Lab 7 –

# Youssef Samwel

# [yo800238@ucf.edu](mailto:yo800238@ucf.edu)

# EEL4742C Embedded Systems

Prof. Dr. Zakhia Abichar - Section 00419

# 3/14/2024



# **Project Description**

In this lab, we will learn the I2C communication interface and apply it by communicating with the light sensor that’s on the on Educational BoosterPack board.

# **2.0 Experiment Code**

#include <msp430fr6989.h>

#include <stdint.h>

#include <stdbool.h>

#define FLAGS UCA1IFG      // Contains the transmit & receive flags

#define RXFLAG UCRXIFG     // Receive flag

#define TXFLAG UCTXIFG     // Transmit flag

#define TXBUFFER UCA1TXBUF // Transmit buffer

#define RXBUFFER UCA1RXBUF // Receive buffer

#define BUTTON1 BIT1

#define BUTTON2 BIT2

int i2c\_read\_word(unsigned char i2c\_addrs, unsigned char i2c\_reg, unsigned int \*data); //

int i2c\_write\_word(unsigned char i2c\_addrs, unsigned char i2c\_reg, unsigned int data); //

void Initialize\_I2C(void);

void Initialize\_UART(void);

void config\_ACLK\_to\_32KHz\_crystal();

void uart\_write\_uint16(unsigned int n);

void uart\_write\_string(const char \*str);

void uart\_write\_nstring(void \*pStr, uint16\_t size);

bool IsButton1Pressed()

{

    return (~P1IN & BUTTON1) ? true : false;

}

bool IsButton2Pressed()

{

    return (~P1IN & BUTTON2) ? true : false;

}

void terminal\_clear()

{

    uart\_write\_string("\033[2J");

}

void terminal\_reset\_cursor()

{

    uart\_write\_string("\033[1;1H");

}

// Read a word (2 bytes) from I2C (address, register)

int i2c\_read\_word(unsigned char i2c\_address, unsigned char i2c\_reg, unsigned int \*data)

{

    unsigned char byte1, byte2;

    // Initialize the bytes to make sure data is received every time

    byte1 = 111;

    byte2 = 111;

    //\*\*\*\*\*\*\*\*\*\* Write Frame #1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    UCB1I2CSA = i2c\_address; // Set I2C address

    UCB1IFG &= ~UCTXIFG0;

    UCB1CTLW0 |= UCTR;    // Master writes (R/W bit = Write)

    UCB1CTLW0 |= UCTXSTT; // Initiate the Start Signal

    while ((UCB1IFG & UCTXIFG0) == 0)

    {

    }

    UCB1TXBUF = i2c\_reg; // Byte = register address

    while ((UCB1CTLW0 & UCTXSTT) != 0)

    {

    }

    if ((UCB1IFG & UCNACKIFG) != 0)

        return -1;

    UCB1CTLW0 &= ~UCTR;   // Master reads (R/W bit = Read)

    UCB1CTLW0 |= UCTXSTT; // Initiate a repeated Start Signal

    //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    //\*\*\*\*\*\*\*\*\*\* Read Frame #1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    while ((UCB1IFG & UCRXIFG0) == 0)

    {

    }

    byte1 = UCB1RXBUF;

    //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    //\*\*\*\*\*\*\*\*\*\* Read Frame #2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    while ((UCB1CTLW0 & UCTXSTT) != 0)

    {

    }

    UCB1CTLW0 |= UCTXSTP; // Setup the Stop Signal

    while ((UCB1IFG & UCRXIFG0) == 0)

    {

    }

    byte2 = UCB1RXBUF;

    while ((UCB1CTLW0 & UCTXSTP) != 0)

    {

    }

    //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    // Merge the two received bytes

    \*data = ((byte1 << 8) | (byte2 & 0xFF));

    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;

    byte1 = (data >> 8) & 0xFF; // MSByte

    byte2 = data & 0xFF;        // LSByte

    UCB1I2CSA = i2c\_address;    // Set I2C address

    UCB1CTLW0 |= UCTR;          // Master writes (R/W bit = Write)

