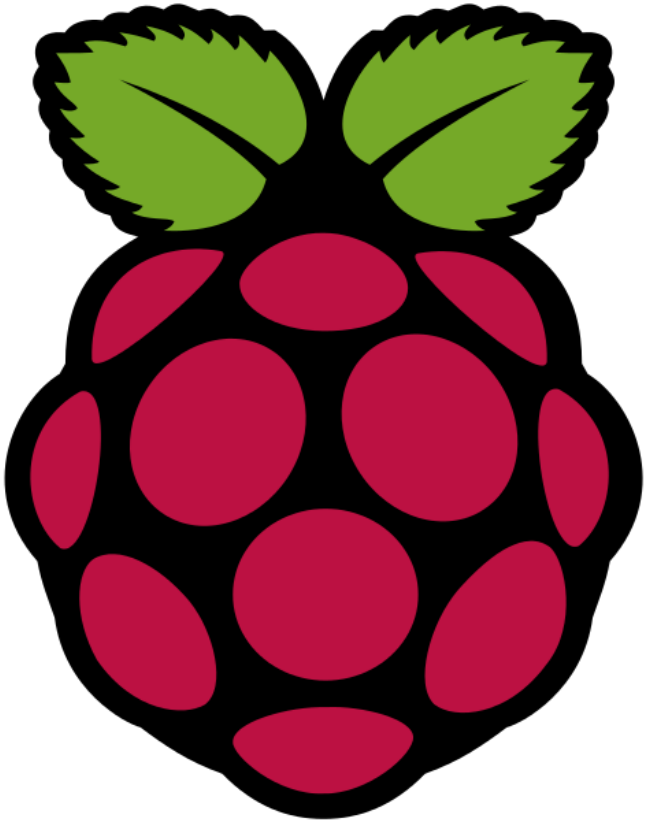
Spoken in Portuguese 



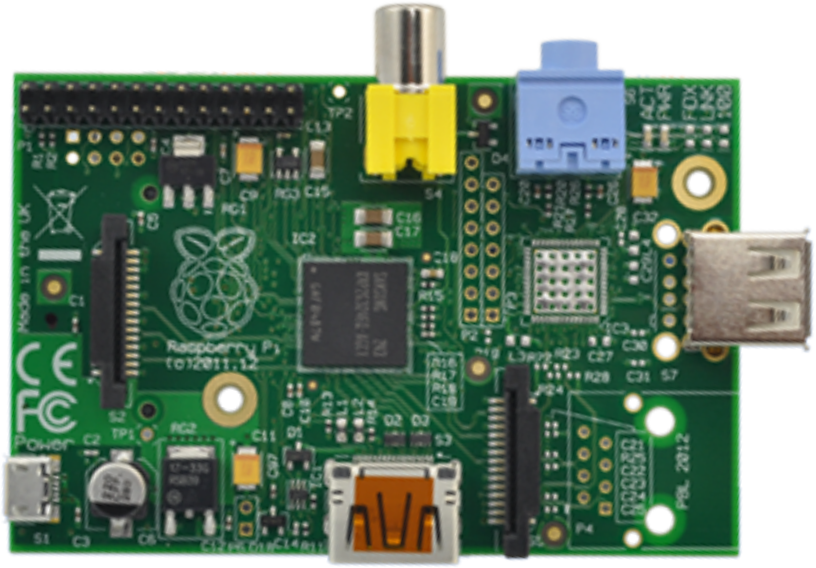
Connecting Electronics to the Raspberry Pi

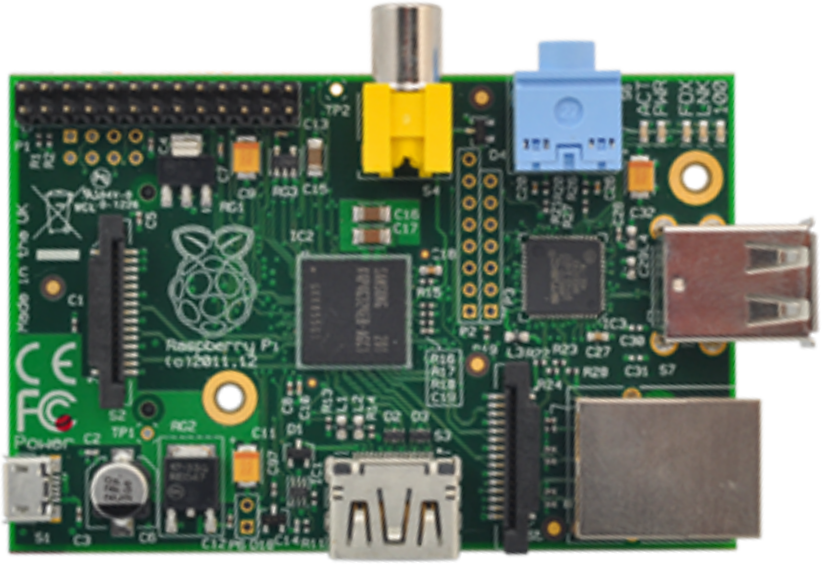
…by a non-Electrical Engineer

CODEBITS

Carlos Rodrigues – April 2014

The Raspberry Pi





Model A

- Cheaper
- Consumes less power

Model B

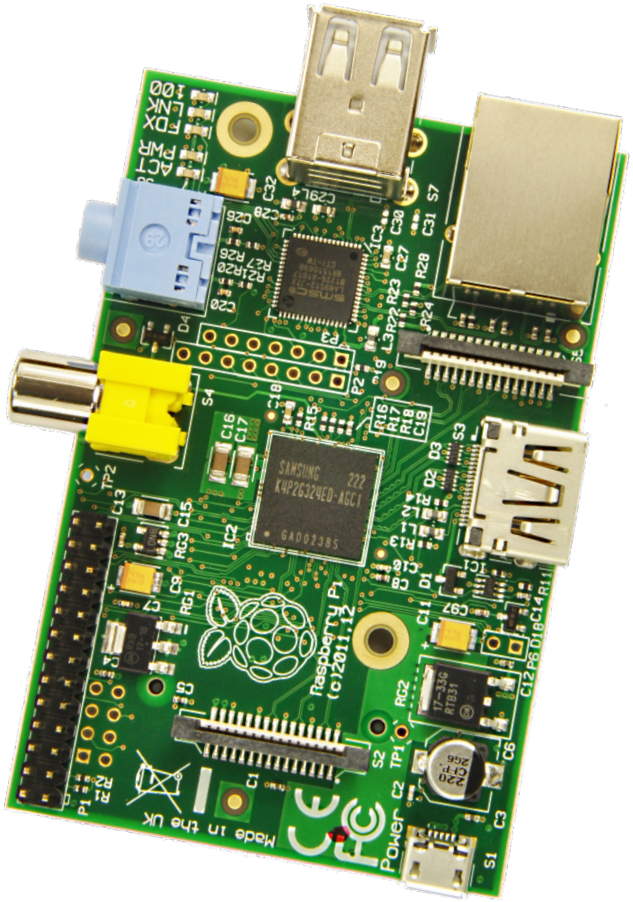
- More RAM
- Extra USB and Ethernet 10/100

Everything else is the same!

Same CPU, same GPIO capabilities, etc…

Board Overview

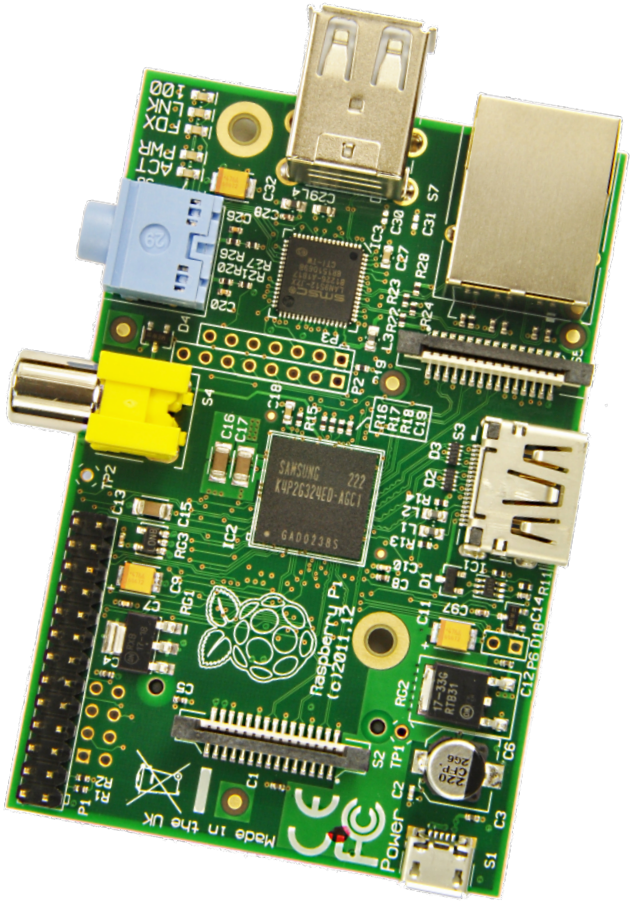
- Broadcom BCM2835 System-on-Chip
  - ARMv6 CPU @700MHz
  - VideoCore IV dual-core GPU @250MHz
- 512MB RAM (256MB on Model A)
- Audio and Video outputs
- USB (host only)
- Expansion connectors
  - GPIO*
  - CSI for high-definition camera
  - DSI for raw LCD display (unreleased)



