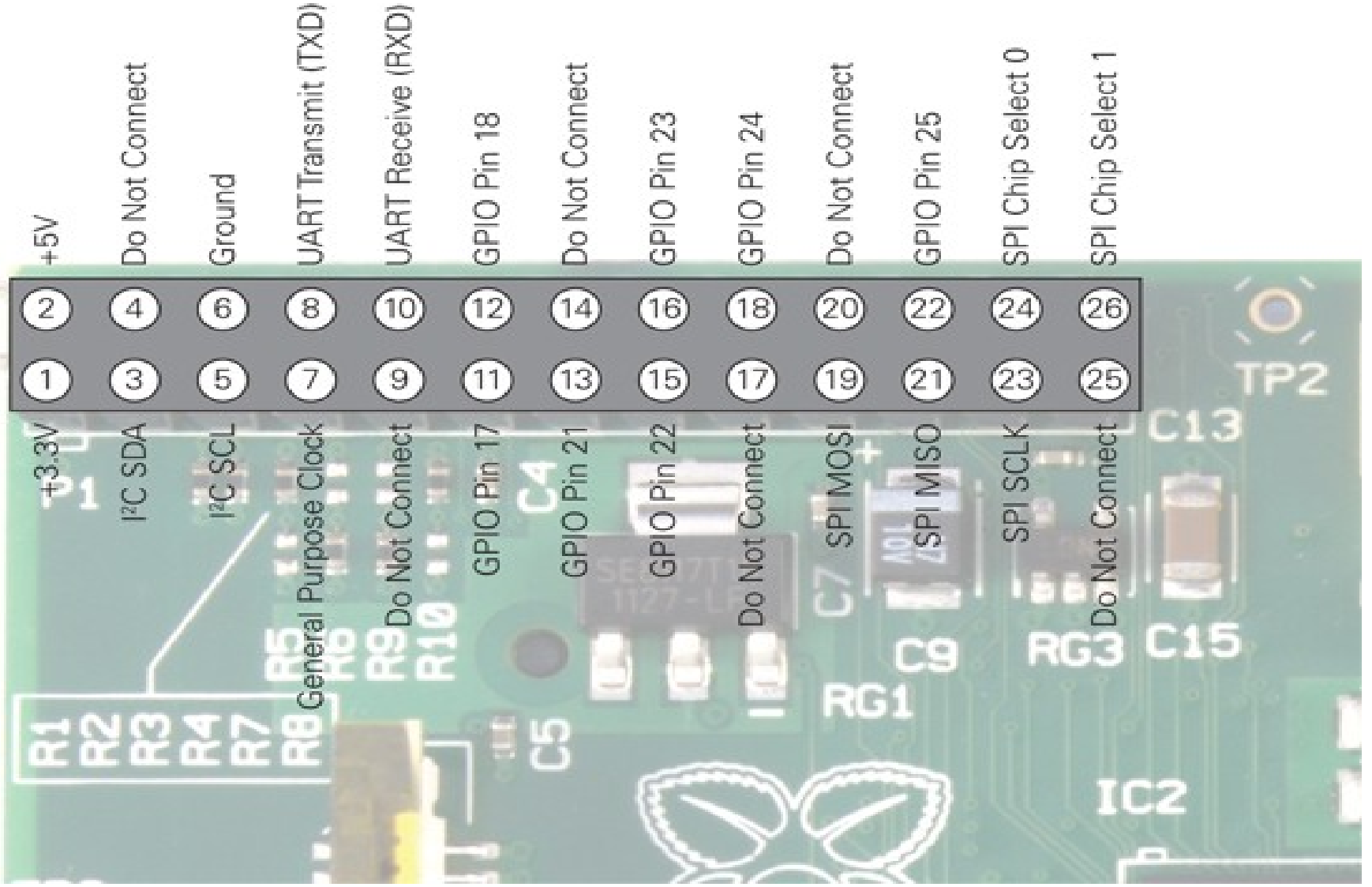
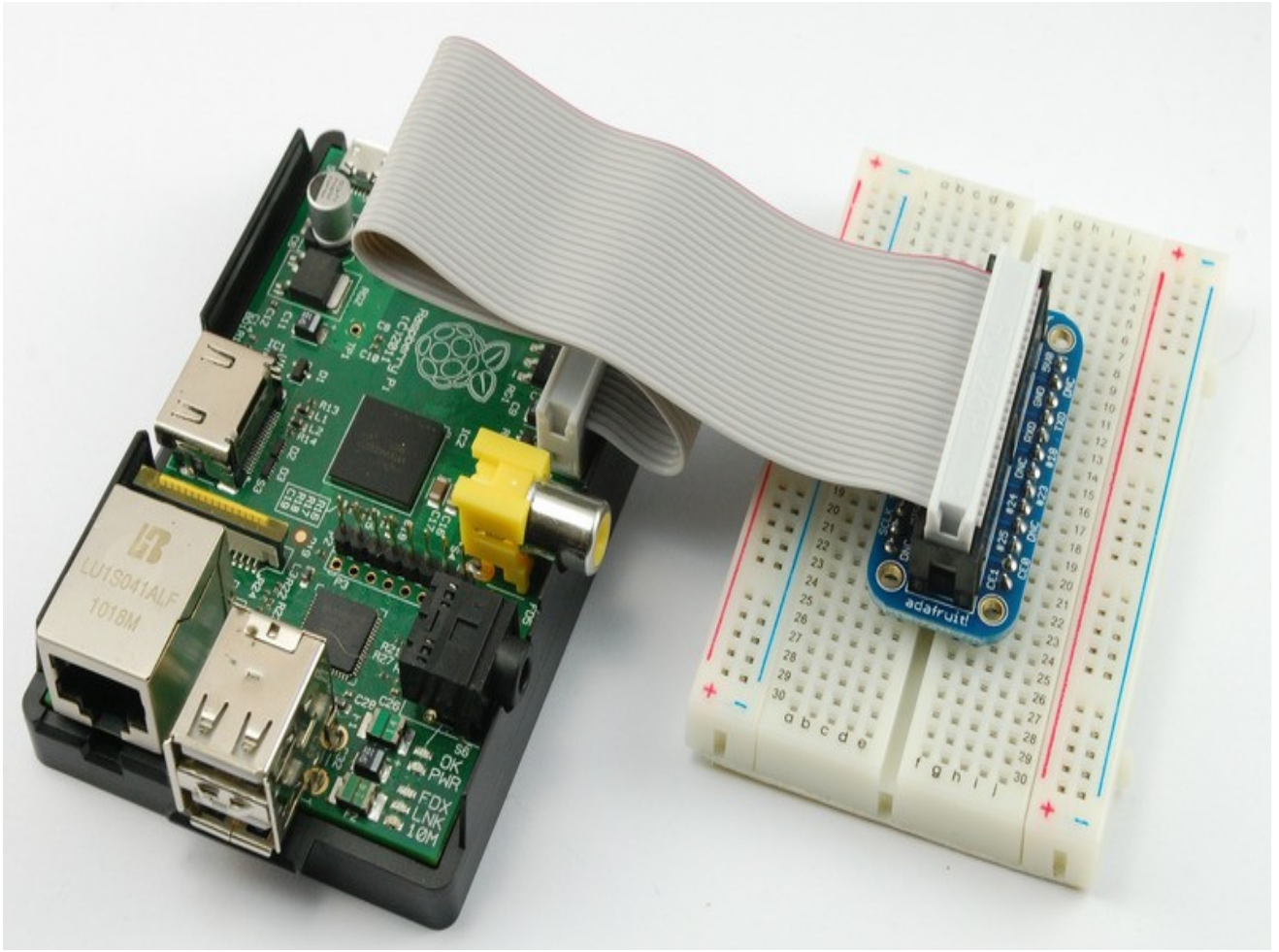
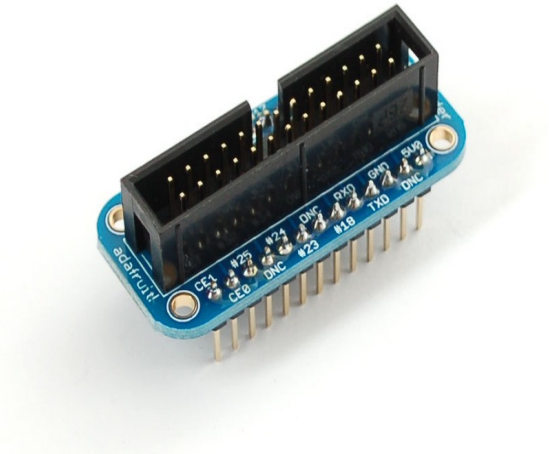
GPIO


