TITLE: TIVA C MIDTERM

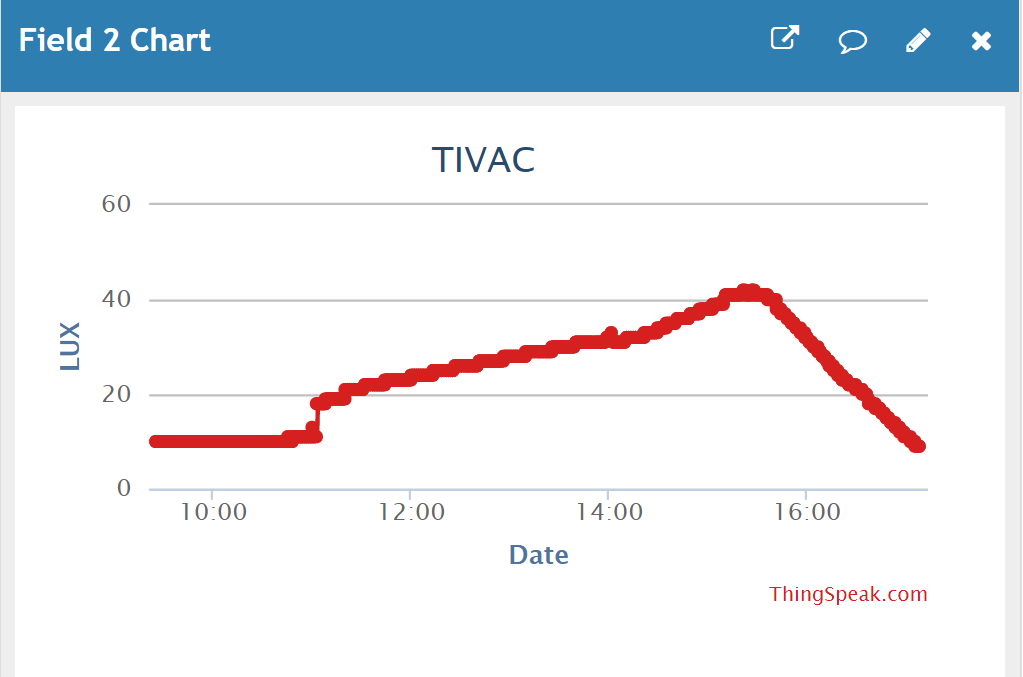
October 30, 2018

GOAL:

