Lab2: Serial Interfacing Using SPI and I^{2}C

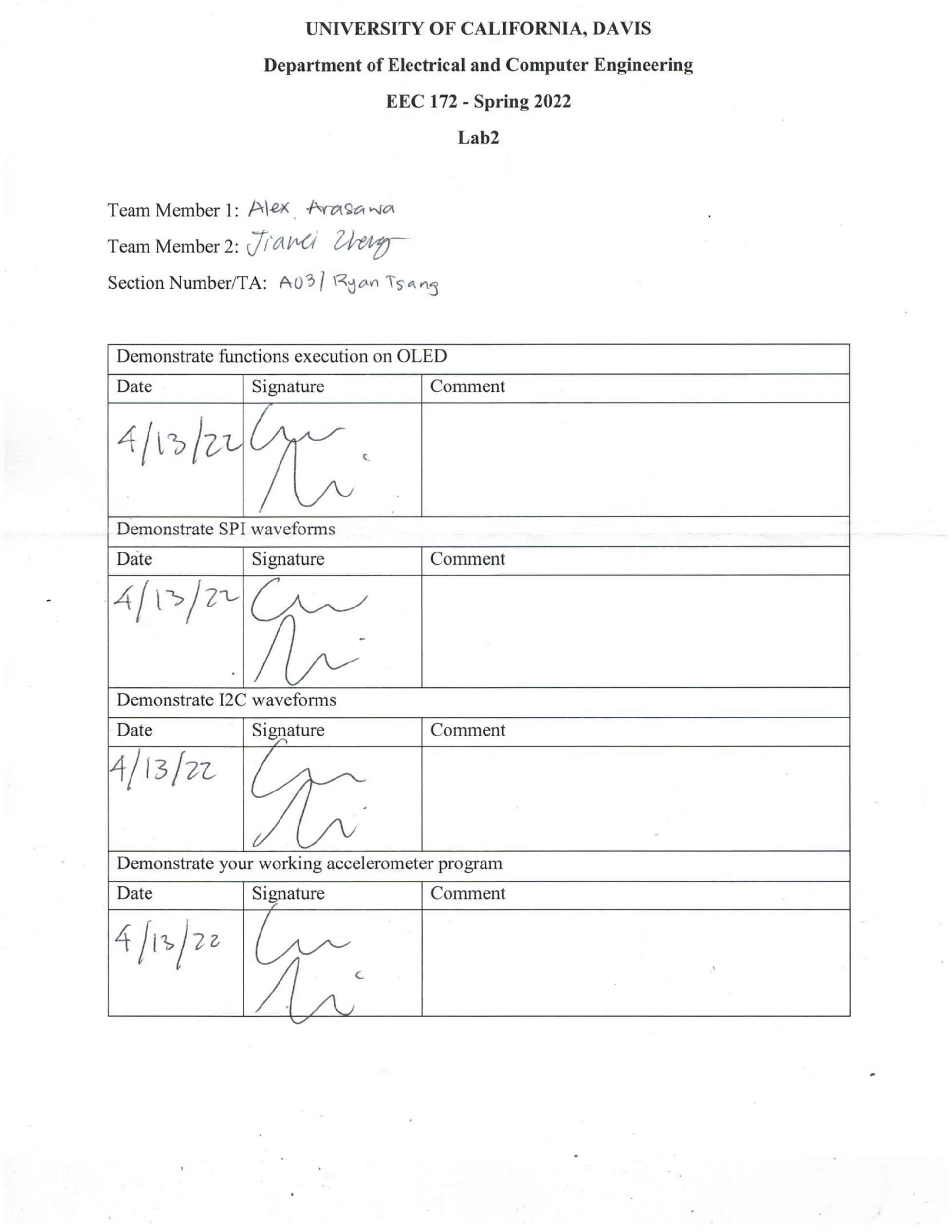
Alexander Arasawa, Jiawei Zheng

June 11, 2022

EEC 172

Professor Soheil Ghiasihafezi

1 Check off from TA:



UNIVERSITY OF CALIFORNIA, DAVIS

Department of Electrical and Computer Engineering

EEC 172 - Spring 2022

Lab2

Team Member 1: Alex Arasawa

Team Member 2: Jianci Zheng

Section Number/TA: A03/ Ryan Tsang

Demonstrate functions execution on OLED		
Date	Signature	Comment
4/13/22	[illegible]	

