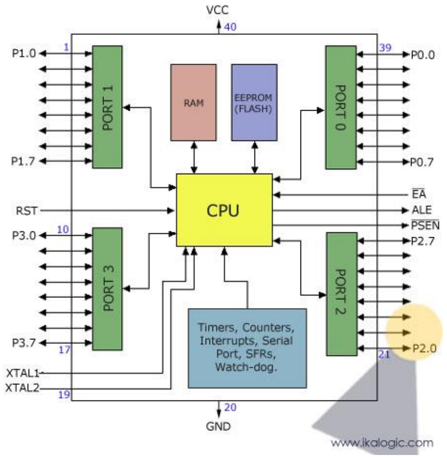
Embedded Systems Design, Spring 2025

Lecture 7



Interrupts

Outline

- Introduction to interrupts
- Types
- Examples
- Considerations

What are interrupts

- As the name implies, an **interrupt** is some event which interrupts normal program execution.
- Program flow is always sequential, but jumps from one place to another are permitted if special instructions are used (functions, conditions etc.)
- What interrupts do is that they allow to put the execution of the current program “on hold”, execute a special task, and then resume the current program as the interrupt never happened
- This task, or subroutine, called an interrupt handler, is only executed when a certain event (interrupt) occurs.

Different types of interrupts

- The **event** may be one of the timers "overflowing", receiving a character via the serial port, transmitting a character via the serial port, or one of two "external events".
    - Timer/Counter 0 Overflow.
    - Timer/Counter 1 Overflow.
    - Reception/Transmission of Serial Character.
    - External Event 0.
    - External Event 1.
- Any of these events may happen at one particular time, but we do not know **when** or **if** it will happen.
- The special function (**interrupt handler**) should do something related to the event that occurred.
- There are many other interrupt sources and possibilities for triggering

Hardware Interrupts

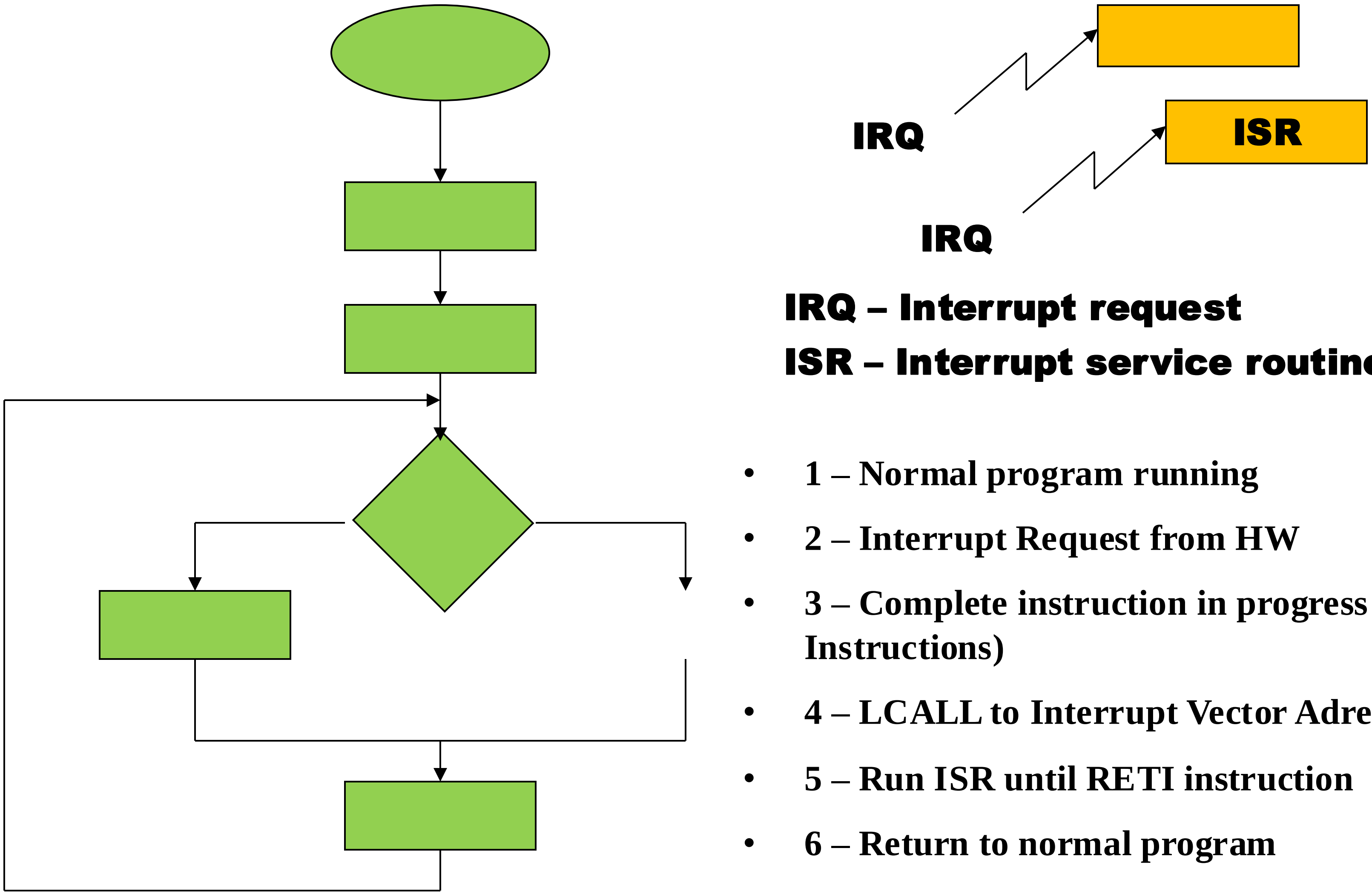
- The interrupts in a controller are generated by hardware (internal modules, external signals)
- If the interrupts are generated by the controller's inbuilt devices, like timer interrupts; or by the interfaced devices, they are called: **hardware interrupts**.