    UCB1CTLW0 |= UCTXSTT;       // Initiate the Start Signal

    while ((UCB1IFG & UCTXIFG0) == 0)

    {

    }

    UCB1TXBUF = i2c\_reg; // Byte = register address

    while ((UCB1CTLW0 & UCTXSTT) != 0)

    {

    }

    while ((UCB1IFG & UCTXIFG0) == 0)

    {

    }

    //\*\*\*\*\*\*\*\*\*\* Write Byte #1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    UCB1TXBUF = byte1;

    while ((UCB1IFG & UCTXIFG0) == 0)

    {

    }

    //\*\*\*\*\*\*\*\*\*\* Write Byte #2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    UCB1TXBUF = byte2;

    while ((UCB1IFG & UCTXIFG0) == 0)

    {

    }

    UCB1CTLW0 |= UCTXSTP;

    while ((UCB1CTLW0 & UCTXSTP) != 0)

    {

    }

    return 0;

}

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: SMCLK/8 = 1 MHz/8 = 125 KHz

    UCB1BRW = 8;

    // Chip Data Sheet p. 53 (Should be 400 KHz max)

    // Exit the reset mode at the end of the configuration

    UCB1CTLW0 &= ~UCSWRST;

}

// Configure UART to the popular configuration

// 9600 baud, 8-bit data, LSB first, no parity bits, 1 stop bit

// no flow control, oversampling reception

// Clock: SMCLK @ 1 MHz (1,000,000 Hz)

void Initialize\_UART(void)

{

    // Configure pins to UART functionality

    P3SEL1 &= ~(BIT4 | BIT5);

    P3SEL0 |= (BIT4 | BIT5);

    // Main configuration register

    UCA1CTLW0 = UCSWRST; // Engage reset; change all the fields to zero

    // Most fields in this register, when set to zero, correspond to the

    // popular configuration

    UCA1CTLW0 |= UCSSEL\_\_SMCLK; // Set clock to SMCLK

    // Configure the clock dividers and modulators (and enable oversampling)

    UCA1BRW = 6; // divider

    // Modulators: UCBRF = 8 = 1000 --> UCBRF3 (bit #3)

    // UCBRS = 0x20 = 0010 0000 = UCBRS5 (bit #5)

    UCA1MCTLW = UCBRF3 | UCBRS5 | UCOS16;

    // Exit the reset state

    UCA1CTLW0 &= ~UCSWRST;

}

// Configures ACLK to 32 KHz crystal

void config\_ACLK\_to\_32KHz\_crystal()

{

    // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz

    // Reroute pins to LFXIN/LFXOUT functionality

    PJSEL1 &= ~BIT4;

    PJSEL0 |= BIT4;

    // Wait until the oscillator fault flags remain cleared

    CSCTL0 = CSKEY; // Unlock CS registers

    do

    {

        CSCTL5 &= ~LFXTOFFG; // Local fault flag

        SFRIFG1 &= ~OFIFG;   // Global fault flag

    } while ((CSCTL5 & LFXTOFFG) != 0);

    CSCTL0\_H = 0; // Lock CS registers

    return;

}

void uart\_write\_uint16(unsigned int number)

{

    // Max number of digits in a uint16\_t is 5

        int divisor = 10000;

        while (divisor > 0) {

            int digit = number / divisor;

            if (digit > 0 || divisor == 1) {

                uart\_write\_char('0' + digit);

            }

            number %= divisor;

            divisor /= 10;

        }

}

void uart\_write\_string(const char \*str)

{

    uint16\_t length = strlen(str);

    uint16\_t i;

    for (i = 0; i < length; i++)

    {

        uart\_write\_char(str[i]);

    }

}

void uart\_write\_nstring(void \*pStr, uint16\_t size)

{

    uint8\_t \*str = (uint8\_t \*)pStr;

    uint16\_t i = 0;

    for (i = 0; i < size; i++)

