Tiva-C Midterm

Implementing a wireless lux sensor with cloud-based data visualization.

Youtube Playlist Link: <https://www.youtube.com/playlist?list=PL4oTyvRrubXcf6an9ZR-QO2O_Jaz2sybg>

Youtube Video 1: <https://youtu.be/M_Vm90tpS3A>

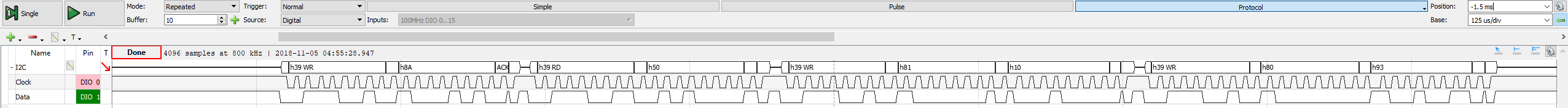
Youtube Video 2: <https://youtu.be/0w7UUtwUdjQ>

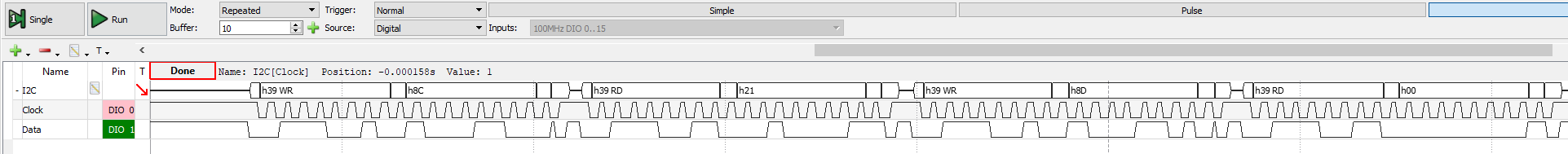
# Goals:

* Collect data from TSL2561 lux sensor at intervals between 15 and 60 seconds for an extended period of time between 1 and 24 hours using I2C protocol.
* Upload data using ESP8266 module into IoT cloud service (ThingSpeak) using UART.

# Deliverables:

* Code used to initialize and use components for the purposes of gathering data.
  + Completed, detailed below.
* Collection of data from lux sensor measuring ambient light over an extended period and associated timestamps.
  + Completed, viewable at <https://thingspeak.com/channels/619310>





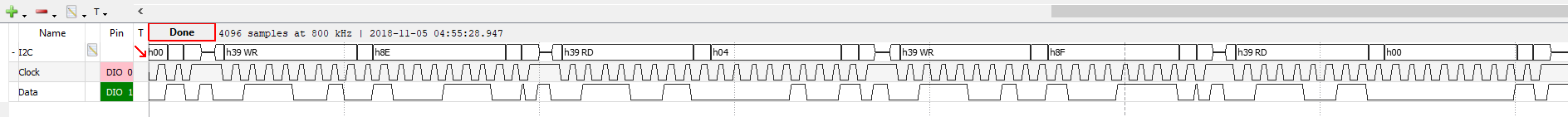


Figure 1 - I2C initialization of TSL2561 and reading Channel 1 and 2 data words.

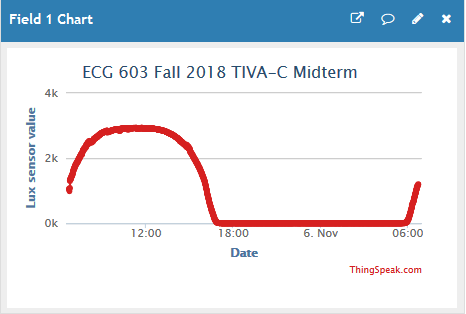


Figure 2 - 24 hours of data collected via ThingSpeak server.

