Lab 10 Report

EEL4742C - 00446

Yousef Awad

November 2025

Contents

Introduction

This experiment investigates the advanced capabilities of the MSP430FR6989 Timer_A module, focusing on using multiple channels to handle concurrent hardware events. Objectives include implementing simultaneous timing intervals to control independent LED flashing rates in continuous mode, generating Pulse Width Modulation (PWM) signals to adjust LED brightness without CPU intervention , and utilizing Input Capture mode to timestamp external user input.

10.1 Timer's Multiple Channels

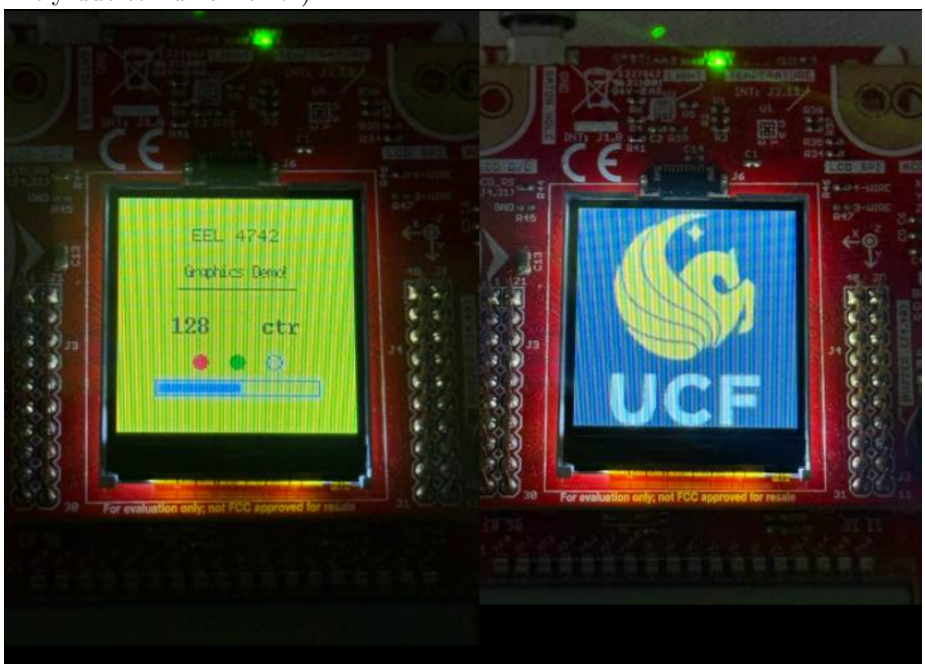
The visual comparison appears to be the same/similar to what I set the values to be at. Alongside this here is the following derivation for the number of cycles for each channel:

```
void HAL_LCD_PortInit(void)
{
  // Configuring the SPI pins
  // Configure UCB0CLK/P1.4 pin to serial clock
  P1SEL0 |= BIT4;  // on according to data sheet for UCB0CLK
  P1SEL1 &= ~BIT4; // off according to data sheet for UCB0CLK
  // Configure UCB0SIMO/P1.6 pin to SIMO
  P1SEL0 |= BIT6;
  P1SEL1 &= ~BIT6;
  // OK to ignore UCB0STE/P1.5 since we are connecting the display's enable bit
  // to low (enabled all the time).
  // OK to ignore UCB0SOMI/P1.7 since the display doesn't give back any data

  // Configuring the display's other pins
  // Set reset pin as output
  P9DIR |= BIT4;
  // Set the data/command pin as output (P2.3)
  P2DIR |= BIT3;
  // Set the chip select pin as output (P2.5)
  P2DIR |= BIT5;

  return;
}
void HAL_LCD_SpiInit(void)
{
  // SPI configuration

  // Put eUSCI in reset state and set all fields in the register to 0
  UCB0CTLW0 = UCSWRST;

  // Fields that need to be nonzero are changed below
  // Set clock phase to "capture on 1st edge, change on following edge"
  // 1 = latched on the first edge and shifted on the following edge
  UCB0CTLW0 |= UCCKPH;
```

```
35    // Set clock polarity to "inactive low" (0)
36    UCB0CTLW0 &= ~UCCKPL;
37    // Set data order to "transmit MSB first" (1)
38    UCB0CTLW0 |= UCMSB;
39    // Set data size to 8-bit (0)
40    UCB0CTLW0 &= ~UC7BIT;
41    // Set MCU to "SPI master" (1)
42    UCB0CTLW0 |= UCMST;
43    // Set SPI to "3-pin SPI" (0) (we won't use eUSCI's chip select)
44    UCB0CTLW0 |= UCMODE0;
45    // Set module to synchronous mode (1)
46    UCB0CTLW0 |= UCSYNC;
47    // Set clock to SMCLK which is 2, in master mode
48    UCB0CTLW0 |= 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    UCB0BRW |= 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    UCB0CTLW0 &= ~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 }
```

