Martin Villasenor

10/30/2018

MidTerm Project

Tiva C Midterm Report

YouTube Link with demo: https://youtu.be/kFJOQJpEcqU

GOAL

The ***goal*** of the project is to obtain intensity light data produced by the sensor TSL2591 and send this data to the ThinkSpeak IoT server.

HARDWARE

The TM4C123GXL is utilized in this project to connect to the sensor TSL2591, the ESP8266 WiFi module, and a PC console.

The TSL2591 sensor is powered using the 3.3 Volt pin on the TM4C123GXL board and grounded to the GND pin. The TSL2591 is an I2C device. I2C is a two-wire protocol. The SDA line is used to transmit data and the SCL is the clock. The SDA line is connected to the PB3 pin which is set to I2C0SDA. The SCL line connects to pin PB2 which is set to I2C0SCL in the I2C0 module.

The ESP8266 is a processor used by the ESP-01 WiFi module in this project to connect to a local WiFi router which connects to the Internet. This module is utilized to send data to the ThinkSpeak cloud server. This service collects the data and employs a data display tool designed by MatLab.

The TM4C123GXL communicates to the ESP-01 via the UART protocol. In this project the UART1 module is enabled to establish communication between the TIVAC board and the ESP board. The pins mapped to UART1 is PB0 (U1Rx) and PB1(U1Tx). The TXD pin on the ESP board is connected to PB0(U1Rx) pin since transmission from the WiFi module is reception from the TIVAC board. Similarly, the RXD pin on the ESP board connects to PB1(U1Tx).

A PC console (Putty) is used to observe the messages sent from the TIVAC board to the ESP module. There is second set pins connecting to the UART1 module. Pins PC4(U1Rx) and PC5(U1Tx). A FTDI cable is utilized to interface the PC COM driver and the UART1 module on the TIVAC board.

The schematic shown in figure 1 represents the implementation described.

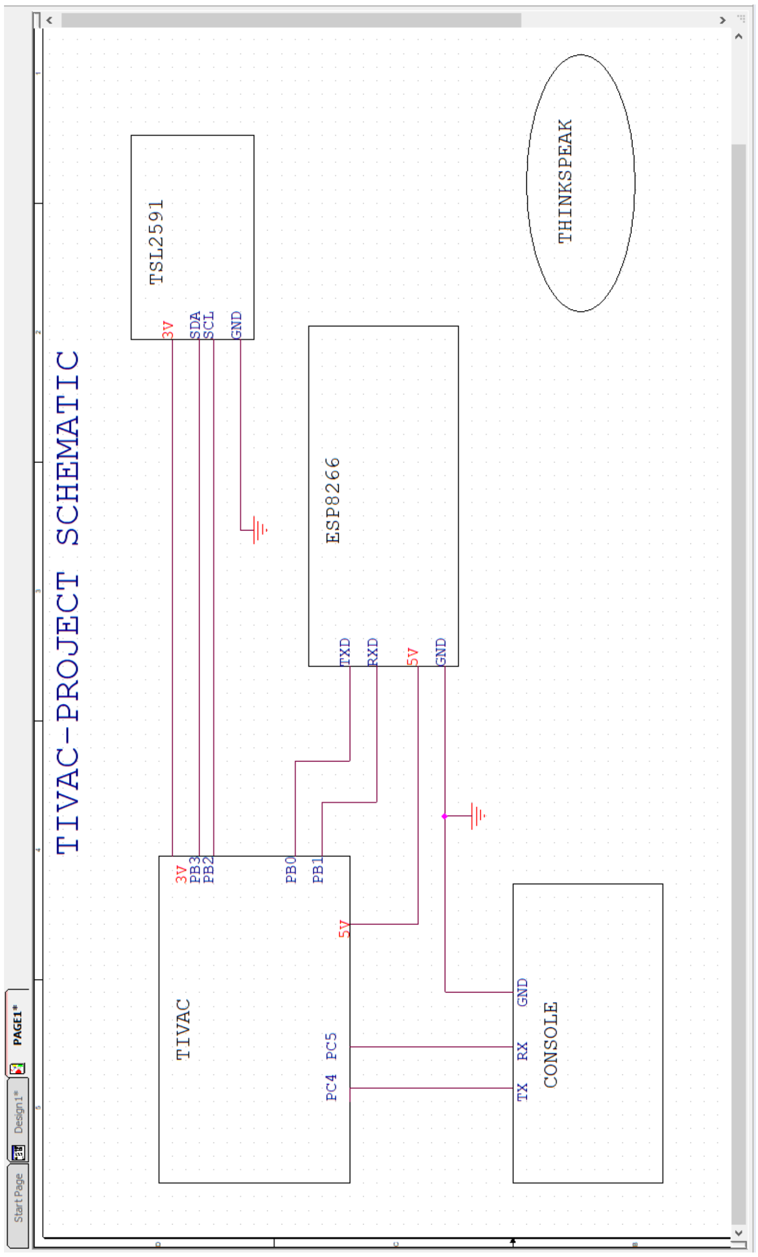
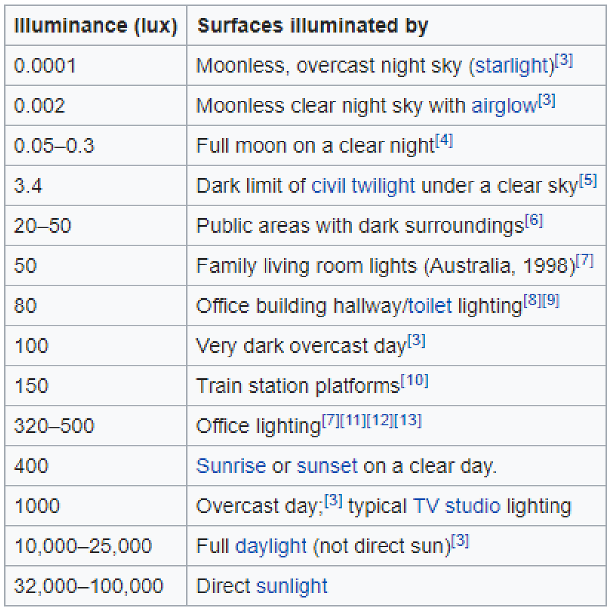

