

Lab 9 Report

EEL4742C - 00446

Yousef Awad

November 2025

Contents

Introduction	2
9.1 Using the ADC SAR-Type	2
9.2 Reading the X- and Y- Coordinates of the Joystick	4
9.3 Application: Platform Balancing Control	8
Student Q&A	14
1	14
2	14
3	14
4	14
5	14
Conclusion	14

Introduction

9.1 Using the ADC SAR-Type

Now, for this setup I used SPI Mode 1.

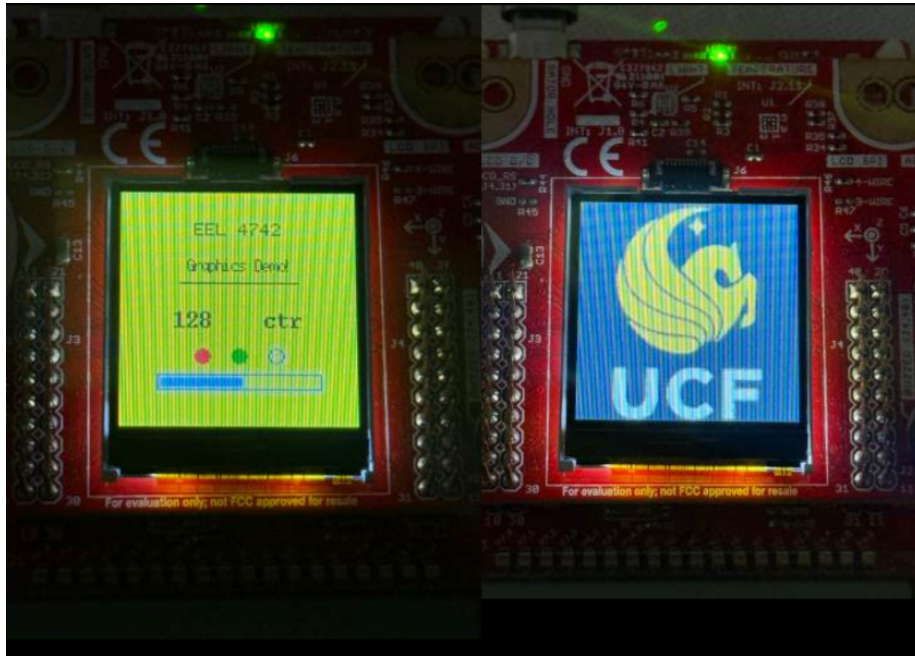
```
1 void HAL_LCD_PortInit(void)
2 {
3     // Configuring the SPI pins
4     // Configure UCBOCLK/P1.4 pin to serial clock
5     P1SEL0 |= BIT4; // on according to data sheet for UCBOCLK
6     P1SEL1 &= ~BIT4; // off according to data sheet for UCBOCLK
7     // Configure UCBOSIMO/P1.6 pin to SIMO
8     P1SEL0 |= BIT6;
9     P1SEL1 &= ~BIT6;
10    // OK to ignore UCBOSTE/P1.5 since we are connecting the display's
        // enable bit
11    // to low (enabled all the time).
12    // OK to ignore UCBOSOMI/P1.7 since the display doesn't give back
        // any data
13
14    // Configuring the display's other pins
15    // Set reset pin as output
16    P9DIR |= BIT4;
17    // Set the data/command pin as output (P2.3)
18    P2DIR |= BIT3;
19    // Set the chip select pin as output (P2.5)
20    P2DIR |= BIT5;
21
22    return;
23 }
24 void HAL_LCD_SpiInit(void)
25 {
26     // SPI configuration
27
28     // Put eUSCI in reset state and set all fields in the register to
        // 0
29    UCBOCTLW0 = UCSWRST;
30
31    // Fields that need to be nonzero are changed below
32    // Set clock phase to "capture on 1st edge, change on following
        // edge"
33    // 1 = latched on the first edge and shifted on the following
        // edge
34    UCBOCTLW0 |= UCCKPH;
35    // Set clock polarity to "inactive low" (0)
36    UCBOCTLW0 &= ~UCCKPL;
37    // Set data order to "transmit MSB first" (1)
38    UCBOCTLW0 |= UCMSB;
39    // Set data size to 8-bit (0)
40    UCBOCTLW0 &= ~UC7BIT;
41    // Set MCU to "SPI master" (1)
42    UCBOCTLW0 |= UCMST;
43    // Set SPI to "3-pin SPI" (0) (we won't use eUSCI's chip select)
44    UCBOCTLW0 |= UCMODE0;
45    // Set module to synchronous mode (1)
46    UCBOCTLW0 |= UCSYNC;
```

```

47 // Set clock to SMCLK which is 2, in master mode
48 UCBOCTLW0 |= UCSSEL_2;
49 // Configure the clock divider (SMCLK is set to 16 MHz; run SPI
    at 8 MHz using
50 // SMCLK)
51 // Maximum SPI supported frequency on the display is 10 MHz
52 UCBOBRW |= 2; // Divider = 2 in order to get 8 Mhz from 16 Mhz
    clock
53 // Exit the reset state (1) at the end of the configuration
54 UCBOCTLW0 &= ~UCSWRST;
55 // Set CS' (chip select) bit to 0 (display always enabled)
56 P2OUT &= ~BIT5;
57 // Set DC' bit to 0 (assume data)
58 P2OUT &= ~BIT3;
59
60 return;
61 }

