EECS 2031A 2018

Lab #2 Simple GPIO operations

EECS 2031 Lab #2

Simple GPIO operations

This lab involves writing code that interacts with the real world through the GPIO (General Purpose Input and Output) pins on your Raspberry Pi. This lab assumes that you have completed Lab #1 successfully. If you have not, then go do that now. You cannot do lab #2 without having an operating Raspberry Pi.

This lab involves completing very simple electrical circuits and connecting them to your Raspberry Pi. These circuits are very simple – they consist of a few LED's and resistors and switches. That being said if you complete these circuits incorrectly you can damage some of the electrical and electronic components. It is thus important that you verify all circuits prior to applying power to them. Suggestion: Always check your wiring prior to providing power to a circuit. If this was carpentry that would be "measure twice, cut once".

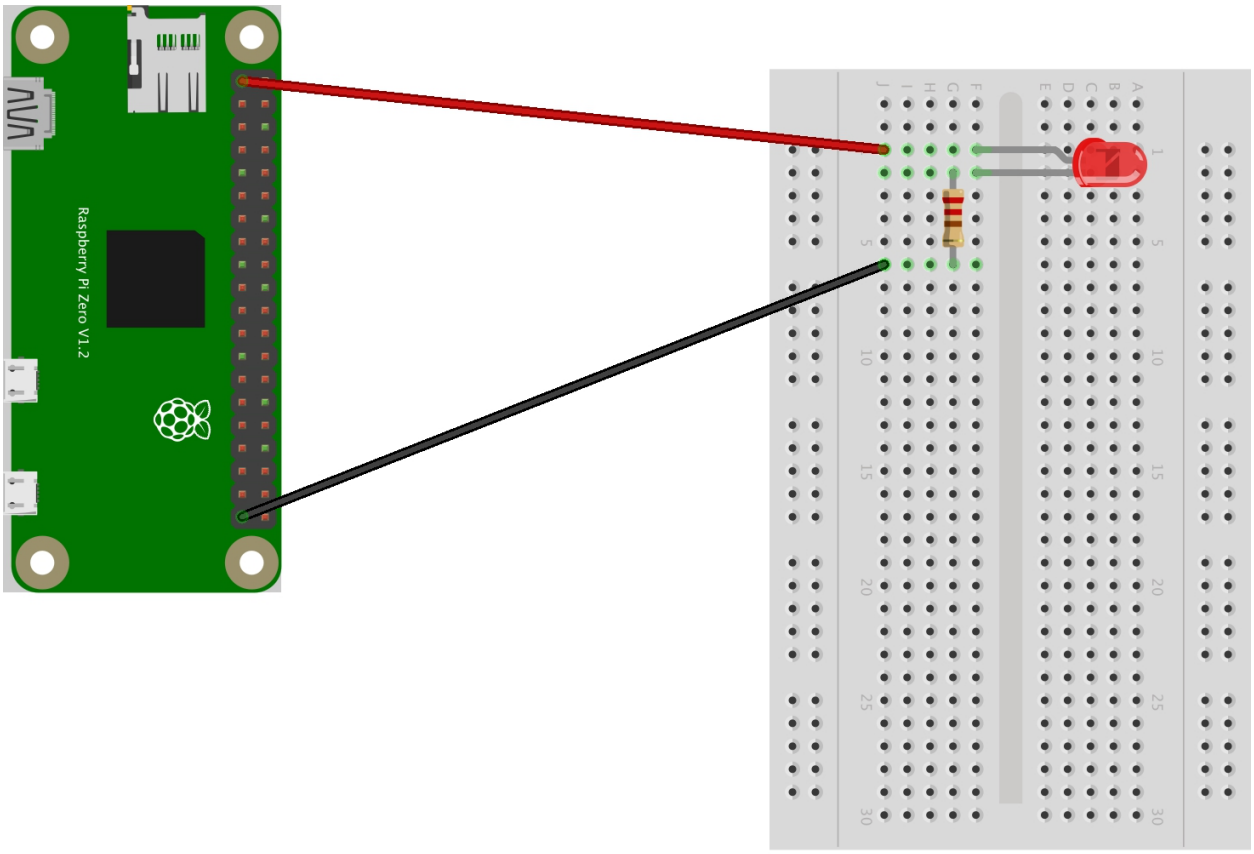
In order to complete this lab and obtain a grade for it you will have to demonstrate to your TA that you have completed the various steps in this lab. This lab, like all labs in this course, are intended to be experiential. You are encouraged to ask for help from both the TA's and your fellow classmates, and to help you fellow classmates in completing the lab. That being said, you need to master the material being presented here.

