TITLE: Light Monitoring

GOAL:

- To learn how to capture raw data from LUX sensor (TSL2591) using I2C
- To learn how calculate Lux and convert data so it can be displayed in a visual manner (string)
- To learn how to make device wireless by using a Wi-Fi with UART interface and upload the data to a server (thingSpeak.com)

DELIVERABLES:

The purpose of this project was to capture raw data of from a TSL2591 and send the data to the Tiva C using I2C. The data is then calculated to find the lux (Light Intensity) and converted into a printable string. Finally, the string is sent to a ESP8266 Wi-Fi-module by using UART interface and is pushed to the server (thingSpeak.com). I was able to read the device ID from the sensor and read data, however, the data seems to be incorrect. Either the data is being read from the wrong registers or my sensor is damaged. However, I was able to successfully to push the data to thingSpeak.com by interfacing the ESP8266 with Tiva C.

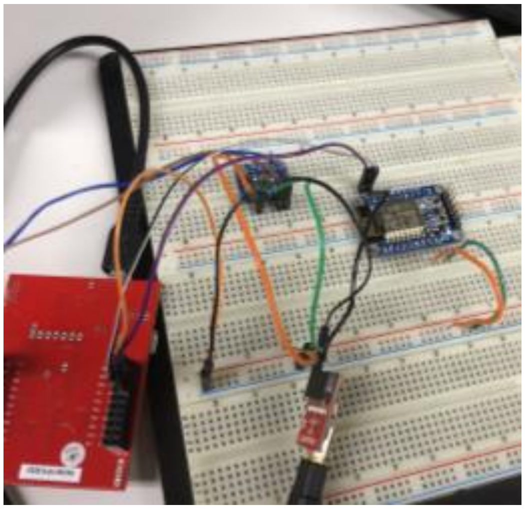
COMPONENTS:

Tiva TM4C123GH6PM- A MCU that controls the entire project. This device needed to be initialized by setting the system clock, enabling GPIOB module, enabling the I2C0 module, enabling and setting the clock for the I2C0 master module, enabling UART1 and configuring port B pins PB0-PB3.

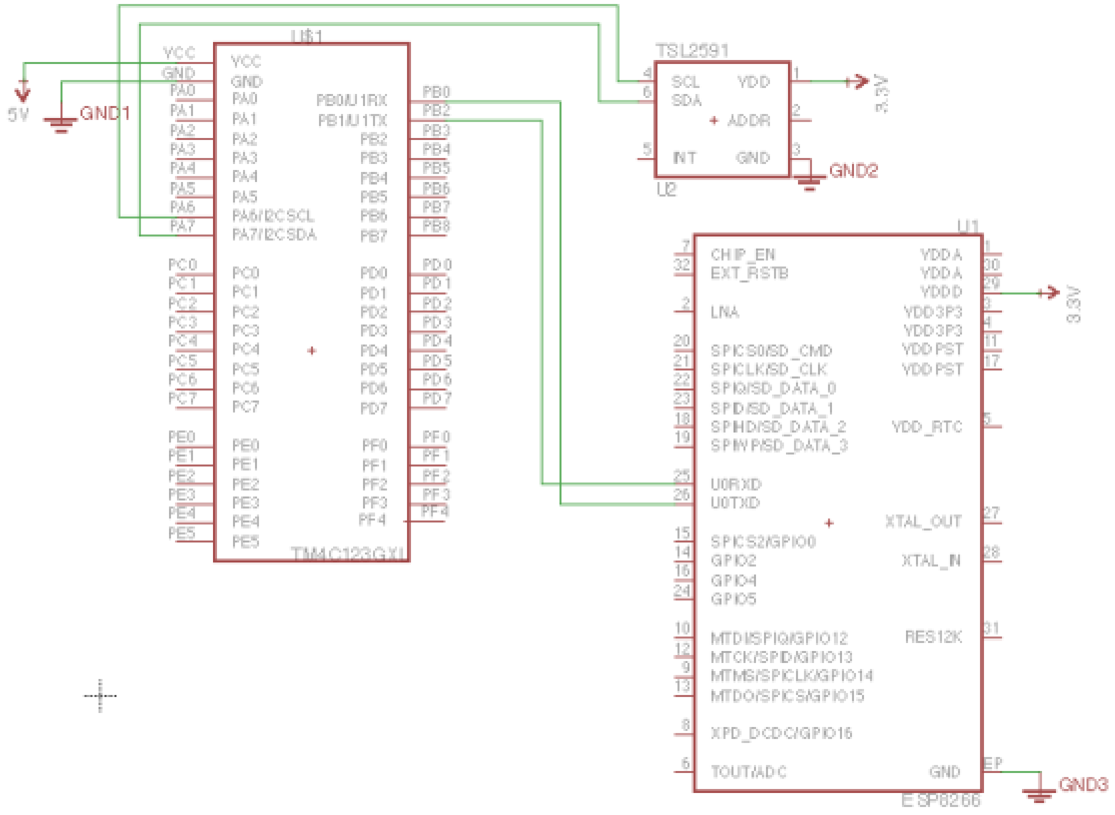
TSL2591-A high dynamic digital light sensor capable of measuring Lux up to ranges 188u – 88,000 Lux. It is interfaced with the Tiva C by using I2C. It was initialized by trying to read the Device ID register, setting the gain, setting the timing, and setting the Power On register to 1.