+5V
Do Not Connect
Ground
UART Transmit (TXD)
UART Receive (RXD)
GPIO Pin 18
Do Not Connect
GPIO Pin 23
GPIO Pin 24
Do Not Connect
GPIO Pin 25
SPI Chip Select 0
SPI Chip Select 1
2
4
6
8
10
12
14
16
18
20
22
24
26
1
3
5
7
9
11
13
15
17
19
21
23
25
+3.3V
I²C SDA
I²C SCL
General Purpose Clock
Do Not Connect
GPIO Pin 17
GPIO Pin 21
GPIO Pin 22
Do Not Connect
SPI MOSI
SPI MISO
SPI SCLK
Do Not Connect
P1
R1
R2
R3
R4
R7
R8
R5
R6
R9
R10
C4
C5
C7
C9
C13
C15
RG1
RG3
IC2
TP2

RPi Setup for Today





- Because the cobbler connector has a notch, you can only put the cable in the right way
- But, it is possible to put the cable in upside down on the Raspberry Pi
  - The colored wire must connect to pin 1 in the Pi

Warming

- Connecting a 5 V supply to any pin on the Raspberry Pi's GPIO port, or directly shorting either of the power supply pins (Pin 1 and Pin 2) to any other pin will result in damage to the Pi.
- The Pi's internal logic operates at 3.3 V.
  - Devices designed for the Arduino may not work with the Pi unless a level translator or optical isolator is used between the two.
  - Connecting pins on a 5 V microcontroller directly to the Raspberry Pi's GPIO port will not work and may permanently damage the Pi.

RPi General Purpose IO (GPIO) Pins

- **17 GPIO pins** brought out onto the P1 header
  - most have alternated functions
  - two pins for UART; two for I2C; six for SPI
- All 17 pins can be GPIO (i.e., INPUT or OUTPUT)
  - all support interrupts
  - internal pull-ups & pull-downs for each pin
  - I2C pins have onboard pull-ups
    - using them for GPIO may not work

