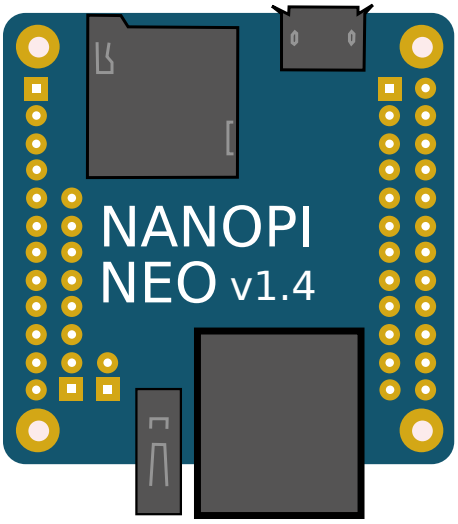
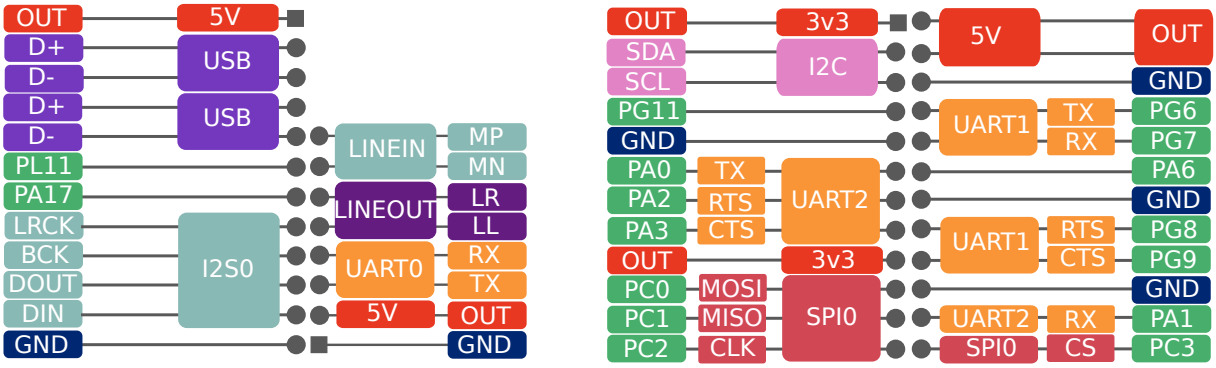
Introduction





This handout accompanies the tutorial, An Introduction to Hardware Hacking on FreeBSD as presented by Tom Jones (tj@enoti.me) at EuroBSDCon 2019. The diagram above labels the outputs on the NanoPi-NEOLTS v1.4 from FriendlyElec.

The NanoPi diagram above shows where connections on are broken out to pin headers, this diagram will be helpful throughout the tutorial to locate connections on the board.

SD Card Image

The NanoPi is running a recent FreeBSD-13 CURRENT with some helpful scripts and programs on the SD card. The programs directory contains the source for the tools in `/root/bin`.

```
root@generic:~ # ls *
programs/
scripts/
overlays/
```

FreeBSD on the NanoPi uses GENERICSD image. This image requires a bootloader to be added before it will work. We can prepare a single image to be copied to many SD cards by

using a memory disk as an intermediate step.

The latest image is as I write is 13 CURRENT from 20190829, it must be decompressed before it can be used:

```
$ fetch ftp://ftp.freebsd.org/pub/FreeBSD/snapshots/arm/armv7/ISO-IMAGES/13.0/
    FreeBSD-13.0-CURRENT-arm-armv7-GENERICSD-20190829-r351591.img.xz
$ xz -d FreeBSD-13.0-CURRENT-arm-armv7-GENERICSD-20190829-r351591.img.xz
```

Each u-boot bootloader platform has its own package, currently there are 46 different bootloaders in the FreeBSD ports system. We want the u-boot for the nanopi_neo (our target).

The `u-boot-nanopi_neo` package contains the binary bootloader we need in `u-boot-sunxi-with-spl.bin`.

```
# pkg install u-boot-nanopi_neo-2019.07
$ pkg info -l u-boot-nanopi_neo-2019.07 | grep bin
    /usr/local/share/u-boot/u-boot-nanopi_neo/u-boot-sunxi-with-spl.bin
```

With the GENERICSD image and the bootloader we need to create the memory disk image we will use for staging. First we need to create a large enough backing file.

```
$ truncate -s 8G nanopi.img
# mdconfig -f nanopi.img
md0
```

Now we can ‘dd‘ the GENERICSD image to the memory disk

```
# dd if=FreeBSD-13.0-CURRENT-arm-armv7-GENERICSD-20190829-r351591.img of=/dev/md0 bs
    =1m
```

We need to ‘dd‘ the bootloader to the start of the SD card, i.e. the entire device and not a partition.

```
# dd if=/usr/local/share/u-boot/u-boot-nanopi_neo/u-boot-sunxi-with-spl.bin of=/dev/
    da0 bs=1k seek=8 conv=sync
```

