UNIVERSITY OF CALIFORNIA, DAVIS

Department of Electrical and Computer Engineering

EEC 172 - Spring 2022

Lab2

Team Member 1: Steven To

Team Member 2: Kunal Pai

Section Number/TA: A03

Demonstrate functions execution on OLED		
Date	Signature	Comment
4/11	[illegible]	
Demonstrate SPI waveforms		
Date	Signature	Comment
4/11	[illegible]	
Demonstrate I2C waveforms		
Date	Signature	Comment
4/13	[illegible]	
Demonstrate your working accelerometer program		
Date	Signature	Comment
4/13	[illegible]	

EEC 172 Lab 2 Report

Steven To, Kunal Pai

***Introduction*:**

The objective of this lab was to explore the serial communication protocols SPI and I^{2}C by using the TI example programs “spi_demo” and “i2c_demo” After observing these demo programs and their respective API’s, we developed two application programs. The first program involved the use of SPI to communicate between the CC3200 with an OLED to present a sequence of patterns and displays. The second program involved the combination of SPI to display a ball on the OLED and I^{2}C to obtain tilt readings from the BMA222 accelerometer on the CC3200 that will alter the ball’s position on the OLED. While using SPI and I^{2}C, we used a logic analyzer to examine the waveforms produced as data was transferred between the controller and the peripheral.

***Background*:**

SPI is a serial communication protocol that allows the transmission of data between a controller and one or more peripheral devices. It involves the use of four data lines: MOSI which allows the controller to transmit data to the peripheral, MISO which allows the peripheral to transmit data to the controller, SCLK which transmits the clock signal from controller to peripheral, and CS which allows the controller to select which peripheral to communicate with.

I^{2}C is a serial communication protocol similar to SPI that allows for data transmission but possesses key differences. Instead of having four data lines, it has two: one line (SDA) for the bidirectional data transfer between controller and peripheral and another line (SCL) for the clock signal to transmit from controller to peripheral. Furthermore, I^{2}C allows for multiple controllers to communicate with multiple peripheral devices.

The OLED is a 1.5” display consisting of 128x128 RGB pixels, with each pixel having 16 bits of resolution to allow for high contrast and a wide range of vivid colors. It uses a SSD1351 driver chip that manages the display through SPI. Adafruit also provides the OLED with an API of functions that can be used to display patterns and text (Because it was originally written for Arduino, it needs to be ported to the CC3200 through the modification of low-level functions).

The Saleae logic analyzer is a tool that allows for the debugging and verification of waveforms and signals of digital circuits and systems. This is especially useful for ensuring that the data bits transmitted for SPI and I^{2}C are correct.

The Bosch BMA222 is an accelerometer that is built into the CC3200 Launchpad. It is a 2 mm by 2 mm acceleration sensor that can detect a wide range of motions for a multitude of applications. For our purposes, we are interested in the accelerometer's ability to detect tilt in the x and y axes. This acceleration data consists of 8 bits using two's complement representation and can be accessed in select registers depending on the specified axis (address 0x03 for the x-axis and address 0x05 for the y-axis). Furthermore, the accelerometer itself is a device that relies on I^2C to transmit its data, possessing device address 0x18.

***Goals*:**

Part 1

- To explore how SPI works using 'spi_demo', which allows two CC3200 Launchpads to communicate with each other
    - One Launchpad acts as the controller and can send a message to another Launchpad that acts as the peripheral
- To interface the OLED using SPI by connecting it to the CC3200 Launchboard with the appropriate pin configurations
- To explore the implementation of 'spi_demo' to port the library code provided by Adafruit to the CC3200 Launchpad
    - Particularly, the low-level functions WriteData() and WriteCommand() must be modified so that higher-level OLED library functions can be used
- To develop a test program that displays a sequence of texts and patterns on the OLED display
- To learn how to use the Saleae Logic Analyzer to analyze and verify the waveforms produced by SPI as the OLED test program runs

Part 2

- To explore how I^2C works using 'i2c_demo' and the Bosch BMA222 accelerometer built into the CC3200 Launchpad
- To understand how to extract acceleration data from the BMA222 using I^2C and 'i2c_demo'
- To analyze and verify the waveforms produced by I^2C using the Saleae Logic Analyzer while running 'i2c_demo'
- To develop an application program that involves the use of SPI to connect the OLED with the CC3200 Launchpad and I^2C to obtain tilt readings from the BMA222
    - For this program, a ball is first displayed at the center of the OLED and can travel in the x or y directions (or both) based on the tilt readings of the BMA222. The greater the tilt in any direction, the faster the ball will travel in said direction, stopping when it hits the edges of the OLED display

***Methods*:**

<u>Making the two boards communicate with each other</u>:

To make the boards communicate with each other, we found the best option to be flashing the programs into the board's memory, and then running the boards on a PuTTY terminal. Also, for one of the programs, the master mode had to be converted to 1 to depict the controller, and the other had to be converted to 0 to be the peripheral.