TIVAC-PROJECT SCHEMATIC
TSL2591
3V
SDA
SCL
GND
ESP8266
TXD
RXD
5V
GND
THINKSPEAK
TIVAC
3V
PB3
PB2
PB0
PB1
5V
PC5
PC4
CONSOLE
GND
RX
TX
PAGE1*
Design1*
Start Page

Fig1.

SOFTWARE AND SERVER

The TM4C123GXL receives light intensity data in digital form from the TSL2591 sensor via the I2C protocol. The data is a representation of the luminous flux per unit area of the visible and infrared sections of the EM spectrum sensed by a photodiode. A typical value for a family living room is 50 lm/m^2. However, in this experiment the values were shown to be less, averaging values between 0 and 30. The table shows different examples of LUX sources and respective values.



Illuminance (lux)	Surfaces illuminated by
0.0001	Moonless, overcast night sky (starlight)[3]
0.002	Moonless clear night sky with airglow[3]
0.05–0.3	Full moon on a clear night[4]
3.4	Dark limit of civil twilight under a clear sky[5]
20–50	Public areas with dark surroundings[6]
50	Family living room lights (Australia, 1998)[7]
80	Office building hallway/toilet lighting[8][9]
100	Very dark overcast day[3]
150	Train station platforms[10]
320–500	Office lighting[7][11][12][13]
400	Sunrise or sunset on a clear day.
1000	Overcast day;[3] typical TV studio lighting
10,000–25,000	Full daylight (not direct sun)[3]
32,000–100,000	Direct sunlight

Table1.

The software code initializes the TSL2591 module addressing it to its fixed value of 0x29 and commands specified by the datasheet. A function GetLuminosity is defined to read the raw data from the device and use these values to calculate the Luminous flux utilizing a formula provided by the datasheet of the device.

This flux value is taken 20 times and an average is taken. This average is sent to the ESP-01 module via UART and displayed on a Putty console.

