

Lab 12: Raspberry Pi controlled RGB light dimmer

The first half of this lab is all about setting up your Raspberry pi. Please follow the tutorial listed below.

NOTE: The GPIO Pins on Raspberry Pi cannot handle more than 3.3V. When connecting your breadboard to your Raspberry Pi you MUST power the breadboard ONLY using the Raspberry Pi GPIO pins.

Check Raspberry Pi to breadboard connections twice.

Learning Objective

- How digital read/write works
- How an LED can be controlled
- How brightness can be controlled using pulse-width modulation

Materials Required

RGB LED, Breadboard, Jumper-wires, 5 pushbutton switches, Raspberry Pi, SD Card, and USB-to-microUSB cable

Introduction

Finally we have our hands on the Raspberry Pi that was sitting idle in the kit. Raspberry Pi is one of the cheapest as well as smallest computers today. Do not confuse Pi with Arduino which is a microcontroller, Pi is a computer! It is more software capable than micro-controllers and can match up hardware capability with addition of few external chips. We will be using Raspberry Pi in this course for:

- GPIO (General Purpose Input-Output) Pins •
ARM Assembly

Most of your time during this lab will be spent on TASK 1 and TASK 2. Do not skip any tutorial in TASK 1 or you will miss out on important procedures and tips.

TASK 1: Raspberry Pi Setup

1. [Wiring Pi Libraries](#)
2. [GPIO in 4 Steps](#)
3. [WiringPi code Examples](#)
4. [raspberrypi-gpio-explained](#)

It is also important to know how to safely shut down your Pi (this command is more stable than `sudo halt`):

```
/* -h argument is for halt*/  
$ sudo shutdown -h now
```

Restart:



















```
/* -r argument is for reboot */  
$ sudo shutdown -r now
```

TASK 2: Digital Read/Write

We are now going to perform digital read and digital write between the Raspberry Pi and breadboard. **Don't power your breadboard until your wiring is complete.**

Note that we are powering the breadboard using 3.3V from the Raspberry Pi GPIO pins XXXXXXXXX so that the pushbutton switch voltage divider circuit on the breadboard generates logic high and low signals at voltages (0 volts and +3.3 volts) that are compatible with the allowable input voltages for GPIO pins of the Pi.

Here is a diagram of the general purpose input/output (GPIO) pins of the Raspberry Pi Model 3B. The even numbered pins are in the row of pins closest to the edge of the Pi circuit board. The odd numbered pins are farther from the edge of the Pi.

Raspberry Pi 3 GPIO Header				
Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I²C)		DC Power 5v	04
05	GPIO03 (SCL1 , I²C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)		(I²C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