Demonstrate SPI waveforms		
Date	Signature	Comment
4/13/22	[illegible]	

Demonstrate I2C waveforms		
Date	Signature	Comment
4/13/22	[illegible]	

Demonstrate your working accelerometer program		
Date	Signature	Comment
4/13/22	[illegible]	

2 Introduction

The first goal of this lab is to learn, through SPI_demo, how to use SPI to interface the CC3200 LaunchPad to a color OLED panel. In addition, we will also learn, through i2c_demo, how to use I^2C to communicate with the Bosch BMA222 acceleration sensor built within the CC3200 LaunchPad. Then we will write an application program combining what we learned to detect the orientation tilt of the board using the BMA222 readings and use that tilt information to control a ball on the OLED display. We will also be using Saleae logic analyzer in this lab to capture SPI and I^2C signal waveforms so we can verify the commands that are sent from the CC3200 LaunchPad.

3 Background

CCS software is an IDE for configuring TI's microcontrollers and embedded processors. The software contains many tools to help with creating and debugging applications: CSS includes an optimized C/C++ compiler, a code editor, a project build environment, a debugger, and many other features.

CCS Uniflash is a standalone tool used to program flash memory of TI's microcontrollers and program flash memory of Sitara processors.

TI Pin Mux Tool is a software used for configuring pins, peripheral signals, and other components of a system.

Saleae logic is a logic analyzer software that is used to capture waveforms and debug ports/programs.

References:

`https://www.ti.com/tool/CCSTUDIO`

`https://www.ti.com/tool/UNIFLASH`

`https://www.ti.com/tool/SYSCONFIG`

`https://www.saleae.com/`

4 Goals/Objectives

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

4.1.1 Implement SPI_demo project on two CC3200 LaunchPads

Run two SPI_demo projects on separate CC3200 LaunchPads and have them communicate with each other.

What you should learn from this task:

- How the SPI_demo program works.
- How is SPI utilized(what functions are called) to enable communication between the master CC3200 LaunchPad and the slave CC3200 LaunchPad.

4.1.2 Implement OLED interface using SPI

Interface the CC3200 LaunchPad to a OLED panel and display things on the OLED.

What you should learn from this task:

- What ports are actually available on the CC3200 LaunchPad.
- By implementing writeCommand() and writeData() in Adafruit_OLED.c, learn how to send command and data bytes to the OLED panel via SPI.
- What commands are used to manipulate pixels on the OLED display.
- How to use the Adafruit API to satisfy the criteria for the test program.

4.1.3 Verifying SPI waveforms using a Saleae logic

Use Saleae logic to capture chip select, dc, clock, and mosi signals.

What you should learn from this task:

- How are the waveforms related to the code for OLED interface.

4.2 Part II. Using I^{2}C to communicate with the on-board BMA222 accelerometer

4.2.1 Implement i2c_demo project on a CC3200 Launchpad

Run the i2c_demo project on the CC3200 LaunchPad, and prompt acceleration data from the Bosch BMA222 acceleration sensor through usage of request commands in the terminal.

What you should learn from this task:

- How does the i2c_demo program work.
- How is I^{2}C utilized to collect acceleration data from the Bosch BMA222 acceleration sensor. Focus on functions within the ProcessReadRegCommand() and especially those that are related to I^{2}C.

4.2.2 Verifying I^{2}C waveforms using Saleae logic

Use Saleae logic to capture SCL and SDA signals when read register for x and y tilt commands are called.

What you should learn from this task:

- Differences in protocols between I^{2}C and SPI.
- How are the waveforms related to the code in the i2c_demo project.

4.2.3 Application Program

Simulate a realistic ball on the OLED.

- How to collect x and y orientation data from the Bosch BMA222 acceleration sensor using I^{2}C and I^{2}C commands.
- How to use accelerometer data to manipulate a ball on the OLED in real time.

