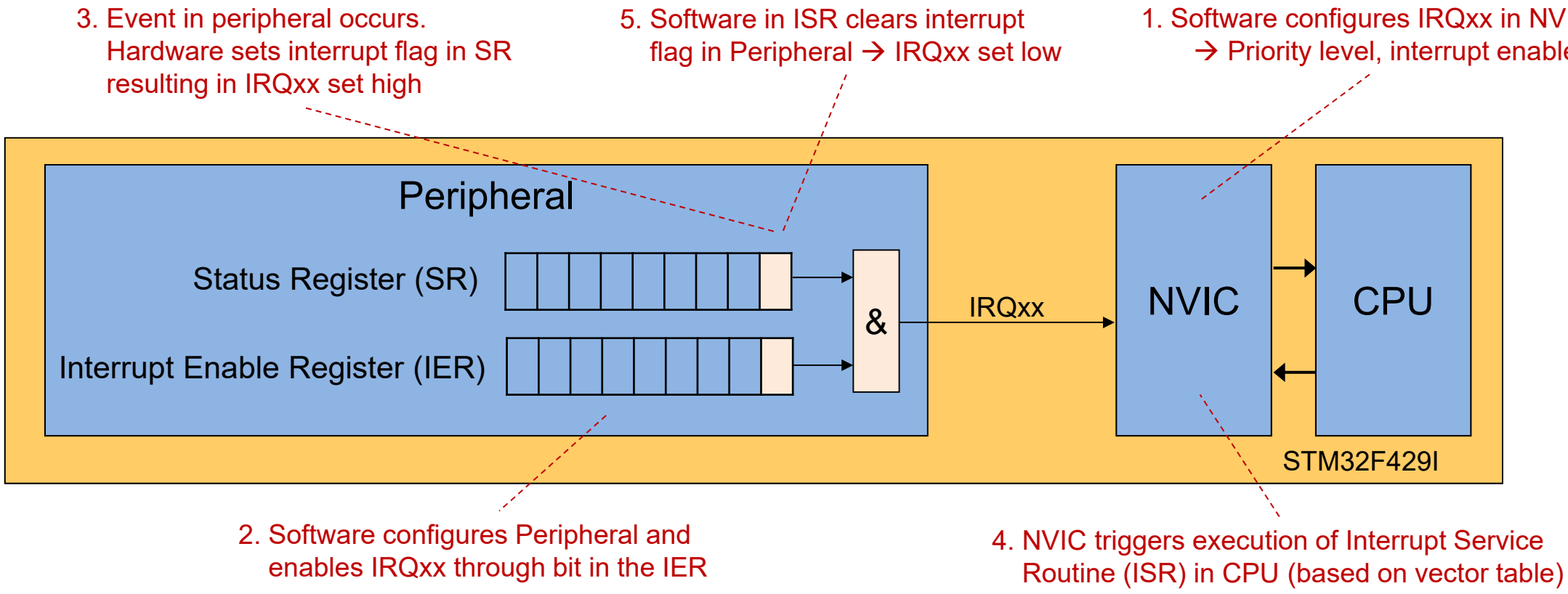
- **NVIC on Cortex-M**
  - Nested Vectored Interrupt Controller





Interrupt Driven I/O

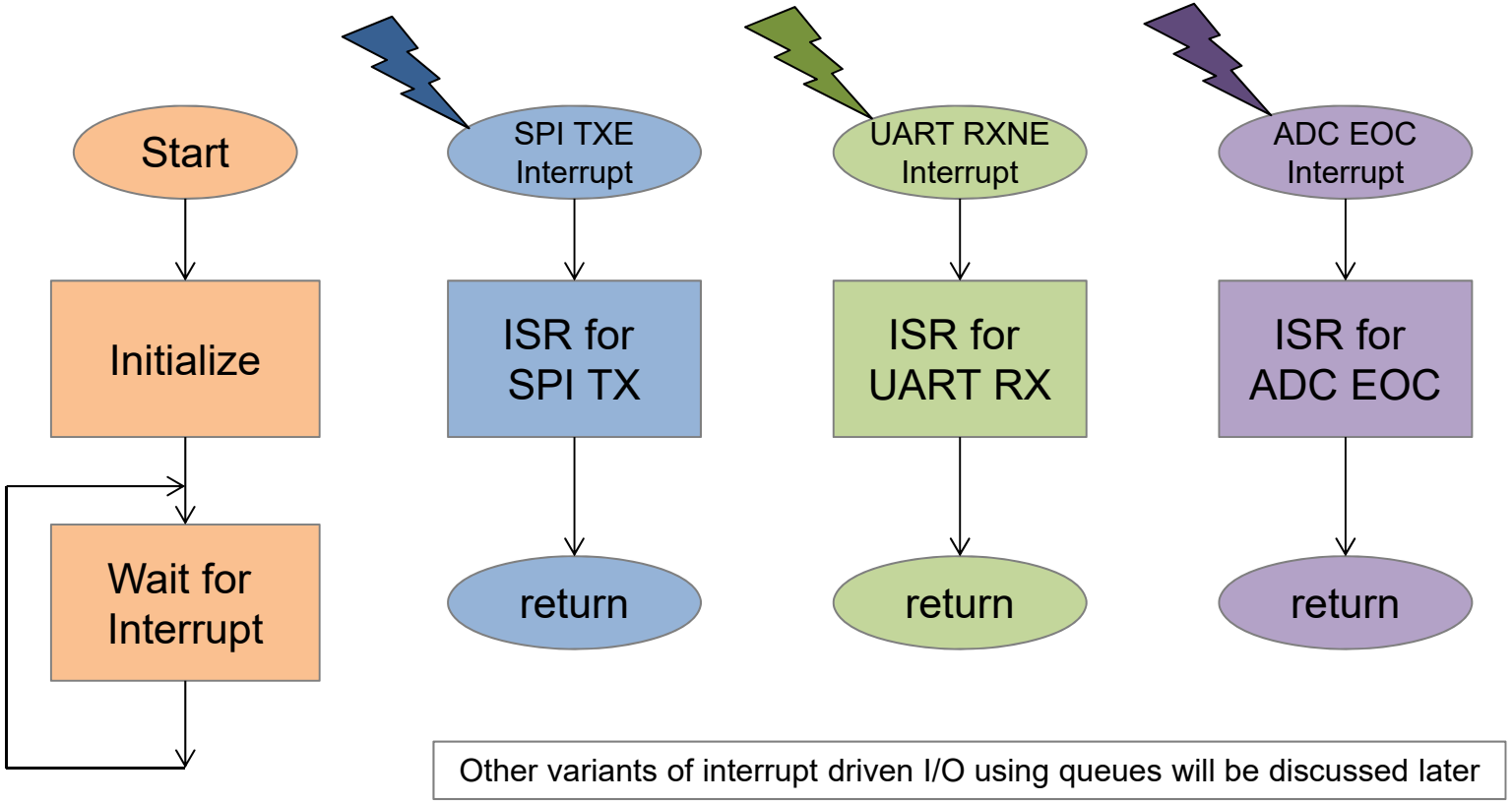
- **Interrupt flags in Status Registers (SR) are hardware set and software reset**
  - Software has to enable IRQxx line in peripheral