*General-Purpose Input/Output

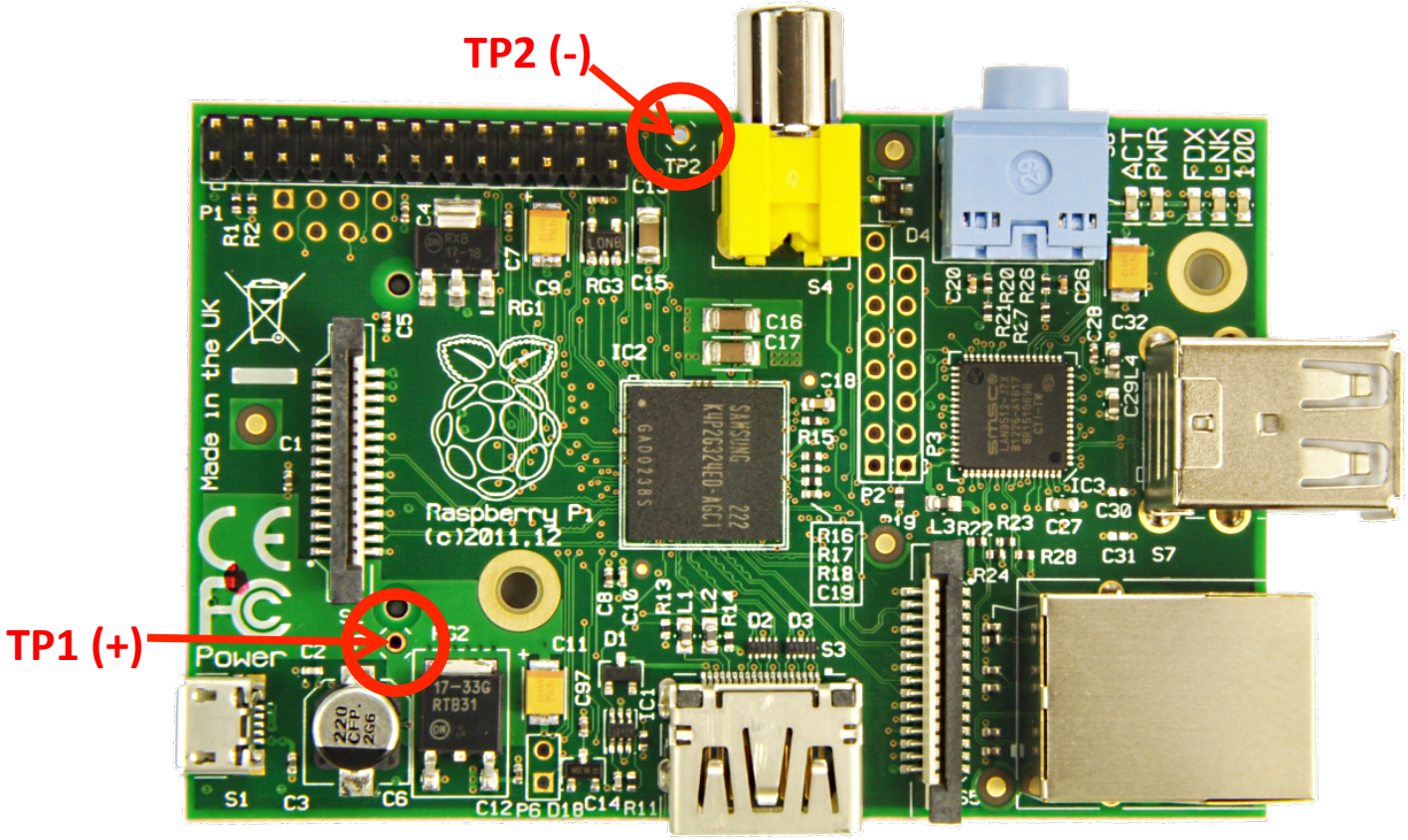
Electrical Overview

- System runs at **3.3V**
  - …although it is powered by 5V micro-USB
- Consumes 500mA (2.5W) on average*
  - …or half that for the Model A
  - …with a 700mA maximum
- Limited to ~1A total consumption
  - …including USB and GPIO power pins
  - …enforced by a power supply *polyfuse*



*Estimate, can vary depending on usage and connected USB devices

Power Supply Test Points

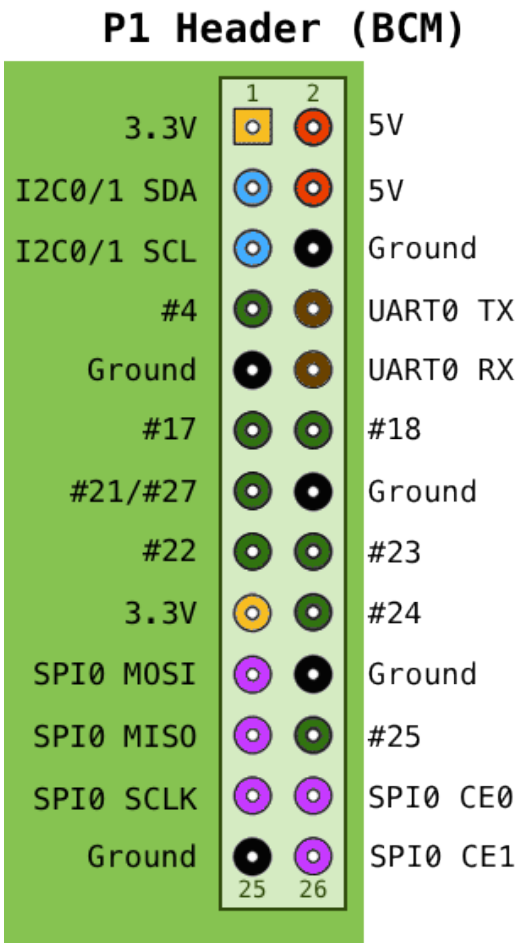


Voltage measurement between these two points should be 5V ± 5%

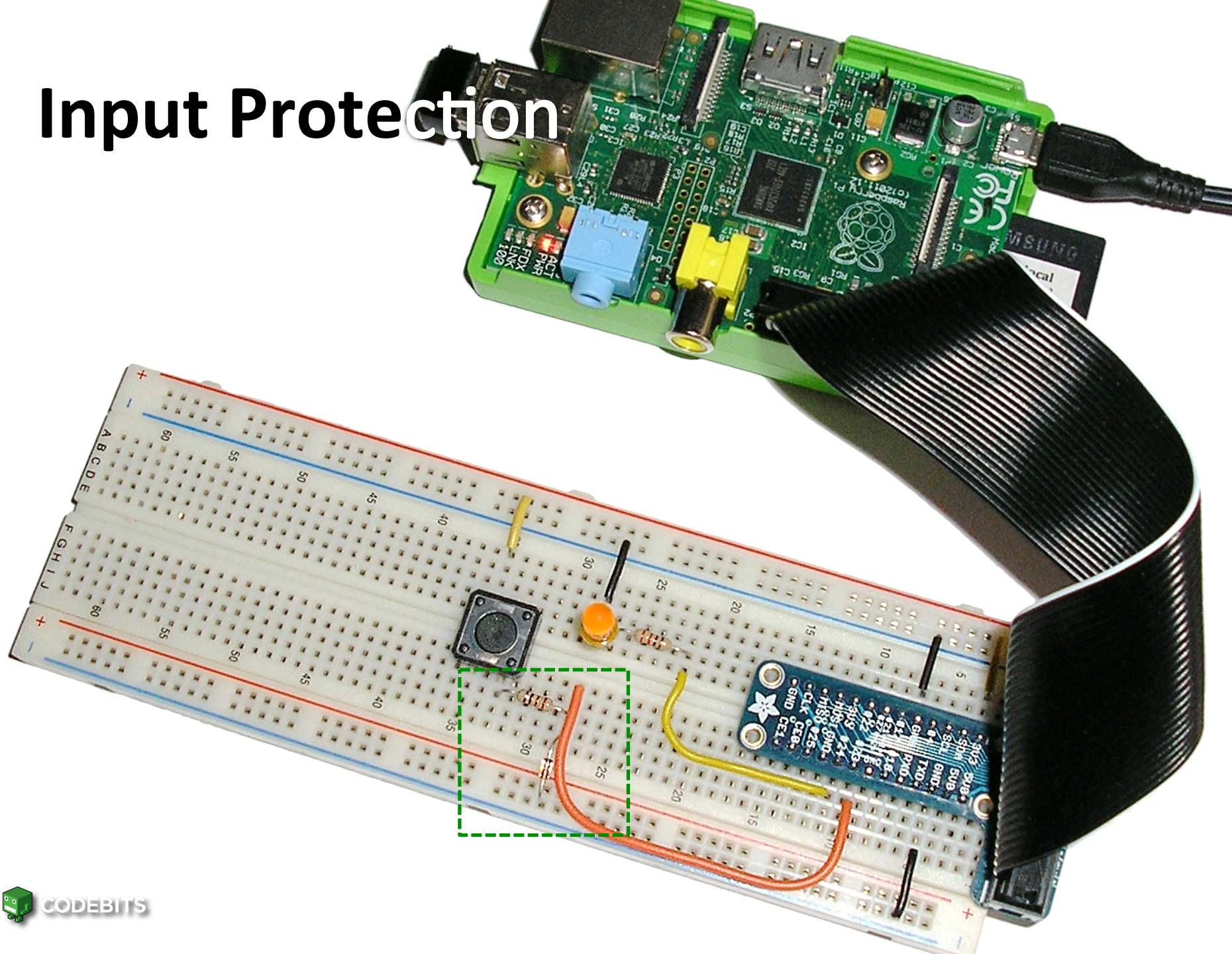
Less than this means you're drawing too much current…

The GPIO Connector

- 5V regulated power
  - …limited to whatever remains after system usage
- 3.3V regulated power
  - …limited to just **50mA** (total)
- Ground pins
- Digital I/O pins
  - …with almost **no protection** from harm
  - …each limited to **16mA** (but keep them under **3mA**)
- I²C bus
- SPI bus (with two channels)
- Serial port (**3.3V**)

