**TIVA C MIDTERM**

**Date Submitted: 10/30/2018**

Components List:

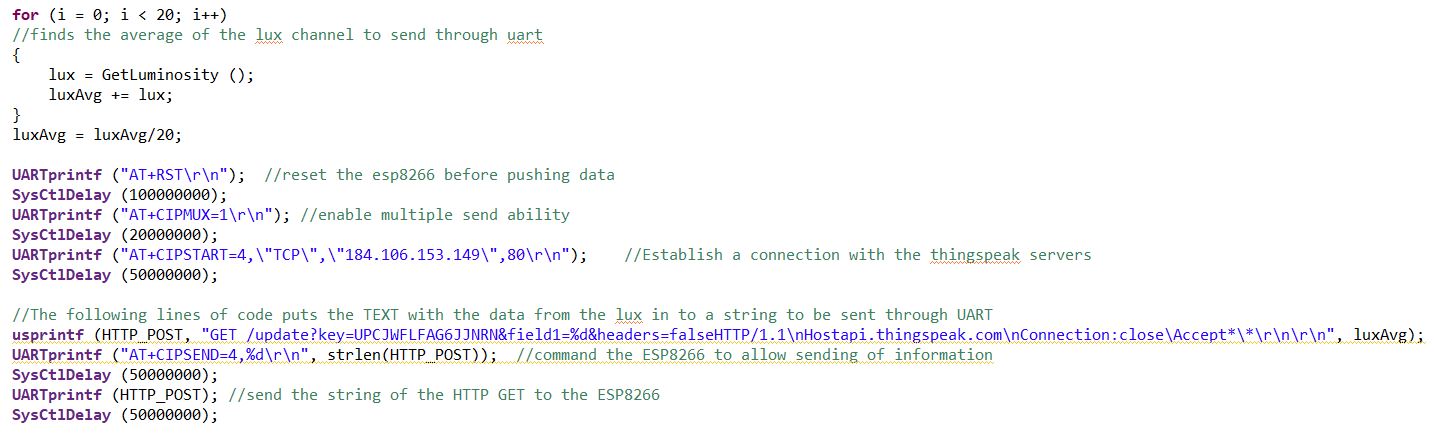
* FTDI Chip: Used to communicate and setup with the Wifi module
* ESP8266 module: Used for receiving lux sensor data from TIVAC board then transmit the data to ThingSpeak server.
* Breadboard and Jumper Wires: Used to connect all the components together
* TM4C123G Launchpad: This will be the foundation of the project. This microcontroller will contain the program we will be using to transfer record and transfer our Lux data to the cloud server (ThingSpeak).
* TSL2561 Lux Sensor: This sensor used to detect light. We will be using the I2C interface to properly communicate and use this sensor.

**1. Goal**

The goal of this project is to use a TSL2591 lux sensor (I had to use a TSL2561 because it was the one provided in my lab kit), to record data between set intervals for an extended period and then upload the data into a ThingSpeak server using an ESP8266 module provided in our lab kit.

**2. Detailed Implementation**

To get started I followed the midterm instructions provided by Dr. Venki in his lectures. By doing so I was able to successfully update the firmware of my ESP8266 WiFi module through the FTDI chip. After updating the firmware, I took the sample code provided by Dr. Venki and I modified it to keep writing entries into my ThingSpeak graph after a given period of time which was done by placing this portion of code located in main() inside a while loop. I also had to modify the usprintf statement with the API key used for my ThingSpeak server.



Another thing that I had to change this line of code shown below. The baud rate was initially set to 19000, but I switched it to 15200 because that was the baud rate I used while updating the firmware of the ESP8266 chip and the baud rate I used while setting up the ESP8266 connection to WiFi.



The portion below shows the full source code for this project.

**Modified Code:**

**// Insert code here**

**#define** PART\_TM4C123GH6PM

**#include** <stdarg.h> //include necessary libraries

**#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** "utils/uartstdio.c"

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

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

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

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

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

//Configures the UART to run at 15200 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, 15200, 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); //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\_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;

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) //this while loop will continuously record the averaged lux inputs

{

**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 /update?key=UPCJWFLFAG6JJNRN&field1=%d&headers=falseHTTP/1.1\nHostapi.thingspeak.com\nConnection: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);