5 Methods

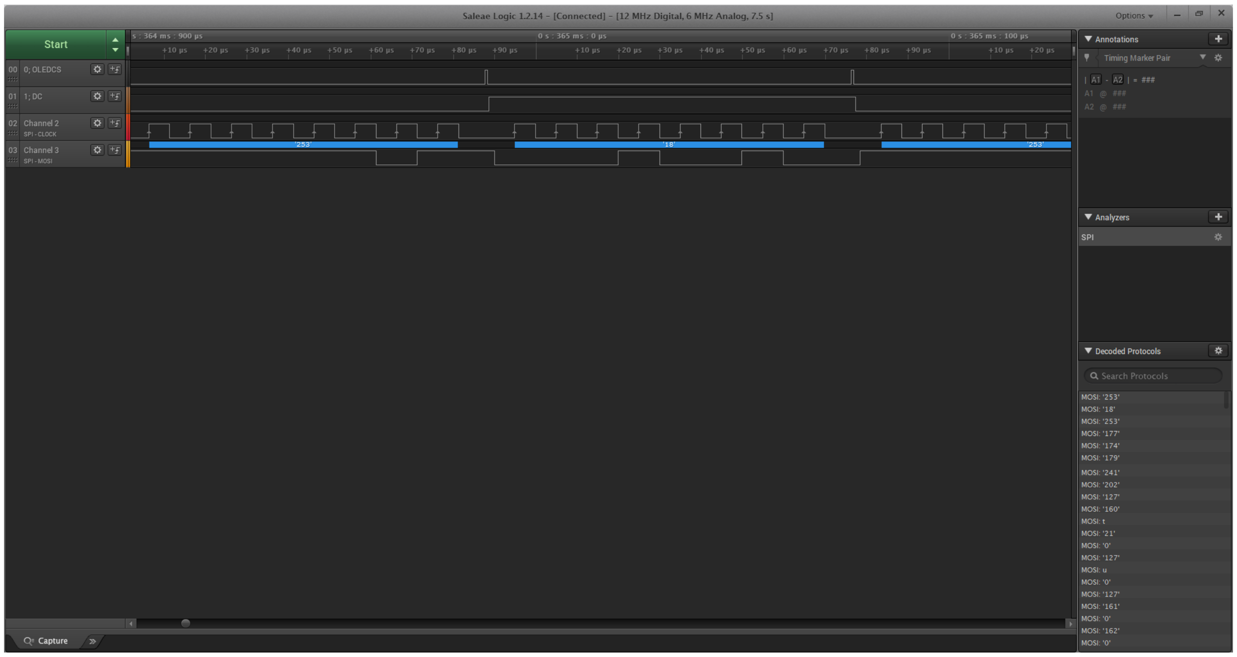
5.1 Part I. Implement SPI_demo project on two CC3200 Launchpads

First, we implemented the SPI_demo program by creating separate workspaces for the two Launchpads. Setting the 'MASTER_MODE' setting in one of the Launchpad's to 0 denoting it as the SPI slave; the other as 1 to denote SPI master. After flashing the boards, we opened two terminals for the different COM ports for each of the Launchpads. Then, we verified the program worked by typing into the master terminal and looking for a response on the slave terminal.

5.2 Part I. Implement OLED interface using SPI

We looked at the pin maps for the CC3200 and OLED to figure out which pins to put the jumper cables on. We then connected the OLEDCS, DC, Reset, Vin, and GND pins on the OLED to GPIO pin 61, GPIO pin 62, GPIO pin 18, GSPI_CLK, and GSPI_MOSI pins on the CC3200 after pin mux tool configuration(those GPIO ports were some of the ones available for free usage). We figured out what to set the OLEDCS(chip select) and DC pins to for writeCommand() and writeData() by looking through the SSD1351 documentation (Table 8-5). We found out that OLEDCS should be set low before writing command/data bytes to the OLED through SPI. Then the DC signal had to be set 0 in writeCommand() and 1 in writeData() to signify whether the byte send was command or data type. Then, we would enable SPI and put the data/byte on the line with SPI commands. At the end, we write high to OLEDCS to reset the state of the system and disable SPI protocol to prevent the OLED from reading in garbage. In terms of selecting what SPI mode, we chose the 4-wire configuration because the CC3200 is a 4-wire interface. Additionally, have a 4-wire configuration allows for simultaneous sending and receiving of data.

