

# Raspberry Pi Hacking

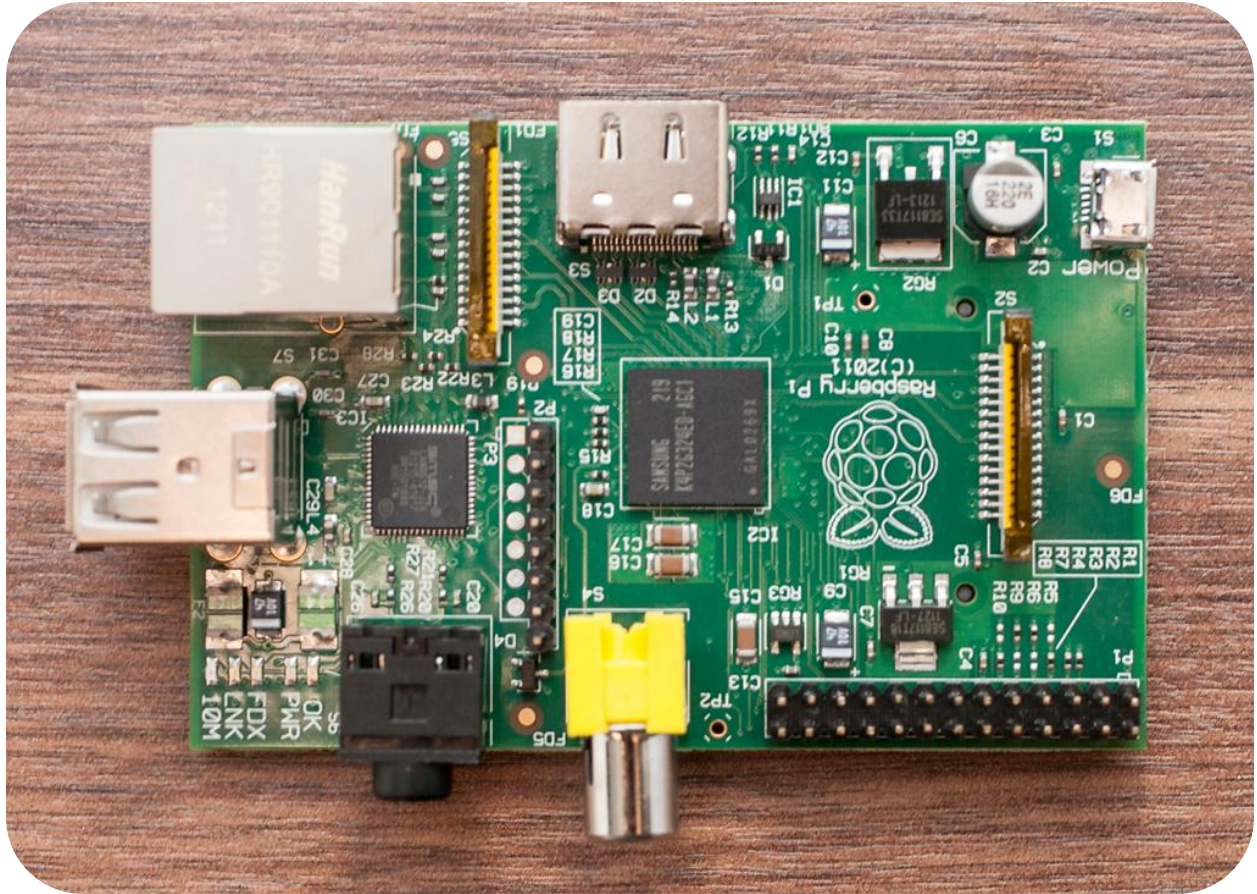
---

Loving your pi and hacking it too...

By: magikh0e

[Http://magikh0e.ihtb.org](http://magikh0e.ihtb.org)

Last Update: **September 2 2012**



Hacking the Raspberry Pi and hacking things with the Raspberry Pi.  
*Best of both worlds...*

## Disclaimer

Follow this guide at your own risk. I take/accept not responsibility for any outcome from anything you attempt to do within this guide. Everything is in a “works for me” state. ;)

## Contents

|   |    |
|---|----|
| Raspberry Pi .....                          | 3  |
| What is a Raspberry Pi? .....               | 3  |
| What are the dimensions? .....              | 3  |
| Raspberry Pi Specs – Model B .....          | 4  |
| Tweaking Raspberry Pi’s Performance .....   | 4  |
| RAM Usage .....                             | 4  |
| Disk Tuning .....                           | 6  |
| CPU – Over Clocking .....                   | 9  |
| Hacking stuff with the Pi .....             | 10 |
| GPIO Introduction .....                     | 11 |
| What is GPIO? .....                         | 11 |
| Raspberry Pi GPIO .....                     | 11 |
| Pin Diagram – Names & Alt 0 Functions ..... | 12 |
| Power Pins .....                            | 14 |
| Protecting your pins and your Pi .....      | 14 |
| GPIO – Interaction .....                    | 15 |
| WiringPi .....                              | 15 |
| Programming Libraries .....                 | 16 |
| Code Examples .....                         | 17 |
| GPIO – External Applications .....          | 22 |
| Interfacing With a Teensy Kit .....         | 22 |
| Interfacing with LCD Displays .....         | 23 |
| MCP23017 I2C I/O Expander .....             | 29 |
| Required drivers and software .....         | 29 |
| Connecting the expander to the Pi .....     | 29 |
| Controlling the MCP23017 .....              | 30 |
| Raspberry Pi Resources .....                | 31 |