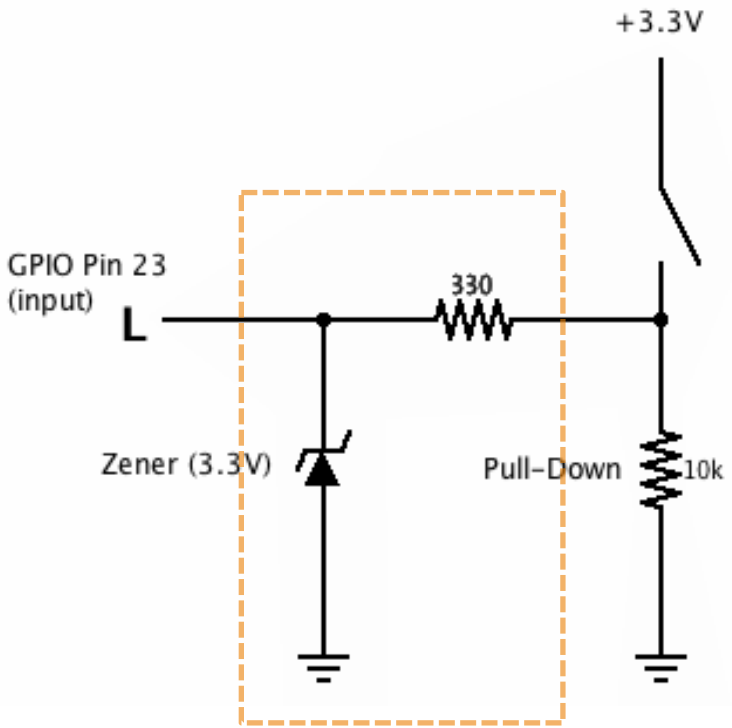


Input Protection



CODEBITS

Input Protection



- Input overvoltage protection*

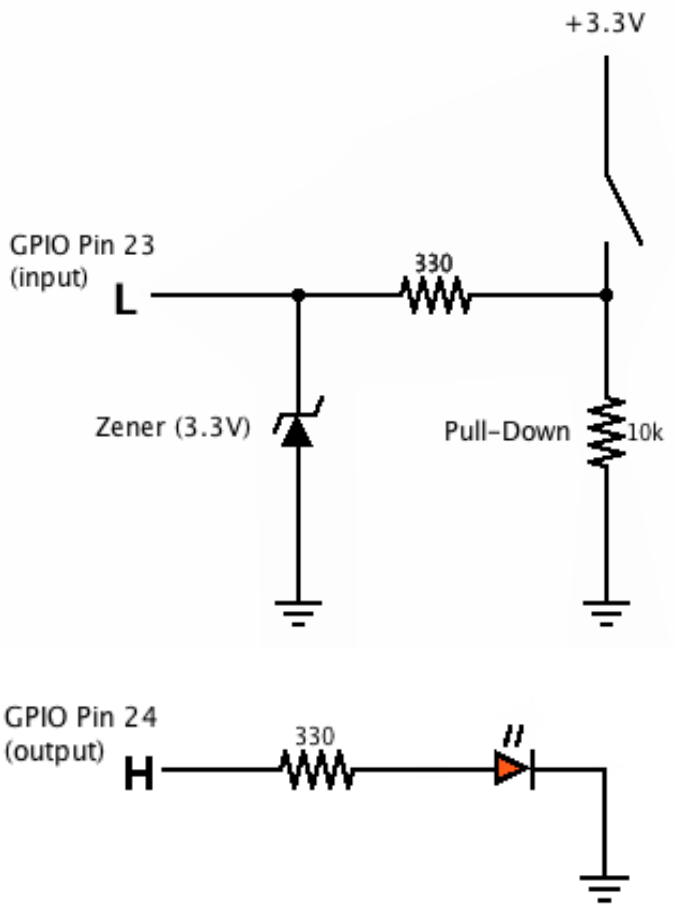
  …keeps it between -0.7V and +3.3V

- Safe "accidental output"

  …keeps pin current under 10mA

A ***zener diode*** is somewhat like a standard diode but with a precisely calibrated reverse breakdown voltage…

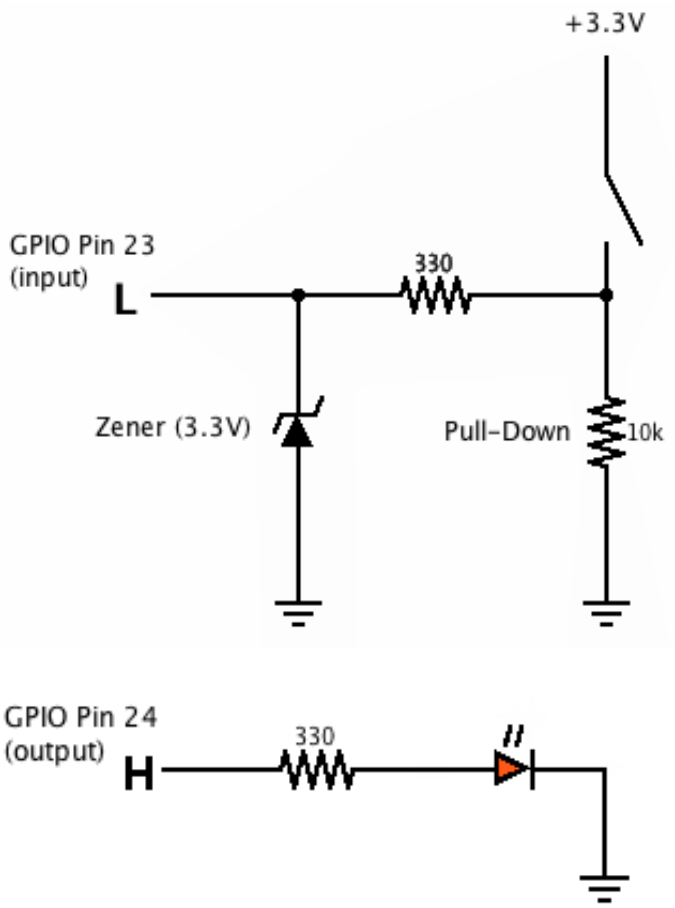
Hello, World!



Must run under **sudo** (needs **/dev/mem** access)

```
import signal, os, time
import RPi.GPIO as gpio

BUTTON_PIN = 23
LED_PIN = 24

def handle_signals(signal, frame):
  gpio.cleanup()
  os._exit(0)

def button_released(pin):
  gpio.output(LED_PIN, not gpio.input(LED_PIN))

gpio.setmode(gpio.BCM)

signal.signal(signal.SIGINT, handle_signals)
signal.signal(signal.SIGTERM, handle_signals)

try:
  gpio.setup(BUTTON_PIN, gpio.IN)
  gpio.setup(LED_PIN, gpio.OUT, initial=gpio.LOW)

  gpio.add_event_detect(BUTTON_PIN, gpio.FALLING,
                        callback=button_released,
                        bouncetime=200)  # ms
  while True:
      time.sleep(1)
finally:
  gpio.cleanup()
```

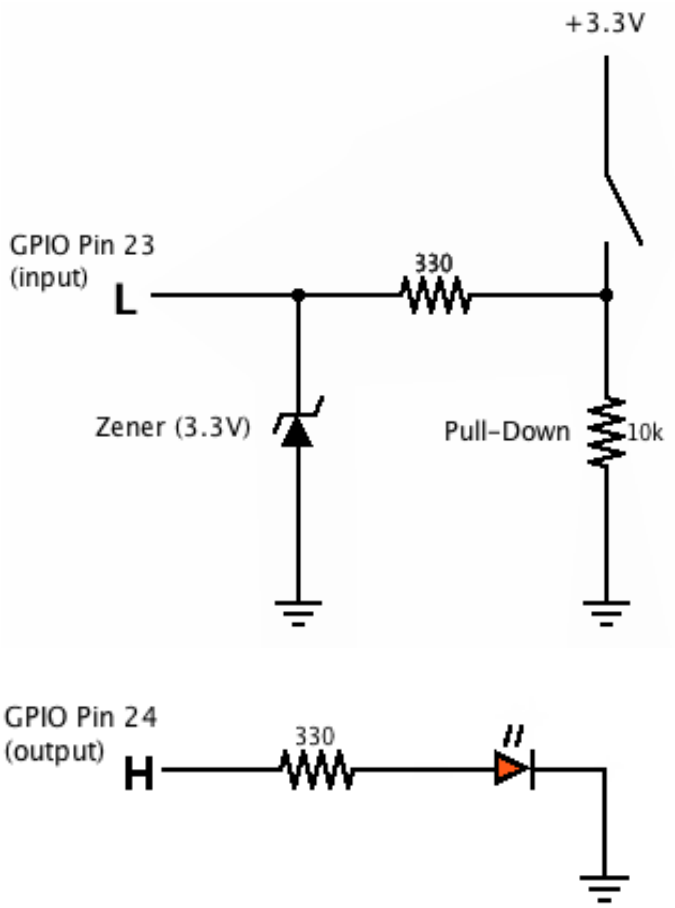
Hello, World!