```

9.2 Reading the X- and Y- Coordinates of the Joystick



```
1 // Part 9.2 Code -----
2 #include "Glib/glib/glib.h" // Graphics library (glib)
3 #include "LcdDriver/lcd_driver.h" // LCD driver
4 #include "msp430fr6989.h"
5
6 #include <stdio.h>
7 #define redLED BIT0
8 #define greenLED BIT7
9 #define S1 BIT1
10 #define S2 BIT2
11
12 extern const tImage UCF_Logo;
13
14 void Initialize_Clock_System()
15 {
16     // DCO frequency = 16 MHz
17     // MCLK = fDCO/1 = 16 MHz
18     // SMCLK = fDCO/1 = 16 MHz
19     // Activate memory wait state
20     FRCTL0 = FRCTLPW | NWAITS_1; // Wait state=1
21     CSCTL0 = CSKEY;
22     // Set DCOFSEL to 4 (3-bit field)
23     CSCTL1 &= ~DCOFSEL_7;
24     CSCTL1 |= DCOFSEL_4;
25     // Set DCORSEL to 1 (1-bit field)
26     CSCTL1 |= DCORSEL;
27     // Change the dividers to 0 (div by 1)
```

```

28 CSCTL3 &= ~(DIVS2 | DIVS1 | DIVS0); // DIVS=0 (3-bit)
29 CSCTL3 &= ~(DIVM2 | DIVM1 | DIVM0); // DIVM=0 (3-bit)
30 CSCTL0_H = 0;
31 return;
32 }
33
34 void main(void)
35 {
36     char mystring[20];
37     // Configure WDT & GPIO
38     WDTCTL = WDTPW | WDTHOLD;
39     PM5CTL0 &= ~LOCKLPM5;
40     // Configure LEDs
41     P1DIR |= redLED;
42     P9DIR |= greenLED;
43     P1OUT &= ~redLED;
44     P9OUT &= ~greenLED;
45     // Configure buttons
46     P1DIR &= ~(S1 | S2);
47     P1REN |= (S1 | S2);
48     P1OUT |= (S1 | S2);
49     P1IFG &= ~(S1 | S2); // Flags are used for latched polling
50     // Set the LCD backlight to highest level
51     // P2DIR |= BIT6;
52     // P2OUT |= BIT6;
53
54     // Configure clock system
55     Initialize_Clock_System();
56
57     // Graphics functions
58     Graphics_Context g_sContext; // Declare a graphic library
59     context Crystalfontz128x128_Init(); // Initialize the display
60     // Set the screen orientation
61     Crystalfontz128x128_SetOrientation(0);
62     // Initialize the context
63     Graphics_initContext(&g_sContext, &g_sCrystalfontz128x128);
64     // 1/2) Set new background and foreground colors
65     Graphics_setBackgroundColor(
66         &g_sContext, GRAPHICS_COLOR_BLACK); // Set new background
        color to white
67     Graphics_setForegroundColor(
68         &g_sContext, GRAPHICS_COLOR_PINK); // Set new foreground
        color to blue
69     while (1) {
70         // FOR IMAGE DISPLAY (DISPLAY 2)
71         // 1/2) Set new background and foreground colors
72         Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
73         // Set new background color to white
74         Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_GOLD);
75         // Set new foreground color to blue
76         // Clear the screen
77         Graphics_clearDisplay(&g_sContext);
78         Graphics_drawImage(&g_sContext, &UCF_Logo, 1, 1);
79         while ((P1IN & S1) != 0)
80         {
81             // wait
82         }

