Ministerul Educatiei al Republicii Moldova
Universitatea Tehnica a Moldovei
Filiera Anglofona

Report

Laboratory Nr.5

Embeded Systems

Performed By: Daniel Ciobanu

Verified By: Andrei Bragarenco

Chisinau 2017

Topic : Timer and interrupts

Task: We have to do like a task schedule using timer and interrupts

Theory:

A **microcontroller** is a self-contained system with peripherals, memory and a processor that can be used as an embedded system. Most programmable microcontrollers that are used today are embedded in other consumer products or machinery including phones, peripherals, automobiles and household appliances for computer systems. Due to that, another name for a microcontroller is "embedded controller." Some embedded systems are more sophisticated, while others have minimal requirements for memory and programming length and a low software complexity. Input and output devices include solenoids, LCD displays, relays, switches and sensors for data like humidity, temperature or light level, amongst others.

AVR Timers – TIMER0

In AVR, there are three types of timers – TIMER0, TIMER1 and TIMER2. Of these, TIMER1 is a 16-bit timer whereas others are 8-bit timers.

TCNT0 Register

The **Timer/Counter Register** – TCNT0 is as follows:

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
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	

TCNT0 Register

This is where the uint 8-bit counter of the timer resides. The value of the counter is stored here and increases/decreases automatically. Data can be both read/written from this register.

Activate the timer:

TCCR0 Register

The **Timer/Counter Control Register** – TCCR0 is as follows:

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	

Choose prescaler:

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.

Interrupts:

In most microcontrollers, there is something called interrupt. This interrupt can be fired whenever certain conditions are met. Now whenever an interrupt is fired, the AVR stops and saves its execution of the main routine, attends to the interrupt call (by executing a special routine, called the Interrupt Service Routine, ISR) and once it is done with it, returns to the main routine and continues executing it.

TIMSK Register

The **Timer/Counter Interrupt Mask** – TIMSK Register is as follows. It is a common register for all the three timers. For TIMER0, bits 1 and 0 are allotted. Right now, we are interested in the 0th bit **TOIE0**. Setting this bit to '1' enables the TIMER0 overflow interrupt.

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
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	

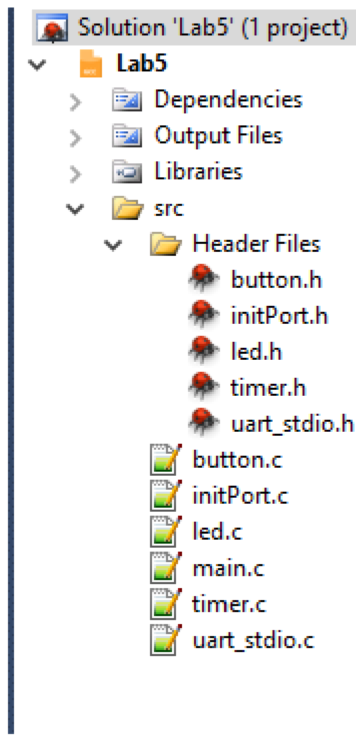
TIFR Register

The **Timer/Counter Interrupt Flag Register**– TIFR is as follows. Even though we are not using it in our code, you should be aware of it.

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
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	

Implementation:

The structure of the laboratory 5 looks like this:



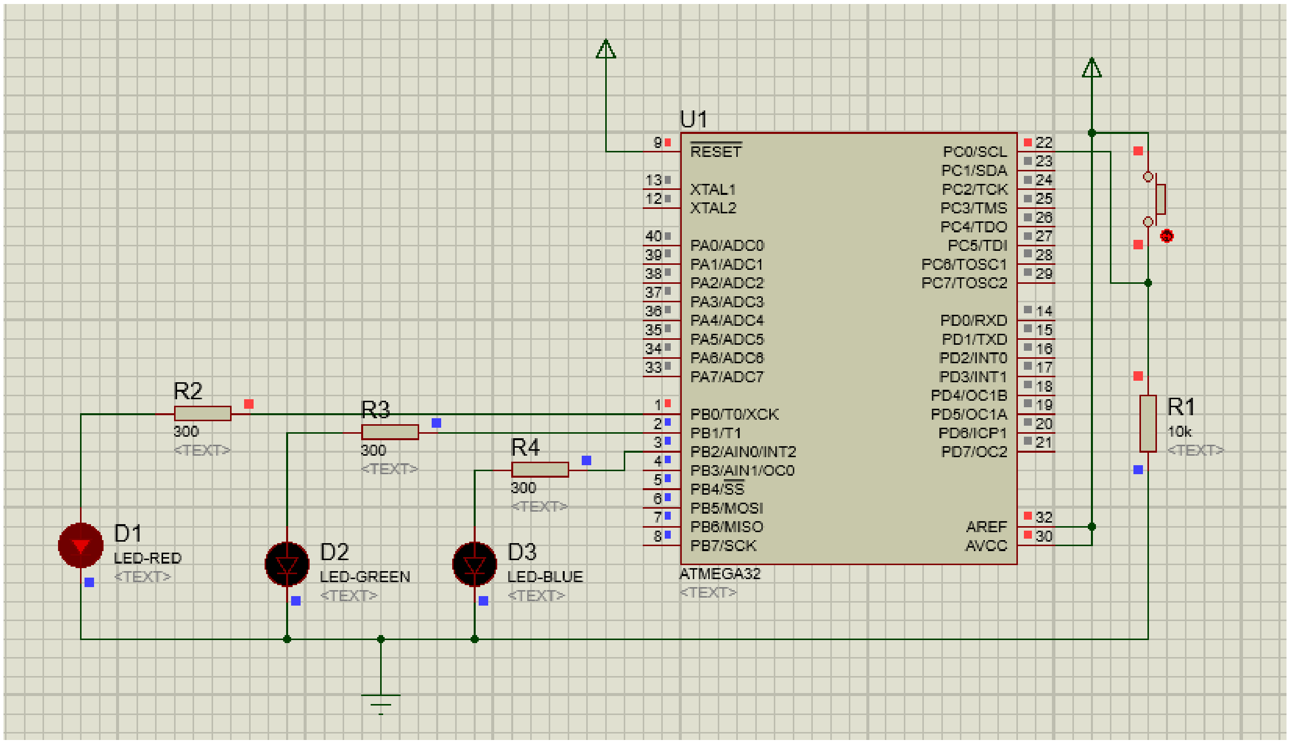
In my laboratory work i have 3 LED’s that flashes every 50 ms when pressing the button and changing from one to another in one second.

```
void toggle_led(uint32_t pin) {

    switch (pin)
    {
        case 0:
            firtsLedOn();
            _delay_ms(1000);
            PORTB = 0x00;
            break;
        case 1:
            secondLedOn();
            _delay_ms(1000);
            PORTB = 0x00;
            break;
        case 2:
            thirdLedOn();
            _delay_ms(1000);
            PORTB = 0x00;
            break;
        default:
            /* Your code here */
            break;
    }


}
```

The scheme in Proteus:



Conclusion :

Doing this laboratory work i learned how to work with timers in AVR Atmega32 microcontroller and interrupts. I also learned how deal with the processor speed using the prescalers.

Appendix:

uart_stdio.h

```
#ifndef UART_STDIO_H_
#define UART_STDIO_H_

#define F_CPU 1000000UL
#include <stdio.h>

void uart_stdio_Init(void);
int uart_PutChar(char c, FILE *stream);
char uart_ReadChar();

#endif /* UART_STDIO_H_ */
```