ESP8266- A Wi-fi module that is interface with Tiva C using UART. This device was initialized by sending AT commands to the device to set CWMODE=1, CIPMUX = 1, Wi-Fi access point, and TCP connection to the ThingSpeak server (184.106.153.149).

FTDI- 5V FTDI was used for debugging. It received the AT commands sent through UART and was displayed on the terminal (PUTTY) on the PC.



SCHEMATICS:


U$1
VCC
GND
PA0
PA1
PA2
PA3
PA4
PA5
PA6/I2CSCL
PA7/I2CSDA
PB0/U1RX
PB1/U1TX
PB2
PB3
PB4
PB5
PB6
PB7
PB8
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
PD0
PD1
PD2
PD3
PD4
PD5
PD6
PD7
PE0
PE1
PE2
PE3
PE4
PE5
PF0
PF1
PF2
PF3
PF4
TM4C123GXL
5V
GND1
TSL2591
SCL
SDA
INT
VDD
ADDR
GND
U2
3.3V
GND2
U1
CHIP_EN
EXT_RSTB
LNA
SPICS0/SD_CMD
SPICLK/SD_CLK
SPIQ/SD_DATA_0
SPID/SD_DATA_1
SPIHD/SD_DATA_2
SPIWP/SD_DATA_3
U0RXD
U0TXD
SPICS2/GPIO0
GPIO2
GPIO4
GPIO5
MTDI/SPIQ/GPIO12
MTCK/SPID/GPIO13
MTMS/SPICLK/GPIO14
MTDO/SPICS/GPIO15
XPD_DCDC/GPIO16
TOUT/ADC
VDDA
VDDA
VDDD
VDD3P3
VDD3P3
VDDPST
VDDPST
VDD_RTC
XTAL_OUT
XTAL_IN
RES12K
GND
EP
3.3V
GND3
ESP8266

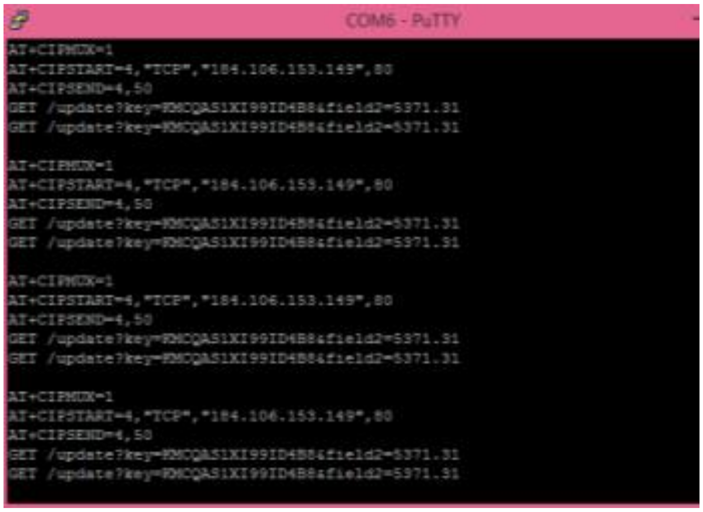
IMPLEMENTATION:

1. Initialize Console (UART):
   In this subroutine, the GPIOB is enabled and port pins, PB0 and PB1, are configured to be used for U1Rx and U1TX. The UART1 clock is set to use the precision internal oscillator and the baud rate is set to 115200.

2. Initialize I2C:
   In this subroutine, the I2C0 module is enabled and the port pins PB2 and PB3 are set to be used as SCL and SDA which will communicate with the identical pins on the TSL2591 sensor. The I2C0 master module is also enabled and the its clock is set to use the system clock and sets the data rate to 400kbps.

