Education Ministry of Moldova

Technical University

Computer Science and Microelectronics

Report

Embedded Systems

laboratory work #5

Author:
Bîrcu Maxim

Supervisor:
Bragarenco Andrei

January 17, 2017

Introduction

Topic

Introduction to Micro Controller Unit programming and implementation of timer.

Objectives

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

Objectives

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.

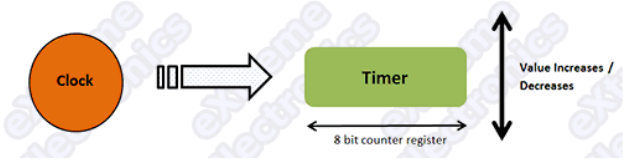
1 AVR Timers

1.1 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.

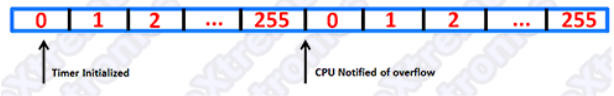
1.2 What is a timer ?

A timer in simplest term is a register. Timers generally have a resolution of 8 or 16 Bits. So a 8 bit timer is 8Bits wide so capable of holding value withing 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.



1.3 Basic operation of a timer

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 upto 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.



1.4 Using the 8 bit timer (TIMER0)

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.

1.5 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

1.6 Timer counter control register TCCR0

As you can see there are 8 Bits in this register each used for certain purpose. For this tutorial I will only focus on the last three bits CS02 CS01 CS00 They are the CLOCK SELECT bits. They are used to set up the Prescaler for timer.

CS02	CS01	CS00	Description
0	0	0	Timer stoped
0	0	1	FCPU
0	1	0	FCPU/8
0	1	1	FCPU/64
1	0	0	FCPU/256
1	0	1	FCPU/1024
1	1	0	External Clock Source on PIN T0.Clock on falling edge
1	1	1	External Clock Source on PIN T0.Clock on rising edge

1.7 TCNT0 – Timer counter 0

Bit	7	6	5	4	3	2	1	0
Name	TCNT0							
Initial Value	0	0	0	0	0	0	0	0

1.8 Timer interrupt mask register (TIMSK)

Bit	7	6	5	4	3	2	1	0
Name							OCIE0	TOIE0
Initial Value	0	0	0	0	0	0	0	0

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. Now time for some practical codes !!! We will set up timer to at a Prescaler of 1024 and our FCPU is 16MHz. We will increment a variable “count” at every interrupt(OVERFLOW) if count reaches 61 we will toggle PORTC0 which is connected to LED and reset “count= 0”. Clock input of TIMER0 = 16MHz/1024 = 15625 Hz Frequency of Overflow = 15625 /256 = 61.0352 Hz if we increment a variable “count” every Overflow when “count reach 61” approx one second has elapse.

2 Resources

2.1 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. Additionally, Atmel Studio includes Atmel Gallery, an online apps store that allows you to extend your development environment with plug-ins developed by Atmel as well as by third-party tool and embedded software vendors. Atmel Studio 7 can also able seamlessly import your Arduino sketches as C++ projects, providing a simple transition path from Makerspace to Marketplace.

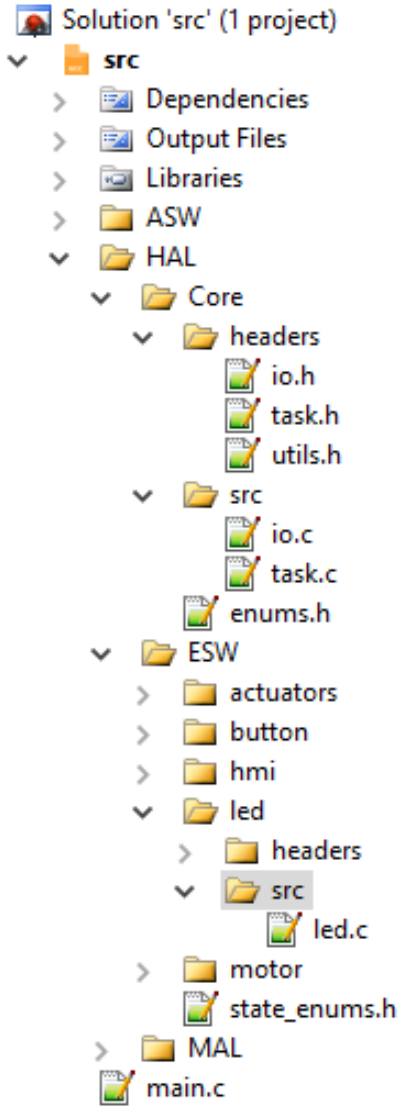
2.2 Proteus

The Proteus Design Suite is an Electronic Design Automation (EDA) tool including schematic capture, simulation and PCB Layout modules. It is developed in Yorkshire, England by Lab center Electronics Ltd with offices in North America and several overseas sales channels. The software runs on the Windows operating system and is available in English, French, Spanish and Chinese languages.