<u>SPI and OLED</u>:

To implement SPI into the Adafruit OLED, the following things were implemented in these files:

- Adafruit_OLED.c:
  - To write a command, we first set both OC and DC as low. Once that was done, we put the unsigned long typecast input into the GSPI_BASE using MAP_SPIDataPut. Then, we got the address of the get request, which is not really important in the context of this function, at GSPI_BASE using MAP_ SPIDataGet. After this, we set OC back to high.
  - To write data, we first set OC to low and DC as high. Once that was done, we put the unsigned long typecast input into the GSPI_BASE using MAP_SPIDataPut. Then, we got the address of the get request, which is not really important in the context of this function, at GSPI_BASE using MAP_ SPIDataGet. After this, we set OC back to high.
  - In Adafruit_Init(), we adjusted the values of the RESET pin to match the pin we chose for this purpose in our pin_mux_config.c.
- We converted test.c into test.h so it could be easily included in main.c and the tests could be called into the main file.
- main.c:
  - We removed all the functions related to the peripheral as the peripheral is the OLED in this case. Therefore, we also set MASTER MODE to 1 to signify that our CC3200 was the controller.
  - For the tests:
    - For printing all the fonts, we ran a for loop from 0 till the capacity of the font array. Within the loop, we printed out the character at every index using Outstr. By trial and error, we discovered that the maximum characters is about 32 per row, and a spacing of 8 between the rows prevents overlapping of characters. Therefore, whenever i reaches a

multiple of 32, we change the row by setCursor to the current row index + 8.

§ For 'Hello World!', we reset the screen to black, and set the cursor back to its original position. Then we set the text color and highlight using setTextColor and its size using setTextSize. We printed out the phrase using Outstr.

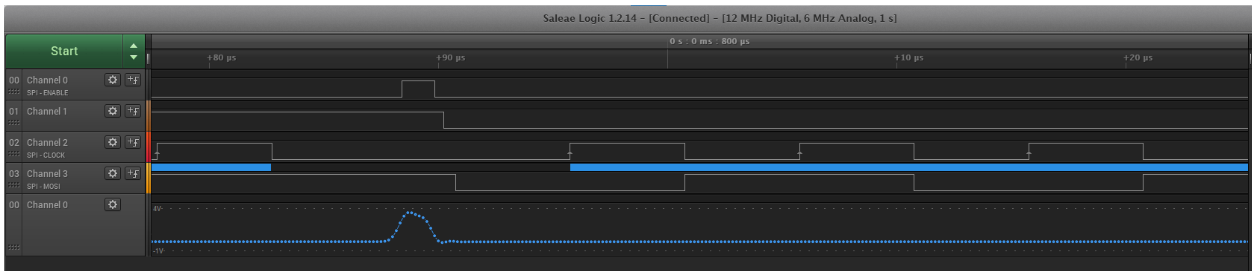
§ For the remaining tests, we just called the test functions from test.h.

§ To add a delay between each test, we used delay(100).

Capturing the Waveforms of the SPI Application:

For capturing the waveforms, we connected the pins of the Logic Analyzer to the assigned pins for the SPI. Then, we ran the program on an instance of the Debugger in the Code Composer Studio, and then captured the waveforms.

Figure 1: SPI Waveform



Capturing the Waveforms of the I^{2}C Demo:

For capturing the waveforms, we connected the pins of the Logic Analyzer to the assigned pins for the I^{2}C, and connected the ground. Then, we ran the program on an instance of the Debugger in the Code Composer Studio, started the waveform capture, and sent the readreg command into the PuTTY terminal and then captured the waveforms.

Figure 2: I^{2}C Waveform (x-axis reading)

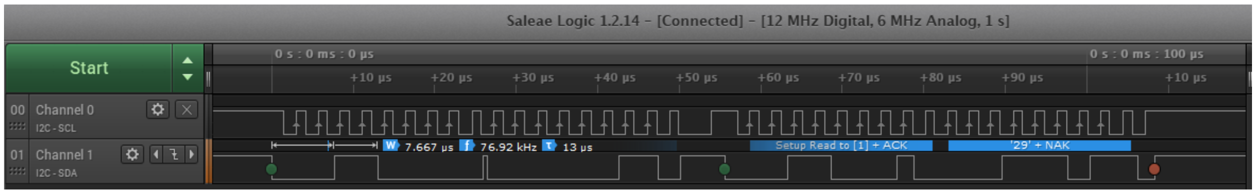
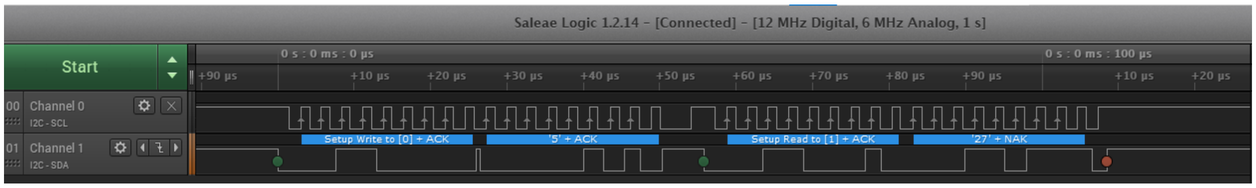