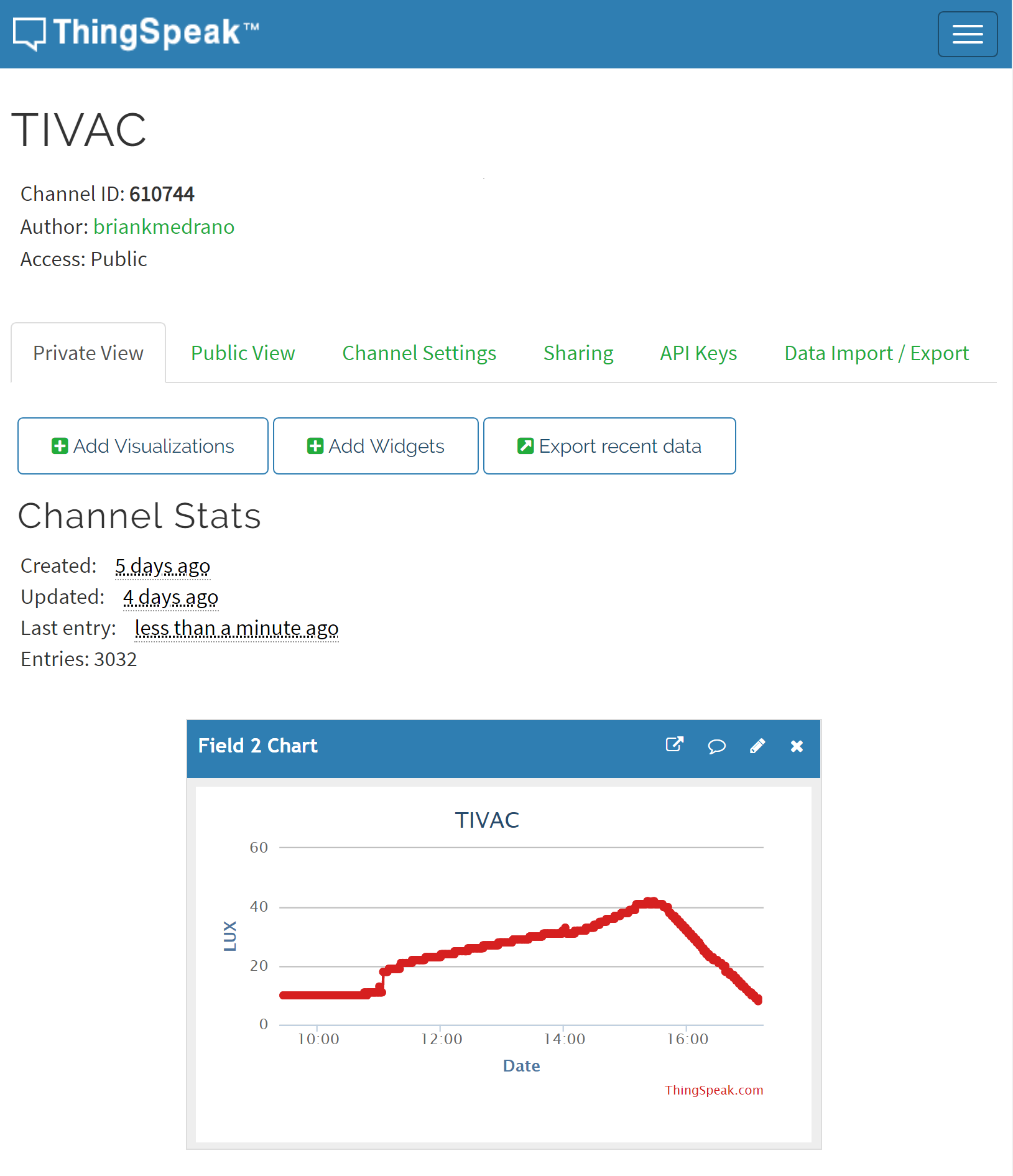
* Collect data at intervals >= 15 seconds and <= 60 seconds for an extended period of time >= 1 hour and <= 24 hours using the TSL2561 Lux Sensor.
* Using the ESP8266 module upload the lux sensor data onto a IoT Cloud (thingspeak server).
* The TSL sensor must be interfaced using I2C and UART must be used for the ESP8266 WiFi module.

DELIVERABLES:

* Code submitted with comments and project support materials. (Completed)
* Documentation with comments, variations for tasks, schematics. (Completed)
* Video Demo in txt file, I- image ScreenShot in document or links. (Completed)