# Components:

* EK-TM4C123GXL-Tiva™ C Series TM4C123G LaunchPad Evaluation Kit
  + Main Characteristics:
    - Primary microcontroller executing provided code.
    - Internal clock set at 40 MHz.
  + Interfaces used:
    - GPIO Channel B
      * UART1 (PB0, PB1)
      * I2C (PB2, PB3)
  + Limitations:
    - Wireless connectivity not built in.
      * Additional BOM cost.
      * Additional wiring.
    - Additional unused features
      * Useful for learning, but less optimized for single-task.
      * Additional power draw.
  + Systems Initialized:
    - System clock (40 MHz)
    - GPIO Port B
    - UART Module 1 (GPIO PB0, 1)
    - I2C (GPIO PB2, PB3)
* Adafruit TSL2561 Digital Luminosity/Lux/Light Sensor Breakout
  + Main Characteristics:
    - Dynamic range 0.1-40,000 lux.
      * Operating in low-gain mode, limited lower-lux resolution.
    - Low power draw, 0.5mA active, 15μA in power-down mode.
    - I2C communication with three selectable addresses.
    - Built-in ADC for ease of use with ADC-less devices.
    - Two photodiodes measuring Broadband and Infrared for use approximating human eye response, particularly in high-IR environments (daylight, incandescent lighting).
    - Can generate interrupts based on lighting levels.
  + Interfaces used:
    - I2C
  + Limitations:
    - Limited upper range limits outdoor/extreme brightness uses.
    - Limited accuracy, only approximates lux with two photodiodes.
  + Register settings used:
    - Timing Register – Low gain, 101mS conversion time.
    - Control Register – Power on, AEN/AIEN/NPIEN enabled (Available on TSL2591, not used for TSL2561).
* Adafruit HUZZAH ESP8266 Breakout (substituted for supplied ESP8266)
  + Main Characteristics:
    - Includes ESP8266 Wireless connectivity (Access Point and Client).
    - Reset and User/Bootloader button for easy programming.
    - Integrated 3.3v 500mA regulator for use in 5V circuits.
      * Useful for powering other 3.3v devices, such as the TSL2561.
    - Dual power input pins for battery/FTDI usage.
  + Interfaces used:
    - UART
  + Limitations:
    - Significant pulsed power requirements may require additional design to avoid interference with other modules.

# Schematic:

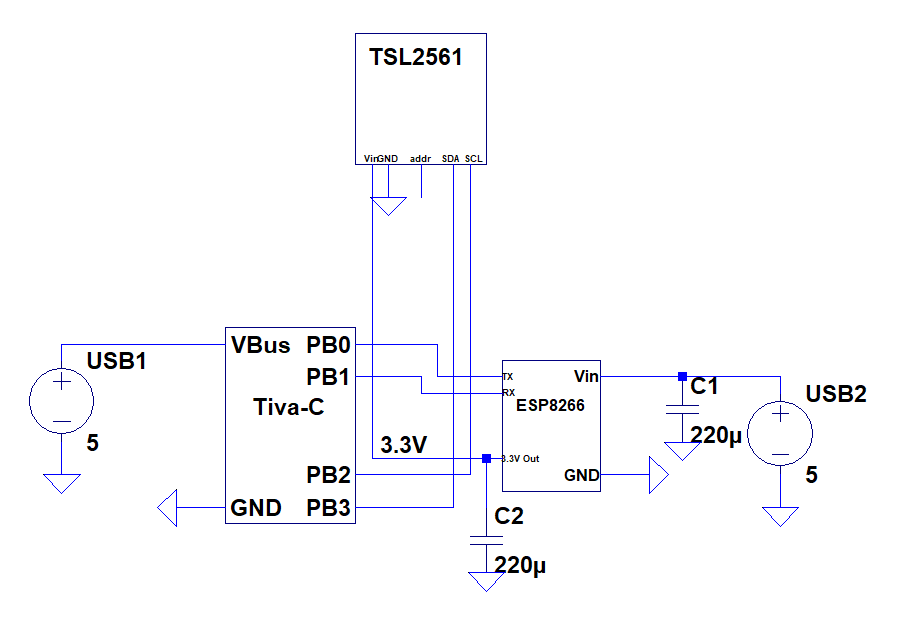


Figure 3 - Simplified schematic of device connections.

Shown above is a basic schematic for our circuit. Our Tiva-C UART connections are PB0 and PB1, connected to our ESP8266 Wireless device, while PB2 and PB3 use I2C to communicate with our TSL2561 lux sensor. We have one USB cable powering our Tiva-C board through the “device” connection, and a separate USB connection powering our ESP8266, which has an onboard 5V-3.3V LDO regulator which powers our TSL2561.