10.2 Using Three Channels

The durations, at least when tested by my stopwatch on my phone and on my watch, are close to the time's that I set (though, there was a slight error most likely due to human error).




```c
// Part 9.2 Code ------------------------------
#include "Grlib/grlib/grlib.h"    // Graphics library (grlib)
#include "LcdDriver/lcd_driver.h" // LCD driver
#include "msp430fr6989.h"

#include <stdio.h>
#define redLED BIT0
#define greenLED BIT7
#define S1 BIT1
#define S2 BIT2

extern const tImage UCF_Logo;

void Initialize_Clock_System()
{
  // DCO frequency = 16 MHz
  // MCLK = fDCO/1 = 16 MHz
  // SMCLK = fDCO/1 = 16 MHz
  // Activate memory wait state
  FRCTL0 = FRCTLPW | NWAITS_1; // Wait state=1
  CSCTL0 = CSKEY;
  // Set DCOFSEL to 4 (3-bit field)
  CSCTL1 &= ~DCOFSEL_7;
  CSCTL1 |= DCOFSEL_4;
  // Set DCORSEL to 1 (1-bit field)
```

```
  CSCTL1 |= DCORSEL;
  // Change the dividers to 0 (div by 1)
  CSCTL3 &= ~(DIVS2 | DIVS1 | DIVS0); // DIVS=0 (3-bit)
  CSCTL3 &= ~(DIVM2 | DIVM1 | DIVM0); // DIVM=0 (3-bit)
  CSCTL0_H = 0;
  return;
}

void main(void)
{
  char mystring[20];
  // Configure WDT & GPIO
  WDTCTL = WDTPW | WDTHOLD;
  PM5CTL0 &= ~LOCKLPM5;
  // Configure LEDs
  P1DIR |= redLED;
  P9DIR |= greenLED;
  P1OUT &= ~redLED;
  P9OUT &= ~greenLED;
  // Configure buttons
  P1DIR &= ~(S1 | S2);
  P1REN |= (S1 | S2);
  P1OUT |= (S1 | S2);
  P1IFG &= ~(S1 | S2); // Flags are used for latched polling
  // Set the LCD backlight to highest level
  // P2DIR |= BIT6;
  // P2OUT |= BIT6;

  // Configure clock system
  Initialize_Clock_System();

  // Graphics functions
  Graphics_Context g_sContext;        // Declare a graphic library
  context Crystalfontz128x128_Init(); // Initialize the display
  // Set the screen orientation
  Crystalfontz128x128_SetOrientation(0);
  // Initialize the context
  Graphics_initContext(&g_sContext, &g_sCrystalfontz128x128);
  // 1/2) Set new background and foreground colors
  Graphics_setBackgroundColor(
      &g_sContext, GRAPHICS_COLOR_BLACK); // Set new background
    color to white
  Graphics_setForegroundColor(
      &g_sContext, GRAPHICS_COLOR_PINK); // Set new foreground
    color to blue
  while (1) {
    // FOR IMAGE DISPLAY (DISPLAY 2)
    // 1/2) Set new background and foreground colors
    Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
    // Set new background color to white
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_GOLD);
    // Set new foreground color to blue
    // Clear the screen
    Graphics_clearDisplay(&g_sContext);
    Graphics_drawImage(&g_sContext, &UCF_Logo, 1, 1);
    while ((P1IN & S1) != 0)
    {
```

```
      // wait
    }
    __delay_cycles(8000000);
    // FOR COUNTER AND SHAPES DISPLAY (DISPLAY 1)
    // 1/2) Set new background and foreground colors
    Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_WHITE);
    // Set new background color to white
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLUE);
    // Set new foreground color to blue
    // Set the default font for strings
    GrContextFontSet(&g_sContext, &g_sFontlucidasans8x15);
    // Clear the screen
    Graphics_clearDisplay(&g_sContext);
    // 3) Draw an outline circle
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED);
    // make circle outline red
    Graphics_drawCircle(&g_sContext, 25, 20, 10);
    // 3) Draw a filled circle
    Graphics_setForegroundColor(
        &g_sContext,
        GRAPHICS_COLOR_DARK_ORANGE); // make filled circle orange
    Graphics_fillCircle(&g_sContext, 50, 20, 10);
    // 3) Draw an outline rectangle
    Graphics_setForegroundColor(
        &g_sContext,
        GRAPHICS_COLOR_YELLOW); // make rectangle outline yellow
    const Graphics_Rectangle RectLoc = {
        .xMin = 15, .yMin = 40, .xMax = 35, .yMax = 65
    };
    Graphics_drawRectangle(&g_sContext, &RectLoc);
    // 3) Draw a filled rectangle
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_GREEN);
    // make filled rectangle green
    const Graphics_Rectangle RectLoc1 = {
        .xMin = 40, .yMin = 40, .xMax = 60, .yMax = 65};
    Graphics_fillRectangle(&g_sContext, &RectLoc1);
    // 3) Draw a horizontal line
    Graphics_setForegroundColor(
        &g_sContext,
        GRAPHICS_COLOR_DODGER_BLUE); // make horizontal line blue
    Graphics_drawLineH(&g_sContext, 15, 60, 80);
    // 4) Display counter
    // Set the default font for strings
    Graphics_setForegroundColor(
        &g_sContext,
        GRAPHICS_COLOR_HOT_PINK); // make horizontal line blue
    GrContextFontSet(&g_sContext, &g_sFontlucidabright6x12);
    Graphics_drawStringCentered(&g_sContext, "ctr",
    AUTO_STRING_LENGTH, 65, 102,
                                OPAQUE_TEXT); // Display number
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
    // make horizontal line blue
    GrContextFontSet(&g_sContext, &g_sFontfixed7x13);
    Graphics_drawStringCentered(&g_sContext, "Font",
    AUTO_STRING_LENGTH, 90, 50,
                                OPAQUE_TEXT); // Display number
    char str[3];
```

```
136     int i = 0;
137     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,
144                                   OPAQUE_TEXT); // Display number
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 }
```