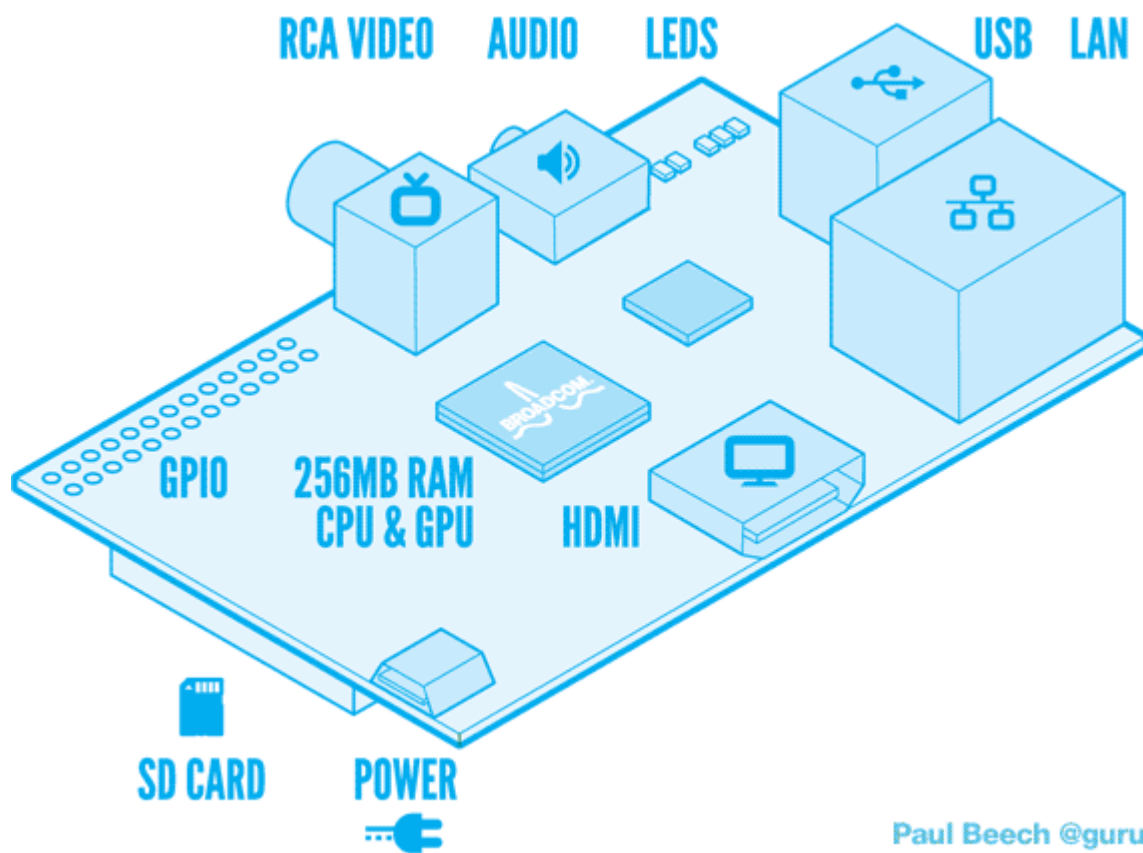
## Raspberry Pi

### What is a Raspberry Pi?

The Raspberry Pi is a credit-card sized computer that plugs into your TV and a keyboard. It's a capable little PC which can be used for many of the things that your desktop PC does, like spreadsheets, word-processing and games. It also plays high-definition video. We want to see it being used by kids all over the world to learn programming.

### What are the dimensions?

The Raspberry Pi measures 85.60mm x 53.98mm x 17mm, with a little overlap for the SD card and connectors which project over the edges. It weighs 45g.



## Raspberry Pi Specs – Model B

**Processor / Chipset:** Broadcom 700 MHz

**RAM:** Installed Size 256 MB

**Graphics Controller:** VideoCore IV

**Operating System / Software OS Provided:** Debian Linux

## Tweaking Raspberry Pi's Performance

Initially I was not planning on covering much hacking the Pi it's self, but seems that over clocking the Pi, and some OS modifications can greatly enhance the performance of the Pi. All of the changes to the pi here will be software based changes, but be forewarned that messing with CPU settings can result in the death of a Pi if not done properly. Everything in this guide has been tested by me, and confirmed to be working on my Pi.

Performing some of these tweaks or modifications can allow you to see a performance boost of up to 25%. Multiple tips have been cropping up online from cutting down on RAM usage, tuning the SD card to hacking some bits in the CPU.

### RAM Usage

By simply removing un-needed services and disabling daemons, you can greatly increase performance.

#### *Modifying Startup Services*

You will first need to install *sysv-rcconf* onto your Pi before you begin. Do so by issuing the following command: *sudo apt-get install sysv-rc-conf*

Once this has been installed, you can begin disabling un-needed services by issuing the following command: *sudo sysv-rc-conf*

*le: samba, nfs etc..*

Most services are safe to disable for normal operation of the Pi. If you know you will not be accessing any windows file shares, samba is safe to disable, same goes for NFS with Linux/Unix shares. If you do not know what it is, it's best to leave it alone. Once you are done you will be required to run the following command to complete the configuration: *dpkg-reconfigure innserv*

### *Inittab Modifications*

By default the Pi will spawn 6 terminals available for use once the Pi boots up. The average use does not need more than one or two at most. We can save some resources by limiting the amount of terminals spawned down from 6 to 2.

To do so, edit the `/etc/inittab` file by issuing the following command: `vi /etc/inittab`

*Once the file has been opened, look for lines matching the following (line 51):*

#### **BEFORE**

```
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6
```

#### **AFTER**

```
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
#3:23:respawn:/sbin/getty 38400 tty3
#4:23:respawn:/sbin/getty 38400 tty4
#5:23:respawn:/sbin/getty 38400 tty5
#6:23:respawn:/sbin/getty 38400 tty6
```

Once the above changes have been made, you can now save and exit the editor.

### *Disabling console access*

Depending how you use your Pi, you can save more resources by disabling console access if you are sure you will not need it. This is useful in cases where you are using your Pi as a Raspbmc media center or something.

To disable the console, you will need to edit the file: `/boot/cmdline.txt`

Remove the following line and save the file:

```
console=ttyAMA0,115200 kgdboc=ttyAMA0,115200
```

### *Enabling DASH*

Using dash as the system shell will improve the system's overall performance. Configure dash by issuing the following command: *dpkg-reconfigure dash*

When prompted to use dash as the default system shell, select: < Yes >

### *House Keeping*

After time the Pi will get full of old update archives etc or maybe even un-used software still left lingering around. To keep things tidy around the Pi, issue the following commands every once in awhile:

```
sudo apt-get autoremove  
sudo apt-get autoclean
```

### *Removing Gnome*

If you never plan on using gnome or maybe you are using your Pi as a Raspbmc media center, you can save some more resource by removing: *gnome* and *gvfs*. If you are sure you will never use the two, you can remove them and anything associated with the two by issuing the following commands:

```
apt-get remove gnome  
apt-get remove gvfs  
apt-get autoremove
```

### *Disk Tuning*

Since the Raspberry Pi uses the SDcard for everything, the read and write performance will drop. Though have no fear as there is a few things we can to minimize the hidden I/O, thus increasing performance of the SDcard. The good think about these improvements is that most of them are not based on modifying the kernel of any sort. ;D

### *Tweaking Syslog*

The first step we can take to improve the performance on the SDcard is to minimize the logging and remove unnecessary logs. Edit the syslog file by issuing the following command: *vi /etc/rsyslog.conf*

To disable a service from logging, you can put '#' in front of the line.

Once you have disabled the unnecessary log files, you can then restart syslog by issuing the command:  
*sudo /etc/init.d/rsyslog restart*

### *Creating partitions aligned with Flash Block*

Before creating this partition, you will need to find the erase block size of your SDcard. Most SDcards have a size of 128k, but you should double check your card before proceeding.

