CPE 403 Tiva C Midterm Project Fall 2018

TITLE: TSL2561 Lux Sensor to ThingSpeak server.

GOAL:

* To collect data from the TSL2561 at intervals between 15 and 60 seconds.
* Upload the data using the supplied ESP8266 module into a IoT cloud.
* Use I2C interface for the TSL sensor and UART interface for the ESP8266 WiFi module.

DELIVERABLES:

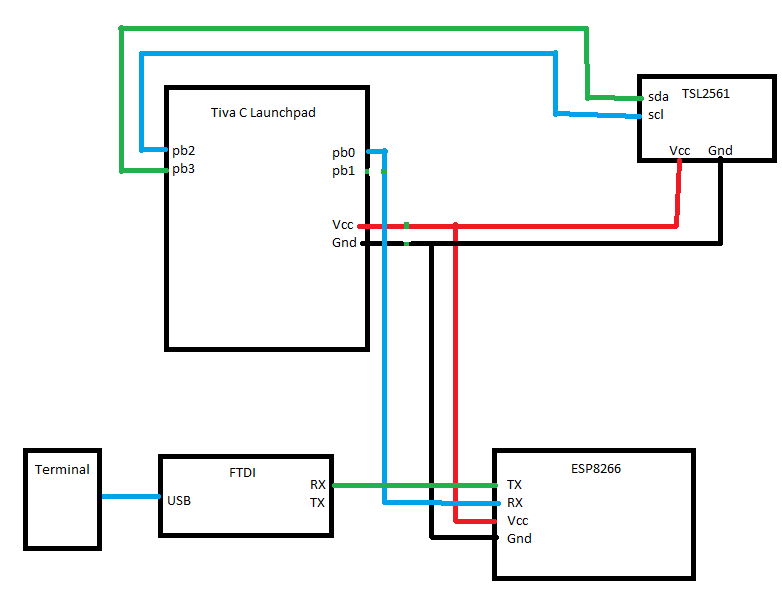
Deliverables are a graph of all the data that was collected in a 24 hour period. Since ThingSpeak doesn’t display the graph properly for a large amount of data, I downloaded all the data into a excel sheet and displayed the graph there. Also all the code with highlighted changed will be included along with schematics and pictures.