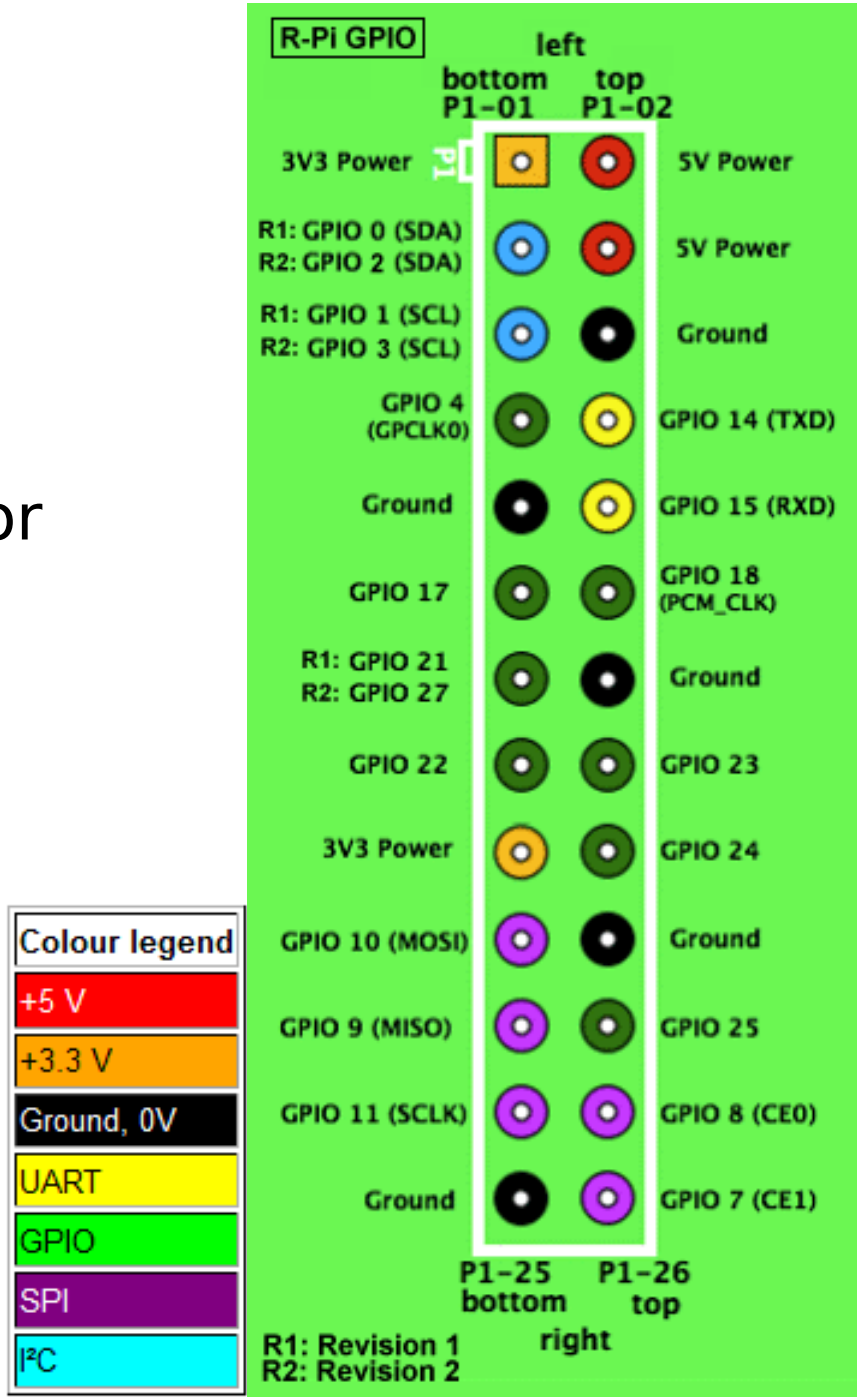


Image credit: http://elinux.org/RPi_Low-level_peripherals

The Bigger Picture

(image credit: http://pihw.wordpress.com/2013/01/30/sometimes-it-can-be-simple)

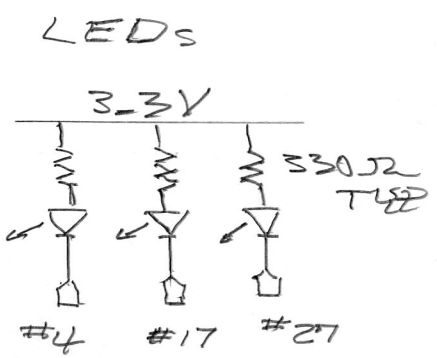
			P1				
<50mA	3V3		1	2		5V	
BCM GPIO00/02	SDA0/1	8	3	4		5V	
BCM GPIO01/03	SCL0/1	9	5	6		GND	
BCM GPIO04		7	7	8	15	TX	BCM GPIO14
	GND		9	10	16	RX	BCM GPIO15
BCM GPIO17		0	11	12	1	PWM0	BCM GPIO18
BCM GPIO21/27		2	13	14		GND	
BCM GPIO22		3	15	16	4		BCM GPIO23
<50mA	3v3		17	18	5		BCM GPIO24
BCM GPIO10	SPIMOSI	12	19	20		GND	
BCM GPIO9	SPIMOSO	13	21	22	6		BCM GPIO25
BCM GPIO11	SPI SCLK	14	23	24	10	SPI CE0 N	BCM GPIO08
	GND		25	26	11	SPI CE1 N	BCM GPIO07