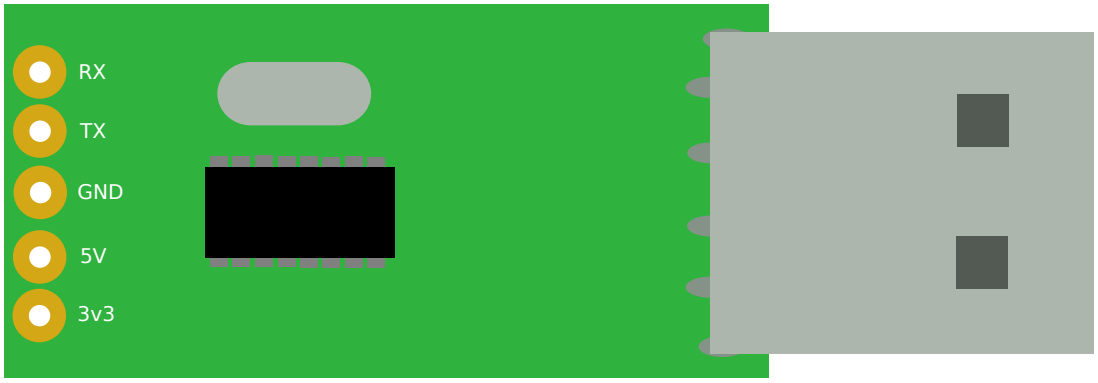
With the memory disk attached we can interact with the image file as if it were a real USB drive or SD card. We can mount the root partition of the SD card and modify or add any files we wish:

```
# mount /dev/md0sa mnt
# cp -r tutorialfiles/* mnt/root/
```

When we are done changing things we have to disconnect the memory disk. Finally we can copy the memory disk to a real SD card using ‘dd‘:

```
# sudo mdconfig -d -u md0
# sudo dd if=nanopi.img of=/dev/da0 bs=1m
```

Activity 1: Connecting to the board and controlling the status LED



In this activity we are going to connect to the NanoPi and interact with the status LED. The NanoPi has a green status LED next to the red power LED, this LED is connected to GPIO `PA10`.

Activity 1a: Connecting to the NanoPi

To connect the USB serial port to the NanoPi we need to connect the `RX`, `TX` and `GND` wires on the inside left pin header to the `TX`, `RX` and `GND` pins on the USB serial. Next you need to find where your operating system has created the device for the serial interface.

- **FreeBSD:** /dev/ttyUx where x is probaby 0
- **Linux:** /dev/ttyUx where x is probaby 0
- **Mac OS:** /dev/usbserial.1420 or /dev/usbserial.1410
- **Windows:** comX, maybe 6

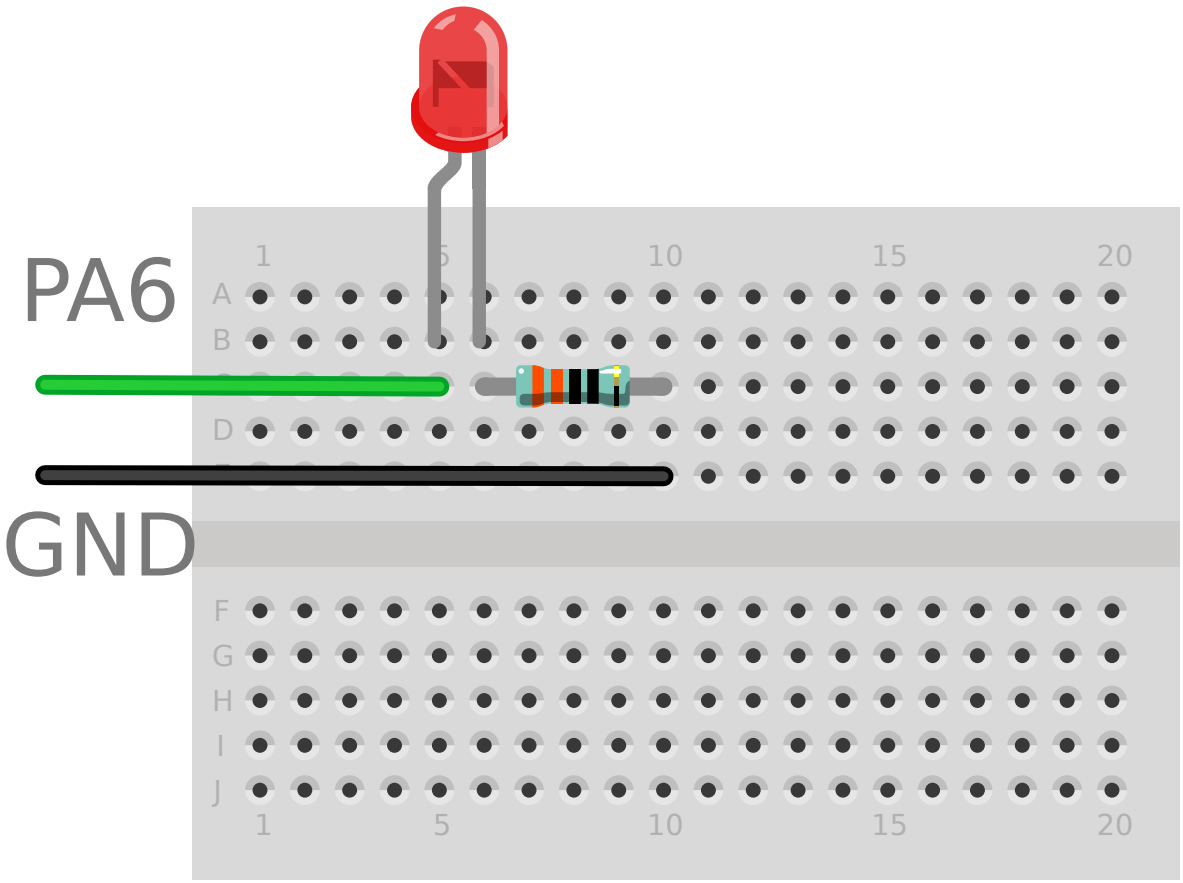
```
root@laptop: # sudo cu -l /dev/ttyU0 -s 11500
```