# Implementation:

The code implemented in this project can be summarized as the following:

1. Declare all #include and #define statements required for the code to compile and run.
   1. Including our .h files allows us to recognize and access functions contained in other .C files, such as our I2C and GPIO code.
   2. Our define statements allow us to conveniently define variables or settings to be considered and substituted on compile, rather than stored as a variable and taking up memory. In this case, I store my ThingSpeak key. It doesn’t need to be saved as a full variable, only inserted into a larger hard-coded string further in the code.
2. Set system clock to 40 MHz.
   1. System clock must be set for any other systems to run. Many of our other systems refer to our system clock for their own internal timing.
3. Configure and initialize our UART subsystem.
   1. We enable our on-board UART module 1 and GPIO Port B for use. We will use PB0 to receive and PB1 to transmit. Next, we set our clock source and configure our baud rate, clocking source, and which module we intend to use (UART 1).
4. Configure and initialize our I2C subsystem.
   1. We enable our I2C 0 module and GPIO Port B (already enabled previously, but good for modularity). PB2 will be our clocking pin, PB3 will be data. Next, we enable the master clock of the module and wait until the device is ready.
      1. Read and Write functions are separately created within our code to send and receive data via the I2C protocol, each sending and receiving data and waiting for the packet to finish before moving on to the next.
5. Configure and initialize our TSL2561 module.
   1. We read our device ID based on registers previously defined in our .h file include with our #include statements. If the device responds with the expected ID, we write to the timer register setting it for low gain and 101ms integration time, and then write to our control register to enable power.
   2. This code also includes enables for the AEN, AIEN, and NPIEN devices which are present on the TSL2591, but not the TSL2561. This does not seem to negatively affect our device and is left included to allow potential future use of the TSL2591.
6. Configure Tiva-C Hibernation.
   1. We enable our hibernation subsystem and enable button 2 to wake our device. Next, we set our hibernation clock to our previously-defined system clock. We set our GPIO pin functions to be retained during and after hibernation and enable our real-time clock to run and wake the device after 30 seconds of hibernation.
7. Find average lux value over 20 readings.
   1. The code calls a function to read the lux value from our TSL2561 20 times and averages their values. To read the lux value, we request the values through I2C of channel 0 and channel 1 on the TSL2561. Channel 0 is our broadband photodiode, and Channel 1 is our IR-sensitive photodiode. Their values can be subtracted to remove much of the IR-component to approximate human-eye response. Each channel supplies a 16-bit ADC, stored in two registers. The two registers for channel 0 and channel 1 are read, the high registers shifted over by 8 bits to represent the whole 16-bit value. Next, lux value is calculated per a series of equations provided by the datasheet and supplied parameters which are checked against. This value is returned to be stored in our lux variable.
8. We send the values to our ThingSpeak server channel.
   1. We use UART to send a reset command and enable multiple send ability in our ESP8266 device, giving pause for the device to process the commands. Next, we instruct the device to connect with our ThingSpeak server. Once connected, we create a string consisting of a command to use the ThingSpeak API to send the lux data to our channel, defined via the pre-defined key, then pass the string to our ESP8266 to UART. Once the string is sent, we instruct our Tiva-C to hibernate, running the program again once it wakes.

# 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** "TSL2561\_mod.h"

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

**#define** key "LGCG7W8V9A9OGSMX"

//Note: Was given TSL2561, not TSL2591. 2591 is high dynamic range, 2561 is not.

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

//Configures the UART to run at 119200 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, 119200, 16000000); //enables UARTstdio baud rate, clock, and which UART to use

}

// lux equation approximation without floating point calculations per TSL2561 datasheet.

//////////////////////////////////////////////////////////////////////////////

// Routine: unsigned int CalculateLux(unsigned int ch0, unsigned int ch0, int iType)

//

// Description: Calculate the approximate illuminance (lux) given the raw

// channel values of the TSL2560. The equation if implemented

// as a piece−wise linear approximation.

