EEL 4742C: Embedded Systems
Name: Hamzah Ullah
Lab 9: Inter-Integrated Circuit(I2C) Communication

Introduction:

This lab introduces us to another communication protocol called Inter-integrated Circuit Communication or I2C for short. It allows us to easily connect multiple peripheral devices to the MCU and send information effectively between them through device addresses and the referenced register address.

*****PART 2 STARTS ON PAGE 7, QUESTIONS ARE ON PAGE 12.******

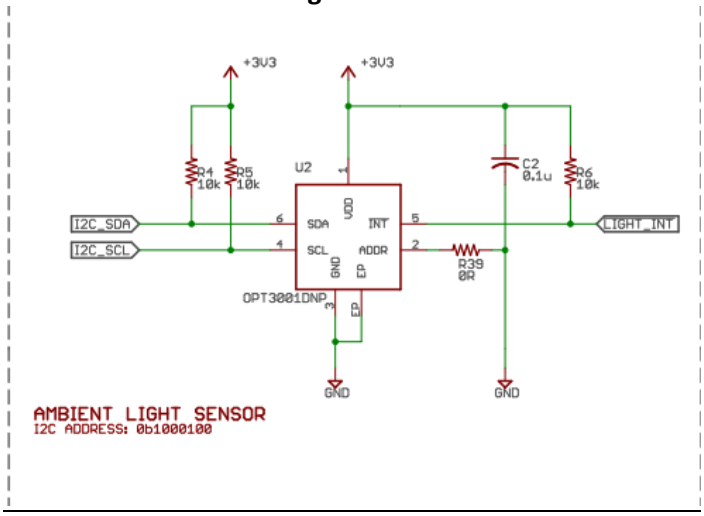
Part 1: I2C Transmission

This part of the lab introduces us to the basics of I2C mainly the serial data wire and the serial clock wire or SDA and SCL for short respectively. It gives a sample diagram of how the MCU and the device communicates and literally 'acknowledges' the transmission and either pushes the data acknowledgement forward or pulls it back for writing and reading.

The basic overview of how I2C works is the requires the address of the device and it pushes data through to it. If it wants to reference a specific register, it would read the register from the address first, than start again to write data from the device address (with the register in memory) or continue reading data in.

For the lab, after diverting the pins for the booster pack to the I2C on the MCU, we are simply supposed to read the light sensors manufacturers ID and device ID by referencing the address of the device, followed by the device of the register. The functions given in the Appendix have the format write/read(address,register,data) which we use to pull information from the light sensor. Below is the schematic to find the correct address.

Schematic for Ambient Light Sensor with address shown below (0x44).



Once the correct address is found, we must choose the correct register found in the data sheet for the Manufacturers ID and Device ID and save the information into the data address.

Below is the code for the device address displayed through the UART functions from previous labs.

Questions:

Manufactures address: 0x7E, it returns 5449h or TI in ASCII values.

Device address: 0x7F, it returns 3001h or 12889 in decimal.

