**11/3/2020**

Joshua Frazer

Lab 9

Inter-Integrated Circuit (I2C) Communication

**Introduction:**

Again, we are given an opportunity to work with a different interface like the recent lab, the I2C interface. With this interface we must learn to read and write data to the MCU. For part 1, we have to read data using the I2C interface. Part 2, we learn to read continuous sensor reading given off by the light sensor on the MCU.

**Part 1:**

This part required much reading and finding what data we needed to present. After looking through the guides and data sheets I was finally able to know what was needed for this part. The information that I gathered is within the first part of my main function commented out. Once all the information was gathered, I just read the manufacturer ID and the Device ID, then I transmitted what I read from I2C over to UART interface so I can validate that I was reading the values correctly.

**#include** <msp430.h>

**#include** <string.h>

**#include** <stdio.h>

**#include** <stdlib.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

**void** **Initialize\_I2C**(**void**);

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

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

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

**void** **Initialize\_UART**(**void**);

**void** **uart\_write\_char**(**unsigned** **char** ch);

**unsigned** **char** **uart\_read\_char**(**void**);

**void** **uart\_write\_intToHex**(**unsigned** **int** n);

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

**int** **main**(**void**)

{

// Manufacturer ID(0h7E) reads 0h5449

// Device ID(0h7F) reads 0h3001

// I2C address of the light sensor is 0b1000100 or 0h44

// Value of pull up resistors used for I2C lines is 10kOhms

**unsigned** **int** manID, devID;

**volatile** **unsigned** **int** i, j, counter = 0;

**char** manIDString[] = " Manufacturer ID: ",

devIDString[] = " Device ID: ";

WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer

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

Initialize\_I2C();

Initialize\_UART();

**for**(;;)

{

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

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

**for**(i = 0; i < 2; i++)

{

**for**(j = 0; j < 60000; j++) {}

}

uart\_write\_uint16(counter++);

uart\_write\_string(manIDString);

uart\_write\_intToHex(manID);

uart\_write\_char('\n');

uart\_write\_char('\r');

uart\_write\_string(devIDString);

uart\_write\_intToHex(devID);

uart\_write\_char('\n');

uart\_write\_char('\r');

}

}

// 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 except the fields below...

// (UCMode 3:I2C) (Master Mode) (UCSSEL 1:ACLK, 2,3:SMCLK)

UCB1CTLW0 |= UCMODE\_3 | UCMST | UCSSEL\_3;

// Clock divider = 8 (SMCLK @ 1.048 MHz / 8 = 131 KHz)

UCB1BRW = 8;

// Exit the reset mode

UCB1CTLW0 &= ~UCSWRST;

}

// Configure UART to the popular configuration

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

// no flow control

// Initial clock: SMCLK @ 1.048 MHz with oversampling

**void** **Initialize\_UART**(**void**){

// Divert pins to UART functionality

P3SEL1 &= ~(BIT4|BIT5);

P3SEL0 |= (BIT4|BIT5);

// Use SMCLK clock; leave other settings default

UCA1CTLW0 |= UCSSEL\_2;

// Configure the clock dividers and modulators

// UCBR=6, UCBRF=13, UCBRS=0x22, UCOS16=1 (oversampling)

UCA1BRW = 6;

UCA1MCTLW = UCBRS5|UCBRF3|UCOS16;

// Exit the reset state (so transmission/reception can begin)

UCA1CTLW0 &= ~UCSWRST;

}

**void** **uart\_write\_intToHex**(**unsigned** **int** n){

**unsigned** **int** i;

**char** hex[4];

**sprintf**(hex, "%x", n);

uart\_write\_char('0');

uart\_write\_char('h');

**for** (i = 0; i < **strlen**(hex); i++)

uart\_write\_char(hex[i]);

}

**void** **uart\_write\_uint16**(**unsigned** **int** n){

**unsigned** **int** r[6] = {0};

**int** i = 0, j;

**if** (n == 0)

uart\_write\_char('0');