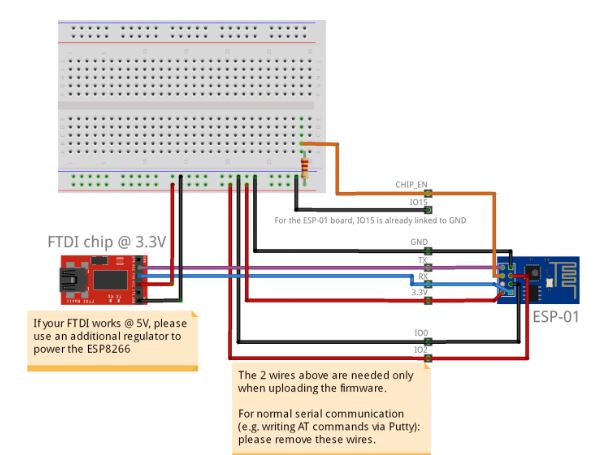
}

}

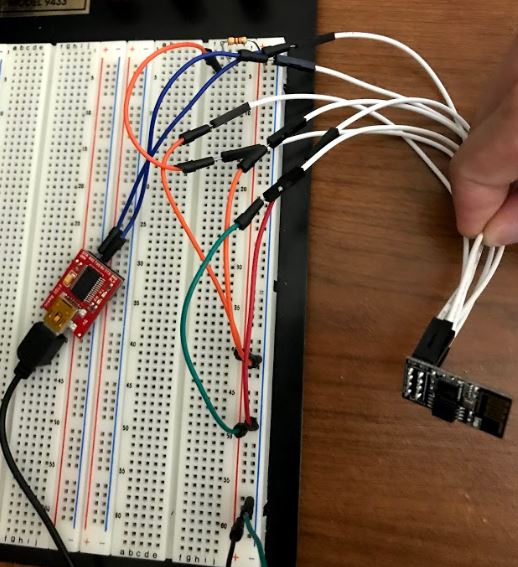
**------------------------------------------------------------------------------------**

**3. Schematics**

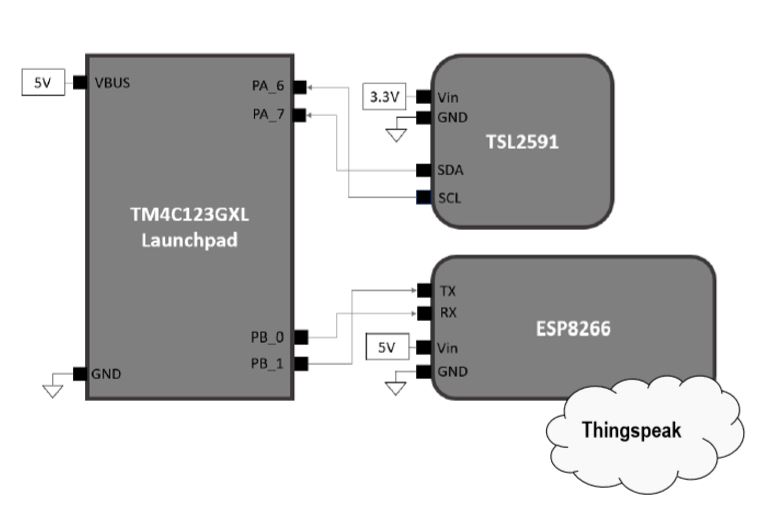
The fritzing schematic shown below is what I followed when I was updating the firmware of the ESP8266 chip. This was found in the link given to us by Dr. Venki. ( <http://flower-platform.com/2015/12/16/esp8266-with-at-commands-flashingupdating-the-firmware-step-by-step/> ). As we can see from the schematic below, the CHIP\_EN connects to one of the legs of the pull-up resistor. The ground pin on the esp chip goes to ground on the breadboard. The IO0 and IO2 go to ground and Vcc respectively. The TX on the esp chip goes to the RX on the FTDI chip and the RX on the esp chip goes to the TX of the FTDI chip. The ground on the FTDI chip goes to ground on the breadboard and the 5V on the FTDI chip (switched to 3.3V) goes to the Vcc column of the breadboard.

