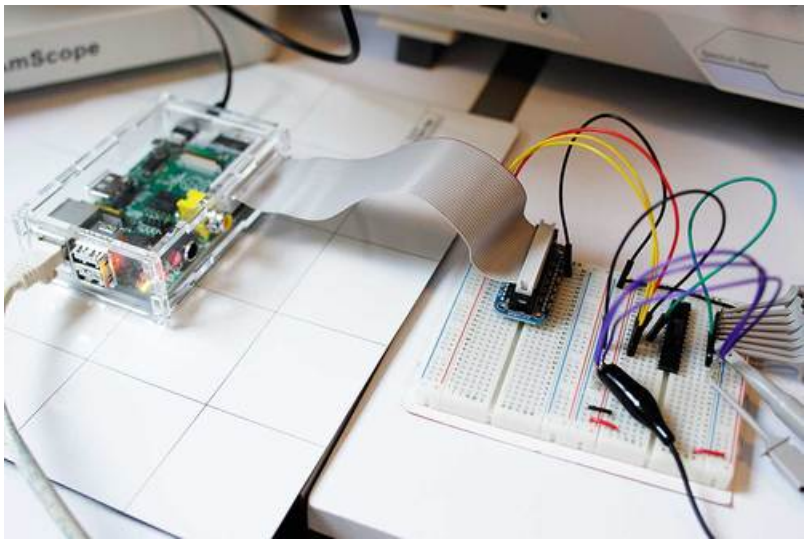
adafruit learning system

MCP230xx GPIO Expander on the Raspberry Pi

Created by Kevin Townsend



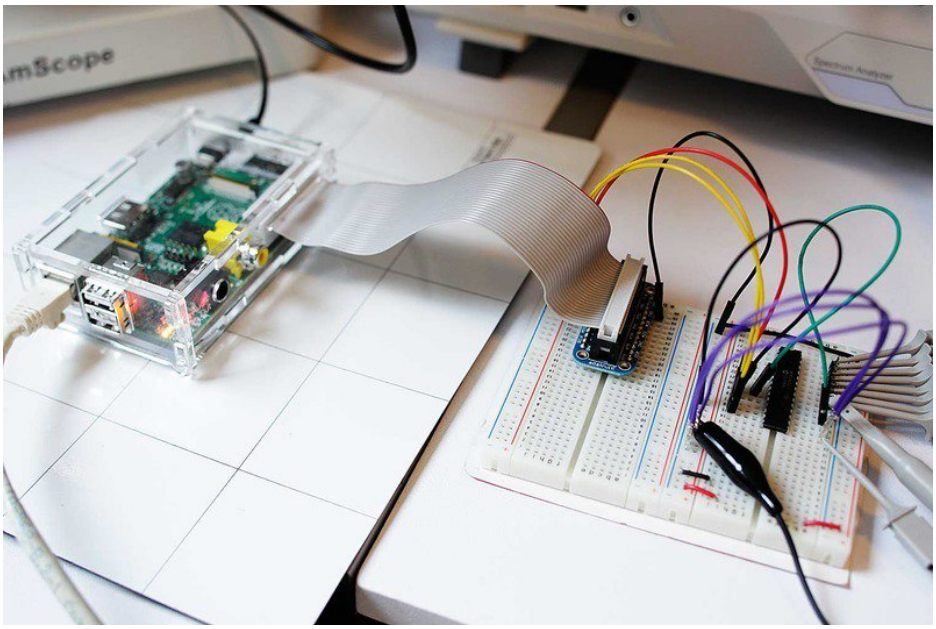
Last updated on 2018-10-16 09:35:37 PM UTC

Guide Contents

Guide Contents	2
Overview	3
What You'll Need	3
Hooking it all up	4
Using the library	6
Instantiating an instance of Adafruit_MCP230xx	7
Pin Numbering	7
Setting a pin as Input	7
Setting a pin as Output	8
Interrupts & Callbacks	8

Overview

The examples in this guide are no longer supported. Check out the MCP23xx guide for CircuitPython and Python usage: https://learn.adafruit.com/using-mcp23008-mcp23017-with-circuitpython/overview



While the Raspberry Pi packs and awful lot of punch for the price, and it's fairly flexible where HW expandability is concerned, there are situations where you might want a bit more basic digital IO. Thankfully, it's an easy problem to solve with an I2C-enabled device like the MCP23008 (for an extra 8 GPIO pins) or the MCP23017 (for an extra 16 GPIO pins). This tutorial will show you how you can get up and running quickly with either of these chips.