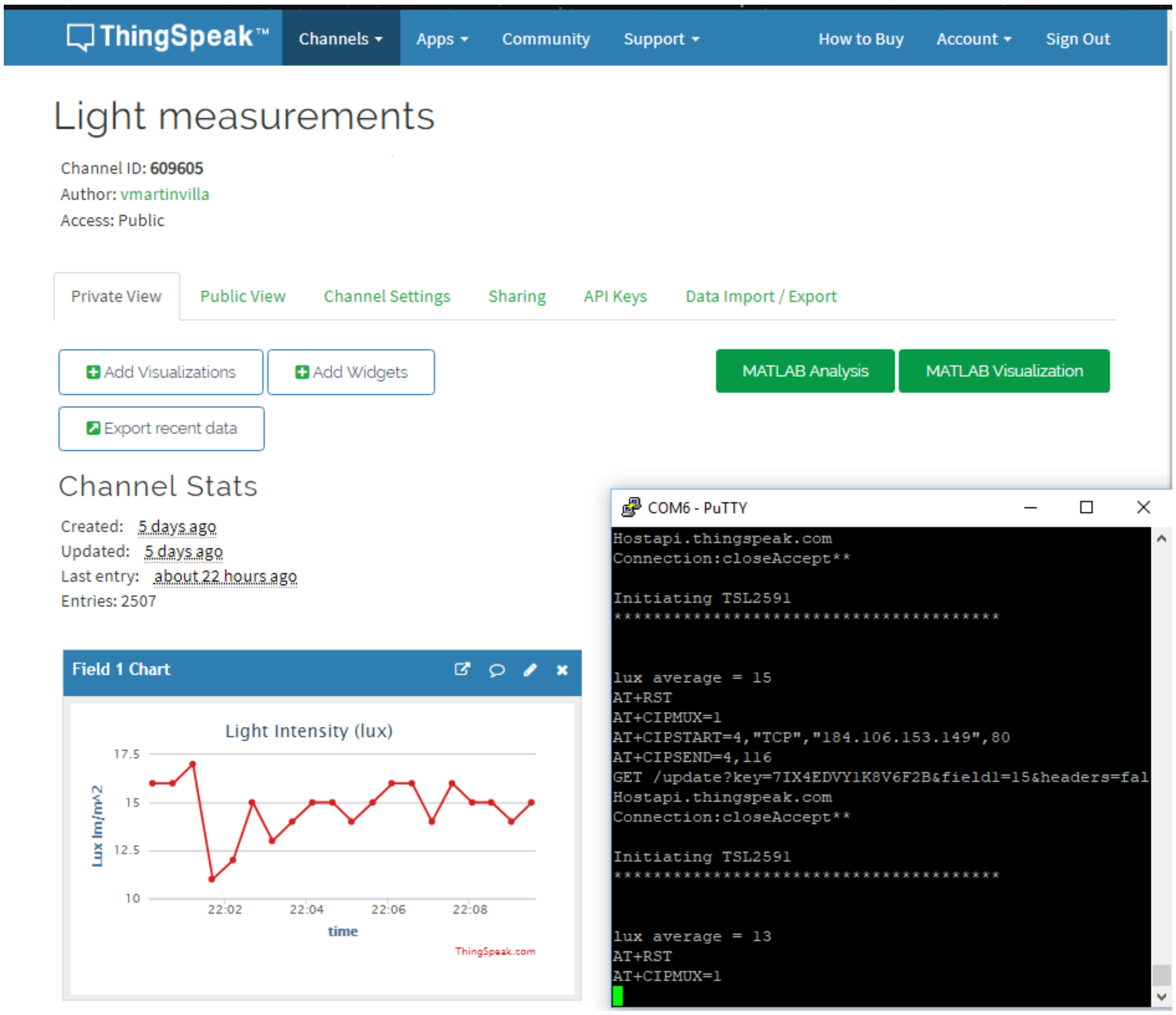
Prior to using the ESP-01 WiFi module, this had to be updated with the latest image available online. Once updated, this was connected to Putty to send commands to it for configuration. Part of the configuration was to obtain the local WiFi Router attributes for connection. Once connected, this is ready to post and get http messages.

The ESP-01 connects to the ThinkSpeak server IP address via TCP. To access the server, a channel needs to be created with a MATLAB account. An API key is provided which is used in the code to send data to the created channel.

The channel created displays a chart displaying the data received in a graphical form.

The code is implements the hibernation functionality to 30 seconds. One sample of the average LUX is collected and sent to ThinkSpeak. The Chart displays the data point distribution on a given time interval.

Figure 2 shows the ThinkSpeak channel created which can be accessed for public view. Channel ID provided to check data from server. The console is also displayed to corroborate the data sent from the TIVAC and the data point collected by the server and displayed on the Field Chart.



Light measurements

Channel ID: **609605**
Author: vmartinvilla
Access: Public

Channel Stats

Created: 5 days ago
Updated: 5 days ago
Last entry: about 22 hours ago
Entries: 2507

```
Hostapi.thingspeak.com
Connection:closeAccept**

Initiating TSL2591
***************************************


lux average = 15
AT+RST
AT+CIPMUX=1
AT+CIPSTART=4,"TCP","184.106.153.149",80
AT+CIPSEND=4,116
GET /update?key=7IX4EDVY1K8V6F2B&field1=15&headers=fal
Hostapi.thingspeak.com
Connection:closeAccept**

Initiating TSL2591
***************************************


lux average = 13
AT+RST
AT+CIPMUX=1
```

Fig2.

Figure 3 shows 20 samples of data. This corresponds to 10 minutes since each data point is collected every 30 seconds.