```

```

83  __delay_cycles(8000000);
84  // FOR COUNTER AND SHAPES DISPLAY (DISPLAY 1)
85  // 1/2) Set new background and foreground colors
86  Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_WHITE);
87  // Set new background color to white
88  Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLUE);
89  // Set new foreground color to blue
90  // Set the default font for strings
91  GrContextFontSet(&g_sContext, &g_sFontlucidasans8x15);
92  // Clear the screen
93  Graphics_clearDisplay(&g_sContext);
94  // 3) Draw an outline circle
95  Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED);
96  // make circle outline red
97  Graphics_drawCircle(&g_sContext, 25, 20, 10);
98  // 3) Draw a filled circle
99  Graphics_setForegroundColor(
100     &g_sContext,
101     GRAPHICS_COLOR_DARK_ORANGE); // make filled circle orange
102  Graphics_fillCircle(&g_sContext, 50, 20, 10);
103  // 3) Draw an outline rectangle
104  Graphics_setForegroundColor(
105     &g_sContext,
106     GRAPHICS_COLOR_YELLOW); // make rectangle outline yellow
107  const Graphics_Rectangle RectLoc = {
108     .xMin = 15, .yMin = 40, .xMax = 35, .yMax = 65
109  };
110  Graphics_drawRectangle(&g_sContext, &RectLoc);
111  // 3) Draw a filled rectangle
112  Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_GREEN);
113  // make filled rectangle green
114  const Graphics_Rectangle RectLoc1 = {
115     .xMin = 40, .yMin = 40, .xMax = 60, .yMax = 65};
116  Graphics_fillRectangle(&g_sContext, &RectLoc1);
117  // 3) Draw a horizontal line
118  Graphics_setForegroundColor(
119     &g_sContext,
120     GRAPHICS_COLOR_DODGER_BLUE); // make horizontal line blue
121  Graphics_drawLineH(&g_sContext, 15, 60, 80);
122  // 4) Display counter
123  // Set the default font for strings
124  Graphics_setForegroundColor(
125     &g_sContext,
126     GRAPHICS_COLOR_HOT_PINK); // make horizontal line blue
127  GrContextFontSet(&g_sContext, &g_sFontlucidabright6x12);
128  Graphics_drawStringCentered(&g_sContext, "ctr",
129     AUTO_STRING_LENGTH, 65, 102,
130     OPAQUE_TEXT); // Display number
131  Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
132  // make horizontal line blue
133  GrContextFontSet(&g_sContext, &g_sFontfixed7x13);
134  Graphics_drawStringCentered(&g_sContext, "Font",
135     AUTO_STRING_LENGTH, 90, 50,
136     OPAQUE_TEXT); // Display number
137  char str[3];
138  int i = 0;
139  while ((P1IN & S1) != 0)

```

```

138 {
139     Graphics_setForegroundColor(&g_sContext,
140                                GRAPHICS_COLOR_PURPLE); // make
counter purple
141     sprintf(str, "%d", i); // Convert number to string
142     __delay_cycles(2000000); // wait a second
143     Graphics_drawStringCentered(&g_sContext, str,
AUTO_STRING_LENGTH, 37, 100,
OPAQUE_TEXT); // Display number
144
145     if (i == 256)
146     {
147         // reset counter by making last number same color as
background before
148         // repeating
149         Graphics_setForegroundColor(&g_sContext,
GRAPHICS_COLOR_WHITE);
150         Graphics_drawStringCentered(&g_sContext, str,
AUTO_STRING_LENGTH, 37,
151                                     100, OPAQUE_TEXT);
152         i = 0;
153     }
154     i += 1;
155 }
156
157 __delay_cycles(8000000);
158 }
159 }

