// Lab 9: Inter-Integrated Circuit

// Lab Purpose 🡪 Familiarizing the user with the I2C interface!

// Procedure:

// **9.1: I2C Transmission: Learning about the I2C communication protocol.**

// Using a bus topology!

// Two wires SDA and SCL!

// Connecting multiple devices can be connected into the same two wires 🡪 Allowing a lot of flexibility!

// Learning about the read and write procedures between the Master and the device 🡪 Using acknowledgment bits!

// Learning how to configure our eUSCI module to I2C communication protocol 🡪 Providing lab manual with a code that can initialize the I2C correctly!

// The Task for Students 🡪 Reading the Manufacturer ID and Device ID registers continuously using I2C transmissions successfully!

// Adding UART functionality to our program 🡪 The values of Manufacturer ID and Device ID registers can be read through a terminal!

// UART functionality 🡪 Our program can be used to read the values of Manufacturer ID and Device ID!

// Needing to add a delay of one second between each read and print a counter in the terminal! 🡪 Showing that the transmission was continuous!

// Addressing of both Manufacturer ID and Device ID 🡪 0x7E and 0x7F!

// Returned Values 🡪 They are 0x5459 and 0x3001!

// Manufacturer ID register 🡪 TI in ASCII Code!

// Testing the I2C transmission and ensuring that it works!

// I2C 🡪 Addressing the light sensor found in the BoosterPack!

// Using Code 🡪 For initialization of UART and I2C!

// Reading a word from I2C and writing word to I2C!

// Writing a char through UART!

// Incorporating our own 16-bit unsigned integer write to UART function!

// Initialization of both UART and I2C 🡪 Into the main function!

// Two integer data type variables 🡪 Usage for storing the returned values of Manufacturer ID and Device ID registers!

// Using a timer loop with a delay of one second 🡪 Using the ACLK configured to the 32 KHz crystal!

// Having a counter for displaying in our terminal through UART!

// Inside our loop 🡪 Connecting to the light sensor of our BoosterPack using 0x44!

// Reading the manufacturer ID and device ID using 0x7E and 0x7F!

// Storing the returned values in our integer data type variables!

// Transmitting the values through UART 🡪 Using the integer data type values with the counter!

**#include** <msp430fr6989.h>

**#include** <stdio.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

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

// Reconfigures ACLK to be rerouted to the 32 KHz crystal on the LaunchPad

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

// The default mode of the ACLK is a built-in oscillator at a frequency of 39KHz normally.

// Rerouted the pins to LFXIN/LFXOUT functionality so that the ACLK can be routed to the 32KHz crystal.

// This information can be found using the LaunchPad user's guide (page 29) and the chip's data sheet (page 123).

PJSEL1 &= ~BIT4;

PJSEL0 |= BIT4;

// We need to for the crystal to settle, once it has started.

// Therefore, we will wait until the local and global oscillator fault flags are cleared and remain cleared.

CSCTL0 = CSKEY; // Unlock CS registers, to divert the pins for the crystal functionality.

//Clears the flag and will do so until they remain cleared.

**do** {

CSCTL5 &= ~LFXTOFFG; // Local oscillator fault flag

SFRIFG1 &= ~OFIFG; // Global oscillator fault flag

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

CSCTL0\_H = 0; // Lock CS registers, returns the pins.

**return**;

}

// Popular Configuration of UART

// 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 to enable transmission/reception of data between PC and MCU

P3SEL1 &= ~(BIT4|BIT5);

P3SEL0 |= (BIT4|BIT5);

// Sets the clock source to SMCLK

UCA1CTLW0 |= UCSSEL\_2;

// Configures the dividers and modulators of the clock with the popular configuration of UART

// Configurations can be found in the Family User's Guide Page 783

UCA1BRW = 6;

UCA1MCTLW = UCBRS5|UCBRS1|UCBRF3|UCBRF2|UCBRF0|UCOS16;

// Enables transmission/reception to start

UCA1CTLW0 &= ~UCSWRST;

}

// Configure eUSCI in I2C master mode

**void** **Initialize\_I2C**(**void**) {

// We enter the reset state, so we can start configuring

UCB1CTLW0 |= UCSWRST;

// We divert the pins P4.1 and P4.0 to I2C functionality

P4SEL1 |= (BIT1|BIT0);

P4SEL0 &= ~(BIT1|BIT0);

// We configure our controller to be in I2C, Master Mode and using the SMCLK

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

// We employ the clock divider to set the clock signal to 131KHz (SMCLK @ 1.048 MHz / 8 = 131 KHz)

UCB1BRW = 8;

// We exit the reset mode to begin I2C communication

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) {

**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 11

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;

}

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

// Ongoing transmission to be over

**while** ( (FLAGS & TXFLAG)==0 ) {}

// Write the input byte into the buffer to display in the terminal

TXBUFFER = ch;

