Jordan Ditzler

G00967092

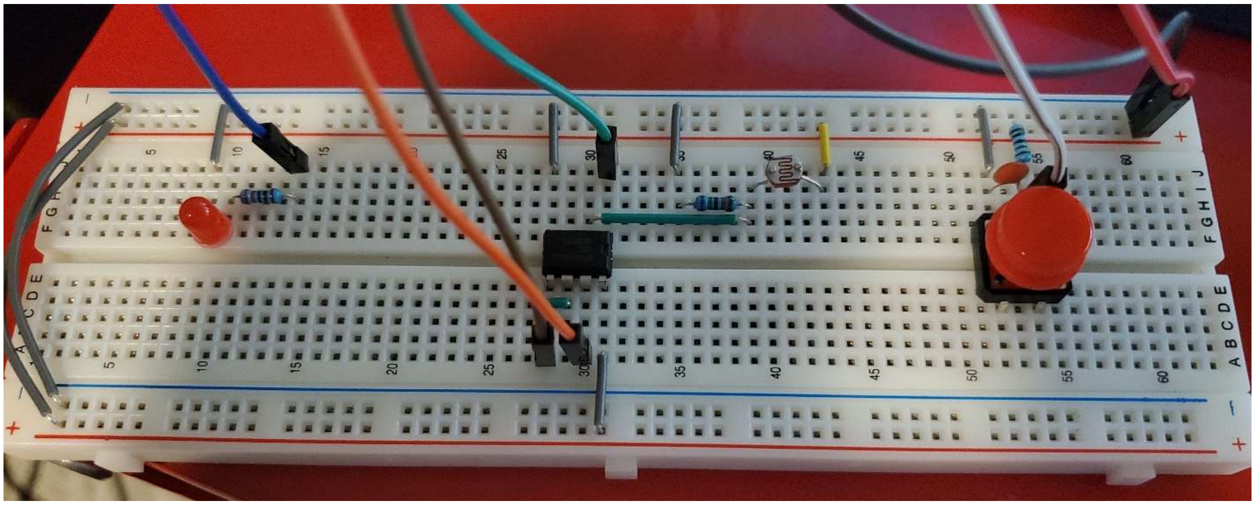
Lab 4 Report

October 19, 2020

Materials Used

- 1 220 Ω resistor
- 1 1 kΩ resistor
- 1 10 kΩ resistor
- 1 0.1 μF capacitor
- 1 red LED
- 1 push button
- 1 NSL-19M51 LDR
- 1 ADC0832CCN
- Jumper cables of varying lengths

Picture of Board



The button is connected to power via 1 kΩ resistor and has a 0.1 μF capacitor in series with it for help with debouncing. The debouncer circuit is not perfect, but paired with software debouncing, it eliminates most transients.

The red LED has a 220 Ω resistor in series with it to dim it. This is to make sure its brightness cannot alone bring the LDR above the threshold to turn it back off, thus causing an endless loop of LED state changes.

The LDR is in series with a 10 kΩ resistor.

Wiring (Left → Right):

> Blue: Connected to GPIO23, pin 16. Handles LED output.
> Brown: Connected to GPIO18, pin 12. Handles the ADC DI and DO pins.
> Orange: Connected to GPIO27, pin 13. Handles the ADC Clk pin.
> Green: Connected to GPIO17, pin 11. Handles the ADC CS pin.
> White: Connected to GPIO 22, pin 15. Handles button input.
> Black: Ground.
> Red: 5V power.

Questions

1) The lowest voltage measured across the 10 kΩ resistor was 0.002498 V with very little light. This means, since the LDR is connected to VCC, it is at its peak resistance. We can find its R_{max} by doing:

$$0.002498 \ = \frac{5 * 10k}{R_{max} + 10k}$$

$$R_{max} \approx 20{,}000{,}000 = 20\ M\Omega$$

The highest voltage measured across the 10 kΩ resistor was 5 V with a flashlight directly on top of the LDR. This is when the LDR would be at its lowest resistance, so finding R_{min} can be found with:

$$5 \ = \frac{5 * 10k}{R_{min} + 10k}$$

$$R_{min} \approx 0\ \Omega$$

The R_{max} fits into the LDR's datasheet and makes sense, even if it may not be exact, the R_{min} could use some explaining.

