# EEC 172 Lab 2 Report

Griffin Kimura
gykimura@ucdavis.edu

Armand Nassari
aanasseri@ucdavis.edu

## 1.INTRODUCTION

In this lab, we were tasked with completing 3 separate parts that would constitute the overall lab. Our first task was to use the Serial Peripheral Interface (SPI) to interface the CC3200 LaunchPad to a color OLED display. Our next task consisted of using the Inter-Integrated Circuit bus to communicate with the on-board Bosch BMA222 acceleration sensor. In addition to both these tasks, we needed to use a Saleae logic probe to capture and display SPI and I^2C signal waveforms.

## 2.BACKGROUND

In this lab, there were three main phases that consisted of completing different objectives. For part 1, we needed to interface the Organic Light Emitting Diode (OLED) using SPI. This consisted of us downloading the SPI project on two CC3200 LaunchPads and us understanding the provided API. Once this was setup, we would have to verify the waveforms using a Saleae logic probe. For part 2, we would need to use the I^2C to communicate with the accelerometer by testing the i2c_demo project on our CC3200 LaunchPad and verify the waveforms. Part 3 consisted of us creating an application program that would move a small ball across a screen based on the accelerometer pitch and roll orientation angles.

## 3.GOALS

Our goals were to successfully interface the OLED using SPI, establish communication with the accelerometer, and to ultimately develop and run a program that would use the accelerometer register values to move across the screen accordingly using I^2C. We were able to understand how SPI and I^2C protocols send data and how the OLED from Adafruit can be configured.

## 4.METHODS

## 4.1. Part I

#### 4.1.1

In part 1, we launched two CCS projects for master and slave. We used the spi_demo project for both. We wired up both of our CC3200 boards according to the project demo configuration. The spi_demo project had a macro for MASTER_MODE allowing us to easily change the project to become the master project. To accomplish this, we flashed the master project to one board and build the slave on CCS. We launched two TeraTerms at the appropriate COM ports and sent text from the master terminal to display on the slave terminal.

#### 4.1.2

In this sub part, we referred to the table of GPIO signals so that we could properly connect the OLED to the CC3200. We needed to write the writeData/Command functions in order to use the Adafruit API. Since we used a GPIO for CS instead of the designated SPI_CS on the board, we needed to manually toggle the pin low when sending data or commands. The Write functions used SPI to send information to the board. We adapted our demo code to call all the library functions in main so that we could see their outputs on the OLED. We included a small delay between calling each function so that each pattern produced could be identified easily.

#### 4.1.3

In this final sub part, we verified our waveforms produced from the board using the logic analyzer. Once we launched the Saleae logic application, we captured the waveforms in order to show how writes and reads were performed. Specifically, we analyzed the waves produced from the Clock, DC, Enable, and MOSI.
To

## 4.2 Part II

### 4.2.1

To begin this first part, we uploaded and tested the provided i2c_demo project on the CC3200 LaunchPad. We ultimately needed to understand the register values produced by the accelerometer given in an 8 bit two's complement representation. We experimented with this given acceleration data by tilting the board at multiple angles and printed out the values from the given registers (0x03) acc_x, (0x05) acc_y and (0x07) acc_z.

### 4.2.2

We verified our waveforms produced from the board using the logic analyzer. Once we launched the Saleae logic application, we captured the waveforms in order to show how reads and writes were performed.We captured the x-axis data as well as the y-axis data and took screenshots of the I^2C waveform.

## 4.3 Part III

In the last part, we needed to develop an application program that would use the accelerometer to move a small ball across the OLED display. In our program, we called the fillCircle() function to draw a ball onto the center of the screen using a radius of 4 pixels. We also had to remove the old circle from the OLED with a second fillCircle(). In order to track the accelerometer register values, we called the I2C_IF_Read() and I2C_IF_Write() functions to take in the register values produced. This allowed us to map both the location and speed of the ball on the board so that it would move when the board was tilted. The accelerometer values were modified from 8-bit two's complement to add or subtract from the current position of the ball based on the axis. Additionally, we added border cases to account for the cases when the ball approached an edge and simply limited it from crossing.

