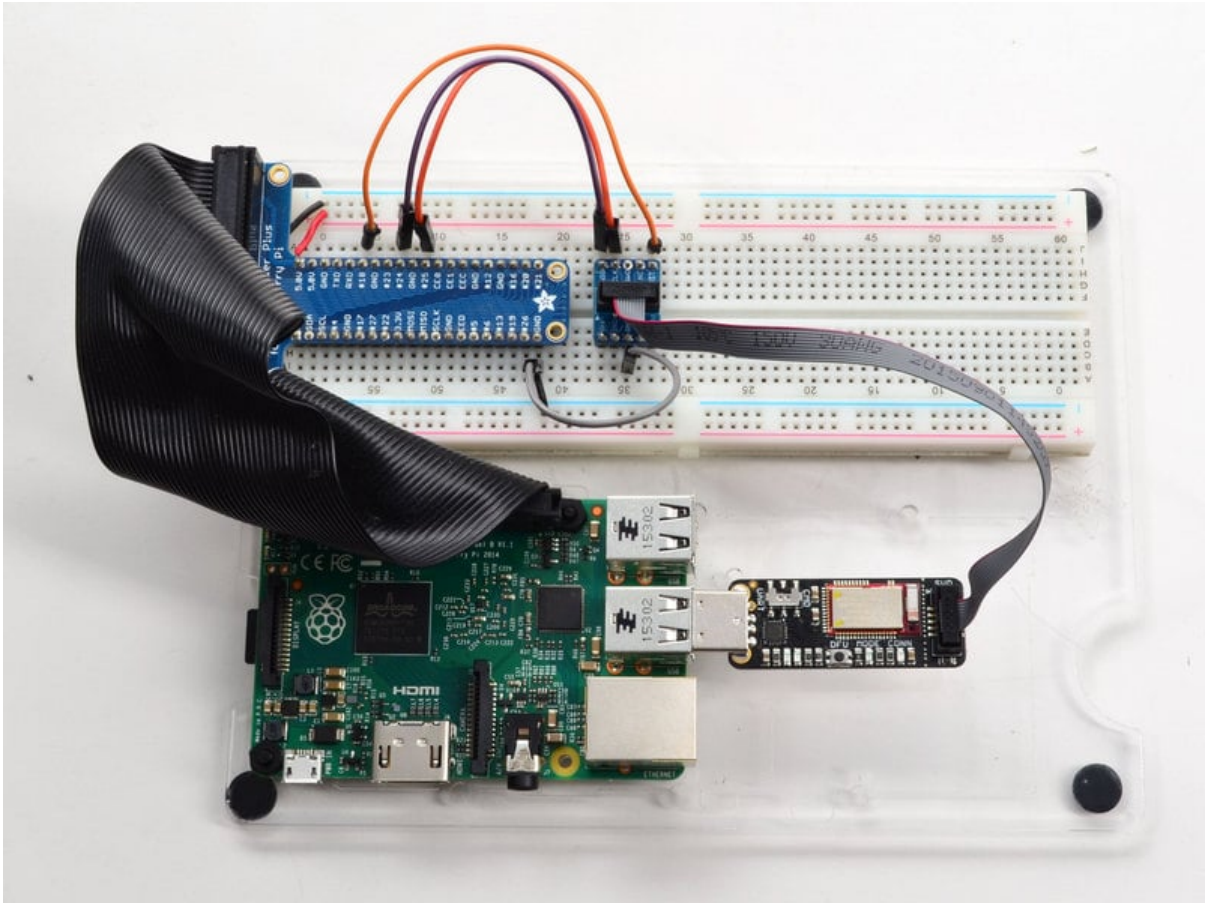
Programming Microcontrollers using OpenOCD on a Raspberry Pi

Created by lady ada

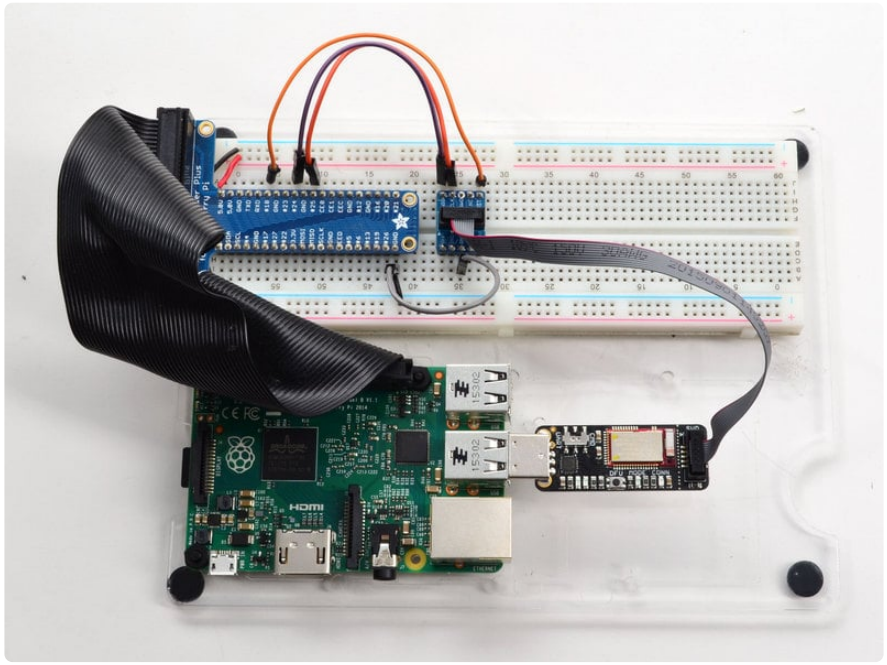


https://learn.adafruit.com/programming-microcontrollers-using-openocd-on-raspberry-pi

Last updated on 2025-05-29 04:50:44 PM EDT

Table of Contents

Overview



Yay you have finally moved on from 8-bit chips and are ready to try out some 32-bit hotness! Those ARM Cortex chips look fun, some have built in bluetooth, or 2.4ghz radios, or usb...all you have to do is learn how to program them.