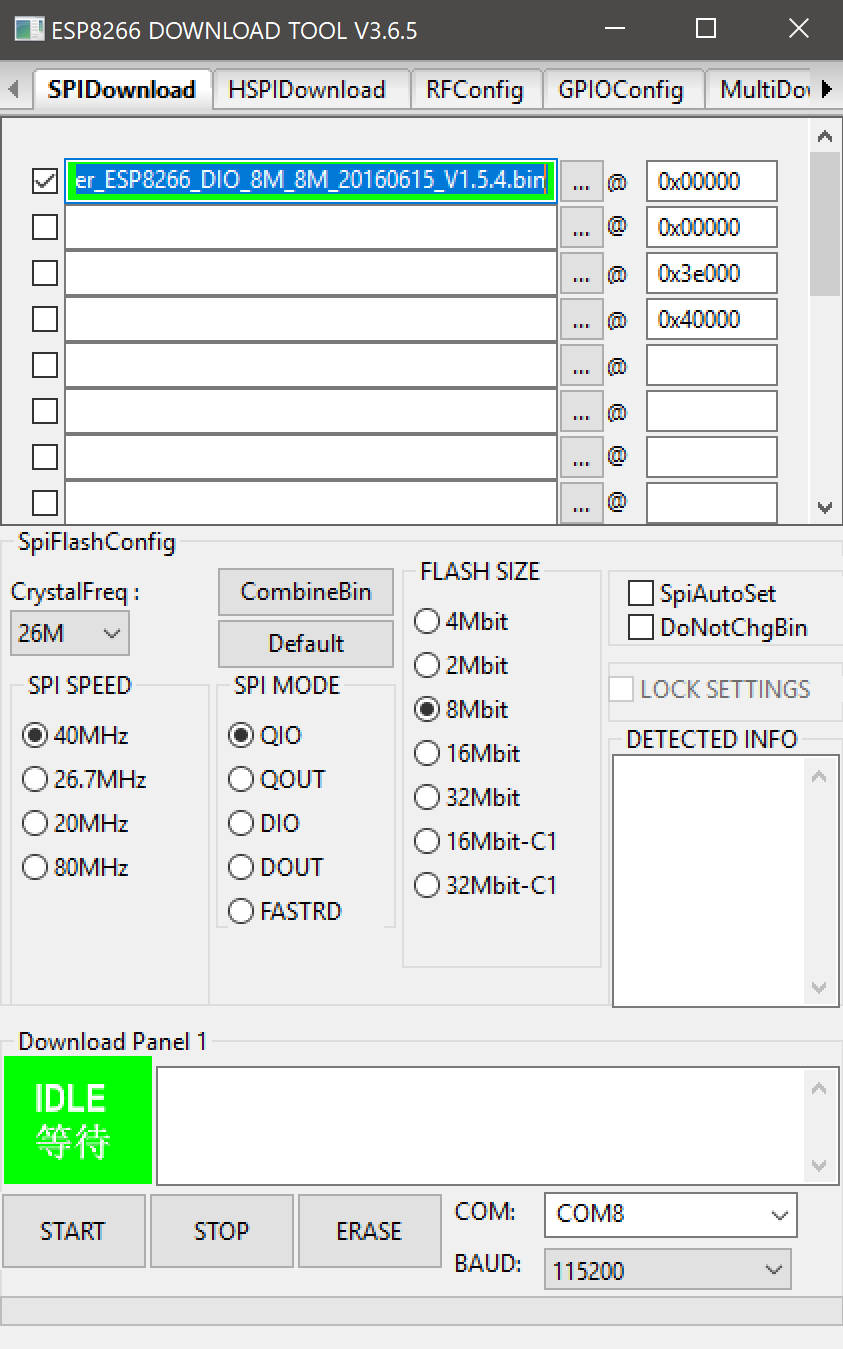
Above are the results of the TIVA C Midterm project which is responsible for sending LUX sensor values from the sensor TSL2561 to an IoT Cloud Server (Thingspeak). This graph has been receiving values since 9AM this morning and has been progressively receiving values in less than a minute interval. The Lux sensor was placed besides my window. We can see that the window began to get brighter around 11AM and continued to get brighter until right before 4PM. After 4PM we can see that the lux values are progressively decreasing.



