**TITLE:** Lux Sensor (TSL2591) Data Collector

**GOAL**

* Build a schematic involving the TIVAC TM4C123G launchpad, TSL2591 lux sensor, and ESP8266 wifi module
* Read data from the lux sensor at intervals of 15 - 60 seconds over the course of an hour to 24 hours
* Upload the data to the cloud, thingspeak was used for this project, using the ESP2866 wifi module
* Finally use the UART interface for the ESP2866 module and the I2C interface for the TSL sensor

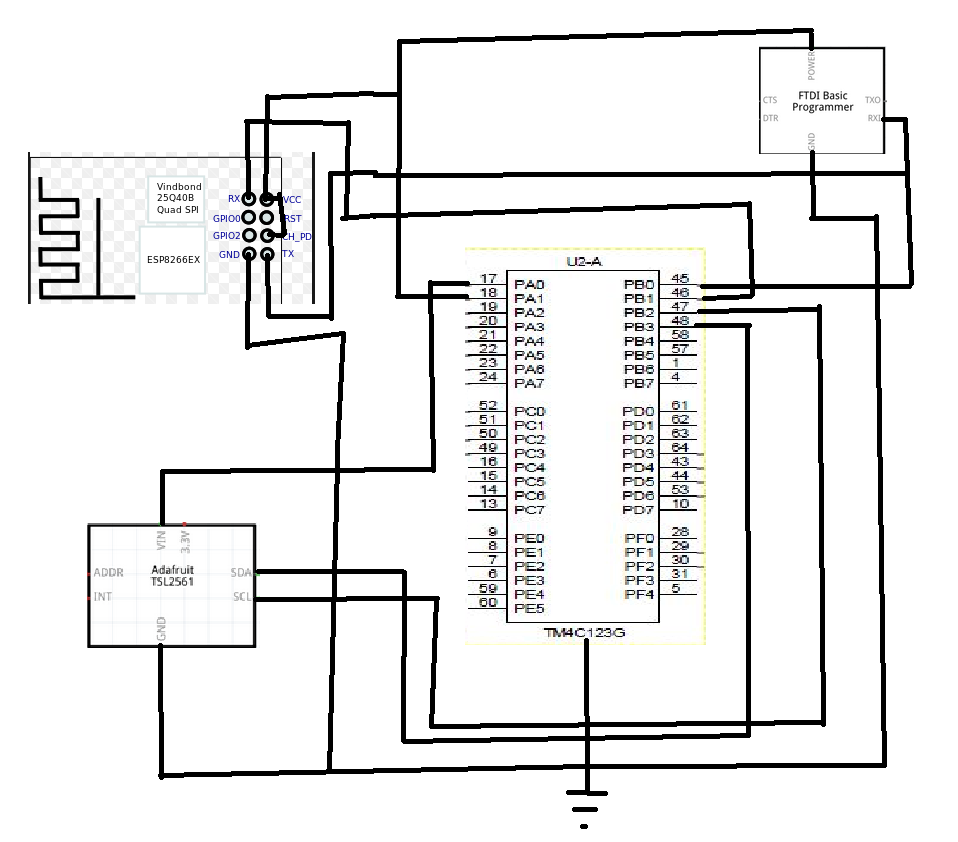
**DELIVERABLES:**

The intended project deliverable was to send lux data to a thingspeak server where it could be interpreted. We did this by using the ESP2866 wifi module to connect to the thingspeak server. The data was also sent over the UART interface where could be read right away and was used for debugging purposes.