3. Get Luminosity
   In this subroutine, the TSL2591 is initialized and reads raw data from Ch0 and Ch1. These two values are then calculated to find the Lux value of the TSL2591 sensor. The result of this calculation was returned to the main.

4. Ftoa
   This subroutine converts a float into a printable character array which is then pass back to main to be printed using UARTprintf().

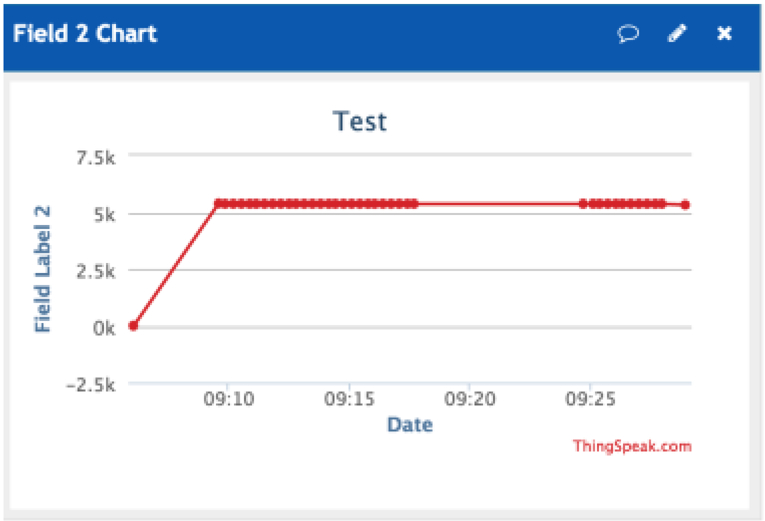


```
AT+CIPMUX=1
AT+CIPSTART=4,"TCP","184.106.153.149",80
AT+CIPSEND=4,50
GET /update?key=KMCQAS1XI99ID4B8&field2=5371.31
GET /update?key=KMCQAS1XI99ID4B8&field2=5371.31

AT+CIPMUX=1
AT+CIPSTART=4,"TCP","184.106.153.149",80
AT+CIPSEND=4,50
GET /update?key=KMCQAS1XI99ID4B8&field2=5371.31
GET /update?key=KMCQAS1XI99ID4B8&field2=5371.31

AT+CIPMUX=1
AT+CIPSTART=4,"TCP","184.106.153.149",80
AT+CIPSEND=4,50
GET /update?key=KMCQAS1XI99ID4B8&field2=5371.31
GET /update?key=KMCQAS1XI99ID4B8&field2=5371.31

AT+CIPMUX=1
AT+CIPSTART=4,"TCP","184.106.153.149",80
AT+CIPSEND=4,50
GET /update?key=KMCQAS1XI99ID4B8&field2=5371.31
GET /update?key=KMCQAS1XI99ID4B8&field2=5371.31
```