10.3 Driving a PWM Signal on the Pin

The brightness level, thankfully, is changed via the changing of TA0CCR1 in all the ranges.

```
// Part 9.3 Code ----------------------------
#include "Grlib/grlib/grlib.h"    // Graphics library (grlib)
#include "LcdDriver/lcd_driver.h" // LCD driver
#include "msp430fr6989.h"
#include <math.h>
#include <stdio.h>

#define redLED BIT0
#define greenLED BIT7
#define S1 BIT1
#define S2 BIT2

// For I2C
#define MANUFACTURERIDREG 0x7E // Manufacturer Register ID
#define DEVICEIDREG 0x7F       // Device Register ID
#define I2CADDR 0x44           // I2C Address

// For reading Sensor data
#define CONFIGREG 0x01
#define RESULTREG 0x00

void Initialize_Clock_System()
{
  // DCO frequency = 16 MHz
  // MCLK = fDCO/1 = 16 MHz
  // SMCLK = fDCO/1 = 16 MHz
  // Activate memory wait state
  FRCTL0 = FRCTLPW | NWAITS_1; // Wait state=1
  CSCTL0 = CSKEY;
  // Set DCOFSEL to 4 (3-bit field)
  CSCTL1 &= ~DCOFSEL_7;
  CSCTL1 |= DCOFSEL_4;
  // Set DCORSEL to 1 (1-bit field)
  CSCTL1 |= DCORSEL;
  // Change the dividers to 0 (div by 1)
  CSCTL3 &= ~(DIVS2 | DIVS1 | DIVS0); // DIVS=0 (3-bit)
  CSCTL3 &= ~(DIVM2 | DIVM1 | DIVM0); // DIVM=0 (3-bit)
  CSCTL0_H = 0;
  return;
}

// I2C Initialization Function (for lux sensor)
void Initialize_I2C(void)
{
  // Configure the MCU in Master mode
  // Configure pins to I2C functionality
  // (UCB1SDA same as P4.0) (UCB1SCL same as P4.1)
  // (P4SEL1=11, P4SEL0=00) (P4DIR=xx)
  P4SEL1 |= (BIT1 | BIT0);
  P4SEL0 &= ~(BIT1 | BIT0);
  // Enter reset state and set all fields in this register to zero
  UCB1CTLW0 = UCSWRST;
```

```
  // Fields that should be nonzero are changed below
  // (Master Mode: UCMST) (I2C mode: UCMODE_3)
  // (Synchronous mode: UCSYNC)
  // (UCSSEL 1:ACLK, 2,3:SMCLK)
  UCB1CTLW0 |= UCMST | UCMODE_3 | UCSYNC | UCSSEL_3;
  // Clock frequency: 16 MHz, 16M/320K =
  UCB1BRW = 50;
  // Chip Data Sheet p. 53 (Should be 400 KHz max)
  // Exit the reset mode at the end of the configuration
  UCB1CTLW0 &= ~UCSWRST;
}

// Read a word (2 bytes) from I2C (address, register)
int i2c_read_word(unsigned char i2c_address,
                  unsigned char i2c_reg,
                  unsigned int *data)
{
  // Intialize to ensure successful
  unsigned char byte1 = 0;
  unsigned char byte2 = 0;
  reading UCB1I2CSA = i2c_address;    // Set address
  UCB1IFG &= ~UCTXIFG0;
  // Transmit a byte (the internal register address)
  UCB1CTLW0 |= UCTR;
  UCB1CTLW0 |= UCTXSTT;
  while ((UCB1IFG & UCTXIFG0) == 0)
  {
    // Wait for flag to raise
  }
  UCB1TXBUF = i2c_reg; // Write in the TX buffer
  while ((UCB1IFG & UCTXIFG0) == 0)
  {
    // Buffer copied to shift
  }
  register;
  // Tx in progress; set Stop bit
  // Repeated Start
  UCB1CTLW0 &= ~UCTR;
  UCB1CTLW0 |= UCTXSTT;
  // Read the first byte
  while ((UCB1IFG & UCRXIFG0) == 0)
  {
    // Wait for flag to raise
  }
  byte1 = UCB1RXBUF;
  // Assert the Stop signal bit before receiving the last byte
  UCB1CTLW0 |= UCTXSTP;
  // Read the second byte
