Interrupts

Interrupts are reserved addresses on the memory for a specific task. When the condition of an interrupt is met, the processor stops whatever it was doing and jumps to the interrupts subroutine address.

1. External Interrupts

External interrupts are used to make subroutines that are triggered for our own specific conditions. There are 8 external interrupt addresses, that we can use for our purpose INT0 - INT7.

```
Address   Labels   Code                 Comments
0x0000             jmp    RESET         ; Reset Handler
0x0002             jmp    INT0          ; IRQ0 Handler
0x0004             jmp    INT1          ; IRQ1 Handler
0x0006             jmp    INT2          ; IRQ2 Handler
0x0008             jmp    INT3          ; IRQ3 Handler
0x000A             jmp    INT4          ; IRQ4 Handler
0x000C             jmp    INT5          ; IRQ5 Handler
0x000E             jmp    INT6          ; IRQ6 Handler
0x0010             jmp    INT7          ; IRQ7 Handler
0x0012             jmp    PCINT0        ; PCINT0 Handler
0x0014             jmp    PCINT1        ; PCINT1 Handler
0x0016             jmp    PCINT2        ; PCINT2 Handler
0x0018             jmp    WDT           ; Watchdog Timeout Handler
0x001A             jmp    TIM2_COMPA    ; Timer2 CompareA Handler
0x001C             jmp    TIM2_COMPB    ; Timer2 CompareB Handler
0x001E             jmp    TIM2_OVF      ; Timer2 Overflow Handler
0x0020             jmp    TIM1_CAPT     ; Timer1 Capture Handler
0x0022             jmp    TIM1_COMPA    ; Timer1 CompareA Handler
0x0024             jmp    TIM1_COMPB    ; Timer1 CompareB Handler
0x0026             jmp    TIM1_COMPC    ; Timer1 CompareC Handler
0x0028             jmp    TIM1_OVF      ; Timer1 Overflow Handler
0x002A             jmp    TIM0_COMPA    ; Timer0 CompareA Handler
0x002C             jmp    TIM0_COMPB    ; Timer0 CompareB Handler
0x002E             jmp    TIM0_OVF      ; Timer0 Overflow Handler
0x0030             jmp    SPI_STC       ; SPI Transfer Complete Handler
0x0032             jmp    USART0_RXC    ; USART0 RX Complete Handler
0x0034             jmp    USART0_UDRE   ; USART0,UDR Empty Handler
0x0036             jmp    USART0_TXC    ; USART0 TX Complete Handler
0x0038             jmp    ANA_COMP      ; Analog Comparator Handler
0x003A             jmp    ADC           ; ADC Conversion Complete Handler
0x003C             jmp    EE_RDY        ; EEPROM Ready Handler
0x003E             jmp    TIM3_CAPT     ; Timer3 Capture Handler
```

```
0x0040          jmp     TIM3_COMPA          ; Timer3 CompareA Handler
0x0042          jmp     TIM3_COMPB          ; Timer3 CompareB Handler
0x0044          jmp     TIM3_COMPC          ; Timer3 CompareC Handler
0x0046          jmp     TIM3_OVF            ; Timer3 Overflow Handler
0x0048          jmp     USART1_RXC          ; USART1 RX Complete Handler
0x004A          jmp     USART1_UDRE         ; USART1,UDR Empty Handler
0x004C          jmp     USART1_TXC          ; USART1 TX Complete Handler
0x004E          jmp     TWI                 ; 2-wire Serial Handler
0x0050          jmp     SPM_RDY             ; SPM Ready Handler
0x0052          jmp     TIM4_CAPT           ; Timer4 Capture Handler
0x0054          jmp     TIM4_COMPA          ; Timer4 CompareA Handler
0x0056          jmp     TIM4_COMPB          ; Timer4 CompareB Handler
0x0058          jmp     TIM4_COMPC          ; Timer4 CompareC Handler
0x005A          jmp     TIM4_OVF            ; Timer4 Overflow Handler
0x005C          jmp     TIM5_CAPT           ; Timer5 Capture Handler
0x005E          jmp     TIM5_COMPA          ; Timer5 CompareA Handler
0x0060          jmp     TIM5_COMPB          ; Timer5 CompareB Handler
0x0062          jmp     TIM5_COMPC          ; Timer5 CompareC Handler
0x0064          jmp     TIM5_OVF            ; Timer5 Overflow Handler
0x0066          jmp     USART2_RXC          ; USART2 RX Complete Handler
0x0068          jmp     USART2_UDRE         ; USART2,UDR Empty Handler
0x006A          jmp     USART2_TXC          ; USART2 TX Complete Handler
0x006C          jmp     USART3_RXC          ; USART3 RX Complete Handler
0x006E          jmp     USART3_UDRE         ; USART3,UDR Empty Handler
0x0070          jmp     USART3_TXC          ; USART3 TX Complete Handler
;
0x0072  RESET:  ldi     r16, high(RAMEND)   ; Main program start
0x0073          out     SPH,r16             ; Set Stack Pointer to top of RAM
0x0074          ldi     r16, low(RAMEND)
0x0075          out     SPL,r16
0x0076          sei                         ; Enable interrupts
0x0077          <instr> xxx
...     ...     ...     ...
```