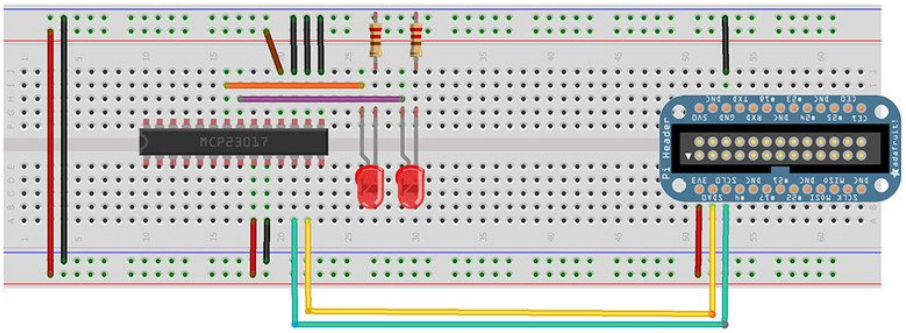
What You'll Need

- A Raspberry Pi (http://adafru.it/998) Model B
- A Pi Cobbler Breakout (http://adafru.it/914)
- An MCP23017 (http://adafru.it/732) or MCP23008 (http://adafru.it/593)
- And LED and a resistor to test with if you don't have a DMM or an oscilloscope
- If you're **not** using Occidentalis (https://adafru.it/aQx), Adafruit's own Raspberry Pi distro, you'll also need to make sure your Pi is configured for I2C (https://adafru.it/aTI) before running through this tutorial. (If you're using Occidentalis, I2C is already enabled, though, and you're ready to go!)

If you're not using Occidentalis, Adafruit's own Raspberry Pi distro, you'll also need to make sure your Pi is configured for I2C before running through our tutorial at http://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c (If you're using Occidentalis, I2C is already enabled, though, and you're ready to go!)

Hooking it all up

The examples in this guide are no longer supported. Check out the MCP23xx guide for CircuitPython and Python usage: https://learn.adafruit.com/using-mcp23008-mcp23017-with-circuitpython/overview



The way that you hook the chip up to your breadboard will depend on the package you use (8-pin MCP23008 or 16-pin MCP23017). The pinouts are quite different between the two chips, so check the datasheet carefully first.

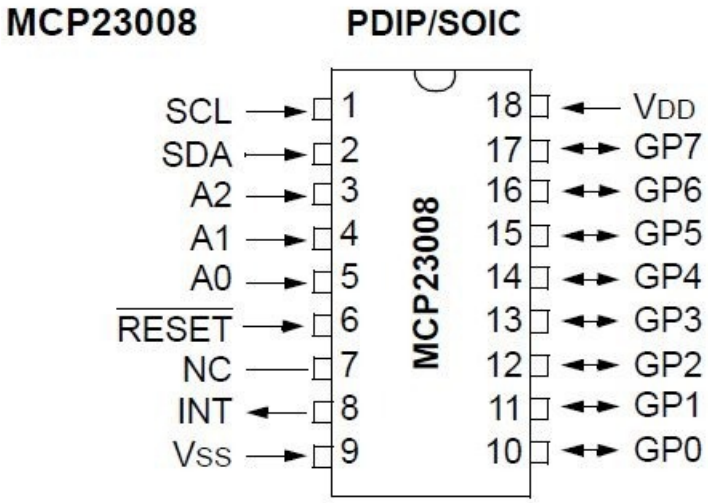
The MCP23017 is shown above with two LEDs connected, on GPA0 and GPA1.