Must run under **sudo** (needs **/dev/mem** access)

```
import signal, os, time
import RPi.GPIO as gpio

BUTTON_PIN = 23
LED_PIN = 24

def handle_signals(signal, frame):
  gpio.cleanup()
  os._exit(0)

def button_released(pin):
  gpio.output(LED_PIN, not gpio.input(LED_PIN))

gpio.setmode(gpio.BCM)

signal.signal(signal.SIGINT, handle_signals)
signal.signal(signal.SIGTERM, handle_signals)

try:
  gpio.setup(BUTTON_PIN, gpio.IN)
  gpio.setup(LED_PIN, gpio.OUT, initial=gpio.LOW)

  gpio.add_event_detect(BUTTON_PIN, gpio.FALLING,
                        callback=button_released,
                        bouncetime=200)  # ms
  while True:
      time.sleep(1)
finally:
  gpio.cleanup()
```

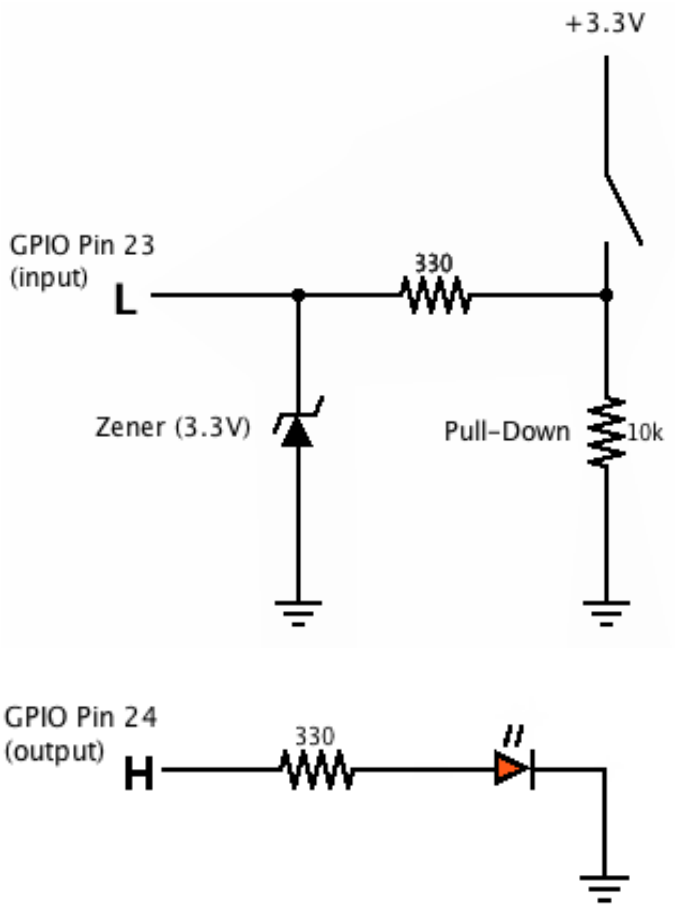
Hello, World!



Must run under **sudo** (needs **/dev/mem** access)

```
import signal, os, time
import RPi.GPIO as gpio

BUTTON_PIN = 23
LED_PIN = 24

def handle_signals(signal, frame):
  gpio.cleanup()
  os._exit(0)

def button_released(pin):
  gpio.output(LED_PIN, not gpio.input(LED_PIN))

gpio.setmode(gpio.BCM)

signal.signal(signal.SIGINT, handle_signals)
signal.signal(signal.SIGTERM, handle_signals)

try:
  gpio.setup(BUTTON_PIN, gpio.IN)
  gpio.setup(LED_PIN, gpio.OUT, initial=gpio.LOW)

  gpio.add_event_detect(BUTTON_PIN, gpio.FALLING,
                        callback=button_released,
                        bouncetime=200)  # ms
  while True:
      time.sleep(1)
finally:
  gpio.cleanup()
```

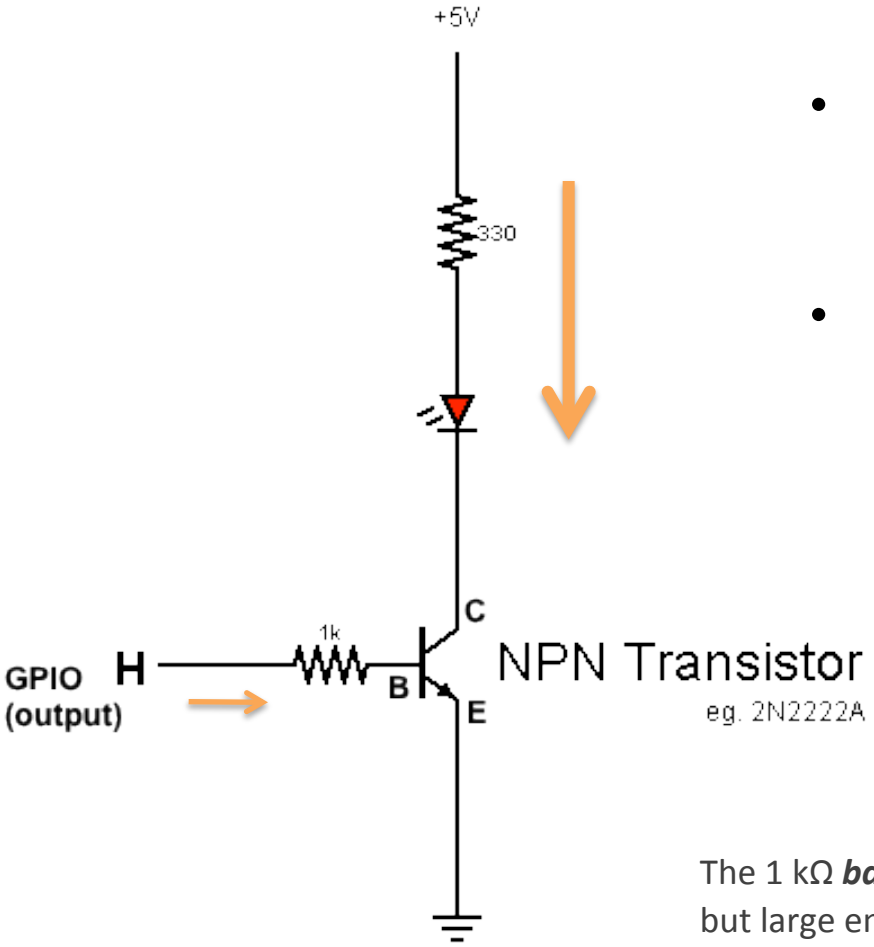
Hello, World!



Must run under **sudo** (needs **/dev/mem** access)

```
import signal, os, time
import RPi.GPIO as gpio

BUTTON_PIN = 23
LED_PIN = 24

def handle_signals(signal, frame):
  gpio.cleanup()
  os._exit(0)

def button_released(pin):
  gpio.output(LED_PIN, not gpio.input(LED_PIN))

gpio.setmode(gpio.BCM)

signal.signal(signal.SIGINT, handle_signals)
signal.signal(signal.SIGTERM, handle_signals)

try:
  gpio.setup(BUTTON_PIN, gpio.IN)
  gpio.setup(LED_PIN, gpio.OUT, initial=gpio.LOW)

  gpio.add_event_detect(BUTTON_PIN, gpio.FALLING,
                        callback=button_released,
                        bouncetime=200)  # ms
  while True:
      time.sleep(1)
finally:
  gpio.cleanup()
```