To be able to use this interrupt, the following steps are needed.

- First, we have to change the reset jump to go to address 0x72, in order not to override the already defined memory addresses.

```
.org 0x00
rjmp start

.org INT0addr                   ; INT1 interrupaddress
rjmp interrupt_0
.org INT1addr
rjmp interrupt_1

.org 0x72
start:
; Inintialize SP, Stack Pointer
ldi r20, HIGH(RAMEND)    ; R20 = high part of RAMEND address
out SPH,R20              ; SPH = high part of RAMEND address
ldi R20, low(RAMEND)     ; R20 = low part of RAMEND address
out SPL,R20              ; SPL = low part of RAMEND address
```

- Then we have to enable the interrupt we want to use from the EIMSK register. EIMSK is an internal register that the processor uses as a flag to enable and disable the 8 interrupts. In the following picture, we enable INT0 and INT1

```
ldi r16, 0b00000011      ; INT0 and INT1 enable
out EIMSK, r16
```

- Next, We define how we want to use the interrupt in the EICRA. For example, if we choose 10 (falling edge), the interrupt is triggered when the switch is pressed. While if we choose 11 (rising edge), the interrupt is triggered when the switch is released.

Table 15-1. Interrupt Sense Control[1]

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request
0	1	Any edge of INTn generates asynchronously an interrupt request
1	0	The falling edge o. INTn generates asynchronously an interrupt request
1	1	The rising edge of INTn generates asynchronously an interrupt request

Note: 1. n = 3, 2, 1or 0

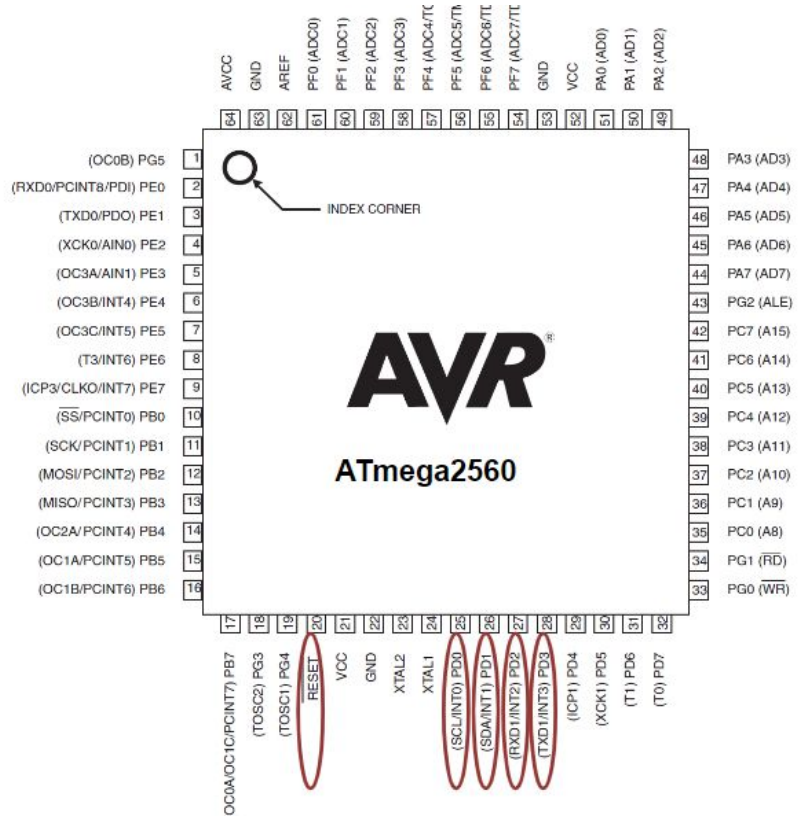
In the following picture, we are using 00 for the INT0 and 10 for INT1.

```
ldi r16, 0b00001000      ; INT1 falling edge, INT0 rising edge
sts EICRA, r16
```

- Finally, we enable the global interrupt flag - now we can use the interrupt we created.

```
sei                      ; Global interrupt enable
```

Note: all interrupts are connected to a specific i/o registers, so before using the interrupts, we have to find the port pin number the interrupt is connected to. You can use the picture to map the i/o port to the corresponding interrupt.



