EECS 2031A 2018

Lab #9 GPIO pin input using BASH

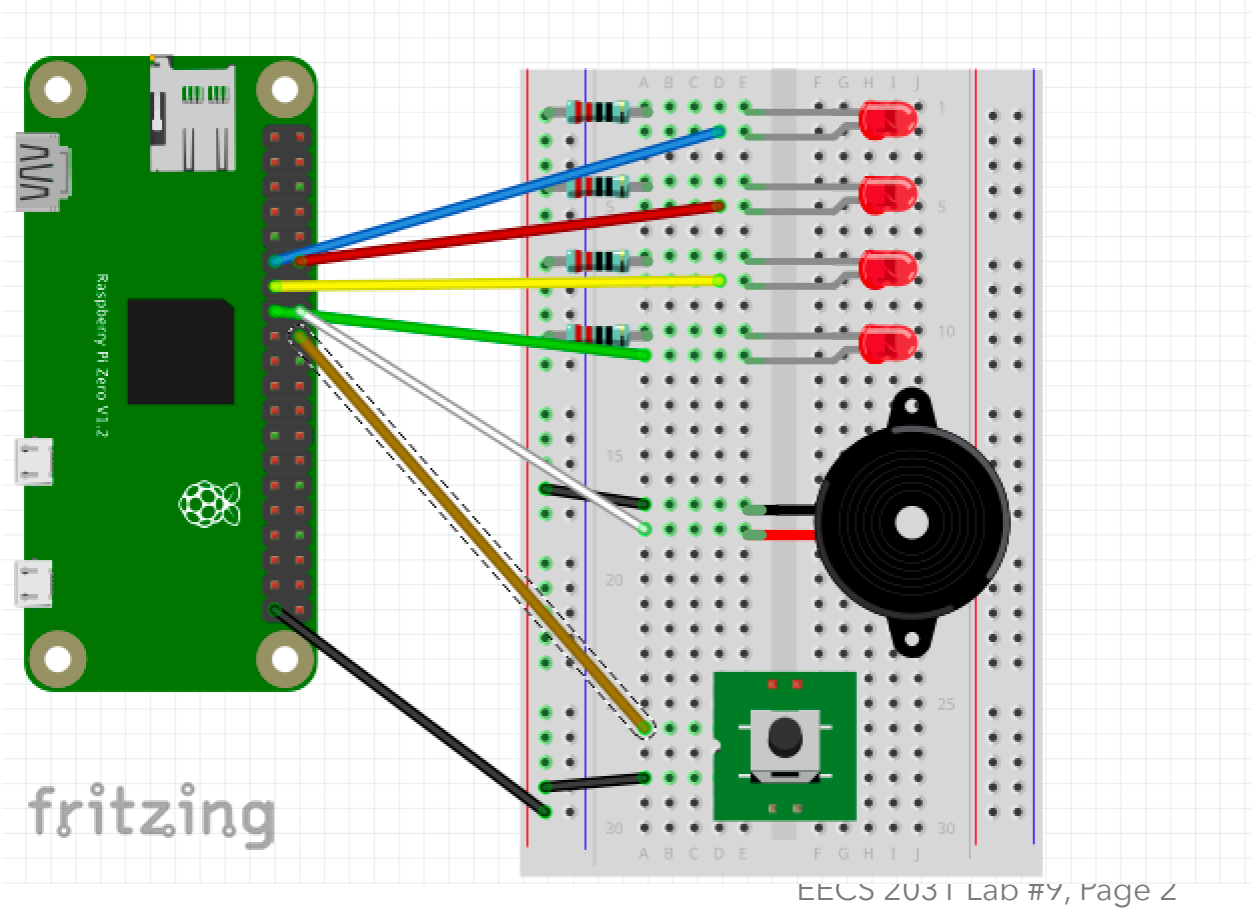
EECS 2031 Lab #9

GPIO pin input using BASH

This lab deals with GPIO pin input (and output) using BASH. That is, to build a simple circuit with LED's making up a display and then using BASH to control power in the circuit and to have your circuit respond to button presses. This lab builds upon the previous lab, so you should have completed the previous lab before doing this one

Four bit display, a buzzer and a switch

So we begin first with the circuit. In building this circuit the lab assumes that you have four working LED's at this point and four resistors. If you



only have three, then only build a circuit with three LEDs. (If you have more, then lend one to someone who only has two.) But you will need at least three working LEDs to make this lab work.