Interrupt Driven I/O

School of Engineering
InES Institute of Embedded Systems

- **Interrupt driven I/O**
  - Extreme case: All processing done in interrupt service routines



Other variants of interrupt driven I/O using queues will be discussed later

Interrupt Performance

School of Engineering
InES Institute of Embedded Systems

- **Interrupt frequency**
  - How often does an interrupt occur
  - Varies from source to source, e.g.

f_{INT}

Keyboard → max. ~ 20 interrupts per second in irregular intervals

Serial interface
e.g. 230'400 baud

→ one interrupt every 8 bit
230'400 / 8 = 28'800 interrupts per second

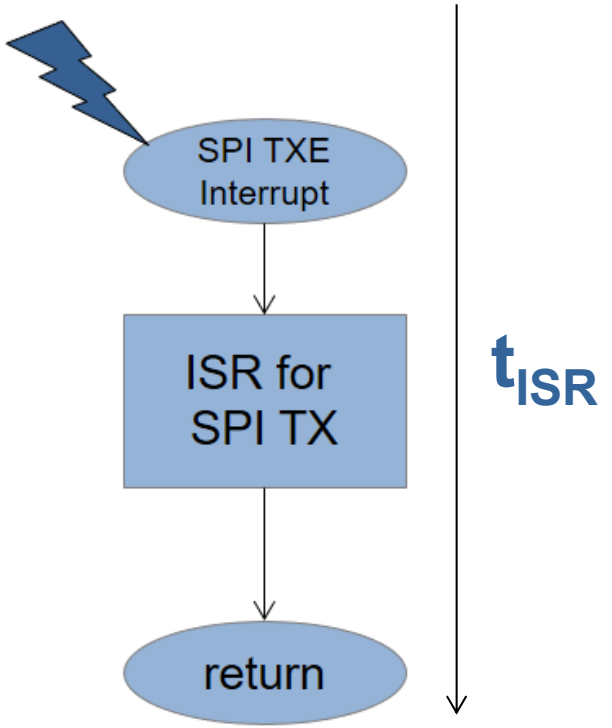
Interrupt Performance

School of Engineering
InES Institute of Embedded Systems

- **Interrupt service time** t_{ISR}

  - Required time to process an interrupt
    - I.e. execution time of ISR

  - Depends on
    - Number of instructions in ISR
    - Required number of clock cycles per instruction
      → depends on CPU architecture
    - CPU clock frequency
    - Time for switching to and returning from ISR



Interrupt Performance

- **Impact on system performance**
  - Percentage of CPU time used to service interrupts

$$\textbf{Impact} = f_{INT} * t_{ISR} * 100\ \%$$

  - Example keyboard

f_{INT} = 20 Hz = 20 $\frac{1}{s}$ t_{ISR} = 6 us [1)]

Impact = 20 Hz * 6 us * 100 % = 0.012 %

  - Example serial interface with 230'400 Baud

f_{INT} = 230'400 / 8 = 28'800 Hz t_{ISR} = 6 us [1)]

