Lab 8: LCD Display and Interrupts

Submit the modified timer_interrupt.asm from your project at the end of your lab.

I. Interrupts

An interrupt is a signal that can interrupt and alter the flow of current program execution. It is a method that allows programs to respond immediately to external events. Interrupts can be triggered by external or internal signals; they can be caused by software or hardware. When a program is interrupted, the routine that is executed in response to the interrupt is called **Interrupt Service Routine (ISR)**, or **interrupt handler**. The process of interrupting (and pausing) the current program, executing the interrupt handler, and returning back to the interrupted program is called **servicing the interrupt**. Interrupts can be dedicated or shared. If interrupts are shared among multiple devices (I/O), then the ISR further has to check which device or signal caused that interrupt. Usually the processor has an interrupt vector (a set of bits), where each position in the vector corresponds to a specific interrupt that can occur. When interrupts are enabled, during each cycle and right before fetching the next instruction, the processor checks this vector to see if an interrupt occurred. It starts the check at the first position in the vector and finishes at the last position. This gives interrupts a natural ordering which determines their priority: first one in the vector will be the first one serviced. When an interrupt occurs, the processor jumps to a pre-defined location in the **interrupt table** assigned to that interrupt. Hence the interrupt table has to be setup to jump and branch to the appropriate ISR to handle each type of interrupt. For completeness, the ATMega2560 processor interrupt table from page 101 of the datasheet is included on the next two pages.

II. Timer 1 in the AVR ATmega2560

In this course, we will use the AVR timers to periodically interrupt our program so that we can perform certain actions based on time. There are 6 built-in timers (two 8-bit and four 16-bit timer/counters) in the AVR processor. They can be setup in different modes. For the purposes of this lab, we will only use the interrupts related to the **overflow** operation modes. Furthermore, in this lab, we will only use the 16-bit TIMER1. Before using it, we need to configure the timer using the relevant special purpose registers, listed below and explained further on:

TCCR1A = Timer/Counter 1 Control Register A
TCCR1B = Timer/Counter 1 Control Register B
TCCR1C = Timer/Counter 1 Control Register C
TIFR1 = Timer/Counter 1 Interrupt Flag Register
TIMSK1 = Timer/Counter 1 Interrupt Mask Register
TCNT1H = High byte of the Timer/Counter 1
TCNT1L = Low byte of the Timer/Counter 1
SREG = Status Register

Table 14-1. Reset and Interrupt Vectors

Vector No.	Program Address[2]	Source	Interrupt Definition
1	$0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	PCINT0	Pin Change Interrupt Request 0
11	$0014	PCINT1	Pin Change Interrupt Request 1
12	$0016[3]	PCINT2	Pin Change Interrupt Request 2
13	$0018	WDT	Watchdog Time-out Interrupt
14	$001A	TIMER2 COMPA	Timer/Counter2 Compare Match A
15	$001C	TIMER2 COMPB	Timer/Counter2 Compare Match B
16	$001E	TIMER2 OVF	Timer/Counter2 Overflow
17	$0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	$0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	$0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	$0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	$0028	TIMER1 OVF	Timer/Counter1 Overflow
22	$002A	TIMER0 COMPA	Timer/Counter0 Compare Match A
23	$002C	TIMER0 COMPB	Timer/Counter0 Compare match B
24	$002E	TIMER0 OVF	Timer/Counter0 Overflow
25	$0030	SPI, STC	SPI Serial Transfer Complete
26	$0032	USART0 RX	USART0 Rx Complete
27	$0034	USART0 UDRE	USART0 Data Register Empty
28	$0036	USART0 TX	USART0 Tx Complete
29	$0038	ANALOG COMP	Analog Comparator
30	$003A	ADC	ADC Conversion Complete

Table 14-1. Reset and Interrupt Vectors (Continued)