Blinking a LED

Connect your RaspberryPi to a machine in the PRISM lab using the UART cable that you used in the first lab. Connect the power connection to the

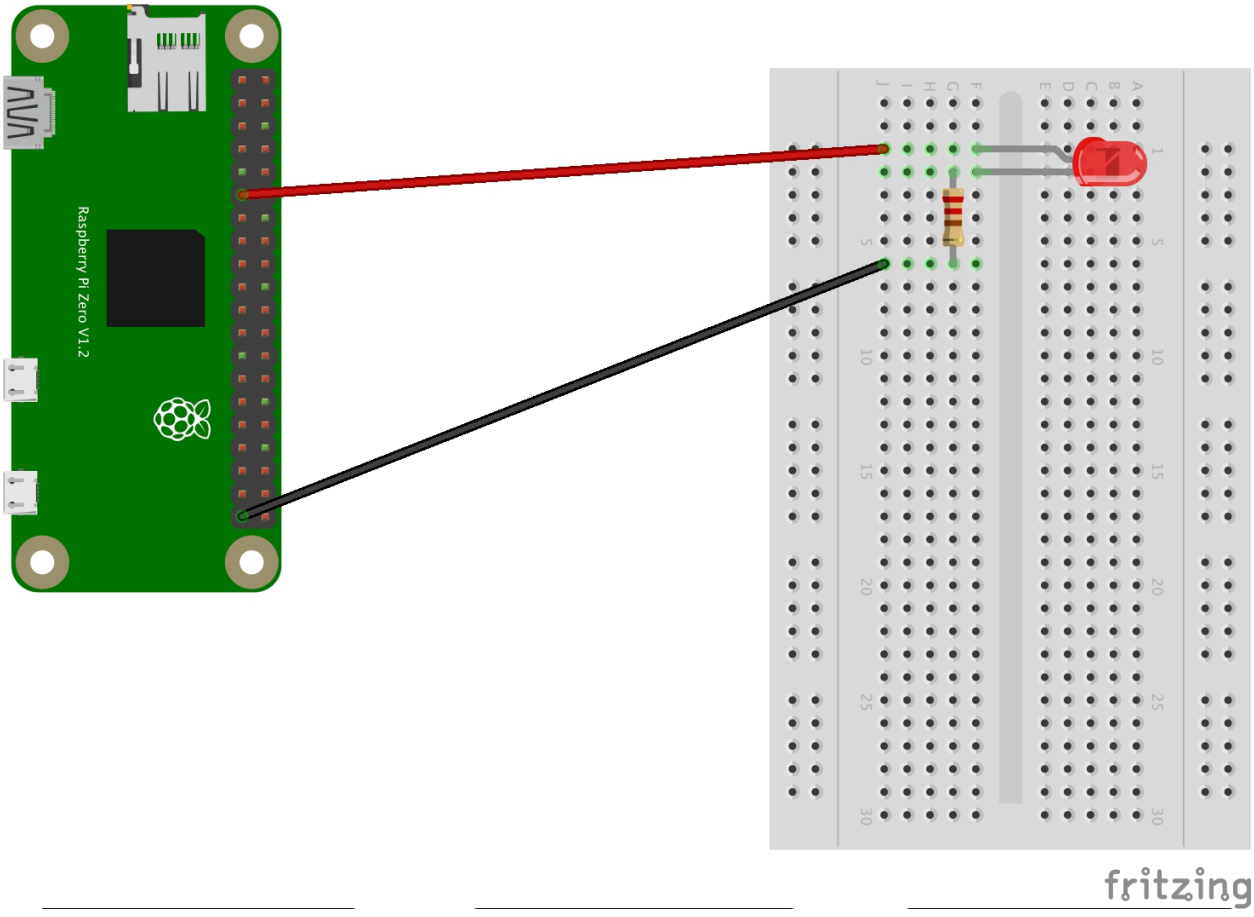
Raspberry Pi but **do not yet connect the other end of the power connection to an external power supply**. Use the parts in your kit and your Raspberry Pi to duplicate the circuit shown below. Note that the red connector goes from the top pin of the second line of pins and that the black connector goes from the bottom pin of the second line of pins. Also note that the long leg of the LED (the anode) is connected to the red (positive) 3.3V power supply on the Raspberry Pi and the shorter leg of the LED (the cathode) is connected to the resistor which is then connected to the ground. Almost any resistor will work here and one one in your kit will work fine.