****

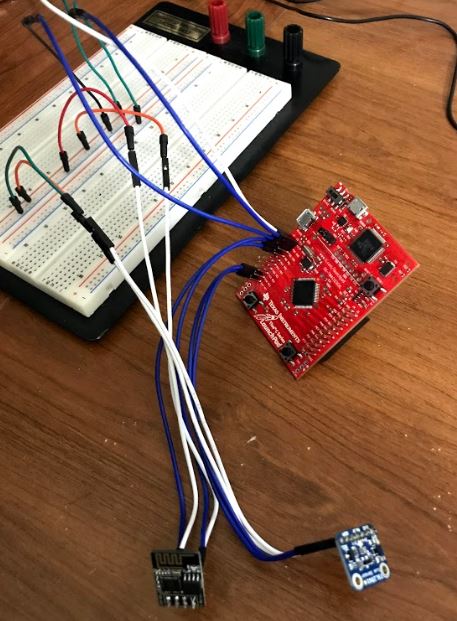
The image below shows my implementation of the fritzing schematic above. Notice that the 5V output on the FTDI chip is not connected, that is because I had to use the TivaC’s 3.3V output to power the chip.



The schematic below (obtained from Dr. Venki’s lecture) shows how to connect the TIVAC launchpad, lux sensor and wifi module together. I tried to make a fritzing schematic, but fritzing did not have the TivaC launchpad nor the TSL sensor, so I had to settle with the schematic/image given to us in the lecture notes.

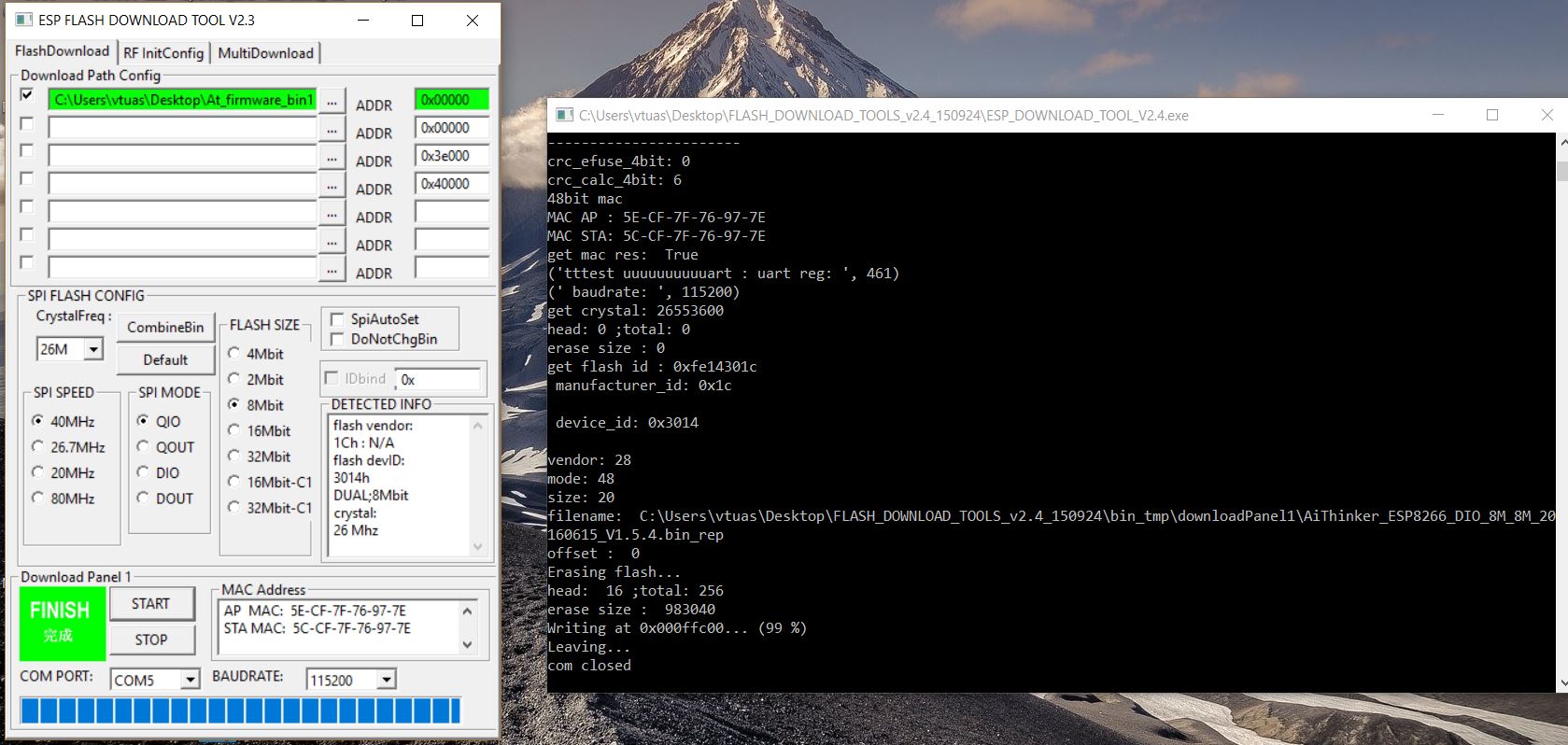


The image below is my implementation of the schematic above.

