An Introduction to Hardware Hacking with FreeBSD

Controlling the World with FreeBSD

Tom Jones tj@enoti.me

1 / 71

whoami

- Internet Engineer
  - Internet protocols design and IETF Standards
  - Hack in the FreeBSD network stack
  - I try to make the Internet better
- Founder, Director and sometime organiser at local hackerspace
  - I take the hackerspace camping every summer, campgnd.com
- Fascinated by Blinkenlights
- I am helping to run a small tech conference in the North East of Scotland
  - northernrst.com
  - CFP is still open (submit early, submit often!)
- I write about my adventures with FreeBSD and hardware
  - blog: adventurist.me
  - fediverse: tj on altelectron.org.uk

2 / 71

Slides and Handout

These slides are available here today:

https://adventurist.me/tutorial/presentation.html

The handout is available here today:

https://adventurist.me/tutorial/handout.pdf

And in the future here: https://adventurist.me/talks

3 / 71

What are we doing here?

- Use FreeBSD to interact with real things:
  - lights
  - fans
  - motors
  - temperature sensors

Learning Goals

I want you to learn how to control the world from FreeBSD.

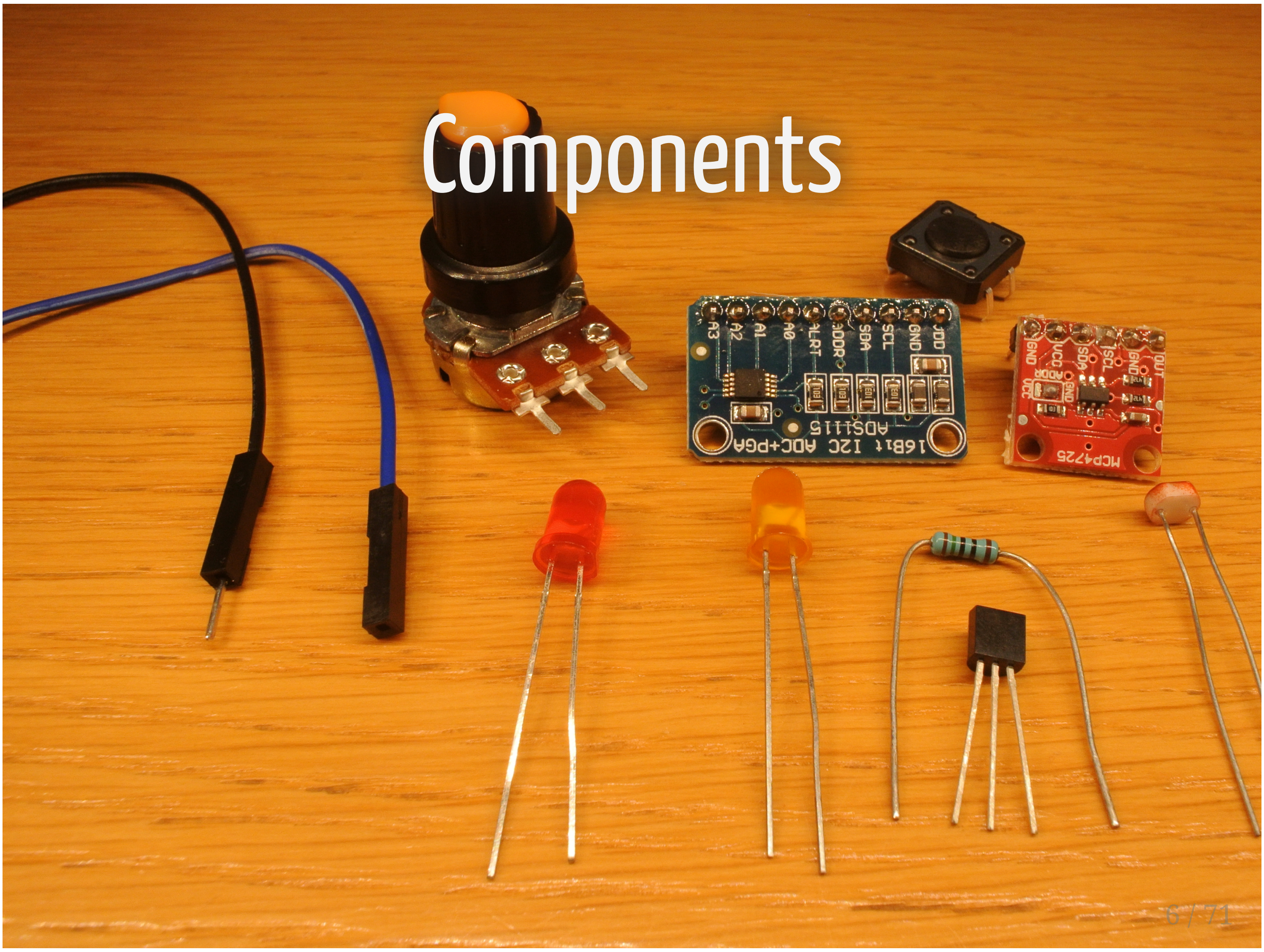
- Hands on
- Self Sufficiency
- Provide advice to help you start **your** projects

