Faculty of Computers, Informatics and Microelectronics
Technical University of Moldova

Embedded Systems
Laboratory work #5
Task runner using Timer

Authors:
Mereuta Alex

Supervisor:
Bragarenco Andrei

2016

1 Topic

Introduction to Micro Controller Unit programming and implementation of the timer.

2 Objectives

- 1. Studying of timer
- 2. Timer registers
- 3. Create a basic task scheduler using timer
- 4. Understanding of AVR timers' usage.

3 Task

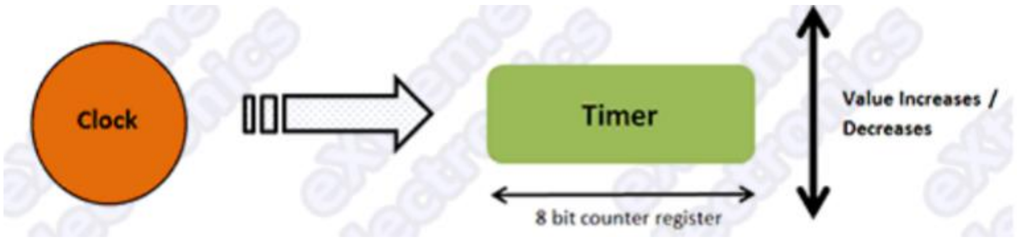
Write a C program and schematics for Micro Controller Unit (MCU) using Universal asynchronous receiver/transmitter. For writing program, use ANSI-C Programming Language with AVR Compiler and for schematics use Proteus, which allow us to explain and to demonstrate the usage of timers in AVR programming.

4 AVR Timers

4.1 Definition

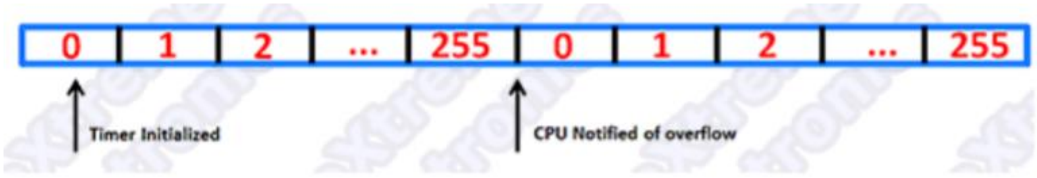
Timers are standard features of almost every microcontroller. So, it is very important to learn their use. Since an AVR microcontroller has very powerful and multifunctional timers, the topic of timer is somewhat "vast". Moreover, there are many different timers on chip. So, this section on timers will be multipart. I will be giving basic introduction first.

A timer in simplest term is a register. Timers generally have a resolution of 8 or 16 Bits. So an 8 bit timer is 8Bits wide so capable of holding value within 0-255. But this register has a magical property! Its value increases/decreases automatically at a predefined rate (supplied by user). This is the timer clock. And this operation does not need CPU's intervention.



4.2 Timer Operation

Since Timer works independently of CPU it can be used to measure time accurately. Timer upon certain conditions take some action automatically or inform CPU. One of the basic condition is the situation when timer OVERFLOWS i.e. its counted up to its maximum value (255 for 8 BIT timers) and rolled back to 0. In this situation timer, can issue an interrupt and you must write an Interrupt Service Routine (ISR) to handle the event.

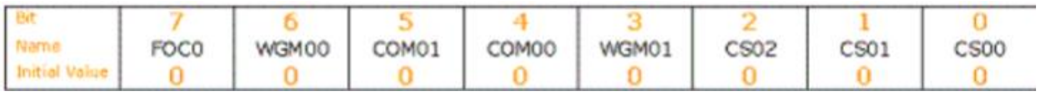


4.3 Using TIMER0, 8-bit timer

The ATmega16 and ATmega32 has three different timers of which the simplest is TIMER0. Its resolution is 8 BIT i.e. it can count from 0 to 255. Note: Please read the "Internal Peripherals of AVRs" to have the basic knowledge of techniques used for using the OnChip peripherals (Like timer!) The Prescaler The Prescaler is a mechanism for generating clock for timer by the CPU clock. As you know that CPU has a clock source such as a external crystal of internal oscillator. Normally these have the frequency like 1 MHz,8 MHz, 12 MHz or 16MHz(MAX). The Prescaler is used to divide this clock frequency and produce a clock for TIMER. The Prescaler can be used to get the following clock for timer. No Clock (Timer Stop). No Prescaling (Clock = FCPU) FCPU/8 FCPU/64 FCPU/256 FCPU/1024 Timer can also be externally clocked but I am leaving it for now for simplicity.