}

// Function that takes a 16-bit unsigned integer and transmit it through UART communication

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

**volatile** **unsigned** **int** digit = 0;

**int** k = 0,l = 0;

**int** array[5];

// Extracts digit by digit of the input number and stores it in an array

**do**{

digit = n % 10; // Extracts the digit

array[k] = digit; // Stores it

n = n/10; // Truncates the input number

k++;

l++;

}**while**(n != 0);

// Extracts from the array backwardly to transmit the each digit through UART

**for**(k = l - 1 ; k >= 0;k--)

uart\_write\_char(array[k] + '0');

// New line

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

// Carry return

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

}

/\*\*

\* main.c

\*/

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

{

WDTCTL = WDTPW | WDTHOLD; // Stops the watchdog timer. We do this so the MCU doesn't reset itself periodically.

PM5CTL0 &= ~LOCKLPM5; // We enable the general purpose I/O pins.

// Initializes the UART communication using SMCLK @ 1.048 MHz with oversampling at a baud rate of 9600

Initialize\_UART();

// Initializes the I2C communication using a clock signal of 131 KHz (SMCLK @ 1.048 MHz / 8 = 131KHz)

Initialize\_I2C();

// Configures the ACLK to the 32KHz crystal

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

// Set the timer to ACLK, Up Mode, Clear TAR

TA0CTL= TASSEL\_1|MC\_1|TACLR;

// Sets the upperbound of TAR to 32,767. This generates a delay of 1 second.

TA0CCR0=(32768-1);

// Variables that will be used for the following:

// data - stores value read from Manufacturer ID register

// data2 - stores value read from Device ID register

// lux - final value converted from the measurement taken by the light sensor

**unsigned** **int** data, data2;

**volatile** **unsigned** **int** j = 0;

**for**(;;){

// Function that stores the Manufacturer ID register's value into an integer data type variable.

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

// Function that stores the Device ID register's value into an integer data type variable.

i2c\_read\_word(0x44, 0x7F, &data2);

// We write to the terminal in the PC through UART, the Manufacturer ID register's value

uart\_write\_uint16(data);

// We write to the terminal in the PC through UART, the Device ID register's value

uart\_write\_uint16(data2);

// We write to the terminal in the PC through UART, the counter to ensure continuous transmission

uart\_write\_uint16(j);

// Increment counter

j++;

// Timer that generates a delay loop of 1 second

**while**((TA0CTL & TAIFG) == 0){}

// Clear the timer's interrupt flag for constant generation of 1 second delay

TA0CTL &= ~TAIFG;

}

}

// **9.2: Reading the Measurements from the Light Sensor.**

// Learning how to read the measurements from the light sensor of our BoosterPack!

// Learning that te light sensor returns its values in a 16-bit result that was composed of a 4-bit exponent (leftmost bits) and a 12-bit result (called a mantisa)!

// Depending on the four-bit exponent 🡪 Needing to multiply our result with a constant value shown as LSB!

// Depending on the 4-bit exponent!

// Multiplying our result with a constant value known as LSB!

// lux = LSB\_Size \* R[11:0]

// LSB\_Size = 0.01 \* 2^E[3:0]

// R: Mantissa and E: Exponent

// Configuration 🡪 Stored inside the configuration register!

// Learning the bit fields of the configuration register!

// Our binary configuration of te light sensor 🡪 0b0111011000000100 OR 0x7604!

// Obtaining the value needed for the configuration 🡪 Beginning to code our program!

// Task? Using the light sensor and measuring lux readings correctly!

// The address of configuration register 🡪 0x01!

// The address of result register is 0x00!

// Writing the configuration to the configuration register!

// Setting an integer data type variable with the hex configuration 0x7604!

// Using i2c\_write\_word(0x44, 0x01, data) 🡪 For setting the configuration into the configuration register!

// Reading from the result register 🡪 Using the following line of code i2c\_read\_word!

// Clearing the leftmost 4-bits of the result since it is the 4-bit exponent field using 0x0FFF and applying the formula (discussed earlier)!

// Transmitting our result to the terminal through UART with a counter to ensure continuous transmission and a delay of one second generated by the timer!

**#include** <msp430fr6989.h>

**#include** <stdio.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

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

// Reconfigures ACLK to be rerouted to the 32 KHz crystal on the LaunchPad

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

// The default mode of the ACLK is a built-in oscillator at a frequency of 39KHz normally.

// Rerouted the pins to LFXIN/LFXOUT functionality so that the ACLK can be routed to the 32KHz crystal.

// This information can be found using the LaunchPad user's guide (page 29) and the chip's data sheet (page 123).

PJSEL1 &= ~BIT4;

PJSEL0 |= BIT4;

// We need to for the crystal to settle, once it has started.

// Therefore, we will wait until the local and global oscillator fault flags are cleared and remain cleared.