    {

        uart\_write\_char(str[i]);

    }

}

void uart\_write\_char(unsigned char ch)

{

    // Wait for any ongoing transmission to complete

    while ((FLAGS & TXFLAG) == 0)

    {

    }

    // Copy the byte to the transmit buffer

    TXBUFFER = ch; // Tx flag goes to 0 and Tx begins!

    return;

}

// The function returns the byte; if none received, returns null character

uint8\_t uart\_read\_char(void)

{

    uint8\_t temp;

    // Return null character (ASCII=0) if no byte was received

    if ((FLAGS & RXFLAG) == 0)

        return 0;

    // Otherwise, copy the received byte (this clears the flag) and return it

    temp = RXBUFFER;

    return temp;

}

#define RESULT 0x00

#define CONFIGURATION 0x01

#define LOW\_LIMIT 0x02

#define HIGH\_LIMIT 0x03

#define MANUFACTURER\_ID 0x7E

#define DEVICE\_ID 0x7F

#define SENSOR\_GND 0x44

#define VK\_ENTER 0x0A

#define VK\_BACKSPACE 0x08

uint16\_t sensor\_read\_lux()

{

    uint16\_t lux = 0;

    i2c\_read\_word(SENSOR\_GND, RESULT, &lux);

    return 1.28 \* lux;

}

void part1\_read\_deviceinfo()

{

    uint16\_t manufacturerId = 0, deviceId = 0, lux = 0;

    i2c\_read\_word(0x44, 0x7E, &manufacturerId);

    i2c\_read\_word(0x44, 0x7F, &deviceId);

    for (;;)

    {

        uart\_write\_string("Manufacturer Id: ");

        uart\_write\_nstring(&manufacturerId, 2);

        uart\_write\_string(", Device Id: ");

        uart\_write\_uint16(deviceId);

        uart\_write\_string("\r\n");

        \_\_delay\_cycles(1e6);

    }

}

void sensor\_config\_register()

{

    // configure the sensor: RN = 0b0111, CT = 0, M = 0b11,  ME = 1

    uint16\_t sensorconfig = 0b0111011000000100;

    i2c\_write\_word(SENSOR\_GND, CONFIGURATION, sensorconfig);

}

void part2\_read\_lux()

{

    sensor\_config\_register();

    for (;;)

    {

        uart\_write\_string("LUX: ");

        uart\_write\_uint16(sensor\_read\_lux());

        uart\_write\_string("\r\n");

    }

}

void add\_minute(uint8\_t \*hour, uint8\_t \*minute)

{

    \*minute += 1;

    if (\*minute >= 60)

    {

        \*minute = 0;

        \*hour += 1;

    }

    if (\*hour > 12)

    {

        \*hour = 1;

    }

}

void read\_time(uint8\_t\* pOutHour, uint8\_t\* pOutMinute) {

    uart\_write\_string("Enter the time...(3 or 4 digits then hit Enter)\r\n");

        uart\_write\_string("Time is set to ");

        char szBuffer[5] = {0};

        char currentKey = VK\_ENTER;

        uint8\_t index = 0;

        do

        {

            currentKey = uart\_read\_char();

            if (currentKey)

            {

                if (currentKey == VK\_BACKSPACE || currentKey == 127)

                {

                    szBuffer[index] = 0;

                    if (index > 0)

                    {

                        // move cursor back, clear character, move cursor back

                        uart\_write\_string("\b \b");

                    }

                }

                else

                {

                    // is this an acceptable key?

                    if (currentKey >= 48 && currentKey <= 58)

                    {

                        // yes, key is a number or colon

                        uart\_write\_char(currentKey); // write key again so it is displayed on UART terminal.