  while ((UCB1IFG & UCRXIFG0) == 0)
  {
    // Wait for flag to raise
  }
  byte2 = UCB1RXBUF;
  while ((UCB1CTLW0 & UCTXSTP) != 0)
  {
    // wait
  }
```

```
  while ((UCB1STATW & UCBBUSY) != 0)
  {
    // wait
  }
  *data = (byte1 << 8) | (byte2 & (unsigned int)0x00FF);
  return 0;
}

// Write a word (2 bytes) to I2C (address, register)
int i2c_write_word(unsigned char i2c_address,
                   unsigned char i2c_reg,
                   unsigned int data)
{
  unsigned char byte1, byte2;
  UCB1I2CSA = i2c_address;    // Set I2C address
  byte1 = (data >> 8) & 0xFF; // MSByte
  byte2 = data & 0xFF;        // LSByte
  UCB1IFG &= ~UCTXIFG0;
  // Write 3 bytes
  UCB1CTLW0 |= (UCTR | UCTXSTT);
  while ((UCB1IFG & UCTXIFG0) == 0)
  {
  }
  UCB1TXBUF = i2c_reg;
  while ((UCB1IFG & UCTXIFG0) == 0)
  {
  }
  UCB1TXBUF = byte1;
  while ((UCB1IFG & UCTXIFG0) == 0)
  {
  }
  UCB1TXBUF = byte2;
  while ((UCB1IFG & UCTXIFG0) == 0)
  {
  }
  UCB1CTLW0 |= UCTXSTP;
  while ((UCB1CTLW0 & UCTXSTP) != 0)
  {
  }
  while ((UCB1STATW & UCBBUSY) != 0)
  {
  }
  return 0;
}

// Initializing Sensor
void Initialize_SENS()
{
  // Configuration for Sensor based on lab manual
  unsigned volatile int configuration_value = 0x7604;
  i2c_write_word(I2CADDR, CONFIGREG, configuration_value);
}

void main(void)
{
  char mystring[20];
  // Configure WDT & GPIO
```

```
  WDTCTL = WDTPW | WDTHOLD;
  PM5CTL0 &= ~LOCKLPM5;
  // Configure LEDs
  P1DIR |= redLED;
  P9DIR |= greenLED;
  P1OUT &= ~redLED;
  P9OUT &= ~greenLED;
  // Configure buttons
  P1DIR &= ~(S1 | S2);
  P1REN |= (S1 | S2);
  P1OUT |= (S1 | S2);
  P1IFG &= ~(S1 | S2); // Flags are used for latched polling
  // Set the LCD backlight to highest level
  // P2DIR |= BIT6;
  // P2OUT |= BIT6;

  // Configure clock system
  Initialize_Clock_System();

  // Initializing the I2C
  Initialize_I2C();

  // Initializing Sensor with new function based on configuration
  Initialize_SENS();

  // Graphics functions
  Graphics_Context g_sContext;        // Declare a graphic library
  context Crystalfontz128x128_Init(); // Initialize the display
  // Set the screen orientation
  Crystalfontz128x128_SetOrientation(0);
  // Initialize the context
  Graphics_initContext(&g_sContext, &g_sCrystalfontz128x128);
  // Set background and foreground colors
  Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
  Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_WHITE);
  // Set the default font for strings
  GrContextFontSet(&g_sContext, &g_sFontFixed6x8);
  // Clear the screen
  Graphics_clearDisplay(&g_sContext);

  // Set up Optical Counter Display
  sprintf(mystring, "Optical Counter");
  // show "Optical Sensor"
  Graphics_drawStringCentered(&g_sContext,
                              mystring,
                              AUTO_STRING_LENGTH,
                              64,
                              40,
                              OPAQUE_TEXT);
  Graphics_setForegroundColor(&g_sContext,
                              GRAPHICS_COLOR_GOLDENROD);
