EEC 172, A03
Lab #2: Serial Interfacing Using SPI and I^2C with an OLED
Joseph Kong, Ming Cheng

UNIVERSITY OF CALIFORNIA, DAVIS
Department of Electrical and Computer Engineering
EEC 172 Winter 2019

LAB 2 Verification

Team Member 1: Joseph Kong

Team Member 2: Ming Cheng

Section Number/TA: A03 / TARA MURALI DHARAN

Demonstrate your working application to your TA:

Demonstrate the working OLED test program

Date	TA Signature	Notes
2/1/2019	Tara.M	COMPLETE

Demonstrate the Saleae logic analyzer waveform capture of SPI

Date	TA Signature	Notes
1/22/19	Tara.M	COMPLETE

Demonstrate the Saleae logic analyzer waveform capture of I2C

Date	TA Signature	Notes
1/22/19	Tara.M	COMPLETE

Demonstrate the Part II application program:
- Move the ball precisely to all parts of the OLED display by tilting the accelerometer
- Show that the speed of the ball is proportional to the accelerometer tilt angle

Date	TA Signature	Notes
1/23/19	Tara.M	COMPLETE

EXTRA CREDIT: Made a game where a random dot is generated on the screen, and everytime the ball moves on the dot, it eats it and gets bigger.

Tara.M

Objective (Goal)

The goal of this lab was to learn how to communicate with an organic light emitting diode (OLED) using SPI and to use I^2C to communicate with a Bosch BMA222 acceleration sensor in order to create a simple program where we can control the movement of a ball by tilting the launchpad. In addition to this, we learned how to check if we have the correct output using Salae USB Logic Analyzers to analyze waveforms. The equipment we used for this lab are Adafruit OLED Breakout Boards and Bosch BMA222 acceleration sensors which were already built in to our CC3200 Launchpads. Software we used included Code Composer Studio, Pin Mux Configuration tools, and a Logic Analyzer Program.

Introduction

Equipment we used that was new and specific to this lab were the Bosch BMA222 acceleration sensor, the Adafruit OLED Breakout Boards, and the Salae USB Logic Analyzers. The Bosch BMA222 acceleration sensor is built in to the launchpad, and we use I^2C protocol to communicate with the acceleration sensor. The sensor works by detecting the tilt and slope of the tilt, which it writes into registers. We can then use the I^2C protocol to read from these registers, decode the bits from the register to extract integers which we can then use in our code to program the movement of the ball. The OLED board is a hardware which we communicate with through SPI protocol. Through SPI, we can talk to the OLED board to display whatever we want it to display. Ultimately, we will display our ball rolling around the display. Finally, the Salae USB Logic Analyzers were used to to check if our pins were displaying the correct output and if the waveforms are what we expect it to be.

Methods

Below is a high level description of the steps we took to accomplish this lab:

1. Our first step was to import and build the spi_demo program on our two CC3200 Launchpads. For this we opened two instances of Code Composer Studio and two instances of the TeraTerm windows.
2. In the code, we changed one instance of the demo to master by setting `#define MASTER_MODE 1` and the other instance of the demo program to the slave by setting `#define MASTER_MODE 0`. We then configured the TeraTerm windows to the correct COM ports and the correct baud rate, then followed the instructions to run it. Based on this, we were able to study and understand the code to get a little bit better understanding of SPI and how it worked
3. Our next step was to use the Pinmux configuration tool to configure the pins on our launchpad. Some pins such as MOSI or the Clock already have pins defined by the launchpad, so the only pins we had to choose are the pins for DC, RESET, and the OLEDCS (chip select). For the DC, we chose pin 62, for the OC we chose pin 61, and for the RESET we chose pin 18.