3 Solution

3.1 Project Structure

Project contains a core folder in HAL that contains task.h and task.c implementations and also IO which is a micro-controller connection abstraction.

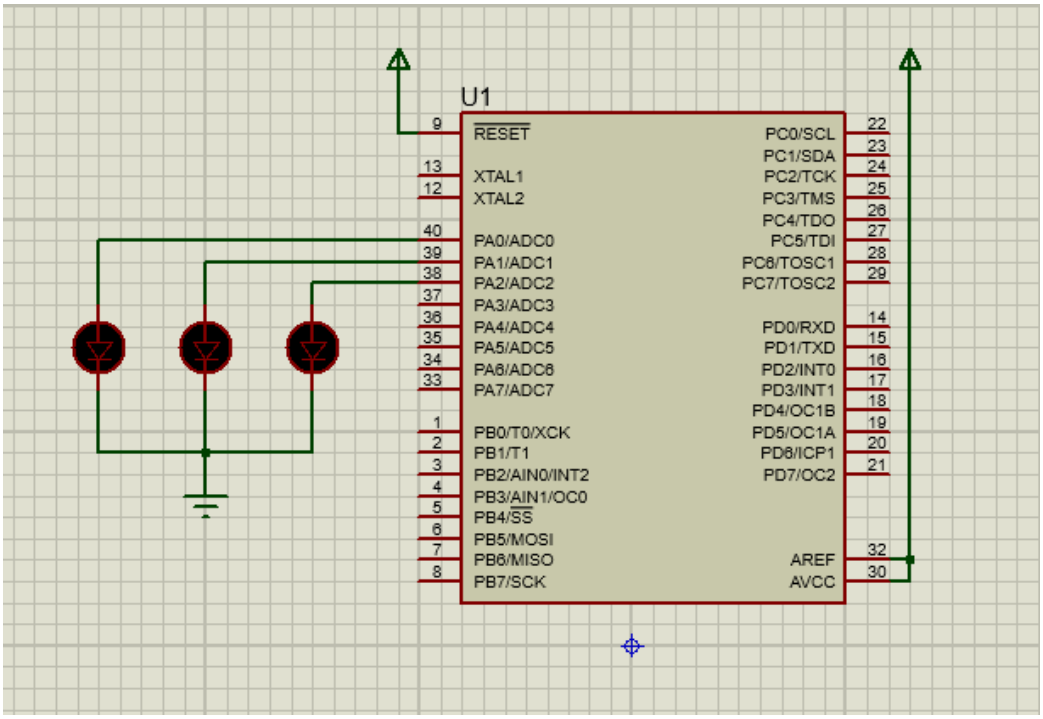


3.2 Main program flow

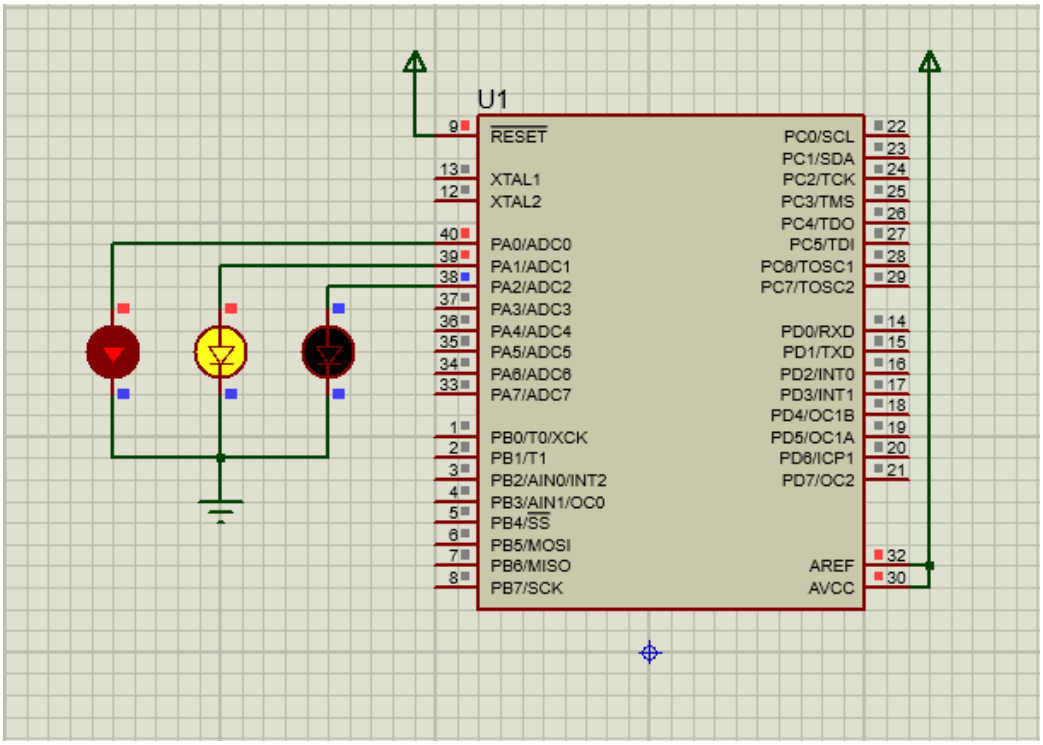
1. Global variable declarations.
2. Tasks initialization
3. Scheduler initialization and running
4. Adding tasks to scheduler
5. Start infinite loop (Controller life cycle)