                        if (index == 5)

                            index = 4; // prevent buffer overflow

                        szBuffer[index] = currentKey;

                        index++;

                    }

                }

            }

        } while (currentKey != '\r' && currentKey != '\n');

        uart\_write\_string("\r\n");

        // parse input

        // if user did not input time, the default time is 12:00

        uint8\_t hour = 0;

        uint8\_t minute = 0;

        if (index)

        {

            uint8\_t i;

            bool meetColon = false;

            for (i = 0; i < index; i++)

            {

                char key = szBuffer[i];

                // input key is either a number or colon

                if (key == ':')

                {

                    if (meetColon)

                    {

                        goto parse\_error;

                    }

                    meetColon = true;

                }

                else

                {

                    // convert ascii number to integer

                    uint8\_t number = key - '0';

                    if (!meetColon)

                    {

                        // hour

                        hour = (hour \* 10) + number;

                        if (hour > 12)

                        {

                            // invalid time

                            goto parse\_error;

                        }

                    }

                    else

                    {

                        // minute

                        minute = (minute \* 10) + number;

                        if (minute > 60)

                        {

                            // invalid time

                            goto parse\_error;

                        }

                    }

                }

                continue;

            parse\_error:

                // invalid input use default time

                hour = 12;

                minute = 0;

                uart\_write\_string("Invalid time input, using default time (12:00).\r\n");

                break;

            }

        }

        uart\_write\_uint16(hour);

        uart\_write\_char(':');

        uart\_write\_uint16(minute);

        uart\_write\_string("\r\n");

        \*pOutHour = hour;

        \*pOutMinute = minute;

}

void part3\_application()

{

    uint8\_t hour = 0;

    uint8\_t minute = 0;

    sensor\_config\_register();

    // Set input direction for buttons

    P1DIR &= ~(BUTTON1 | BUTTON2);

    // enable pull-up resistors to avoid false triggers caused by static fields

    P1REN |= BUTTON1 | BUTTON2;

    // pull high buttons

    P1OUT |= BUTTON1 | BUTTON2;

    \_\_delay\_cycles(1e5);

    // wait until button 1 and 2 are pressed to begin LUX

    //while (P1IN & BUTTON1 || P1IN & BUTTON2)

    //{

    //}

    // begin LUX

    uart\_write\_string("\*\*\* Lux Logger \*\*\*\r\n\r\n");

    read\_time(&hour, &minute);

    int32\_t baseline\_lux = sensor\_read\_lux();

    if (baseline\_lux < 10)

        baseline\_lux = 10; // clamp to 10 to avoid underflow problems

    for (;;)

    {

        if(IsButton1Pressed()) {

            read\_time(&hour, &minute);

        }

        int32\_t lux = sensor\_read\_lux();

        if (hour > 9)

        {

            uart\_write\_uint16(hour);

        }

        else

        {

            uart\_write\_char(' ');

            uart\_write\_uint16(hour);

        }

        uart\_write\_char(':');

        if (minute < 10)

        {

            uart\_write\_char('0');

            uart\_write\_uint16(minute);

        }

        else

        {

            uart\_write\_uint16(minute);

        }

        uart\_write\_char('\t');

        uart\_write\_uint16(lux);

        uart\_write\_string(" lux");

        if (lux >= baseline\_lux + 10)

        {

            uart\_write\_string(" (+");

            uart\_write\_uint16(lux - baseline\_lux);

            uart\_write\_string(")   <Up>");

            baseline\_lux = lux;

        }

        else if (lux <= baseline\_lux - 10 && (baseline\_lux - 10) >= 0)

        {

            uart\_write\_string(" (-");

            uart\_write\_uint16(baseline\_lux - lux);

            uart\_write\_string(")   <Down>");

            baseline\_lux = lux;

        }

        uart\_write\_string("\r\n");

        \_\_delay\_cycles(5e6); // wait one minute

        add\_minute(&hour, &minute);

    }

}

/\*\*

 \* main.c

 \*/

int main(void)

{

    WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer

    PM5CTL0 &= ~LOCKLPM5;     // Enable the GPIO pins

    Initialize\_UART();

    Initialize\_I2C();

    terminal\_clear();

    terminal\_reset\_cursor();

    part3\_application();

    return 0;

}

The Initialize\_I2C and Initialize\_UART functions set up the I2C and UART modules, configuring pins and communication parameters such as clock frequency and baud rate. The I2C module is configured as a master device, allowing the microcontroller to communicate with other I2C devices, such as sensors, using a serial protocol. The UART module is configured for serial communication with a host computer terminal, allowing for text input and output.