Vector No.	Program Address[2]	Source	Interrupt Definition
31	$003C	EE READY	EEPROM Ready
32	$003E	TIMER3 CAPT	Timer/Counter3 Capture Event
33	$0040	TIMER3 COMPA	Timer/Counter3 Compare Match A
34	$0042	TIMER3 COMPB	Timer/Counter3 Compare Match B
35	$0044	TIMER3 COMPC	Timer/Counter3 Compare Match C
36	$0046	TIMER3 OVF	Timer/Counter3 Overflow
37	$0048	USART1 RX	USART1 Rx Complete
38	$004A	USART1 UDRE	USART1 Data Register Empty
39	$004C	USART1 TX	USART1 Tx Complete
40	$004E	TWI	2-wire Serial Interface
41	$0050	SPM READY	Store Program Memory Ready
42	$0052[3]	TIMER4 CAPT	Timer/Counter4 Capture Event
43	$0054	TIMER4 COMPA	Timer/Counter4 Compare Match A
44	$0056	TIMER4 COMPB	Timer/Counter4 Compare Match B
45	$0058	TIMER4 COMPC	Timer/Counter4 Compare Match C
46	$005A	TIMER4 OVF	Timer/Counter4 Overflow
47	$005C[3]	TIMER5 CAPT	Timer/Counter5 Capture Event
48	$005E	TIMER5 COMPA	Timer/Counter5 Compare Match A
49	$0060	TIMER5 COMPB	Timer/Counter5 Compare Match B
50	$0062	TIMER5 COMPC	Timer/Counter5 Compare Match C
51	$0064	TIMER5 OVF	Timer/Counter5 Overflow
52	$0066[3]	USART2 RX	USART2 Rx Complete
53	$0068[3]	USART2 UDRE	USART2 Data Register Empty
54	$006A[3]	USART2 TX	USART2 Tx Complete
55	$006C[3]	USART3 RX	USART3 Rx Complete
56	$006E[3]	USART3 UDRE	USART3 Data Register Empty
57	$0070[3]	USART3 TX	USART3 Tx Complete

Notes:
1. When the BOOTRST Fuse is programmed, the device will jump to the Boot Loader address at reset, see "Memory Programming" on page 325.
2. When the IVSEL bit in MCUCR is set, Interrupt Vectors will be moved to the start of the Boot Flash Section. The address of each Interrupt Vector will then be the address in this table added to the start address of the Boot Flash Section.
3. Only available in ATmega640/1280/2560.

III. Timer-driven interrupt example with timer counter overflow.

We are going to use Time/Counter 1. This timer will be set to normal overflow mode. That is: the 16-bit timer/counter is initialized to a value (say 0) and a hardware clock signal is used to increment the timer/counter. When the timer/counter overflows, it generates interrupt 21 (see the above table) causing the processor to jump to a pre-defined location 0x0028 in program memory. Let's learn the structure of interrupt in AVR assembly language (see **timer_interrupt.asm**):

```
.org 0x0000
    jmp setup

.org 0x0028
    jmp timer1_ISR

.org 0x0072
```

Initialization of the **Interrupt Vector Table (IVT)**

…

```
setup:
    ; initialize the stack pointer
    ldi r16, high(RAMEND)
    out SPH, r16
    ldi r16, low(RAMEND)
    out SPL, r16
```

Stack Pointer must be initialized, since we're using functions.

…

```
timer1_setup:
    ; timer mode
    ldi r16, 0x00          ; normal operation
    sts TCCR1A, r16
```

Set up operation mode (norm.) in **Control Register A**

…

```
    ldi r16, (1<<CS12)|(1<<CS10)    ; clock / 1024
    sts TCCR1B, r16
```

Set up the pre-scaler value in **Control Register B**

…

```
    ; set timer counter to TIMER1_TOP (defin
    ldi r16, high(TIMER1_TOP)
    sts TCNT1H, r16      ; must WRITE high by
    ldi r16, low(TIMER1_TOP)
    sts TCNT1L, r16      ; low byte
```

Initialize starting value for the counter. It will overflow when the counter reaches 0xFFFF. Hence, this is a 16-bit integer stored in **two 8-bit registers**, which are **latched together** – both must be written together. Thus, the high byte must be written first, because the value of both will update only when low byte is written.

…

```
    ; allow timer to interrupt the CPU when it's counter overflows
    ldi r16, 1<<TOV1
    sts TIMSK1, r16
```

Allow Timer 1 to interrupt the CPU via the interrupt vector. in **Timer Mask Register**

…

```
    ; enable interrupts (the I bit in SREG)
    sei
```

Enable interrupts by setting the global interrupt flag in SREG

…

```
; timer interrupt flag is automatically
; cleared when this ISR is executed
; per page 168 ATmega datasheet
timer1_ISR:
    push r16
    push r17
    push r18
    lds r16, SREG
    push r16
```

Inside the ISR, remember to protect SREG along with the others that this function affects.

…

```
    ; RESET timer counter to TIMER1_TOP (defined above)
    ldi r16, high(TIMER1_TOP)
    sts TCNT1H, r16      ; must WRITE high byte first
    ldi r16, low(TIMER1_TOP)
    sts TCNT1L, r16      ; low byte
```

Inside the ISR, reset the Timer Counter register, since it’s currently at 0 (after overflow).

…

```
    pop r16
    sts SREG, r16
    pop r18
    pop r17
    pop r16
    reti
```

This instruction sets the PC to the top three bytes on stack, adjusts the stack pointer, and only then enables interrupts. This order is important and cannot be achieved otherwise.