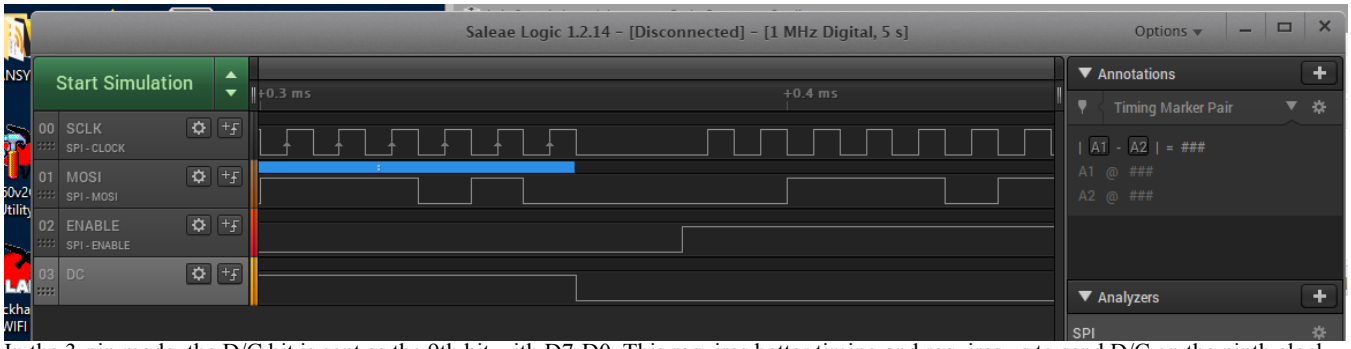
## 5.DISCUSSION

In this lab we encountered several any major issues that required a significant amount of time to be spent fixing them. One issue was the fact that both of our boards had hardware problems. The accelerometer of one of the boards did not function since the values produced were always 255 in the x-axis. However, the board with the working accelerometer had a faulty cable that caused unstable connections and in addition the OLED screen was cracked. We also had several debugging issues with our code when it came to moving the ball across the screen. The ball would either stay stationary, produce a trail, or crash the entire computer altogether, which caused several setbacks. To resolve these issues, we used different pin configurations and experimented with the code to see how to ball behaved on the screen.

## 6.CONCLUSION

We now have a basic understanding of how to interface our programs with the CC3200 and the OLED display and use the built-in accelerometer. In the end, we were able to successfully utilize the provided API functions, which were highly useful and necessary for the board. We also have a good understanding of how SPI and I^2C send information as well as analyzing the waveforms produced using the Saleae logic analyzer.

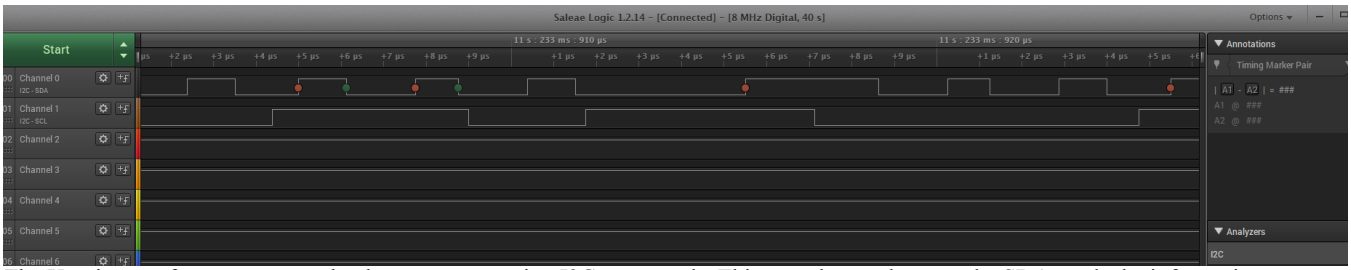
SPI Waveform:




In the 3-pin mode, the D/C bit is sent as the 9th bit with D7-D0. This requires better timing and requires us to send D/C on the ninth clock cycle. Therefore, we chose SPI 4-pin mode, this allowed us to include D/C as a separate pin rather than using the 3-pin mode. It was much simpler to implement with the SPI API we were given.

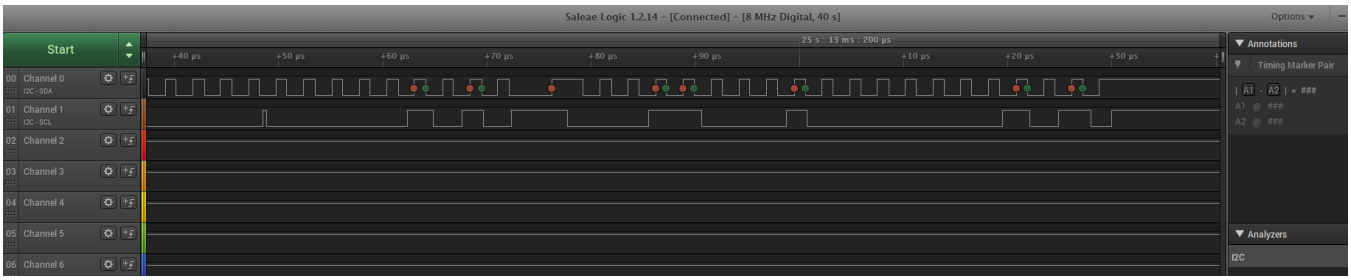
When we want to send data or a command, we must set the enable pin (CS) to low while sending the data. Afterwards, we set the pin high and wait for another command. The MOSI sends the data/command. And SCLK is our SPI clock.

X-axis:




The X-axis waveform represents the data sent to us using I2C commands. This was shown above as the SDA sends the information to us from the micro controller when we use the I2C_IF_Read command.

Y-axis:




In our program, we read all three register values with one I2C_IF_Read command. This allows us to receive 6 characters of information from the I2C bus in one process.