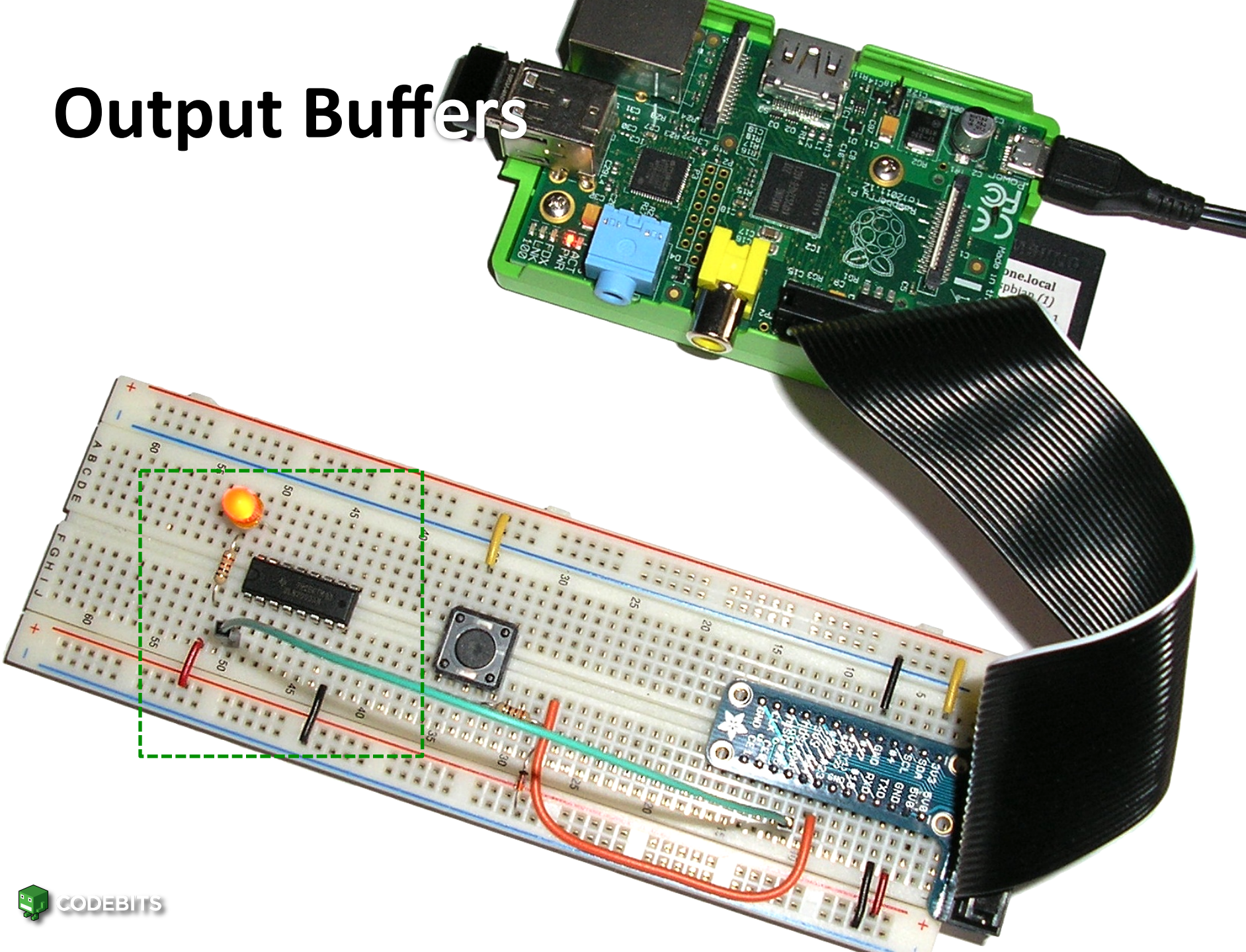
Output Buffers



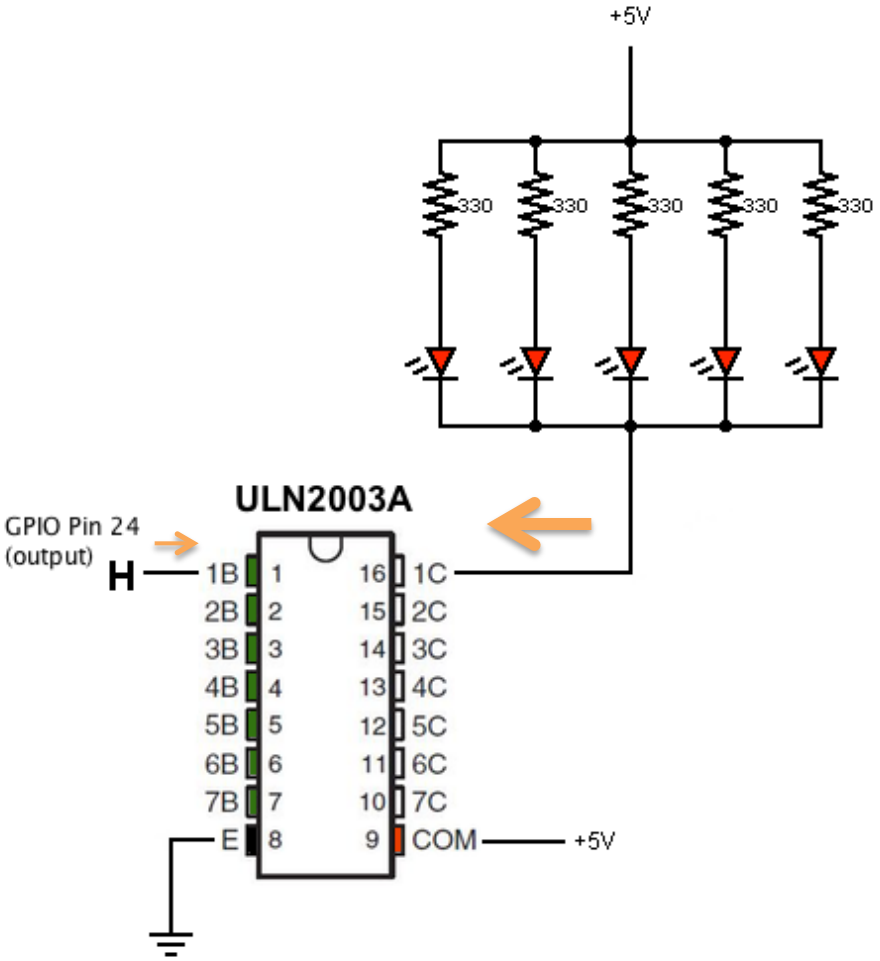
- **Transfers** load out of the pin and into the transistor (keep pin current small)
- **Isolates** load voltage from the pin voltage (can drive more than 3.3V)

The 1 kΩ ***base resistor*** keeps the current small to protect the pin, but large enough for the transistor to turn fully on…

Output Buffers

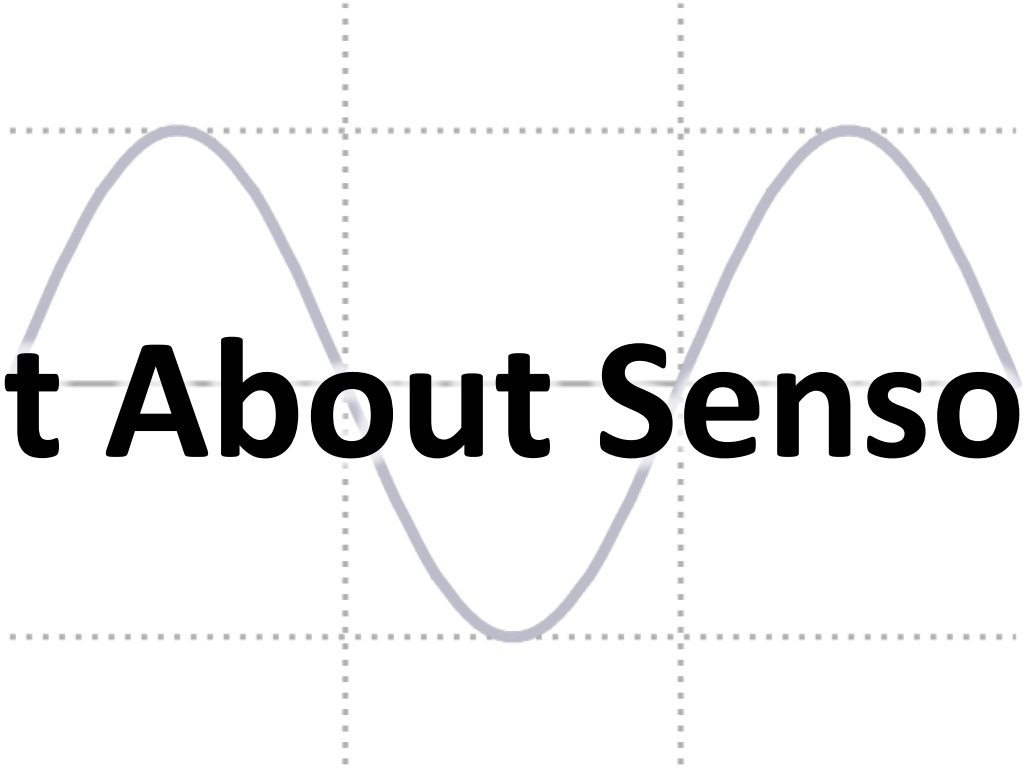


Output Buffers



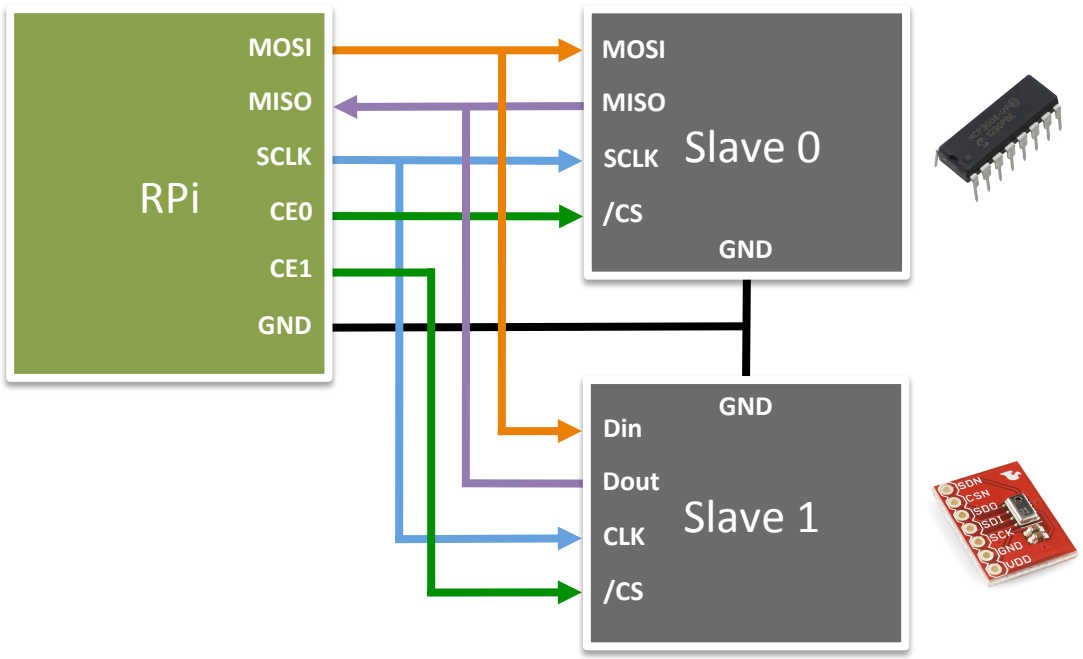
- Output can go up to 50V
  …driving 5V electronics is just fine
- Handles up to 300mA per-output*
  …but no more than 600mA overall
- Can handle inductive loads
  …such as DC motors and relays
  …that's the purpose of the COM pin

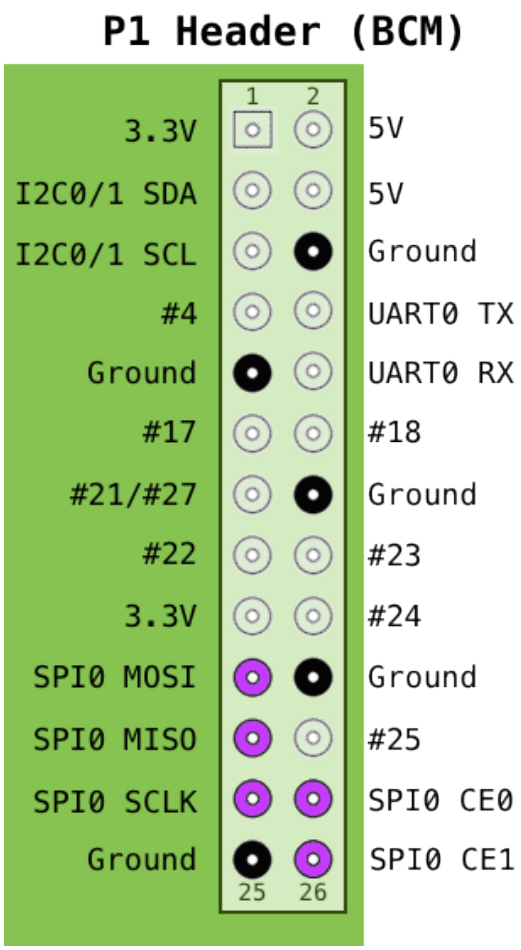
*Specs say 500mA, but that's for 5V logic inputs, not 3.3V



What About Sensors…?