Once connected you can login as root with **root**:**root**.

Activity 1b: Controlling the LED

```
root@generic:~ # ls /dev/gpioc*
root@generic:~ # gpioctl -l | grep PA10
pin 10: 0       PA10<OUT>
root@generic:~ # gpioctl 10 1
root@generic:~ # gpioctl 10 0
root@generic:~ # gpioctl -t 10
```

Activity 2: Making our First Cicruit



We are going to use a breadboard to connect an external LED to the NanoPi.

Activity 2a: Wiring the Circuit

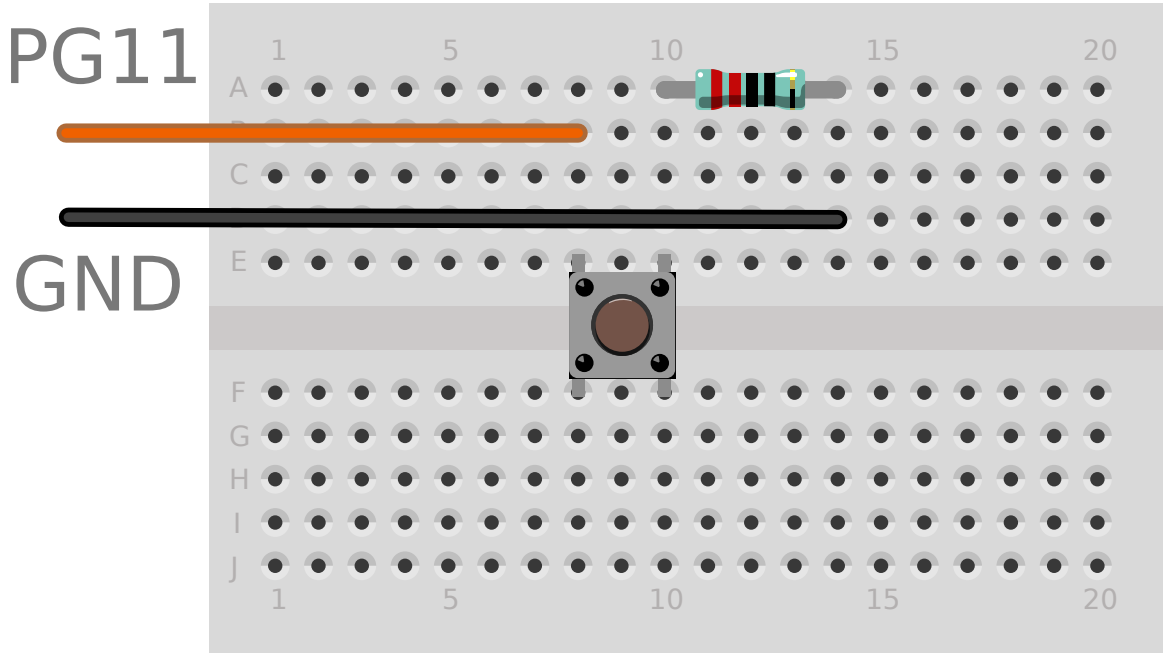
Connect the long leg of the LED with a jumper with to pin `PA6` on the NanoPi. Connect the short leg of the LED to a resistor. Connect the other leg of the resistor with a jumper with to `GND` on the NanoPi

Activity 2b: Controlling the LED

This time we have to configure the GPIO as an output.

```
root@generic:~ # gpioctl -l | grep PA6
pin 06: 0       PA6<>
root@generic:~ # gpioctl -c 6 OUT
root@generic:~ # gpioctl -l | grep PA6
pin 06: 0       PA6<OUT>
```

Activity 3: Reading an Input



We are going to connect a button to the NanoPi and read from it as an input.