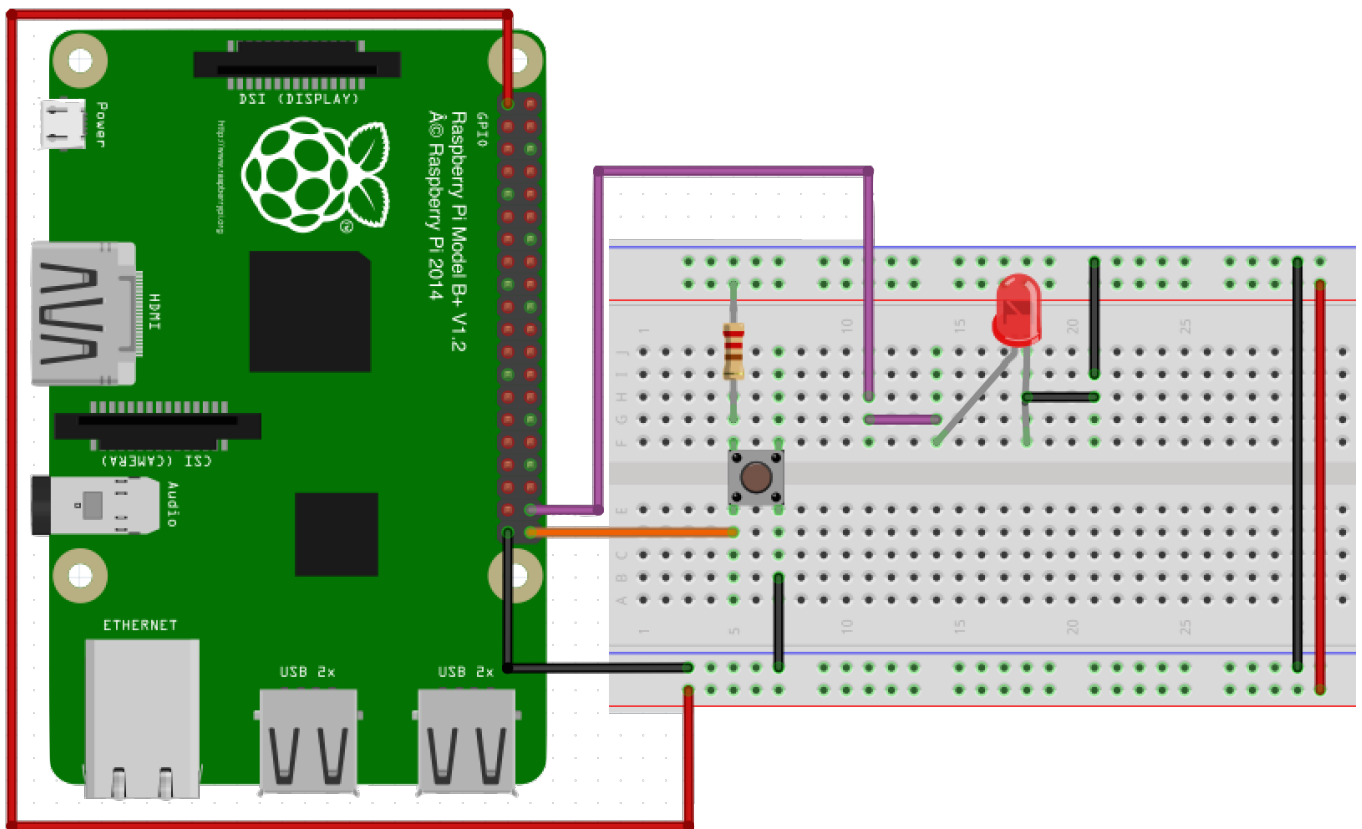
Rev. 2
29/02/2016

www.element14.com/RaspberryPi

The definitions of GPIO pins has changed over time and referring to a specific pin by its hardware name is cumbersome. To get a full mapping of the GPIO pins on your Raspberry pi, type **gpio readall** to a shell prompt.

This will work regardless of the Pi make, model or version you have. The central columns of the gpio readall output are the GPIO hardware pin definitions as shown in the diagram above. The wPi left and right columns show the pin numbers that the wiringPi library code uses as symbolic names for each GPIO pin for which the voltage level can be controlled by software. GPIO pins that provide a fixed 0 volts or +3.3 volts do not have wiringPi symbolic pin numbers because their voltage is fixed and not controllable by software. Using the wiringPi library to refer to pins in your application software means that if future versions of the Pi again change the hardware configuration of the GPIO pins, the provision of a new version of wiringPi for that new Pi hardware will be sufficient to make all old software using GPIO pins able to work on the new Pi.

First build the Task 2 circuit on your breadboard as shown below. Second, connect the breadboard circuit to your Pi. The lavender and orange colored wires indicate female-to-male jumpers from your lab kit that connect to GPIO pins 20 and 21, respectively, using the female jumper connector and to the breadboard tie point of choice using the male jumper connector. The black and red wires are jumpers for +3.3 volts and Ground that connect to GPIO pins 01 and 39, respectively, and the power bus rails on your breadboard. Once this wiring is complete, your Pi may be powered from a USB port using the USB cable in your lab kit.



Resistor value = 10 KOhm.

Check your Pi to breadboard wiring twice. Now power your Pi from a USB port.

Login to your Pi.

```
$ mkdir cs250
$ cd cs250
$ mkdir lab12
$ cd lab12
$ nano readWrite.c # choose your favorite text editor
```

Then select the source code from below, copy, and paste into your editor.

```
#include <stdio.h>
#include <wiringPi.h>

/* Set Pin Numbers */
#define LED 28
#define BUTTON 29

int main(void) {

    /* Initialize GPIO pins to map to wiringPi numbers */
    if(wiringPiSetup() == -1)
        return 1; //return with error status (initialization failed)

    /* Set modes for the GPIO pins */
    pinMode(LED, OUTPUT);
    pinMode(BUTTON, INPUT);

    int value = LOW;

    while(1) {

        /* Read from the pin number defined for button */
        value = digitalRead(BUTTON);

        /* Write the value to positive pin of the led */
        digitalWrite(LED, value);

        /* Use of delay is important as there is a small lag in
        reading/writing values each time */

        delay(100);

    }

    return 0;
}
```

To compile the program you will have to link the wiringPi library. **When editing the Makefile use TAB for indentation instead of spaces. Directly copying the code below will result in your Makefile not running correctly.**

```
$ nano Makefile

/* Inside the Makefile */
all: main

main: readWrite.c      gcc -std=c99 -o main readWrite.c -lwiringPi
# use a <tab> for indentation at the beginning of this line

clean:
    rm -r *.o # <tab> on this line, too
```

To compile the program, type “make” in response to a shell prompt in the terminal.

```
$ make
```

Since you are making changes to default GPIO pin modes and reading and writing values, you will need to run the program as root or type:

```
$ sudo ./main
```

TASK 3: Take-home Assignment: Program an LED Dimmer Control using Pulse-Width Modulation

Demo here: <https://www.youtube.com/watch?v=jfpvuolFGMM> Note that the way the switches are wired on the breadboard is incorrect, leaving the GPIO inputs floating. You should use the voltage divider circuit shown above.