The SPI Bus

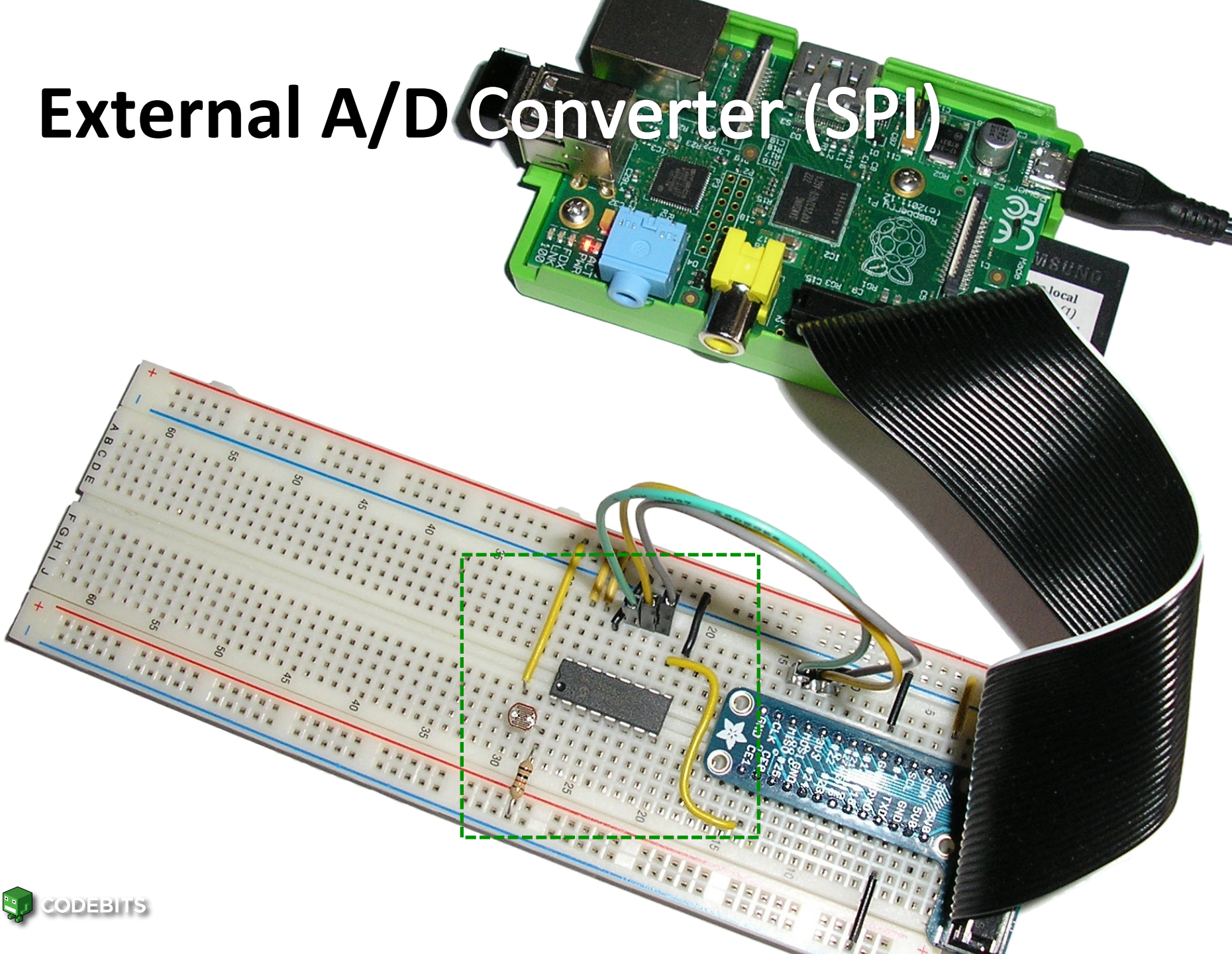




Data transfers are always symmetrical

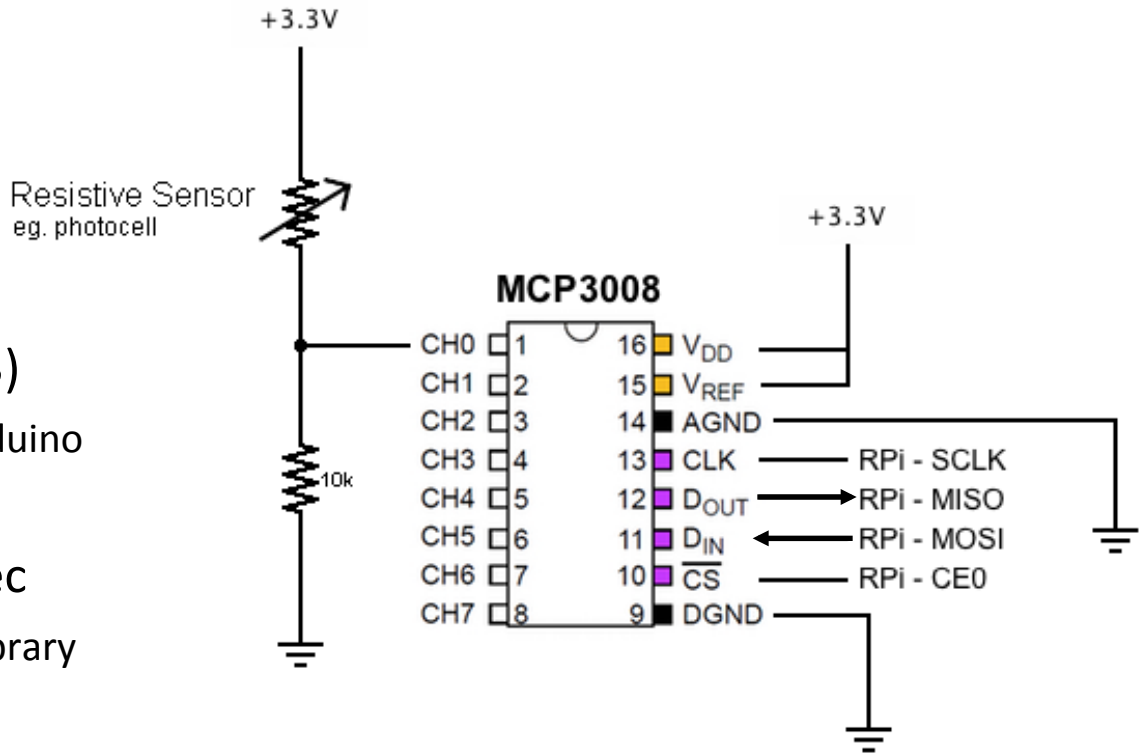
To receive a byte you must send a byte (and vice-versa)…

External A/D Converter (SPI)

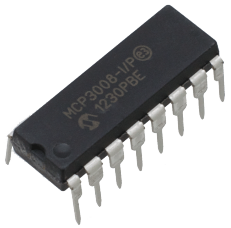


CODEBITS

External A/D Converter (SPI)



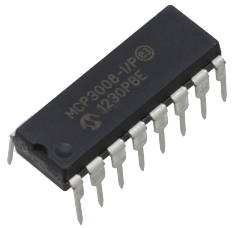
- 10-bit resolution (0-1023)
  …same as you get with an Arduino
- At least 4000 samples/sec
  …with the “python-spidev” library
- Consumes just a few μA
  …the Raspberry Pi can only provide **50mA** at 3.3V



CODEBITS

External A/D Converter (SPI)

```
import time
import spidev

SPI_PORT = 0
SPI_CHANNEL = 0
SENSOR_PORT = 0

spi = spidev.SpiDev()
spi.open(SPI_PORT, SPI_CHANNEL)  # opens "/dev/spidev0.0"

try:
    while True:
        data = spi.xfer2([0x01, 0x80 + (SENSOR_PORT << 4), 0x00])  # …uh??
        value = ((data[1] & 0x03) << 8) + data[2]

        print(value)
        time.sleep(0.2)

except KeyboardInterrupt:
    spi.close()
```

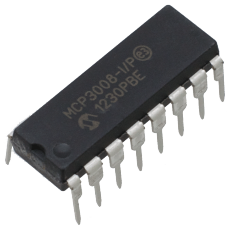


Must remove the **spi-bcm2708** module out of **/etc/modprobe.d/raspi-blacklist.conf**

External A/D Converter (SPI)

```
import time
import spidev

SPI_PORT = 0
SPI_CHANNEL = 0
SENSOR_PORT = 0

spi = spidev.SpiDev()
spi.open(SPI_PORT, SPI_CHANNEL)  # opens "/dev/spidev0.0"

try:
    while True:
        data = spi.xfer2([0x01, 0x80 + (SENSOR_PORT << 4), 0x00])
        value = ((data[1] & 0x03) << 8) + data[2]

        print(value)
        time.sleep(0.2)

except KeyboardInterrupt:
    spi.close()
```