Activity 3a: Wiring the Circuit

Connect `PG11` to one side of the button with a jumper wire. Connect the otherside of the button to a resistor. Connect the other leg of the resistor to `GND` on the NanoPi using jumper wire.

Activity 3b: Configuring the Output

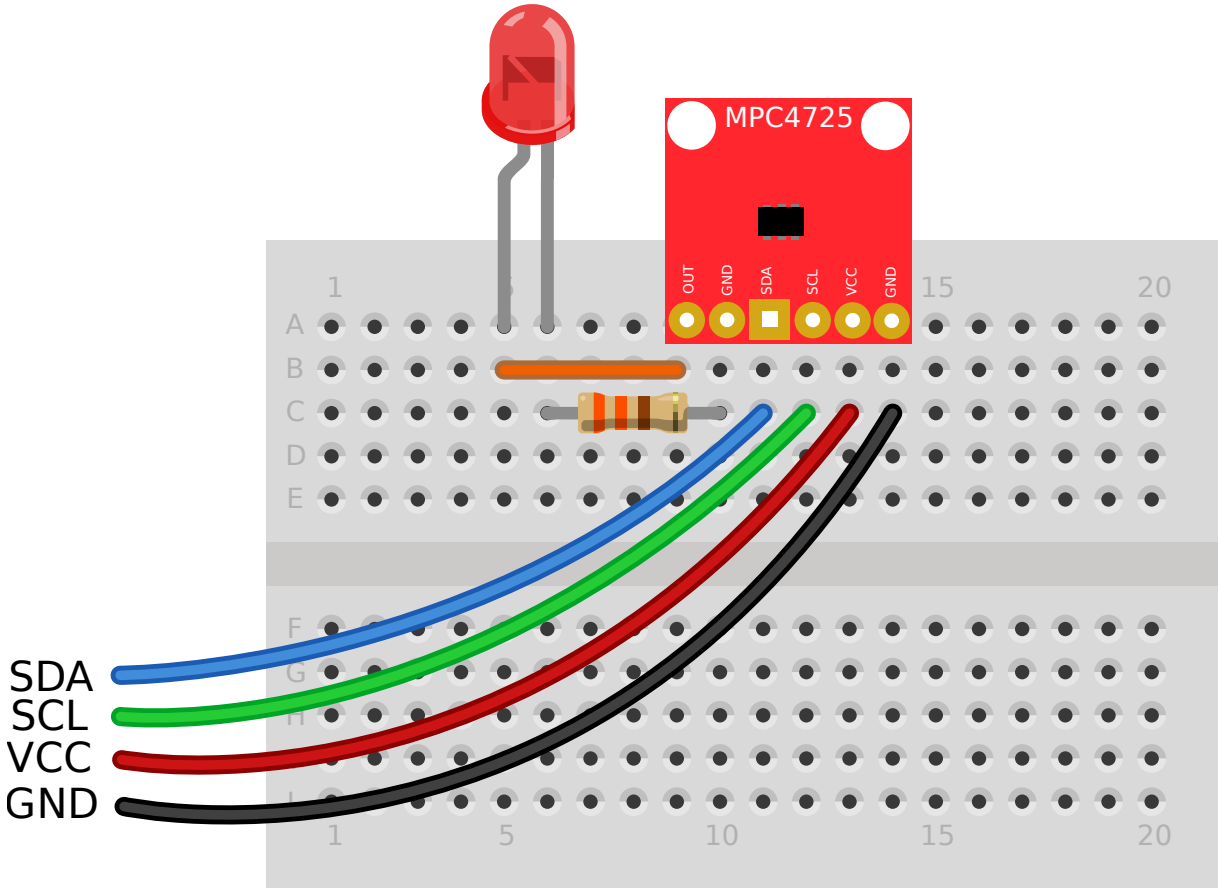
We have to configure our GPIO as an input with a pull up resistor.

```
root@generic:~ # gpioctl -l | grep PG11
pin 91: 0        PG11<>
root@generic:~ # gpioctl -c 91 IN PU
root@generic:~ # gpioctl -l | grep PG11
pin 91: 1        PG11<IN,PU>
```

```
root@generic:~ # gpioctl 91
1
root@generic:~ # gpioctl 91
0
```

When the button is not pressed the GPIO is pulled up to 3v3 by the internal pull up resistor. When we press the button we connect the GPIO to GND through the resistor, pulling it down to ground.

Activity 4: Adding Analog Output with I2C



We are going to connect the NanoPi to a MPC4725 Digital to Analog Converter (DAC) using the I2C bus. We must make the circuit and configure the NanoPi to use I2C. We will connect an LED to the analog output and use the MPC4725 to dim the LED.