Finding out the size is simple using the below python script.

```
#!/usr/bin/env python
import sys

def unstuff(x, start, size):
    return (x >> start) & (2**size - 1)

def main(name, args):
    if len(args) != 1:
        print "Syntax: %s <card>" % (name, )
        print "Example: %s mmcblk0" % (name, )
        return 100

    card = args[0]
    dev = "/sys/class/block/%s/device/csd" % (card, )
    csd = int(file(dev).read(), 16)
    write_block_size = 2**unstuff(csd,22,4)
    erase_block_size = write_block_size*(unstuff(csd,39,7)+1)
    print "Erase block size of %s is %d bytes." % (card, erase_block_size)

sys.exit(main(sys.argv[0], sys.argv[1:]))
```

### *Formatting partitions with journaling turned off*

Journaling ensures the integrity of the filesystem by keeping a log of the ongoing disk changes. However, it is known to have a small overhead. Some people with special requirements and workloads can run without a journal and its integrity advantages. In Ext4 the journaling feature can be disabled, which provides a small performance improvement.

**WARNING:** Make sure all of the data on the SDcard has been backed up before attempting this.  
**DATA LOSS will occur!**

To disable journaling on the SDcard, issue the following command:

```
mkfs.ext4 -O ^has_journal -L PiBoot /dev/mmcblk0p1
fsck.ext4 -f /dev/mmcblk0p1
```



### *Tweaking Disk Scheduler*

To further tweak the disk performance, there is a few more things that can be disabled. The first step you can do is to tell disk scheduler to enabling the *deadline* I/O scheduler.

The Deadline scheduler excels at attempting to reduce the latency of any given single I/O for real-time like environments, which makes it perfect for the Pi.

To enable the *deadline* I/O scheduler, you will need to modify the */boot/cmdline.txt* file.

```
sudo vi /boot/cmdline.txt
```

Change the file to match the following, by adding *elevator=deadline*.

```
dwc_otg.lpm_enable=0 root=/dev/mmcblk0p3 rootfstype=ext4 elevator=deadline rootwait quiet
```

You can also increase disk performance by disabling *Access Time* for files and directories.

You can do so by editing the */boot/cmdline.txt* file and editing the *rootflags=* option to match the following:

```
rootflags=data=writeback,commit=120
```

This can also be enabled permanently with a kernel re-build, but for simplicity sake of the guide we are using the command line method for enabling these options.



## CPU – Over Clocking

Unless you truly understand what you are doing, safely skip this section...

### USE THIS TWEAK AT YOUR OWN RISK

The CPU on the Pi is quite simple to over clock, you can easily get a 15% performance increase without even over volting the CPU. Since you are not applying anymore voltage to the CPU, fans or heat sinks **should not** be required?

### USE THIS TWEAK AT YOUR OWN RISK

By default the Raspberry Pi comes with the *arm\_freq* set at 800. If you wish to improve performance just a bit and hang out on the safe side. Configure your */boot/config.txt* file to match the following:

#### WARNING:

While these settings have been tested on my Pi. Your mileage may vary, **use at your own risk**. Modification of these settings will greatly increase the risk of causing damage to your Pi.

|   |   |
|---|---|
| <b>/boot/config.txt – Safe Bet</b><br><i>arm_freq=900</i><br><i>gpu_freq=250</i><br><i>sdram_freq=500</i> | <b>/boot/config.txt – Not So Safe Bet</b><br><i>arm_freq=1000</i><br><i>Core_freq=500</i><br><i>sdram_freq=500</i><br><i>over_voltage=6</i><br><br><i>**If you are paranoid, use a fan with this config**</i> |
|---|---|

## Hacking stuff with the Pi

While there is already an extensive list of documentation and guides for getting up and running with your Pi, there have not been many for how to extend the use of your Pi or how to use your Pi for hacking other things or projects you may have in mind. In this document we will be mainly focusing on the GPIO pins of the Raspberry Pi.

The GPIO pins that can be found available on the PCB of the pi will allow you to interface with external applications via headers on the side of the board. These GPIO pins are very useful for controlling things like LEDs, Motors or reading from switches.

See below for a picture of the pi, the 26 GPIO pins have been highlighted on the bottom right corner.

**IMPORTANT:** Make sure to take note of **P1**, which has been circled in **red** below. It is important to know which way the pins are associated on the board as compared to the diagram provided.



## GPIO Introduction

### What is GPIO?

General Purpose Input/Output (a.k.a. GPIO) is a generic pin on a chip whose behavior (including whether it is an input or output pin) can be controlled (programmed) through software.

The Raspberry Pi allows peripherals and expansion boards (such as the upcoming [Rpi Gertboard](#)) to access the CPU by exposing the inputs and outputs.

The production Raspberry Pi board has a 26-pin 2.54 mm (100 mil) expansion header, marked as P1, arranged in a 2x13 strip. They provide 8 GPIO pins plus access to I<sup>2</sup>C, SPI, UART), as well as +3.3 V, +5 V and GND supply lines. Pin one is the pin in the first column and on the bottom row.

For a complete list of all available pins, see [http://elinux.org/RPi\\_BCM2835\\_GPIOs](http://elinux.org/RPi_BCM2835_GPIOs)

### Raspberry Pi GPIO

The Raspberry Pi has a General Purpose Input/Output (GPIO) connector and this carries a set of signals and buses. There are 8 general purpose digital I/O pins – these can be programmed as either digital outputs or inputs. One of these pins can be designated for PWM output too. Additionally there is a 2-wire I2C interface and a 4-wire SPI interface (with a 2nd select line, making it 5 pins in total) and the serial UART with a further 2 pins.

The I2C and SPI interfaces can also be used a general purpose I/O pins when not being used in their bus modes, and the UART pins can also be used if you reboot with the serial console disabled, giving a grand total of  $8 + 2 + 5 + 2 = 17$  I/O pins.



***The GPIO header contains 2 rows of pins, with 13 pins on each row as shown above.***

## Pin Diagram – Names & Alt 0 Functions

Out of the 26 pins that are provided by the GPIO header, 17 pins can be used as inputs or outputs to external applications. In a Pi's default state, all of the pins have been configured as inputs except GPIO pins 14 and 15. These pins are initialised as serial data lines TX & RX, these allow you to connect a terminal for logging in. In order to use these pins as Input or Output pins, they will need to first be re-configured.