CODE:

```
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_i2c.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/i2c.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/rom_map.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "utils/uartstdio.c"



uint32_t ui32SysClock;

#define TSL2591_VISIBLE          (2)      // channel 0 - channel 1
#define TSL2591_INFRARED         (1)      // channel 1
#define TSL2591_FULLSPECTRUM     (0)      // channel 0

#define TSL2591_ADDR             (0x29)//address register

#define TSL2591_COMMAND_BIT      (0xA0)    // 1010 0000: bits 7 and 5 for 'command
normal'
#define TSL2591_WORD_BIT         (0x20)    // 1 = read/write word (rather than byte)
#define TSL2591_BLOCK_BIT        (0x10)    // 1 = using block read/write

#define TSL2591_ENABLE_POWERON   (0x01)    //This register turns on the TSL2591
#define TSL2591_ENABLE_AEN       (0x02)    // ALS Enable. This field activates ALS function.
Writing a one activates the ALS. Writing a zero disables the ALS.
#define TSL2591_ENABLE_AIEN      (0x10)    // ALS Interrupt Enable. When asserted permits
ALS interrupts to be generated, subject to the persist filter.
#define TSL2591_ENABLE_NPIEN     (0x80)    // No Persist Interrupt Enable. When asserted
NP Threshold conditions will generate an interrupt, bypassing the persist filter

#define TSL2591_LUX_DF           (408.0F)
#define TSL2591_LUX_COEFB        (1.64F)  // CH0 coefficient
#define TSL2591_LUX_COEFC        (0.59F)  // CH1 coefficient A
#define TSL2591_LUX_COEFD        (0.86F)  // CH2 coefficient B
#define  TSL2591_REGISTER_ENABLE        (0x00)      //enable register
#define  TSL2591_REGISTER_CONTROL       (0x01)      //control bit register
#define  TSL2591_REGISTER_DEVICE_ID     (0x12)      //device id register
#define  TSL2591_REGISTER_CHAN0_LOW      (0x14) //ch0 low byte register
#define  TSL2591_REGISTER_CHAN0_HIGH     (0x15) //ch0 high byte register
```

```
#define  TSL2591_REGISTER_CHAN1_LOW       (0x16) //ch1 low byte register
#define  TSL2591_REGISTER_CHAN1_HIGH      (0x17) //ch1 high byte register


#define  TSL2591_INTEGRATIONTIME_100MS    (0x00) //time 100ms
#define  TSL2591_GAIN_MED             (0x10)    // medium gain (25x)




const bool DebuggingMode = true;



void InitConsole(void)
{

        //enable GPIO module
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    //configure PB0  as U1Rx
    GPIOPinConfigure(GPIO_PB0_U1RX);
    //configure PB1  as U1Tx
    GPIOPinConfigure(GPIO_PB1_U1TX);
    //enable UART1
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
    //SET UART1 to use the precision internal oscillator
    UARTClockSourceSet(UART1_BASE, UART_CLOCK_PIOSC);
    //set PB0 and PB1 function to UART
    GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    //set baudrate to 115200
    UARTStdioConfig(1, 115200, 16000000);

}


void i2c0_init()
{

        //enable I2C0 module
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

    //configure pin muxing for I2C0 functions on Port B3
    MAP_GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);
    MAP_GPIOPinConfigure(GPIO_PB3_I2C0SDA);

    //Configure the pin muxing for I2C) functions on Port B2
    MAP_GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
    MAP_GPIOPinConfigure(GPIO_PB2_I2C0SCL);

    // Enable and initialize the I2C0 master module.  Use the system clock for
    // the I2C0 module.  The last parameter sets the I2C data transfer rate.
    // If false the data rate is set to 100kbps and if true the data rate will
    // be set to 400kbps
    I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), false);
    //wait for MCU to finish transaction
         while (I2CMasterBusy(I2C0_BASE));


}
uint8_t i2c0_read(uint8_t addr, uint16_t reg)
{
       uint8_t  x;     //unsigned 8-bit variable that will store the data fro the sensor

         //specify that we want to communicate to device addr with an intended write to bus
       I2CMasterSlaveAddrSet(I2C0_BASE, addr, false);
       //specify reg to be read
       I2CMasterDataPut(I2C0_BASE, reg);
       //send control byte and reg addr byte to slave device
       I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);
       //wait for MCU & device to complete transaction
         while (I2CMasterBusy(I2C0_BASE));
       //read from the specified slave device
       I2CMasterSlaveAddrSet(I2C0_BASE, addr, true);
       //send control byte and read from the specified register
       I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);\
       //wait for MCU & device to complete transaction
       while (I2CMasterBusy(I2C0_BASE));
       //get data from sepcified register
       x = I2CMasterDataGet(I2C0_BASE);
       //wait for MCU & device to complete transaction