OpenOCD

On your way to learning how to use your favorite new ARM Cortex you may have heard of OpenOCD (https://adafru.it/fMy). OpenOCD is the software that we will use to do the actual programming of chips. Unlike the AVR ISP programming protocol, every ARM chip is significantly different to program, with platform-unique commands, flash locations, fuse bits, settings, etc. Teasing out those details is a struggle and if you change chips you have to start all over even if both chips are, say, Cortex-M3 based!

Each chip fab tends to supply its own programming software - Atmel has Atmel Studio, Nordic has NRFGo, ST has ST Link - but often times that software is Windows only.

OpenOCD is great because its cross platform, open source, and has support for a vast number of chips & programmers.

You can use OpenOCD with dongle-programmers such as J-Link and ST-Link or even an FTDI chip. But, if you have a spare Raspberry Pi (and who doesn't these days?) you can use it as a native OpenOCD programmer with just a few wires.

It's also really fast to program chips natively, and if you have to program a mess of chips, it can make things speedy - an extra 30 seconds adds up when you're doing 1000!

Compiling OpenOCD

Compiling OpenOCD takes about 15 minutes but is worth the effort to get the latest code. You'll need to have command line access and a Pi on the Internet so you can download packages and software.

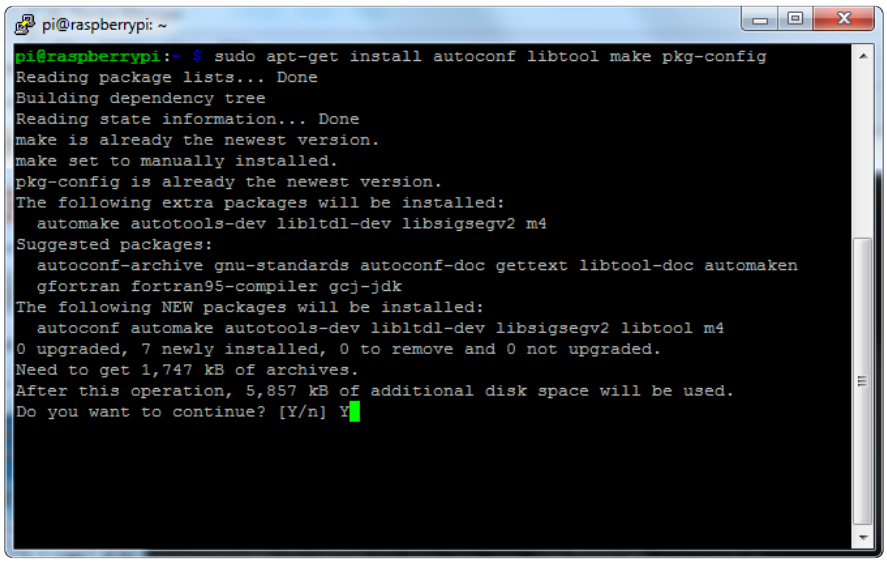
Thanks to https://petervanhoyweghen.wordpress.com/2015/10/11/burning-zero-bootloader-with-beaglebone-as-swd-programmer/ (https://adafru.it/mbC) for the great tutorial, we're just adapting it for Pi usage!

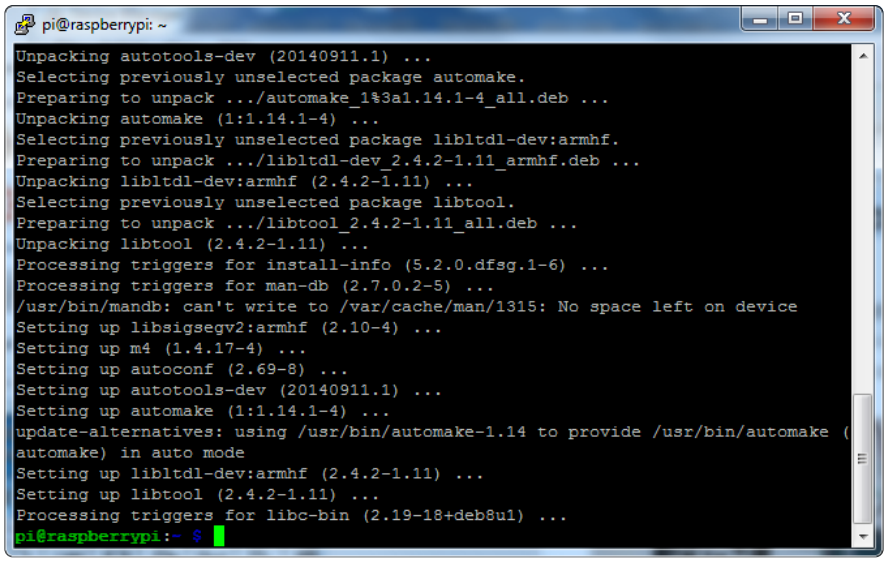
Compiling OpenOCD

Start by doing a fresh **sudo apt-get update** this will make sure you have the latest packages and repository set up.

Install all the tools you'll need to compile OpenOCD. OpenOCD changes a lot and is under constant development so we do suggest compiling your own!