Activity 4a: Enabling I2C

Before we can use I2C we need to load an FDT overlay to enable the I2C hardware.

Add the following to /boot/loader.conf

```
fdt_overlays="sun8i-h3-i2c0.dtbo"
```

Activity 4b: Wiring the Circuit

Put the MP4725 (the small red board) on the breadboard. Connect `VIN` to 5V on the NanoPi and `GND`, `SDA` and `SCL` to the corresponding IO on the NanoPi. Connect the long leg of a LED to the OUT pin on the MPC4725, through a resistor to the `GND` on the breadboard.

Activity 4c: Using the I2C Bus

The I2C bus is addressable, we can scan the bus to find the address the MPC4725 uses.

```
root@generic:~ # i2c -s
Hardware may not support START/STOP scanning; trying less-reliable read method.
Scanning I2C devices on /dev/iic0: 62
```

The MPC4725 datasheet tells us how to send commands to the device and the format of the data. We are going to use the Fast Write command to set the DAC value without storing it in the EEPROM.

The Fast Write command is 16 bits long. The first 4 bits are the command, in this case all zero. The next 12 bits are the value to set the DAC to. We can set the analog value fully on (the LED is fully lit) by sending the command bits as `0000 1111 1111 1111` binary or `00FF FFFF` hexidecimal.

`printf` on FreeBSD doesn't support hexidecimal literals, so we have to use octal literals. Octal `printf` are in the form of a slash followed by a number such as "`\17`".

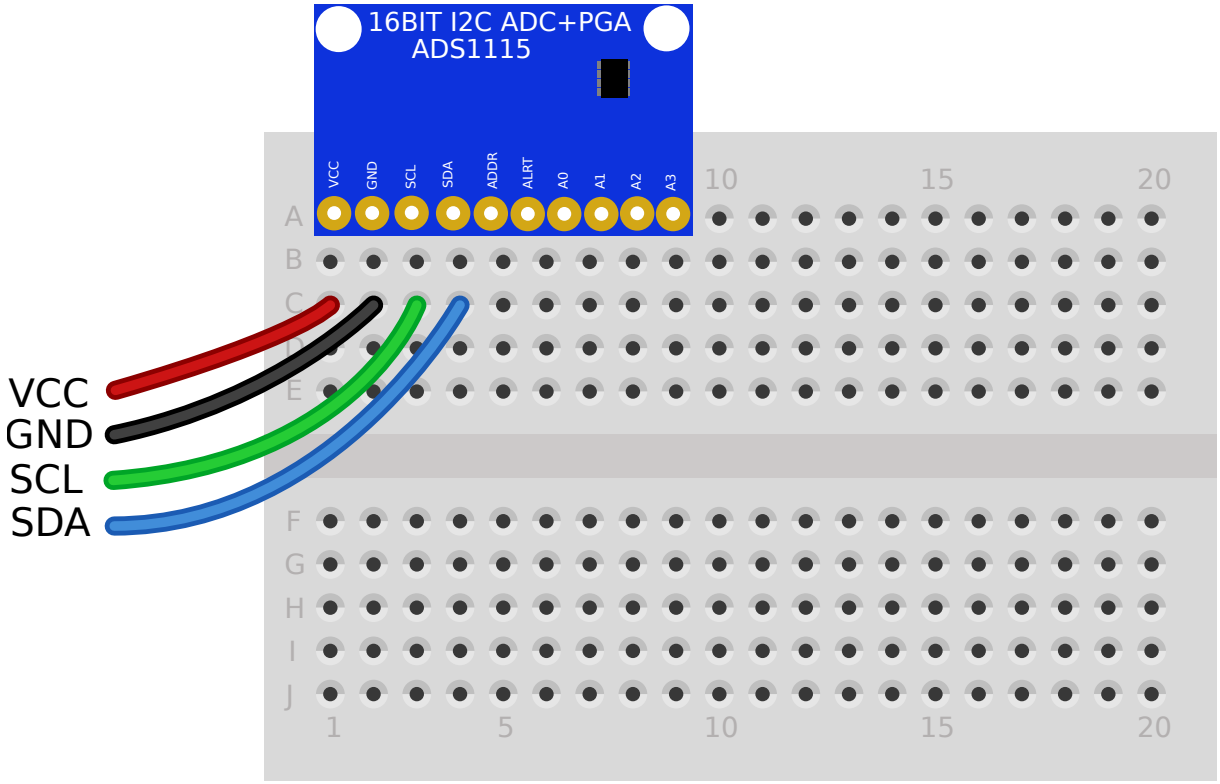
```
# turn on LED:
root@generic:~ # printf "\17\377" | i2c -a 0x62 -d w -c 2 -m tr
# turn off LED:
root@generic:~ # printf "\0\0" | i2c -a 0x62 -d w -c 2 -m tr
```

```
root@generic:~ # cd programs/mpc4725-fade
root@generic:~ # make
root@generic:~ # ./mpc4725-fade
```

Activity 5:

We are going to connect the NanoPi to a ADS1115 analog to digital converter and read some sensor values. The ADS1115 has 4 ADC channels that can be configured as either comparators or as values referenced to ground.