  sprintf(mystring, "lux");
  // show "lux"
  Graphics_drawStringCentered(&g_sContext,
                              mystring,
                              AUTO_STRING_LENGTH,
                              90,
```

```
                              80,
                              OPAQUE_TEXT);
  // Make rectangular outline for optical sensor value
  Graphics_setForegroundColor(&g_sContext,
                              GRAPHICS_COLOR_GOLDENROD);
  const Graphics_Rectangle RectLoc = {
      .xMin = 10, .yMin = 100,
      .xMax = (128 - 10), .yMax = (100 + 10)
  };
  Graphics_drawRectangle(&g_sContext, &RectLoc);
  int LUX_Val = 0;
  int Object_Counter = 0;
  char str[5];
  bool update_object_num = 0;
  while (1)
  {
    // Storing sensor value based on value in result
    // reg and lux reading from sensor
    i2c_read_word(I2CADDR, RESULTREG, &LUX_Val);
    // make counter purple
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED);
    LUX_Val = LUX_Val * 1.28;
    if (LUX_Val >= 1000)
    {
      LUX_Val = 1000; // Caps LUX_Val at 1000 (based on manual)
    }
    if ((LUX_Val <= 200) && (update_object_num == 0))
    {
      // For object, count the object
      detection Object_Counter++;
      update_object_num = 1;
    }
    else if ((LUX_Val > 200) && (update_object_num == 1))
    {
      // Won't update again until object passes
      update_object_num = 0;
    }
    sprintf(str, "%d", Object_Counter); // Convert number to string
    Graphics_drawStringCentered(&g_sContext,
                                str,
                                AUTO_STRING_LENGTH,
                                64,
                                60,
                                OPAQUE_TEXT); // Display number
    // Convert number to string
    sprintf(str, "%4d", LUX_Val);
    Graphics_drawStringCentered(&g_sContext,
                                str,
                                AUTO_STRING_LENGTH,
                                62,
                                80,
                                OPAQUE_TEXT); // Display number
    __delay_cycles(500000);                   // wait a second
    // Make rectangular outline for showing optical sensor value
    if (round(LUX_Val / 9.434) + 1 < 105)
    {
      Graphics_setForegroundColor(&g_sContext,
```

```
                                    GRAPHICS_COLOR_BLACK);
      const Graphics_Rectangle RectLoc = {
        .xMin = 11 + round(LUX_Val / 9.434) + 1,
        .yMin = 101,
        .xMax = (128 - 11),
        .yMax = (100 + 9)
      };
      Graphics_fillRectangle(&g_sContext, &RectLoc);
    }
    // Make rectangular outline for showing optical sensor value
    Graphics_setForegroundColor(&g_sContext,
                                GRAPHICS_COLOR_GOLDENROD);
    const Graphics_Rectangle RectLoc1 = {
      .xMin = 11,
      .yMin = 101,
      .xMax = 11 + round(LUX_Val / 9.434),
      .yMax = (100 + 9)
    };
    Graphics_fillRectangle(&g_sContext, &RectLoc1);
  }
}
```