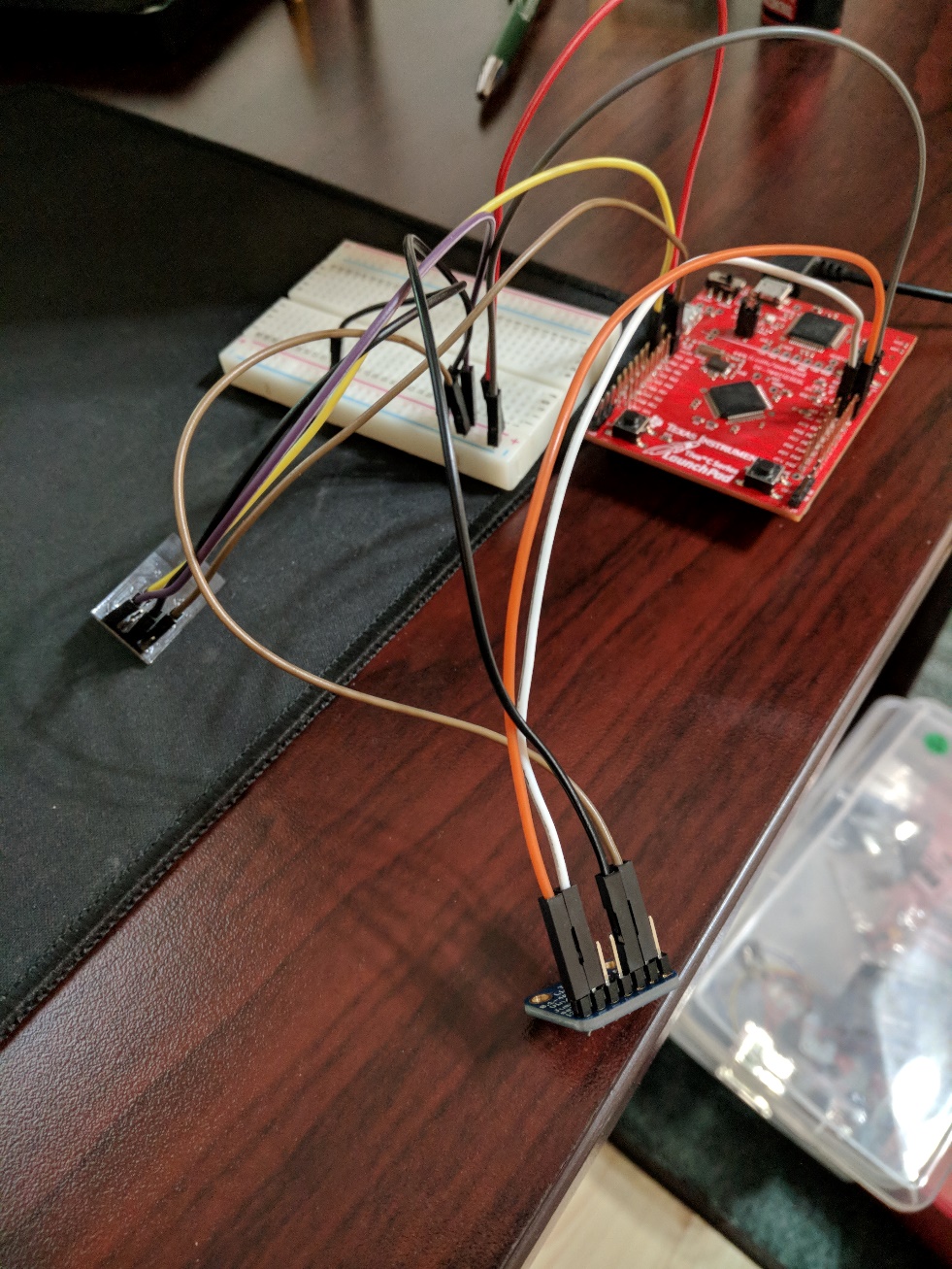
Video Link: https://www.youtube.com/watch?v=GQKfKfVgKIk

COMPONENTS:

This project uses TSL2561 lux sensor, esp8266 WiFi module, the Tiva C 1350 micro controller, FTDI for debugging the program. I had an esp8266 from CpE 301 but I had to reprogram it with software again which took some time to do. I also used jumper cables to connect all the components.

SCHEMATICS:





IMPLEMENTATION:

For this project, the lux sensor would grab a value and then send that data to the Tiva C using I2C protocol which the board would then do calculations on the value sent in from the lux sensor. The board would then send the data to the esp8266 module using uart which then would send the data to the thingspeak server.

CODE:

**Main C Code**

**#include** <stdarg.h>

**#include** <stdbool.h>

**#include** <stdint.h>

**#include** "inc/tm4c123gh6pm.h"

**#include** "inc/hw\_i2c.h"

**#include** "inc/hw\_memmap.h"

**#include** "inc/hw\_types.h"

**#include** "inc/hw\_gpio.h"

**#include** "driverlib/i2c.h"

**#include** "driverlib/sysctl.h"

**#include** "driverlib/gpio.h"

**#include** "driverlib/pin\_map.h"

**#include** "driverlib/uart.h"

**#include** "uartstdio.h"

**#include** "driverlib/interrupt.h"

**#include** "driverlib/hibernate.h"

**#include** "TSL2591\_def.h"

**#include** "ustdlib.h"

**void** **ConfigureUART**(**void**)

//Configures the UART to run at 19200 baud rate

{

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART1); //enables UART module 1