4.4 TIMER0 Registers

As you may be knowing from the article "Internal Peripherals of AVRs" every peripheral is connected with CPU from a set of registers used to communicate with it. The registers of TIMERs are given below. TCCR0 – Timer Counter Control Register. This will be used to configure the timer.



Bit	7	6	5	4	3	2	1	0
Name	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Initial Value	0	0	0	0	0	0	0	0

4.5 Timer interrupt mask register

This register is used to activate/deactivate interrupts related with timers. This register controls the interrupts of all the three timers. The last two bits (BIT 1 and BIT 0) Controls the interrupts of TIMER0. TIMER0 has two interrupts but in this article, I will tell you only about one (second one for next tutorial). TOIE0: This bit when set to "1" enables the OVERFLOW interrupt.

5 Resources

5.1 Atmel Studio

Atmel Studio 7 is the integrated development platform (IDP) for developing and debugging Atmel® SMART ARM®-based and Atmel AVR® microcontroller (MCU) applications. Studio 7 supports all AVR and Atmel SMART MCUs. The Atmel Studio 7 IDP gives you a seamless and easy-to-use environment to write, build and debug your applications written in C/C++ or assembly code. It also connects seamlessly to Atmel debuggers and development kits.

5.2 Proteus Design Suite

Proteus lets you create and deliver professional PCB designs like never before. With over 785 microcontroller variants ready for simulation straight from the schematic, built in STEP export and a world class shape based autoroute as standard, Proteus Design Suite delivers the complete software package for today and tomorrow's engineers. Proteus let's use simulate our hardware before creating it. It's very useful tool especially for beginners. It makes virtual "hardware" which will work like real one.

