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

LAB 2 Verification

Team Member 1: Brian Dang

Team Member 2: Jugal Jain

Section Number/TA: A01 FIGIPE

Demonstrate your working application to your TA:

Demonstrate the working OLED test program

Date	TA Signature	Notes
01/24/19	RE Barrage	

Demonstrate the Saleae logic analyzer waveform capture of SPI

Date	TA Signature	Notes
01/24/19	RE Barrage	

Demonstrate the Saleae logic analyzer waveform capture of I2C

Date	TA Signature	Notes
01/24/19	RE Barrage	

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
1/25/2019	Shawn	COMPLETED 1 DAY late.

Lab 2: Serial Interfacing Using SPI and I^2C

Our project directory is lab2redux on github and accel2 is for the accelerameter and oled_test is our oled testing project.

Intro

Serial Interfaces generally provide communication between digital systems by transmitting data bits in the form of high and low voltage electric signals. This lab provides experience in dealing with various serial interfaces, and the peripherals they control that are common to embedded systems, such as the Adafruit OLED display, as well as sensors like the CC3200's on board accelerometer. The two interfacing protocols that are used to send and receive data to and from these devices are SPI and I^2C respectively.

Background

SPI stands for Serial Peripheral Interface. It is a specification used for short distance communication, commonly in embedded systems, to interface with displays and SD cards. It consists of various synchronously controlled signals that depending on the signal can take the form of 4-wire, 3-wire or even 2 and 1-wire buses. The bus used for this lab consisted of four main signals – Serial Clock, Master Output Slave Input, Data/Command line and Chip Select. The interfacing with the OLED display provided for the lab was done by altering the levels of these four signals in two main functions, WriteData and WriteCommand. The WriteData function controls the actual RGB output of a display at a specific pixel, whereas the WriteCommand function controls the position on the OLED screen that the WriteData function writes to. In both cases, Chip Select is set low and the word to be written to the OLED's shift register is transferred through MOSI. The only difference between the two functions is the status of the Data/Command signal, which is set high to signal Data, and low to signal a Command.

I^2C is a two-way protocol that consists of a data bus made of two wires. Multiple devices can use the I^2C bus, and are differentiated by the use of device addresses. For example, the accelerometer has the address 0x18, meaning that the data from the accelerometer is read from that register. The X and Y coordinates of the tilt of the accelerometer are stored in registers acc_x at 0x3, and acc_y at 0x5. This information is in 2's complement. The two signals used by the I^2C bus are Serial Clock (SCL) and Serial Data (SDA).

Goals

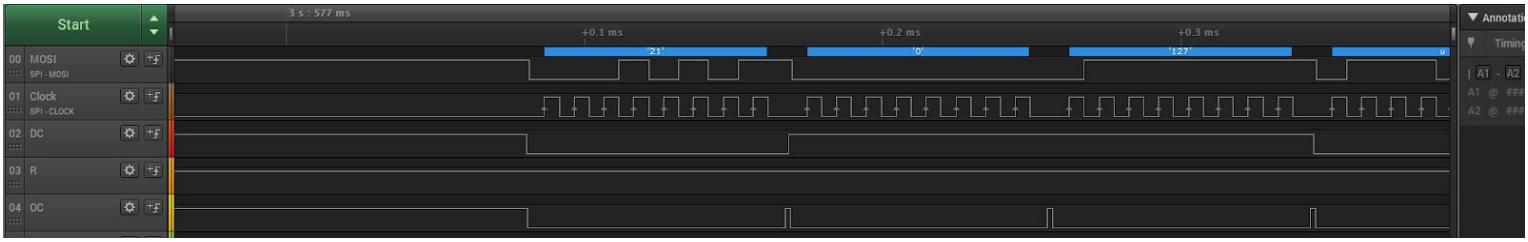
The goal of this lab was to use SPI and I^2C to interface, respectively, with the Adafruit OLED display and the CC3200's onboard accelerometer to create an application program that senses the XY tilt of the Launchpad to control the motion of a circle drawn on the OLED screen. The Salae logic analyzer was also used to analyze the waveforms of the various signals that make up these two interfaces.

Methods

The Pin Mux Tool was used to assign the pins being used to interface with the display. SCLK, MOSI and CS signals were set to the defaults given in the SPI demo, and the DC, OLEDCS and Reset signals were assigned to GPIO output pins. Since there is no output from the board itself, MISO was ignored and not connected to any pins. Once this was set up it was just a matter of going into the WriteCommand and WriteData functions themselves to set pins high and low to control the signals based on the need.

