# EEC172 Lab 1 Report

Name: Chen Chen SID: 998820114

Name: Weidong Wu SID: 912341051

## Introduction

In this lab, we used SPI to interface the CC3200 LaunchPad to a color OLED display. We also used I2C to communicate with an on-board Bosch BMA222 acceleration sensor. Our application program detects the XY tilt of our board based on BMA222 readings, and uses the tilt to control a ball on our OLED display. We also used a Saleae logic probe to capture and display SPI and I2C signal waveforms. We then explained those waveforms with our codes and SPI/I2C communication protocols. In our report, we will explain the goals and procedures for each part of the Lab thoroughly.

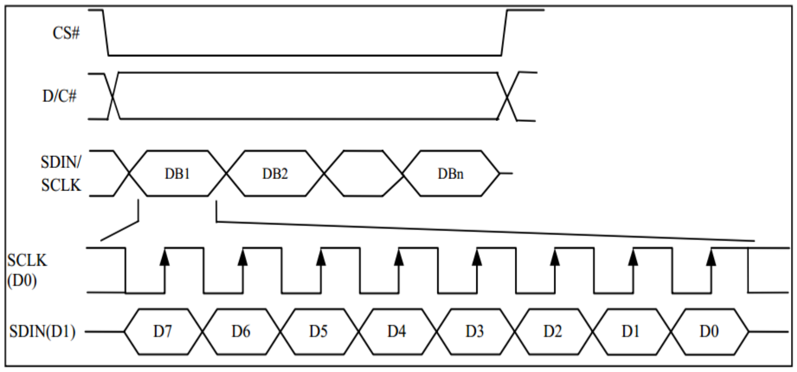
## Methods and Procedures

### Part I. Interfacing to the OLED using the Serial Peripheral Interface (SPI)

We first studied the spi_demo example and found out how to use the functions to initialize and communicate through SPI channels. We also used Saleae logic probe to capture a wave form to confirm our understanding of the SPI communication protocol. We finally had the programming running on two boards and they were able to communicate with each other through SPI channels. We also strengthened our understanding of UART terminals which came in handy later when we were debugging for our application program.

After we fully understood the spi_demo example, we started to make sense of the libraries and headers that comes with the Adafruit OLED. We downloaded the data sheet for the unit and read about the wave form behaviors of the OLED in order to properly communicate with it. Figure 8-5 is a very important graph that helped us to figure out how to initialize the SPI communication.