The prototyping board that you have in your kit includes two small breadboards with detachable power rails. The power rails are connected



as are the rows. So the circuit above provides power from the 3.3V supply on the Raspberry Pi through the LED, through a resistor back to ground. Check the circuit and plug your Raspberry Pi into power and the LED should light up. If not, check your circuit.

Some things to note: If you did not have a resistor in the circuit the LED will glow very brightly for a very short period of time and then burn out. The resistor is necessary to limit the current passing through the LED. Second, LED's have a polarity (they only work with power running through the circuit in one direction). So if you plug the LED in the wrong way around nothing will work.

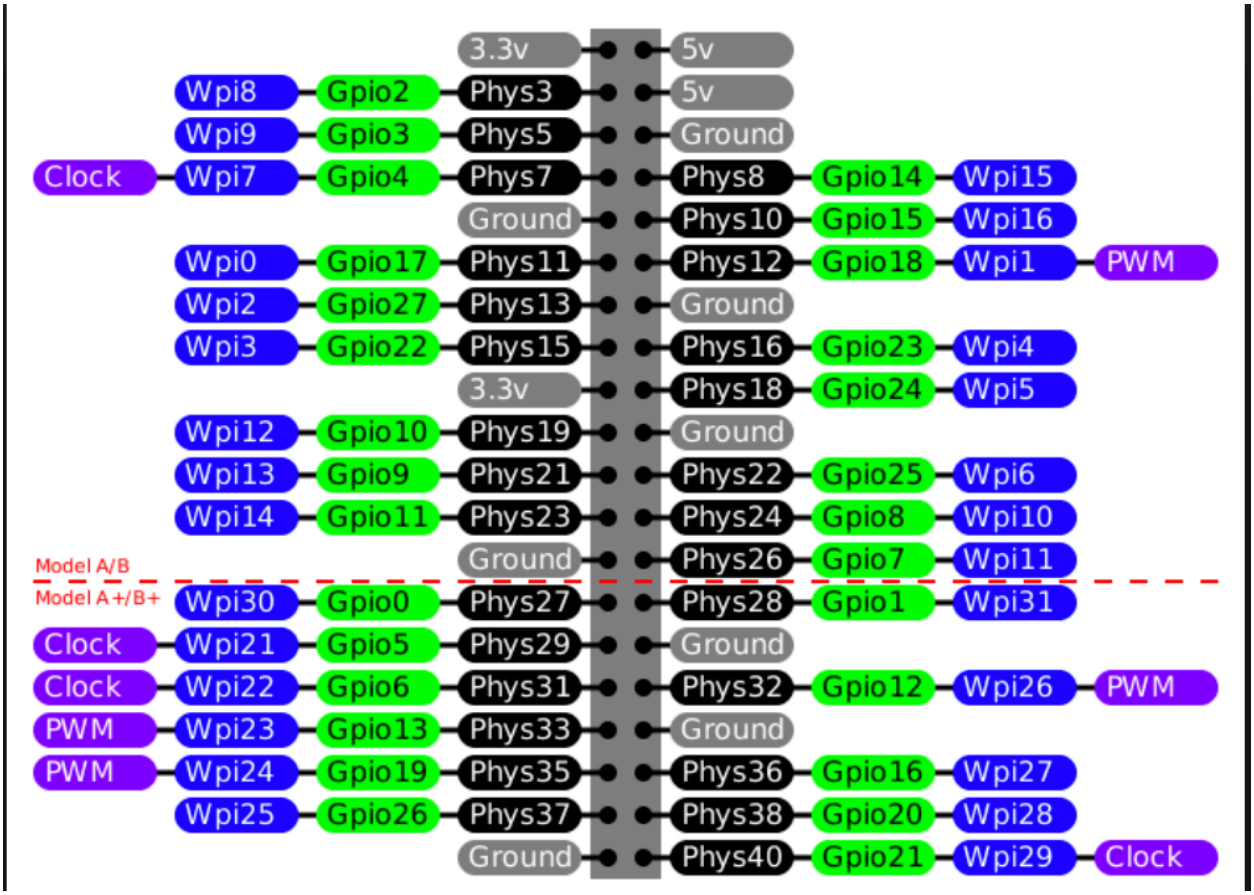


So once this is working, carefully unplug the red 3.3V connector from your PI and plug it back in. The LED should flash off and on. If you could control the power coming out of the pin on your Raspberry Pi then you could control the LED from software. That is what you are going to do now. Slightly modify the circuit you built above so that it looks like the circuit shown above. (Basically disconnect the connector that went from the top pin on the second rail of the GPIO connector and connect it to the sixth pin of the second row of pins.