Video Link: <https://www.youtube.com/watch?v=FS4U2-8DjXA>

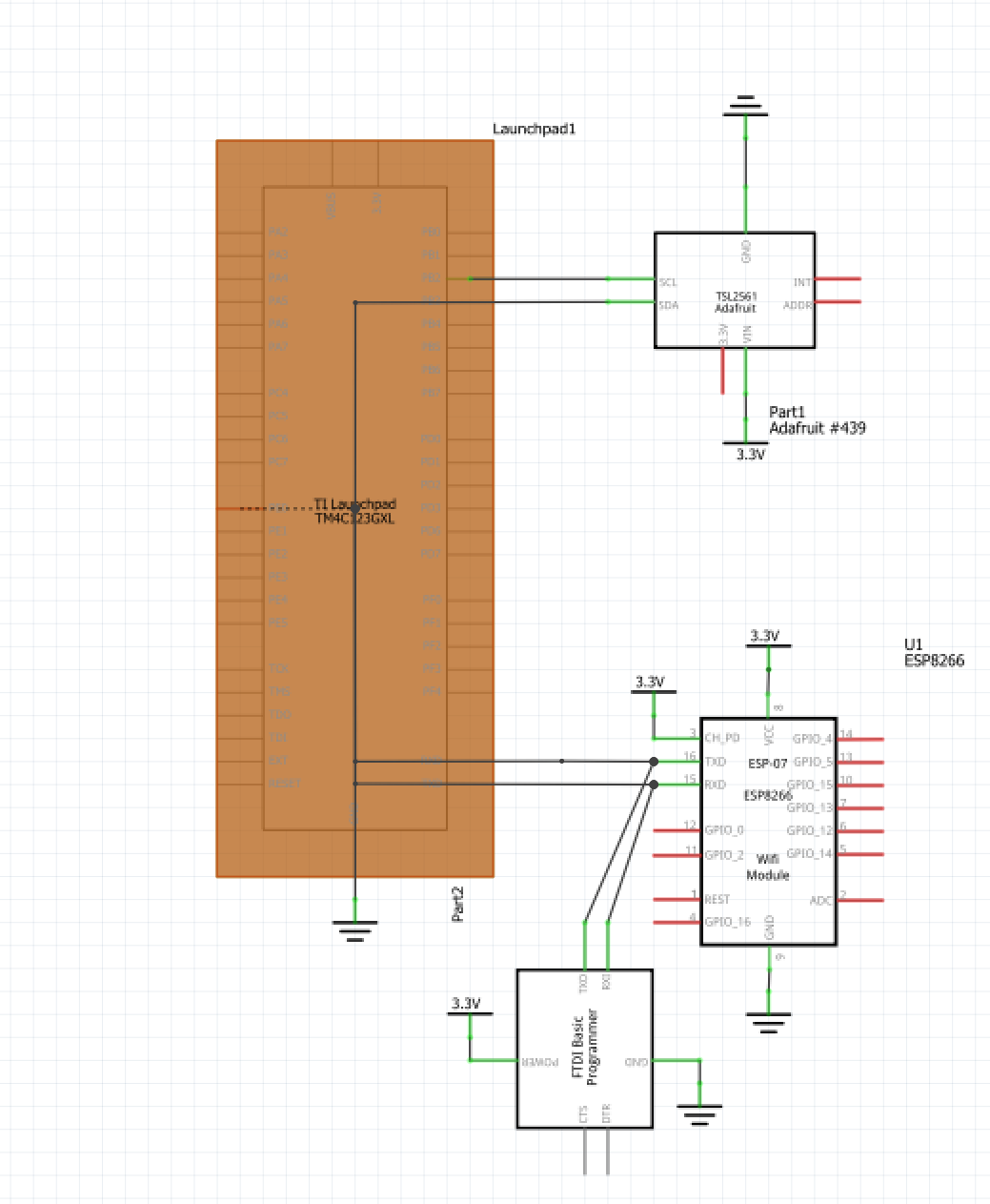
COMPONENTS:

* TSL2561 Lux Sensor
  + The purpose of this component is to read LUX values indicated by the environment the sensor is placed in. This sensor uses I2C interface depending on the connection of the ADDR pin depends on the I2C address. If the ADDR pin is disconnected then the address is 0x39. If it is connected to VDD (3.3V) the address is 0x49 or 0x29 if it is connected to GND. The purpose for this is if another I2C component has the same address it can be easily changed.
* ESP8266 WiFi Module
  + The purpose of this component is to provide WiFi capabilities to different applications. For the purpose of this project, the ESP8266 will send information over WiFi using AT commands to an IoT cloud server, Thingspeak. The ESP8266 has to be flashed with AT firmware prior to being used the image below shows the ESP8266 Download Tool and which file and settings used to flash the ESP8266 chip.