```

9.3 Application: Platform Balancing Control

```
1 // Part 9.3 Code -----
2 #include "Grlib/grlib/grlib.h" // Graphics library (grlib)
3 #include "LcdDriver/lcd_driver.h" // LCD driver
4 #include "msp430fr6989.h"
5 #include <math.h>
6 #include <stdio.h>
7
8 #define redLED BIT0
9 #define greenLED BIT7
10 #define S1 BIT1
11 #define S2 BIT2
12
13 // For I2C
14 #define MANUFACTURERIDREG 0x7E // Manufacturer Register ID
15 #define DEVICEIDREG 0x7F // Device Register ID
16 #define I2CADDR 0x44 // I2C Address
17
18 // For reading Sensor data
19 #define CONFIGREG 0x01
20 #define RESULTREG 0x00
21
22 void Initialize_Clock_System()
23 {
24     // DCO frequency = 16 MHz
25     // MCLK = fDCO/1 = 16 MHz
26     // SMCLK = fDCO/1 = 16 MHz
27     // Activate memory wait state
28     FRCTL0 = FRCTLPW | NWAITS_1; // Wait state=1
29     CSCTL0 = CSKEY;
30     // Set DCOFSEL to 4 (3-bit field)
31     CSCTL1 &= ~DCOFSEL_7;
32     CSCTL1 |= DCOFSEL_4;
33     // Set DCORSEL to 1 (1-bit field)
34     CSCTL1 |= DCORSEL;
35     // Change the dividers to 0 (div by 1)
36     CSCTL3 &= ~(DIVS2 | DIVS1 | DIVS0); // DIVS=0 (3-bit)
37     CSCTL3 &= ~(DIVM2 | DIVM1 | DIVM0); // DIVM=0 (3-bit)
38     CSCTL0_H = 0;
39     return;
40 }
41
42 // I2C Initialization Function (for lux sensor)
43 void Initialize_I2C(void)
44 {
45     // Configure the MCU in Master mode
46     // Configure pins to I2C functionality
47     // (UCB1SDA same as P4.0) (UCB1SCL same as P4.1)
48     // (P4SEL1=11, P4SEL0=00) (P4DIR=xx)
49     P4SEL1 |= (BIT1 | BIT0);
50     P4SEL0 &= ~(BIT1 | BIT0);
51     // Enter reset state and set all fields in this register to zero
52     UCB1CTLW0 = UCSWRST;
53     // Fields that should be nonzero are changed below
54     // (Master Mode: UCMST) (I2C mode: UCMODE_3)
55     // (Synchronous mode: UCSYNC)
```



```

56 // (UCSSEL 1:ACLK, 2,3:SMCLK)
57 UCB1CTLW0 |= UCMST | UCMODE_3 | UCSYNC | UCSSEL_3;
58 // Clock frequency: 16 MHz, 16M/320K =
59 UCB1BRW = 50;
60 // Chip Data Sheet p. 53 (Should be 400 KHz max)
61 // Exit the reset mode at the end of the configuration
62 UCB1CTLW0 &= ~UCSWRST;
63 }
64
65 // Read a word (2 bytes) from I2C (address, register)
66 int i2c_read_word(unsigned char i2c_address,
67                  unsigned char i2c_reg,
68                  unsigned int *data)
69 {
70     // Intialize to ensure successful
71     unsigned char byte1 = 0;
72     unsigned char byte2 = 0;
73     reading UCB1I2CSA = i2c_address; // Set address
74     UCB1IFG &= ~UCTXIFG0;
75     // Transmit a byte (the internal register address)
76     UCB1CTLW0 |= UCTR;
77     UCB1CTLW0 |= UCTXSTT;
78     while ((UCB1IFG & UCTXIFG0) == 0)
79     {
80         // Wait for flag to raise
81     }
82     UCB1TXBUF = i2c_reg; // Write in the TX buffer
83     while ((UCB1IFG & UCTXIFG0) == 0)
84     {
85         // Buffer copied to shift
86     }
87     register;
88     // Tx in progress; set Stop bit
89     // Repeated Start
90     UCB1CTLW0 &= ~UCTR;
91     UCB1CTLW0 |= UCTXSTT;
92     // Read the first byte
93     while ((UCB1IFG & UCRXIFG0) == 0)
94     {
95         // Wait for flag to raise
96     }
97     byte1 = UCB1RXBUF;
98     // Assert the Stop signal bit before receiving the last byte
99     UCB1CTLW0 |= UCTXSTP;
100    // Read the second byte
101    while ((UCB1IFG & UCRXIFG0) == 0)
102    {
103        // Wait for flag to raise
104    }
105    byte2 = UCB1RXBUF;
106    while ((UCB1CTLW0 & UCTXSTP) != 0)
107    {
108        // wait
109    }
110    while ((UCB1STATW & UCBBUSY) != 0)
111    {
112        // wait