4. We then ported open source graphics library provided by Adafruit and put it into our project. We used the spi_demo program as a base and just tweaked some things and took out the things we didn't need from the spi_demo program so that we didn't have to manually redo the SPI configuration.
5. Next, we implemented the WriteData() and WriteCommand() functions in the Adafruit_OLED.c file. When we send a command, we programmed CS and DC to both be low, then we send the command byte and read the command byte, then set CS and DC to high again. When we write data, we set DC to high and CS to low, send the data, read the data, then set the CS to high to end sending data.
6. We then used the test file and ran all the functions in the test file, which displayed things such as "Hello World", lines, circles, etc. to the OLED display board.
7. The next step was to verify the SPI waveforms using Salae logic. We opened the software that displays the waveforms from the Salae logic analyzer and figured out which color wires needed to be connected to what in order to match the correct channel with what waveform we wanted to see. Based on this, we connected the appropriate color wire from the Salae logic analyzer to the correct pins. We then ran the program and collected the sample and screenshotted the waveforms for discussion later.
8. The next step was to learn how to use the I^2C protocol. We imported the i2c_demo project and built and ran it. Using the terminal, we figured out what the accelerometer did and how it received input and how to read that input. We figured out the X, Y, and Z directions of the tilt.
9. The next step was to verify the I^2C waveforms using Salae logic. We connected 3 wires, which were the ground, I2C_SCL, and I2C_SDA which were pin 1 and 2. After running the program and collecting the samples, we took a screenshot of the waveforms we saw to be used for discussion later.
10. The last part of the lab was to create our own application program which would move a ball displayed on the OLED screen in whichever direction we tilted the launchpad. The ball would also increase or decrease in speed based on the angle of the launchpad tilt. A very high level overview of our logic is as follows: We create a ball 4 pixels wide using the fillCircle() command given to us. We set the starting position to be in the middle of the display. We create two buffers to hold the string of the commands for reading registers. We then pass this into the ParseNProcessCmd() function to get the X and Y values from the registers written to by the accelerometer. We then rescale the X and Y values to fit the OLED board, and add it to the current position of the ball. We animate the ball by alternating between drawing the circle blue and black at the same spot, so that the ball looks like it's moving. Also, in the displaybuffer() function we check to see if the most significant bit of the register is a 1. Since the register returns values in 2's complement, this will tell us whether the value should be positive or negative. Based, on this, we update the ball's position.

11. Finally, for extra credit we created a “snake” game where we start off with a small ball and try to eat another small ball we display at random places on the screen. Everytime the moving ball touches/eats the small ball, it gets bigger.

Discussion

Below is an image of the SPI waveforms captured by our Salae logic probe using the SPI protocol analyzer.

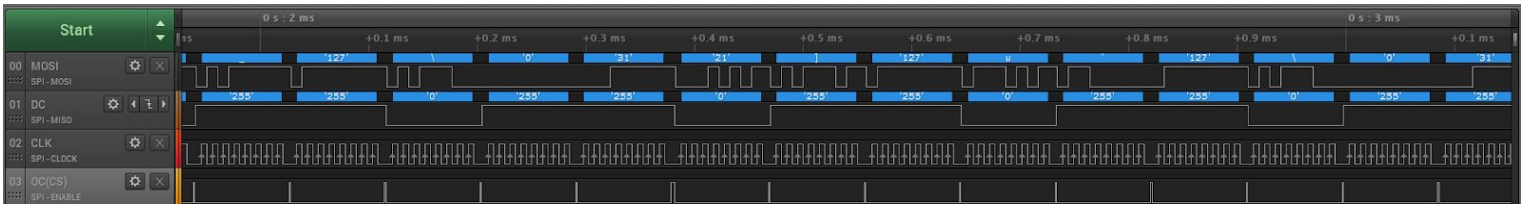

Figure 1: SPI waveforms

We are using Master mode for SPI since we are using our board as a “master” to send data and command and control the OLED as a “slave”.

These waveforms are what we expected to see. As shown by the TA during the lab, when CS (chip select) and DC (data command) were low, that means it’s a command. When DC is high, that means it is writing the data after the command. This is exactly what the waveform shows us. First of all, as we can see from the clock waveforms, every command is 8 cycles long. So as seen in figure 1, every eight cycles the CS toggles, and when it is a command, the DC toggles every eight cycles. The data takes two cycles of eight to process the entire data command. We can see that when both CS and DC are low, that means it’s a command which is 8 cycles long. Then DC toggles high, and then the data goes through. The MOSI waveform is simply the data that is being transmitted. These waveforms correspond to the calls to WriteData(unsigned char data) when DC is high, or WriteCommand(unsigned char command) when DC is low in our code. In these functions, we use GPIOPinWrite to the specific pin to control CS to indicate the start and end of writing data (CS low -> start, CS high -> end), and DC to indicate it is a write-command or write-data.

In figure 2 and 3 below, we show the X and Y I^2C waveforms respectively.