The ADS1115 is supported by FreeBSD with a kernel driver, we have to use an FDT overlay to tell the system to use it.



The ADS1115 is an I2C device, to connect it to the NanoPi you need to connect VCC to 5v, SDA to SDA, SCL to SCL and GND to GND.

Activity 5a: Enabling the ADS1115

The `ads111x(4)` man page tells us how to write part of the overlay. We need to augment the overlay we are using for I2C and add the ADS1115 as a child device. The complete overlay we are going to use is in `/overlays/ads111x.dts`:

```
1  /dts-v1/;
2  /plugin/;
3
4  / {
5          compatible = "allwinner,sun8i-h3";
6  };
7
8  &{/soc/i2c@1c2ac00} {
9
10         status = "okay";
```

```
11          pinctrl-names= "default";
12          pinctrl-0 = <&i2c0_pins>;
13
14          adc@48 {
15                  compatible = "ti,ads1115";
16                  reg = <0x48>;
17                  status = "okay";
18                  #address-cells = <1>;
19                  #size-cells = <0>;
20          };
21  };
```

The overlay has be compiled into a device tree blob and copied into the overlays directory:

```
1 # dtc -I dts -O dtb -o ads111x.dtbo ads111x.dts
2 # mv ads111x.dtbo /boot/dtb/overlays
```

We then need to substitute the overlay for the previous one we set up:

```
1 fdt_overlays="ads111x.dtbo"
```

To enable the new overlay reboot the NanoPi.

On reboot you can check for the ADS1115 in `dmesg` and in `sysctl`.

```
1  root@generic:~ # dmesg | grep ads111x
2  ads111x0: <ADS1115> at addr 0x90 on iicbus0
3
4  root@generic:~ # sysctl dev.ads111x
5  dev.ads111x.0.4.rate_index: 4
6  dev.ads111x.0.4.gain_index: 1
7  dev.ads111x.0.5.rate_index: 4
8  dev.ads111x.0.5.gain_index: 1
9  dev.ads111x.0.6.rate_index: 4
10 dev.ads111x.0.6.gain_index: 3
11 dev.ads111x.0.7.rate_index: 4
12 dev.ads111x.0.7.gain_index: 1
13 dev.ads111x.0.hi_thresh: 32767
14 dev.ads111x.0.lo_thresh: -32768
15 dev.ads111x.0.config: 3
16 dev.ads111x.0.%parent: iicbus0
17 dev.ads111x.0.%pnpinfo: name=adc@48 compat=ti,ads1115
18 dev.ads111x.0.%location: addr=0x90
19 dev.ads111x.0.%driver: ads111x
20 dev.ads111x.0.%desc: ADS1115
21 dev.ads111x.%parent:
```

Activity 5b: Reading from ADS111x

In a full sysctl listing the channel voltage is not displayed, it is only displayed when accessed directly. To read the voltage on the ADS1115 A0 we need to read from channel 4.

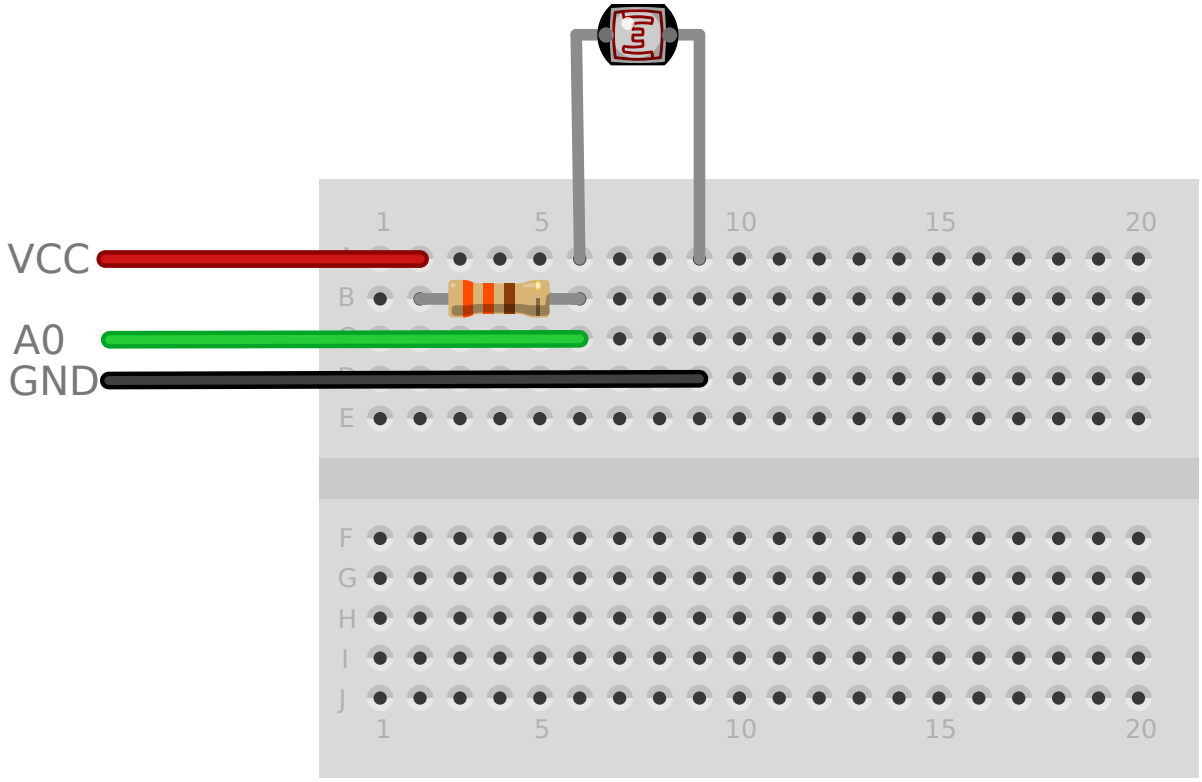
```
# sysctl hw.dev.ads111x.0.4.voltage
```