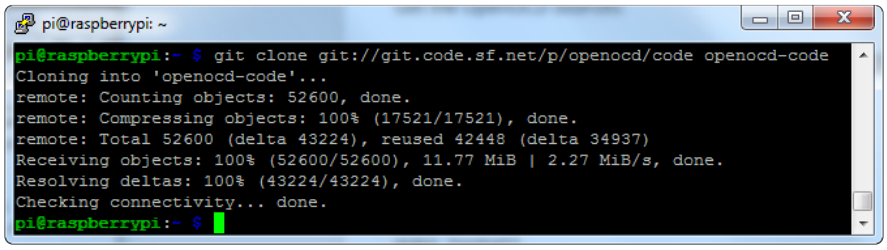
```
sudo apt-get install git autoconf libtool make pkg-config libusb-1.0-0 libusb-1.0-0-dev libjim-dev
```





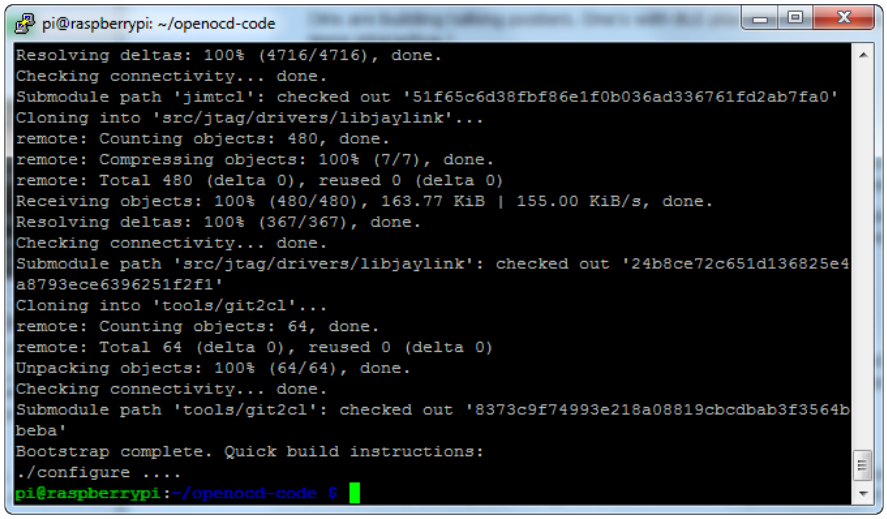
Download the latest source code for OpenOCD with:

```
git clone http://openocd.zylin.com/openocd
```



Change into the code directory and run the bootstrapper with:

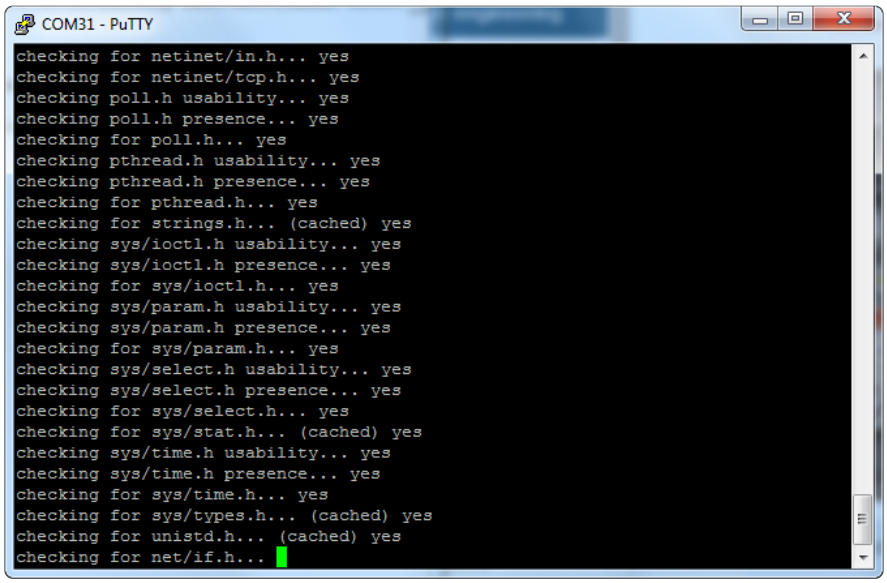
```
cd openocd
./bootstrap
```

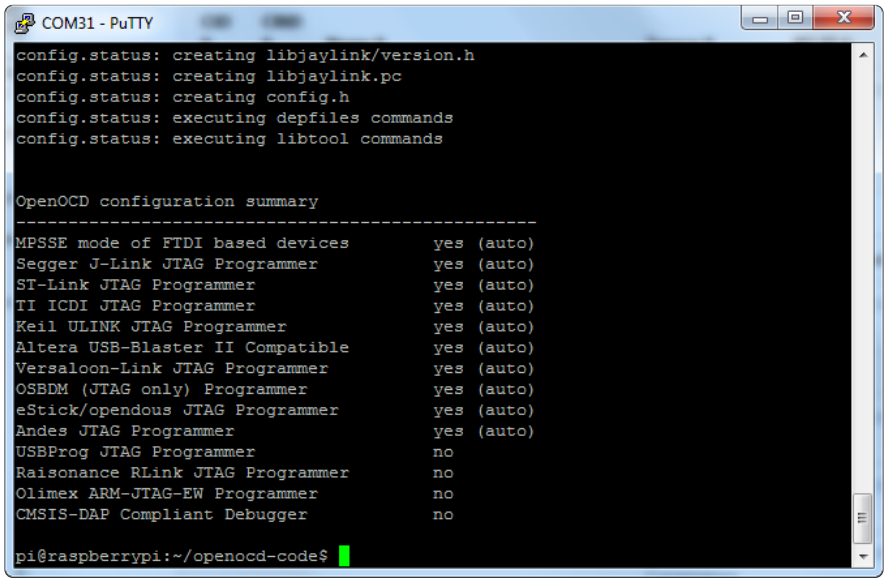


Next, we will compile OpenOCD with the Raspberry Pi native GPIO twiddling support - this will work on various Raspberry Pi's despite being called 'bcm2835gpio'