Similarly for I2C, we referenced the readreg method in the I2C demo. In this method in order to get the values from I2C, it would first use I2C_IF_Write to write the memory location of the x or y register, 0x3 or 0x5, to the accelerometer's register location 0x18. Then it would use I2C_IF_Read to get the data stored into that register and store it into a char array. We would then use the values of the accelerometer to add and subtract from a base value of 64 for x and y, since that is the middle of the OLED display. The values for x and y would then be checked to see if they passed the boundaries of the display. Afterwards the x and y values were used for the fillCircle method which would immediately be filled black after in order to reset the board. This would be done in a loop.

Discussion

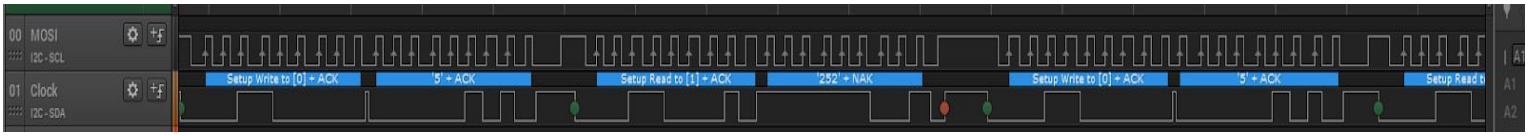


As can be seen in the screen shot, DC is set low, the word written to the shift register using MOSI is '21', OC is set to low to allow for the Write Operation to take place, and it takes place over 8 clock cycles. This means the WriteCommand function was called, with an argument of '21'. Right after we can see that MOSI is writing the value '0', and, all other signals being the same, DC is set high. This denotes a WriteData function call, writing the word '0'.

For I2C I just ran I2C_IF_Read and I2C_IF_Write on loop to get values from the X and Y registers.

X-tilt of the accelerometer: The SCL value is the clock for I2C when it is transmitting data. For SDA, the first part for the Setup Write ACK is when I run right using the X offset of 0x3. This is displayed as the ACK value. Setup read then reads for the X value in the NAK section. This is 1 in the screenshot. This is because it is in a level position close to zero. Write and read have their own waveforms that the I2C analyzer can read. The waveform of the value being read represents the full 8 bit number, high is 1 and 0 is low. These values are read during the rising edge of the clock. In this example the value the value is 1

so only 1 bit is high. The waveforms in between read and write I think are just for switching and also dont feed during a rising edge of the clock.



Y-tilt: This is similar to X-tilt except the value being written is 5 instead, for the 0x5 acc_y register. You can see this from the bit value as well. The value being read is 252, which is a small negative number in 2's complement. This is because the board is level while being read. You can tell based on the waveform values on the clocks rising edge.

A major issue in this lab was getting the OLED to display anything. Some of the issues were: erroneous pin assignments, not correctly labelling GPIO pins as output pins, and using the wrong pin to read the signal from MOSI. This confusion in using the wrong pin was caused by the presence of two different pin 7's on the launchpad. Switching from one to the other immediately fixed the problem. Aside from this the code for the display as well as the accelerometer were fairly straightforward to write and debug, and the rest of the lab was completed with minimal issues.

Conclusion

This was an effective introduction to serial interfacing, as it provided even more basic tools to be used in class, such as the OLED screen, which will be used in future labs. Serial interfacing is a useful tool in reading from and writing to peripheral devices in embedded systems.