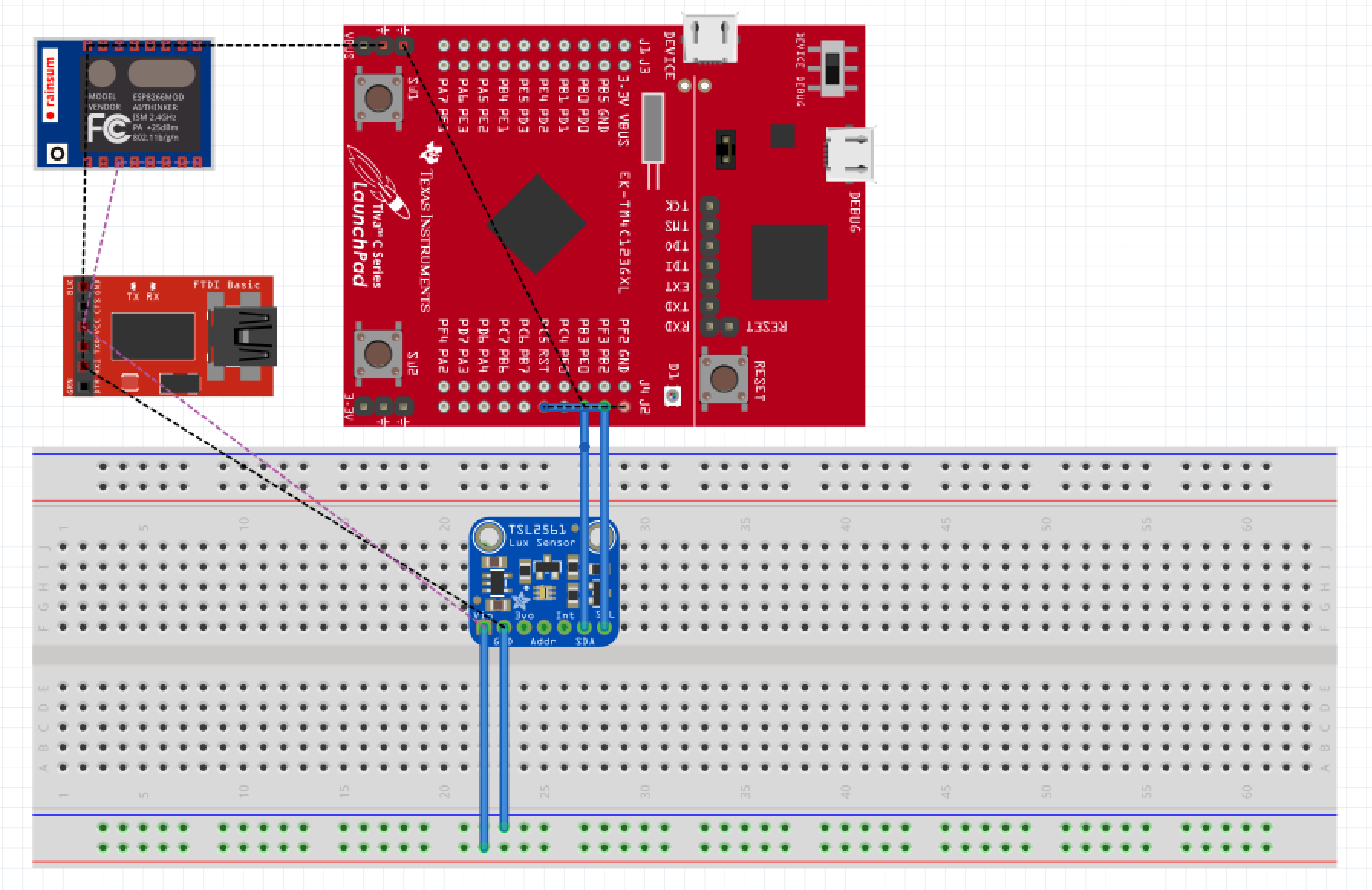
* Sparkfun FTDI Breakout Chip
  + The FTDI chip is responsible to read Serial input/output and display it so that the user can visibly see what is being transmitted through terminal (PuTTY)
* Breadboard
  + The breadboard was used mainly to connect the components needed for the whole project.
* Power Supply
  + The Power supply was used to output a 3.3V voltage and have a common ground.
* TIVA C TM4C123G Microcontroller
  + The purpose of this microcontroller is to facilitate the controls of the ESP8266 and the TSL2561 Lux sensor. The Tiva C will be programmed with a written code via Code Composer Studio, the program will be responsible for sending AT Commands to the ESP8266 chip. Prior to the AT commands the program will calculate the LUX values through I2C and will put those values within an AT command to send to thingspeak.
* Male to Male/Female to Female wires
  + Multiple wires were used to connect all the components above.