### Pi Pin Layout

### Pin Names & Alt 0 Functions

|    |    |                           |                           |
|----|----|---------------------------|---------------------------|
| 1  | 2  | (1)P1 = +3.3v (50mA)      | (2) = +5v                 |
| 3  | 4  | (3) = GPIO0 (I2C0_SDA)    | (4) = (DNC)               |
| 5  | 6  | (5) = GPIO1 (I2C0_SCL)    | (6) = Ground (0v)         |
| 7  | 8  | (7) = GPIO4               | (8) = GPIO14 (UART0_TxD)  |
| 9  | 10 | (9) = (DNC)               | (10) = GPIO15 (UART0_RxD) |
| 11 | 12 | (11) = GPIO17             | (12) = GPIO18             |
| 13 | 14 | (13) = GPIO21 (PCM_DIN)   | (14) = (DNC)              |
| 15 | 16 | (15) = GPIO22             | (16) = GPIO23             |
| 17 | 18 | (17) = (DNC)              | (18) = GPIO24             |
| 19 | 20 | (19) = GPIO10 (SPI0_MOSI) | (20) = (DNC)              |
| 21 | 22 | (21) = GPIO9 (SPI0_MISO)  | (22) = GPIO25             |
| 23 | 24 | (23) = GPIO11 (SPI0_SCLK) | (24) = GPIO8 (SPI0_CEO)   |
| 25 | 26 | (25) = (DNC)              | (26) = GPIO7 (SPI0_CE1)   |

[ Legend ]

+5 Volt

3.3 Volt

Ground, 0V

DNC – Do not connect

UART

GPIO

SPI

### Hardware Notes

**PIN 2** – Supply through input poly fuse

**GPIO 0** – 1k8 pull up resistor

**GPIO 1** - 1k8 pull up resistor

**GPIO 14** – Boot to Alt 0 ->

**GPIO 15** – Boot to Alt 0 ->

**GPIO 4** - GPCLK0

When starting out, ALWAYS make sure to locate **P1** first. This will make locating the pins in proper order much easier. **Pin 1** will provide 3.3v (50ma) MAX.

Starting at **P1** or **Pin 1**, you should be able to figure out the other pins.

### Other Alternative Functions

**GPIO 14** – ALT5 = UART1\_TXD

**GPIO 15** – ALT5 = UART1\_RXD

**GPIO 18** – ALT4 SPI1\_CE0\_N ALT5 = PWM0

**GPIO 23** – ALT3 = SD1\_CMD ALT4 = ARM\_RTCK

**GPIO 24** – ALT3 = SD1\_DATA0 ALT4 = ARM\_TDO

**GPIO 25** – ALT4 = ARM\_TCK

**GPIO 0** – I2C0\_SDA

**GPIO 1** – I2C0\_SCL

**GPIO 17** – ALT3 = UART0\_RTS, ALT5 = UART1\_RTS

**GPIO 21** – ALT5 = GPCLK1

**GPIO 22** – ALT3 = SD1\_CLK ALT4 = ARM\_TRST

### Notes:

- **Pin 3** (SDA0) and **Pin 5** (SCL0) are preset to be used as an I<sup>2</sup>C interface. So there are 1.8 kilohm pull up resistors on the board for these pins.
- **Pin 12** supports PWM.
- It is possible to reconfigure GPIO connector pins **P1-7, 15, 16, 18, 22** (chipset GPIOs 4 and 22 to 25) to provide an ARM JTAG interface. However ARM\_TMS isn't available on the GPIO connector (chipset **pin 12** or **27** is needed). Chipset **pin 27** is available on **S5**, the CSI camera interface however.

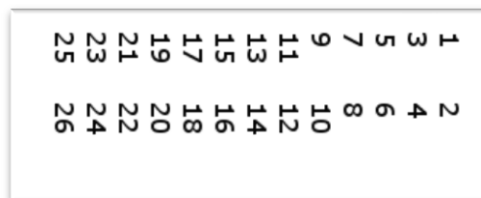
### WARNING

**Make sure that you are looking at the pins the correct way. Failure to do so could result in a dead Pi!**

**The Raspberry Pi is a 3.3 volt device. If you attempt to connect to any 5V logic application, Failure to adhere to this can result in a dead pi!**

### Example Pi Pin Diagram

**Hint:** Even number pins are on the inner side of the pi, while the odd number pins reside on the outer side of the pi.



## Power Pins

The GPIO header provides a 5V source on **Pin 2** and 3.3V on **Pin 1**. The 3.3V supply on **Pin 1** is limited to a maximum draw of 50mA. The 5V supply on **Pin 2** will draw current directly from the microUSB supply, whatever is left over from the board can be used via this pin. Using a 1A power supply, 300mA can be used once the board has drawn it's required 700mA.

**Model A:** 1000 mA - 500 mA -> max current draw: 500 mA

**Model B:** 1000 mA - 700 mA -> max current draw: 300 mA

**Warning:** Be very careful with the 5V pin.

If you short it to any other P1 pin you may permanently damage your Pi.

**Pro Tip:** Strip a short piece of insulation from another wire and push it over the 5V pin so you don't accidentally touch it with a probe.

The maximum you can draw from the power pin is between: 150-250mA and again this all depends on what you have currently running, this could be much less. See the link below for more information:  
<http://nathan.chantrell.net/20120610/raspberry-pi-and-i2c-devices-of-different-voltage#f3fuse>

## Protecting your pins and your Pi

Before you go connecting stuff up and playing around, **make sure you know what you are doing!**

Almost all of the GPIO pins located on the header go directly into the Broadcom chip.

A simple short circuit or mistake in wiring can result in the quick death of your Pi.

## GPIO – Interaction

Having your way with the Pi's pins...

### WiringPi

WiringPi is a Wiring library written in C and should be usable from C++ and many other languages with suitable wrappers.

If you have ever used an Arduino before, you will know they are composed of two things. One is the hardware platform, and the other is the software platform. Part of the software side of things is a tool called **Wiring**. Wiring is the core of the input and output for the Arduino system.

### Pin numbering

WiringPi supports both an Arduino style pin numbering scheme which numbers the pins sequentially from 0 through 16, as well as the Raspberry Pi's native BCM GPIO pin numbering scheme.

### Downloading WiringPi

<https://projects.drogon.net/raspberry-pi/wiringpi/download-and-install/>

### Special Pin Functions

WiringPi defines 17 pins, but some of them and the functions we can use may potentially cause problems with other parts of the Raspberry Pi Linux system.

- **Pins 0 through 7** (GPIO 17, 18, 21, 22, 23, 24, 25, 4 respectively): These are safe to use at any time and can be set to input or output with or without the internal pull-up or pull-down resistors enabled.
- **PWM**: You can change the function of pin 1 (GPIO 18) to be PWM output, however if you are currently playing music or using the audio system via the 3.5mm jack socket, then you'll find one channel of audio PWM coming through the pin! If you are not using the audio at all, (or the audio is going via the HDMI cable), then this pin is free to be used in PWM mode.
- **Pins 8 and 9** (GPIO 0 and 1): These are the I2C pins. You may use them for digital IO if you are not using any I2C drivers which use these pins, however note that they have on-board 1k8 resistors pulling the signals to the 3v3 supply. This feature does make them handy for switch inputs where the switch simply shorts the pin to ground without having to enable the internal pull-up resistors
- **Pins 10,11,12,13 and 14** (GPIO 8, 7, 10, 9 and 11 respectively): These are used for the SPI interface. Like the I2C interface, if you are not using it, then you can freely use them for your own purposes. Unlike I2C, these pins do not have any external pull up (or pull down) resistors.
- **Pins 15 and 16** (GPIO 14 and 15): These are used by the UART for Tx and Rx respectively. If you want to use these pins as general purpose I/O pins then you need to make sure that you reboot your Pi with the serial console disabled. See the file **/boot/cmdline.txt** and edit it appropriately.