2. Timers

Timers are also internal interrupts that we can use to schedule a task depending on time. You can follow the following steps to create a timer interrupt.

1. First, we have to use the predefined address for the timer interrupts. We do this as follows.

```
.equ    OC2Aaddr        = 0x001a        ; Timer/Counter2 Compare Match A
.equ    OC2Baddr        = 0x001c        ; Timer/Counter2 Compare Match B
.equ    OVF2addr        = 0x001e        ; Timer/Counter2 Overflow
.equ    ICP1addr        = 0x0020        ; Timer/Counter1 Capture Event
.equ    OC1Aaddr        = 0x0022        ; Timer/Counter1 Compare Match A
.equ    OC1Baddr        = 0x0024        ; Timer/Counter1 Compare Match B
.equ    OC1Caddr        = 0x0026        ; Timer/Counter1 Compare Match C
.equ    OVF1addr        = 0x0028        ; Timer/Counter1 Overflow
.equ    OC0Aaddr        = 0x002a        ; Timer/Counter0 Compare Match A
.equ    OC0Baddr        = 0x002c        ; Timer/Counter0 Compare Match B
.equ    OVF0addr        = 0x002e        ; Timer/Counter0 Overflow
```

```
.include "m2560def.inc"

.CSEG                   ; Assembly directive Code Segment
.ORG 0x0000             ; place this code in PM address 0x0000
rjmp start              ; jump to label start

.ORG OVF0addr
jmp timer0_int          ; timer interrupt service routine
```

2. second, we have to set the Prescaler register. The use of this step is to choose the time the processor executes every command. We do that by using the TIMER registers. On the picture below we are setting the TIMER register to be 101, which will make the processor executed every command approximately in 1ms(1024).

TIMER0 – TCCR0

I

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	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	

- **Bit 2:0 – CS02:0: Clock Select**

The three Clock Select bits select the clock source to be used by the Timer/Counter.

Table 42. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

```
;ldi temp, 0x05          ; prescaler value to TCCR0 / 1024
out TCCR0B, temp         ; CS2 - CS2 = 101, osc.clock / 1024
```

3. Then we have to enable the time interrupts flag on TIMSK0 register.

```
ldi temp, (1<<TOIE0)     ; Timer 0 enable flag, TOIE0
sts TIMSK0, temp         ; to register TIMSK
```

4. Finally, as we always do when using interrupts we enable the global interrupt flag.

```
sei                     ; enable global interrupt
```

Serial communication (USART)

UART or USART is a standard I/O device

•The ATxmega128A1 has 8 independent USART integrated on-chip, producing logic signals for RS232 communication