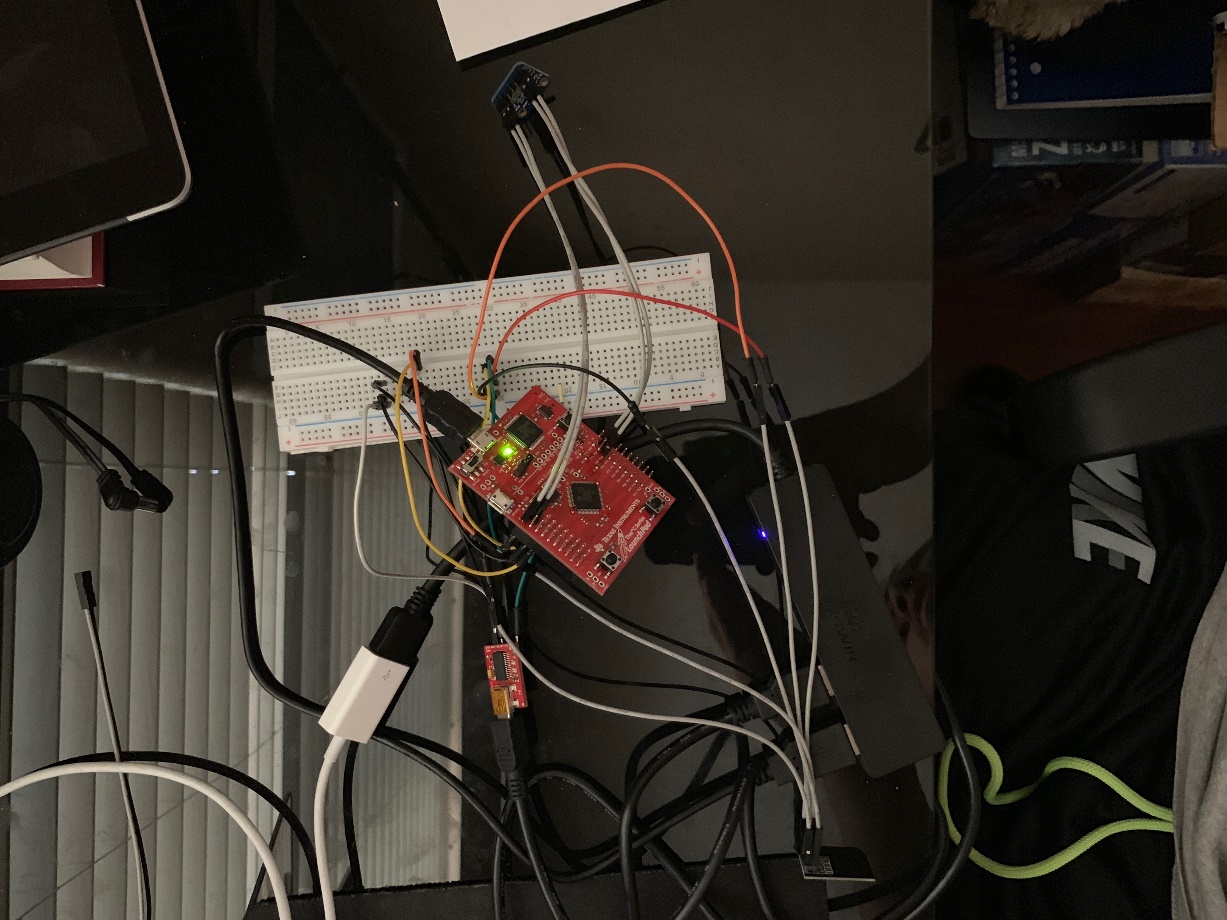
SCHEMATICS:



Above is the schematic for the project in its entirety. The TIVA C TM4C123G Microcontroller is connected to the TSL2561 using SCL (PB2) and SDA (PB3). The TSL2561 is also connected to 3.3V and ground. Furthermore, the ESP8266 WiFi Module is connected to the TIVA C RX (PB0) and TX (PB1). The module is also connected to 3.3V and Ground to be powered independently. To ensure that the AT Commands were working, I also connected the FTDI Breakout board to the RX and TX pins of the microcontroller to read the serial input/output.

Below is the breadboard portion modified from the schematic using Fritzing.





Above is the breadboard connection created through Fritzing and the actual implementation/circuit. The difficulty I found taking the photo was that the TIVA C has connections on the top using female wires and on the bottom using male wires.

IMPLEMENTATION:

**UART:**

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

{

**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

}

The above selected portion of the program is responsible for configuring the UART peripheral. This UART configuration is different then previous labs. When configuring this UART section, it is supposed to send information to the ESP8266 pins (PB0 and PB1) instead of through the PuTTY terminal. Furthermore, the UART has to be configured to 115200 Baud Rate as well.

**I2C Initialization:**

**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

}

When initializing the I2C peripheral PORTB needs to be set since PB3 and PB2 are responsible for the I2C pins: SDA and SCLK respectively. Furthermore, the clock for I2C needs to be set and the program will wait until the SDA is not busy.

**I2C Write:**

**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);

}

}

The I2C Write function is responsible for writing data from the master to the slave. This is a crucial function for retrieving information and having the component and the microcontroller talk accordingly.

**I2C Read:**

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

}

The I2C Read function is responsible for reading data from the slave to the master. This is a crucial function for I2C devices. This function gets called repeatedly, and if there are any issues among the I2C components then the program will get stuck within one of the while loops.