## Programming Libraries

Controlling the GPIO pin's using libraries from various programming languages.

### *Python Library*

RPi.GPIO Python library - <http://pypi.python.org/pypi/RPi.GPIO>

*See Code Example A for example.*

### *Java Library*

RPi-GPIO-Java - <http://code.google.com/p/rpi-gpio-java/>

*See Code Example B for example.*

**C** – Using the bcm2835 Library

<http://www.open.com.au/mikem/bcm2835>

*See Code Example C for example.*

**Perl** – Using the bcm2835 library and Device::BCM2835 module from CPAN.

<http://www.open.com.au/mikem/bcm2835>

<http://search.cpan.org/~mikem/Device-BCM2835-1.0/lib/Device/BCM2835.pm>

*See Code Example D for example.*

### **C#**

RaspberryPiDotNet library – <https://github.com/cypherkey/RaspberryPi.Net/>

*See Code Example E for example.*

### *Ruby*

WiringPi Ruby Gem – <http://pi.gadgetoid.co.uk/post/015-wiringpi-now-with-serial>

*See Code Example F for example.*

### *Shell Script*

*See Code Example G for example.*

## Code Examples

### *Code Example A – Python*

```
import RPi.GPIO as GPIO

# Set up the GPIO channels - one input and one output
GPIO.setup(11, GPIO.IN)
GPIO.setup(12, GPIO.OUT)

# Input from pin 11
input_value = GPIO.input(11)

# Output to pin 12
GPIO.output(12, True)

# The same script as above but using BCM GPIO 00..nn numbers
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.IN)
GPIO.setup(18, GPIO.OUT)
input_value = GPIO.input(17)
GPIO.output(18, True)
```

### *Code Example B – Java*

```
public static void main(String[] args) {
    GpioGateway gpio = new GpioGatewayImpl();

    //set up the GPIO channels - one input and one output
    gpio.setup(Boardpin.PIN11_GPIO17, Direction.IN);
    gpio.setup(Boardpin.PIN12_GPIO18, Direction.OUT);

    // input from pin 11
    boolean input_value = gpio.getValue(Boardpin.PIN11_GPIO17);

    // output to pin 12
    gpio.setValue(Boardpin.PIN12_GPIO18, true);
}
```

### Code Example C – C

```
// blink.c
//
// Example program for bcm2835 library
// Blinks a pin on an off every 0.5 secs
//
// After installing bcm2835, you can build this
// with something like:
// gcc -o blink blink.c -l bcm2835
// sudo ./blink
//
// Or you can test it before installing with:
// gcc -o blink -I ../../src ../../src/bcm2835.c blink.c
// sudo ./blink
//
// Author: Mike McCauley (mikem@open.com.au)
// Copyright (C) 2011 Mike McCauley
// $Id: RF22.h,v 1.21 2012/05/30 01:51:25 mikem Exp $

#include <bcm2835.h>

// Blinks on RPi pin GPIO 11
#define PIN RPI_GPIO_P1_11

int main(int argc, char **argv)
{
    // If you call this, it will not actually access the GPIO
    // Use for testing
    // bcm2835_set_debug(1);

    if (!bcm2835_init())
        return 1;

    // Set the pin to be an output
    bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_OUTP);

    // Blink
    while (1)
    {
        // Turn it on
        bcm2835_gpio_write(PIN, HIGH);

        // wait a bit
        delay(500);

        // turn it off
```

```

        bcm2835_gpio_write(PIN, LOW);

        // wait a bit
        delay(500);
    }

    return 0;
}

```

### *Code Example D – Perl*

```

use Device::BCM2835;
use strict;

# call set_debug(1) to do a non-destructive test on non-RPi hardware
#Device::BCM2835::set_debug(1);
Device::BCM2835::init()
    || die "Could not init library";

# Blink pin 11:
# Set RPi pin 11 to be an output
Device::BCM2835::gpio_fsel(&Device::BCM2835::RPI_GPIO_P1_11,
                           &Device::BCM2835::BCM2835_GPIO_FSEL_OUTP);

while (1)
{
    # Turn it on
    Device::BCM2835::gpio_write(&Device::BCM2835::RPI_GPIO_P1_11, 1);
    Device::BCM2835::delay(500); # Milliseconds
    # Turn it off
    Device::BCM2835::gpio_write(&Device::BCM2835::RPI_GPIO_P1_11, 0);
    Device::BCM2835::delay(500); # Milliseconds
}

```

### *Code Example E – C#*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using RaspberryPiDotNet;
using System.Threading;

namespace RaspPi
{
    class Program
    {
        static void Main(string[] args)
        {
            // Access the GPIO pin using a static method
            GPIOFile.Write(GPIO.GPIOPins.GPIO00, true);

            // Create a new GPIO object
            GPIOMem gpio = new GPIOMem(GPIO.GPIOPins.GPIO01);
            gpio.Write(false);
        }
    }
}
```

### *Code Example F – Ruby*

```
MY_PIN = 1

require 'wiringpi'
io = WiringPi::GPIO.new
io.mode(MY_PIN, OUTPUT)
io.write(MY_PIN, HIGH)
io.read(MY_PIN)
```

### *Code Example G – Shell Script*

```
#!/bin/sh

# GPIO numbers should be from this list
# 0, 1, 4, 7, 8, 9, 10, 11, 14, 15, 17, 18, 21, 22, 23, 24, 25

# Note that the GPIO numbers that you program here refer to the pins
# of the BCM2835 and *not* the numbers on the pin header.
# So, if you want to activate GPIO7 on the header you should be
# using GPIO4 in this script. Likewise if you want to activate GPIO0
# on the header you should be using GPIO17 here.

