Andrew Scott
E155
Lab 6
October 26, 2015

Lab 6: Internet of Things

Introduction

In this lab, I set up an Apache server on my Raspberry Pi and used the server to display a webpage which shows the voltage output from a light sensing circuit and also provides buttons to change the status of an LED on my MuddPi board.

Design and Testing Methodology

On the hardware side, the main design I had to do was designing my phototransistor circuit. I decided to place a resistor on the emitter side of the phototransistor, as well as reading from the emitter side, which would give a high voltage when light was detected and a low voltage when there was no light (this was just a decision I made because I liked the idea of having a higher voltage mean more light was detected and a lower voltage mean less light was detected). I tested the circuit by confirming that I got different voltage values when leaving the phototransistor exposed to the lab lights and when covering up the phototransistor to limit the light striking it.

On the software side, there were several decisions to be made. One of these was how to display the voltage in the HTML on the server that was running on the Pi. I decided to do this using an `<iframe>` tag, which displays an HTML within another HTML page. I was able to define the source of the embedded page to be a CGI script that accessed the ADC and printed out a simple html page that contained the result of accessing the ADC, which allowed me to keep the rest of the HTML the same. Another way I could have done this would have been to have a CGI script generate my entire HTML page, but I felt that the `<iframe>` approach was simpler and made my code cleaner. Another decision I had to make was how exactly to read the voltage in from the ADC. Reading through the MCP3002 documentation, I found a very useful and simple diagram showing exactly how to communicate with the MCP3002 through the SPI interface when using a microcontroller that communicates in 8 bit segments. I saw that the first 8 bits sent from the Pi had important information about how the ADC would send the data back and the second 8 bits sent from the Pi had no information in them. I chose the following settings:

X	1 (start bit)	SGL/DIFF	ODD/SIGN	MSBF	X	X	X
0	1	1	0	1	0	0	0

These settings chose channel 0 to be the input to the ADC and chose for the Pi to read back the most significant bit first. I did some simple bit manipulation with the two sets of 8 bits I read back from the Pi to get the 10 bits of information about the voltage the ADC read, and then converted these 10 bits to a voltage using the MCP3002's formula: $10\ bits = \frac{1024 * V_{IN}}{V_{DD}}$ I tested that I was properly reading voltages by connecting the channel 0 ADC input to Vin (5V), and when I tried loading my page it read approximately 5V. I also connect the ADC input to ground and when I loaded my page it read 0V.

