September 5th, 2017

Lab 02 – Reading From and Writing to I/O Ports

Zachary Davis

Christopher Bywaletz

**Reading from and Writing to I/O Ports (Lights and Switches, The Simulator)**

**Introduction**

The purpose of this lab was to further my knowledge and experience of the EVBplus2 68HC11, BUFFALO, and the AsmIDE as well as familiarize myself with the board simulator for the first time. More specifically, this lab focused on manipulating the input port, port C, and the output port, port B. This lab served as good practice for working with and troubleshooting programs with the EVB board, BUFFALO and the AVOCET simulator.

The goal of this lab was to further explore the components of the lab and understand input and output of the EVBplus2 both in real life and the simulator. Having not had a lot of exposure to the EVB board, BUFFALO, and AsmIDE, one of the first things that I need to become familiar with is the inputs and outputs on the board as the will become an essential part of working with the board in future labs. On top of that, future programs will be fully developed and written by us so learning how to use the simulator helps understand what is going on in the board and what issue may need addressing.

**Procedures and Discussion**

The procedure of this lab were split into two sections: (1) Familiarizing myself with the board I/O and manipulating the switches and LEDs on the board to achieve desired results, (2) Setup the exact program from the end of part one in the simulator. Although this is the opposite order that these procedures would normally occur, they still serve the same purpose.

The first part of the lab began by practicing reading from input ports on the board. This was done by manipulating the position of the eight on board switches with one end connected to port C as well as VCC, through a 100k ohm pull-up resistor, and the other end connected to ground through a 4.7k ohm pull-down resistor. To start, we positioned the switches as shown in a provided picture with half in one position and half in the other. We the displayed the memory in address $1003 in the terminal and found it to be 0F in hexadecimal. In binary this is, of course, 00001111, which reflects the positions of our switches. Since the switches connected to port C are connected to a pull-up resistor a 1 corresponds to an open switch and a 0 to a closed switch. This whole process is then repeated a couple of times only to reaffirm what was stated above. This not only familiarized us with the way inputs work on the EVBplus2 board, but also showed us that the pull-down resistor connected to the switches on the board is redundant and will act as a pull-up resistor anyway.

The next step in part one was too experiment with writing to and output port. This was done by changing the value at the address of port B, $1004, which is connected to eight LEDs. Each of these LEDs is connected to a pin on port B as well as a resistor which is connected to ground. Using BUFFALO commands in the terminal, we were able to view and modify the values at port B and see the output in the LEDs. To do this we would use the BUFFALO command MM $1004 x or modify memory at the address 1004, port B, to x, where x is some integer number base ten from 0 to 255. Whatever number we choose to store in port B’s memory address would be shown in binary by the LEDs with a lit LED representing a one and non-lit LED representing a zero.

The final step of part one was to write an assembly program that reads from the input ports and constantly write that value to the output ports. In other words, we put both of the previous sections together and had the switches connected to port C controller the on/off state of the LEDs connected to port B. The initial part of the assembly program was provided in the lab manual and we filled in the vital lines. Our portion of the code was to load the value in address 1003, ldaa $1003, store that in address 1004, staa $1004, and finally loop back to the start, jmp start. This, of course, means that once we compiled and ran the program when the DIP switch was open the LED would be in the one state and vice versa.

In step two we took the program that was created previously and loaded it into the AVOCET simulator that helps display what is going on in the different components of the EVB board. Once we determined the lower and upper bounds of the memory and loaded the program, we first used the simulator to just run the program. As this moves to fast to actually see and comprehend what is happening we then ran the program sequentially, which step through each line of assembly code. This allowed us to see exactly what each line does and in the future with more complex programs will help us troubleshoot and see were errors may occur.

This lab was relatively simple and served more as a learning experience that one focused on problem solving. That does not mean however that the lab was completed without any problems. While working with the input port and output port were relatively straight forward, combining them and coding in assembly for the first time was not easy. We knew fundamentally what we wanted to do, but actually implementing it took some time to solve. Also working with the simulator was very daunting at first and still is somewhat, but now I have basic understanding of how it works.