Light Sensor Address: 0x44 or 100 0100 in binary

Pull up, 10k ohms

Transmitter Manufacturer ID and Device ID

```
#include <msp430.h>
#define FLAGS UCA1IFG //Contains the transmit and receive flags
```

```
#define RXFLAG UCRXIFG //receive flag
#define TXFLAG UCTXIFG //Transmit flag
#define TXBUFFER UCA1TXBUF //Transmit buffer
#define RXBUFFER UCA1RXBUF //Receive Buffer

/**
 * main.c
 */
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;   //enable GPIO pins
    unsigned int data;
    //Divert Pins to backchannel UART functionality
    //(UCA1TXD same as 3.4)(UCA1RXD same as 3.5)
    //P3SEL=00, P3SEL0 = 11 P2DIR=XX
    P3SEL1 &= ~(BIT4 | BIT5);
    P3SEL0 |= (BIT4 | BIT5);

    //Divert Pins to I2C Functionality
    P4SEL1 |= (BIT1 | BIT0);
    P4SEL0 &= ~(BIT1 | BIT0);

    Initialize_I2C();
    Initialize_UART();
    int i=0;
    while(i<1){
        data = 0;
    i2c_read_word(0x44,0x7E, &data);
    uart_write_char(data>>8);
    uart_write_char(data);
    i2c_read_word(0x44,0x7F, &data);
    uart_write_char('\n');
    uart_write_char('\r');
    uart_write_uint16(data);
    uart_write_char('\n');
    uart_write_char('\r');
    i++;
    }
}
//Configure eUSCI in I2C master mode
void Initialize_I2C(void)
{
    //Enter reset state before the configuration starts...
    UCB1CTLW0 |= UCSWRST;

    //Divert Pins to I2C Functionality
    P4SEL1 |= (BIT1 | BIT0);
    P4SEL0 &= ~(BIT1 | BIT0);

    //Keep all the default values excep thte fields below...
    //UCMode 3:I2C Master Mode UCSSEL1:ACLK, 2,3: SMCLK
    UCB1CTLW0 |= UCMODE_3 | UCMST | UCSSEL_3;

    //Clock divider = 8 (SMCLK @ 1.047Mhz /8 = 131khz
```

```
    UCB1BRW = 8;

    //Exit the reset Mode
    UCB1CTLW0 &= ~UCSWRST;
}
void Initialize_UART(void)
{
    //Divert Pins to UART functionality
    P3SEL1 &= ~(BIT4|BIT5);
    P3SEL0 |= (BIT4|BIT5);

    //Use smclk clock; leave the other settings default
    UCA1CTLW0 |= UCSSEL_2;

    //Configure clock dividers and modulators
    //UCBR = 6, UCBRF = 13, UCBRS = 0x22, UCOS16=1 (oversampling)
    UCA1BRW = 6;
    UCA1MCTLW = UCBRS5 | UCBRS1| UCBRF3| UCBRF2| UCBRF0| UCOS16;

    //exit the reset state(so transmission/reception cna begin)
    UCA1CTLW0 &= ~UCSWRST;
}
void uart_write_char(unsigned char ch)
{
    //wait for any ongoing transmission to complete
    while ((FLAGS & TXFLAG) == 0)
    {
    }

    //Write the byte to the transmit buffer
    TXBUFFER = ch;
}
void uart_write_string(char * str)
{
    int i;
    for(i=0; i < strlen(str); i++){
        uart_write_char(str[i]);
    }
}

void uart_write_uint16(unsigned int n)
{
    int temp;
    if(n>=10000)
    {
        temp = n/10000;
        uart_write_char(temp+48);
    }
    if(n>=1000)
    {
        temp = n/1000 % 10;
        uart_write_char(temp+48);
    }
    if(n>=100)
    {
```

```
        temp = n/100 % 10;
        uart_write_char(temp+48);
    }
    if(n>=10)
    {

        temp = (n/10) % 10;
        uart_write_char(temp+48);
    }
     n = n % 10;
     uart_write_char(n+48);

    //Write the byte to the transmit buffer
    //TXBUFFER = n;
}

void config_ACLK_to_32KHz_crystal()
{
    //By default, ACLK runs on LFMODCLK at 5MHz/128 = 39kHz

    //Reroute pins to LFXIN/LFXOUT functionality
    PJSEL1 &= ~BIT4;
    PJSEL0 |= BIT4;

    //Wait until the oscillator fault flags remain cleared
    CSCTL0 = CSKEY;
    do
    {
        CSCTL5 &= ~LFXTOFFG;    //local fault flag
        SFRIFG1 &= ~OFIFG;      //Global fault flag
    }
    while ((CSCTL5 & LFXTOFFG) != 0);

    CSCTL0_H = 0; //lock CS registers
    return;
}
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;
}
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)
    {
    }
//********** 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;
}
```

Part 2: Reading Measurements from the Light Sensor

This part of the lab uses the light sensor result register. We read the data from the result register 0x00 and display the output through UART. It simply requires us changing the value of the register function to the correct register. Furthermore, we are to change the configuration to the recommended settings which are shown below.

```
RN=0111b=7        The LSB bit is worth 1.28
CT=0              Result produced in 100 ms
M=11b=3           Continuous readings
ME=1              Mask (hide) the Exponent from the result
```

To do this we write the data to the configuration register by sending the hexadecimal equivalent of the 16 bit register. Below is the code.

Questions

Config register address : 0x01

Hex value written to sensor 0x7604

bits field: 0111 0110 0000 0100

The data did make sense with base values between 200-250 and rose up to 4095 and as low as 5 in the lab.