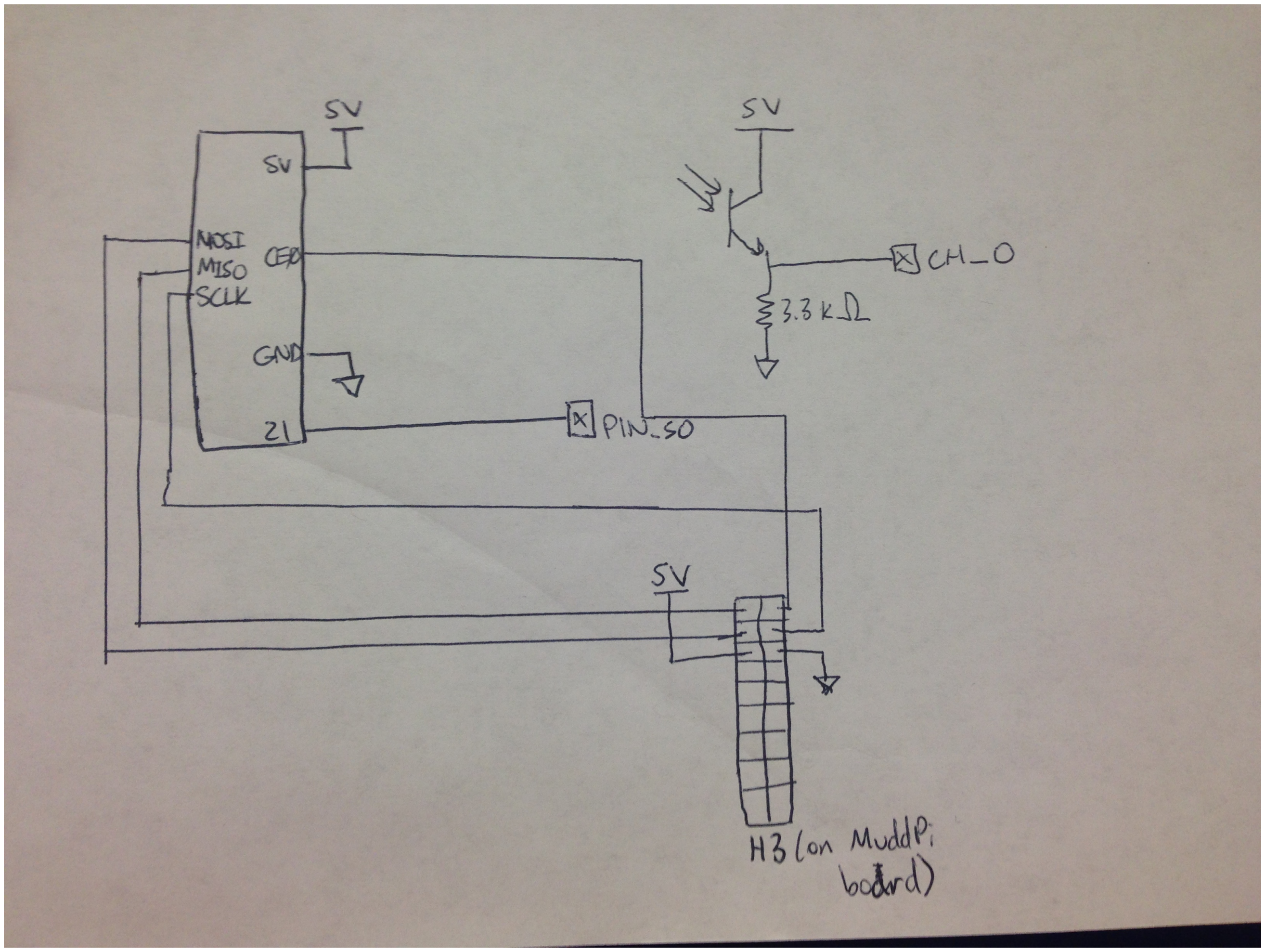
Results and Discussion

I was able to complete the entire lab, and as far as I can tell everything is working properly.

Conclusion

I started working on the lab during fall break before some of the bug fixes were posted, so I spent a fair amount of time figuring out what was wrong with some of the given code (for example, the submit forms having /cgi-bin/PROGRAM as opposed to just PROGRAM). In total I spent about 10 hours on this lab.