**Figure 8-5 : Write procedure in 4-wire Serial interface mode**

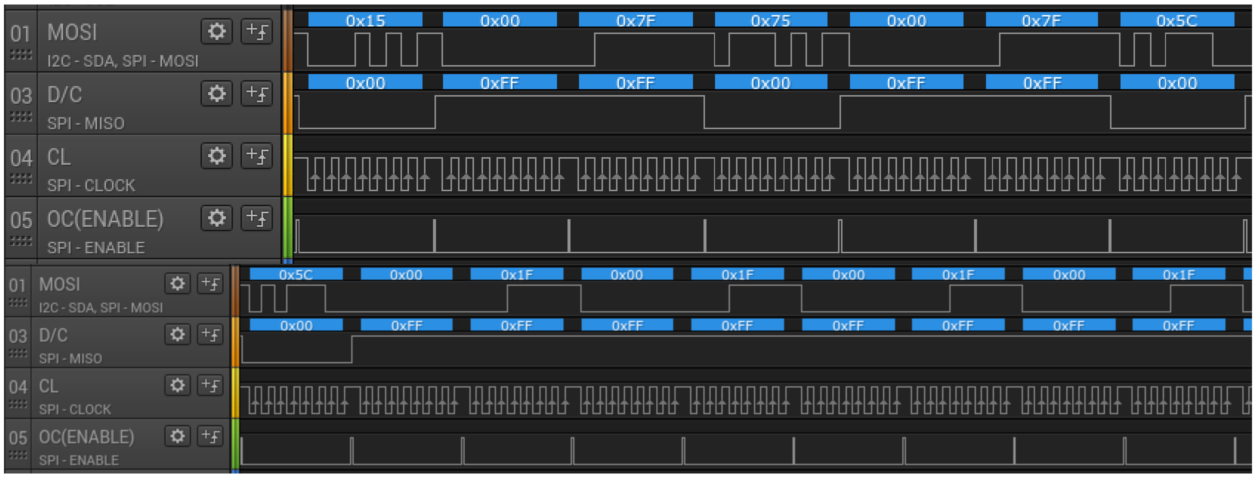


From Figure 8-5 we knew that the CS is active when pulled low and SCLK is initially high (active low). We also knew that data are captured at the clock's rising edge and transmitted at falling edge. All these evidents pointed out that clock polarity (CPOL) = 1 and clock phase (CPHA) = 1, which corresponds to SPI mode number 3 in the case of cc3200. So we configured the SPI communication at the start of our program, and we put SPI_CS_ACTIVELOW and SPI_SUB_MODE_3 into the MAP_SPIConfigSetExpClk() function. One other thing that is worth noting from Figure 8-5 is that the OLED unit will interpret the input signal as Data when the D/C is high and as Commands when the D/C is low. This information is crucial when we were writing codes for writeCommand() and writeData() functions.

After we read about the OLED data sheet, we moved on to implement writeCommand() and writeData() functions. Those functions are similar to those in spi_demo, and the only difference is that we need to make sure that D/C pin is pulled correctly in each case. In our program we set PIN 15 to be our D/C pin.

So the OLED screen eventually worked and we quickly tried all the test cases without any difficulties. The following screen shot was the wave form taken by our Saleae logic probe sampling over the SPI channels when we ran the tests.

**SPI Waveforms for fillScreen(BLUE)**



We found that the wave form gives almost all the information about the SPI communication. The screen shots were taken over the fillScreen(BLUE) function, which actually calls a fillRect() function to get the job done. All relevant codes and MACRO values can be found in the Appendix or from the Smartsite. The following are the parts of the code that were relevant to SPI communication.

```
fillRect(0, 0, SSD1351WIDTH, SSD1351HEIGHT, BLUE)

// set location

writeCommand(SSD1351_CMD_SETCOLUMN);

writeData(x);

writeData(x+w-1);

writeCommand(SSD1351_CMD_SETROW);

writeData(y);

writeData(y);

// fill!

writeCommand(SSD1351_CMD_WRITERAM);

for (i=0; i < w; i++) {

        writeData(color >> 8);

        writeData(color);

}
```

We can see that the wave form reflects exactly the information in our code, 1 command followed by 2 data, then 1 command followed by 2 more data, etc. We also confirmed that the values of the wave form match the actually parameters of the functions too.

**Part II. Using I2C to communicate with the on-board BMA222 accelerometer**

We moved on to use I2C channels to interface with BMA222 accelerometer. This part of the lab was relatively easier. We simply understood the data sheet, used the data from polling and came up with ways to translate raw data onto pixal behaviors. The goal was to drive the ball to accelerate proportional to the degree of tilt at any direction in x-y plane.

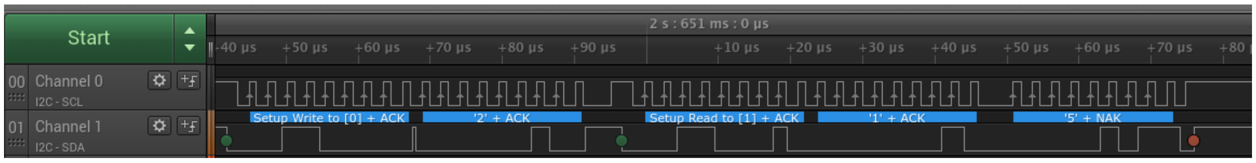
First and foremost, we spent time to fully understand the i2c_demo example so that we know how to use the functions in the header to interface with I2C devices. We then read through Section 5.4, 5.2 and 4.4 of the BMA222 Data sheet to understand where are those acceleration information. The next thing we did was to utilize UART terminal to print the acceleration data to our terminal. It always helps to visualize the data. Here is the part of Section 5.2 that was helpful for us to understand the locations of the acceleration data.