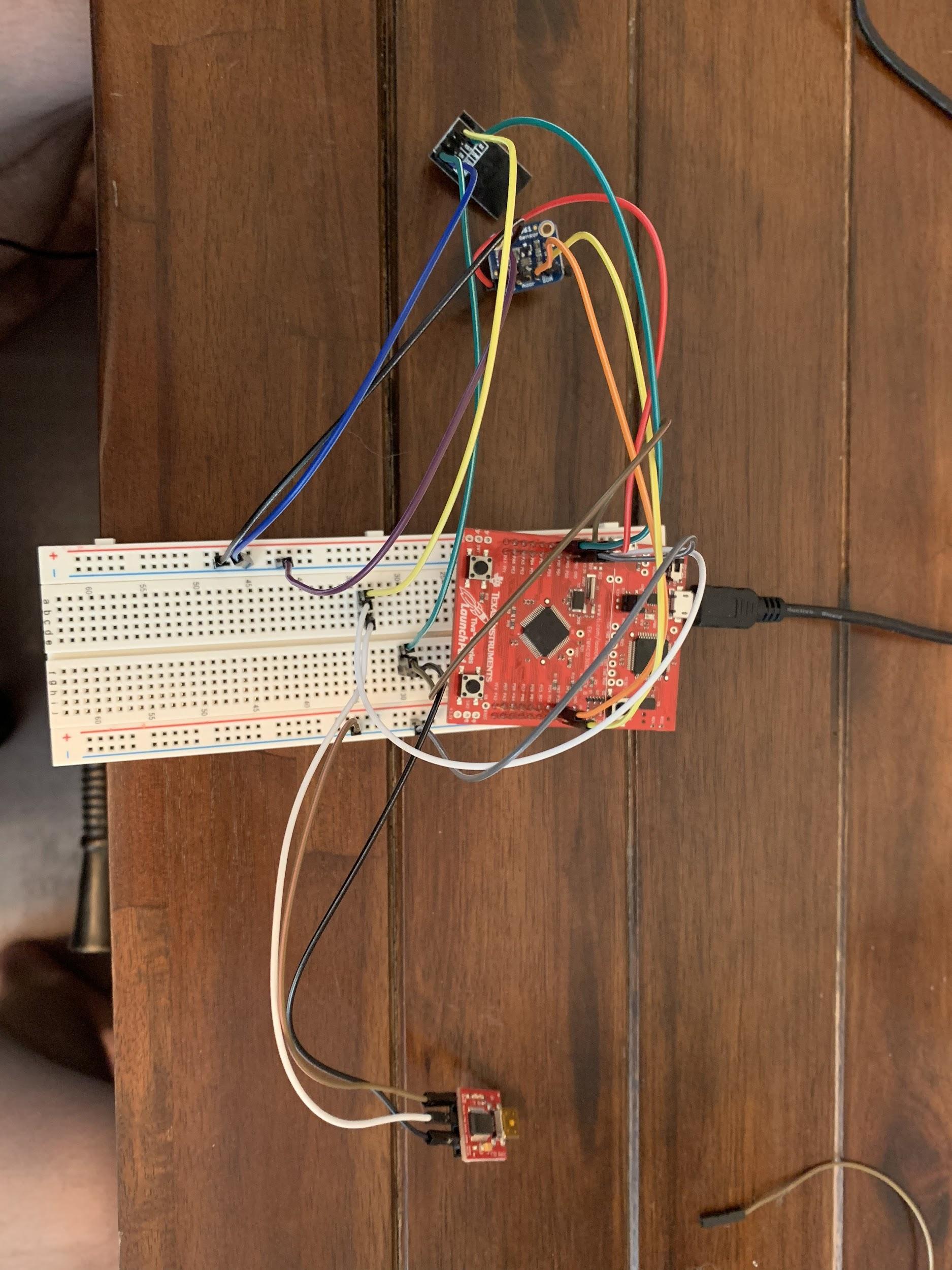
**COMPONENTS:**

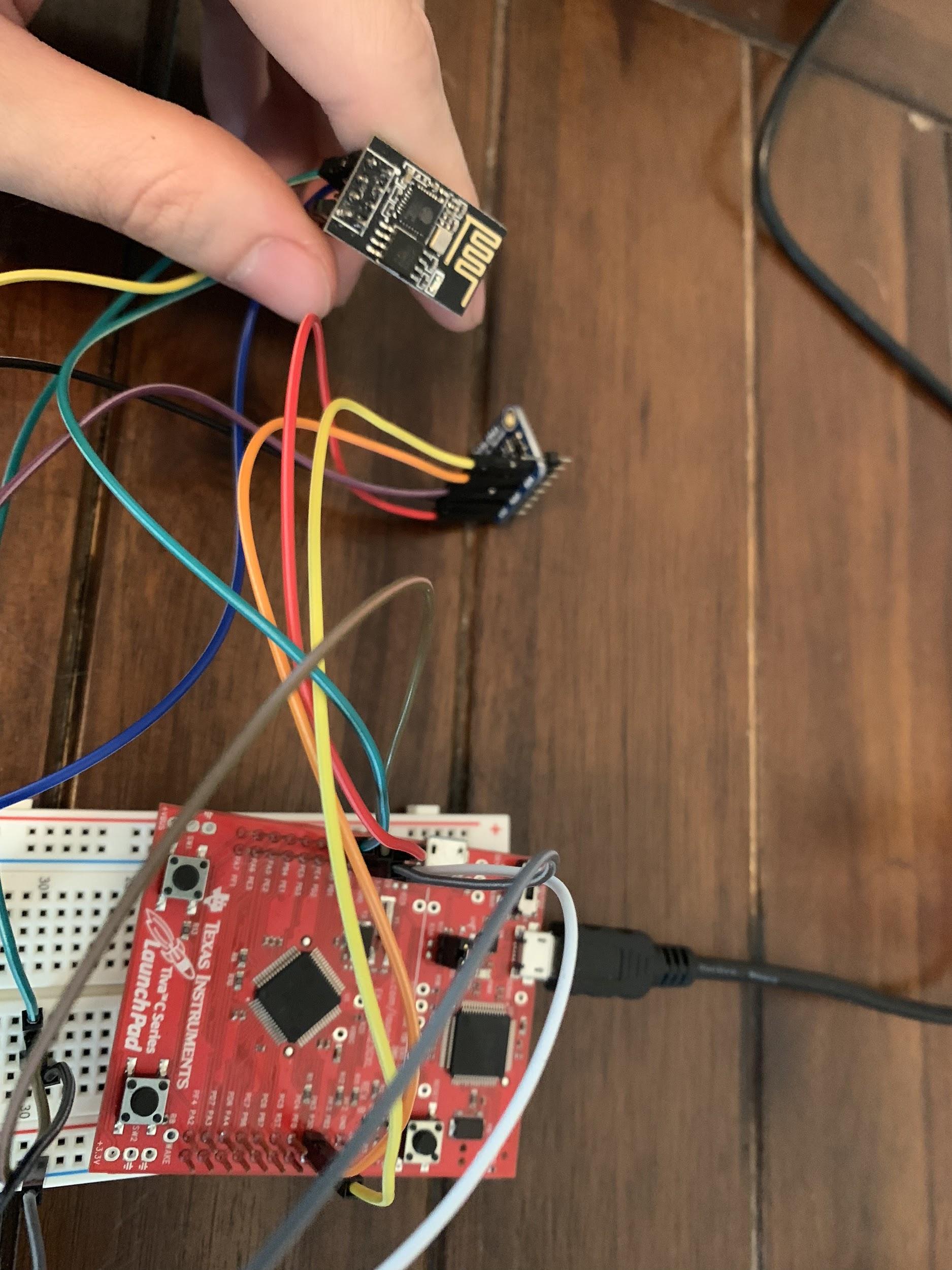
The main components of this project were the TIVAC TM4C123G launchpad which was connected to the TSL2591 lux sensor and the ESP2866 wifi module. The lux sensor utilized the I2C interface while the ESP2866 utilized the UART interface. The FTDI was used for debugging purposes. The mini IoT project sent the data to the cloud to a thingspeak server where a graph was formed. In the beginning of the program, I have a function that configures and initialization the UART interface, which enables UART module 1, enables GPIO port b, configures PB1 for TX and PB0 for RX, finally it will set the UART pin type, clock source, and enable the baud rate that UART uses. After that the program will initialize the I2C interface. Here I enable the I2C0 and PORTB, set the I2C PB3 as SDA and PB2 as SDA. I set the clock of the I2C which ensures a proper connection, finally there is a while loop to wait while the master SDA is busy. Last component to be initialized is the TSL2591 sensor, here it reads the device’s ID and configures for medium gain and integration time of 100 ms.