What happens in short?

- Program runs
- External event occurs
- Program halts and the special routine is executed
- Program is resumed as nothing happened

- Plenty of advantages!
- How about disadvantages?

How everything works in detail



IRQ – Interrupt request

ISR – Interrupt service routine

- 1 – Normal program running
- 2 – Interrupt Request from HW
- 3 – Complete instruction in progress (Machine Instructions)
- 4 – LCALL to Interrupt Vector Adress
- 5 – Run ISR until RETI instruction
- 6 – Return to normal program

Setting up interrupts

- By default at powerup, all interrupts are disabled.
- You must specifically tell in your program that you want to enable interrupts and, specifically, which interrupts you want serviced
- To enable interrupts:
  - `sei(); // will enable interrupts in STATUS_ REGISTER`

  `//by setting Global Interrupt Enable bit (I) to 1`
- To disable interrupts:
  - `cli(); // will disable interrupts in STATUS_ REGISTER`

  `//by clearing Global Interrupt Enable bit (I) to 0`

What about the ISR?

- You need to specify the microcontroller what to do when an interrupt occurs.

- This is done by writing a subroutine or function for the interrupt.

- This is the ISR and gets automatically called when an interrupt occurs.
- <u>It is not required to call the Interrupt Subroutine explicitly in the code</u>.

What happens when interrupts occur?

- The current Program Counter is saved on the stack
- All other interrupts are disabled. (can be enabled though)
- **The corresponding interrupt flag is cleared.** *
- Program execution transfers to the corresponding interrupt handler vector address.
- The Interrupt Handler Routine/ISR executes.
- Returns from Interrupt back to the main program, where at least one instruction executes before jumping to another interrupt
- * the microcontroller automatically clears the interrupt flag before passing control to your interrupt handler routine.

Interrupt vectors

Table 12-6. Reset and Interrupt Vectors in ATmega328 and ATmega328P

VectorNo.	Program Address[2]	Source	Interrupt Definition
1	0x0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B

Interrupt vectors - continued

14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

RESET

- Depending if a bootloader is present or not, the first thing the program will do after reset
  - Jump to the bootloader part (to reprogram the flash memory), then jumping to the main code
  - Jump to the main code of your application

External Interrupts

- INT0 / INT1 *(PD2 / PD3)*
    - External Interrupt Request 0 / 1
  - External Interrupts triggered by:
    - Falling edge
    - Rising edge
    - Low level (as long as the level is low)

- PCINT2 / PCINT1 / PCINT0
    - Pin change Interrupt Request 2 / 1 / 0

- Can also be used to wake up device in sleep mode (some timing considerations must be met)

INT0 / INT1 Registers

- EICRA – External Interrupt Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Configuring **INT1**, bits [3:2]:

Table 13-1. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

- Configuring **INT0**, bits [1:0]

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

INT0 / INT1 Registers

- EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **1**
  - Enable External Interrupt1
- **0**
  - Disable External Interrupt1

- **1**
  - Enable External Interrupt0
- **0**
  - Disable External Interrupt0

INT0 / INT1 Registers

- EIFR – External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	–	–	–	–	–	–	INTF1	INTF0	EIFR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Any logic change or edge presence : bit gets set to 1 automatically.**
  - If the interrupt for corresponding pin is enabled, the microcontroller will jump and execute the corresponding Interrupt Vector
  - The flag will be automatically cleared in the interrupt handler
  - Can be manually cleared by writing 1 to it.