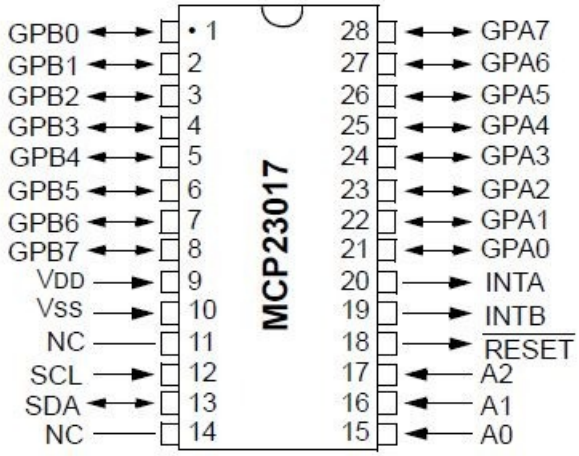
1. The yellow line is SDA
2. The green line is SCL
3. The three black lines on top are the address pins
4. The brown pin is RESET which must be pulled high for normal operation
5. Red is 3.3V
6. Black is GND.

Since these io expander chips use i2c to communiate, you *can* theoretically power them from 5V while still connecting the i2c data lines to a 3.3V device like the pi. That's because the Pi has two i2c resistors that pull up SDA/SCL to 3.3V. Just make sure not to connect any resistors to SDA/SCL to 5V and you can power the chip from 5V (and have 5V input/output on the MCP chip). Its also fine of course to power the MCP chip from 3.3V but the 5V line on the Pi has more current capability so you might find its better to go that way.

BUT if your Pi power supply drifts a little higher than 5V, it might stop being able to register the 3.3V signal. So we recommend starting with 3.3V, and if you need 5V GPIO signalling on the MCP expander, try swapping the red wire to 5.0V

You can compare the two pinouts below to figure out how the 8-pin package should be hooked up depending on the pin names:





You're free to hook anything you want up to the 8 or 16 GPIO pins, but LEDs are used here since most people have one or two laying around and it's an easy way to verify the pin outputs. Be sure to connect a resistor in series to GND, though, to prevent the LED from burning out (if you don't know what value or the details of your LED try something large like 1K to start with).

Here's a quick video of the setup I was using during testing and development. An MCP23017 is used here, running out to a mixed-signal oscilloscope with an 8-channel logic analyzer (ergo the white clip-ons on all the GPIO pins).

Using the library

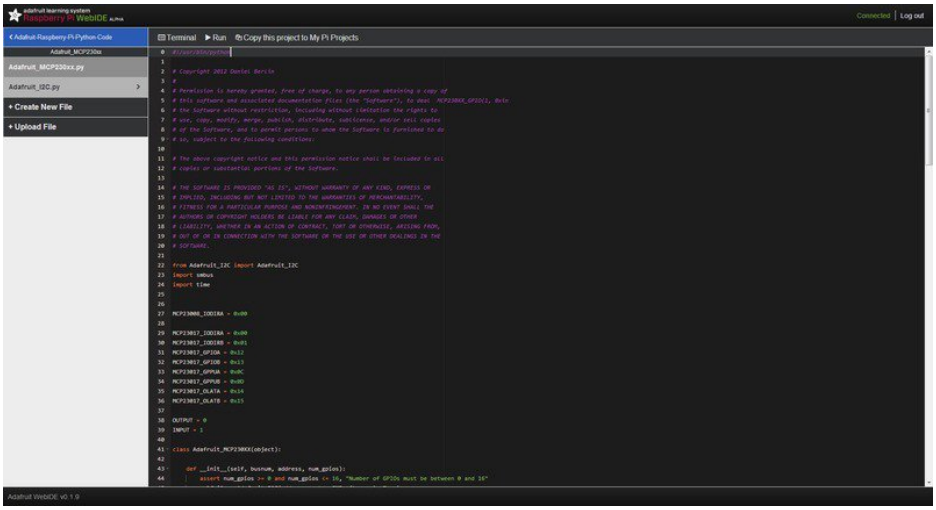
The examples in this guide are no longer supported. Check out the MCP23xx guide for CircuitPython and Python usage: https://learn.adafruit.com/using-mcp23008-mcp23017-with-circuitpython/overview

Never one to leave you with just a breakout board or an IC and a goodbye, Adafruit provides a library for the MCP23008 and MCP23017 in our Pi repository on github (https://adafru.it/r6A). The easiest way to use it is with our convenient WebIDE (https://adafru.it/aRn), which will automatically point to the Adafruit github repository.