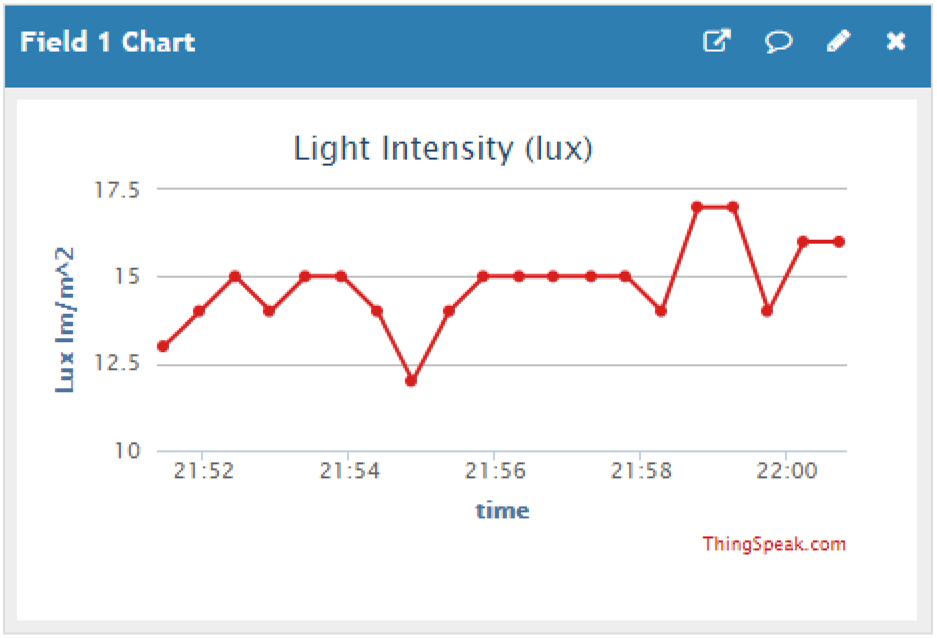


Fig3. 20 samples, 10 minutes

Figure 4 shows 120 samples of data collected in one hours

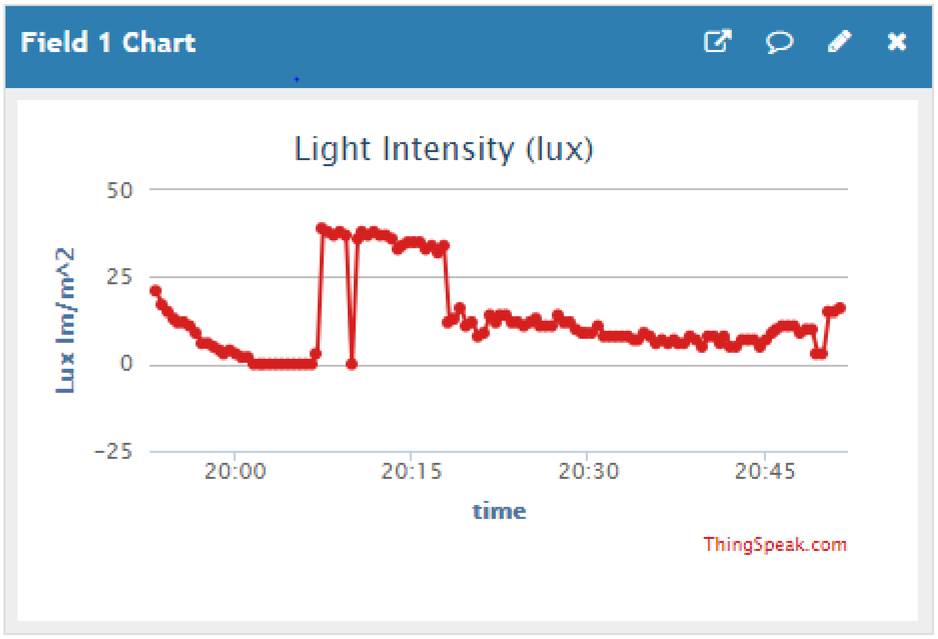


Fig4. 120 samples, 1 hr

Figure 5 shows 2880 samples of data collected in 24 hrs.

One observation about this data is that during night time when lights were on during the time working on the experiment, the collected data shows an average of 30 to 35 Lux, then it falls to an average of zero which is when the lights were off, about after midnight. Then in the morning, around 6:30 am, the graph shows activity which is light coming from the sun outside. Then fluctuates, but the average is maintained at about 35 to 40 Lux. It starts to decay in the evening again. That's because the room where the sensor is located is opposite to the sunset direction and therefore sun light does not come through the room. Lights were turn on later in the evening, that the last light activity is displayed on the graph.