Now for a small technical detail. The GPIO pins on the Raspberry Pi are known by different names. The software library that you are going to use (called the wiringPi library) labels the pins differently than one of the more commonly used standards. The figure below shows the labelling used by a number of different standards. You will note that the sixth pin on the second rail is known as Wpio0 and Gpio17 or Phys11. They are all the same pin. Similarly the UART-USB cable you use is plugged into the ground, Phys8-Gpio14-Wpi15 and Phy10-Gpio-15-Wpi16. The software library (WiringPi) used the labelling identified as WPi0 so you have connected the circuit to pins Wpio0 (wiring pin 0) and Ground. This is not a huge issue, its just that you must pay careful attention to making sure that the pins are the one you want.

Now with the circuit connected (the LED will not be lit). Log onto your Raspberry Pi following the instructions given in the first Lab. Your Raspberry Pi will have the UART-USB connector on the outer rail and your two connections on the inner rail. It should boot up and the LED will not be lit. Log in.

Use the cd (change directory) command to change directory to Documents and create a directory (using the command mkdir) lab02.



While you are there, create another directory lab01 and move all of the files associated with your first lab there.

- mkdir file - creates a directory file either in the current directory or in some other directory if file is an absolute path name
- cd dir - changes your current (working) directory. Files in UNIX are stored in a tree of directories. .. moves up a level in the tree. cd filename moves down into the directory filename. If filename is an absolute path then you will move to there.
- mv src dst - move (rename) a file called src to dst. By giving path names to each you can move files around your file system.

Use the editor nano to create a file blink.c that contains the following

```
#include <wiringPi.h>
int main (int argc, char *argv[])
{
  wiringPiSetup () ;
  pinMode (0, OUTPUT) ;
  for (;;)
  {
    digitalWrite (0, HIGH) ; delay (1000) ;
    digitalWrite (0,  LOW) ; delay (500) ;
  }
  return 0 ;
}
```

Hint: cut and paste this from the pdf document to the editor. Save yourself some work. Compile blink.c using

```
gcc –ansi –pedantic –Wall blink.c –lwiringPi –o blink
```

And run it using ./blink

Assuming it all works, the LED should blink. It will run forever. Type ^C (control C) to abort the program.

Read through the code. You should be able to follow that

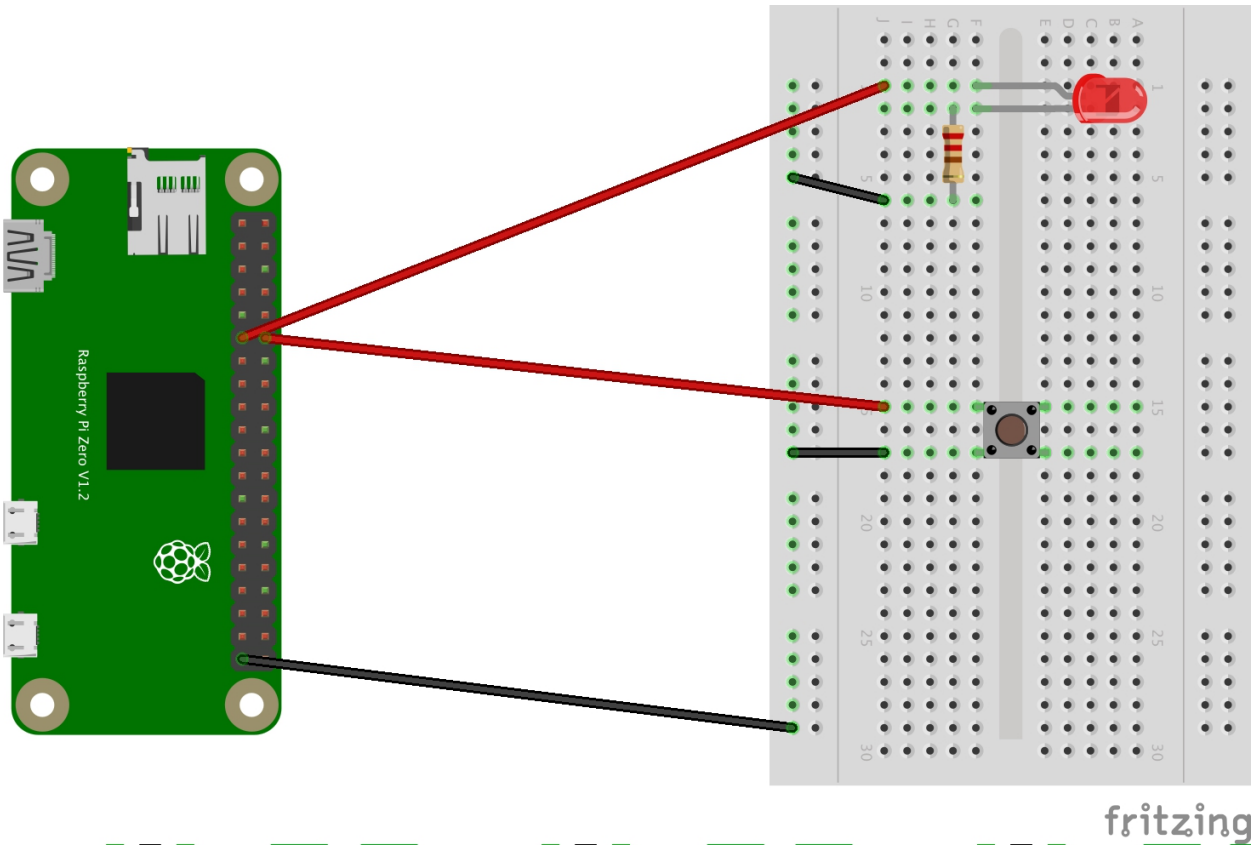
- The code relies on the wiringPi library for which it includes a header file and links an external library
- The code does some setup and the loops forever.
- The loop calls digitalWrite followed by a delay, twice. The only difference in the digital write calls is that the second argument is either