CSCTL0 = CSKEY; // Unlock CS registers, to divert the pins for the crystal functionality.

//Clears the flag and will do so until they remain cleared.

**do** {

CSCTL5 &= ~LFXTOFFG; // Local oscillator fault flag

SFRIFG1 &= ~OFIFG; // Global oscillator fault flag

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

CSCTL0\_H = 0; // Lock CS registers, returns the pins.

**return**;

}

// Popular Configuration of UART

// 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 to enable transmission/reception of data between PC and MCU

P3SEL1 &= ~(BIT4|BIT5);

P3SEL0 |= (BIT4|BIT5);

// Sets the clock source to SMCLK

UCA1CTLW0 |= UCSSEL\_2;

// Configures the dividers and modulators of the clock with the popular configuration of UART

// Configurations can be found in the Family User's Guide Page 783

UCA1BRW = 6;

UCA1MCTLW = UCBRS5|UCBRS1|UCBRF3|UCBRF2|UCBRF0|UCOS16;

// Enables transmission/reception to start

UCA1CTLW0 &= ~UCSWRST;

}

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

// Ongoing transmission to be over

**while** ( (FLAGS & TXFLAG)==0 ) {}

// Write the input byte into the buffer to display in the terminal

TXBUFFER = ch;

}

// Function that takes a 16-bit unsigned integer and transmit it through UART communication

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

**volatile** **unsigned** **int** digit = 0;

**int** k = 0,l = 0;

**int** array[5];

// Extracts digit by digit of the input number and stores it in an array

**do**{

digit = n % 10; // Extracts the digit

array[k] = digit; // Stores it

n = n/10; // Truncates the input number

k++;

l++;

}**while**(n != 0);

// Extracts from the array backwardly to transmit the each digit through UART

**for**(k = l - 1 ; k >= 0;k--)

uart\_write\_char(array[k] + '0');

// New line

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

// Carry return

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

}

// Configure eUSCI in I2C master mode

**void** **Initialize\_I2C**(**void**) {

// We enter the reset state, so we can start configuring

UCB1CTLW0 |= UCSWRST;

// We divert the pins P4.1 and P4.0 to I2C functionality

P4SEL1 |= (BIT1|BIT0);

P4SEL0 &= ~(BIT1|BIT0);

// We configure our controller to be in I2C, Master Mode and using the SMCLK

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

// We employ the clock divider to set the clock signal to 131KHz (SMCLK @ 1.048 MHz / 8 = 131 KHz)

UCB1BRW = 8;

// We exit the reset mode to begin I2C communication

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) {

**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 11

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;

}

/\*\*

\* main.c

\*/

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

{

WDTCTL = WDTPW | WDTHOLD; // Stops the watchdog timer. We do this so the MCU doesn't reset itself periodically.

PM5CTL0 &= ~LOCKLPM5; // We enable the general purpose I/O pins.

// Initializes the UART communication using SMCLK @ 1.048 MHz with oversampling at a baud rate of 9600

Initialize\_UART();

// Initializes the I2C communication using a clock signal of 131 KHz (SMCLK @ 1.048 MHz / 8 = 131KHz)

Initialize\_I2C();

// Variables that will be used for the following:

// data - light sensor configuration

// data2 - stores result register value from the light sensor

// lux - final value converted from the measurement taken by the light sensor

**unsigned** **int** data, data2, lux;

**volatile** **unsigned** **int** j = 0;

// Configures the ACLK to the 32KHz crystal

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

// Set the timer to ACLK, Up Mode, Clear TAR

TA0CTL |= TASSEL\_1 | MC\_1| TACLR;

// Sets the upperbound of TAR to 32,767. This generates a delay of 1 second.

TA0CCR0 = (32768 - 1);

// Configuration data that was tasked to write into the light sensor's configuration register

data = 0x7604;

// Function that writes the tasked configuration into the light sensor's configuration register

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

**for**(;;){

// Function that stores the result register's value into an integer data type variable.

i2c\_read\_word(0x44, 0x00, &data2);

// We eliminate the 4 leftmost bits of the result since they are the 4-bit exponent field which are required to determine the LSB\_Size but not part of the mantissa

data2 &= 0x0FFF;

// The conversion is : lux = mantissa \* 2^(the value of the 4-bit exponent field) \* 0.01

lux = (**unsigned** **int**) data2 \* 1.28;

// We write to the terminal in the PC through UART, the calculated values of lux

uart\_write\_uint16(lux);

// We write to the terminal in the PC through UART, the counter to ensure continuous transmission

uart\_write\_uint16(j);

// Increment counter

j++;

// Timer that generates a delay loop of 1 second

**while**((TA0CTL & TAIFG) == 0){}

// Clear the timer's interrupt flag for constant generation of 1 second delay

TA0CTL &= ~TAIFG;

}

}