//

// Arguments: unsigned int iGain − gain, where 0:1X, 1:16X

// unsigned int tInt − integration time, where 0:13.7mS, 1:100mS, 2:402mS,

// 3:Manual

// unsigned int ch0 − raw channel value from channel 0 of TSL2560

// unsigned int ch1 − raw channel value from channel 1 of TSL2560

// unsigned int iType − package type (T or CS)

//

// Return: unsigned int − the approximate illuminance (lux)

//

//////////////////////////////////////////////////////////////////////////////

**unsigned** **int** **CalculateLux**(**unsigned** **int** iGain, **unsigned** **int** tInt, **unsigned** **int** ch0,

**unsigned** **int** ch1, **int** iType)

{

//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

// first, scale the channel values depending on the gain and integration time

// 16X, 402mS is nominal.

// scale if integration time is NOT 402 msec

**unsigned** **long** chScale;

**unsigned** **long** channel1;

**unsigned** **long** channel0;

**switch** (tInt)

{

**case** 0: // 13.7 msec

chScale = CHSCALE\_TINT0;

**break**;

**case** 1: // 101 msec

chScale = CHSCALE\_TINT1;

**break**;

**default**: // assume no scaling

chScale = (1 << CH\_SCALE);

**break**;

}

// scale if gain is NOT 16X

**if** (!iGain) chScale = chScale << 4; // scale 1X to 16X

// scale the channel values

channel0 = (ch0 \* chScale) >> CH\_SCALE;

channel1 = (ch1 \* chScale) >> CH\_SCALE;

//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

// find the ratio of the channel values (Channel1/Channel0)

// protect against divide by zero

**unsigned** **long** ratio1 = 0;

**if** (channel0 != 0) ratio1 = (channel1 << (RATIO\_SCALE+1)) / channel0;

// round the ratio value

**unsigned** **long** ratio = (ratio1 + 1) >> 1;

// is ratio <= eachBreak ?

**unsigned** **int** b, m;

**switch** (iType)

{

**case** 0: // T, FN and CL package

**if** ((ratio >= 0) && (ratio <= K1T))

{b=B1T; m=M1T;}

**else** **if** (ratio <= K2T)

{b=B2T; m=M2T;}

**else** **if** (ratio <= K3T)

{b=B3T; m=M3T;}

**else** **if** (ratio <= K4T)

{b=B4T; m=M4T;}

**else** **if** (ratio <= K5T)

{b=B5T; m=M5T;}

**else** **if** (ratio <= K6T)

{b=B6T; m=M6T;}

**else** **if** (ratio <= K7T)

{b=B7T; m=M7T;}

**else** **if** (ratio > K8T)

{b=B8T; m=M8T;}

**break**;

**case** 1:// CS package

**if** ((ratio >= 0) && (ratio <= K1C))

{b=B1C; m=M1C;}

**else** **if** (ratio <= K2C)

{b=B2C; m=M2C;}

**else** **if** (ratio <= K3C)

{b=B3C; m=M3C;}

**else** **if** (ratio <= K4C)

{b=B4C; m=M4C;}

**else** **if** (ratio <= K5C)

{b=B5C; m=M5C;}

**else** **if** (ratio <= K6C)

{b=B6C; m=M6C;}

**else** **if** (ratio <= K7C)

{b=B7C; m=M7C;}

**else** **if** (ratio > K8C)

{b=B8C; m=M8C;}

**break**;

}

**unsigned** **long** temp;

temp = ((channel0 \* b) - (channel1 \* m));

// do not allow negative lux value

**if** (temp < 0) temp = 0;

// round lsb (2^(LUX\_SCALE−1))

temp += (1 << (LUX\_SCALE - 1));

// strip off fractional portion

**unsigned** **long** lux = temp >> LUX\_SCALE;

**return**(lux);

}

**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** **TSL2561\_init** ()

//Initializes the TSL2591 to have low gain, 100ms integration.

