Lab 0: Intro to AVR Programming

EE222: Microprocessor Systems

Contents

1 Acknowledgements

This lab exercise is prepared by Mohammad Azfar Tariq and Muhammad Usman under the supervision of Dr. Rehan Ahmed for the course EE-222 Microprocessor Systems. Reporting any error or discrepancy found in the text is appreciated.

2 Administrivia

2.1 Learning Outcomes

By the end of this lab you will be able to;

1. Create Project in Atmel Studio
2. Simulate Assembly code for AVR ATmega16A
3. Burn Hex file in ATmega16A using the Universal Programmer

2.2 Deliverables

You are required to submit

- Simulated observations in a tabular form
- Quantitative and qualitative comparison of simulation and implementation

in the beginning of next lab.

3 Hardware Resources

- ATmega16A Microcontroller Unit
- Universal Programmer
- LEDs (may use from trainer kit)
- Power source with voltage regulator (may use from trainer kit)

4 Introduction

You should not worry about nitty gritty details of code for the moment. The purpose of this lab is to familiarize with you, the AVR toolchain and it's workflow. We employ following code for this purpose.

```
                            ; initial constants
    ldi R16, 0xFF
    ldi R17, 0xFF
                            ; set DDRB as output
    out DDRA, R17
                            ; code to toggle LEDs
toggler:
    subi R16, 0xFF
    out PORTA, R16
    rjmp idle_loop
                            ; delay loop
idle_loop :
    ldi R19, 0xFF
    ldi R20, 0x0F
    ldi R21, 0x01
idle_loop_0 :
idle_loop_1 :
idle_loop_2 :
    dec R19
    brne idle_loop_2
    dec R20
    brne idle_loop_1
    dec R21
    brne idle_loop_0
rjmp toggler
```

The code is designed to count from 0x00 to 0xFF which will be displayed on 8 LEDs. There are two nested loops in the code, the outer loop (named "toggler" in the code) increments the count on each iteration and the inner loop (named "idle_loop" in the code) just iterate for some time doing nothing, wasting the time of CPU (a sort of time delay). We will simulate and burn the code in ATmega16A. You can find this code here[1] as well.

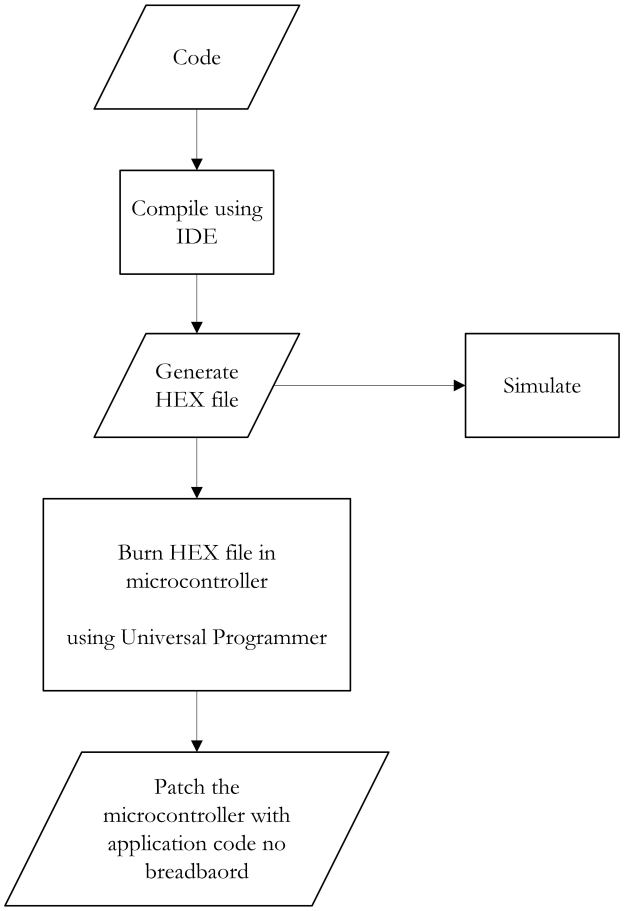
4.1 Workflow

Following is the workflow we shall observe.

1. Create project and code in Atmel Studio.
2. Simulate the code.
3. Create Hex file.
4. Burn Hex file in microcontroller using Universal Programmer
5. Patch the circuit on breadboard to observe output.

