CPE 403 Fall 2018
Brian Lopez

Midterm Project 1

Goal:

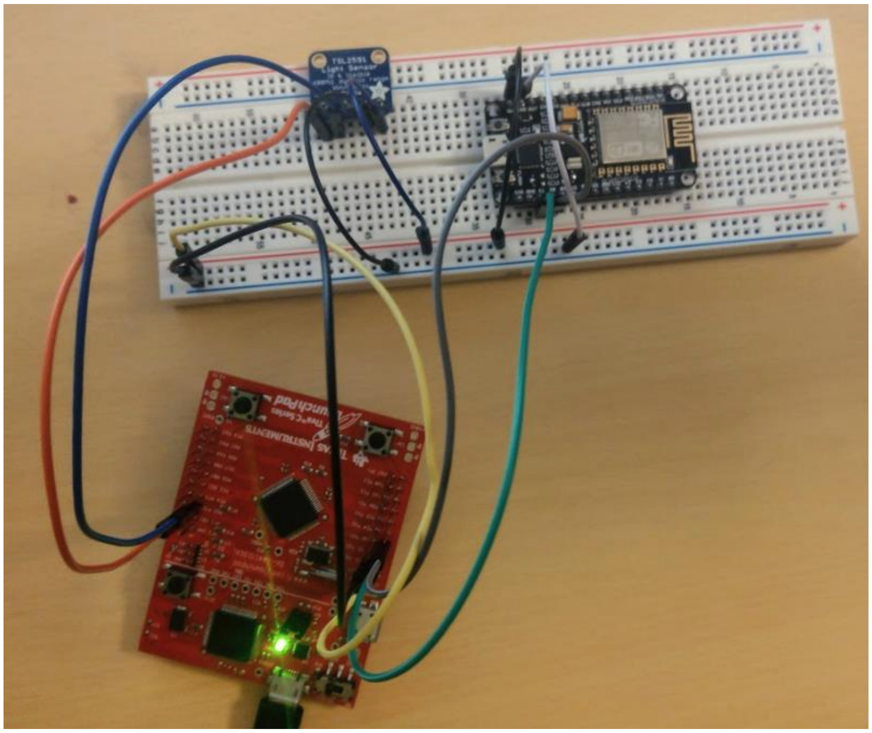
- Interface with the TSl2561 Lux sensor with the Tivac microcontroller
- Send data from Lux sensor to an ESP8266 that will upload the data to a ThingSpeak server

Deliverables:

- I was able to gather data from the TSL2561 and then send it to ThingSpeak every ~15 seconds.

Components:

- TivaC TM4C123G: The main microcontroller that will connect both the TSL2561 lux sensor and the ESP8266 together. It will run at 40MHz and interface with the TSL2561 using I2C and send the data to the ESP through UART.
- TSL2561: An I2C sensor that calculates the luminosity that is hitting it.
- NodeMCU: The NodeMCU is a version of the ESP8266 that is capable of running entirely on its own due to having its own processor. However, in this project it is flashed with AT firmware and can only be used through UART using AT commands.



Implementation:

- The first step is to simply configure the clock of the microcontroller. I chose to run it at 40MHz.
- After that UART is configured to be able to communicate with the ESP. Pins PB0 and PB1 were used for RX and TX respectively and a baud rate of 115200 was used.
- I2C is then initialized with PB2 and PB3 being used as SCLK and SDA. Once it is initialized, the TSL2561 sensor is initialized. It will first obtain the address of the sensor and check to see if it is the correct address, since the program won't work if the correct address cannot be obtained. The sensor is then configured with median gain and an integration time of 100ms, along with setting the power as well.
- The final component that will need to be configured is the timer. The timer is set up to trigger an interrupt every 15 seconds. A prescaler of 16 is used with a period of 37500000 (3750000 * 16 = 6000000 and 60000000 / 40M = 15 seconds).
- After that, nothing is down in main and the program will go into an infinite loop. Everything else will be done on the Timer interrupt handler, where it will get 10 lux values and get the average. After that it will prepare the AT commands to connect to the ThingSpeak, prepare to send the appropriate amount of characters, and then send the 'Get' command that will chart the value found. While the interrupt is done every 15 seconds, the delays while issuing the AT commands will make it send the data to the servers around every 23 seconds.