If you're following this guide on a non-Pi embedded linux board, you can skip the **--enable-bcm2835gpio** part and try to just use sysfsgpio. Sysfsgpio is much slower than native GPIO twiddling but it may not matter too much in your application.

```
./configure --enable-sysfsgpio --enable-bcm2835gpio
```

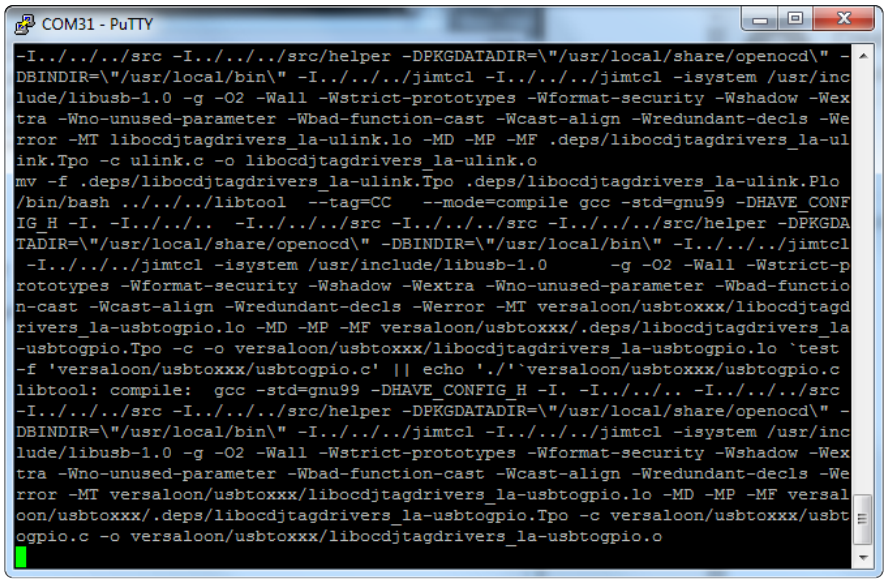


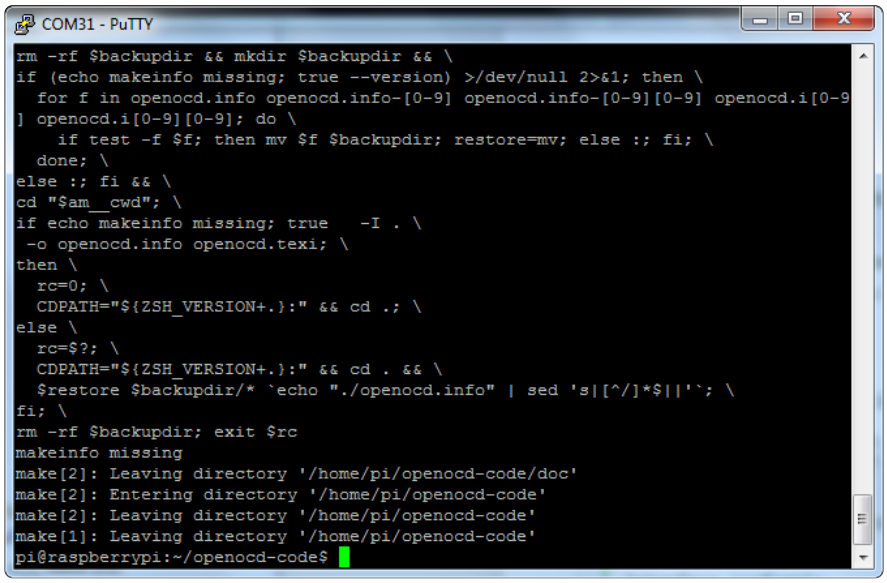


Note that when done, it wont mention GPIO support in the configuration summary, thats OK!

Run make. This can take a few minutes so you can also pass it the -j$(nproc) to build across all your processor cores.

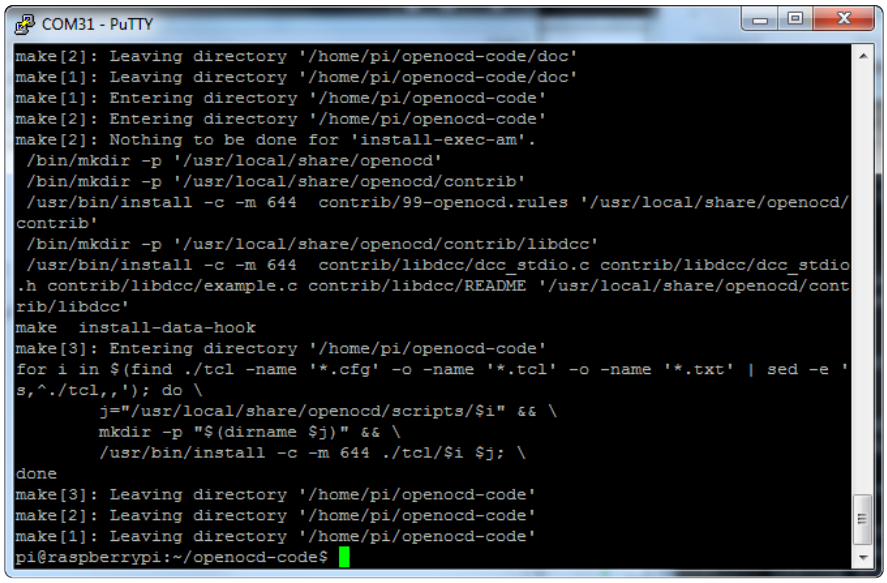
```
make
```





Assuming compilation completes successfully as above, you can install with:

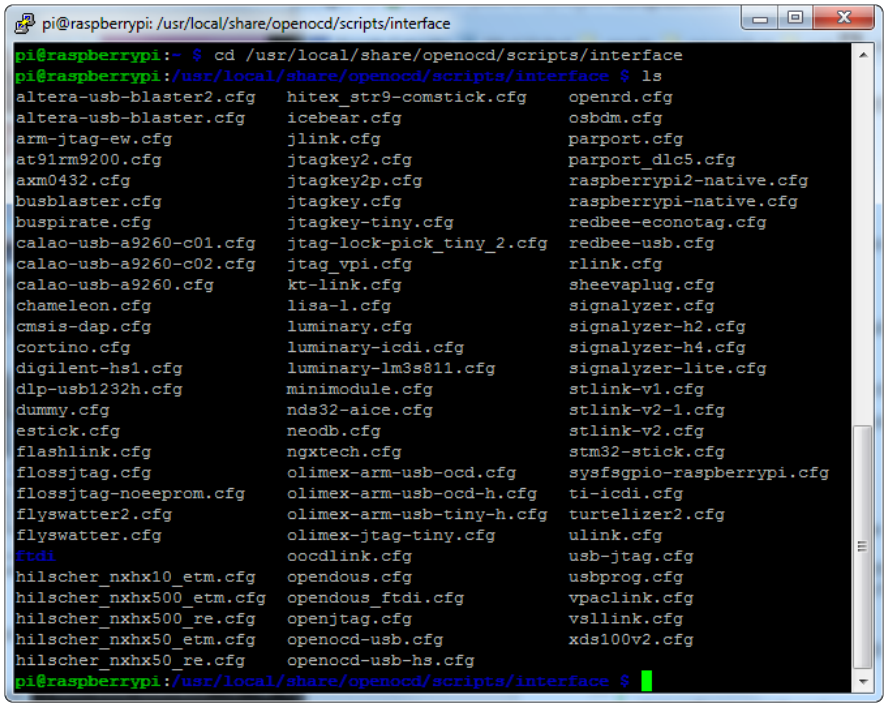
```
sudo make install
```

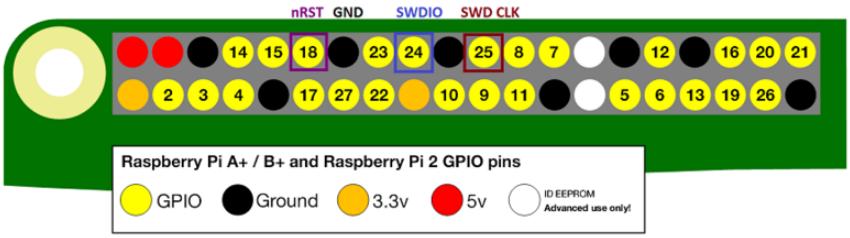


That's pretty much it!

You can see the list of interfaces available in **/usr/local/share/openocd/scripts/interface**

There's a lot of options, in particular check out **raspberrypi2-native.cfg** and **raspberrypi-native.cfg** if you are interested in using OpenOCD with a non-Pi, look at **sysfsgpio-raspberrypi.cfg** which can help you port to a different linux computer





Wiring and Test

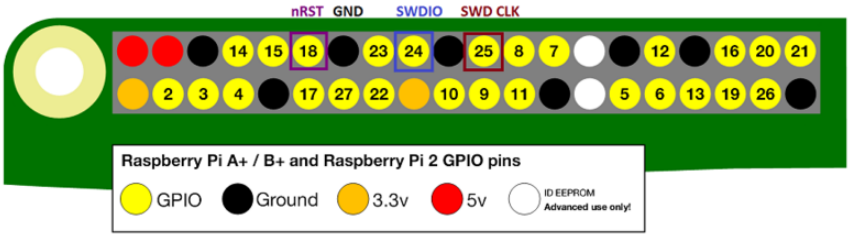
Connecting to Target

OK you've done the compiling, now you are ready to connect!

In this case, I'll be connecting to an Atmel ATSAMD21G18 Cortex-M0 over SWD and uploading the Arduino bootloader to it. You can, of course, connect to any processor that OpenOCD supports but this is the one I've got handy

Wire up the target to SWD

Of course connections must be made! [Note that we are using the "BCM" pin numbering convention](https://adafru.it/jEa) (https://adafru.it/jEa)

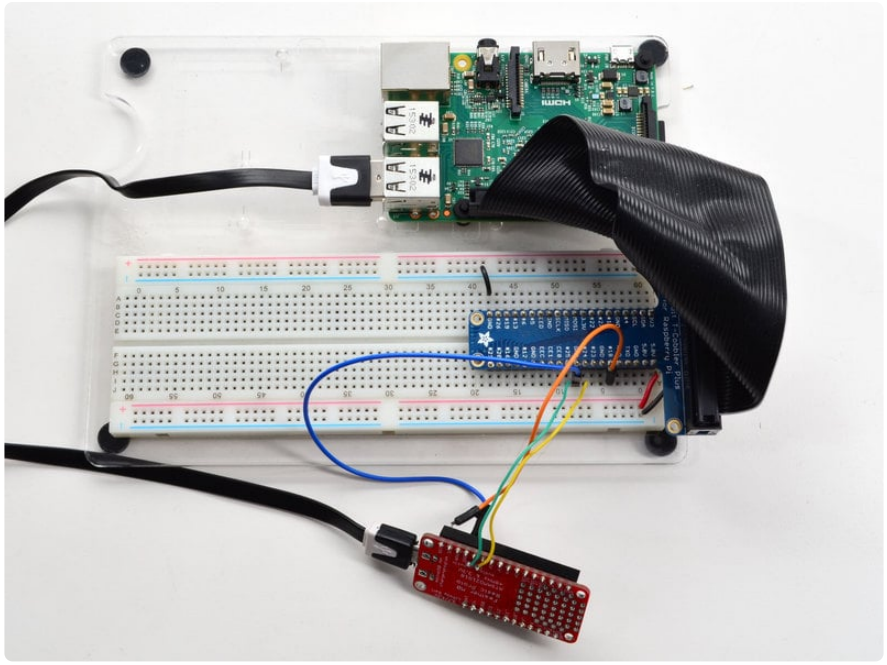


Connect:

- **Target GND** to **Pi GND**
- **Target SWDIO** to **Raspberry Pi #24**
- **Target SWCLK** to **Raspberry Pi #25**
- **Target Reset** to **Raspberry Pi #18** (may not be required)
- If powering the chip directly from the Pi, connect **3.3V** to **3.3V** (I'm just powering the chip over USB)

Of course, this assumes that your chip is running at 3.3V logic. For 1.8 or 5V logic, level shifting may be required.

You can later change the pins used in the **interfaces** configuration file but for now, I suggest just going with the default



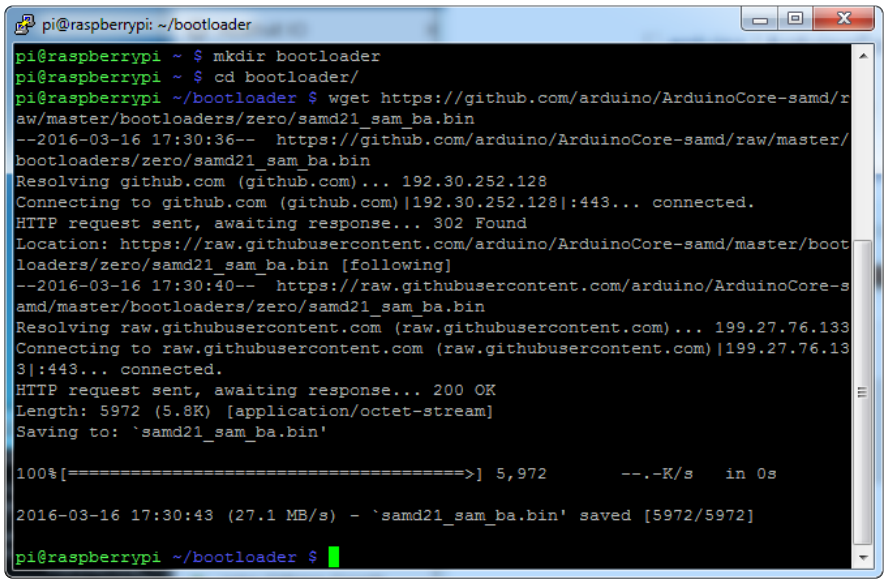
Create OpenOCD config

The easiest way to connect is creating a new directory in your home dir

cd ~
mkdir bootloader
cd bootloader

and then putting the file you want to program there, in this case I'm going to just grab the latest Arduino Zero bootloader (of course, substitute your own binary or hex!)

wget https://github.com/arduino/ArduinoCore-samd/raw/master/bootloaders/zero/samd21_sam_ba.bin



In the same directory, make a new file called **openocd.cfg**

nano openocd.cfg

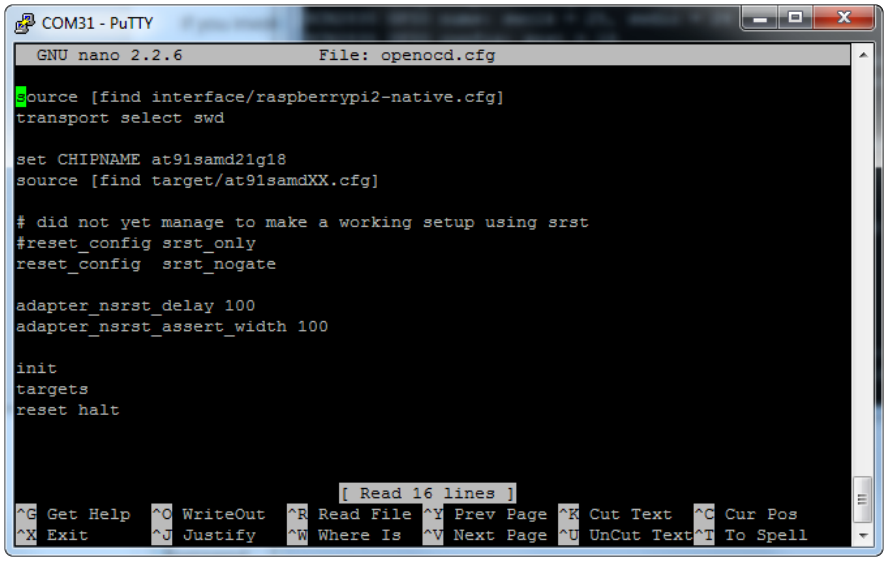
and put the following into it:

```
source [find interface/raspberrypi2-native.cfg]
transport select swd

set CHIPNAME at91samd21g18
source [find target/at91samdXX.cfg]

# did not yet manage to make a working setup using srst
#reset_config srst_only
reset_config  srst_nogate

adapter_nsrst_delay 100
adapter_nsrst_assert_width 100

init
targets
reset halt
```

Change **raspberrypi2-native.cfg** to whatever config you are using, e.g. for a Pi Zero or 1 use **raspberrypi1-native.cfg** or **raspberrypi-native.cfg**