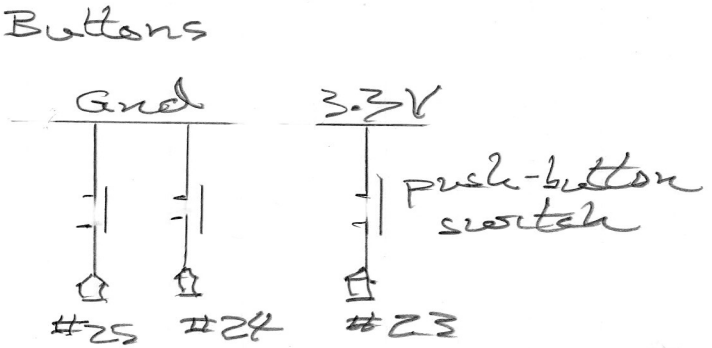
			P5				
<50mA	3V3		2	1		5V	
BCM GPIO29	SCL0	18	4	3	17	SDA0	BCM GPIO28
BCM GPIO31		20	6	5	19		BCM GPIO30
	GND		8	7		GND	

Diagram includes BCM GPIO references (GPIO.BCM), common functions, WiringPi pin references, and Pin numbers (GPIO.BOARD).

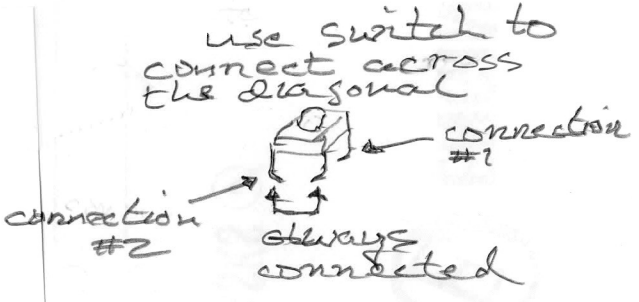
A cheat nice sheet

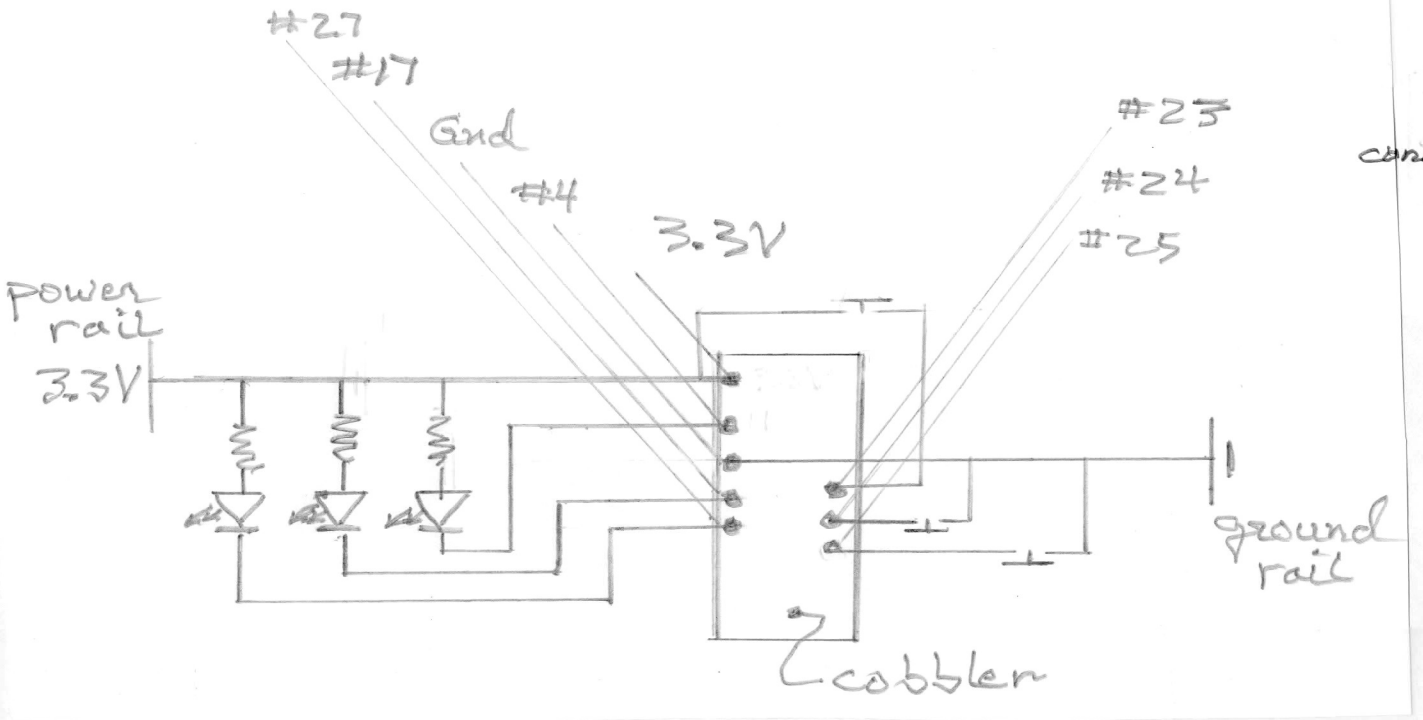
Wiring





push button switch





Using the GPIO Pins

There are two different methods to read or write these pins using Linux

- Method 1: Creating a file-type access in the file system using /sys
  - /sys is a way the kernel provides information about (physical and virtual) devices
- Method 2: Write/read memory addresses allocated to the GPIO peripheral of the SoC using pointers
  - Memory locations can be found in the datasheet for the BCM2835
  - RPi.GPIO
    - Comes with Raspbian
  - RPIO
    - Supports PWM & servos
  - WiringPi-python
  - Quick2Wire
- The best way to access the GPIOs?

Getting the Code

- If the hyperlinks don't work on the Pi, try this:
  - In a terminal window on the Pi, type the following command to download an archive file containing all of the scripts:

    wget http://www.cs.unca.edu/~bruce/Fall14/gpio.tar.gz
  - Uncompress the archive file using the following command:

    tar xfvz gpio.tar.gz
  - The archive file should uncompress into a directory named gpio containing 6 files

Method 1: using the /sys file system

- Download this shell script (i.e., *ledBash.sh*) and run using ***sudo***
  - Controlls the LEDs
- To use a GPIO pin in Bash, you have to do the following:
  - Export the pin number
  - Set the direction of the pin (input = “in” or output = “out”)
  - Initialise it to a safe/default state
  - Use it, by setting it to a value 0 or 1
  - Return it to a safe/default state (not required but good practise)
  - Release the pin when you are done with it