The circuit you will use in this lab is shown above. Note that it just adds to the circuit that you built in the previous lab. (So if you have not taken the old one apart, you should start there.)

Basically, this is a collection of four LED's, each of which is connected via a resistor to ground and has power provided by WiringPi Pins 0, 1, 2 and 3. Physically the four LEDs should be laid out as shown so that they are in a row and wired up in order. This will make much more sense soon. Wiring Pi pin 4 is connected to a buzzer which is connected to ground. And Wiring Pi pin 5 is connected to a switch that is connected to ground.

GPIO Command

Once you have the circuit built and are sure that it is wired correctly, power up your Raspberry Pi and log onto the machine. The gpio command can be used to control the gpio pins from the shell, much like you did using the wiringPi library in earlier labs. Full details on the gpio command can be found at wiringpi.com and also through the man command (man gpio) on the Raspberry Pi.

As with the wiringPi library, the gpio command can be used to manipulate the gpio pins on the Raspberry Pi. Critical commands include

- gpio mode pin out - sets the mode of 'pin' to be output
- gpio mode pin in - set the mode of 'pin' to be input

- gpio mode pin up - set the input mode of the specified pin to be wired to a pull up resistor.
- gpio write pin value - sets the output value of 'pin' to be 'value'
- gpio read pin - reads the current value of 'pin' and returns that as the

There are, of course, many other possible arguments for the gpio command, and you are encouraged to test out a few of them. (man gpio is a very helpful place to start).

You are now going to develop a collection of shell scripts to manipulate the circuit you have created. You should create a GitHub site and clone this repository onto your Raspberry Pi. Include each of these scripts in your repository. Each script should start with an appropriate hash-bang line to ensure that the script is executed by /bin/bash. Each script should have its permissions set to 755 so that you can execute the file.

Bash scripts

initMode.sh - this should set the output mode of each of pins 0 through 4 inclusive to be output and set their current output value to be 0. (off). It should also set the mode of pin 5 to be input and enable a pull up resistor on this input pin. Note that gpio mode values remain in their input/output mode until changed, so once you run initMode.sh it should not have to be run again until you power the Raspberry Pi on and off. Note: Your code should use a for loop to loop over the pins, not just set each pin manually.

setbits.sh - Imagine that the pins corresponds to bits in a 4 bit number (from 0 through 15). setbits.sh should take one argument, a number in the range 0 through 15 and set the on value of the four LEDs so that the

bit pattern corresponds to the number in base 2. Or if you prefer, the bits should be set as follows. Call the bits (or LED's) from left to right bit1, bit2, bit3, bit4 and the number n. Then

- Bit1 is on iff (n)& 1
- Bit2 is on iff (n>>1) & 1
- Bit3 is on iff (n>>2) & 1
- Bit4 is on iff (n >> 3) & 1

Remember that the first argument to a shell script is in the variable $1 and that the let command supports both shift and bitwise and operators. This is the same code that you wrote in the previous lab.

waitForButtonPress.sh - Write a shell script that monitors the value of gpio pin 5. It should busy wait (continually check and loop) as long as the pin is 1 (not pressed). It should then busy wait as long as the pin is 0 (pressed). The goal here is that this shell should wait until the button is pressed and then released. This problem of 'de-bouncing' an input value is common.

countButtonPress.sh - Write a shell script that counts button presses. For button presses 1 through 15 it should light the LED's so that they represent that number as a binary number. That is, if the LEDS correspond to 4 bits in a number 0000, then as you push the buttons the pins will be lit 0001, 0010, 0011, …, 1111. On the 16th button push the pins should all go off (0000) and the buzzer should sound for one second and then go off.

Some observations: turning the individual bits on and off was solved in the previous lab. So the major effort here is in terms of dealing with button presses. Debug that portion of the code first.

Grading

To obtain a grade for this lab you must show your TA your Raspberry Pi running the code that you developed and that your code has been properly added your own GitHub repository. Your TA will ask you to demonstrate the 10 commands you have added.