6 Schemes

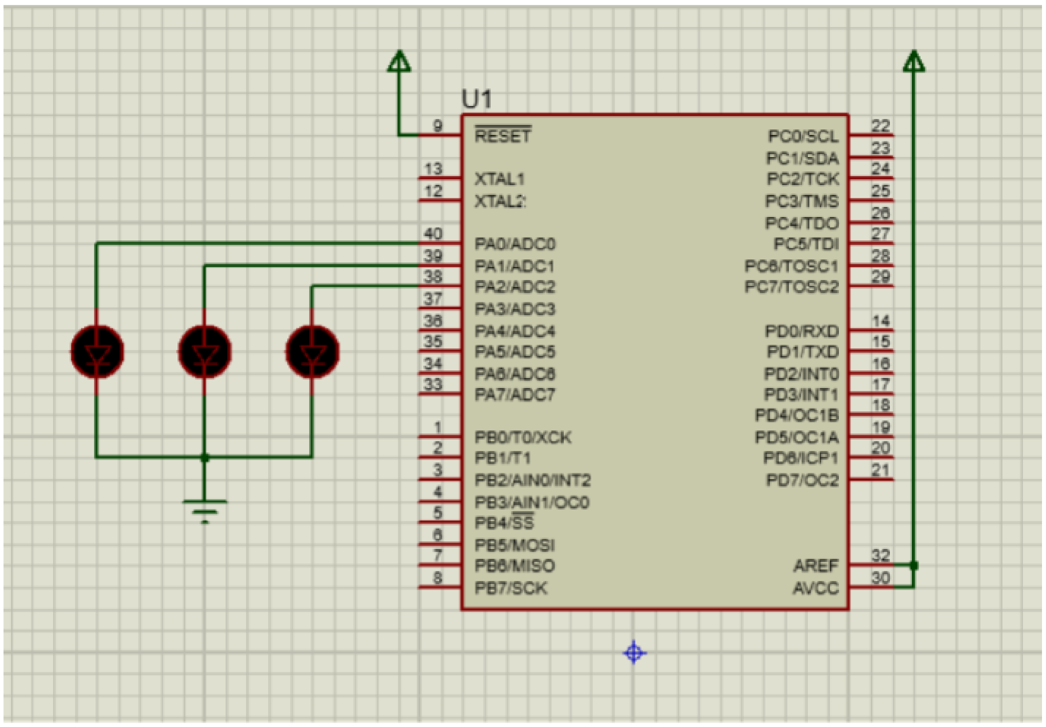


Figure 1 Here we have 3 different colored LEDs that are connected to a micro-controller that ca be turned off and on by the task runner.