|  |  |  |  |
|---|---|---|---|
| 0x07 | 0x00 | acc_z<7:0> |  |
| 0x06 | 0x00 | *0* | new_data_z |
| 0x05 | 0x00 | acc_y<7:0> |  |
| 0x04 | 0x00 | *0* | new_data_y |
| 0x03 | 0x00 | acc_x<7:0> |  |
| 0x02 | 0x00 | *0* | new_data_x |

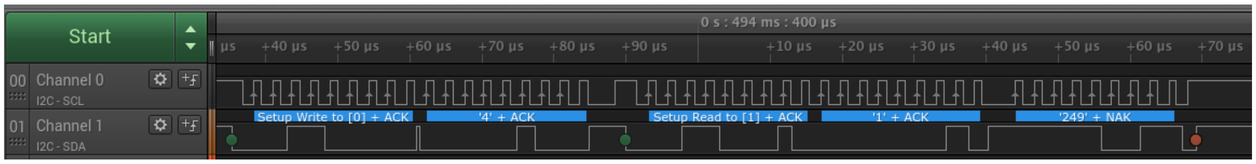
After we knew the addresses of the acceleration data, we wrote functions to poll those data via I2C channels, and use those data to update the ball's position with the drawing functions that are available to us after completing Part I. All the detailed implementation can be found in the Appendix or from the Smartsite.

**I2C Waveforms**

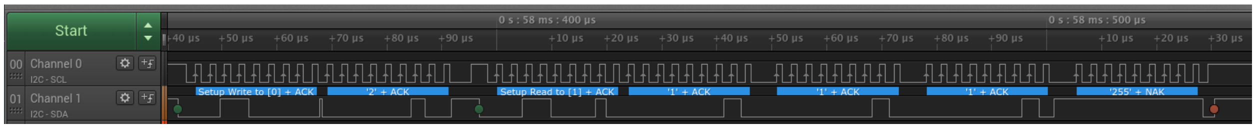
Fetch x-axis data:



Fetch y-axis data:



Fetch both x-axis and y-axis data together:



Like those wave forms we observed in SPI communications, I2C waveforms carry many information as well. With the help of I2C analyzer we could actually see all the parameters were pushed through the I2C channel. We could even see that the actual readings from BMA222 device. The general format of communication is: Write to [0], offset, read to [1], followed by the number of data register we requested. Notice that if we fetch data for x-axis and y-axis separately, we have 2 data coming back after the read request. In our code, we got all 4 of them by fetching the data all at once. Listed below are the relevant codes in our program that corresponds to the above waveform.

```
int fetchAccelerationData(unsigned char* aucRdDataBuf)
    I2C_IF_Write(ucDevAddr,&ucRegOffset,1,0);
    I2C_IF_Read(ucDevAddr, &aucRdDataBuf[0], ucRdLen);
```

## Challenges

When we received the Lab1 starter files, we were very confused about what each function and structure stands for. By looking at the data sheets for OLED screen and cc3200, we eventually managed to figure out the meaning of each function. Another challenge which took us majority of our time was the implementation of writeCommand() and writeData(). We figured it out by reviewing the data sheet posted on Smartsite and the SSD1355 Data Sheet we found on the manufacturer's website. We learned to differentiate data and command by flipping D/C pin. When configure SPI clock in the main function, we were not sure which SPI Mode we should choose. After finding out about the SPI polarity and phase settings in data sheet, we then figured out that the mode should be 3 (Polarity = 1, Phase = 1) through the wave form. The application program was relatively less challenging because we had had a better understanding of how to interface with different modules. The most challenging part of the ball application was translating from raw data into behavior on screen.

## Conclusion

After finishing this lab, we have learned how to interface with the OLED using the SPI, how to use I2C to communicate with the on-board BMA222 accelerometer, and how to capture waveforms using Saleae logic probe. We also combined those knowledge and implement a application program at the end.

## Reference

CC3200 Peripheral Driver Library User's Guide

CC3200 SimpleLink™ Wi-Fi and Internet-of Things Solution, a Single Chip Wireless MCU Technical Reference Manual

CC3200 SimpleLink™ Wi-Fi® and Internet-of-Things Solution, a Single-Chip Wireless MCU Data Sheet

BMA222 Data Sheet

SSD1355 Data Sheet