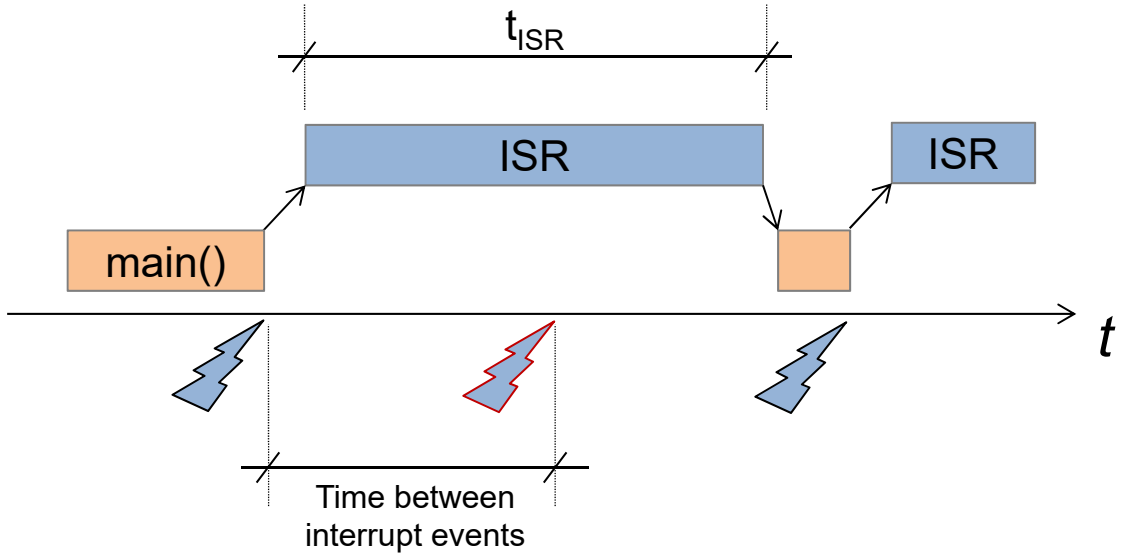
Impact = 28'800 Hz * 6 us * 100 % = 17.3 %

[1)] = assumed value: 6 us ≈ 1000 cycles @ 168 MHz on CT Board

Interrupt Performance

School of Engineering
InES Institute of Embedded Systems

- **t_{ISR} > "Time between two interrupt events"**
  - Some interrupt events will not be serviced (lost)
    - Data will be lost
  - f_{INT} as well as t_{ISR} may vary over time
    - Average may be ok, but individual interrupt events may still be lost



Caution in case of several interrupt sources
- Interrupts can occur simultaneously
- Computing power required for both ISRs

Interrupt Performance

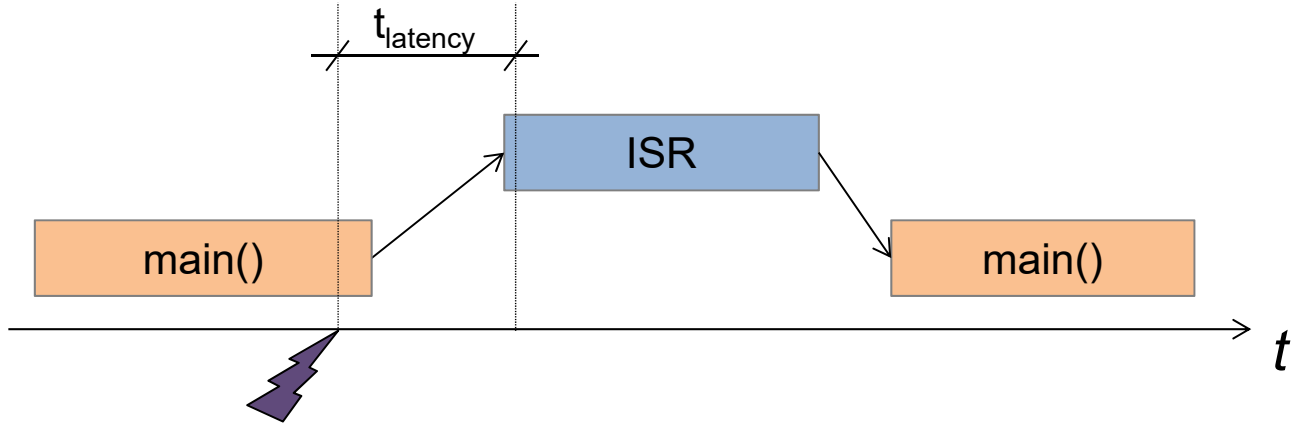
School of Engineering
InES Institute of Embedded Systems

- **Strive for short ISRs**
  - Simpler debugging
  - Execute only time-critical tasks in ISR
    - Move tasks with relaxed time-constraints to main loop
    - Makes ISR available for other time-critical tasks
    - Often simply feed interrupt to queue; dispatch queue in main loop

