HLAB08
Interrupt Systems for UART and Timer

This lab will introduce the Atmega328 Interrupt systems and its use for a UART and a timer. An interrupt is a hardware-generated signal typically from peripheral devices. A special program called Interrupt Service Routine (ISR) also called Interrupt Handler, is evoked to check and process the request. Atmega328 microprocessor provides support for 26 different interrupt sources as listed in Table 16.1 of the datasheet. In this lab, we will investigate some of the timer and UART interrupts.

1. Aims

- To understand the interrupt-based UART operation and to implement programs of interrupt-based UART transmit and receive using ring-buffers.
- To understand how a timer works in a CTC (Clear timer on compare match) mode and to implement a code which can set the LED blinking frequency from the serial.

2. Interrupt based UART Operations

As you might have noticed in the previous lab, the polling-method is not efficient as the program should continuously check for the relevant event to happen, wasting the invaluable CPU time. The interrupt system is the mechanism to handle this matter, enabling a background processing. For example, if a serial frame arrives on the UART, the receiver hardware generates an interrupt signal, which evokes a special function called an "*Interrupt Service Routine (ISR)"*, to process the requests. To use the UART interrupts, we have to set 2 related bits in the UCSR0B register and the global SREG register.

- UDRIE0-bit – Enables the UART TX Data Register Empty (UDRE) Interrupt.
- RXCIE0-bit– Enables the UART RX Complete (RXC) Interrupt.
- SREG (Status Register) – The Interrupt-bit enables or disables the global interrupt of the Atmega328. sei() (set enable interrupt) or cli()(clear interrupt) are used for this purpose.

The last step is to write the ISR to handle the events. The general guideline on writing ISRs is to make them as short as possible and do not use *delays*. Also, you should spell and capitalize the ISR names correctly – they are reserved words as listed in Table 16.1 of the datasheet. The codes below are the ISRs for UDRE, and RXC interrupts. Note that there is a Transmit Complete (TXC) interrupt as well which is used in a half-duplexing mode (the TX line should be converted to the RX line) and thus will not be used here.

```
ISR(USART_UDRE_vect)
{
    // Write a byte to UDR0
}

ISR(USART_RX_vect)
{
    // Read a byte from UDR0
}
```

A final note is if there are any shared variables changed within the ISR, the variables should be declared as the *volatile* type which tells the compiler not to use caching or optimisation.

To utilise the interrupted based operation, a buffer is typically required between the user program and the ISR. A ring-buffer (that is if the buffer index reaches its max size it simply resets to zero) will be used in this lab. Figure 1 illustrates an example of a ring buffer for transmitting data. The user program starts writing the data in the buffer incrementing the head index. Once it writes the last byte, it should set the UDRIE0-bit to activate the TX interrupt. The ISR will then be evoked, checking whether there are data in the buffer. If so, it fetches a byte at the tail index, incrementing the tail position. Once it fetches the last byte, it should disable the TX interrupt by clearing the UDRIE0-bit. One tricky condition is when the buffer becomes full (for example the TX can be slower than writing to the buffer). In this lab we will discard the last byte by incrementing the tail index.

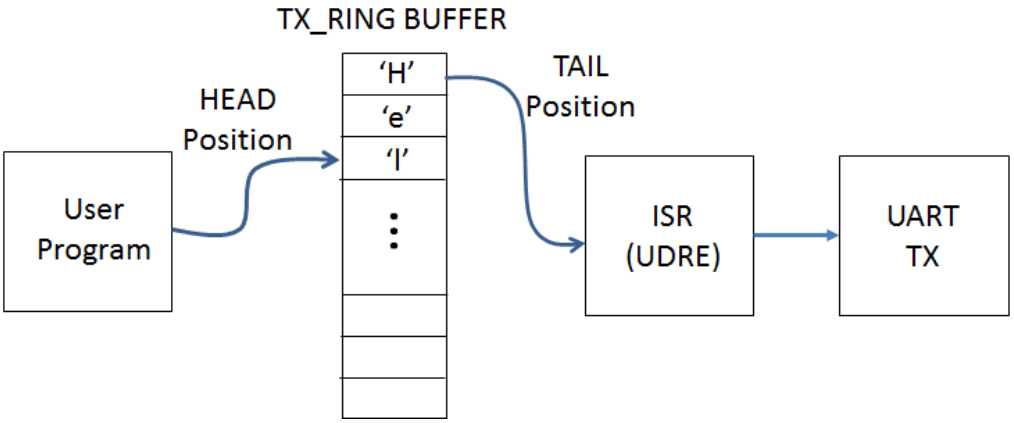

Figure 1 A ring-buffer for interrupted based UART transmit.

- Download and run two sample codes from Wattle
  - *6_uart_interrupt_echo.c*
  - *7_uart_interrupt_tx_buffered.c*
- Add comments if necessary and complete the relevant codes so that the codes run as intended. Check the comments within the code. Note that to use the transmit-interrupt, we need a ring-buffer. Demonstrate the results to the tutors.
- Using this knowledge, implement an interrupt-version code of *uart_rx_str.c* in HLAB07. That is the code should receive a line of character data (finished by an [enter] key or '\r') from the Putty terminal using the RXC interrupt and a ring buffer. Demonstrate the system to the tutors to get marks.