{

uint32\_t x;

x = I2C0\_Read (TSL2561\_ADDR, (TSL2561\_COMMAND\_BIT | TSL2561\_ID)); //read the device ID

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

{

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

}

**else**

{

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

}

I2C0\_Write (TSL2561\_ADDR, 2, (TSL2561\_COMMAND\_BIT | TSL2561\_CONFIG), 0x10); //configures the TSL2591 to have low gain and integration time of 100ms

I2C0\_Write (TSL2561\_ADDR, 2, (TSL2561\_COMMAND\_BIT | TSL2561\_ENABLE), (TSL2561\_ENABLE\_POWERON | TSL2561\_ENABLE\_AEN | TSL2561\_ENABLE\_AIEN | TSL2561\_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

//Channel 0 is the broadband photodiode, over both visible and infrared.

//Channel 1 is the primarily infrared detecting photodiode, to be subtracted out for our visible approximation.

//This section could be compressed to save memory and use less variables, but is broken out

//for ease of use, as we are not currently memory-constrained.

{

uint16\_t ch0, ch1; //variables to hold the channels of the TSL2561.

uint32\_t H0 = 0;

uint32\_t L0 = 0;

uint32\_t H1 = 0;

uint32\_t L1= 0;

uint32\_t lux = 0;

//Read High and Low channel bytes into ch0 word.

L0 = I2C0\_Read (TSL2561\_ADDR, (TSL2561\_COMMAND\_BIT | TSL2561\_C0DATAL));

H0 = I2C0\_Read (TSL2561\_ADDR, (TSL2561\_COMMAND\_BIT | TSL2561\_C0DATAH));

ch0 = (H0 \* 256) + L0;

//Read High and Low channel bytes into ch1 word.

L1 = I2C0\_Read (TSL2561\_ADDR, (TSL2561\_COMMAND\_BIT | TSL2561\_C1DATAL));

H1 = I2C0\_Read (TSL2561\_ADDR, (TSL2561\_COMMAND\_BIT | TSL2561\_C1DATAH));

ch1 = (H1 \* 256) + L1;

lux = CalculateLux(0, 1, ch0, ch1, 0); //low gain, 100ms integration, channels 0 and 1, T-type package.

**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 run at 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

TSL2561\_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, 30); //hibernate 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** ("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 https://api.thingspeak.com/update?api\_key=%s&field1=%d&headers=falseHTTP/1.1\nHostapi.thingspeak.com\nConnection:close\Accept\*\\*\r\n\r\n", key, 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)

{};

}

# Future Work / Areas of Improvement:

In this assignment, much was learned about the operation of I2C and UART, as well as interfacing and powering multiple devices. In finishing the project, I identified multiple areas of potential improvement.

First, verification of provided files and code against the datasheet would be a valuable first action. Not only will it prevent potential mistakes, but it will allow me to become more familiar with the tools with which I am working. In this design, I had multiple register definitions and configurations that were originally intended for a similar but different lux sensor. The registers that were defined were slightly different, and much time was lost before realizing the subtle difference.

Second, the code could be considerably optimized through both modularity and scrubbing of obsolete commented code. The flow of the code could be improved considerably by breaking out many of the functions into separate areas or potentially into separate .C files entirely. This would enable an easier method of verifying the code flow at a glance and make sweeping changes to the functionality much easier, as only single parameters would need to be changed rather than large portions of code. Many portions of code in this design were intended for another device with a wider feature set, these could easily be enabled or disabled via parameter calls, rather than hard-coding them in.

Third, more verification and debugging functions would be useful throughout the code. During the project I found myself using a logic analyzer to watch the I2C packet information directly, but many options such as writing to control registers had no such verification. A useful addition would be to request read of each write function to verify the changes occurred. There were a number of settings such as gain and integration time with our lux sensor that were difficult to verify, as default settings were in place if not correctly set and I did not have a well-calibrated way to verify the information given.

Fourth, a more developed power system will need to be developed to power these devices if they are to operate long-term wirelessly. While my 2-output USB power bank is providing power to the devices, I suspect that it may be shutting off due to low power drain from the light loads intermittently, as the ThingSpeak data has occasional gaps of 1-2 data cycles. For a device serving this function, a smaller rechargeable battery pack and small solar panel may be a good future development to investigate, possibly using the Tiva-C ADC to measure battery levels. The lux sensor could also be used to conserve power more aggressively, hibernating the unit for longer periods in periods of consistently-low light.