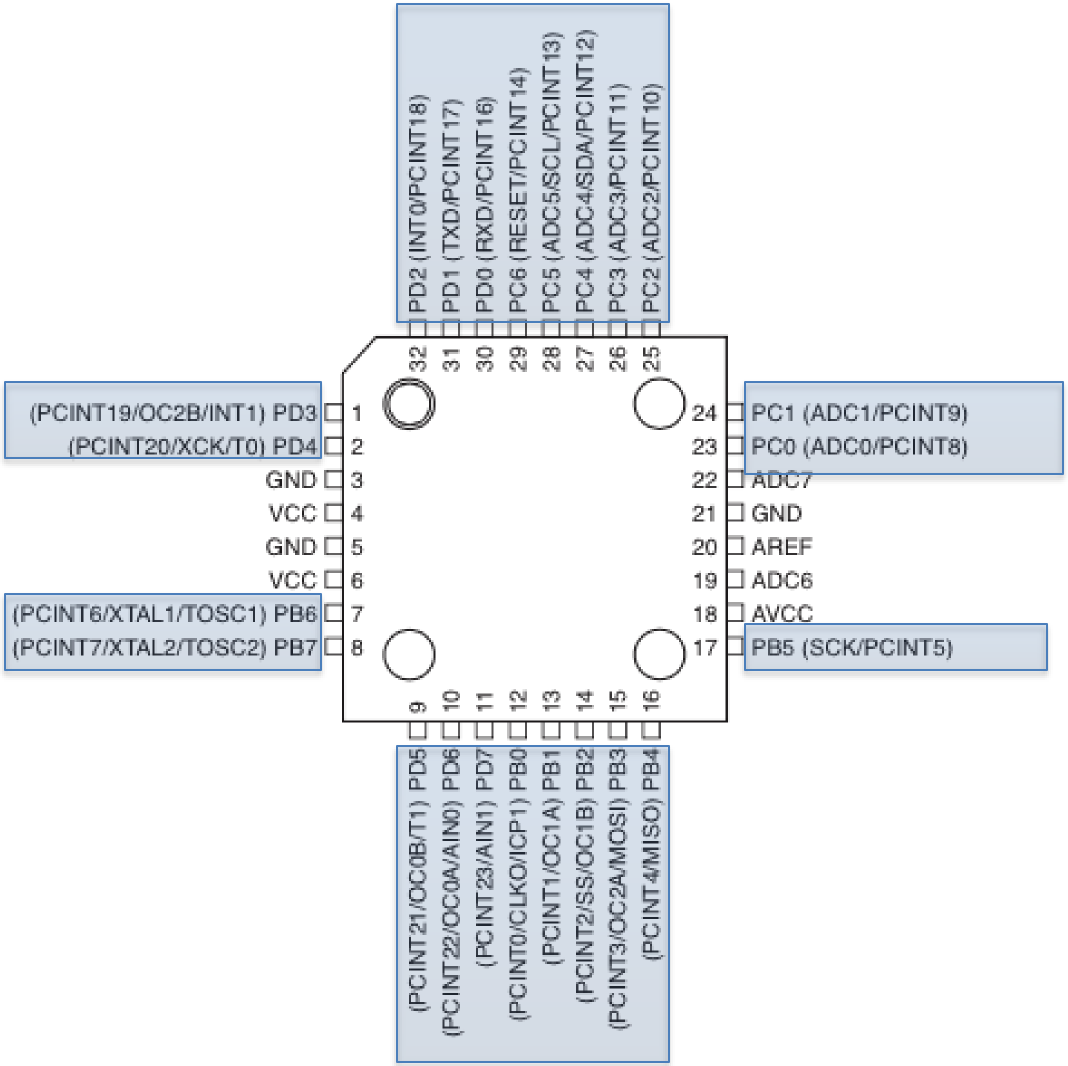
Example

```
// this code sets up an interrupt triggered by a change on the INT0 pin.

#include <avr/io.h>
#include <avr/interrupt.h>

int main(void)
{
        DDRD &= ~(1 << DDD2);      // Clear the PD2 pin
        // PD2 (INT0 pin) is now an input
        PORTD |= (1 << PORTD2);    // turn On the Pull-up
        // PD2 is now an input with pull-up enabled (MIGHT NOT BE NEEDED)

        EICRA |= (1 << ISC00);     // set INT0 to trigger on ANY logic change
        EIMSK |= (1 << INT0);      // Turns on interrupt for INT0
        sei();                     // turn on interrupts

        while (1)
        {
                //main loop - nothing happens
        }
}
//… continues on next page
```