**while** (n != 0)

{

r[i++] = n % 10;

n /= 10;

}

**for** (j = i-1; j >= 0; --j)

uart\_write\_char(r[j] + '0');

}

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

{

**volatile** **unsigned** **int** i;

**for** (i = 0; i<**strlen**(str); i++)

{

uart\_write\_char(str[i]);

}

}

**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;

}

// The function returns the byte; if none received, returns NULL

**unsigned** **char** **uart\_read\_char**(**void**){

**unsigned** **char** temp;

// Return NULL if no byte received

**if**( (FLAGS & RXFLAG) == 0)

**return** NULL;

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

temp = RXBUFFER;

**return** temp;

}

// 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) {}

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

UCB1TXBUF = byte1;

**while** ( (UCB1IFG & UCTXIFG0) == 0) {}

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

UCB1TXBUF = byte2;

**while** ( (UCB1IFG & UCTXIFG0) == 0) {}

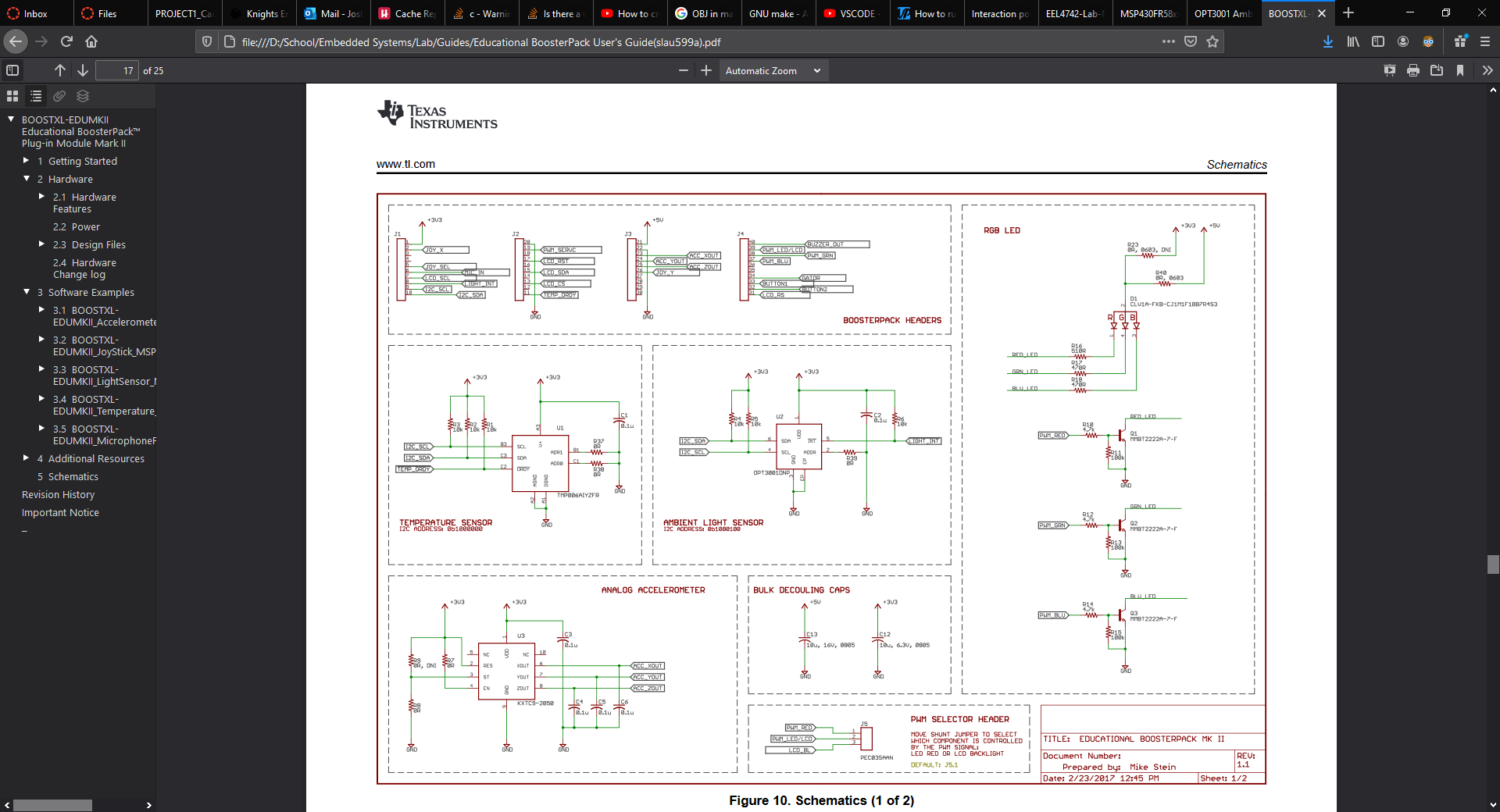
UCB1CTLW0 |= UCTXSTP;