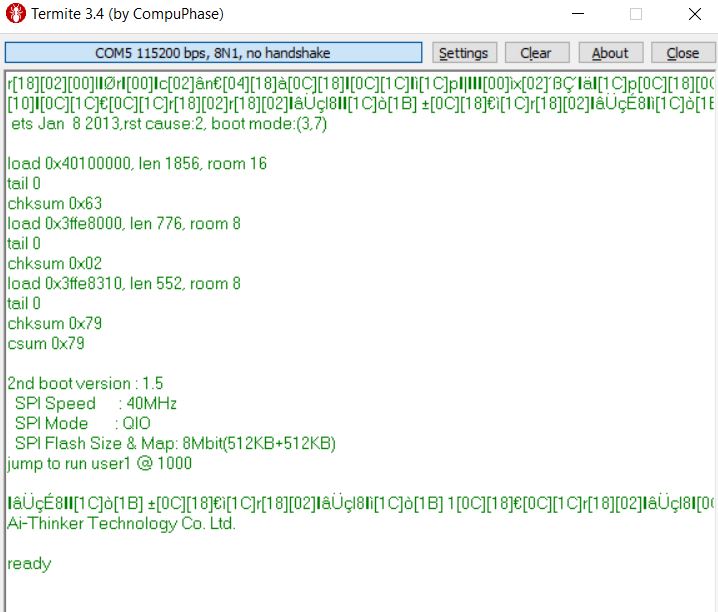


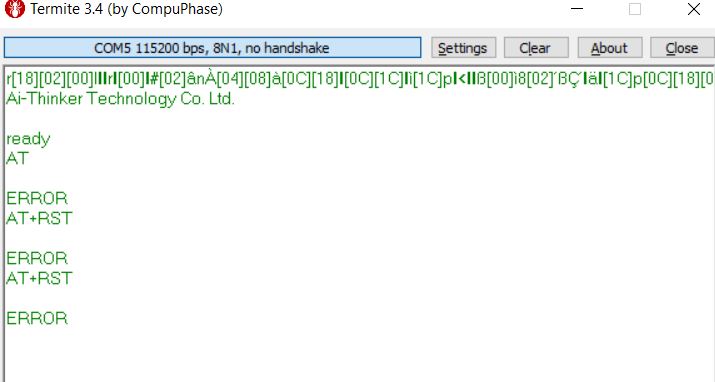
**4. Screenshots**

The image below shows that the ESP2866 chip was flashed with the latest AT firmware (the firmware which was linked in Dr. Venki’s lecture notes)