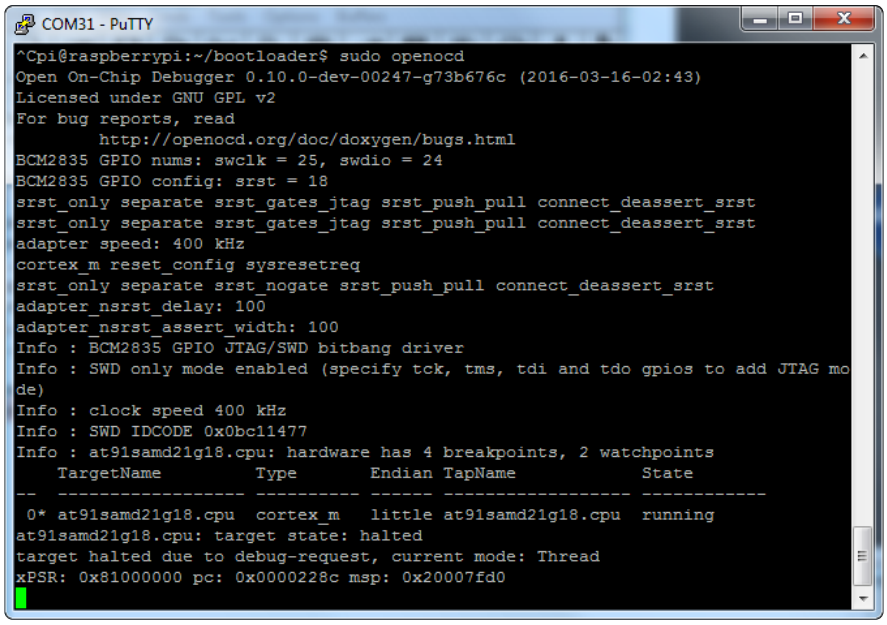
If you're using a Pi Zero/1 you may also need to add

```
bcm2835gpio_swd_nums 25 24
bcm2835gpio_trst_num 7
bcm2835gpio_srst_num 18
```



You may need to also comment out `reset_config  srst_nogate`, some people report that is required to make it work

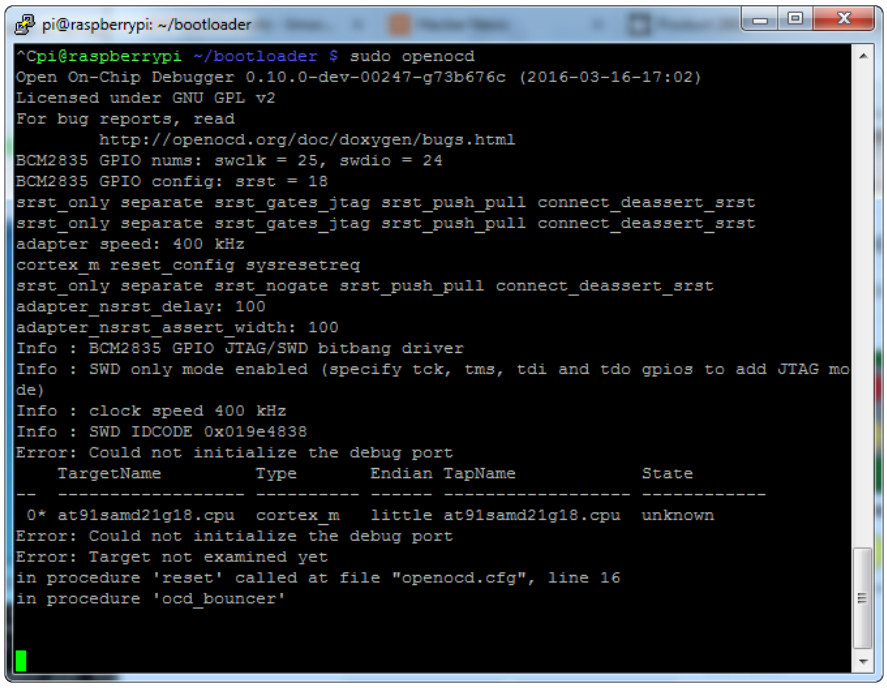
Save the config file and then run **sudo openocd** (no other args, its all in the config!) in the directory. You should get the following indicating a good connection



In particular make sure you get that **target state:halted** to you know it was able to connect!

Hit control-C to cancel out of openocd.

If you get **unknown** for state, or other errors, check your wiring! You may also need to powercycle or disconnect parts from the chip to get it into a good programming state. You may also need to change the programming frequency



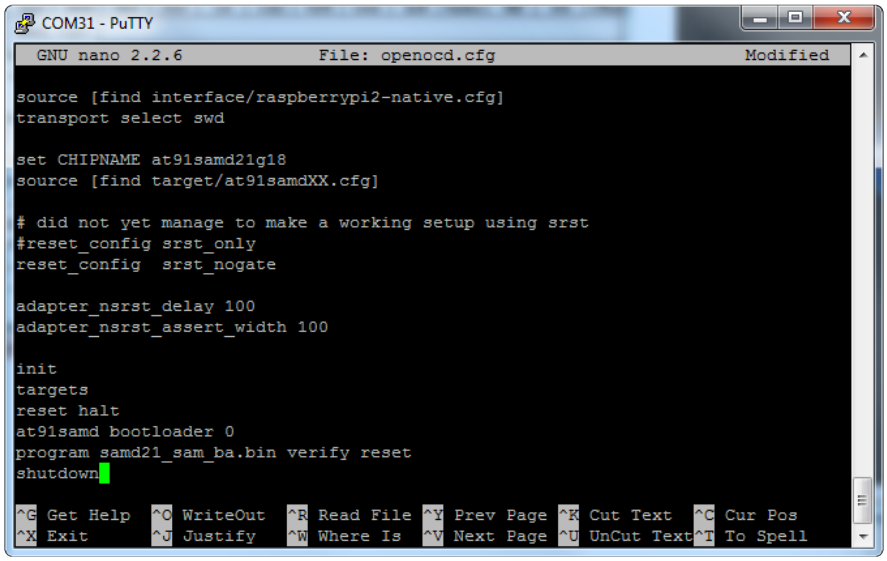
Hit control-C to cancel out of openocd (or you can **telnet 127.0.0.1 4444** if you want to send commands, won't be covered here.