uart_stdio.c

```
#include "Header Files/uart_stdio.h"

#define UART_BAUD  9600

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

FILE uart_output = FDEV_SETUP_STREAM(uart_PutChar, NULL,
_FDEV_SETUP_WRITE);
FILE uart_input = FDEV_SETUP_STREAM(NULL, uart_ReadChar,
_FDEV_SETUP_READ);

void uart_stdio_Init(void) {
	stdout = &uart_output;
	stdin = &uart_input;

	#if F_CPU < 2000000UL && defined(U2X)
	UCSRA = _BV(U2X);             /* improve baud rate error by using
2x clk */
	UBRRL = (F_CPU / (8UL * UART_BAUD)) - 1;
	#else
	UBRRL = (F_CPU / (16UL * UART_BAUD)) - 1;
	#endif
	UCSRB = _BV(TXEN) | _BV(RXEN); /* tx/rx enable */
}

int uart_PutChar(char c, FILE *stream) {
	if (c == '\n')
	uart_PutChar('\r', stream);

	while (~UCSRA & (1 << UDRE));
	UDR = c;


	return 0;
}

char uart_ReadChar() {
	//Wait untill a data is available
	while(!(UCSRA & (1<<RXC)))
	{
		//Do nothing
	}
	//Now USART has got data from host
	//and is available is buffer

	return UDR;
```

```
}
```

main.c

```
#include "Header Files/uart_stdio.h"
#include "Header Files/led.h"
#include "Header Files/timer.h"
#include "Header Files/initPort.h"
#include "Header Files/button.h"
#include <util/delay.h>


// global variable to count the number of overflows
volatile uint8_t tot_overflow = 0;
uint8_t count = 0;
	// initialize overflow counter variable
	//tot_overflow = 0;

// TIMER0 overflow interrupt service routine
// called whenever TCNT0 overflows
ISR(TIMER0_OVF_vect)
{
	// keep a track of number of overflows
	tot_overflow++;
	if (tot_overflow >= 12)  // NOTE: '>=' is used
	{
		if (isButtonPressed() == 1){
			toggle_led(count);
			count += 1;
			if (count > 2) {
				count = 0;
			}

		}
		tot_overflow = 0;
	}


}


int main(void)
{

	initPorts();
	// initialize timer
	timer0_init();
```

```
	while(1)
	{

	}
}
```

timer.h

```
# #ifndef TIMER_H_
#define TIMER_H_

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

// initialize timer, interrupt and variable
void timer0_init();


#endif /* TIMER_H_ */
```

timer.c

```
#include "Header Files/timer.h"

// initialize timer, interrupt and variable
void timer0_init()
{
	// set up timer with prescaler = 256
	TCCR0 |= (1 << CS02);

	// initialize counter
	TCNT0 = 0;

	// enable overflow interrupt
	TIMSK |= (1 << TOIE0);

	// enable global interrupts
	sei();

}
	PORTC = 0xA5;
}

void moveRight() {
	PORTC = 0x66;
}
```

```
void stop() {
	PORTC = 0xE7;
}
```

led.h

```
#ifndef LED_H_
#define LED_H_
#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>


void firtsLedOn();
void secondLedOn();
void thirdLedOn();


void toggle_led(uint32_t pin);

#endif /* LED_H_ */
```

led.c

```
#include "Header Files/led.h"

void firtsLedOn() {

	PORTB |= (1 << PINB0);
}
void secondLedOn() {
	PORTB |= (1 << PINB1);
}
void thirdLedOn() {
	PORTB |= (1 << PINB2);
}

void toggle_led(uint32_t pin) {

	switch (pin)
	{
		case 0:
			firtsLedOn();
			_delay_ms(1000);
			PORTB = 0x00;
```

```
                break;
            case 1:
                secondLedOn();
                _delay_ms(1000);
                PORTB = 0x00;
                break;
            case 2:
                thirdLedOn();
                _delay_ms(1000);
                PORTB = 0x00;
                break;
            default:
                /* Your code here */
                break;
        }

}
```

initPort.h

```
#ifndef INITPORT_H_
#define INITPORT_H_

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

void initPorts();

#endif /* INITPORT_H_ */
```

initPort.c

```
#include "Header Files/initPort.h"

void initPorts() {

        DDRB|= 0xFF;

        DDRC &= ~(1<<PC0);//Makes firs pin of PORTD as Input

}
```

button.h

```c
#ifndef BUTTON_H_
#define BUTTON_H_

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

int isButtonPressed();

#endif /* BUTTON_H_ */
```

button.c

```c
#include "Header Files/button.h"

int isButtonPressed() {
	if ((PINC & (1<<PC0)) == 1) {
		return 1;
	}
	return 0;
}
```