There is a script in `/scripts` to convert and watch the voltage of an ADC. With no circuit connected you will see a low 'floating' voltage on the ADC.

```
# sh ~/scripts/watch-adc.sh 4
```

Activity 5c: Connection Analog Circuits

Activty 5c: LDR Voltage Divider

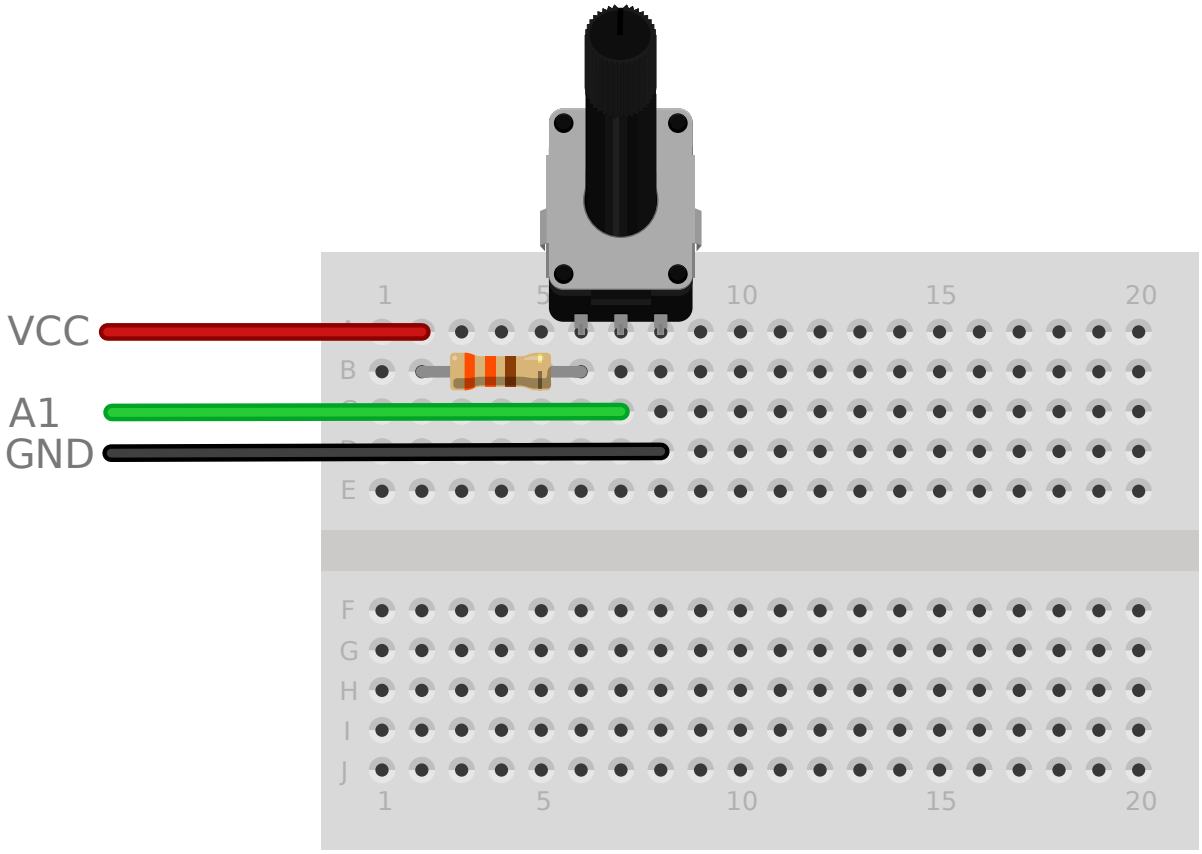


Connect the ADC supply to one leg of a resistor, on the other leg connect a jumper wire and one leg of the LDR, connect this jumper wire to `A0` on the ADS1115. Connect the second leg of the LDR to `GND` with a jumper wire.