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOB); //enables GPIO port b

**GPIOPinConfigure**(GPIO\_PB1\_U1TX); //configures PB1 as TX pin

**GPIOPinConfigure**(GPIO\_PB0\_U1RX); //configures PB0 as RX pin

**GPIOPinTypeUART**(GPIO\_PORTB\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1); //sets the UART pin type

**UARTClockSourceSet**(UART1\_BASE, UART\_CLOCK\_PIOSC); //sets the clock source

UARTStdioConfig(1, 115200, 16000000); //enables UARTstdio baud rate, clock, and which UART to use

}

**void** **I2C0\_Init** ()

//Configure/initialize the I2C0

{

**SysCtlPeripheralEnable** (SYSCTL\_PERIPH\_I2C0); //enables I2C0

**SysCtlPeripheralEnable** (SYSCTL\_PERIPH\_GPIOB); //enable PORTB as peripheral

**GPIOPinTypeI2C** (GPIO\_PORTB\_BASE, GPIO\_PIN\_3); //set I2C PB3 as SDA

**GPIOPinConfigure** (GPIO\_PB3\_I2C0SDA);

**GPIOPinTypeI2CSCL** (GPIO\_PORTB\_BASE, GPIO\_PIN\_2); //set I2C PB2 as SCLK

**GPIOPinConfigure** (GPIO\_PB2\_I2C0SCL);

**I2CMasterInitExpClk** (I2C0\_BASE, **SysCtlClockGet**(), false); //Set the clock of the I2C to ensure proper connection

**while** (**I2CMasterBusy** (I2C0\_BASE)); //wait while the master SDA is busy

}

**void** **I2C0\_Write** (uint8\_t addr, uint8\_t N, ...)

//Writes data from master to slave

//Takes the address of the device, the number of arguments, and a variable amount of register addresses to write to

{

**I2CMasterSlaveAddrSet** (I2C0\_BASE, addr, false); //Find the device based on the address given

**while** (**I2CMasterBusy** (I2C0\_BASE));

va\_list vargs; //variable list to hold the register addresses passed

va\_start (vargs, N); //initialize the variable list with the number of arguments

**I2CMasterDataPut** (I2C0\_BASE, va\_arg(vargs, uint8\_t)); //put the first argument in the list in to the I2C bus

**while** (**I2CMasterBusy** (I2C0\_BASE));

**if** (N == 1) //if only 1 argument is passed, send that register command then stop

{

**I2CMasterControl** (I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_SEND);

**while** (**I2CMasterBusy** (I2C0\_BASE));

va\_end (vargs);

}

**else**

//if more than 1, loop through all the commands until they are all sent

{

**I2CMasterControl** (I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_START);

**while** (**I2CMasterBusy** (I2C0\_BASE));

uint8\_t i;

**for** (i = 1; i < N - 1; i++)

{

**I2CMasterDataPut** (I2C0\_BASE, va\_arg(vargs, uint8\_t)); //send the next register address to the bus

**while** (**I2CMasterBusy** (I2C0\_BASE));

**I2CMasterControl** (I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_CONT); //burst send, keeps receiving until the stop signal is received

**while** (**I2CMasterBusy** (I2C0\_BASE));

}

**I2CMasterDataPut** (I2C0\_BASE, va\_arg(vargs, uint8\_t)); //puts the last argument on the SDA bus

**while** (**I2CMasterBusy** (I2C0\_BASE));

**I2CMasterControl** (I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_FINISH); //send the finish signal to stop transmission

**while** (**I2CMasterBusy** (I2C0\_BASE));

va\_end (vargs);

}

}

uint32\_t **I2C0\_Read** (uint8\_t addr, uint8\_t reg)

//Read data from slave to master

//Takes in the address of the device and the register to read from

{

**I2CMasterSlaveAddrSet** (I2C0\_BASE, addr, false); //find the device based on the address given

**while** (**I2CMasterBusy** (I2C0\_BASE));

**I2CMasterDataPut** (I2C0\_BASE, reg); //send the register to be read on to the I2C bus

**while** (**I2CMasterBusy** (I2C0\_BASE));

**I2CMasterControl** (I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_SEND); //send the send signal to send the register value