Accelerometer Main in accel2 project

```
void main()
{
  float x = 64;
  float y = 64;
  //int r = 4;
  //
  // Initialize Board configurations
  //
  BoardInit();
```

```
//
// Muxing UART and SPI lines.
//
PinMuxConfig();

MAP_PRCMPeripheralClkEnable(PRCM_GSPI,PRCM_RUN_MODE_CLK);

    //
    // Initialising the Terminal.
    //
    InitTerm();

    //
    // Clearing the Terminal.
    //
    ClearTerm();



    //
    // Reset the peripheral
    //
    MAP_PRCMPeripheralReset(PRCM_GSPI);

//
// Initialize the message
//
//memcpy(g_ucTxBuff,MASTER_MSG,sizeof(MASTER_MSG));
```

```
    //
    // Set Tx buffer index
    //
//    ucTxBuffNdx = 0;
//    ucRxBuffNdx = 0;

        I2C_IF_Open(I2C_MASTER_MODE_FST);//enable I2C

        unsigned char XBuf[256];//buffer to store X input
        unsigned char YBuf[256];//buffer to store Y input
        unsigned char temp[256];//temp for 2s complement conversion
        unsigned char ucRegOffsetX = 0x3;//X offset
        unsigned char ucRegOffsetY = 0x5;//Y offset


    //
    // Reset SPI
    //
    MAP_SPIReset(GSPI_BASE);

    //
    // Configure SPI interface
    //
    MAP_SPIConfigSetExpClk(GSPI_BASE,MAP_PRCMPeripheralClockGet(PRCM_GSPI),
                           SPI_IF_BIT_RATE,SPI_MODE_MASTER,SPI_SUB_MODE_0,
                           (SPI_SW_CTRL_CS |
                           SPI_4PIN_MODE |
```

```
            SPI_TURBO_OFF |
            SPI_CS_ACTIVEHIGH |
            SPI_WL_8));

//
// Enable SPI for communication
//
MAP_SPIEnable(GSPI_BASE);



//
// Send the string to slave. Chip Select(CS) needs to be
// asserted at start of transfer and deasserted at the end.
//
// MAP_SPITransfer(GSPI_BASE,g_ucTxBuff,g_ucRxBuff,50,
//       SPI_CS_ENABLE|SPI_CS_DISABLE);

// unsigned long ulDummy;

// Enable Chip select
//

MAP_SPICSEnable(GSPI_BASE);
GPIOPinWrite(GPIOA1_BASE, 0x1, 0x1);
//GPIOPinWrite(GPIOA0_BASE, 0x40, 0x00);
//GPIOPinWrite(GPIOA0_BASE, 0x80, 0x00);

//MAP_SPIDataGet(GSPI_BASE,&ulDummy);
```

```
//oled initialization and setting background to black
Adafruit_Init();
fillScreen(BLACK);

while(1){
    //get x values
    I2C_IF_Write(0x18,&ucRegOffsetX,1,0);
    I2C_IF_Read(0x18, &XBuf[0], 1);

    //get y value from I2C
    I2C_IF_Write(0x18,&ucRegOffsetY,1,0);
    I2C_IF_Read(0x18, &YBuf[0], 1);

    UART_PRINT("X: 0x%x, Y: 0x%x \n ", XBuf[0], YBuf[0]);//testing to see right values in terminal
    fillCircle((int) y, (int) x, 4, BLACK);//reset screen by filling circle back to black

    //if X negative number reverse back to positive and subtract
    if(XBuf[0] & 0x80){
       temp[0] = XBuf[0];
       temp[0] = temp[0] - 0x1;
       temp[0] = ~temp[0];
       x = x - (temp[0] / 8);//divide by 8 to slow down movement
    }
    else{//else add
       x = x + (XBuf[0] / 8);
    }
```

```
//if Y is negative reverse the 2s complement back to positive and subtract
if(YBuf[0] & 0x80){
   temp[0] = YBuf[0];
   temp[0] = temp[0] - 0x1;
   temp[0] = ~temp[0];
   y = y - (temp[0] / 8);
}
else{//else add
   y = y + (YBuf[0] / 8);
}

//checking for bounds
if(x > 123){
   x = 123;
}

if(y > 123){
   y = 123;
}

if(x < 4){
   x = 4;
}

if(y < 4){
   y = 4;
}
```

```
            //draw circle in x and y value, switched because axis was flipped from what I originally planned
            fillCircle((int) y, (int) x, 4, BLUE);

        }

    //
    // Disable chip select
    //
    //MAP_SPICSEnable(GSPI_BASE);
    //GPIOPinWrite(GPIOA1_BASE, 0x1, 0x0);
}
```