# Set up GPIO 4 and set to output
echo "4" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio4/direction

# Set up GPIO 7 and set to input
echo "7" > /sys/class/gpio/export
echo "in" > /sys/class/gpio/gpio7/direction

# Write output
echo "1" > /sys/class/gpio/gpio4/value

# Read from input
cat /sys/class/gpio/gpio7/value

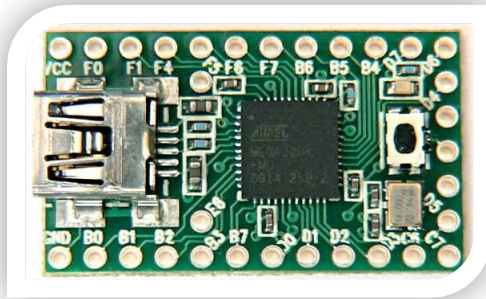
# Clean up
echo "4" > /sys/class/gpio/unexport
echo "7" > /sys/class/gpio/unexport
```

## GPIO – External Applications

### Interfacing With a Teensy Kit

Teensy Pinout: <http://www.pjrc.com/teensy/pinout.html>

Logic Level Converter: <https://www.sparkfun.com/products/8745?>



### UART/Serial

Using a logic level converter you can work with the UART / Serial interface to allow a Pi to communicate with a Teensie board. The TX from the teensy should go to the RX on the Raspberry Pi, vice versa.

To connect up the Pi, connect up the following GPIOs to the corresponding pins on the logic level converter.

#### Raspberry Pi to Logic level converter

**GPIO 14 (TXD)** connects too **TXI**

**GPIO 15 (RXD)** connects too **RX0**

**3v3 Power P1** connects too **LV**

**PIN 6 - Ground** connects to **Ground**

#### Logic level converter to Teensy

**HV** connects too **VCC**

**GND** connects too **GND**

**TX0** connects too **D2**

**RXI** connects too **D3**

Ensure both **GND** on the Logic Level Converter have been connected to **GND**.

You should be able to purchase a logic level converter for cheap, usually under 3\$.



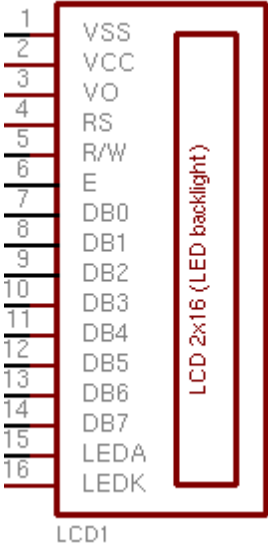
## Interfacing with LCD Displays

Hooking the Pi up to a 2x16 HD44780 compatible LCD via GPIO



Another cool thing to control with your Pi is a LCD screen. In this example I will be using a HD55780 compatible LCD display. These can be found pretty cheap on ebay for a few dollars. Double check the data sheet for your LCD as pins may vary from vendor to vendor.

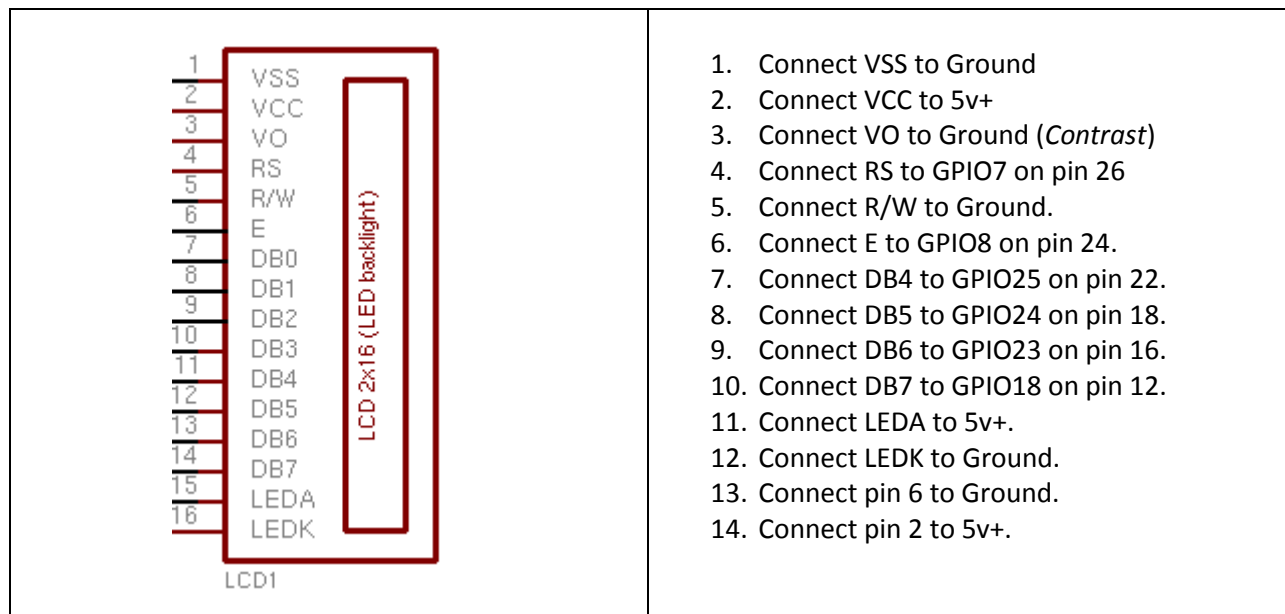
### LCD Pinout Overview

|   |   |
|---|---|
|  | <ol style="list-style-type: none"><li>1. Ground</li><li>2. VCC (Usually +5V)</li><li>3. Contrast adjustment (VO)</li><li>4. Register Select (RS).<br/>RS=0: Command, RS=1: Data</li><li>5. Read/Write (R/W).<br/>R/W=0: Write, R/W=1: Read</li><li>6. Enable</li><li>7. Bit 0 (Not required in 4-bit operation)</li><li>8. Bit 1 (Not required in 4-bit operation)</li><li>9. Bit 2 (Not required in 4-bit operation)</li><li>10. Bit 3 (Not required in 4-bit operation)</li><li>11. Bit 4</li><li>12. Bit 5</li><li>13. Bit 6</li><li>14. Bit 7</li><li>15. LED Backlight Anode (+)</li><li>16. LED Backlight Cathode (-)</li></ol> |
|---|---|

## Wiring things up to the LCD

Normally a HD44780 LCD would require 8 data lines to provide data to bits 0-7. However you can set this device to operate in “4 bit” mode which will then allow you to send data in two chunks or 4 bits. This is handy as it reduces the amount of required GPIO connections from the Pi, leaving them free for other things.

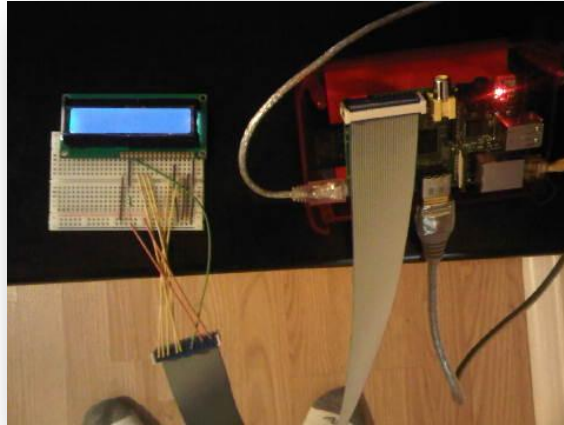
The HD44780 LCD will also allow you to control the brightness of the LCD by adjusting the voltage flowing to VO. The voltage must be between the range of 0 and 5volts. In the above example, VO has been connected into ground. Using a potentiometer, you could add an adjustable knob to control the brightness of the LCD screen in real time.



### NOTE(s):

- pin numbers are referring to pins on the Raspberry pi, where as names refer to the image on the left.
- LEDA provides 5 volts to the backlight LED of the LCD. HD44780 compatible devices should operate between 2.2 and 5.5 volts. LEDA can be directly connected to the 5v source.
- The RW pin allows you to set the LCD in read or write mode, for this example we want to send data to the LCD, but not allow the LCD to send data back to the Pi. The reason for this is that the Pi will not take more than 5V of input on the GPIO header. Doing so may result in damage to your Pi. Tying the RW pin into ground will ensure that the LCD will **NOT** attempt to pull the lines over 5volts.

Once you have everything connected up properly, power on and boot up your Pi. If everything was done correctly thus far, the LCD screen should now power on and show either one or two rows of boxes. These boxes will remain until the LCD has been initialized for the first time.



### *Using Python to control the LCD*

Now that everything looks to be up and running, you can now control what is displayed onto the screen. Using any of the programming language libraries discussed earlier, as an example we will be using some simple python code with the RPi.GPIO library. Since we will be accessing the GPIO interface, you will need to run python as root when running the code.

I am not the author of this code, I just hacked it up a bit to better fit the document. The original code was written by: Matt Hawkins.

