MICROPROCESSOR II AND EMBEDDED SYSTEM DESIGN

Lab Number: 2

Date: 03/09/2016

Student Name: Trever Wagenhals

Student ID:01425986

Group Name: *STEEL ELEPHANTS*

PERSONAL CONTRIBUTION TO THE LAB ASSIGNMENT

For this lab, I ended up writing up the majority of both codes. Among the three of us, we ended up having 2 PICs and Galileos because Boyang bought himself one. So, I created a lot of the code, and he tried to get it to work properly on his setup. We had a lot of issues along the way, however, so I found myself spending a lot of time tying to diagnose the issues and fixing mistakes on the fly on my own setup.

PURPOSE OF THE LAB

The purpose of this lab was to learn about how to integrate microprocessors with other systems. Using the Galileo, it allowed us to get familiar with the Linux environment, including commands and file structure. It also gave us a basic understanding of the idea of a master-slave relationship between two devices. Through this relationship, the concept of handshaking while sending data was also explained as it was necessary to utilize this concept in the lab. The concept of general purpose input/output pins were explained and used to bit-bang data in nibbles back and forth between the two devices to fetch required data. The end goal was to be able to get the ADC result to print on the screen once the Galileo requested and fetched it from the microprocessor.

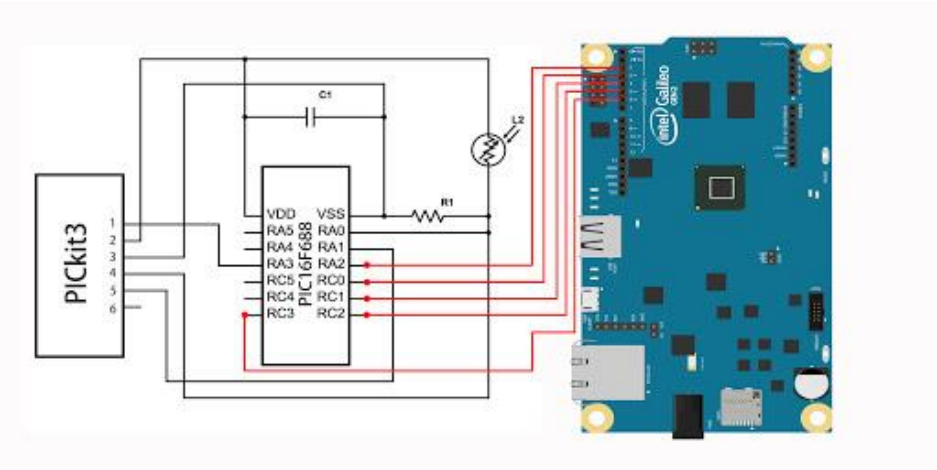
INTRODUCTION OF THE LAB AND REPORT

This lab built on a lot of concepts learned from the first lab in relationship to the microprocessor. In this lab, however, the LED was not necessary to verify results, but it was helpful in error checking. To execute this lab, each team must demonstrate their understanding of GPIO functions and handshake/bit-banging communication by writing a program that sends 4 bits at a time back and forth between a Galileo and a PIC microprocessor. These bits from the Galileo will inform the microprocessor what command it wants it to executing between MSG_GET, MSG_PING, and MSG_RESET. Once received, the PIC will send back an acknowledgement message that will be verified for functionality, and then it will execute the proper command. If asked to MSG_GET, the PIC will start to send back the value of the 10 bit ADC in nibbles until the full message is read.

MATERIALS, DEVICES AND INSTRUMENTS

PICkit3
Galileo 2nd Generation
Breadboard
Wires
10k Ohm Resistor
0.1 uF Capacitor
Laptop for programming
MicroSD Card

SCHEMATICS



LAB METHODS AND PROCEDURE

The first step of the lab would be to understand how to operate the Galileo, including booting it up within your OS environment, accessing the Linux OS on the MicroSD card, and navigating the files on the MicroSD card.

The OS was easy to operate in a Linux environment. Simply, just go to the terminal, and type:

sudo screen /dev/ttyUSB0 115200

Press Enter twice, then type root and hit enter. Now, at this point it is possilbe to navigate the files by typing cd <file>. The most common one was cd /media/ssx1, where this represents the flash drive that has the program saved to it.

The other option was to navigate to the MicroSD card file if you saved the program to that location.

Other major commands used were:

gcc someprogram.c -o nameprogram
./nameprogram
vi someprogram.c

Where gcc compiled the program, ./nameprogram executed it, and vi allowed you to make changed to that program in the terminal. It's important to know to press 'I' to begin editing and 'esc :wq' to save changes.

The next step of this lab would be to verify how exactly the GPIO functions on the Galileo board. To do this, small scripts should be written to test that you can get an LED to light on each pin, and write a program to detect input voltage.

The same functionality check should be done on the 5 pins on the PIC that are going to be used, making sure they can all light an LED and can detect input. The best way to do this is to toggle a pin high if another pin detects input.

Finally, you can start writing your programs to meet the requirements specified by the lab.

