Serial Interfacing Using SPI and I2C

Lab Report
Kendall Lui and Mary Florek
2/1/18

1. INTRODUCTION

There are many different protocols that peripherals can use to communicate with devices. Two of these protocols are the I2C and the SPI interfaces. The I2C consists of two wires for clock and data; while SPI uses a minimum of four wires: clock,data out, data in, and chip select. Because SPI has more signals to use, it is generally a faster method of communication. This lab was designed to help understand the use of these two protocols in order to control an accelerometer and OLED display.

2. GOALS

The tasks given us for this lab were threefold. First, we were to learn how to use the SPI interface to communicate with the OLED, first by communicating between two devices and secondly by portraying various geometric shapes on the OLED screen. Our second task was to understand the I2C protocol in order to communicate with the on-board accelerometer. As the accelerometer was tilted, we could see the x and y axis data displayed on TeraTerm. Our final task was to combine these new skills to display a ball on the OLED screen and move the ball based on the tilt of the accelerometer. Saleae logic waveforms were recorded for verification at each step.

3. METHODS

The first part of the lab made use of the SPI demo project provided in the CC3200 sdk examples. This program demonstrated SPI communication between two Master and Slave SPI devices. It allowed us to transfer a message between two launchpads, one acted as a slave and the other as a master. After understanding how the SPI protocol worked, we were tasked at implementing an SPI protocol to communicate with an OLED display. Most of the code was provided except for the direct SPI write data/command functions. In order to implement these we had to understand the data sheets that were provided for the display.

The implementation for the writeCommand and readCommand functions was simple, we used a non-standard chip select pin, pin 62, to enable and disable SPI communication by simply pulling the pin LOW or HIGH. In order to determine whether the SPI write was a command or data we had another pin, pin 63, that went LOW to assert a command or HIGH to assert data. After chip select and data/command mode was determined we called on the SPIDataPut() function which wrote the data to the SPI line on the MOSI, pin 63. We then had to clear the buffer so a SPIDataGet() was called and after that completed we brought the chip select line back to high to disable the SPI communication.

Next, we tested the functions were working by calling Adafruit_Init() which initialized the OLED display. The final step was to call on the various functions to display the various test cases. This required that we look at the various functions and their comments to appropriately set up colors and positions.

The most challenging part was implementing the character set print out since we had to calculate when the text would roll over onto the next line. The OLED display was then able to run through various sequences to show that the display was fully working.

The second part of the lab was fairly simple, as we were able to use the the I2C project demo from the CC3200 project example folder. After importing and building this project, we opened Tera Term and used the readreg command to view the data generated as the accelerometer was tilted one way or the other. The x values were stored in register 0x3, the y in 0x5 and z in 0x7, so we were able to specify exactly which locations we wished to view. We viewed the code and found that a write was called then a read. This was to request a specific register that was to be read. The read function read the data from I2C. This data would be helpful in the last part of this lab to help us implement a rolling ball based on accelerometer data.

The final task would combine what we had learned about the SPI and the I2C protocols to control a rolling ball on the OLED display. Using the provided fillCircle function, we were able to draw a small ball about 4 pixels wide in the center of the screen. We then implemented a while loop that read the current tilt of the accelerometer and added it to the current x and y values of the ball's location. We had to make sure we were casting the numbers appropriately so that the 2's complement data remained negative. In this way, the ball would move across the screen incrementally as the accelerometer was tilted. Of course, we also had to take boundary conditions into account so that the ball wouldn't roll off the screen. Knowing the screen was 128 pixels square, and the ball was 4 pixels round, we tested every x and y value to see if it had exceeded 124 or went below 4. If it did, we would simply re-assign the value to 124 or 4 so the ball would remain in place against the imaginary wall until tilted the other direction.

During the process we used a Salae Logic analyzer to inspect the SPI data signal of the OLED interface. The attached screenshot shows the first writeData(0x12) found in the Adafruit_Init function. The data/command pin is pulled up and the data written is shown to be 0x12. In addition, we were able to analyze the I2C signal as shown in the attached image. It demonstrates the initial write to register 0x3 and then a read from the actual register with a response of 0x2.

4. DISCUSSION

The most challenging part of this lab was understanding what needed to be implemented to get the OLED working. Once, we looked at the data sheet we found the timing diagram that demonstrates the SPI communication of the device with the various additional device specific lines. On the first implementation we did not realize we needed to call the SPIDataGet function before the next write was called. Additionally, there is a small quirk that requires us to call the internal SPIEnable function in order to allow the SPIDataPut function to work properly even though we were using a different chip select pin. After we discovered this the code worked perfectly.

One of the main issues that we had with the display was its low refresh rate clearing the entire screen required around 1 second. As a result when we implemented the rolling ball we found the ball to flicker a lot in order to try and reduce this we modified our code

to have the erase ball screen and display the ball at new location functions called immediately after one another. We have thought of several other ways to help reduce the flicker on the screen. We could attempt to only erase and draw portions of the non overlapping ball rather than erasing the entire ball. Additionally, we could modify the way the drawing and write functions work and reduce the number of times the chip select pins were pulled high and low. An important tool that we learned to use in this lab was the Salae logic analyzer. This simple tool allowed us to directly see the signals that were actually coming out of our launchpad. It was especially helpful when we discovered that the write data command for SPI was not working because we could see that the data was simply not being sent over the bus. For I2C we had very little issues but the Salae Logic analyzer allowed us to actually see what was going on and helped us identify different parts of the signals automatically.

5. CONCLUSION

We were able to successfully implement all three tasks for this lab. We learned a great deal about the differences in implementation of the two protocols in question, and how to use them to control both the OLED display and the accelerometer. Our Saleae waveforms confirmed our understanding of the material.