```
#!/usr/bin/python

import RPi.GPIO as GPIO
import time

# Define GPIO to LCD mapping
LCD_RS = 7
LCD_E  = 8
LCD_D4 = 25
LCD_D5 = 24
LCD_D6 = 23
LCD_D7 = 18
# Define some device constants
LCD_WIDTH = 16 # Maximum characters per line
LCD_CHR = True
LCD_CMD = False
LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line
```

```

# Timing constants
E_PULSE = 0.00005
E_DELAY = 0.00005

def main():
    # Main program block
    GPIO.setmode(GPIO.BCM)    # Use BCM GPIO numbers
    GPIO.setup(LCD_E, GPIO.OUT) # E
    GPIO.setup(LCD_RS, GPIO.OUT) # RS
    GPIO.setup(LCD_D4, GPIO.OUT) # DB4
    GPIO.setup(LCD_D5, GPIO.OUT) # DB5
    GPIO.setup(LCD_D6, GPIO.OUT) # DB6
    GPIO.setup(LCD_D7, GPIO.OUT) # DB7

    # Initialise display
    lcd_init()

    # Send some test
    lcd_byte(LCD_LINE_1, LCD_CMD)
    lcd_string("Raspberry Pi")
    lcd_byte(LCD_LINE_2, LCD_CMD)
    lcd_string("Model B")

    time.sleep(3) # 3 second delay

    # Send some text
    lcd_byte(LCD_LINE_1, LCD_CMD)
    lcd_string("magikh0e")
    lcd_byte(LCD_LINE_2, LCD_CMD)
    lcd_string("DARPA net")
    time.sleep(20)

def lcd_init():
    # Initialize display
    lcd_byte(0x33, LCD_CMD)
    lcd_byte(0x32, LCD_CMD)
    lcd_byte(0x28, LCD_CMD)
    lcd_byte(0x0C, LCD_CMD)
    lcd_byte(0x06, LCD_CMD)
    lcd_byte(0x01, LCD_CMD)

def lcd_string(message):
    # Send string to display
    message = message.ljust(LCD_WIDTH, " ")

    for i in range(LCD_WIDTH):
        lcd_byte(ord(message[i]), LCD_CHR)
def lcd_byte(bits, mode):

```

```
GPIO.output(LCD_RS, mode) # RS
```

```
# High bits
```

```
GPIO.output(LCD_D4, False)
```

```
GPIO.output(LCD_D5, False)
```

```
GPIO.output(LCD_D6, False)
```

```
GPIO.output(LCD_D7, False)
```

```
if bits&0x10==0x10:
```

```
    GPIO.output(LCD_D4, True)
```

```
if bits&0x20==0x20:
```

```
    GPIO.output(LCD_D5, True)
```

```
if bits&0x40==0x40:
```

```
    GPIO.output(LCD_D6, True)
```

```
if bits&0x80==0x80:
```

```
    GPIO.output(LCD_D7, True)
```

```
# Toggle 'Enable' pin
```

```
time.sleep(E_DELAY)
```

```
GPIO.output(LCD_E, True)
```

```
time.sleep(E_PULSE)
```

```
GPIO.output(LCD_E, False)
```

```
time.sleep(E_DELAY)
```

```
# Low bits
```

```
GPIO.output(LCD_D4, False)
```

```
GPIO.output(LCD_D5, False)
```

```
GPIO.output(LCD_D6, False)
```

```
GPIO.output(LCD_D7, False)
```

```
if bits&0x01==0x01:
```

```
    GPIO.output(LCD_D4, True)
```

```
if bits&0x02==0x02:
```

```
    GPIO.output(LCD_D5, True)
```

```
if bits&0x04==0x04:
```

```
    GPIO.output(LCD_D6, True)
```

```
if bits&0x08==0x08:
```

```
    GPIO.output(LCD_D7, True)
```

```
# Toggle 'Enable' pin
```

```
time.sleep(E_DELAY)
```

```
GPIO.output(LCD_E, True)
```

```
time.sleep(E_PULSE)
```

```
GPIO.output(LCD_E, False)
```

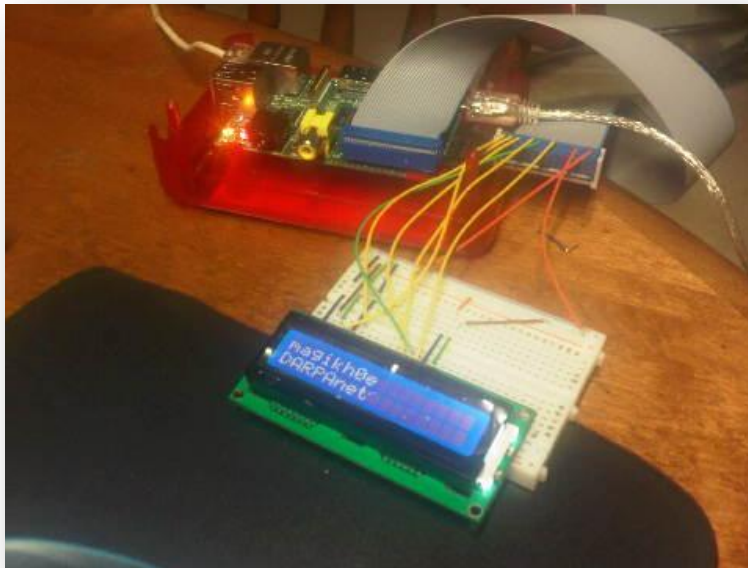
```
time.sleep(E_DELAY)
```