Breadboard Schematic



C Code
piHelpers.h

```
#include <sys/mman.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

////////////////////////////////////////////////////////////////////////
// Constants
////////////////////////////////////////////////////////////////////////

#define LEDPIN 21

// GPIO FSEL Types
#define INPUT  0
#define OUTPUT 1
#define ALT0   4
#define ALT1   5
#define ALT2   6
#define ALT3   7
#define ALT4   3
#define ALT5   2

#define GPFSEL   ((volatile unsigned int *) (gpio + 0))
```

```
#define GPSET    ((volatile unsigned int *) (gpio + 7))
#define GPCLR    ((volatile unsigned int *) (gpio + 10))
#define GPLEV    ((volatile unsigned int *) (gpio + 13))
#define INPUT  0
#define OUTPUT 1

// Physical addresses
#define BCM2836_PERI_BASE       0x3F000000
#define GPIO_BASE               (BCM2836_PERI_BASE + 0x200000)
#define BLOCK_SIZE (4*1024)
#define SYS_TIMER_BASE          (BCM2836_PERI_BASE + 0x3000)
#define SPIO_BASE               (BCM2836_PERI_BASE + 0x204000)

// Pointer that will be memory mapped when pioInit() is called
volatile unsigned int *gpio; //pointer to base of gpio

// Pointer that will be memory mapped when pTimerInit() is called
volatile unsigned int *sys_timer; //pointer to base of sys_timer

// Pointer that will be memory mapped when spiInit() is called
volatile unsigned int *spi0; //pointer to base of spio

//////////////////////////////////////////////////////////////////////
// Rasperry Pi Helper Functions
//////////////////////////////////////////////////////////////////////

void pioInit() {
    int  mem_fd;
    void *reg_map;

    // /dev/mem is a psuedo-driver for accessing memory in the Linux filesystem
    if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0) {
    printf("can't open /dev/mem \n");
    exit(-1);
    }

    reg_map = mmap(
         NULL,                  //Address at which to start local mapping (null means don't-care)
         BLOCK_SIZE,            //Size of mapped memory block
         PROT_READ|PROT_WRITE, // Enable both reading and writing to the mapped memory
         MAP_SHARED,            // This program does not have exclusive access to this memory
         mem_fd,                // Map to /dev/mem
         GPIO_BASE);            // Offset to GPIO peripheral

    if (reg_map == MAP_FAILED) {
    printf("gpio mmap error %d\n", (int)reg_map);
    close(mem_fd);
    exit(-1);
    }

    gpio = (volatile unsigned *)reg_map;
}

void pTimerInit() {
    int  mem_fd;
    void *reg_map;

    // /dev/mem is a psuedo-driver for accessing memory in the Linux filesystem
    if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0) {
    printf("can't open /dev/mem \n");
    exit(-1);
    }

    reg_map = mmap(
         NULL,                  //Address at which to start local mapping (null means don't-care)
         BLOCK_SIZE,            //Size of mapped memory block
         PROT_READ|PROT_WRITE, // Enable both reading and writing to the mapped memory
         MAP_SHARED,            // This program does not have exclusive access to this memory
         mem_fd,                // Map to /dev/mem
```

```
          SYS_TIMER_BASE);      // Offset to SYS_TIMER peripheral

    if (reg_map == MAP_FAILED) {
    printf("sys_timer mmap error %d\n", (int)reg_map);
    close(mem_fd);
    exit(-1);
    }

    sys_timer = (volatile unsigned *)reg_map;
}

/*
Function to set the mode of a pin. Function is the new GPFSEL value for the
specified pin.
*/
void pinMode(int pin, int function)
{
    unsigned int offset, shift;
    if (pin > 53 || pin < 0) {
    printf("bad pin, got pin %d\n", pin);
    return;
    } else if (function > 7 || function < 0) {
    printf("bad function, got function %d\n", function);
    }
    offset = pin / 10;
    shift = (pin % 10) * 3;

    // AND gpio[offset] with all 1s and function at the proper location
    gpio[offset] &= ~((~function & 7) << shift);
    // OR gpio[offset] with all 0s and function at the proper location
    gpio[offset] |= function << shift;
}

/*
Function to write val to a pin. Val can be either 0 or 1.
*/
void digitalWrite(int pin, int val)
{
    unsigned int set, clr;
    if (pin > 53 || pin < 0) {
    printf("bad pin, got pin %d\n", pin);
    return;
    }

    if (val){
    set = pin < 32 ? 7 : 8;             // select the proper set address
    gpio[set] = 0x1 << (pin % 32);      // write to the set address
    } else {
    clr = pin < 32 ? 10 : 11;           // select the proper clear address
    gpio[clr] = 0x1 << (pin % 32);      // write to the clear address
    }
}

int digitalRead(int pin)
{
    int out;
    if (pin > 53 || pin < 0) {
    printf("bad pin, got pin %d\n", pin);
    return 0;
    }

    // read from the proper address (depends on the pin number)
    if (pin < 32) {
    out = (gpio[13] >> pin) & 1;
    } else {
    out = (gpio[14] >> (pin - 32)) & 1;
    }
    return out;
}
```

```
void sleepMicros(int micros)
{
    if (micros == 0) {
    return;
    }
    sys_timer[4] = sys_timer[1] + micros;    // C1 = CLO + micros
    sys_timer[0] = 0b0010;                   // clear M1
    while (!!(sys_timer[0] & 0b0010) == 0);  // wait for M1 to go high again
}

void sleepMillis(int millis)
{
    sleepMicros(1000 * millis);     // sleep 1000 microseconds for each millisecond
}

void spiInit(int freq, int settings)
{
    int  mem_fd;
    void *reg_map;

    // /dev/mem is a psuedo-driver for accessing memory in the Linux filesystem
    if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0) {
    printf("can't open /dev/mem \n");
    exit(-1);
    }

    reg_map = mmap(
         NULL,                  //Address at which to start local mapping (null means don't-care)
         BLOCK_SIZE,            //Size of mapped memory block
         PROT_READ|PROT_WRITE, // Enable both reading and writing to the mapped memory
         MAP_SHARED,            // This program does not have exclusive access to this memory
         mem_fd,                // Map to /dev/mem
         SPIO_BASE);      // Offset to SYS_TIMER peripheral

    if (reg_map == MAP_FAILED) {
    printf("sys_timer mmap error %d\n", (int)reg_map);
    close(mem_fd);
    exit(-1);
    }

    spi0 = (volatile unsigned *)reg_map;


    // set pins 8-11 to be used for spi0
    pinMode(8, ALT0);
    pinMode(9, ALT0);
    pinMode(10, ALT0);
    pinMode(11, ALT0);

    spi0[2] = 250000000 / freq;     // set clock rate
    spi0[0] = settings;             // set the settings
    spi0[0] |= 0x00000080;          // set Transfer Active bit
}

char spiSendReceive(char send)
{
    spi0[1] = send;                        // send the data
    while (!(spi0[0] & 0x00010000));     // wait for the tranfer to finish
    return spi0[1];                        // return the received data
}


double getVoltage()
{
    char one = spiSendReceive(0x68);  // format is: x, startBit, SGI/DIFF, ODD/SIGN, MSBF, x, x, x
    char two = spiSendReceive(0x00);  // all bits are x (don't care)

    int result = 0x00000000;
    result = (result | (one & 0x03)) << 8;  // move two most significant bits to the proper location
    result |= two;                          // put the lower eight bits in their location
    return (double)(result * 5) / 1024;     // calculate the voltage from MCP3002 formula
}
```