3.3 Circuit in Proteus

I've connected 3 led of 3 different colors to micro-controller that will be turned on and off by task runner with different interval



3.4 Simulation



Conclusion

In this laboratory work I've learned basic concepts of MCU programming in C language and building a simple printed circuit using Proteus using timers. I've understood that timers is a very important component in avr programming.

Apendix

Main

```
/*
 * src.c
 *
 * Created: 11/20/2016 3:59:23 PM
 *  Author: max
 */

#include <avr/io.h>
#include <util/delay.h>
#include "HAL/Core/headers/io.h"
#include "HAL/Core/headers/task.h"
#include "HAL/ESW/led/headers/led.h"

Led red_led;
Led yellow_led;
Led green_led;

Task toggle_red_led_task;
Task toggle_yellow_led_task;
Task toggle_green_led_task;

void toggle_red_led(){
        toggle_led(&red_led);
}

void toggle_yellow_led(){
        toggle_led(&yellow_led);
}

void toggle_green_led(){
        toggle_led(&green_led);
}
```

```
void init(){
        toggle_red_led_task.delay = 0;
        toggle_red_led_task.is_enabled = 1;
        toggle_red_led_task.interval = 300;
        toggle_red_led_task.handler = &toggle_red_led;

        toggle_yellow_led_task.delay = 100;
        toggle_yellow_led_task.is_enabled = 1;
        toggle_yellow_led_task.interval = 300;
        toggle_yellow_led_task.handler = &toggle_yellow_led;

        toggle_green_led_task.delay = 200;
        toggle_green_led_task.is_enabled = 1;
        toggle_green_led_task.interval = 300;
        toggle_green_led_task.handler = &toggle_green_led;
}

int main(void){
        red_led = create_led(0, &DDRA, &PORTA);
        yellow_led = create_led(1, &DDRA, &PORTA);
        green_led = create_led(2, &DDRA, &PORTA);

        init();

        scheduler_start();
        scheduler_add_task(&toggle_red_led_task);
        scheduler_add_task(&toggle_yellow_led_task);
        scheduler_add_task(&toggle_green_led_task);

        while(1) { };
        return 1;
}
```

Task.h

```
/*
 * task.h
 *
 * Created: 1/17/2017 2:14:53 AM
 *  Author: bircumaxim
 */


#ifndef TASK_H_
#define TASK_H_

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


#define MAX_TASKS 20
#define MS_PER_CLOCK 10

typedef struct Task {
        uint32_t delay;
        uint32_t interval;
        uint8_t is_enabled;
        void (*handler)();
} Task;

void scheduler_start();
void scheduler_add_task(Task *task);

#endif /* TASK_H_ */
```

Task.c

```
/*
 * task.c
 *
 * Created: 1/17/2017 2:17:08 AM
 *  Author: bircumaxim
 */

#include "../headers/task.h"
#include "../headers/utils.h"

uint8_t tasks_count = 0;
Task **tasks;
int32_t tasks_remaining_time[MAX_TASKS];

void scheduler_start(){
        tasks = malloc(sizeof(Task*) * MAX_TASKS);

        bit_set_1(&TCCR1B, CS11);

        bit_set_1(&TCCR1B,WGM12);

        bit_set_1(&TIMSK,OCIE1A);


        TCNT1 = 0;
        OCR1A = 1250; // 0.01 ms

        sei();
}

void scheduler_add_task(Task *task){
        if(tasks_count < MAX_TASKS - 1){
                tasks[tasks_count] = task;
                tasks_remaining_time[tasks_count] = task->delay;
                tasks_count++;
        }
```

```
}

ISR(TIMER1_COMPA_vect){
        int i;
        for(i = 0; i < tasks_count;i++){
                // avoid overflow
                if(tasks_remaining_time[i] >= 0){
                        tasks_remaining_time[i] -= MS_PER_CLOCK;
                }

                uint8_t should_run = tasks[i]->is_enabled && tasks_rema
                if(should_run){
                        tasks[i]->handler();
                }

                if(!tasks[i]->is_enabled || should_run){
                        tasks_remaining_time[i] = tasks[i]->interval;
                }
        }
}
```