Shows Light reading every second as well as increments how many readings have appeared with a heading for differentiation of data.

```
#include <msp430.h>
#define FLAGS UCA1IFG //Contains the transmit and receive flags
#define RXFLAG UCRXIFG //receive flag
#define TXFLAG UCTXIFG //Transmit flag
#define TXBUFFER UCA1TXBUF //Transmit buffer
#define RXBUFFER UCA1RXBUF //Receive Buffer

/**
 * main.c
 */
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;    // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;    //enable GPIO pins
    unsigned int data;
    //Divert Pins to backchannel UART functionality
    //(UCA1TXD same as 3.4)(UCA1RXD same as 3.5)
    //P3SEL=00, P3SEL0 = 11 P2DIR=XX
    P3SEL1 &= ~(BIT4 | BIT5);
    P3SEL0 |= (BIT4 | BIT5);

    //Divert Pins to I2C Functionality
    P4SEL1 |= (BIT1 | BIT0);
    P4SEL0 &= ~(BIT1 | BIT0);
```

```
    //continuous mode every 1 second
    TA0CTL = TASSEL_1 | ID_0 | MC_2 | TACLR;
    TA0CTL &= ~TAIFG;
    //initialize UART/I2C
    Initialize_I2C();
    Initialize_UART();
    //Variables for header and infinite loop
    int i=0;
    int j=1;
    char head1[] = "Light Level";
    char head2[] = "Reading Number";
    i2c_write_word(0x44,0x01,0x7604);
    while(i<1)
    {
    //data starts at 0
    data = 0;
    //check for roll back
    if((TA0CTL & TAIFG) != 0){
    //read data from result register
    i2c_read_word(0x44,0x00, &data);
    //headers for light level and reading number
    uart_write_string(head1);
    uart_write_char('\t');
    uart_write_string(head2);
    uart_write_char('\n');
    uart_write_char('\r');
    //light level data
    uart_write_uint16(data);
    uart_write_char('\t');
    uart_write_char('\t');
    //increment number
    uart_write_uint16(j);
    uart_write_char('\n');
    uart_write_char('\r');
    uart_write_char('\n');
    uart_write_char('\r');
    //increment number
    j++;
    //clear flag
    TA0CTL &= ~TAIFG;
    }
    }
}
//Configure eUSCI in I2C master mode
void Initialize_I2C(void)
{
    //Enter reset state before the configuration starts...
    UCB1CTLW0 |= UCSWRST;

    //Divert Pins to I2C Functionality
    P4SEL1 |= (BIT1 | BIT0);
    P4SEL0 &= ~(BIT1 | BIT0);

    //Keep all the default values excep thte fields below...
    //UCMode 3:I2C Master Mode UCSSEL1:ACLK, 2,3: SMCLK
```

```
    UCB1CTLW0 |= UCMODE_3 | UCMST | UCSSEL_3;

    //Clock divider = 8 (SMCLK @ 1.047Mhz /8 = 131khz
    UCB1BRW = 8;

    //Exit the reset Mode
    UCB1CTLW0 &= ~UCSWRST;
}
void Initialize UART(void)
{
    //Divert Pins to UART functionality
    P3SEL1 &= ~(BIT4|BIT5);
    P3SEL0 |= (BIT4|BIT5);

    //Use smclk clock; leave the other settings default
    UCA1CTLW0 |= UCSSEL_2;

    //Configure clock dividers and modulators
    //UCBR = 6, UCBRF = 13, UCBRS = 0x22, UCOS16=1 (oversampling)
    UCA1BRW = 6;
    UCA1MCTLW = UCBRS5 | UCBRS1| UCBRF3| UCBRF2| UCBRF0| UCOS16;

    //exit the reset state(so transmission/reception cna begin)
    UCA1CTLW0 &= ~UCSWRST;
}
void uart_write_char(unsigned char ch)
{
    //wait for any ongoing transmission to complete
    while ((FLAGS & TXFLAG) == 0)
    {
    }

    //Write the byte to the transmit buffer
    TXBUFFER = ch;
}
void uart_write_string(char * str)
{
    int i;
    for(i=0; i < strlen(str); i++){
        uart_write_char(str[i]);
    }
}

void uart_write_uint16(unsigned int n)
{
    int temp;
    if(n>=10000)
    {
        temp = n/10000;
        uart_write_char(temp+48);
    }
    if(n>=1000)
    {
        temp = n/1000 % 10;
        uart_write_char(temp+48);
```

```
    }
    if(n>=100)
    {
        temp = n/100 % 10;
        uart_write_char(temp+48);
    }
    if(n>=10)
    {

        temp = (n/10) % 10;
        uart_write_char(temp+48);
    }
     n = n % 10;
     uart_write_char(n+48);

    //Write the byte to the transmit buffer
    //TXBUFFER = n;
}

void config_ACLK_to_32KHz_crystal()
{
    //By default, ACLK runs on LFMODCLK at 5MHz/128 = 39kHz

    //Reroute pins to LFXIN/LFXOUT functionality
    PJSEL1 &= ~BIT4;
    PJSEL0 |= BIT4;

    //Wait until the oscillator fault flags remain cleared
    CSCTL0 = CSKEY;
    do
    {
        CSCTL5 &= ~LFXTOFFG;    //local fault flag
        SFRIFG1 &= ~OFIFG;      //Global fault flag
    }
    while ((CSCTL5 & LFXTOFFG) != 0);

    CSCTL0_H = 0; //lock CS registers
    return;
}
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;
}
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)
    {
    }
//********** 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;
}
```

QUESTIONS:

1. **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 sensors data sheet.**
   4 addresses.

Table 1. Possible I^2C Addresses with Corresponding ADDR Configuration

DEVICE I^2C ADDRESS	ADDR PIN
1000100	GND
1000101	VDD
1000110	SDA
1000111	SCL

2. **According to the light sensors data sheet, what should be the value of the pull up resistors on the I2C wires? Did the Booster pack use the same values?**
   10k Ohms. The booster pack had the same values.

4	SCL	Digital input	I^2C clock. Connect with a 10-kΩ resistor to a 1.6-V to 5.5-V supply.
5	INT	Digital output	Interrupt output open-drain. Connect with a 10-kΩ resistor to a 1.6-V to 5.5-V supply.
6	SDA	Digital input/output	I^2C data. Connect with a 10-kΩ resistor to a 1.6-V to 5.5-V supply.