```
if __name__ == '__main__':
```

```
    main()
```

If you get an error like “`RPi.GPIO.SetupException: No access to /dev/mem.`”  
Make sure you are running python as root: `sudo python testlcd.py`.

If everything went well, you should first see “*Raspberry Pi Model B*” appear, shortly after “*magikh0e, DARPAneT*” should appear.



Common issues I have ran into...

**Only see squares across the LCD:** Double check all of your connections are going to the right place, and ensure good connectivity with the LCD.

**Weir characters appearing:** Check the connectivity on the LCD.

## MCP23017 I2C I/O Expander

Not enough GPIO pins for you, well not a problem if you have a *16bit MCP23017 I2C I/O Expander* kicking around. This will also work with the *8bit model, MCP23008*. They both also come in a DIP form, so using them build your own expansion board for the Pi should be fairly simple. If not they are simple enough to use on any breadboard as well. The data sheet for the *16bit* version of the *MCP23017 I2C I/O Expander* can be found here: <http://ww1.microchip.com/downloads/en/DeviceDoc/21952b.pdf>

The *16bit* version of the *MCP23017* chip has 28 pins that will give you a total of 16 pins that can be used. These pins can be used as either inputs or outputs. Up to 8 of these pins can be used on 1 I2C bus, thus giving you a lot more I/O than the Pi has built in. The best thing about this chip is that you can reduce the risk of damaging your Pi each pin has a maximum of *25mA* for input or output. The expander can also be placed away from the Pi its self, and connecting up using only 4 wires. If space is a concern, go with the *8bit MCP23008* model.

### Required drivers and software

Before you will be able to control the expander, you will require some drivers and tools first. Keep in mind that the work being done on the I2C drivers are still in pretty early stages. Your Pi will need to be running a kernel with the bitbanging driver, or have the driver available for the kernel you are currently running.

After verifying you have a kernel with the bitbanging driver enabled, you will need to install the *i2c-tools* package by issuing the following command:

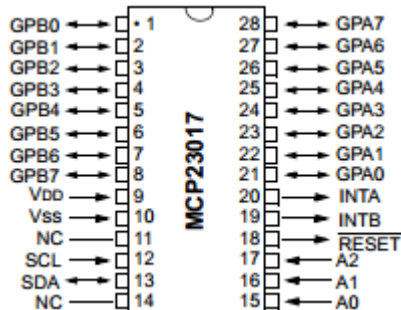
```
sudo apt-get install i2c-tools
```

The *i2c-tools* package will give us the ability to scan the I2C bus and sending values to I2C addresses and registers using command line tools.

### Connecting the expander to the Pi

Now that you have verified all the proper software is in place, you can now wire the expander into the Pi. Using the chart below connect up the pins from the *MCP23017* to the pins on your Pi accordingly.

**MCP23017**



| MCP23017      | Pi GPIO                 |
|---------------|-------------------------|
| PIN 9 - VDD   | PIN 2 – <b>Vcc 5v+</b>  |
| PIN 10 – Vss  | <b>Ground</b>           |
| PIN 12 – SCL  | PIN 5 – <b>I2C0_SCL</b> |
| PIN 13 – SDA  | PIN 3 – <b>I2C0_SDA</b> |
| PINS 15,16,17 | <b>Ground</b>           |
| PIN 18        | PIN 2 – <b>Vcc 5v+</b>  |



## Notes:

**PIN 9:** This can be connected to the Pi's 5v source, or any external source up to 5.5volts.

**PINS 15(A0), 16(A1), 17(A2):** Setting these pins to ground selects the I2C address as 0x20, other combinations can set a different address. *See data sheet.*

**PIN 18:** Setting this pin to Vcc turns the expander on.

### *Testing the Pi and Expander communication*

Once everything has been connected and verified. You can now test your Pi's communication with the expander you have just connected.

*i2cdetect -y 0*

If everything is happy, you should see an ASCII representation of a table with 20 in the first column on the row marked 20. This will show that there is something there with an I2C address of 0x20. As we expect.

### **Controlling the MCP23017**

As you read in the data sheet for the MCP23017, the I/O pins are laid out in 2 banks. A and B and each bank is controlled together. In order to set a pin as an input or output, you will need to send a hex value to the correct register. You can find this in *Table 1.4* of the datasheet linked above. *IODIRA (0x00)* will sets the input/output state for bank A and *IODIRB (0x01)* for bank B. In order to change a pin to be an input, you need to set each of the 8bits to 1. To setup the pin as an output, each bit will need to be set to 0. Keep in mind in a default state all of the pins are setup to be inputs.

So if you wish to set pins 0,1, and 7 to be inputs and the rest of the pins as outputs. You would set *10000011* in binary or *0x83* in hex. To set the entire bank as outputs, you can simply use *0x00*.

Once the pins have been configured as inputs/outputs, you can turn them on or off by sending a hex value to the register for the particular bank you wish to control. *0x12* for bank A, *0x13* for bank B.

As always 1 is on, 0 is off, using the same form as above. So if you wish to turn pin 0 on, you will send *00000001* as binary, or *0x01* as hex.

### *I2cset examples*

**Set all of bank A to be outputs:** *i2cset -y 0 0x20 0x00 0x00*

**Set GPA0 as on:** *i2cset -y 0 0x20 0x12 0x01*

**Set GPA0 as off:** *i2cset -y 0 0x20 0x12 0x00*

**i2cset command format:** *i2cset i2c-bus i2c-address i2c-register value*

## Raspberry Pi Resources

**Raspberry Pi for beginners – Unofficial YouTube Channel:**

<http://www.youtube.com/user/RaspberryPiBeginners>

**Hardware lesson with Gert: make your own ribbon cable connector:**

<http://www.raspberrypi.org/archives/1404>

**Raspberry Pi - How to use the GPIO #23:**

[http://www.youtube.com/watch?v=q\\_NvDTZlaS4](http://www.youtube.com/watch?v=q_NvDTZlaS4)

**Raspberry Pi Quick Start Guide:**

<http://www.raspberrypi.org/quick-start-guide>

**Raspberry Pi Wiki:**

<http://elinux.org/RaspberryPiBoard>

**SSH Phone Home: Using the Raspberry Pi as a proxy/pivot (Shovel a Shell):**

[http://www.irongeek.com/i.php?page=security/raspberry-pi-recipes#SSH Phone Home: Using the Raspberry Pi as a proxy/pivot \(Shovel a Shell\)](http://www.irongeek.com/i.php?page=security/raspberry-pi-recipes#SSH Phone Home: Using the Raspberry Pi as a proxy/pivot (Shovel a Shell))

**Raspberry-PWN:**

<https://github.com/pwnieexpress/Raspberry-Pwn>

**Raspberry Pi Kernel:**

<http://www.bootc.net/projects/raspberry-pi-kernel/>

**Display Interface Specifications:**

<http://www.mipi.org/specifications/display-interface>

**Camera Interface Specifications:**

<http://www.mipi.org/specifications/camera-interface>