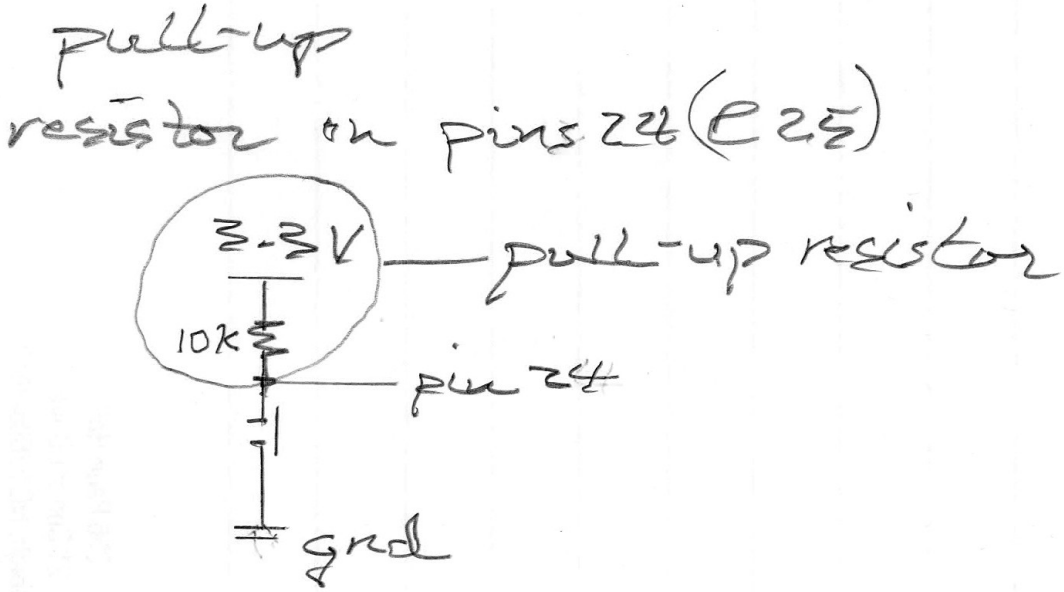
Understanding /sys/class/gpio/

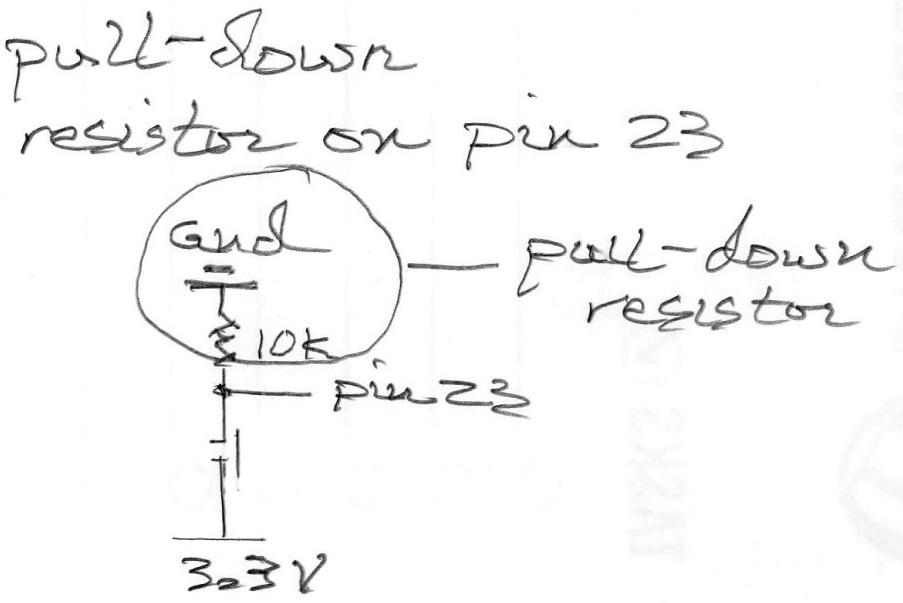
- In Linux, everything is a file: /dev/ttyUSB0, /sys/class/net/eth0/address, /dev/mmcblk0p2,...
- sysfs in a kernel module providing a virtual file system for device access at /sys/class
  - provides a way for users (or code in the *user-space*) to interact with devices at the system (kernel) level
- A demo
- Advantages / Disadvantage
  - Allows conventional access to pins from userspace
  - Always involves mode switch to kernel, action in kernel, mode switch to user, and could have a context switch
  - Much slower the digitalWrite()/digitalRead() of Arduino

Method 2 (maybe): Using the Python GPIO Library

- Download this file (i.e., *ledPython.py*) and run it using ***sudo***
- Notice the GPIO library functions
- Notice the programming style

Working with input: pull-down and pull-up resistors





Working with Input

- Begin by verifying your wiring on the buttons connected to pins 23 and 24
  - The button connected to pin 23 should connect to 3.3V when pushed
  - The button connected to pin 24 should connect to ground when pushed
- Polling example (i.e., *button1.py*)
- Wait function example (i.e., *button2.py*)
- Interrupt example (i.e., *button3.py*)

Putting it all together

- This script (i.e., *ledButton.py*) uses a single button to drive a single LED
- Modify the script so that three buttons are used to toggle three LEDs---each button controls one LED
  - Submit your modified script to Moodle before next Wednesday for full credit