OLED Test Main:

```
void main()
{
    //
    // Initialize Board configurations
    //
    BoardInit();

    //
    // Muxing UART and SPI lines.
    //
    PinMuxConfig();

    MAP_PRCMPeripheralClkEnable(PRCM_GSPI,PRCM_RUN_MODE_CLK);

        //
```

```
    // Initialising the Terminal.
    //
    InitTerm();

    //
    // Clearing the Terminal.
    //
    ClearTerm();

    //
    // Display the Banner
    //
    Message("\n\n\n\r");
    Message("\t\t   ****************************************\n\r");
    Message("\t\t       CC3200 SPI Demo Application  \n\r");
    Message("\t\t   ****************************************\n\r");
    Message("\n\n\n\r");

    //
    // Reset the peripheral
    //
    MAP_PRCMPeripheralReset(PRCM_GSPI);

//
// Initialize the message
//
//memcpy(g_ucTxBuff,MASTER_MSG,sizeof(MASTER_MSG));
```

```
    //
    // Set Tx buffer index
    //
//    ucTxBuffNdx = 0;
//    ucRxBuffNdx = 0;

    //
    // Reset SPI
    //
    MAP_SPIReset(GSPI_BASE);

    //
    // Configure SPI interface
    //
    MAP_SPIConfigSetExpClk(GSPI_BASE,MAP_PRCMPeripheralClockGet(PRCM_GSPI),
                SPI_IF_BIT_RATE,SPI_MODE_MASTER,SPI_SUB_MODE_0,
                (SPI_SW_CTRL_CS |
                SPI_4PIN_MODE |
                SPI_TURBO_OFF |
                SPI_CS_ACTIVEHIGH |
                SPI_WL_8));

    //
    // Enable SPI for communication
    //
    MAP_SPIEnable(GSPI_BASE);
```

```
//
// Send the string to slave. Chip Select(CS) needs to be
// asserted at start of transfer and deasserted at the end.
//
// MAP_SPITransfer(GSPI_BASE,g_ucTxBuff,g_ucRxBuff,50,
//       SPI_CS_ENABLE|SPI_CS_DISABLE);

// unsigned long ulDummy;

char* text = "Hello World";
// Enable Chip select
//

//enable SPICS and OC to high
MAP_SPICSEnable(GSPI_BASE);
GPIOPinWrite(GPIOA1_BASE, 0x1, 0x1);
//GPIOPinWrite(GPIOA0_BASE, 0x40, 0x00);
//GPIOPinWrite(GPIOA0_BASE, 0x80, 0x00);

//MAP_SPIDataGet(GSPI_BASE,&ulDummy);

Adafruit_Init();
int x; //x axis counter for displaying ASCII
int y;//y axis counter for ASCII
char c;//counter for the character
while(1){
   //drawing ascii values
   fillScreen(BLACK);
```

```
x =0;
y =0;
//code to cycle through and print ASCII characters
for(c = 0; c <= 254; c++){
   drawChar(x, y, c, WHITE, BLACK, 1);
   x += 5;
   if(x >= 125){
      x = 0;
      y += 7;
   }
   if(y >= 125){
      y = 0;
      fillScreen(BLACK);
   }

}
delay(100);
//hello world display
setTextColor(WHITE, BLACK);
setTextSize(1);
Outstr(text);
delay(100);
//horizontal bands
fillScreen(BLACK);
drawLine(0, 0, 127, 0, WHITE);
drawLine(0, 21, 127, 21, BLUE);
drawLine(0, 42, 127, 42, GREEN);
drawLine(0, 63, 127, 63, CYAN);
```

```
drawLine(0, 84, 127, 84, RED);
drawLine(0, 105, 127, 105, MAGENTA);
drawLine(0, 127, 127, 127, YELLOW);
delay(100);
//vertical bands
fillScreen(BLACK);
drawLine(0, 0, 0, 127, WHITE);
drawLine(21, 0, 21, 127, BLUE);
drawLine(42, 0, 42, 127, GREEN);
drawLine(63, 0, 63, 127, CYAN);
drawLine(84, 0, 84, 127, RED);
drawLine(105, 0, 105, 127, MAGENTA);
drawLine(127, 0, 127, 127, YELLOW);
fillScreen(BLACK);
lcdTestPattern();
fillScreen(BLACK);
lcdTestPattern2();
testlines(YELLOW);
delay(100);
testfastlines(BLUE, GREEN);
delay(100);
testdrawrects(GREEN);
delay(100);
testfillrects(YELLOW, MAGENTA);
delay(100);
testfillcircles(10, BLUE);
testdrawcircles(10, WHITE);
fillCircle(64, 64, 4, WHITE);
```

```
    delay(100);
    testroundrects();
    delay(100);
    testtriangles();
    delay(100);
  }

  //
  // Disable chip select
  //
  //MAP_SPICSEnable(GSPI_BASE);
  //GPIOPinWrite(GPIOA1_BASE, 0x1, 0x0);
}
```

AdaFruit_OLED writeData and writeCommand:

```
void writeCommand(unsigned char c) {

//TODO 1
/* Write a function to send a command byte c to the OLED via
*  SPI.
*/
    //enable CS and set OC to low to start writing command, set DC to low for command
    unsigned long ulDummy;
    MAP_SPICSEnable(GSPI_BASE);
    GPIOPinWrite(GPIOA0_BASE, 0x40, 0x00);

    GPIOPinWrite(GPIOA1_BASE, 0x1, 0x0);
```

```
    MAP_SPIDataPut(GSPI_BASE,c);//write command
    MAP_SPIDataGet(GSPI_BASE,&ulDummy);//get buffer

    //set OC AND CS back to high to stop
    GPIOPinWrite(GPIOA1_BASE, 0x1, 0x1);
    MAP_SPICSDisable(GSPI_BASE);

}
//*****************************************************************************

void writeData(unsigned char c) {

//TODO 2
/* Write a function to send a data byte c to the OLED via
*  SPI.
*/
    //enable CS and set OC to low to start writing command, set DC to high for data
     unsigned long ulDummy;
     MAP_SPICSEnable(GSPI_BASE);
     GPIOPinWrite(GPIOA0_BASE, 0x40, 0x40);

     GPIOPinWrite(GPIOA1_BASE, 0x1, 0x0);


     MAP_SPIDataPut(GSPI_BASE,c);
     MAP_SPIDataGet(GSPI_BASE,&ulDummy);
```

```
        //set OC back to high to stop
        GPIOPinWrite(GPIOA1_BASE, 0x1, 0x1);
        MAP_SPICSDisable(GSPI_BASE);

}
```