3. Interrupt based Timer

A timer, or a timer/counter in the Atmega328 datasheet, is a piece of peripheral hardware built in most of the microcontrollers. It is a counter which increments and then resets to zero once it reaches the max value. It can be used to schedule events, generate waves, or time-stamp events like a clock. Atmega328 has three timers, and we will investigate the 16-bit timer (TC1) in a CTC (Clear Timer on Compare match) mode. As it is a 16-bit timer, the value changes from 0x0 to 0xFFFF (or 65535) using the pre-scaled CPU clock. There is an Overflow Interrupt event when it has reached its maximum value, and the period can be computed. The top value can also be set in a CTC (Clear Timer on Compare match) mode which compares the counter value to that of the OCR register. Once it matches the counter resets to zero. Figure 2 illustrates the CTC operation by changing the top value within the OC1A ISR. The COM (Compare Output Match) bits are also set to generate the toggling output to PB1 pin (where the LED is connected).

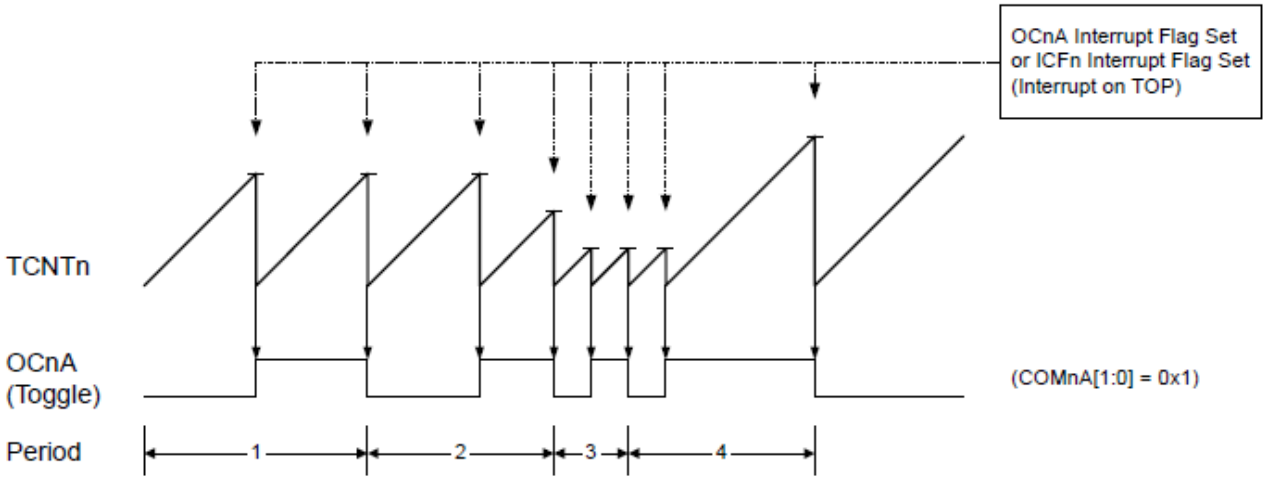

Figure 2 A CTC mode timer operation. The top match value is set in the ISR. By further setting a COM0A0-bit, the PB1 pin is used to generate a toggling signal. (On page 161, datasheet)

- Open the tutorial code "*8_timer_interrupt.c*"
- Build and download the executable and observe the behaviours.
- Timer1 is a 16-bit counter. That is it starts from 0x0000 to 0xFFFF (65535). If we select the Clock Select (CS12, CS11, CS10) = 101 (divided by 1024), and suppose we want to toggle every 1 second. The required timer overflow value would be: 20MHz/1024 = 19531.1Hz, and for 1 second period, the required timer ticks are 19531 (rounded) which should be set to OCR1A (H/L) registers.
- You can also refer to: http://eleccelerator.com/avr-timer-calculator/
- Implement a timer program so that the LED blinking frequency can be set from the UART terminal. For example, typing "10\r" in the Putty terminal will change blinking frequency to 10Hz (note that the LED should toggle twice the frequency of the blinking frequency). *scanf()* function can be used to extract the number from the received string. Demonstrate the program to the tutors to get the marks.

Appendix A. Timer1 Registers

Related Timer1 Registers

- Timer Counter Control Register 1A (TCCR1A) – waveform generation
- Timer Counter Control Register 1B (TCCR1B) – pre-scaling clock
- Timer Interrupt Mask 1 (TIMSK1) – enable/disable timer interrupt
- Timer Output Compare Register 1A (OCR1AH/L) – set the top value

Name: TCCR1A

Bit	7	6	5	4	3	2	1	0
	COM1	COM1	COM1	COM1			WGM11	WGM10
Access	R/W	R/W	R/W	R/W			R/W	R/W
Reset	0	0	0	0			0	0

Compare Output Match. Only required in generating output signals. (‘0000’: normal mode, ‘0011’ = inverting output in PWM)

Waveform Generation Mode. [WGM13:WGM12] are further defined in TCCR1B (‘0000’: normal mode, ‘0101’ = PWM mode, 8-bit)

Name: TCCR1B

Bit	7	6	5	4	3	2	1	0
	ICNC1	ICES1		WGM13	WGM12	CS12	CS11	CS10
Access	R/W	R/W		R/W	R/W	R/W	R/W	R/W
Reset	0	0		0	0	0	0	0

Waveform Generation Mode.

Clock Select. (‘011’: clock/64 = 1Mhz/64 = 15.625KHz)

Name: TIMSK1

Bit	7	6	5	4	3	2	1	0
			ICIE			OCIEB	OCIEA	TOIE
Access			R/W			R/W	R/W	R/W
Reset			0			0	0	0

Other interrupt masks.

Overflow Interrupt Enable.