[1] https://github.com/Uthmanhere/EE222/blob/master/00_lab1Toggle.asm

Same workflow is elaborated in the following flowchart:



Hex File A file format that conveys binary information in ASCII text form. The file is used to burn code in microcontrollers. IDE generates the hex file. It is then used by Universal Programmer to burn the code in microcontroller unit. After burning the code, microntroller becomes ready to be installed in breadboard or PCB designed for the specific application in mind

5 Lab Tasks

5.1 Introduction to Atmel Studio

Atmel Studio is an IDE for programming 8 bit and 16 bit AVR microcontrollers. The IDE generates hex file for a range of such microcontrollers. Moreover, quite a handful of simulations can be undertaken and observed in it.

Familiarize yourself with Atmel Studio by

1. Creating new Project
2. Coding Environment
3. Simulation/Debugging Features

Introductory Tutorial Slides[2] have been compiled to help you through this.

Add a break-point on line `9` (as directed in Introductory tutorial slides) of the code (which is inside the "toggler" loop) for simulation purposes. It will assist you in understanding the overall functionality of the code. Observe the following parameters,

- Status Register
- Time taken in iteration
- Cycles for iteration
- Register R16
- I/O Port A

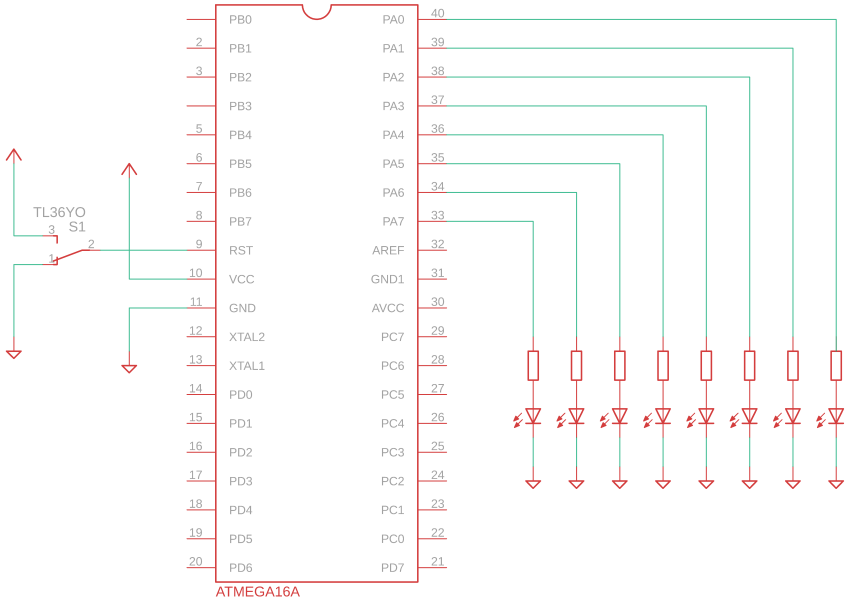
Fill the table for first **20 iterations**. The table to be filled is available in doc and pdf format.

Note Time and cycles for each iteration can be calculated from Stop Watch time and Cycle Counter in the simulation panel.

[2] https://github.com/Uthmanhere/EE222/blob/master/00_atmel_tutorial.pdf

5.2 Burning the Code in Microcontroller

Patch the circuit as schematic suggests. You may utilize the resources available in the trainer kits of lab.



Burn the hex file generated in the above project using universal programmers. Follow the Universal Programmer Tutorial[3] to familiarize yourself with the task.

Oscillator Clock is an essential component of any processing unit. It is the primary signal, the entire circuitry synchronizes itself to. Generally clock can be generated in two ways:

- External Crystal/Ceramic Resonator
- External Low-frequency Crystal
- External RC Oscillator
- Calibrated Internal RC Oscillator
- External Clock

Do not worry about these different sources of clock for the moment. By default, **1 MHz Internal RC Oscillator** is being used. This frequency is enough for lab exercise under consideration.

Task Does the implementation follow simulation? Time blinking of any reasonable LED and compare it with simulation. Document these comparisons and observations.

[3] https://github.com/Uthmanhere/EE222/blob/master/universal_programmer.pdf