TROUBLE IN THE LAB AND HOW DID YOU SOLVE IT

This lab posed a numerous amount of problems the entire time. Even just initially running everything was harder than expected. On my laptop, I attempted to download putty to be able to run the Linux OS of the Galileo. For some reason, however, it would not allow me to assign the port properly in this program and so it was unusable. We were able to use Boyang's Macbook for the meantime without any issue to pull up the OS. In the end, I also installed Linux Mint on my laptop that made the whole process a lot easier on my laptop.

Once the Galileo was up and running and viewable, now I just started writing the programs. Minus the error messages that were fixed after a few attempts, the programs both compiled and a good prototype was created. When we attempted to test these two programs together, everything appeared to run fine, except for the values that were returned by the ADC and the acknowledge were showing up as all 1s. When we decided to ask the teacher about it, he had questioned what tests we had performed, which was replied with none since I had just written both programs up and decided to try them. He insisted that we need to run smaller test programs to verify functionality.

Following his instruction, I wrote a few tests; a set that has the Galileo as the input and the PIC as the output, and one that has the Galileo as the output and the PIC as the input. Once written, it was time to test each individual pin. By putting an LED on each pin of the PIC while it was set as an output, it was easy to verify that the program I wrote was functioning as expected. The same attempt was made when the Galileo was set to output, however no lights would come on. Moving on to further testing, I tried setting the Galileo as an input and creating a message when voltage was read. As soon as my test program was run however, it was constantly detecting voltage without anything connected. Lastly, I attempted to light an LED if a pin on the PIC detected input from the Galileo, but nothing I did worked. So, clearly there were some issues with my Galileo program. To try to get to the bottom of it, I posted my issues on the Piazza page, and based on a new chart that mapped the IO pins for the Galileo Gen2, I realized that I did not have all of the proper IO pins open

for functionality. Adding these additional pins and reattempting to light an LED with the pins showed positive results this time. Moving on, I decided to try the input test again, however voltage was detected instantly still. So, here I decided to change which IO pins were the ones being read for the voltage values, and after some testing, found the correct IO pins to get proper readings. At this point, all 4 of the tests worked properly and so it was time to test the program again.

At this point, the test ran and finally a value other than 1023 came back, 187. I decided to run it more times trying to vary the light, except every time the same value of 187 would be returned. I decided to try to take to the forums again to try to receive guidance. After receiving a little bit, it gave me some ideas of different things to try. So, the first thing I did was simplify my PIC code; instead of using PORTCbits for every individual bit, I used the command PORTC and sent 4 bits at a time. This shortened the code and made it easier to follow. After running the test again though, the same value were being seen. Thinking it may be related to lack of delays, I decided to insert a few while functions that acted as delays to see if that helped. Nothing changed. Next, I assumed it had something to do with me trying to do all 10 bits in one function, so I divided up the three groups into three functions. Somewhere amongst all of these changes, the value that was being received changed, but the repetition when ran was still the same. At this point, I thought it had to do with something on my Galileo code.

On my Galileo code, I decided to start altering a bunch of different functions to see if anything would change. I rewrote my openGPIOforRead function hoping that would change the result, which it did not. At some point, I also realized that my results were being read backwards and that my MSG_GET command was actually set to 4, not 2. I believed fixing this would fix the problem, but it continued to persist.

On my PIC program again, I decided to try to use the ADC flag to see if controlling that could give different results. Using that seemed to completely mess up my program however, giving really inconsistent results. The ADC was finally changing, but the ACK value was not being set to 14 each time like it was supposed to be. This made me believe that these changes were doing more harm than good because the MSG_ACK was at least outputting the proper values each time even though the ADC was wrong. So, I decided to stop using the ADC flag bit.

At some point during all of this, I had realized that I foolishly had some of my connectors plugged in wrong. I changed this to the proper setup, but much to my surprise I had similar issues still.

At this point, I decided to stop trying to read the ADC bits and just type the bits that I wanted to read. By doing this, a few observations were able to be made. First, it appeared that the first 4 bits that I wanted to be sent weren't being read properly, so everything was being shifted by 4 bits. To fix this, I just put a blank bit statement in the front as a temporary fix, which did appear to shift the bits back. The only problem at this point then was that the last 2 bits would never turn off, thus giving a much higher value than should be received. At this point, there was very little I could do to figure out what could be wrong, so I decided to rewrite the ADC function during some free time at work. While I started writing this, I quickly realized that I had use if(RA2 = 1) instead of == in my program, which I noted while I rewrote the function. I decided that if my new functions did not work, I would fix this issue and see if that fixed the problem. After attempting my new functions, only values of 0 were being received. So, instead of diving deeper I decided to fix the old small mistakes, where the program finally started working properly!

RESULTS AND CONCLUSION

Overall, this lab was a very difficult one that introduced a lot of new concepts that most students probably have not been exposed to before. Through a lot of testing and debugging, the concepts of GPIO pins, bitbanging, master-slave relationships, handshaking, microprocessor programming, and C programming in Linux were learned. Trying to program something to send only 10 bits and having so many issues makes you realize how hard it is to implement a data transfer standard such as GPIO or i2c, which can send millions of bits a second across a line with minimal errors. Overall though, I feel that a lot was learned from the lab that can make using these techniques in the future significantly easier due to the similar style no matter the platform. The final results showed how the ADC value is lower when there is less light and higher when there is more light. These values are higher than the first lab because the resistor and diode were removed that accounted for some voltage drop.