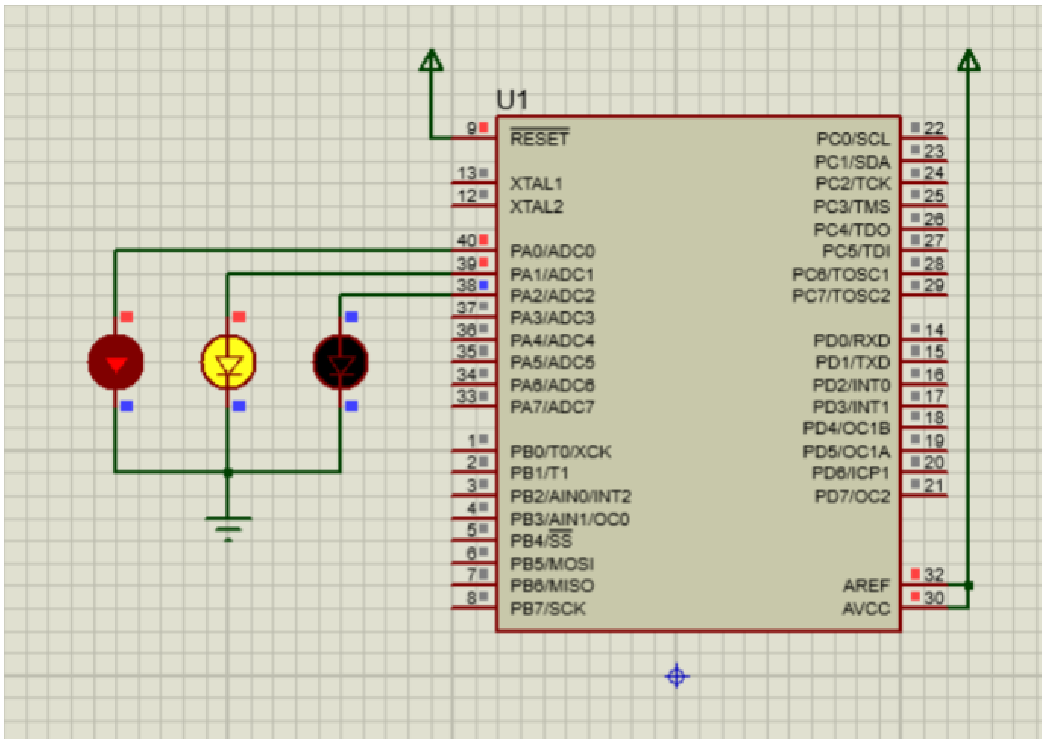


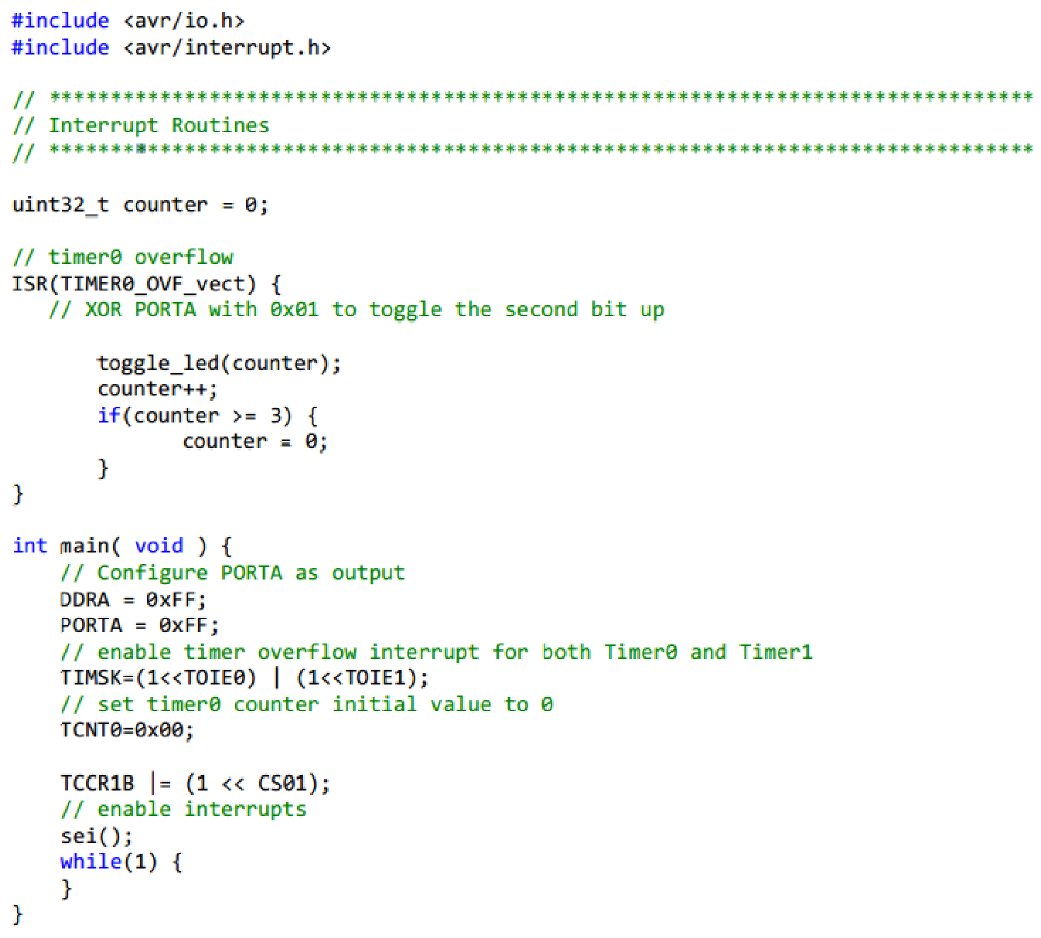
Figure 2 Here it is while working

7 Conclusion

During this laboratory, we have learned the basic concepts of MCU programing in C and building a simple printed circuit with proteus. To do this we used timers. This made me realize how important the timer is in AVR programming.

Appendix

Main



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

// ********************************************************************************
// Interrupt Routines
// ********************************************************************************

uint32_t counter = 0;

// timer0 overflow
ISR(TIMER0_OVF_vect) {
   // XOR PORTA with 0x01 to toggle the second bit up

       toggle_led(counter);
       counter++;
       if(counter >= 3) {
              counter = 0;
       }
}

int main( void ) {
    // Configure PORTA as output
    DDRA = 0xFF;
    PORTA = 0xFF;
    // enable timer overflow interrupt for both Timer0 and Timer1
    TIMSK=(1<<TOIE0) | (1<<TOIE1);
    // set timer0 counter initial value to 0
    TCNT0=0x00;

    TCCR1B |= (1 << CS01);
    // enable interrupts
    sei();
    while(1) {
    }
}
```

Led.h

```c
#ifndef SRC_LED_H_
#define SRC_LED_H_
#include <avr/io.h>


#include "stdint.h"

void init_led();
void turn_on(uint32_t pin);
void turn_off(uint32_t pin);
void toogle_led(uint32_t pin);

#endif /* SRC_LED_H_ */
```

Led.c

```c
#include "led.h"
#include <stdint.h>
#define MAX 5

void init_led() {
	DDRA |= 0xFF;
}

void turn_on(uint32_t pin) {
	PORTA |= pin;
}

void turn_off(uint32_t pin) {
	PORTA &= pin;
}

void toggle_led(uint32_t pin) {
	 PORTA ^= (1 << pin);
}
```