Interrupt Latency

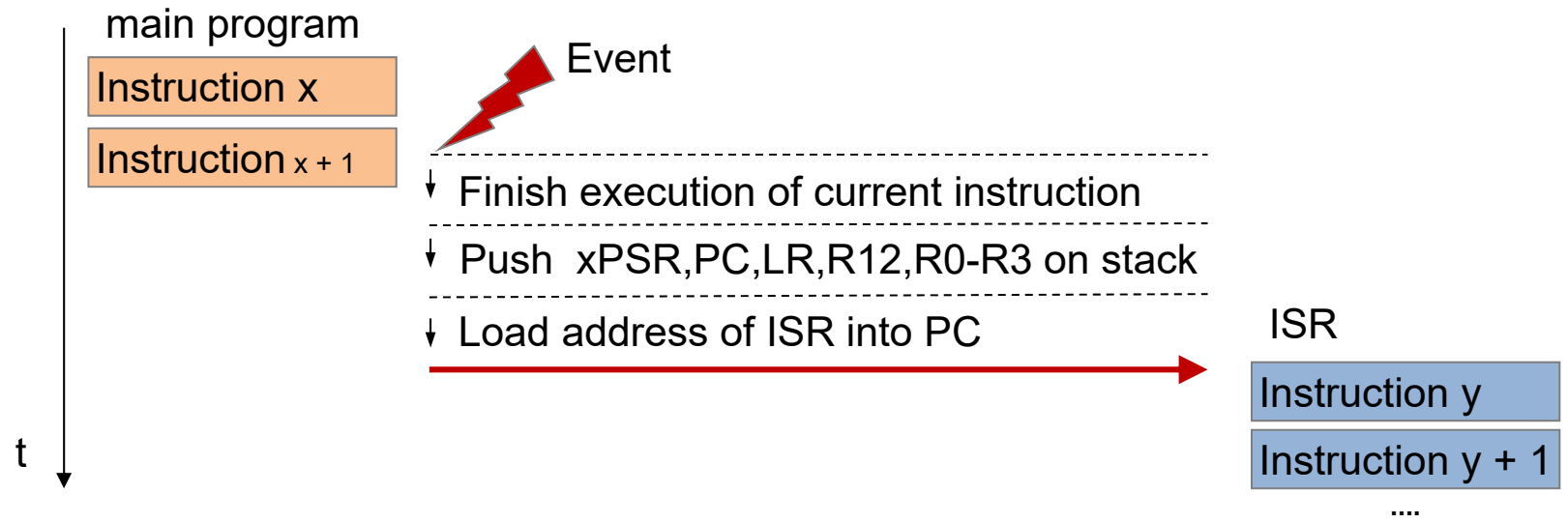
School of Engineering
InES Institute of Embedded Systems

- **Interrupt latency definition**
  - Time between interrupt event and start of servicing by ISR
    - How long does it take until first 'useful' instruction in ISR is executed
  - Range
    - From about 50 nanoseconds (ns) up to several milliseconds (ms)
  - Relevant in cases where guaranteed service times are required
    - E.g. audio/video streaming



Interrupt Latency

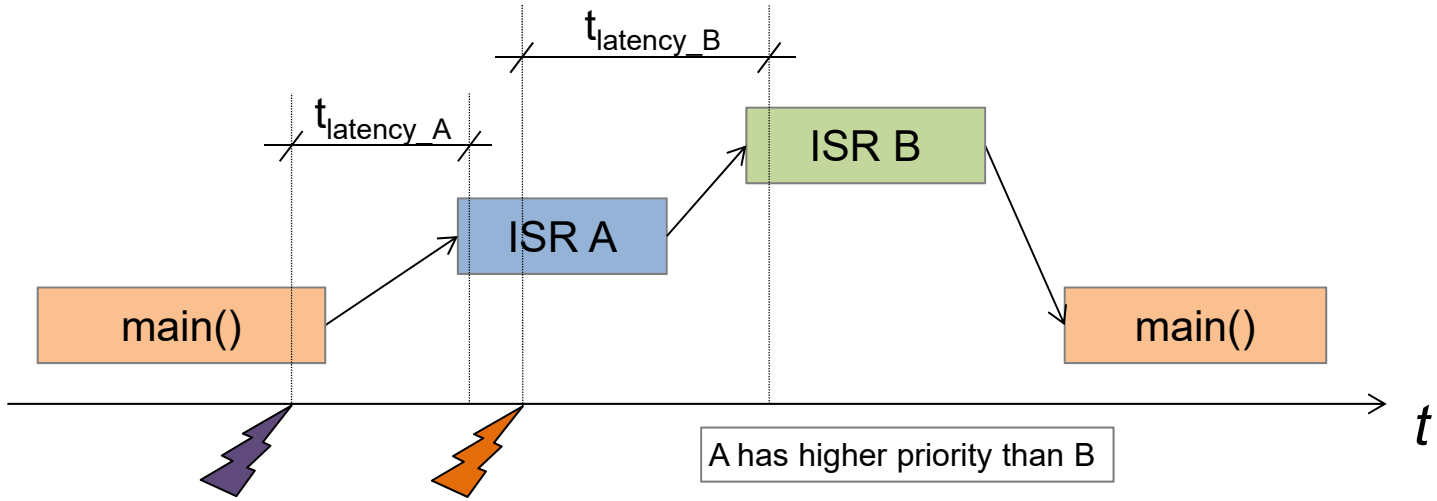
- **Latency influenced by hardware (CPU)**



- Different instructions may have different execution times
- Multi-cycle instructions on Cortex-M3/M4
  - Some (e.g. SDIV/UDIV) are abandoned and restarted after ISR
  - Some (e.g. LDM/STM) are interrupted and resumed after ISR

Interrupt Latency

- **Latency influenced by software (code)**
  - Saving additional registers on stack
  - Process ongoing or higher prioritized ISRs
  - Masked (disabled) interrupts → CPSID i / CPSIE i
  - In case several sources are using the same interrupt line
    - SW has to use polling to know which peripheral requires servicing



