Enrique Saldana

CPE 403 ADV EMB SYS DES FALL 2018

Title: TIVAC MIDTERM

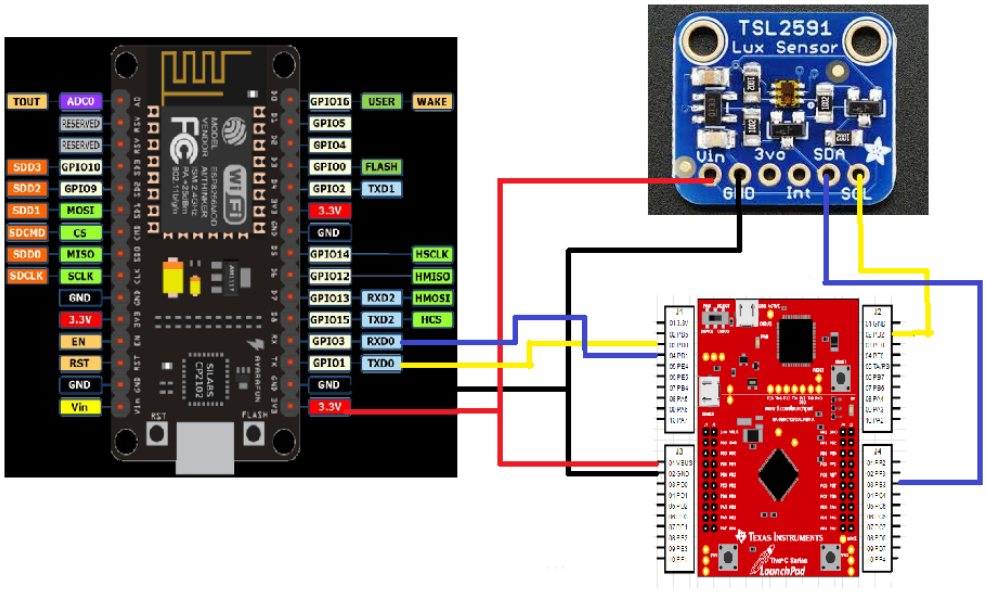
Goals:

- Read lux values using TSL 2591 Lux Sensor
- Communicate lux valuse to ESP using UART
- Send lux values to Thinkspeak Server using the ESP
- Hibernate program to send values every 45 seconds

Deliverables:

The project is intended to collect lux values for an extended period of time and write these values to the Thinkspeak server which will display these values in a graph.

Schematics:



Implementation:

- Initialize integer variables
- Initialize UART, I2C, and TSL2591(lux sensor)
- Using a 40MHz clock configure system to hibernate for 45 seconds
- Take 20 samples of the lux value and use the average as the sample to send throught UART
- Send commands throught UART in order to have the ESP upload data to Thinkspeak server
- Hibernate for 45 seconds and then repeat sending lux data

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, 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_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_MAI
N);	//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
	HibernateRTCEnable ();	//enable RTC hibernation
	HibernateRTCSet (0);	//Set RTC hibernation
	HibernateRTCMatchSet (0, HibernateRTCGet()+45);	//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=AJSGNO2JLU6Y93LZ&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)
	{};
}
```