HIGH (Wpi0 pin is set to HIGH or 3.3V) or LOW (Wpi0) pin is set to LOW or 0V).

- The initialization sets pin 0 to be an output pin

Modify the code so that the LED blinks on for 500 (half a second) then off for 500 (half a second) and run it. Test it.

Now you are going to create a second circuit on your prototyping board. This circuit builds on the work you did in your first circuit and is shown below.



As the power rails run the full length of the rail, you should see that the original circuit is essentially left unchanged while a new circuit connects

Wpi01 through a button to ground. You have a button in your kit. Note that buttons typically have four connectors, pairs are always connected while the two halves become connected when the button is pressed. Your TA can help you identify which way to connect the button to your prototyping board.

Create a new file button.c in your lab02 directory that contains the following

```
#include <wiringPi.h>
#include <stdio.h>
int main (int argc, char *argv[])
{
  wiringPiSetup () ;
  pinMode (1, INPUT) ;
  pullUpDnControl(1, PUD_UP) ;
  for (;;)
  {
    int x = digitalRead(1);
    printf("Got a %d\n",x);
  }
  return 0 ;
}
```

Following the pattern established for blink.c, compile button.c into button and execute it. Your code should print out 'Got a 1' over and over. If you push and hold the button down the output should change to 'Got a 0'.

Examine the code and based on what you did in 'hello.c' and 'blink.c' understand what this code does. Specifically

- It assigns pin1 (WPi1) as input and then calls pullUpDnControl(1, PUD_UP).

- It then loops forever reading the value of pin 1 (WPi1) and prints out it state.

So a couple of interesting points here. GPIO pins can either be used for input or output. So you have to tell the library which way you are going to use the pin. The next critical electrical thing is that you might want to just check an input to see if it was high, or not. Unfortunately a circuit that is not connected to anything will float and so it's electrical state will not be reliable. So its much safer to have a button circuit that either reads ground (0) or 3.3V (1). This is such a common problem that many chip sets – including the GPIO pins on the RapsberryPi have resistors built into the board. These are typically either pull up resistors (they connect to 3.3V on the Pi) or pull down resistors (they connect to ground). The call to pullUpDnControl tells the library that we want to use a pull up resistor so that if the button is not connected to ground we will see 3.3V (or a digital 1), and if the button is connected to ground we will see 0V (or a digital 0).

Finally, the code that you have to write for this lab. Based on the code provided in blink.c and button.c, write a new piece of C code button2.c. This code should use the circuits you have built already and turn the LED on when the button is pressed and turn it off when it is not pressed.

LINUX tools introduced in this lab

- cd - change directory
- mkdir - create (make) a directory
- mv - move (rename) a file
- Linking against an external library and including the necessary include file for that library.

Grading

To obtain a grade for this lab you must show your TA your Raspberry Pi running the blink2 program. You can do this in your lab, during any of the TA's office hours before your next lab, or at the very start of your next lab. Late solutions will not be accepted.