Interrupt Latency

School of Engineering
InES Institute of Embedded Systems

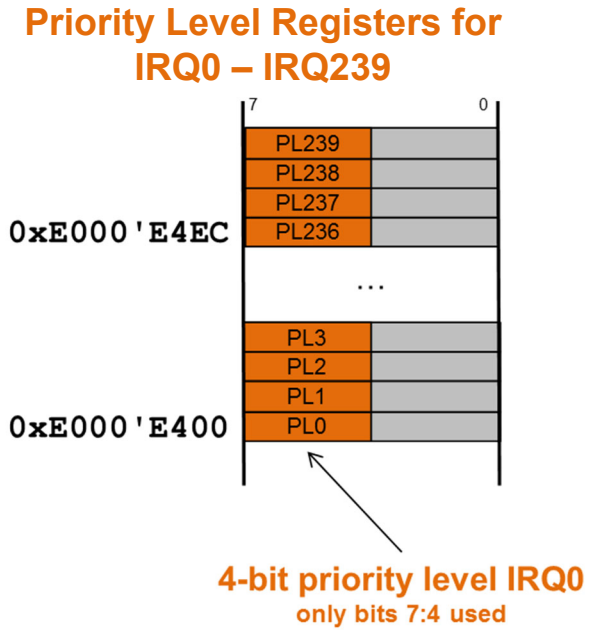
- **Required response time for a specific event**
  - Certain peripherals require fast response (< 1us)
    - Real-time systems, e.g. anti-lock braking system
  - Cannot be guaranteed if maximum interrupt latency is too high

- **f_{INT} too high → Too many interrupts**
  - No CPU cycles left for data processing
    - System is busy calling ISRs
    - Neither ISR nor main loop can process any data
    - Performance of CPU is used only for latencies / context switching

Interrupt Latency

School of Engineering
InES Institute of Embedded Systems

- **Fast response times – low latency**
  - Fast CPU
    - High clock rate
    - Low number of clock cycles per instruction
  - Extremely short polling loops can be fast
  - Pre-emption with appropriate priorities
    - Priority levels programmed in NVIC
    - Real Time OS



source: Techopedia Inc.

Definition - What does *Pre-Emption* mean?

Pre-emption refers to the temporary interruption and suspension of a task, without asking for its cooperation, with the intention to resume that task later. This act is called a context switch and is typically performed by the pre-emptive scheduler, a component in the operating system authorized to pre-empt, or interrupt, and later resume tasks running in the system.

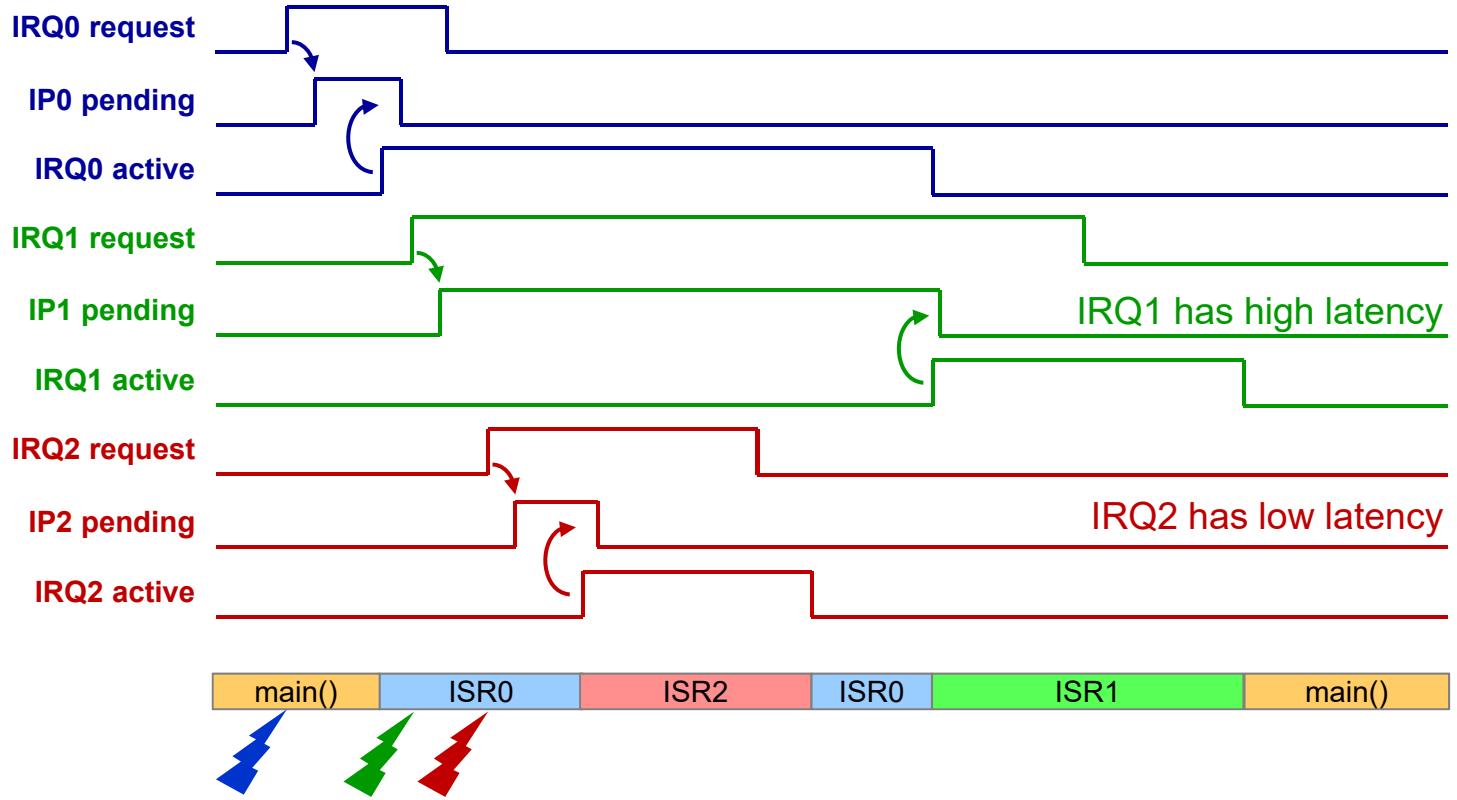
Interrupt Latency

School of Engineering
InES Institute of Embedded Systems

- **Example interrupt priorities**
  - ISR1 does not pre-empt ISR0
  - ISR2 pre-empts ISR0

assuming

IRQ0	PL0 = 0x2	medium priority
IRQ1	PL1 = 0x3	lowest priority
IRQ2	PL2 = 0x1	highest priority