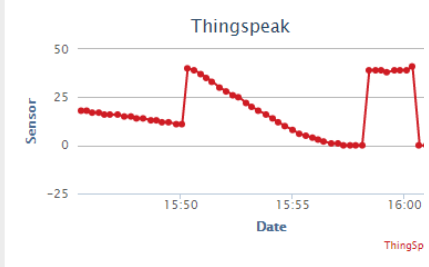
Video:

https://www.youtube.com/watch?v=aSPZz3yu3BM

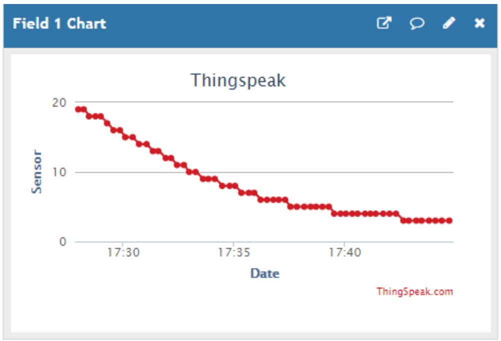
- Note: The code in the program is a slightly modified version of the one below. The one in the video displays the value from the lux sensor every second onto the console (UART0) and will send the data to ThingSpeak every 15 seconds. This is to easily show the lux sensor reacting to light.

ThingSpeak Images:

- Here is one image where the sensor is next to a window sill. The second spike was when I opened the blinds of the window, letting more light inside. I am not sure what caused the first spike however.



- This next screenshot shows a normal decrease in lux values in the span of around 20 minutes. This was around sunset:



Code:

```
#include <stdarg.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include <stdio.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 "uartstdio.h"
#include "driverlib/interrupt.h"
#include "Adafruit_TSL2591.h"
#include "driverlib/timer.h"

void timerInit(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
    // turn on timer 1 for 0.5 second intervals
    TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
    TimerLoadSet(TIMER1_BASE, TIMER_A, 37500000-1); // (40000000*15) / 16
    TimerPrescaleSet(TIMER1_BASE, TIMER_A, 16);      // prescalor of 16
    IntEnable(INT_TIMER1A);
    TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
    TimerEnable(TIMER1_BASE, TIMER_A); // turn on Timer
}

void ConfigureUART1(void)
{
    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, 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_REGISTER_DEVICE_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
    }
    // Configure with median gain and integration time of 100ms
    I2C0_Write(TSL2591_ADDR, 2, (TSL2591_COMMAND_BIT | TSL2591_REGISTER_CONTROL),
0x10);
    // enable power and interrupts for sensor
    I2C0_Write(TSL2591_ADDR, 2, (TSL2591_COMMAND_BIT | TSL2591_REGISTER_ENABLE),
               (TSL2591_ENABLE_POWERON | TSL2591_ENABLE_AEN | TSL2591_ENABLE_AIEN |
TSL2591_ENABLE_NPIEN));
}

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 Timer1IntHandler(void)
{
    uint32_t i, lux;
    lux = 0;
    for(i = 0; i < 10; i++)
        lux += GetLuminosity();  // gather 10 lux values

    lux = lux / 10;       // average the values received
    char getCmd[200] = "GET 
/update?key=DN72F6LKT2GVW8RD&field1=150&headers=falseHTTP/1.1\nHostapi.thingspeak.com
\nConnection:close\Accept*\*\r\n\r\n";

    UARTprintf ("AT+CIPMUX=1\r\n"); //enable multiple send ability
    SysCtlDelay (13333333);
    UARTprintf ("AT+CIPSTART=4,\"TCP\",\"184.106.153.149\",80\r\n");    //Establish a 
connection with the thingspeak servers
    SysCtlDelay (13333333);
    UARTprintf ("AT+CIPSEND=4,%d\r\n", strlen(getCmd));  //command the ESP8266 to 
allow sending of information
    SysCtlDelay(13333333);
    UARTprintf("GET 
/update?key=DN72F6LKT2GVW8RD&field1=%d&headers=falseHTTP/1.1\nHostapi.thingspeak.com\
nConnection:close\Accept*\*\r\n\r\n", lux);
    SysCtlDelay(10000);
    UARTprintf("AT+CIPCLOSE\r\n");
    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
}


int main(void)
{
    // set clock to 40MHz
    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);

    ConfigureUART1();

    I2C0_Init();  // initialize I2C, TSL sensor, and the timer
    TSL2591_init();
    timerInit();

    while(1)
    {

    }

    // return 0;
}
```