**while** (**I2CMasterBusy** (I2C0\_BASE));

**I2CMasterSlaveAddrSet** (I2C0\_BASE, addr, true); //set the master to read from the device

**while** (**I2CMasterBusy** (I2C0\_BASE));

**I2CMasterControl** (I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_RECEIVE); //send the receive signal to the device

**while** (**I2CMasterBusy** (I2C0\_BASE));

**return** **I2CMasterDataGet** (I2C0\_BASE); //return the data read from the bus

}

**void** **TSL2591\_init** ()

//Initializes the TSL2591 to have a medium gain,

{

uint32\_t x;

x = I2C0\_Read (TSL2591\_ADDR, (TSL2591\_COMMAND\_BIT | TSL2591\_ID)); //read the device ID

**if** (x != 0x50)

**while** (1){}; //loop here if the dev ID is not correct

I2C0\_Write (TSL2591\_ADDR, 2, (TSL2591\_COMMAND\_BIT | TSL2591\_CONFIG), 0x10); //configures the TSL2591 to have medium gain adn integration time of 100ms

I2C0\_Write (TSL2591\_ADDR, 2, (TSL2591\_COMMAND\_BIT | TSL2591\_ENABLE), (TSL2591\_ENABLE\_POWERON | TSL2591\_ENABLE\_AEN | TSL2591\_ENABLE\_AIEN | TSL2591\_ENABLE\_NPIEN)); //enables proper interrupts and power to work with TSL2591

}

uint32\_t **GetLuminosity** ()

//This function will read the channels of the TSL and returns the calculated value to the caller

{

**float** atime = 100.0f, again = 25.0f; //the variables to be used to calculate proper lux value

uint16\_t ch0, ch1; //variable to hold the channels of the TSL2591

uint32\_t cp1, lux1, lux2, lux;

uint32\_t x = 1;

x = I2C0\_Read (TSL2591\_ADDR, (TSL2591\_COMMAND\_BIT | TSL2591\_C0DATAH));

x <<= 16;

x |= I2C0\_Read (TSL2591\_ADDR, (TSL2591\_COMMAND\_BIT | TSL2591\_C0DATAL));

ch1 = x>>16;

ch0 = x & 0xFFFF;

cp1 = (uint32\_t) (atime \* again) / TSL2591\_LUX\_DF;

lux1 = (uint32\_t) ((**float**) ch0 - (TSL2591\_LUX\_COEFB \* (**float**) ch1)) / cp1;

lux2 = (uint32\_t) ((TSL2591\_LUX\_COEFC \* (**float**) ch0) - (TSL2591\_LUX\_COEFD \* (**float**) ch1)) / cp1;

lux = (lux1 > lux2) ? lux1: lux2;

**return** lux;

}

**void** **ESP\_Init**()

// This function initializes ESP8266 using UART to set up Wi-Fi connection.

{

// set ESP8266 Baud Rate

UARTprintf("AT+CIOBAUD=%d\n\r", 115200);

**SysCtlDelay** (50000000);

// set Wi-Fi mode to Infrastructure

UARTprintf("AT+CWMODE=1\n\r");

**SysCtlDelay** (50000000);

// configure SSID and password for Wi-Fi connection

UARTprintf("AT+CWJAP=\"SSID\",\"Password\"\n\r");

**SysCtlDelay** (200000000);

}

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

{

**char** HTTP\_POST[256]; //string buffer to hold the HTTP command

**SysCtlClockSet**(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_XTAL\_16MHZ|SYSCTL\_OSC\_MAIN); //set the main clock to runat 40MHz

uint32\_t lux = 0;

uint32\_t i = 0;

uint32\_t luxAvg = 0;

ConfigureUART (); //configure the UART of Tiva C

I2C0\_Init (); //initialize the I2C0 of Tiva C

TSL2591\_init (); //initialize the TSL2591

UARTprintf ("AT+RST\r\n"); //reset the esp8266 before pushing data

**SysCtlDelay** (100000000);

ESP\_Init();