5.3 Part I. Verifying SPI waveforms using Saleae logic



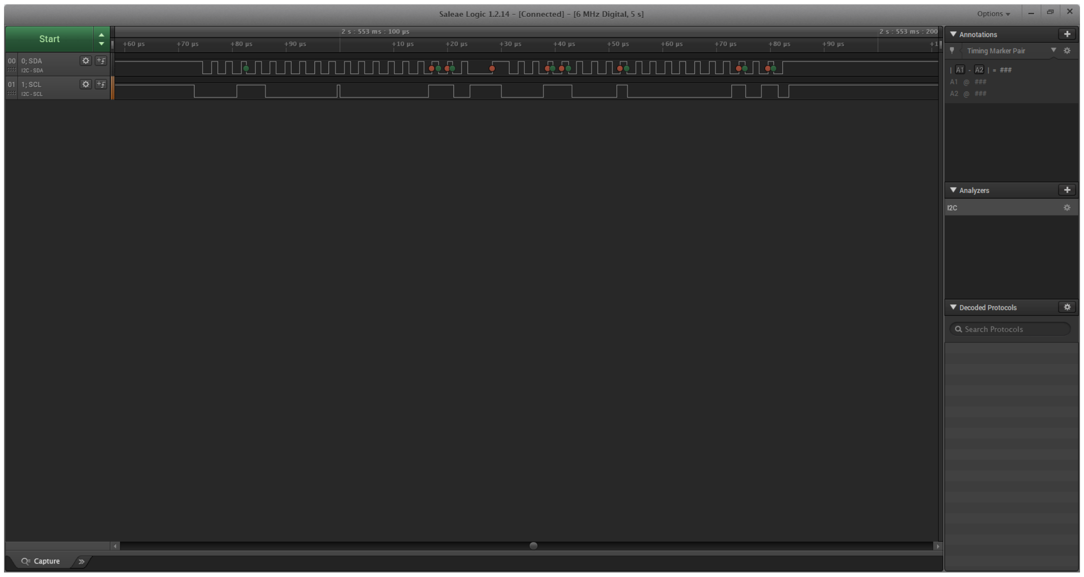
```
MOSI: '253'
MOSI: '18'
MOSI: '253'
MOSI: '177'
MOSI: '174'
MOSI: '179'
MOSI: '241'
MOSI: '202'
MOSI: '127'
MOSI: '160'
MOSI: t
MOSI: '21'
MOSI: '0'
MOSI: '127'
MOSI: u
MOSI: '0'
MOSI: '127'
MOSI: '161'
MOSI: '0'
MOSI: '162'
MOSI: '0'
```

We had to verify that the SPI waveforms were correct by hooking up the Saleae logic analyzer to pick up signals from the Launchpad to the OLED. We configured the SPI analyzer on the Saleae software and this is the waveform that we got. The first half corresponds to the first call to writeCommand() in the initialization sequence of Adafruit_Init() because both OLEDCS and DC was set low. Then, you see a small time frame of OLEDCS going back high to reset the state of the system before it goes low again. The second half, corresponds to the call to writeData() that comes after the first writeCommand() in the initialization sequence of Adafruit_Init(). Here, OLEDCS is set low and DC is set high to signify that it was a data byte that is being send.

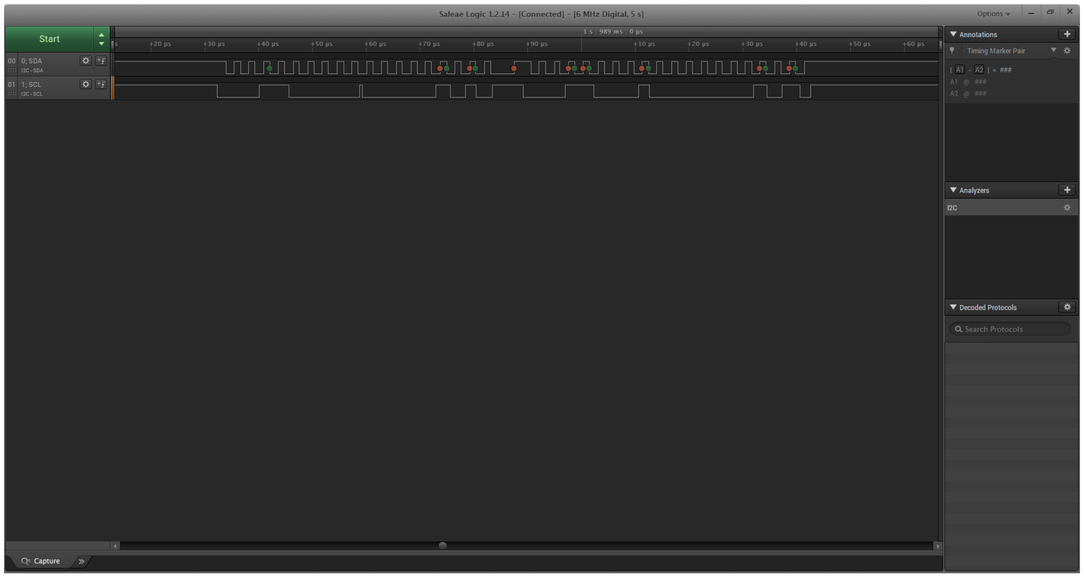
5.4 Part II. Implement i2c_demo project on a CC3200 Launchpad