**while** ( (UCB1CTLW0 & UCTXSTP) != 0) {}

**return** 0;

}

**Questions:**

1. What is the address of the Manufacturer ID register? What value does this register return?
   1. **The address of the Manufacturer ID register is 0h7E. The value that gets returned from this register is 0h5449.**
2. What is the address of the Device ID register? What value does this register return?
   1. **The address of the Device ID register is 0h7F. The value that gets returned from this register is 0h3001.**
3. What is the light sensor’s I2C address?
   1. **The light sensor’s I2C address is 0h44**
4. 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.
   1. **The value of the pull-up resistor on the I2C wires is 10kΩ.**   
      

**Part 2:**

Once you set up everything properly in part 1, you get a feel of how you are to read with I2C. With writing, it took me a couple tries upon succession. Once I got the configuration setup, it was similar to Part 1 with reading and printing the values to the terminal via UART.

**#include** <msp430.h>

**#include** <string.h>

**#include** <stdio.h>

**#include** <stdlib.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

**void** **Initialize\_I2C**(**void**);

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

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

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

**void** **Initialize\_UART**(**void**);

**void** **uart\_write\_char**(**unsigned** **char** ch);

**unsigned** **char** **uart\_read\_char**(**void**);

**void** **uart\_write\_intToHex**(**unsigned** **int** n);

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

**void** **uart\_write\_double**(**double** n);

**int** **main**(**void**)

{

// Configuration(0h01)

// I2C address of the light sensor is 0b1000100 or 0h44

// Value of pull up resistors used for I2C lines is 10kOhms

**unsigned** **int** configuration = 0x7604,

result;

**double** lux;

**volatile** **unsigned** **int** i, j, counter = 0;

**char** luxString[] = " Lux: ";

WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer

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

Initialize\_I2C();

Initialize\_UART();

i2c\_write\_word(0x44, 0x01, configuration);

**for**(;;)

{

i2c\_read\_word(0x44, 0x00, &result);

lux = 1.28\*result;

**for**(i = 0; i < 2; i++)

{

**for**(j = 0; j < 60000; j++) {}

}

uart\_write\_uint16(counter++);

uart\_write\_string(luxString);

uart\_write\_uint16(lux);

uart\_write\_char('\n');

uart\_write\_char('\r');

}

}

// 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 except the fields below...