**Conclusion**

The goal of this lab was to further explore the components of the lab and understand input and output of the EVBplus2 both in real life and within the simulator. Having worked with the input and output ports I definitely have gain a better understanding and was able to recall many functions from the previous lab as well. Having been exposed to assembly it is clear that there is a lot of work to go, but I feel more comfortable with it now than I did a week ago. Finally, being exposed to the simulator was very interesting. Unlike most of what I have seen before this is a very effective tool for debugging programs on the EVB board. This lab effectively expanded my knowledge with the lab components.

**References**

Not Applicable

**Appendices**

**Appendix A: Pseudo Code for the Software Developed**

program input/output\_demo

set the value at address 1004 equal to portb

set the value at address 1003 equal to portc

set the value at address 1007 equal to ddrc

set the value at address 8FFF equal to STACK

begin the program at address D000

start

load the stack

load the address 00 into accumulator a

store the value of accumulator a into ddrc

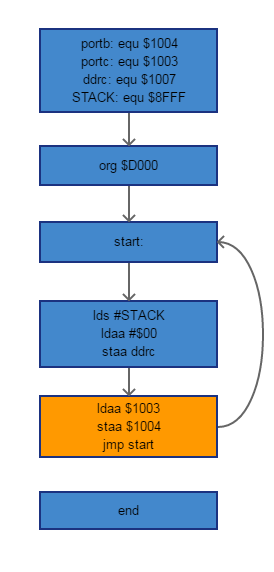
load the value at address 1003 into accumulator a

store the value of accumulator a into 1004

loop back to start

end

**Appendix B: Program Flow Chart**



**Appendix C: Program Listing**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

portb: equ $1004

portc: equ $1003

ddrc: equ $1007

STACK: equ $8FFF

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

org $D000

start

lds #STACK

ldaa #$00

staa ddrc

ldaa $1003

staa $1004

jmp start

end

**Lab Questions**

**2.1**

1. What is the value of PORT C for figure 1(a)?

0F (hex), 00001111(binary)

1. What do the 0 and 1 correspond to relative to the switch position?

1 Corresponds to an open switch

0 Corresponds to a closed switch

1. What is the value of PORT C for figure 3?

FF (hex), 11111111 (binary)

1. What is the value of PORT C for figure 4?

00 (hex), 00000000(binary)

1. Explain how the switches work in conjunction with the pull-up and pull-down resistors to generate the observed voltages on PORT C.

The switches are single pull double throw. When the input of the switch is high the pull-up resistor’s switch is open and the pull-down resistor’s switch is closed. This will cause Vin to be high from both the pull-up and pull-down. When the input switch is low the pull-up resistor’s switch will be closed and the pull-down resistor’s will be open. Both the pull-up and pull-down resistors force Vin to be low. In this way they are rather redundant and the pull-up resistor or pull-down resistor would be fine on its own.

**2.2**

1. What is the BUFFALO command used to change the contents of an output port?

MM address x, where address is the address of the output port and x is the value you are modifying it to.

1. What is the correlation between what you store in the output port and the state of the LEDs (on/off)?

1 corresponds to an on LED

0 corresponds to an off LED

**2.3**

1. What are the missing lines from the provided program that will cause the switches to control the on/off state of the LEDs?

Line 10: ldaa $1003

Line 11: staa $1004

Line 12: jmp start

1. What is the relationship between the switches and the LED’s state?

DIP positions open, LEDs on

DIP positions closed, LEDs off

**2.4**

1. What happens when you try to change the contents of PORT B in the simulator?

This cannot be done because PORT B is an output port and therefore does not receive inputs.

**Lab Participation**

For this lab splitting the work evenly while working both with the board and the computer was more difficult. For that reason, my lab partner was at the computer and I worked with the board, but we both helped each other as well. I was able to notice that the position of the switches determined the value stored in PORT C’s address and that modifying the value in PORT B’s address would produce a similar output on the boards LEDs. My partner focused on using the simulator as well as writing the assembly program that combined the use of the input and output ports.