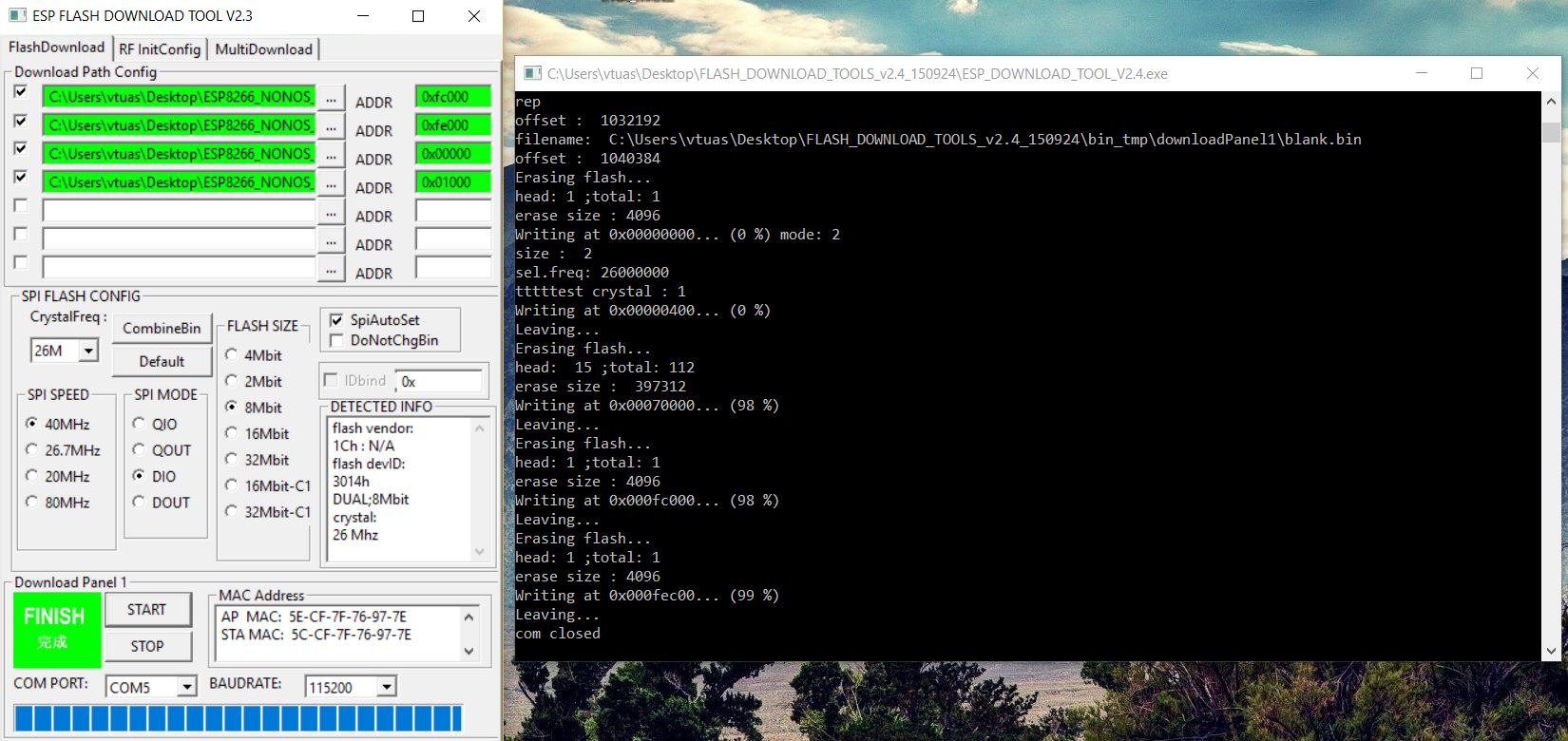
****

After flashing the wifi module, I attempted to communicate with the module via Termite and but ran into an error after typing AT commands. In the end I could not figure out why it kept giving me this error. I tried changing the baud rate to a value other than 115200 but it only made the problem worse. I also tried rewiring my circuit but the same error was encountered again.

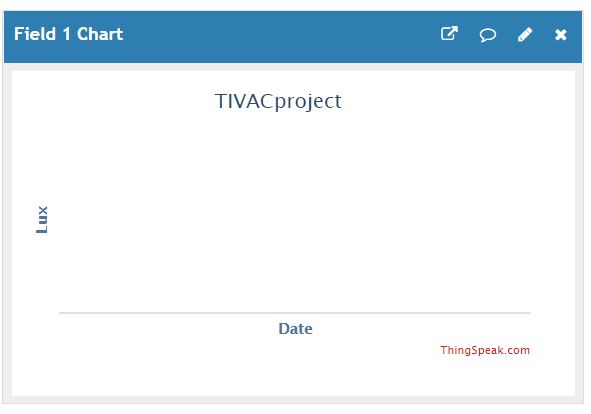




I also later tried flashing the wifi module with a different set of firmware found on one of the tutorials linked on the lecture notes which is shown below.

****

The image below is my graph from the ThingSpeak server.

****

**5. Conclusions**

The main problem I had with this project was the fact that the flash tool would state that the ESP8266 chip had been successfully updated with the latest AT firmware (as seen in the screenshot section) but every time I try to communicate with the ESP8266 chip through the FTDI chip using AT commands, I would only get an error message. I have tried fixing this by changing my baud rate settings to each available setting, but the terminal only outputs error messages. As a last resort, I also tried using the other versions of firmware given to us by Dr. Venki but the result was still the same. I am pretty sure the code in section 2 (Detailed Information) works just fine as the API key is updated in the code, as well as the baud rate configuration. As far as completion, I was able complete the code, screenshots, the detailed documentation for tasks, and schematics.