Transmit

› Manages stream of bits for each byte

•Receive

› Manages receipt of bits and assembly into byte

•Clock Generator

› Allows the USART to operate in synchronous or asynchronous modes

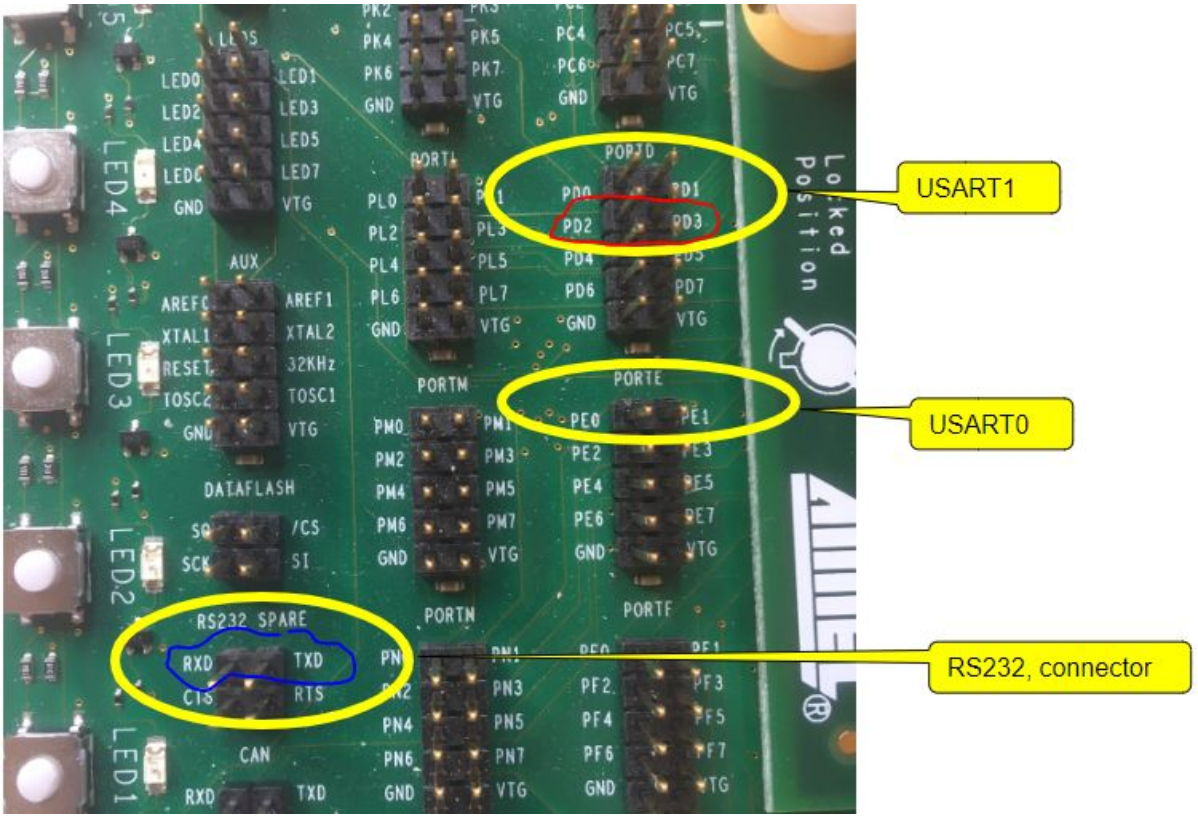
Physical Connections

•The USART utilizes two external pins on the microcontroller for transmitting and receiving

›TXD0 is PortD pin2

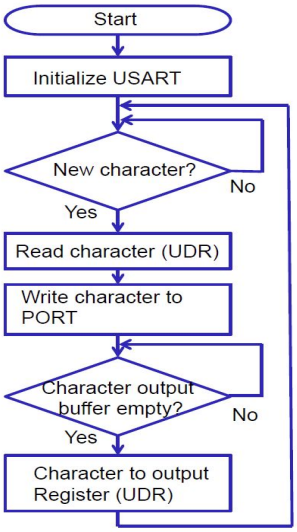
›RXD0 is PortD pin3

•These bits of PORTD cannot be utilized for general digital I/O while serial communication is taking place



How to use USART on the code?

The flowchart below explains the steps we are going to take.



1. On our main program we enable TX (transmitter) and the RX (receiver) flags on, and set the prescale value to be 12 or 25, which will give us 4800 bps or 2400 bps (see the old lecture 7 for more).

```
.include "m2560def.inc"

.def Temp = r17
.def char = r16
.equ UBRR_val = 12          ; osc.=1MHz, 4800 bps => UBBRR = 12

.org 0x00
    rjmp start

.org 0x30
start:
    ldi Temp, 0xFF          ; PORTB outputs
    out DDRB, Temp
    ldi Temp, 0X55          ; Initial value to outputs
    out PORTB, Temp

    ldi Temp, UBRR_val      ; store Prescaler value in UBRR1L
    sts UBRR1L, Temp

    ldi Temp, (1<<TXEN1) | (1<<RXEN1)
    sts UCSR1B, Temp        ; set TX and RX enable flags
```

STK600

2. To start receiving from the RS232, we use the following subroutine. The character we received will be stored in the Char register(r16).

```
GetChar:                    ; receive data

    lds Temp, UCSR1A        ; read UCSR1A I/O register to r20
    sbrs Temp, RXC1         ; RXC1=1 => new character
    rjmp GetChar            ; RXC1=0 => no character received
    lds Char, UDR1          ; read character in UDR
```

3. To start sending through the RS232, we use the following subroutine. The character we are going to send is stored in Char register(r16) before writing it to UDR1.

```
PutChar:                ; send data

    lds Temp, UCSR1A    ; read UCSR1A I/O register to r20
    sbrs Temp, UDRE1    ; UDRE1-1 -> buffer is empty
    rjmp PutChar        ; UDRE1-0 -> buffer is not empty
    sts UDR1, Char      ; write character to UDR1
    rjmp GetChar        ; return to loop
```

4. Finally, open the application PUTTY (you can download it from here), then select the serial checkbox in session and the correct baud rate (4800bs or 2400bs) then click open-this should start a session between the STK and putty terminal. Now, try typing on ur PUTTY terminal and see if you are receiving signals.