LOADVOLTAGE.c

```
#include "piHelpers.h"

int main()
{
  pioInit();
  spiInit(244000, 0);
  double voltage = getVoltage();
  printf("%s%c%c\n", "Content-Type:text/html;charset=iso-8859-1",13,10);
  printf("<html>\n<body>\n<h1>VOLTAGE %f</h>", voltage);
  if (voltage > 0.5) {
    printf("<img src=\"http://jesterjournals.com/wp-content/uploads/2015/04/stock-illustration-32884400-smil
  } else {
    printf("<img src=\"http://image.spreadshirtmedia.com/image-server/v1/designs/10599300,width=190,height=1
  }
  return 0;
}
```

LEDON.c

```
//Turn On LED

#include "piHelpers.h"
int main(void)
{

        // Turn on the LED
        pioInit();
        pinMode(LEDPIN, OUTPUT);
        digitalWrite(LEDPIN, 1);

        getVoltage();

        //HTML header
        printf("%s%c%c\n",
                "Content-Type:text/html;charset=iso-8859-1",13,10);

        // Redirect to LEDCON with no delay
        printf("<META HTTP-EQUIV=\"Refresh\" CONTENT=\"0;url=index.html\">");
        return 0;

}
```

LEDOFF.c

```
// Turn off LED

#include "piHelpers.h"
int main(void)
{

        //Turn off LED
        pioInit();
        pinMode(LEDPIN, OUTPUT);
        digitalWrite(LEDPIN, 0);

        //Print header and redirect
        printf("%s%c%c\n",
                "Content-Type:text/html;charset=iso-8859-1",13,10);
        printf("<META HTTP-EQUIV=\"Refresh\" CONTENT=\"0;url=/index.html\">");
        return 0;

}
```

lab6.sv

```
module lab6(input logic inLED,
            ouput logic outLED);
    assign outLED = inLED;
endmodule
```