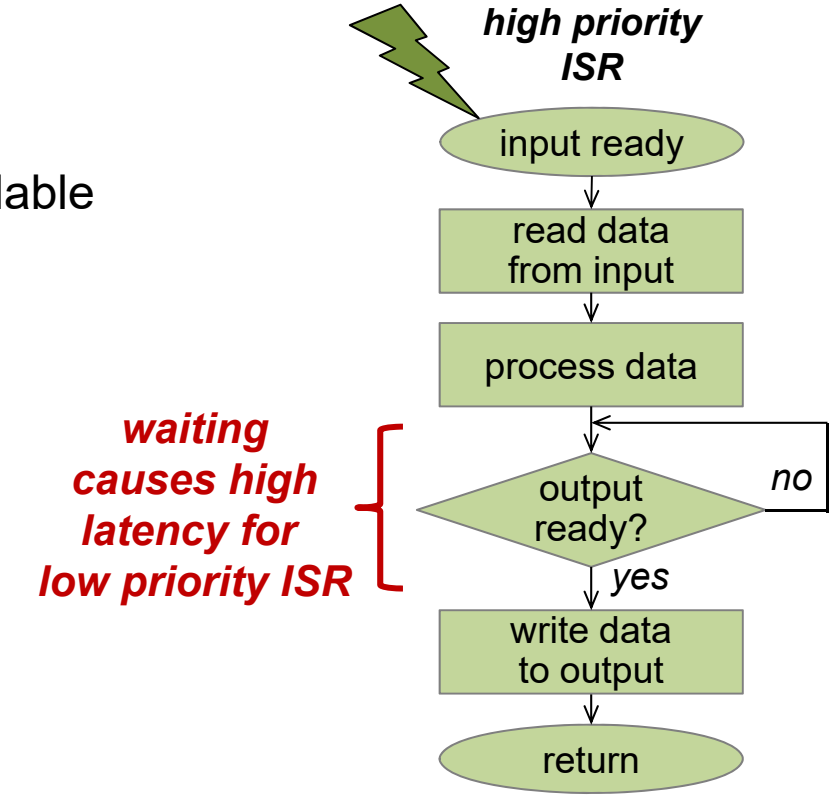
Interrupt Latency

School of Engineering
InES Institute of Embedded Systems

- **Latency consistency**
  - Some applications can tolerate considerable interrupt latency as long as it is consistent
    - I.e. same amount of latency from one interrupt event to next one
  - Example: Measurements with periodic time intervals
    - Path measurements to detect whether an object is being accelerated, e.g. counting of step pulses on an incremental position encoder for a motor
    - Works only with a constant time interval