Example - continued

```
//… continues from previous page

ISR (INT0_vect)  // external interrupt 0
{
    //event to be executed when a logic change / edge on
INT0 pin occurs
}
```

Pin Change Interrupts



PCI Registers

- PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Pin Change Interrupt Enable :**

- enables any logic change on one of the pins PCINT[23:16] to cause an interrupt
- enables any logic change on one of the pins PCINT[14:8] to cause an interrupt
- enables any logic change on one of the pins PCINT[7:0] to cause an interrupt

PCI Registers

- PCIFR – Pin Change Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1B (0x3B)	–	–	–	–	–	PCIF2	PCIF1	PCIF0	PCIFR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Any logic change or edge presence : this bit gets set to 1 automatically.**
  - If the interrupt for any of the corresponding group of pins is enabled PCINT[23:16], PCINT[14:8], PCINT[7:0] the microcontroller will jump and execute the corresponding Interrupt Vector
  - The flag will be automatically cleared in the interrupt handler
  - Can be manually cleared by writing 1 to it.

PCI Registers

PCMSK2 – Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PCMSK1 – Pin Change Mask Register 1

Bit	7	6	5	4	3	2	1	0	
(0x6C)	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Each bit from these 3 registers selects whether a logic level change on the selected pin will trigger an interrupt. Set to 1 for enabling the corresponding pin.**

Example

```
// this code sets up an interrupt triggered by a level change on PCINT0
#include <avr/io.h>
#include <avr/interrupt.h>

int main(void)
{
    DDRB &= ~(1 << DDB0);      // Clear the PB0 pin
    // PB0 (PCINT0 pin) is now an input
    PORTB |= (1 << PORTB0);     // turn On the Pull-up
    // PB0 is now an input with pull-up enabled (MIGHT NOT BE NEEDED)

    PCICR |= (1 << PCIE0);     // set PCIE0 to enable the group for PCINT7..PCINT0
    PCMSK0 |= (1 << PCINT0);   // Enable only PCINT0 interrupt from the group
    sei();                     // turn on interrupts
    while(1)
    {
        //main loop
    }
}
// … continues on next page
```

Example - continued

```
//… continues from previous page

ISR (PCINT0_vect)  // pin change in group 0 interrupt routine
{
        //event to be executed when PCINT0 occurred
        // to find out what value triggered the interrupt
        // check value of PINB
if( (PINB & (1 << PINB0)) == 1 )
    {
        /* LOW to HIGH pin change */
    }
    else
    {
        /* HIGH to LOW pin change */
    }
}
```

Timer/Counter 0 - Registers

TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Logical 1 will enable corresponding interrupts:

- Timer/Counter 0 Compare Match B Interrupt Enable
- Timer/Counter 0 Compare Match A Interrupt Enable
- Timer/Counter 0 OverflowInterrupt Enable

Timer/Counter 0 - Registers

TIFR0 – Timer/Counter 0 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x15 (0x35)	–	–	–	–	–	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- These bits will be set when a compare match with B, compare match with A or timer overflow will occur.

- These flags will be cleared when executing the corresponding handler vector
- Alternatively, can be cleared by writing a 1 to this flag

Example

```
// this code sets up a timer0 for 4ms @ 16Mhz clock cycle
// an interrupt is triggered each time the interval occurs.

#include <avr/io.h>
#include <avr/interrupt.h>

int main(void)
{
    // Set the Timer Mode to CTC
    TCCR0A |= (1 << WGM01);
    // Set the value that you want to count to
    OCR0A = 0xF9;
    TIMSK0 |= (1 << OCIE0A);    //Set the ISR COMPA vect
    sei();         //enable interrupts
    TCCR0B |= (1 << CS02);
    // set prescaler to 256 and start the timer
    while (1)
    {
        //main loop
    }
}
// … continues on next page
```

Example - continued

```
//… continues from previous page

ISR (TIMER0_COMPA_vect)  // timer0 overflow interrupt
{
    //event to be executed every 4ms here
}
```

Passing variables

- Only way to pass variables to interrupts is by using global variables
- It is important to use the keyword **volatile** (so they can be updated by other processes than your program, even by hardware interrupts)
- Use the variable in the ISR as any other variable;

```
volatile int my_variable;
int main(){
    …
}
```

What else? Priorities

- The interrupts are serviced according to priorities.
- An interrupt with a higher priority will be serviced first.
- The highest priority: 0 - RESET
- Lowest priority: 26 – STORE PROGRAM MEMORY READY

Some considerations

- Register protection?
  - When in the main program and working with various registers, it is important to protect their information from modification in the ISR, otherwise it is possible to have data corruption.

- Actions taken in the ISR should be FAST!
- printf and other slow operations must not be used in ISR!