The main functionality of the program involves reading data from an I2C sensor (configured as a lux sensor) and displaying it on the UART terminal. The program includes functions for reading sensor data, configuring sensor registers, and parsing user input from the terminal to set the time. Additionally, the program handles button inputs to trigger actions such as resetting the time or logging sensor data.

# **3.0 Student Q&A**

1. What possible addresses can the sensor have and how is the address chosen?

A screenshot of a computer

Description automatically generated

Figure 1

A diagram of a computer component

Description automatically generated

Figure 2

There are four possible addresses for the sensor as shown in the accompanying screenshots from the datasheet. The addressing mode is determined by the connection made to Pin 2 (known as ADDR pin).

2. What is the address of the sensor are wired on the BoosterPack board?

The sensor uses the GND address pin or specifically: 0x44 (1000100).

3. What is the value of the pull-up resistors on the I2C wires? Include a screenshot of the schematics highlighting the I2C address and the pull-up resistors.

10 kOhm

A diagram of a light sensor

Description automatically generated

4. What are the addresses of the Device ID and Manufacturer ID registers and what values are they supposed to return?

|  |  |  |
| --- | --- | --- |
| Register | Address | Expected Value |
| Manufacturer ID | 0x7E | 0x5449 (‘TI’ in ASCII) |
| Device ID | 0x7F | 0x3001 |

5. What is the address of the configuration register on the sensor?

0x01

6. What configuration value (hex) did you write to the sensor? Show how this value is formatted into bit fields.

    // configure the sensor: RN = 0b0111, CT = 0, M = 0b11, ME = 1

    uint16\_t config = 0x7604; // 0b0111011000000100;

    i2c\_write\_word(SENSOR\_GND, CONFIGURATION, config);

7. Do the sensor’s readings seem reasonable and consistent?

The readings are reasonable they remain relatively constant when the lighting in the room is constant. I tried using flashlight in my phone and it behaved as expected.

8. The light sensor has an address pin that allows customizing the I2C address. How many addresses are possible? What are they and how are they configured? Look in the sensor’s data sheet.

There are four possible I2C addressing modes. They are configured by connecting ADDR pin to ground.

A screenshot of a computer

Description automatically generated

9. According to the light sensor’s data sheet, what should be the value of the pull-up resistors on the I2C wires? Did the BoosterPack use the same values?

10kOhm should be the pull up wires, they match the datasheet.

A screenshot of a computer program

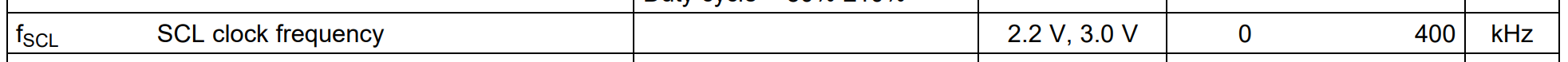
Description automatically generated

A diagram of a light sensor

Description automatically generated

10. What I2C clock frequency do each of the eUSCI module and the sensor support?

Only one eUSCI module supports I2C with clock frequency range of [0, 400] kHz. The sensor requires a minimum frequency of 10 kHz with support for fast mode of 400 kHz and a high-speed mode of 2.6 MHz.



# **4.0** **Conclusion**

In summary, Lab 7 on I2C communication provided practical experience in configuring and utilizing the I2C and UART modules within embedded systems. By interfacing with a light sensor, essential skills in sensor integration and serial communication were developed. Challenges encountered during the lab contributed to problem-solving abilities and deepened understanding of hardware-software interactions.