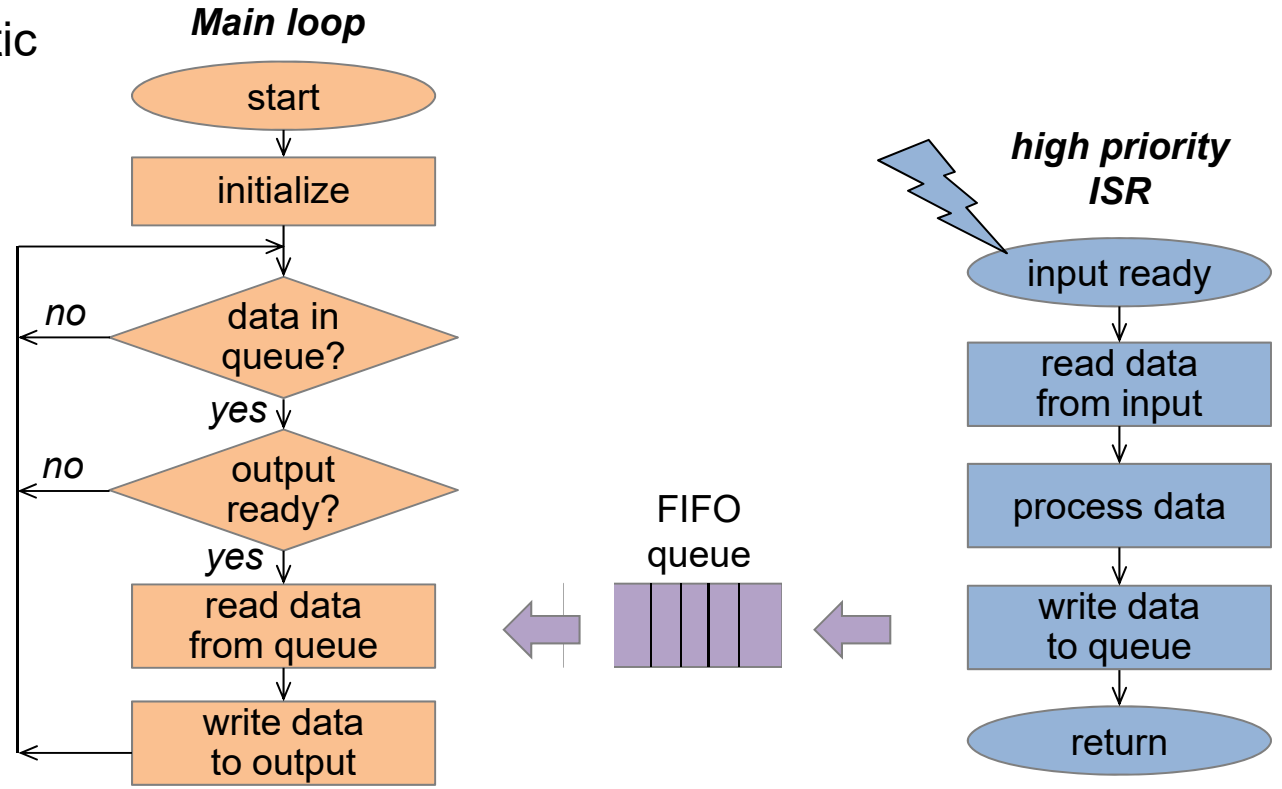
Managing Latency

- **High priority interrupts may cause high latencies for lower priority interrupts**
  - Example for high priority ISR
    - Interrupt triggers reading of input data
    - Process data
    - Write to output
  - Writing may have to wait for output device to become available
    - E.g. because SPI is still transmitting previous data



Managing Latency

- **Remedy: Move 'waiting loop' to main program**
  - Use queue (FIFO) for decoupling
  - Makes input ISR short and deterministic



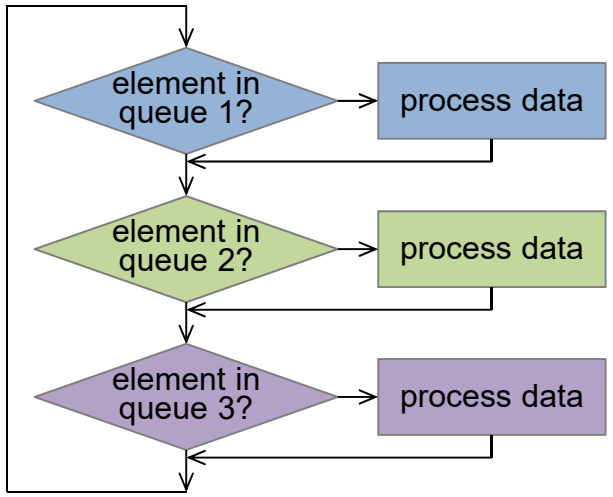
Example for functions to enter and retrieve data from a queue

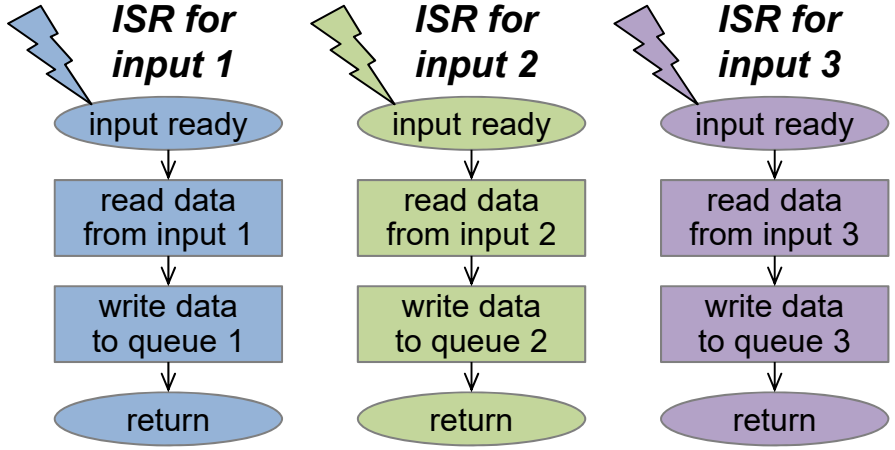
```
/**
 *  \brief  Enqueues data at the tail of the specified queue
 *  \param  queue   Points to the queue to which the element should be added
 *  \param  data
 *  \return 0 if data NOT added (queue is full),
 *          ~0 if data added (queue not full)
 */
uint32_t queue_enqueue(queue_t *queue, uint32_t data);

/**
 *  \brief  Grabs and removes the element at the head of the specified queue
 *  \param  queue   Points to the queue from which the element should be polled
 *  \return element at the head of queue, returns 0 if no element in queue.
 */
uint32_t queue_dequeue(queue_t *queue);
```