// (UCMode 3:I2C) (Master Mode) (UCSSEL 1:ACLK, 2,3:SMCLK)

UCB1CTLW0 |= UCMODE\_3 | UCMST | UCSSEL\_3;

// Clock divider = 8 (SMCLK @ 1.048 MHz / 8 = 131 KHz)

UCB1BRW = 8;

// Exit the reset mode

UCB1CTLW0 &= ~UCSWRST;

}

// Configure UART to the popular configuration

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

// no flow control

// Initial clock: SMCLK @ 1.048 MHz with oversampling

**void** **Initialize\_UART**(**void**){

// Divert pins to UART functionality

P3SEL1 &= ~(BIT4|BIT5);

P3SEL0 |= (BIT4|BIT5);

// Use SMCLK clock; leave other settings default

UCA1CTLW0 |= UCSSEL\_2;

// Configure the clock dividers and modulators

// UCBR=6, UCBRF=13, UCBRS=0x22, UCOS16=1 (oversampling)

UCA1BRW = 6;

UCA1MCTLW = UCBRS5|UCBRF3|UCOS16;

// Exit the reset state (so transmission/reception can begin)

UCA1CTLW0 &= ~UCSWRST;

}

**void** **uart\_write\_intToHex**(**unsigned** **int** n){

**unsigned** **int** i;

**char** hex[4];

**sprintf**(hex, "%x", n);

uart\_write\_char('0');

uart\_write\_char('h');

**for** (i = 0; i < **strlen**(hex); i++)

uart\_write\_char(hex[i]);

}

//void uart\_write\_double(double n){

// unsigned int i;

// char stringOfDouble[10];

//

// fprintf(stringOfDouble, "%e", n);

//

// for (i = 0; i < strlen(stringOfDouble); i++)

// uart\_write\_char(stringOfDouble[i]);

//}

**void** **uart\_write\_uint16**(**unsigned** **int** n){

**unsigned** **int** r[6] = {0};

**int** i = 0, j;

**if** (n == 0)

uart\_write\_char('0');

**while** (n != 0)

{

r[i++] = n % 10;

n /= 10;

}

**for** (j = i-1; j >= 0; --j)

uart\_write\_char(r[j] + '0');

}

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

{

**volatile** **unsigned** **int** i;

**for** (i = 0; i<**strlen**(str); i++)

{

uart\_write\_char(str[i]);

}

}

**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;

}

// The function returns the byte; if none received, returns NULL

**unsigned** **char** **uart\_read\_char**(**void**){

**unsigned** **char** temp;

// Return NULL if no byte received

**if**( (FLAGS & RXFLAG) == 0)

**return** NULL;

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

temp = RXBUFFER;

**return** temp;

}

// 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) {}

//\*\*\*\*\*\*\*\*\*\*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. What is the address of the configuration register on the sensor?
   1. **The address configuration of the register on the sensor is 0h01.**
2. What configuration value (hex) did you write to the sensor? Show how this value is for-matted into bit fields.
   1. **The configuration value that I wrote to the sensor is 0h7604.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **1** | **1** | **0** | **1** | **1** | **0** |
| **RN3** | **RN2** | **RN1** | **RN0** | **CT** | **M1** | **M0** | **OVF** |
| **R/W** | **R/W** | **R/W** | **R/W** | **R/W** | **R/W** | **R/W** | **R** |
| **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** |
| **CRF** | **FH** | **FL** | **L** | **POL** | **ME** | **FC1** | **FC0** |
| **R** | **R** | **R** | **R/W** | **R/W** | **R/W** | **R/W** | **R/W** |

1. Does the data make sense based on what you expected?
   1. **Yes, the data makes sense based on what I expected.**

**Student Q&A:**

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 sensor’s data sheet.
   1. **There are four addresses possible. They are GRD, VDD, SDA, and SCL. These are configured through the ADDR pin.**
2. 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?
   1. **According to the light sensors data sheet, the value of the pull-up resistors on the I2C wires should be 10kΩ. The BoosterPack uses the same values.**

**Conclusion:**

This lab was my favorite. I love that I learned how to program the configuration of the light sensor and got to work it hands on. I wish there was more labs like this to incorporate everything together. All in all, I would say part 1 was the most significant part of the lab since without knowing how to properly configure I2C, you wouldn’t be able to work with the light as easily.