10.4 Timer Input Capture

The button push lasts [**answer here**]

```
// Part 9.3 Code ---------------------------
#include "Grlib/grlib/grlib.h"    // Graphics library (grlib)
#include "LcdDriver/lcd_driver.h" // LCD driver
#include "msp430fr6989.h"
#include <math.h>
#include <stdio.h>

#define redLED BIT0
#define greenLED BIT7
#define S1 BIT1
#define S2 BIT2

// For I2C
#define MANUFACTURERIDREG 0x7E // Manufacturer Register ID
#define DEVICEIDREG 0x7F       // Device Register ID
#define I2CADDR 0x44           // I2C Address

// For reading Sensor data
#define CONFIGREG 0x01
#define RESULTREG 0x00

void Initialize_Clock_System()
{
  // DCO frequency = 16 MHz
  // MCLK = fDCO/1 = 16 MHz
  // SMCLK = fDCO/1 = 16 MHz
  // Activate memory wait state
  FRCTL0 = FRCTLPW | NWAITS_1; // Wait state=1
  CSCTL0 = CSKEY;
  // Set DCOFSEL to 4 (3-bit field)
  CSCTL1 &= ~DCOFSEL_7;
  CSCTL1 |= DCOFSEL_4;
  // Set DCORSEL to 1 (1-bit field)
  CSCTL1 |= DCORSEL;
  // Change the dividers to 0 (div by 1)
  CSCTL3 &= ~(DIVS2 | DIVS1 | DIVS0); // DIVS=0 (3-bit)
  CSCTL3 &= ~(DIVM2 | DIVM1 | DIVM0); // DIVM=0 (3-bit)
  CSCTL0_H = 0;
  return;
}

// I2C Initialization Function (for lux sensor)
void Initialize_I2C(void)
{
  // Configure the MCU in Master mode
  // Configure pins to I2C functionality
  // (UCB1SDA same as P4.0) (UCB1SCL same as P4.1)
  // (P4SEL1=11, P4SEL0=00) (P4DIR=xx)
  P4SEL1 |= (BIT1 | BIT0);
  P4SEL0 &= ~(BIT1 | BIT0);
  // Enter reset state and set all fields in this register to zero
  UCB1CTLW0 = UCSWRST;
  // Fields that should be nonzero are changed below
  // (Master Mode: UCMST) (I2C mode: UCMODE_3)
```

```
  // (Synchronous mode: UCSYNC)
  // (UCSSEL 1:ACLK, 2,3:SMCLK)
  UCB1CTLW0 |= UCMST | UCMODE_3 | UCSYNC | UCSSEL_3;
  // Clock frequency: 16 MHz, 16M/320K =
  UCB1BRW = 50;
  // Chip Data Sheet p. 53 (Should be 400 KHz max)
  // Exit the reset mode at the end of the configuration
  UCB1CTLW0 &= ~UCSWRST;
}

// Read a word (2 bytes) from I2C (address, register)
int i2c_read_word(unsigned char i2c_address,
                  unsigned char i2c_reg,
                  unsigned int *data)
{
  // Intialize to ensure successful
  unsigned char byte1 = 0;
  unsigned char byte2 = 0;
  reading UCB1I2CSA = i2c_address;    // Set address
  UCB1IFG &= ~UCTXIFG0;
  // Transmit a byte (the internal register address)
  UCB1CTLW0 |= UCTR;
  UCB1CTLW0 |= UCTXSTT;
  while ((UCB1IFG & UCTXIFG0) == 0)
  {
    // Wait for flag to raise
  }
  UCB1TXBUF = i2c_reg; // Write in the TX buffer
  while ((UCB1IFG & UCTXIFG0) == 0)
  {
    // Buffer copied to shift
  }
  register;
  // Tx in progress; set Stop bit
  // Repeated Start
  UCB1CTLW0 &= ~UCTR;
  UCB1CTLW0 |= UCTXSTT;
  // Read the first byte
  while ((UCB1IFG & UCRXIFG0) == 0)
  {
    // Wait for flag to raise
  }
  byte1 = UCB1RXBUF;
  // Assert the Stop signal bit before receiving the last byte
  UCB1CTLW0 |= UCTXSTP;
  // Read the second byte
  while ((UCB1IFG & UCRXIFG0) == 0)
  {
    // Wait for flag to raise
  }
  byte2 = UCB1RXBUF;
  while ((UCB1CTLW0 & UCTXSTP) != 0)
  {
    // wait
  }
  while ((UCB1STATW & UCBBUSY) != 0)
  {
```

```
    // wait
  }
  *data = (byte1 << 8) | (byte2 & (unsigned int)0x00FF);
  return 0;
}

// Write a word (2 bytes) to I2C (address, register)