**SysCtlDelay** (100000000);

**while**(1)

{

**for** (i = 0; i < 20; i++)

//finds the average of the lux channel to send through uart

{

lux = GetLuminosity ();

luxAvg += lux;

}

luxAvg = luxAvg/20;

**SysCtlDelay** (100000000);

UARTprintf ("AT+CIPMUX=0\r\n\r\n"); //enable multiple send ability

**SysCtlDelay** (50000000);

UARTprintf("AT+CIPSTART=\"TCP\",\"184.106.153.149\",80\n\r"); // setup connection

**SysCtlDelay** (50000000);

//The following lines of code puts the TEXT with the data from the lux in to a string to be sent through UART

//usprintf (HTTP\_POST, "GET /update?key=W86U4X7NUJNKEMGG&field1=\n\r", lux);

**SysCtlDelay** (10000000);

UARTprintf("AT+CIPSEND=%i\r\n", 46);//strlen(HTTP\_POST)); //command the ESP8266 to allow sending of information

**SysCtlDelay** (10000000);

UARTprintf("GET /update?key=W86U4X7NUJNKEMGG&field1=%i\n\r", lux); //send the string of the HTTP GET to the ESP8266

//HibernateRequest (); //Hibernate

**SysCtlDelay** (50000000);

**SysCtlDelay** (50000000);

**SysCtlDelay** (50000000);

}

}

**TSL2591.h File**

uint32\_t ui32SysClock;

**const** uint8\_t TSL2591address = 0x39;

**const** bool DebuggingMode = true;

**#define** TSL2591\_VISIBLE (2) // channel 0 - channel 1

**#define** TSL2591\_INFRARED (1) // channel 1

**#define** TSL2591\_FULLSPECTRUM (0) // channel 0

**#define** TSL2591\_ADDR (0x39)

**#define** TSL2591\_READBIT (0x01)

**#define** TSL2591\_COMMAND\_BIT (0x80) // 1010 0000: bits 7 and 5 for 'command normal'

**#define** TSL2591\_CLEAR\_INT (0x40)

**#define** TSL2591\_TEST\_INT (0xE4)

**#define** TSL2591\_WORD\_BIT (0x20) // 1 = read/write word (rather than byte)

**#define** TSL2591\_BLOCK\_BIT (0x10) // 1 = using block read/write

**#define** TSL2591\_ENABLE\_POWEROFF (0x00)

**#define** TSL2591\_ENABLE\_POWERON (0x03)

**#define** TSL2591\_ENABLE\_AEN (0x02) // ALS Enable. This field activates ALS function. Writing a one activates the ALS. Writing a zero disables the ALS.

**#define** TSL2591\_ENABLE\_AIEN (0x10) // ALS Interrupt Enable. When asserted permits ALS interrupts to be generated, subject to the persist filter.

**#define** TSL2591\_ENABLE\_NPIEN (0x80) // No Persist Interrupt Enable. When asserted NP Threshold conditions will generate an interrupt, bypassing the persist filter

**#define** TSL2591\_LUX\_DF (408.0F)

**#define** TSL2591\_LUX\_COEFB (1.64F) // CH0 coefficient

**#define** TSL2591\_LUX\_COEFC (0.59F) // CH1 coefficient A

**#define** TSL2591\_LUX\_COEFD (0.86F) // CH2 coefficient B

**#define** TSL2591\_ENABLE 0x00

**#define** TSL2591\_CONFIG 0x01

**#define** TSL2591\_ID 0x0A

**#define** TSL2591\_REGISTER\_DEVICE\_STATUS 0x13

**#define** TSL2591\_C0DATAL 0x0C

**#define** TSL2591\_C0DATAH 0x0D

**#define** TSL2591\_C1DATAL 0x0E

**#define** TSL2591\_C1DATAH 0x0F

**#define** TSL2591\_INTEGRATIONTIME\_100MS 0x01

**#define** TSL2591\_GAIN\_MED 0x10