While it may be possible on some LDRs, this LDR is not built to get to 0 Ω. Since the ADC only has a resolution of 8 bits (256 distinct numbers), there is some data loss when reading from the ADC. The next lowest value of the ADC could help us find the range:

$$\frac{254}{255} * 5 = \frac{5 * 10k}{R + 10k}$$

$$R \approx 408\ \Omega$$

This means R_{min} is not actually 0 Ω, but between 0 and 408 Ω.

The required resistance for the other resistor in the voltage divider LDR circuit would be < R_{max} (20 MΩ) and > R_{min} (~ 0 - 408 Ω). I used a 10 kΩ resistor which falls inside this range.

2) The minimum current flowing to Ch0 of the ADC is at R_{max}, so:

$$I_{max} = \frac{5}{20M + 10k} \approx 0.25\ \mu A$$

The maximum current flowing to Ch0 of the ADC is at R_{min}, so:

$$I_{min} = \frac{5}{0 + 10k} = 0.5\ mA$$

Video Demonstration

The video demonstration has been uploaded alongside this lab report.

NOTE: I had to change the light threshold values from 127 to 191 in the demonstration to adjust the circuit to my room's light level.

```
#include <wiringPi.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <time.h>

#define ADC_CS      0   // GPIO17, pin 11
#define ADC_DIO     1   // GPIO18, pin 12
#define ADC_CLK     2   // GPIO27, pin 13

#define LED     3   // GPIO23, pin 16
#define BTN     4   // GPIO22, pin 15

#define HALFPERIOD  5   // Half period for wait cmds

unsigned char ADC_val, LED_FLAG, LED_mem, LED_set;
time_t new_Time, old_Time;

// Button ISR
void btn_Interrupt(void)
{
    // Sets mem variable to remember original LED state before manual 
override
    if (!LED_FLAG) LED_set = (ADC_val > 127) ? 1 : 0;

    // Sets FLAG to stop automatic LED changing
    LED_FLAG = 1;

    delay(10);

    // Checks for previous LED state and sets to the opposite
    // Updates mem variable accordingly
    // Updates time variables and prints respective timings
    if (digitalRead(BTN) == 0)
    {
        if (LED_mem)
        {
            digitalWrite(LED, HIGH);
            old_Time = new_Time;
            new_Time = time(NULL);
            printf("LED turned ON on %s", ctime(&new_Time));
            printf("LED was OFF for %.f second(s).\n", difftime(new_Time, 
old_Time));
            LED_mem = 0;

        } else
        {
            digitalWrite(LED, LOW);
            old_Time = new_Time;
            new_Time = time(NULL);
            printf("LED turned OFF on %s", ctime(&new_Time));
            printf("LED was ON for %.f second(s).\n", difftime(new_Time, 
old_Time));
            LED_mem = 1;
        }
    }
    // Debouncer helper
    while(!digitalRead(BTN));
}

// Changes LED based on current ADC value
void change_LED(void)
{
    // Turns off LED if ADC value > 127 (2.5V)
    // Turns on LED if ADC value <= 127 (2.5V)
    // Updates time variables and prints respective timings
    if (ADC_val > 127)
    {
        digitalWrite(LED, LOW);
        if (!LED_mem)
        {
            old_Time = new_Time;
            new_Time = time(NULL);
            printf("LED turned OFF on %s", ctime(&new_Time));
            printf("LED was ON for %.f second(s).\n", difftime(new_Time,
old_Time));
        }
        LED_mem = 1;
    } else
    {
        digitalWrite(LED, HIGH);
        if (LED_mem)
        {
            old_Time = new_Time;
            new_Time = time(NULL);