Managing Latency

School of Engineering
InES Institute of Embedded Systems

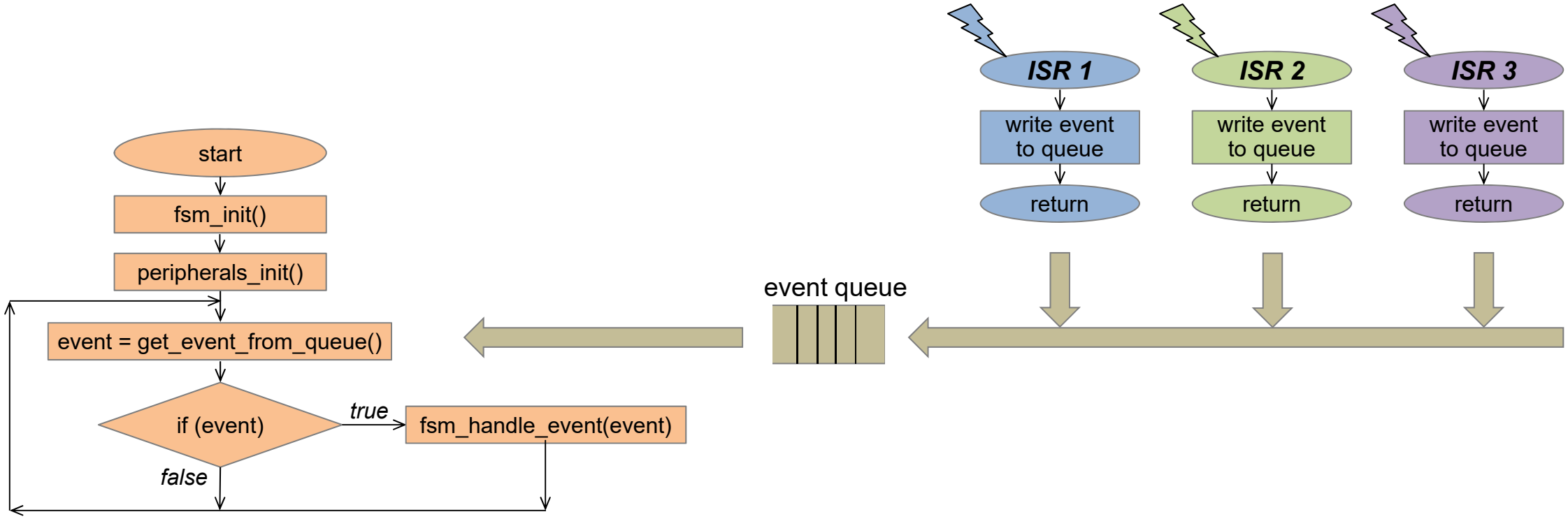
- **Move non-time-critical work from ISRs to main loop**
  - Several ISRs write input data into their dedicated queues
  - Main loop checks all queues and processes their contents
  - May result in many tasks in main loop
    - Not all processing tasks have the same priority
    - Requires scheduling of tasks
    - May require real-time OS in case of many ISRs





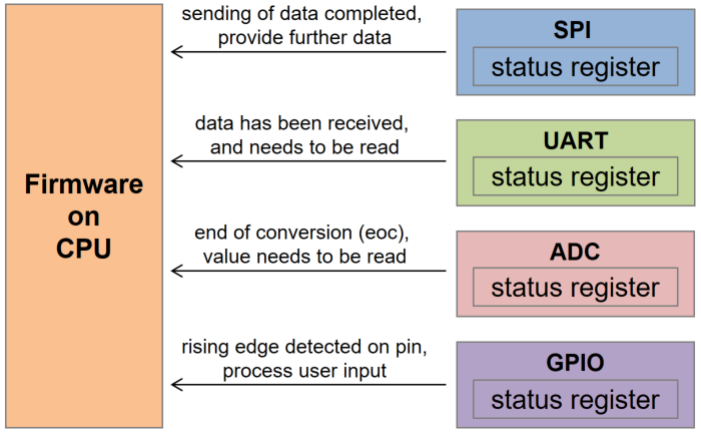
Interrupt Driven FSM

- **Events Trigger Interrupts**
  - Several Interrupt Service Routines (ISR) enter events into queue
  - FSM in main loop reads events from queue



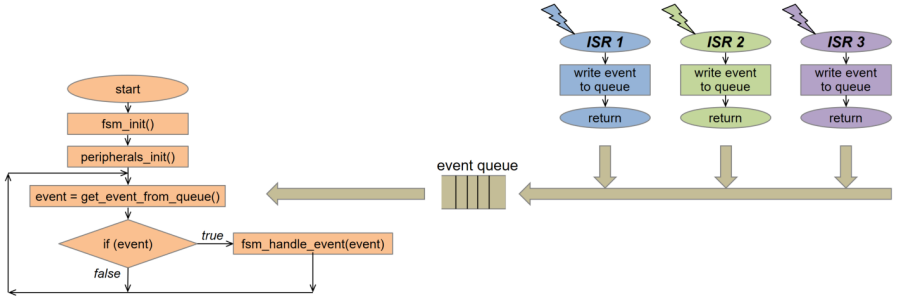
Conclusions

- **Detection of Events → Polling vs. Interrupt Driven I/O**
- **Interrupt Performance**
  - Interrupt frequency f_{INT}
  - Interrupt service time t_{ISR}
  - Impact = f_{INT} * t_{ISR} * 100%



- **Interrupt Latency**
  - How fast does an application require a response?
  - Pre-emption: High priority interrupts may cause high latencies for lower priority interrupt
  - Move ‘waiting loops’ and non-time-critical work from ISR to main loop

- **Interrupt Driven FSM**
  - Interrupt Service Routines (ISR) enter events into queue
  - FSM in main loop reads events from queue



Literature

School of Engineering
InES Institute of Embedded Systems

- **Interrupts In General**
  - "The Art of Assembly Language Programming"
    - by Randall Hyde, Chapter 17
  - "Fundamentals of Embedded Software with the ARM Cortex-M3"
    - by Daniel W. Lewis, Chapter 9

- **Interrupt Latency**
  - http://www.segger.com/interrupt-latency.html
  - http://www.ganssle.com/articles/interruptlatency.htm