```

```

113     }
114     *data = (byte1 << 8) | (byte2 & (unsigned int)0x00FF);
115     return 0;
116 }
117
118 // Write a word (2 bytes) to I2C (address, register)
119 int i2c_write_word(unsigned char i2c_address,
120                   unsigned char i2c_reg,
121                   unsigned int data)
122 {
123     unsigned char byte1, byte2;
124     UCB1I2CSA = i2c_address; // Set I2C address
125     byte1 = (data >> 8) & 0xFF; // MSByte
126     byte2 = data & 0xFF; // LSByte
127     UCB1IFG &= ~UCTXIFG0;
128     // Write 3 bytes
129     UCB1CTLW0 |= (UCTR | UCTXSTT);
130     while ((UCB1IFG & UCTXIFG0) == 0)
131     {
132     }
133     UCB1TXBUF = i2c_reg;
134     while ((UCB1IFG & UCTXIFG0) == 0)
135     {
136     }
137     UCB1TXBUF = byte1;
138     while ((UCB1IFG & UCTXIFG0) == 0)
139     {
140     }
141     UCB1TXBUF = byte2;
142     while ((UCB1IFG & UCTXIFG0) == 0)
143     {
144     }
145     UCB1CTLW0 |= UCTXSTP;
146     while ((UCB1CTLW0 & UCTXSTP) != 0)
147     {
148     }
149     while ((UCB1STATW & UCBBUSY) != 0)
150     {
151     }
152     return 0;
153 }
154
155 // Initializing Sensor
156 void Initialize_SENS()
157 {
158     // Configuration for Sensor based on lab manual
159     unsigned volatile int configuration_value = 0x7604;
160     i2c_write_word(I2CADDR, CONFIGREG, configuration_value);
161 }
162
163 void main(void)
164 {
165     char mystring[20];
166     // Configure WDT & GPIO
167     WDTCTL = WDTPW | WDTHOLD;
168     PM5CTL0 &= ~LOCKLPM5;
169     // Configure LEDs

```

```

170 P1DIR |= redLED;
171 P9DIR |= greenLED;
172 P1OUT &= ~redLED;
173 P9OUT &= ~greenLED;
174 // Configure buttons
175 P1DIR &= ~(S1 | S2);
176 P1REN |= (S1 | S2);
177 P1OUT |= (S1 | S2);
178 P1IFG &= ~(S1 | S2); // Flags are used for latched polling
179 // Set the LCD backlight to highest level
180 // P2DIR |= BIT6;
181 // P2OUT |= BIT6;
182
183 // Configure clock system
184 Initialize_Clock_System();
185
186 // Initializing the I2C
187 Initialize_I2C();
188
189 // Initializing Sensor with new function based on configuration
190 Initialize_SENS();
191
192 // Graphics functions
193 Graphics_Context g_sContext; // Declare a graphic library
194 context Crystalfontz128x128_Init(); // Initialize the display
195 // Set the screen orientation
196 Crystalfontz128x128_SetOrientation(0);
197 // Initialize the context
198 Graphics_initContext(&g_sContext, &g_sCrystalfontz128x128);
199 // Set background and foreground colors
200 Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
201 Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_WHITE);
202 // Set the default font for strings
203 GrContextFontSet(&g_sContext, &g_sFontFixed6x8);
204 // Clear the screen
205 Graphics_clearDisplay(&g_sContext);
206
207 // Set up Optical Counter Display
208 sprintf(mystring, "Optical Counter");
209 // show "Optical Sensor"
210 Graphics_drawStringCentered(&g_sContext,
211                             mystring,
212                             AUTO_STRING_LENGTH,
213                             64,
214                             40,
215                             OPAQUE_TEXT);
216 Graphics_setForegroundColor(&g_sContext,
217                             GRAPHICS_COLOR_GOLDENROD);
218 sprintf(mystring, "lux");
219 // show "lux"
220 Graphics_drawStringCentered(&g_sContext,
221                             mystring,
222                             AUTO_STRING_LENGTH,
223                             90,
224                             80,
225                             OPAQUE_TEXT);
226 // Make rectangular outline for optical sensor value