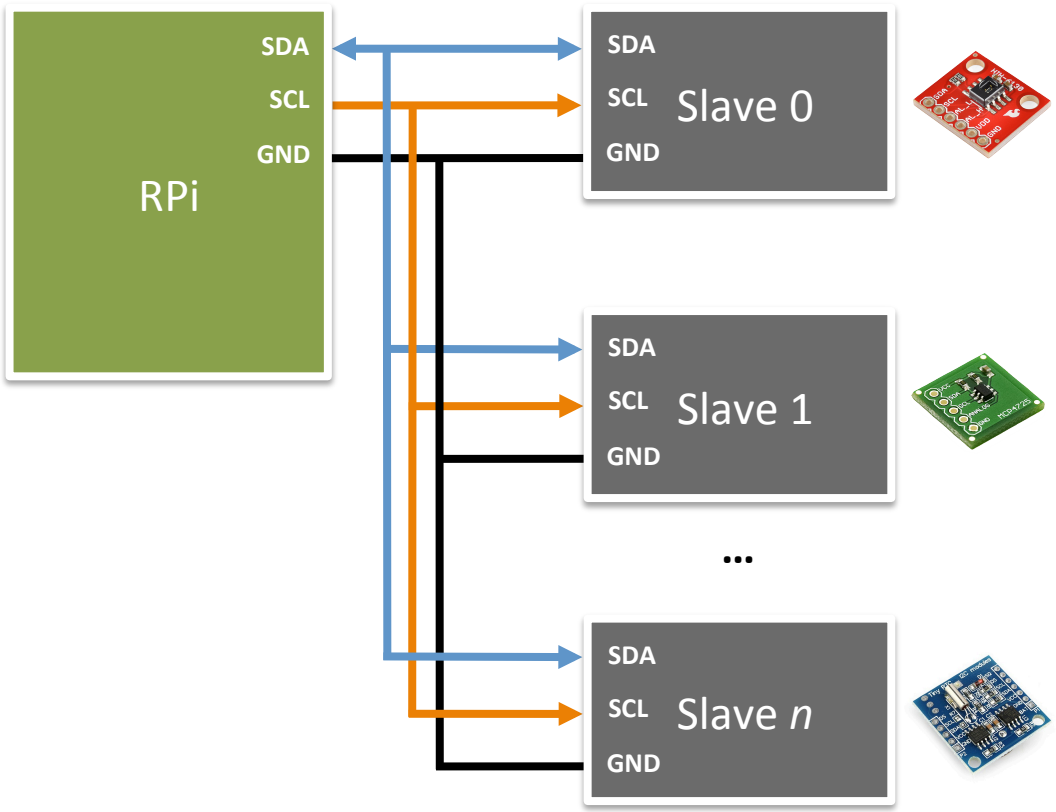
Transmitted	Received
00000001	....
1nnn0000	vv
00000000	vvvvvvvv

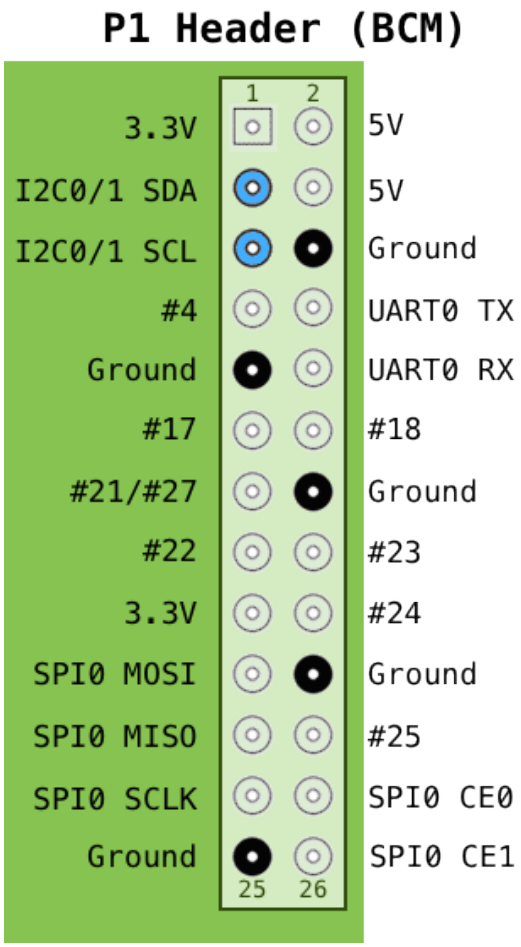


Must remove the **spi-bcm2708** module out of **/etc/modprobe.d/raspi-blacklist.conf**

CODEBITS

The I²C Bus*



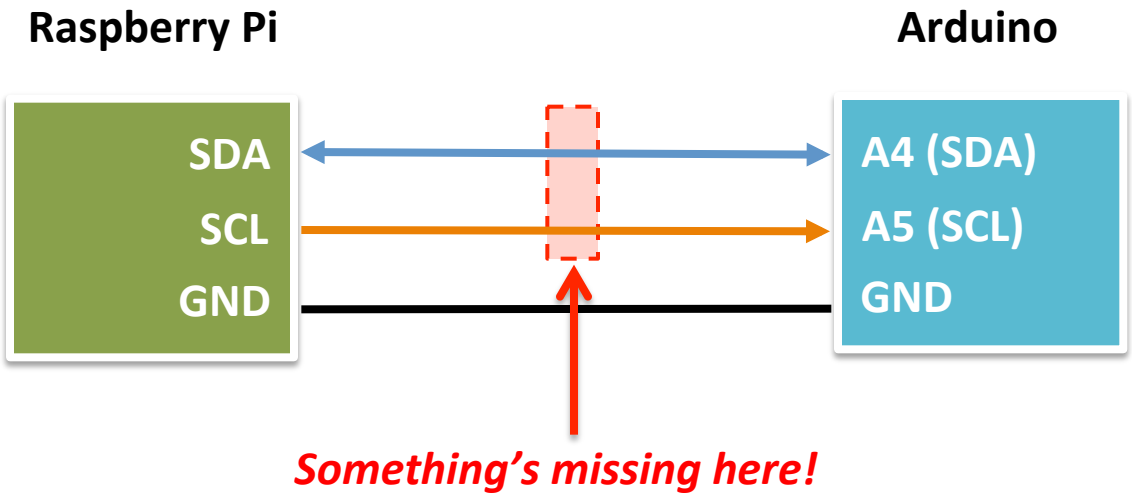


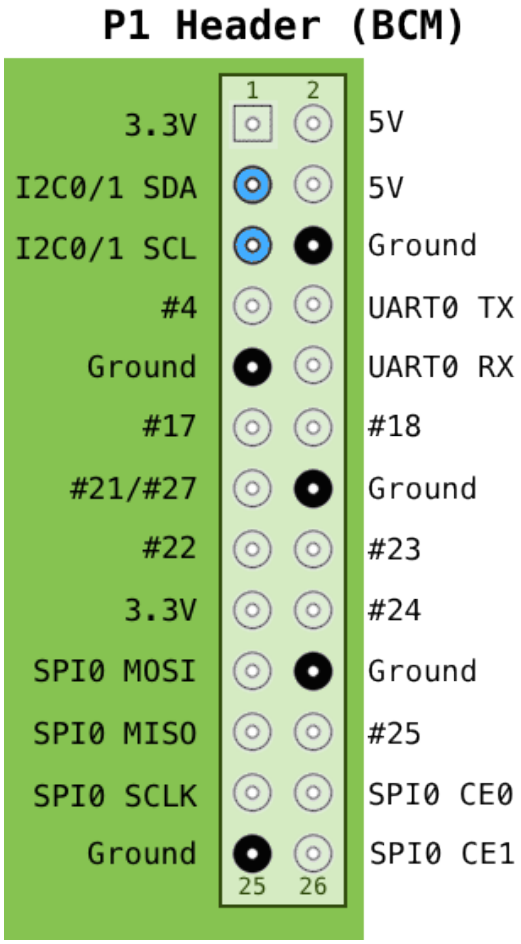
Up to 127 devices can share a single I²C bus