Below are the explanations of how each of the Timer 1 registers are used in this example.

TCCR1A - Timer 1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
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	

TCCR1A is set to 0, which means in normal mode, disconnect Pin OC1 from Timer/Counter 1. The other modes are: "COM1A1:COM1A0": Compare Output Mode for Channel A, "WGM11:WGM10": Waveform Generation Mode. Only when one of the OC1A/B/C is connected to the pin, the function of the COM1x1:0 bits is dependent of the WGM13:0 bits setting. It doesn't apply to this example and the values are set to all 0's.

TCCR1B - Timer 1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 2:0 – CSn2:0: Clock Select**

The three clock select bits select the clock source to be used by the Timer/Counter, see Figure 17-10 and Figure 17-11 on page 152.

Table 17-6. Clock Select Bit Description

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (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 Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

If external pin modes are used for the Timer/Countern, transitions on the Tn pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

The least significant three bits of TCCR1B are used to slow down the interval in our example. Instead of counting 1 per clock cycle, we count 1 every 1024 clock cycles in this example.

TCCR1C - Timer 1 Control Register C

Bit	7	6	5	4	3	2	1	0	
(0x82)	FOC1A	FOC1B	FOC1C	–	–	–	–	–	TCCR1C
Read/Write	W	W	W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

This register is not explicitly initialized in this example.

TIMSK1 – Timer 1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	OCIE1C	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

This register is set to 0x01. "Bit 0 – TOIEn: Timer/Counter Overflow Interrupt Enable
When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter Overflow interrupt is enabled. The corresponding Interrupt Vector is executed by program control jumping to address 0x0028 and further jumping to timer1_isr.

TIFR1 – Timer 1 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	OCF1C	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

This register is not explicitly initialized in this example.

TCNT1 – Timer 1 Counter Register

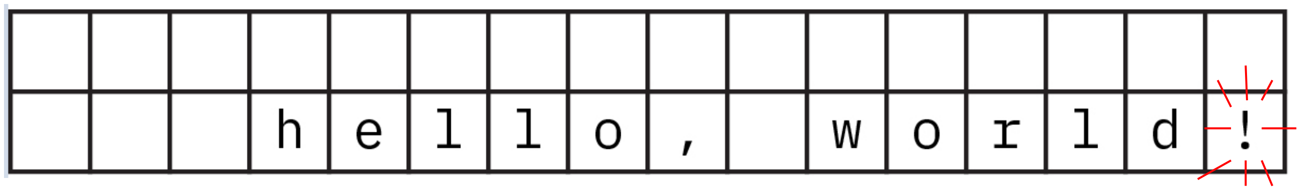
The TCNT1 is a 16-bit register and can be accessed by the AVR CPU via the 8-bit data bus. The 16-bit register must be byte accessed using two read or write operations. Each 16-bit timer has a single 8-bit register for temporary storing of the high byte of the 16-bit access. Accessing the low byte triggers the 16-bit read or write operation. When the low byte of a 16-bit register is written by the CPU, the high byte stored in the Temporary Register, and the low byte written are both copied into the 16-bit register in the same clock cycle. When the low byte of a 16-bit register is read by the CPU, the high byte of the 16-bit register is copied into the Temporary Register in the same clock cycle as the low byte is read.

IV. Exercise 1.

Download timer_interrupt.asm and complete the ISR implementation to toggle the two bits on PORTB that drive the LEDs to make the two LEDs blink. This can be achieved in a few lines of code using a masking operation. First retrieve the current PORTB values, then apply the mask and store the result back to PORTB. When your code is implemented correctly, you will see the LEDs on PORTB behave similarly to those on PORTL.
Note that the LEDs on both ports start in sync with each other but over time the LEDs on one of the ports start to lag behind those on the other port. Questions to consider: can you make them synchronized? which way is harder to achieve perfect timing – adjusting the delay loop or adjusting the timer configuration? what affects the timing of the delay loop? what affects the timing of the interrupts? **Note that changing the number of instructions in the ISR affects the delay loop, because the ISR can interrupt the delay loop and thus cause a delay in its execution!!**

V. Exercise 2.

Modify the program from Exercise 1 to display “hello, world!” on the LCD, using the LCD library provided during the last lab. Then, using the timer-driven interrupts, make the exclamation mark on the LCD display blink at a certain rate.



Play around with the timing of the blinking and make it blink once per second, twice per second, 10 Hz, 100 Hz, 1000 Hz. What happens when the blinking is too frequent? You may use the online AVR timer calculator (https://eleccelerator.com/avr-timer-calculator/) to figure out the appropriate settings for TIMER1 starting value and pre-scaler.

Submit the modified timer_interrupt.asm from your project at the end of your lab.