```

```

227 Graphics_setForegroundColor(&g_sContext,
228                             GRAPHICS_COLOR_GOLDENROD);
229 const Graphics_Rectangle RectLoc = {
230     .xMin = 10, .yMin = 100,
231     .xMax = (128 - 10), .yMax = (100 + 10)
232 };
233 Graphics_drawRectangle(&g_sContext, &RectLoc);
234 int LUX_Val = 0;
235 int Object_Counter = 0;
236 char str[5];
237 bool update_object_num = 0;
238 while (1)
239 {
240     // Storing sensor value based on value in result
241     // reg and lux reading from sensor
242     i2c_read_word(I2CADDR, RESULTREG, &LUX_Val);
243     // make counter purple
244     Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED);
245     LUX_Val = LUX_Val * 1.28;
246     if (LUX_Val >= 1000)
247     {
248         LUX_Val = 1000; // Caps LUX_Val at 1000 (based on manual)
249     }
250     if ((LUX_Val <= 200) && (update_object_num == 0))
251     {
252         // For object, count the object
253         detection Object_Counter++;
254         update_object_num = 1;
255     }
256     else if ((LUX_Val > 200) && (update_object_num == 1))
257     {
258         // Won't update again until object passes
259         update_object_num = 0;
260     }
261     sprintf(str, "%d", Object_Counter); // Convert number to string
262     Graphics_drawStringCentered(&g_sContext,
263                                str,
264                                AUTO_STRING_LENGTH,
265                                64,
266                                60,
267                                OPAQUE_TEXT); // Display number
268     // Convert number to string
269     sprintf(str, "%4d", LUX_Val);
270     Graphics_drawStringCentered(&g_sContext,
271                                str,
272                                AUTO_STRING_LENGTH,
273                                62,
274                                80,
275                                OPAQUE_TEXT); // Display number
276     __delay_cycles(500000); // wait a second
277     // Make rectangular outline for showing optical sensor value
278     if (round(LUX_Val / 9.434) + 1 < 105)
279     {
280         Graphics_setForegroundColor(&g_sContext,
281                                     GRAPHICS_COLOR_BLACK);
282         const Graphics_Rectangle RectLoc = {
283             .xMin = 11 + round(LUX_Val / 9.434) + 1,

```

```

284     .yMin = 101,
285     .xMax = (128 - 11),
286     .yMax = (100 + 9)
287 };
288 Graphics_fillRectangle(&g_sContext, &RectLoc);
289 }
290 // Make rectangular outline for showing optical sensor value
291 Graphics_setForegroundColor(&g_sContext,
292                             GRAPHICS_COLOR_GOLDENROD);
293 const Graphics_Rectangle RectLoc1 = {
294     .xMin = 11,
295     .yMin = 101,
296     .xMax = 11 + round(LUX_Val / 9.434),
297     .yMax = (100 + 9)
298 };
299 Graphics_fillRectangle(&g_sContext, &RectLoc1);
300 }
301 }

```

Student Q&A

1

Given: *Is SPI implemented as simplex or full-duplex in this experiment?* SPI is only implemented as simplex as the Serial Data Out Pin does not transmit data back to the MCU. This means that there is only one lane of communication, thereby not being duplexed.

2

Given: *What SPI clock frequency did we set up in this lab?* The SPI Clock Frequency was set to 8MHz as instructed in the lab manual...

3

Given: *What I2C clock frequency did we set up in this lab?* The I2C Clock Frequency was set to 320KHz as was instructed by the lab manual....

4

Given: *What is the maximum SPI clock frequency that is supported by the eUSCI module? Look in the microcontroller's data sheet in Table 5-18.* The Maximum SPI Clock Frequency supported by the eUSCI Module for the MSP430 is 16MHz, according to the data sheet (Table 5-18, shown below).

Table 5-18. eUSCI (SPI Master Mode) Clock Frequency

PARAMETER	TEST CONDITIONS	MIN	MAX	UNIT
f _{eUSCI} eUSCI input clock frequency	Internal: SMCLK or ACLK, Duty cycle = 50% ±10%		16	MHz

5

Given: *Show how you computed the I2C clock divider in the 3rd part.* To calculate the I2C Clock Divider for Part 3, I simply did the following:

$$Clock = \frac{16MHz_{master\ clock}}{320KHz_{desired\ clock}} = \frac{16 * 10^6}{320 * 10^3} = 50$$

Conclusion