These can be sensors, accelerometers, DACs, RTCs, etc…

*Also known as SMBus or TWI

The I²C Bus

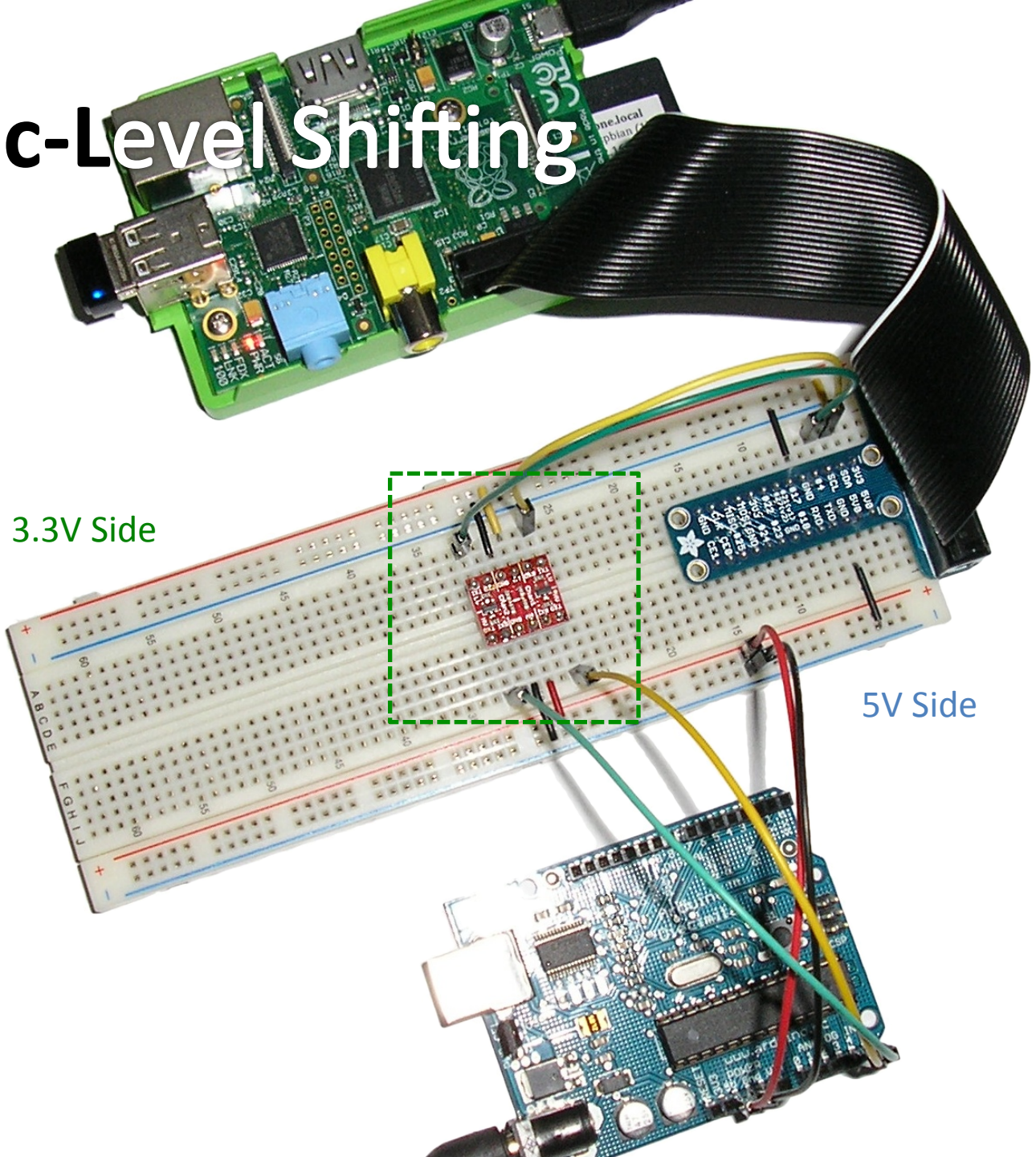




There are 3.3V Arduinos, but most models are <u>5V</u>

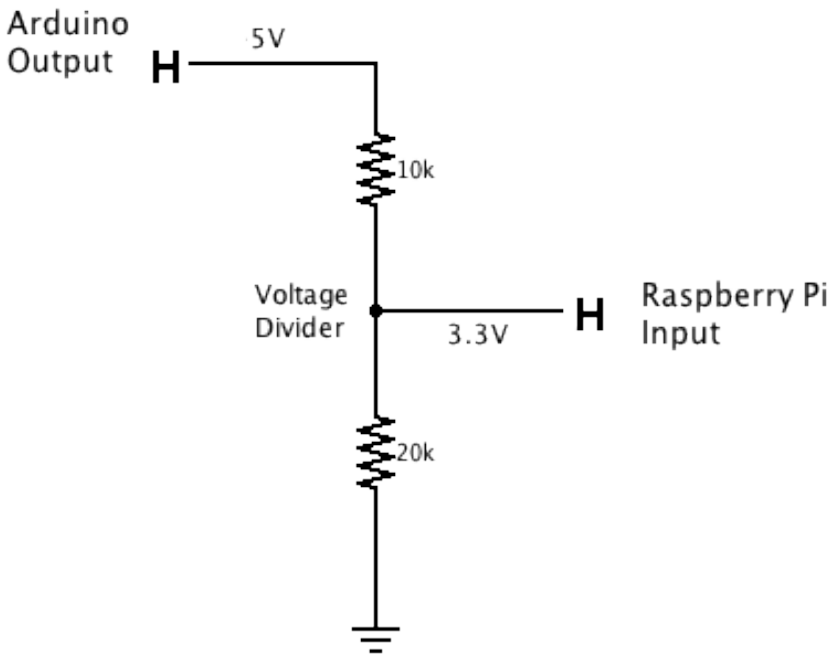
We need something to isolate 3.3V parts from 5V parts…

Logic-Level Shifting

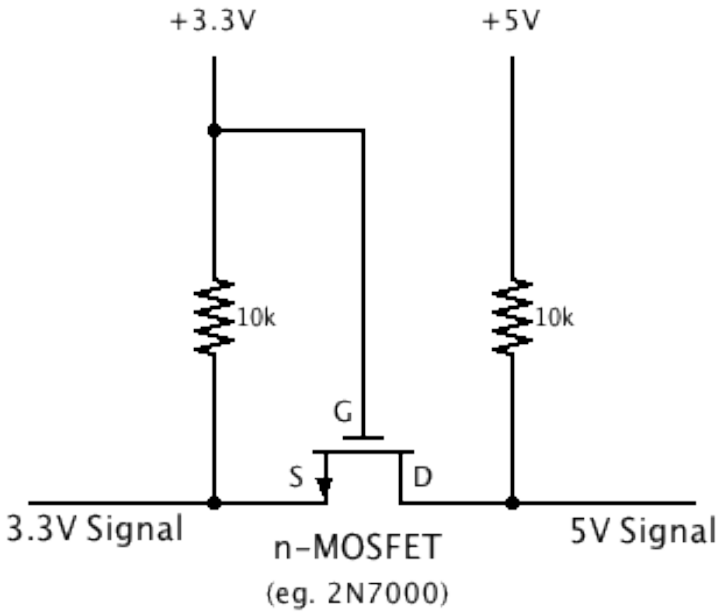


Logic-Level Shifting

Uni-directional

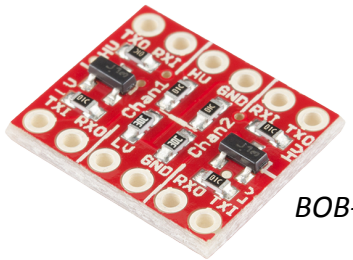


Bi-directional



You can find this in a Sparkfun Logic-Level Converter

Only the leftmost and rightmost channels (“TX”) are bi-directional…



BOB-11978

I²C Hello, World!

Raspberry Pi

```
import smbus
from time import sleep

ADDRESS = 0x03  # …the I²C slave

i2c = smbus.SMBus(1)  # …I²C bus #1

while True:
  try:
    b = i2c.read_byte(ADDRESS)

    print("read: %d" % b)
    b += 1

    i2c.write_byte(ADDRESS, b)

  except IOError as e:
    print("IOError: %s" % e)
    continue

  finally:
    sleep(1)
```

Arduino

```
#include <Wire.h>

byte b = 0;

void setup() {
  Wire.begin(0x03);  // …address

  Wire.onRequest(outgoing);
  Wire.onReceive(incoming);
}

void outgoing() {
  Wire.write(b);
}

void incoming(int bytes) {
  b = Wire.read();
}

void loop() {}
```

Must remove the **i2c-bcm2708** module out of **/etc/modprobe.d/raspi-blacklist.conf**
Must add an **i2c-dev** line to **/etc/modules**

I²C Hello, World!

Raspberry Pi

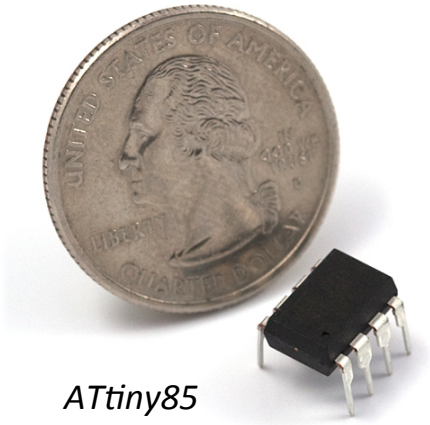
```
import smbus
from time import sleep

ADDRESS = 0x03  # …the I²C slave

i2c = smbus.SMBus(1)  # …I²C bus #1

while True:
  try:
    b = i2c.read_byte(ADDRESS)

    print("read: %d" % b)
    b += 1

    i2c.write_byte(ADDRESS, b)

  except IOError as e:
    print("IOError: %s" % e)
    continue

  finally:
    sleep(1)
```

Arduino

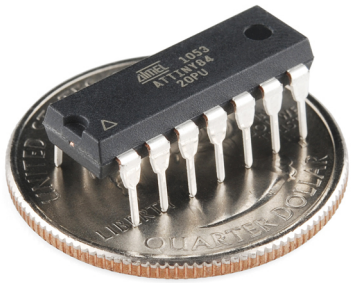
```
#include <Wire.h>

byte b = 0;

void setup() {
  Wire.begin(0x03);  // …address

  Wire.onRequest(outgoing);
  Wire.onReceive(incoming);
}

void outgoing() {
  Wire.write(b);
}

void incoming(int bytes) {
  b = Wire.read();
}

void loop() {}
```

Must remove the **i2c-bcm2708** module out of **/etc/modprobe.d/raspi-blacklist.conf**
Must add an **i2c-dev** line to **/etc/modules**

Bare Microcontroller Chips

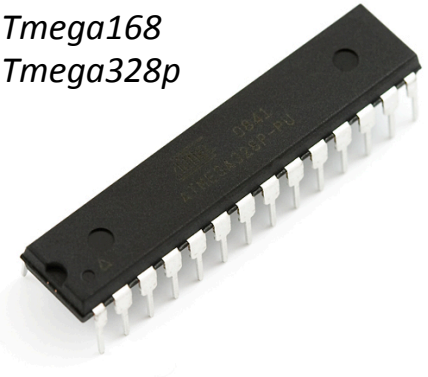


ATtiny85



ATtiny84

ATmega168
ATmega328p



Can even be programmed from the Arduino IDE!

You need an ISP programmer, but a regular Arduino running the “ArduinoISP” sketch is enough.

github.com/carlosefr/atmega – ATmega with no external components from the Arduino IDE
highlowtech.org/?p=1695 – ATtiny from the Arduino IDE
github.com/rambo/TinyWire – software assisted I^2C on the ATtiny*

* This might need the bus to be slowed down: **echo "options i2c_bcm2708 baudrate=10000" > /etc/modprobe.d/i2c.conf**

Thanks for Listening!

Any Questions?

This presentation and related files can be found at:
cloud.carlos-rodrigues.com/codebits/2014.zip

Carlos Rodrigues

cefrodrigues@gmail.com
twitter.com/carlosefr