Now you can change the **openocd.cfg** with nano to add commands for burning the binary file. At the bottom put in:

```
init
targets
reset halt
at91samd bootloader 0
program samd21_sam_ba verify
at91samd bootloader 8192
reset
shutdown
```

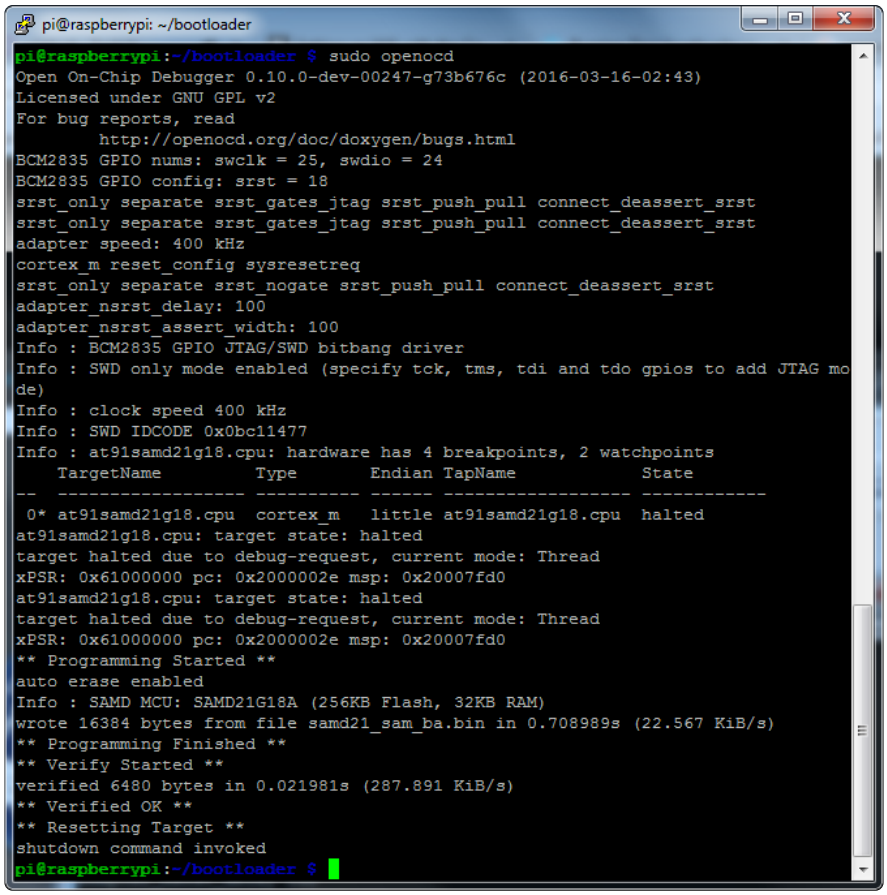
This will init, look for targets, reset and halt the chip, turn off bootloader protection, burn in the bootloader file and verify it, re-turn-on bootloader protection, reset and shutdown openocd

You can skip the bootloader protection parts if you are not burning in a bootloader, of course



Of course, change the commands if you have a different file name, different chip, etc.

Save the file and run **sudo openocd** again:



Zoom! Programmed the bootloader in 0.02 seconds!

More Options

If you don't want to set up the configure file, you can actually do it all from the command line:

sudo openocd -f interface/raspberrypi2-native.cfg -c "transport select swd; set WORKAREASIZE 0; adapter_nsrst_delay 100; adapter_nsrst_assert_width 100; source [find target/nrf51.cfg]" -c "init; reset; halt; nrf51 mass_erase; reset" -c "shutdown"

This will, for example, erase and reset a Nordic nRF51822 (which is a pretty finicky chip by the way, you may need to do hard resets to get it to talk to openocd)

```
pi@raspberrypi: ~
pi@raspberrypi ~ $ sudo openocd -f interface/raspberrypi2-native.cfg -c "transp
ort select swd; set WORKAREASIZE 0; adapter_nsrst_delay 100; adapter_nsrst_asse
rt_width 100; source [find target/nrf51.cfg]" -c "init; halt; nrf51 mass_erase;
 reset" -c "shutdown"
Open On-Chip Debugger 0.10.0-dev-00247-g73b676c (2016-03-16-17:02)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
BCM2835 GPIO nums: swclk = 25, swdio = 24
BCM2835 GPIO config: srst = 18
srst_only separate srst_gates_jtag srst_push_pull connect_deassert_srst
adapter_nsrst_delay: 100
adapter_nsrst_assert_width: 100
cortex_m reset_config sysresetreq
adapter speed: 1000 kHz
Info : BCM2835 GPIO JTAG/SWD bitbang driver
Info : SWD only mode enabled (specify tck, tms, tdi and tdo gpios to add JTAG m
ode)
Info : clock speed 1001 kHz
Info : SWD IDCODE 0x0bb11477
Info : nrf51.cpu: hardware has 4 breakpoints, 2 watchpoints
Info : nRF51822-QFAC(build code: A1) 256kB Flash
Error: nrf51.cpu -- clearing lockup after double fault
nrf51.cpu: target state: halted
target halted due to debug-request, current mode: Handler HardFault
xPSR: 0xc1000003 pc: 0xfffffffe msp: 0xfffffff8
Polling target nrf51.cpu failed, trying to reexamine
Info : nrf51.cpu: hardware has 4 breakpoints, 2 watchpoints
shutdown command invoked
pi@raspberrypi ~ $
```