Run the `watch-adc.sh` script with `4` as the argument. Cover and uncover the LDR with your finger and watch how the voltage changes.

Activty 5c: Potetiometer Voltage Divider

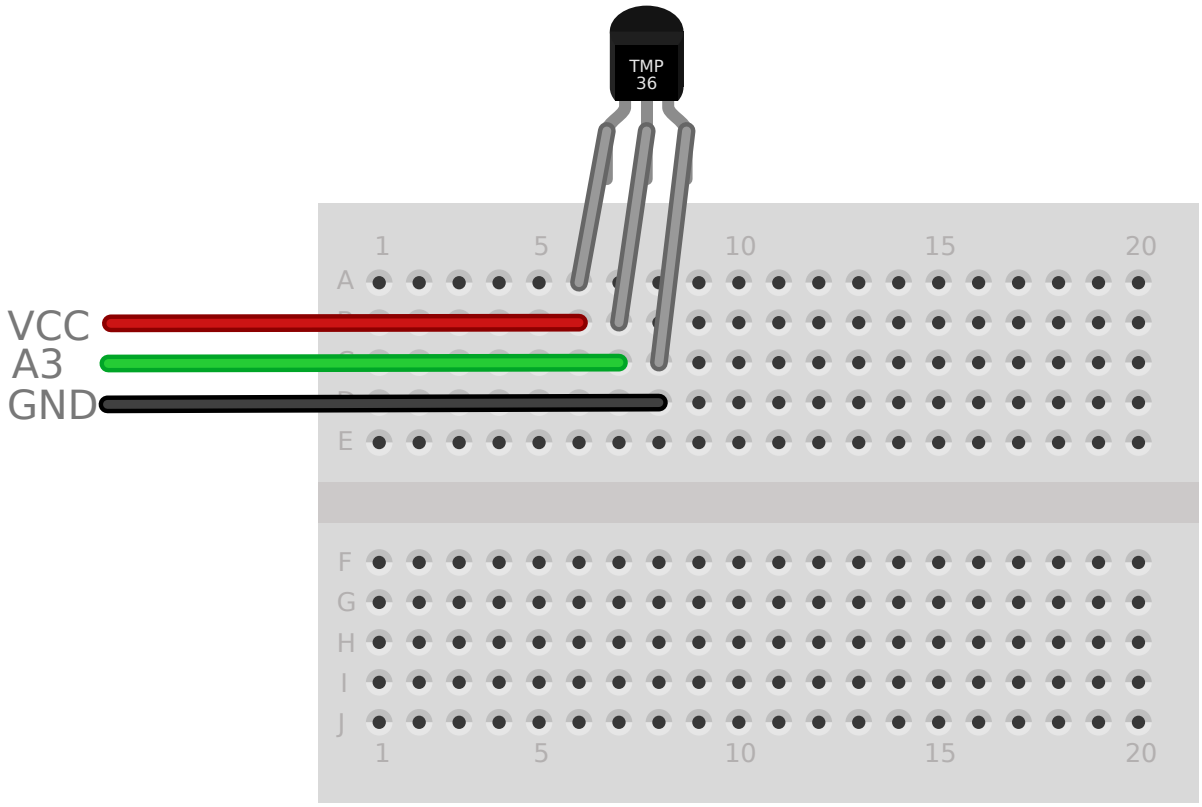
Connect the ADC supply to one leg of a resistor and the other leg to one of the outer pins on the potentiometer. Connect a jumper wire from the center leg of the potentiometer to `A1` on the



ADS1115. Connect the final leg of the potentiometer to `GND`.

Run the `watch-adc.sh` script with `5` as the argument. Turn the knob on the potentiometer and watch how the voltage changes.

Activty 5c: TMP36



The TMP36 is a low voltage precision temperature sensor, it requires power, GND and returns an analog voltage proportional to the supply that indicates the temperature.

The pins on the TMP36 are labeled from left to right with the flat side to the front. The left leg is the supply, the middle leg is analog out and the right leg is `GND`. If you connect supply and `GND` swapped the TMP36 will get very hot and eventually stop working.

Connect the left leg to the ADC supply, the center leg to `A3` on the ADS1115 and the right leg to `GND`.

To read from the TMP36 run the `watch-tmp36.sh` script with the argument `6`.

```
# sh ~/scripts/watch-tmp36.sh 6
```