Figure 3: I^{2}C Waveform (y-axis reading)



<u>I^{2}C and SPI</u>:

To interact with the OLED, we imported spi.h and the complete files from the earlier part of the Lab. For interacting with the board using I^{2}C, we used the pre-given values of ucDevAddr as 0x18, the x-axis tilt offset of 0x03, the y-axis tilt offset of 0x05, and ucRdLen of 1. We also declared a buffer, aucRdDataBuf to read from.

For writing the values from the accelerometer to a register, we used the I2C_IF_Write function. There were two function calls for the xOffset and the yOffset. After this, we read from the buffer to get the values from the registers using the I2C_IF_Read function.

To get the ball in the center of the screen, we hardcoded the values of 64 for the x-coordinate and the y-coordinate.

To get the speed from the buffer that contains the values of the register, we converted the specific index at the buffer into a signed character, which gives a range of values from negative to positive, meaning that we can move the ball across all the quadrants of the OLED. To get a value of speed we can work with, we convert the signed character into an integer. We computed the new values for the coordinates by taking the converted integer speed, dividing it by 10, and adding that value to the current coordinate values. To make sure the ball does not go out of bounds, if the x or y coordinate values are greater than 123, then they reset to 123 and if they are less than 4, then they reset to 4. To give the impression that the ball is “moving”, the circle at the original coordinates is filled with black color, and at the new position is filled with blue color.

***Discussion*:**

While making both the boards communicate using SPI, we learned that running two different instances of the Debugger on two different instances of the Code Composer Studio, regardless of whether it was on the same system or two different systems, resulted in an error in communicating with the target for the peripheral. We found a workaround and used Uniflash to flash both the programs into the boards and then the boards could communicate successfully.

For the write function in Adafruit_OLED.c, we discovered that even though we do not use the get value, it was important to use the MAP_ SPIDataGet function correctly. Originally, we used the variable get as is. This resulted in only a quarter of the screen turning black, as our first command was fillScreen(BLACK). On changing the variable to the address of the variable (&get), the commands started working correctly.

While measuring the waveforms of the I^2C demo, we were getting a straight horizontal line. This was because we were not sending the command to read the acceleration registers via PuTTY. Therefore, we had to start the waveform capture, enter the command on PuTTY, and then we could observe the waveforms.

For the application of I^2C with SPI, we discovered that it was important to check the acceleration values that the CC3200 was reading. On finishing the application and moving onto the testing stage, the ball moved by itself to one corner and stayed there. Being unable to fix this bug, we decided to use the readreg commands on the terminal, and then found out that the accelerometer on the board could not read the values correctly and instead, had a fixed x component and y component of acceleration. On switching the board, this error was solved, and the ball moved regularly.

Regarding the captured waveforms for SPI and I^2C, they correspond to the execution of their respective programs.

- For the SPI waveforms captured using the OLED application, we can observe that the MOSI waveform only fluctuates when the clock runs, which matches the fact that SPI is synchronous and requires clock cycles to keep the controller and peripheral in sync as bits are extracted from the data line upon detecting an edge. Furthermore, the waveforms for DC (channel 1) appropriately go from high to low depending on whether data is written (high) or a command is written (low) and the waveforms for ENABLE essentially match the behavior of a chip select, going low just before data is sent to the peripheral and made high after the data is sent. These waveforms match the behavior of functions WriteData() and WriteCommand() when they are called.
- Similarly for the I^2C waveforms, we can observe that the SDA waveform only fluctuates when the clock runs, which matches the fact that I^2C is also synchronous and requires clock cycles to keep the controller and peripheral in sync as bits are extracted from the data line upon detecting an edge. These waveforms match the behavior of 'i2c_demo' when acceleration data is read from their respective registers.

***Conclusion*:**

In this lab, we reinforced the concepts behind SPI and I^2C methods of communication, and applied it between two boards, a board and an OLED, and the accelerometer of the board and an

OLED. We also visualized the components of SPI communication, MOSI, MISO, CLK and CS, and figured out how a drop in one signal, for example, corresponds to data being transmitted.

We learned how reading the address while reading data in both SPI and I^{2}C is important. We also learned more about Code Composer Studio, and how running more than one instance of the debugger does not help in running the program on the board accurately.

We learned about the Adafruit API and how to interact with it to display on the OLED. We also learned how to read values from the inbuilt accelerometer and use it in an application.