Once you've opened up the WebIDE in the browser, you simply need to click in the left-hand navigation on the following folders and filenames:

- Adafruit-Raspberry-Pi-Python-Code
- Adafruit_MCP230xx
- Adafruit_MCP230xx.py

This should give you something similar to the following:



```
# Use busnum = 0 for older Raspberry Pi's (256MB)
mcp = Adafruit_MCP230XX(busnum = 0, address = 0x20, num_gpios = 16)
# Use busnum = 1 for new Raspberry Pi's (512MB with mounting holes)
# mcp = Adafruit_MCP230XX(busnum = 1, address = 0x20, num_gpios = 16)

# Set pins 0, 1 and 2 to output (you can set pins 0..15 this way)
mcp.config(0, OUTPUT)
mcp.config(1, OUTPUT)
mcp.config(2, OUTPUT)

# Set pin 3 to input with the pullup resistor enabled
mcp.pullup(3, 1)
# Read pin 3 and display the results
print "%d: %x" % (3, mcp.input(3) >> 3)

# Python speed test on output 0 toggling at max speed
while (True):
  mcp.output(0, 1)  # Pin 0 High
  mcp.output(0, 0)  # Pin 1 Low
```

This file contains both the base MCP230xx class that makes it easy to use the chip, along with a very simple demo that will toggle a single pin as fast as possible. The example code shows how you can set pins to both input and output:

Instantiating an instance of Adafruit_MCP230xx

To instantiate an instance of the wrapper class that allows you to access the MCP230xx, you need to uncomment one of the two lines at the top of the above code. There are two options because earlier versions of the Pi Model B (pre 512MB SDRAM) used I2C0, whereas the latest Model B devices (with 512MB SDRAM) use I2C1.

The address assumes you are using an MCP23017 with all three address pins set to GND. If you are using a different address pin configuration, you can open up the datasheet to see how the address scheme works (MCP23017 datasheet (https://adafru.it/aRo) or .the MCP23008 datasheet (https://adafru.it/aRp).)

```
# Use busnum = 0 for older Raspberry Pi's (pre 512MB)
mcp = Adafruit_MCP230XX(busnum = 0, address = 0x20, num_gpios = 16)

# Use busnum = 1 for new Raspberry Pi's (512MB)
# mcp = Adafruit_MCP230XX(busnum = 1, address = 0x20, num_gpios = 16)
```

Pin Numbering

The MCP23008 has 8 pins - A0 thru A7. **A0** is called **0** in the library, and **A7** is called **7** (the rest follow the same pattern)

The MCP23017 has 16 pins - A0 thru A7 + B0 thru B7. **A0** is called **0** in the library, and **A7** is called **7**, then **B0** continues from there as is called **8** and finally **B7** is pin **15**

Setting a pin as Input

You can enable or disable the internal pullup resistor and set the pins as input with the following lines of code:

```
# Set pin 3 to input with the pullup resistor enabled
mcp.pullup(3, 1)

# Read pin 3 and display the results
print "%d: %x" % (3, mcp.input(3) >> 3)
```

The second line reads pin 3, and shifts the value left 3 bits so that it will equal 0 or 1 depending on whether the pin is high or low when it is sampled. This will results in output similar to the following: "3: 0" or "3: 1" (depending on the pin state).

Setting a pin as Output

To set a pin as output, you also need two lines of code:

```
# Set pin 0 to output (you can set pins 0..15 this way)
mcp.config(0, OUTPUT)

# Set pin 0 High
mcp.output(0, 1)

# Set pin 0 Low
mcp.output(0, 0)
```

That's all there is to it! The default sample code will toggle the GPIO pin as fast as possible, and if you hooked it up to an oscilloscope you'd end up with something similar to the following:

Interrupts & Callbacks

As it currently stands, the library does not support any sort of interrupt or call back functionality (there is a hardware interrupt pin on the MCP but we don't use it in this code). Only polling is currently supported!