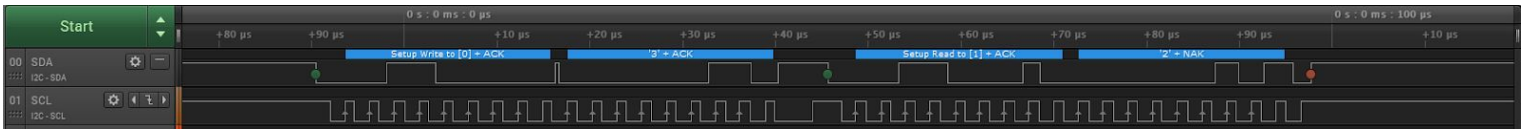

Figure 2: X component of I^2C waveform (readreg 0x18 0x3 1)

In ProcessReadRegCommand function, it passes the device address and the register address that correspond to X-axis, which are 0x18 and 0x3 respectively to I2C_IF_Write

function to write the register address to be read from. Then it passes the length of data to be read to I2C_IF_Read to read the actual data. The first waveform that says “Set up write to [0] + ACK” is showing that the program is passing the device address which is 0x18 from the command through I2C to be read from. The second waveform that says “3 + ACK” is showing that the program is passing the register address that corresponds to X-axis to be read from, which is 0x3. The third waveform that says “Setup read to [1] + ACK” is showing that the program is passing the data length to be read, which is 1 from our readreg command. The final waveform is showing the result of the readreg command, which is 0x2 as we saw after running this command. Therefore, we expect to see that the data of length 1 from the device in the address 0x18, register offset address 0x3(corresponds to X-axis) is read, and the result is 0x2 from the waveform.

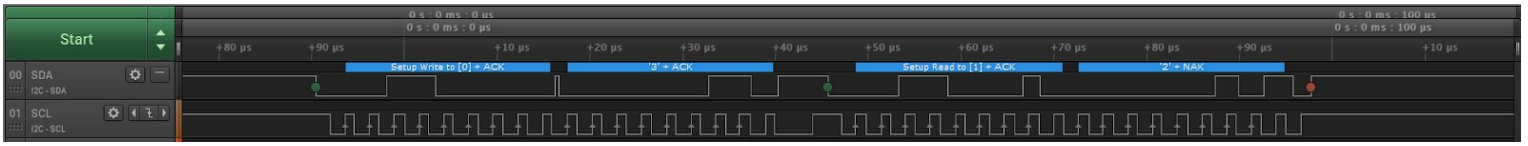

Figure 3: Y component of I²C waveform (readreg 0x18 0x5 1)

For Y component, it is the same as X component but we are reading from a different register offset address(0x5). The first waveform that says “Set up write to [0] + ACK” is showing that the program is passing the device address which is 0x18 from the command through I2C to be read from. The second waveform that says “5 + ACK” is showing that the program is passing the register address that corresponds to Y-axis to be read from, which is 0x5. The third waveform that says “Setup read to [1] + ACK” is showing that the program is passing the data length to be read, which is 1 from our readreg command. The final waveform is showing the result of the readreg command, which is 255(0xff) as we saw after running this command. Therefore, we expect to see that the data of length 1 from the device in the address 0x18, register offset address 0x5(corresponds to Y-axis) is read, and the result is 0xff from the waveform.

One of the main challenges we faced was initially getting the OLED to display anything at all. The entire debugging process was frustrating but also a learning experience. First we figured out that the pins we initially tried to use weren’t working, and that forced us to consult the manual to see which pins could be used. The biggest problem was finding out that in our pinmux configuration, we forgot to set the pins to be output pins which was one of the main reasons we couldn’t get anything to display. We also found out that one of our OLEDs wasn’t working. At least from all those challenges, we were forced to take a closer look at how the SPI was configured and how it worked. After we got the OLED to display things, we didn’t have many other problems getting through the rest of the lab. One of the more fun parts of the lab was doing the extra credit game at the end because we could be creative and create whatever game

we wanted. We ended up doing a ball version of the game “snake” where we had a ball that would grow bigger and bigger.

Conclusion

Although we were a bit behind because we couldn’t get the OLED to work after two lab sessions and the open lab session, we managed to figure out the problem and complete the lab in time. In this lab we learned how to use SPI to talk to a display and how to use I^2C to talk to the built in accelerometer that we can read values from and manipulate to create fun stuff on the OLED. We also learned how to analyze and confirm our inputs and outputs using the Salae logic analyzers.