**Initializing the TSL2561 Lux Sensor:**

**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 == 0x00)

{

// UARTprintf ("GOT IT! %i\n", x); //used during debuging to make sure correct ID is received

}

**else**

{

**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

}

This function is responsible for checking to ensure that the correct address has been identified within the provided TSL2591 header file. The reason why is when I2C0\_Read is called it will return either a high or low (0xFF or 0x00) if the address is correct it will return 0x00 else 0xFF. By removing the “TSL2591\_COMMAND\_BIT” from the I2C0\_Read function, the x value will be the ID defined in the header which is 0x12.

**GetLuminosity() Get Lux Values:**

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\_C0DATAH));

x <<= 16;

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

ch1 = x>>16; //shift

ch0 = x & 0xFFFF; //and

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;

}

The above code is written to calculate lux values that have been read from the I2C sensor. After extensive troubleshooting and debugging, I realized that the header file provided had different addresses defined for ch0 and ch1, I changed them to be 0x8C and 0x8D.

**Receive Lux Value:**

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

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

{

lux = GetLuminosity (); //receive lux values

luxAvg += lux; //sum

}

luxAvg = luxAvg/20; //average value of LUX sensor

This function is responsible for calculating the average value of the lux sensor. The for loop will add the sum of 20 lux values and then after the for loop, the program will calculate the average value and store that in the luxAvg variable.

**AT Commands:**

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

**SysCtlDelay** (100000000);

**UARTprintf** ("AT+CWMODE=1\r\n"); //set mode to WiFi

**SysCtlDelay** (100000000);

**UARTprintf** ("AT+CIPMUX=0\r\n"); //enable one connection

**SysCtlDelay** (20000000);

**UARTprintf** ("AT+CIPSTART=\"TCP\",\"184.106.153.149\",80\r\n"); //Establish a connection with the thingspeak servers

**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=R73A74QVEV8C1XZM&field2=%d&headers=falseHTTP/1.1\nHostapi.thingspeak.com\nConnection:close\Accept\*\\*\r\n\r\n", luxAvg);

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

**SysCtlDelay** (50000000);

**UARTprintf** (HTTP\_POST); //send the string of the HTTP GET to the ESP8266

**SysCtlDelay** (50000000);

**Entire 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** "utils/uartstdio.h"

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

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

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

**#include** "utils/ustdlib.h"

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

{

**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 == 0x00)

{

// UARTprintf ("GOT IT! %i\n", x); //used during debuging to make sure correct ID is received

}

**else**

{

**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\_C0DATAH));

x <<= 16;

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

ch1 = x>>16; //shift

ch0 = x & 0xFFFF; //and

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

{

**char** HTTP\_POST[300]; //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, i;

uint32\_t luxAvg = 0;

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

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

TSL2591\_init(); //initialize the TSL2591

**SysCtlPeripheralEnable** (SYSCTL\_PERIPH\_HIBERNATE); //enable button 2 to be used during hibernation

**HibernateEnableExpClk** (**SysCtlClockGet**()); //Get the system clock to set to the hibernation clock

**HibernateGPIORetentionEnable** (); //Retain the pin function during hibernation

**HibernateRTCSet** (0); //Set RTC hibernation

**HibernateRTCEnable** (); //enable RTC hibernation

**HibernateRTCMatchSet** (0, 1800); //hibernate for 30 minutes

**HibernateWakeSet** (HIBERNATE\_WAKE\_PIN | HIBERNATE\_WAKE\_RTC); //allow hibernation wake up from RTC time or button 2

**while**(1)

{

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

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

{

lux = GetLuminosity (); //receive lux values

luxAvg += lux; //sum

}

luxAvg = luxAvg/20; //average value of LUX sensor

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

**SysCtlDelay** (100000000);

**UARTprintf** ("AT+CWMODE=1\r\n"); //reset the esp8266 before pushing data

**SysCtlDelay** (100000000);

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

**SysCtlDelay** (20000000);

**UARTprintf** ("AT+CIPSTART=\"TCP\",\"184.106.153.149\",80\r\n"); //Establish a connection with the thingspeak servers

**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=R73A74QVEV8C1XZM&field2=%d&headers=falseHTTP/1.1\nHostapi.thingspeak.com\nConnection:close\Accept\*\\*\r\n\r\n", luxAvg);

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

**SysCtlDelay** (50000000);

**UARTprintf** (HTTP\_POST); //send the string of the HTTP GET to the ESP8266

**SysCtlDelay** (50000000);

}

}