Given a constant voltage input an LED will shine with a constant brightness. With a varying voltage an LED will vary in brightness.

With the Raspberry Pi, writing HIGH to a GPIO output pin causes that pin to deliver the logic high voltage for the Pi, which is 3.3V. Sending this voltage to an LED will cause the LED to shine with a constant brightness. How, then, can the Pi be used as a programmable dimmer control for an LED?

The answer is to take advantage of the behavior of the human visual perception system. In particular, our eyes cannot see as individual events a series of events that occurs faster than around 20 times per second. Events presented to our eyes at roughly this rate or faster are perceived as one continuous event rather than as a sequence of events. So, first, if we flash an LED faster than this perceptual rate limit, then our eyes will not perceive the LED as flashing. Second, if we vary the ratio of the times that the LED is ON and OFF, we will affect the perceived brightness of the flashing LED. The larger the fraction of time that the LED is off, the dimmer the LED will seem. Varying the ratio of ON to OFF time of a pulsing digital signal is called pulse-width modulation (PWM). For PWM, the pulse is the ON part of the signal to the LED, and the pulse width is the ratio of the ON time to the OFF time. PWM has many applications in addition to dimmer controls for lights, including voltage changing power supplies, data transmission, class-D energy efficient audio amplifiers, and some sound synthesis, particularly string and chorus sounds.

So, for this lab you can make use of the PWM library provided in the Pi to control the pulse. [Here](#) is an explanation of how this works.

Raspberry Pi OS has software Pulse-Width Modulation pre-installed. You need to include `softPwm.h` in your source code. The `softPwm` has the following API:

```
/* This creates a software controlled PWM pin. You can use any GPIO pin
and the pin * numbering will be that of the wiringPiSetup() function you
used. Use 100 for the pwmRange, * then the value can be anything from 0
(off) to 100 (fully on) for the given pin.

* The return value is 0 for success. Anything else and you should check
the global errno * variable to see what went wrong.
*/
int softPwmCreate (int pin, int initialValue, int pwmRange);

/* This updates the PWM value on the given pin. The value is checked to
be in-range and pins that haven't previously been initialized via
softPwmCreate will be silently ignored. */

void softPwmWrite (int pin, int value) ;
```

Your graded task for this lab is to implement an [RGB color blender](#). You will need 5 pushbuttons for the following:

- Select red color
- Select green color
- Select blue color
- Increase brightness.
- Decrease brightness.

In addition to these you will need an RGB Led and jumper wires for connecting the components to GPIO pins.

Your blender can either have color range from 0-255 for all three colors or 0-100%. If you are implementing it on a 0-100 scale, you could do a +5 brightness on increase button and -5 on decrease button. Similarly, if you are implementing it on a 0-255 scale, you could do a +12.75 brightness on increase button and -12.75 on decrease button.

Do not keep increasing the brightness once max brightness has been reached or it may cause undefined behavior. Every time you want to change the wiring, make sure to unplug power from your Pi for the safety of your hardware components.

Write your code in `pwm.c`

```
#include <stdio.h>
#include <wiringPi.h>
#include <softPwm.h>
```

```

/*You can set Pin numbers here */

int main(void) {

/*Write your code here */


    return 0;
}

```

You will have to link pthread library along with wiringPi library

```
$ gcc -std=c99 -o pwm pwm.c -lwiringPi -lpthread
```

Grading Criteria

Criteria	Points
Task 3: Demonstrate your code and circuit to your TA	
Demonstrate Increase/Decrease brightness of one color using buttons/switches	30 Points
Demonstrate Increase/Decrease brightness of multiple colors	30 Points
Be able to explain what each part of the code does	10 points
Comments in code	10 points
Answer questions about the lab from your TA	20 points

Turnin

Create a ~/cs250/lab12-src directory on your Pi and place all of your work in it.

Before your lab session starts, submit your lab12-src directory electronically on **data.cs.purdue.edu** using the turnin command:

```
$ turnin -c cs250 -p lab12 lab12-src
```

```
$ turnin -c cs250 -p lab12 -v
```

Remember to submit from data.cs.purdue.edu.