We implemented the i2c_demo project by creating a workspace for it. Then we copied over the demo project and ran it. Then we typed readreg commands in a terminal window to read the acceleration data for all three axes from the Launchpad.

5.5 Part II. Verifying I^{2}C waveforms using a Saleae logic



This is the signal we got for SCL and SDA when reading acceleration data for x.



This is the signal we got for SCL and SDA when reading acceleration data for y.

These two waveforms correspond to I2C_IF_Write() in ProcessReadRegCommand when we call on readreg commands from terminal. We see a slight variation between the two commands in the middle because &ucRegOffset is 0x3 for x axis and 0x5 for y axis.

5.6 Part II. Application Program

First, we went through the main to decide where we were going to place the code for the ball and acceleration logic. We got the dimensions for the OLED and divided it in half to to place the ball at the center of the screen when the program initializes. Then we enter a forever while loop. Variables x and y is the position of the ball on the OLED panel. Variable x_offset and y_offset are the x and y tilt values read from the accelerometer via I2C_IF_Write() and I2C_IF_READ(); these two variables are used for offsetting and predicting the new position of the ball on the OLED. Then using x, y, x_offset, and y_offset, we calculate the new position of the ball. If it stays in frame, we update the ball to its new x and y position. Toward the end of the while loop, we quickly flash the ball with fillCircle() and then erase it with fillCircle() by using the color of the background.

6 Extra Credit

I modified the application program to become a ball game. Using rand() from c libraries, I created random size food circles at random positions on the OLED panel. When the user's ball get close to the food, calculated using absolute value, the user's ball consumes the food and gain 1 point. When the user gain 3 points, the user wins and get the message "YOU WIN!!!!".

7 Discussion

The first problem we encountered was running the SPI_demo. We tried to run it several times and each time we got the error that we could not run multiple Launchpads at once. After getting help from the TA we decided to flash both the CC3200's(one with master code and the other with slave code) and it worked.

The second problem was that, initially, we could not get anything to display on the OLED. Our first configuration of the pins was using the GPIO on pins 15, 16, and 26. After reading the Adafruit_OLED.c, we found out that the Adafruit_Init() was written using pin 18 for Reset, so we changed pin 26 to 18. We tested the program and nothing showed up on the OLED. The TA helped us again and there was no errors to be found on our hardware configuration so we tested the OLED by executing fillScreen(WHITE). This worked for 0.5 seconds and then never again; this phenomenon was probably due some garbage values on the pins that were not properly flushed out on reset. Then we looked through "Launchpad Pin-out Nuances.txt" and found out that we have been trying to use pins that were already being used by the launchpad. We changed the GPIO pins to 61 and 62 for the OLEDCS and DC, respectively, and it finally worked.

8 Conclusion

This lab helped in learning how to use the OLED, on-board components like the BMA222 accelerometer, and Saleae logic analyzer. We learned a couple of new communication protocols and how to analyze the waveforms of both in this lab. First, how to take advantage of the SPI protocol to interface the CC3200 LaunchPad to a color OLED panel. Then we had to hook up the Saleae logic analyzer in between the CC3200 and OLED to interpret the waveforms and learn how they corresponded to commands in the code. Second, how to take advantage of the I^2C protocol to learn how to use the on-board BMA222 accelerometer. Much like with SPI, we analyzed waveforms measured by the logic analyzer to map them onto the commands in our code. Later we used the knowledge we gathered to create an application that communicates between the accelerometer and the OLED; Using axes or tilt data to manipulate a ball on the OLED.