       while (I2CMasterBusy(I2C0_BASE));

       return x;
}

void i2c0_write(uint8_t addr, uint16_t reg, uint8_t data)
{

         //specify that we want to communicate to device addr with an intended write to bus
         I2CMasterSlaveAddrSet(I2C0_BASE, addr, false);
         //register to be read
         I2CMasterDataPut(I2C0_BASE, reg);
         //send control byte and reg addr byte to slave device
         I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
         //wait for MCU & device to complete transaction
         while (I2CMasterBusy(I2C0_BASE));
         I2CMasterSlaveAddrSet(I2C0_BASE, addr, true);
         //specify data to be written to the above mentioned reg
         I2CMasterDataPut(I2C0_BASE, data);
         //wait while checking for MCU to complete the transaction
         I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH);
```

```
        //wait for MCU & device to complete transaction
        while (I2CMasterBusy(I2C0_BASE));
}

void TSL2591_init()
{
        //i2c0_write(TSL2591_ADDR, TSL2591_ADDR, 0x00);
    i2c0_read(TSL2591_ADDR, TSL2591_COMMAND_BIT|TSL2591_REGISTER_DEVICE_ID);
//read device ID

    // Set Gain and Timing
    i2c0_write(TSL2591_ADDR,TSL2591_COMMAND_BIT|TSL2591_REGISTER_CONTROL ,
TSL2591_INTEGRATIONTIME_100MS|TSL2591_GAIN_MED );
    //Power on the sensor
    i2c0_write(TSL2591_ADDR,TSL2591_COMMAND_BIT|TSL2591_REGISTER_ENABLE,
TSL2591_ENABLE_POWERON| TSL2591_ENABLE_AEN | TSL2591_ENABLE_AIEN |
TSL2591_ENABLE_NPIEN);


}

float getLuminosity ()
{
        float    atime = 100.0F, again=25.0F; //For 100ms integration time and med gain
        float    cpl, lux1, lux2, lux;
        uint16_t ch0 = 0;
        uint16_t ch1 = 0;

        // Get full luminosity
        //read the high byte of channel 1
        ch1 = i2c0_read(TSL2591_ADDR, TSL2591_COMMAND_BIT |
TSL2591_REGISTER_CHAN1_HIGH);
        ch1 <<= 8;
        //read the low byte of channel 1
        ch1 |= i2c0_read(TSL2591_ADDR, TSL2591_COMMAND_BIT |
TSL2591_REGISTER_CHAN1_LOW);

        //read the low byte of channel 0
        ch0 = i2c0_read(TSL2591_ADDR,TSL2591_COMMAND_BIT |
TSL2591_REGISTER_CHAN0_LOW);
        ch0 <<= 8;
        //read the high byte of channe0
        ch0 |= i2c0_read(TSL2591_ADDR,TSL2591_COMMAND_BIT |
TSL2591_REGISTER_CHAN0_HIGH);


        //Check for overflow conditions first
        if((ch0 == 0xFFFF) |(ch1 == 0xFFFF) )
        {
                UARTprintf("\n Overflow");
                return 0;
        }
        //Calculate Lux value from sensor
        cpl = (atime * again) / TSL2591_LUX_DF;          // cpl = (ATIME * AGAIN) / DF
        lux1 = ( (float)ch0 - (TSL2591_LUX_COEFB * (float)ch1) ) / cpl;
        lux2 = ( ( TSL2591_LUX_COEFC * (float)ch0 ) - ( TSL2591_LUX_COEFD * (float)ch1 ) ) /
cpl;
        // The highest value is the approximate lux equivalent
        lux = lux1 > lux2 ? lux1 : lux2;
        return lux;
}

void ftoa(float f,char *buf)
{
      /*Function acquired from forum:
       * http://e2e.ti.com/support/microcontrollers/stellaris_arm/f/471/p/44193/156824.aspx
       */
   int pos=0,ix,dp,num;
   if (f<0)
   {
      buf[pos++]='-';
      f = -f;
   }
   dp=0;
   while (f>=10.0)
   {
      f=f/10.0;
      dp++;
   }
   for (ix=1;ix<8;ix++)
   {
         num = (int)f;
         f=f-num;
         if (num>9)
            buf[pos++]='#';
         else
            buf[pos++]='0'+num;
         if (dp==0) buf[pos++]='.';
         f=f*10.0;
         dp--;
   }
}

int main(void)
{
        //52 bit string of AT command to send data to thingkSpeak, 42 instr, 6 are lux. (append \n\r
at end)
        char Lux[48] = "GET /update?key=KMCQAS1XI99ID4B8&field2=";
        float lux_read;     //float variabl that will store the lux value
        //set system clock frequency
        ui32SysClock = MAP_SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ
|SYSCTL_OSC_MAIN | SYSCTL_USE_PLL |SYSCTL_CFG_VCO_480), 120000000);
```

```
	//initialize UART
	InitConsole();
	//initialize i2C0
	i2c0_init();
	//initialize sensor
	TSL2591_init();

	while(1)
	{

		SysCtlDelay(5000000);
		//get luminosity from the sensor
		lux_read = getLuminosity();
		//convert luminosity from a float to a string
		ftoa(lux_read, &Lux[40]);
		Lux[47]='\0';

		UARTprintf("\n");
		SysCtlDelay(20000000);
		//set CIPMUX = 1
		UARTprintf("AT+CIPMUX=1\n\r");
		SysCtlDelay(20000000);
		//connect to thingSpeak server
		UARTprintf("AT+CIPSTART=4,\"TCP\",\"184.106.153.149\",80\n\r");
		SysCtlDelay(20000000);
		//set channel id and size of data
		UARTprintf("AT+CIPSEND=4,50\n\r");
		SysCtlDelay(20000000);
		//send data to thingSpeak
		UARTprintf("%s\n\r",&Lux);
		SysCtlDelay(20000000);
		//send data to thingSpeak
		UARTprintf("%s\n\r",&Lux);


	}
}
```