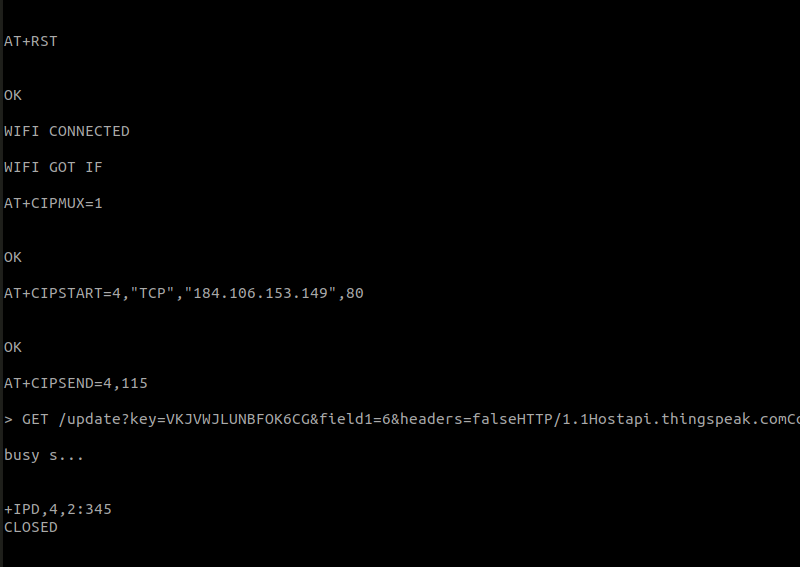
**SCHEMATICS:**



**SCREENSHOTS:**







**IMPLEMENTATION:**

1. Configure UART for TIVAC, initialize I2C0 and the TSL2591
2. Enable button 2 for hibernation, setup hibernate clock, enable retention during hibernation, set and enable the RTC
3. Hibernation for 30 mins, wake up if button 2 pressed.
4. Get luminosity for 20 cycles then calculate the average
5. Display the average to the UART interface
6. Reset the ESP2866, enable multiple sends
7. Establish a connection with the thingspeak server
8. Send the data to the server through UART
9. Allow the ESP2866 to send the information to the given HTTP\_POST
10. Hibernate

**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 "TSL2591\_def.h" #include "utils/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, 19200, 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   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=VKJVWJLUNBFOK6CG&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);   HibernateRequest (); //Hibernate  while (1)  {}; } |

**CONCLUSION:**

In conclusion the coding aspect of the project wasn’t too hard as it was mainly provided to us. Using the I2C interface with the lux sensor was relatively easy to follow. The hardest part was having all the components talk to each other in one big IoT design. I had a lot of hiccups with trying to get my data to my thingspeak server but I also had a lot trouble when I had to do that functionality in CPE 300 also. All together I learned how to read data and transfer it to a server.