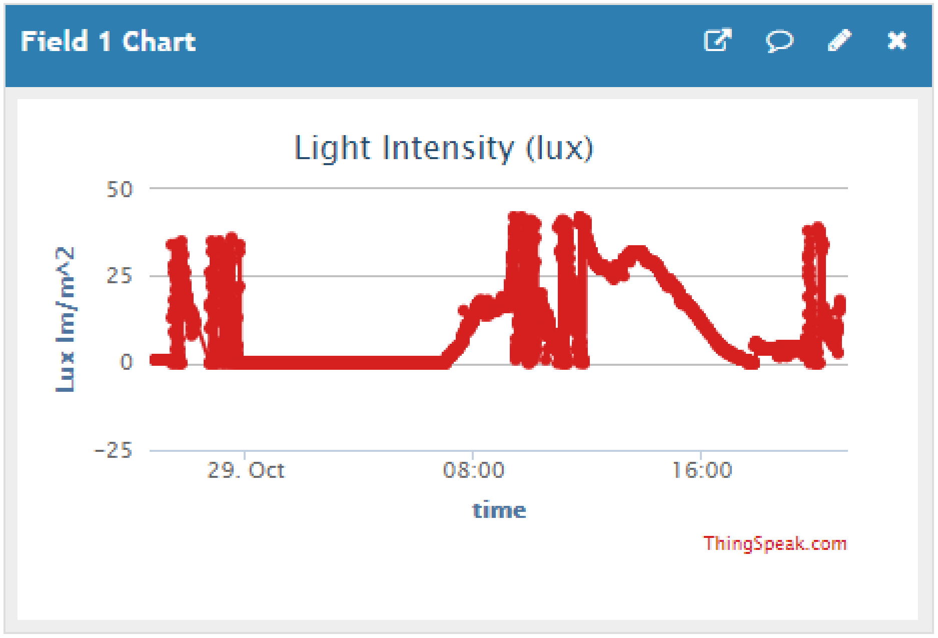


Fig5. 2880 samples, 24 hrs

Code used to implement this project.

```
#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 "TSL2591_def.h"
#include "driverlib/debug.h"
#include "utils/ustdlib.h"




void ConfigureUART1(void)
//Configures the UART to run at 115200 baud rate
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);    //enables UART module 1
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);    //enables GPIO port b

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);    //enables GPIO port c for PC console
viewing

    GPIOPinConfigure(GPIO_PB1_U1TX);    //configures PB1 as TX pin
    GPIOPinConfigure(GPIO_PB0_U1RX);    //configures PB0 as RX pin

    GPIOPinConfigure(GPIO_PC5_U1TX);    //configures PC5 as TX pin for port c
    GPIOPinConfigure(GPIO_PC4_U1RX);    //configures PC4 as RX pin for port c

    GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);  //sets the UART pin type

    GPIOPinTypeUART(GPIO_PORTC_BASE, GPIO_PIN_4 | GPIO_PIN_5);  //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)
    {
       // UARTprintf ("GOT IT! %i\n", x);
      //  UARTprintf ("Initiating TSL2591 \n");
      //  UARTprintf ("*************************************** \n\n\n");
    }
    else
    {
        UARTprintf ("DO NOT HAVE IT! %i\n", x);
        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 and 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 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;

    ConfigureUART1 ();   //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
    HibernateRTCMatchSet (0, 30); // hibernating for 30 seconds
    HibernateWakeSet (HIBERNATE_WAKE_PIN | HIBERNATE_WAKE_RTC); //allow hibernation wake up
from RTC time or button 2

    for (i = 0; i < 20; i++)
    //finds the average of the lux channel to send through uart
    {
        lux = GetLuminosity ();
        luxAvg += lux;
    }
    luxAvg = luxAvg/20;


    UARTprintf ("lux average = %d\r\n",luxAvg); // Viewing values on console every 30 seconds

    UARTprintf ("AT+RST\r\n");  //reset the esp8266 before pushing data
    SysCtlDelay (100000000);


    UARTprintf ("AT+CIPMUX=1\r\n"); //enable multiple send ability
    SysCtlDelay (20000000);
    UARTprintf ("AT+CIPSTART=4,\"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=7IX4EDVY1K8V6F2B&field1=%d&headers=falseHTTP/1.1\nHostapi.thingspeak.com\nConnecti
on:close\Accept*\*\r\n\r\n", luxAvg);
    UARTprintf ("AT+CIPSEND=4,%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);

    HibernateRequest ();    //Hibernate
    while (1)
    {};
}
```