int i2c_write_word(unsigned char i2c_address,
                   unsigned char i2c_reg,
                   unsigned int data)
{
  unsigned char byte1, byte2;
  UCB1I2CSA = i2c_address;    // Set I2C address
  byte1 = (data >> 8) & 0xFF; // MSByte
  byte2 = data & 0xFF;        // LSByte
  UCB1IFG &= ~UCTXIFG0;
  // Write 3 bytes
  UCB1CTLW0 |= (UCTR | UCTXSTT);
  while ((UCB1IFG & UCTXIFG0) == 0)
  {
  }
  UCB1TXBUF = i2c_reg;
  while ((UCB1IFG & UCTXIFG0) == 0)
  {
  }
  UCB1TXBUF = byte1;
  while ((UCB1IFG & UCTXIFG0) == 0)
  {
  }
  UCB1TXBUF = byte2;
  while ((UCB1IFG & UCTXIFG0) == 0)
  {
  }
  UCB1CTLW0 |= UCTXSTP;
  while ((UCB1CTLW0 & UCTXSTP) != 0)
  {
  }
  while ((UCB1STATW & UCBBUSY) != 0)
  {
  }
  return 0;
}

// Initializing Sensor
void Initialize_SENS()
{
  // Configuration for Sensor based on lab manual
  unsigned volatile int configuration_value = 0x7604;
  i2c_write_word(I2CADDR, CONFIGREG, configuration_value);
}

void main(void)
{
  char mystring[20];
  // Configure WDT & GPIO
  WDTCTL = WDTPW | WDTHOLD;
  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);
```

```
// Make rectangular outline for optical sensor value
Graphics_setForegroundColor(&g_sContext,
                            GRAPHICS_COLOR_GOLDENROD);
const Graphics_Rectangle RectLoc = {
    .xMin = 10, .yMin = 100,
    .xMax = (128 - 10), .yMax = (100 + 10)
};
Graphics_drawRectangle(&g_sContext, &RectLoc);
int LUX_Val = 0;
int Object_Counter = 0;
char str[5];
bool update_object_num = 0;
while (1)
{
  // Storing sensor value based on value in result
  // reg and lux reading from sensor
  i2c_read_word(I2CADDR, RESULTREG, &LUX_Val);
  // make counter purple
  Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED);
  LUX_Val = LUX_Val * 1.28;
  if (LUX_Val >= 1000)
  {
    LUX_Val = 1000; // Caps LUX_Val at 1000 (based on manual)
  }
  if ((LUX_Val <= 200) && (update_object_num == 0))
  {
    // For object, count the object
    detection Object_Counter++;
    update_object_num = 1;
  }
  else if ((LUX_Val > 200) && (update_object_num == 1))
  {
    // Won't update again until object passes
    update_object_num = 0;
  }
  sprintf(str, "%d", Object_Counter); // Convert number to string
  Graphics_drawStringCentered(&g_sContext,
                              str,
                              AUTO_STRING_LENGTH,
                              64,
                              60,
                              OPAQUE_TEXT); // Display number
  // Convert number to string
  sprintf(str, "%4d", LUX_Val);
  Graphics_drawStringCentered(&g_sContext,
                              str,
                              AUTO_STRING_LENGTH,
                              62,
                              80,
                              OPAQUE_TEXT); // Display number
  __delay_cycles(500000);                   // wait a second
  // Make rectangular outline for showing optical sensor value
  if (round(LUX_Val / 9.434) + 1 < 105)
  {
    Graphics_setForegroundColor(&g_sContext,
                                GRAPHICS_COLOR_BLACK);
    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 implmented 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 teh following:

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

Conclusion