            printf("LED turned ON on %s", ctime(&new_Time));
            printf("LED was OFF for %.f second(s).\n", difftime(new_Time,
old_Time));
        }
        LED_mem = 0;
    }
}

// Simple method to go to the next clock cycle
void next_CLK(void)
{
    // Set CLK to HIGH for one 1/2 period and to LOW for one 1/2 period
    digitalWrite(ADC_CLK, HIGH);
    delayMicroseconds(HALFPERIOD);
    digitalWrite(ADC_CLK, LOW);
    delayMicroseconds(HALFPERIOD);
}

// Reads the current ADC value
void read_ADC(void)
{
    unsigned char data_1 = 0, data_2 = 0;
    int i;

    // Sets up the required start signals to read from the ADC
    // Follows pp. 9, Fig. 19 of the ADC0832-N datasheet
    digitalWrite(ADC_CS, LOW);
    digitalWrite(ADC_CLK,LOW);
    digitalWrite(ADC_DIO,HIGH);
    delayMicroseconds(HALFPERIOD);

    digitalWrite(ADC_CLK,HIGH);
    delayMicroseconds(HALFPERIOD);

    digitalWrite(ADC_CLK,LOW);
    digitalWrite(ADC_DIO,HIGH);
    delayMicroseconds(HALFPERIOD);

    digitalWrite(ADC_CLK,HIGH);
    delayMicroseconds(HALFPERIOD);

    digitalWrite(ADC_CLK,LOW);
    digitalWrite(ADC_DIO,LOW);
    delayMicroseconds(HALFPERIOD);

    digitalWrite(ADC_CLK,HIGH);
    digitalWrite(ADC_DIO,HIGH);
    delayMicroseconds(HALFPERIOD);

    digitalWrite(ADC_CLK,LOW);
    digitalWrite(ADC_DIO,HIGH);
    delayMicroseconds(HALFPERIOD);

    // Sets DIO pin to take input after ADC is ready to be read
    pinMode(ADC_DIO, INPUT);

    // Reads ADC MSB -> LSB
    // Stores value into a 1 byte variable
    // Left shifts old variable and ORs it with current bit value
    for (i = 0; i < 8; i++)
    {
        next_CLK();
        data_1 = data_1 << 1 | digitalRead(ADC_DIO);
    }

    // Reads ADC LSB -> MSB
    // Stores value into a 1 byte variable
    // Left shifts current bit value "i" times and ORs it with old variable
    for (i = 0; i < 8; i++)
    {
        data_2 = data_2 | digitalRead(ADC_DIO) << i;
        next_CLK();
    }

    // Sets ADC back up to take input for next read
    digitalWrite(ADC_CS, HIGH);
    pinMode(ADC_DIO, OUTPUT);

    // Checks for errors by comparing MSB -> LSB and LSB -> MSB values
    ADC_val = (data_1 == data_2) ? data_1 : 0;
}

int main(void)
{
    // Sets up WiringPi
    if (wiringPiSetup() < 0)
    {
        printf("Setup wiringPi failed!\n");
        return -1;
    }

    // Sets up BTN interrupt
    if (wiringPiISR(BTN, INT_EDGE_FALLING, &btn_Interrupt) < 0)
    {
        fprintf (stderr, "Unable to setup ISR: %s\n", strerror (errno));
        return -1;
    }

    // Sets up button/LED
    pinMode(LED, OUTPUT);
    pinMode(BTN, INPUT);
    pullUpDnControl(BTN, PUD_UP);

    // Sets up ADC
    pinMode(ADC_CS, OUTPUT);
    pinMode(ADC_DIO, OUTPUT);
    pinMode(ADC_CLK, OUTPUT);

    // Sets up ADC initial values
    digitalWrite(ADC_CS, HIGH);
    digitalWrite(ADC_DIO, LOW);
    digitalWrite(ADC_CLK, LOW);

    // Get current time for LED switching
    new_Time = time(NULL);

    while(1)
    {
        read_ADC();
        printf("Voltage = %f\n", ADC_val/255.0 * 5);

        if (!LED_FLAG) change_LED();

        if ((LED_FLAG && LED_set == 0 && ADC_val > 127) || (LED_FLAG &&
LED_set == 1 && ADC_val <= 127))
        {
            change_LED();
            LED_FLAG = 0;
        }
        delay(500);
    }
}
```