Lab Equipmet

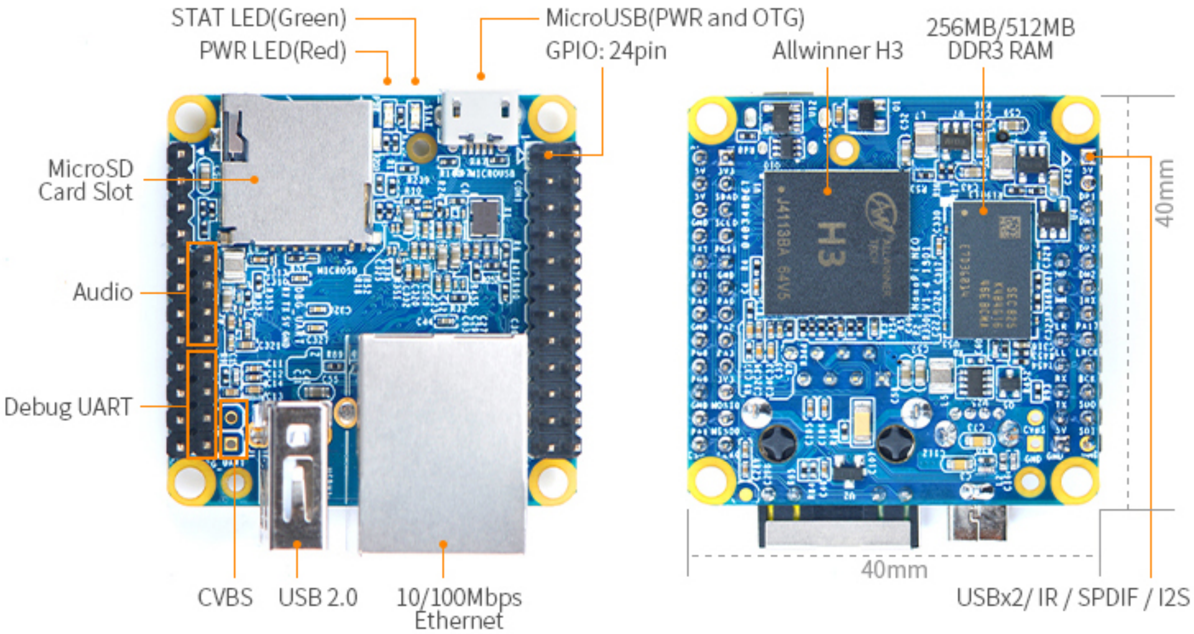
- NanoPi NEO-LTS
- MicroUSB cable
- Pre-prepared MicroSD card
- USB Serial adapter
- Breadboard
- A mess of jumper wires
- A button
- Resistors (330Ω)
- LEDs (red, green, yellow, mystery)
- I2C devices (more later)
  - MPC4725 DAC
  - ADS1115 ADC
- TMP36 Temperature Sensor
- LDR
- 10k Potentiometer

All in a nice wee box.

5 / 71


Components
A3
A2
A1
A0
ALRT
ADDR
SDA
SCL
GND
VDD
ADS1115
16Bit I2C ADC+PGA
GND
VCC
SDA
SCL
GND
OUT
MCP4725
6 / 71

The NanoPi-NEO LTS



NanoPi NEOLTS

- NanoPi NEO-LTS from FriendlyElec
- Allwinner H3
  - Quad-Core ARM Cortex-A7 @ 1.296GHz
  - Mali400 MP2 @ 600Mhz GPU
  - GBit MAC, integrated 10/100M PHY
  - Datasheet
- Available with 256MB or 512MB of RAM (you have the 256MB verion)
- 1 USB Host port, 1 USB OTG port
- 10/100 Ethernet
- Exposed on pin headers
  - 3x UART - Serial Ports
  - 2x USB - USB
  - SPI - Serial Peripheral Interface
  - I2C - Inter-Integrated Circuit
  - I2S - Inter-IC Sound
  - GPIO - General Purpose Input/Output
- FriendlyElec Long Term Support
  - "LTS - Long Term Support, We will not make any changes to this model and will provide support as long as possible"

8 / 71

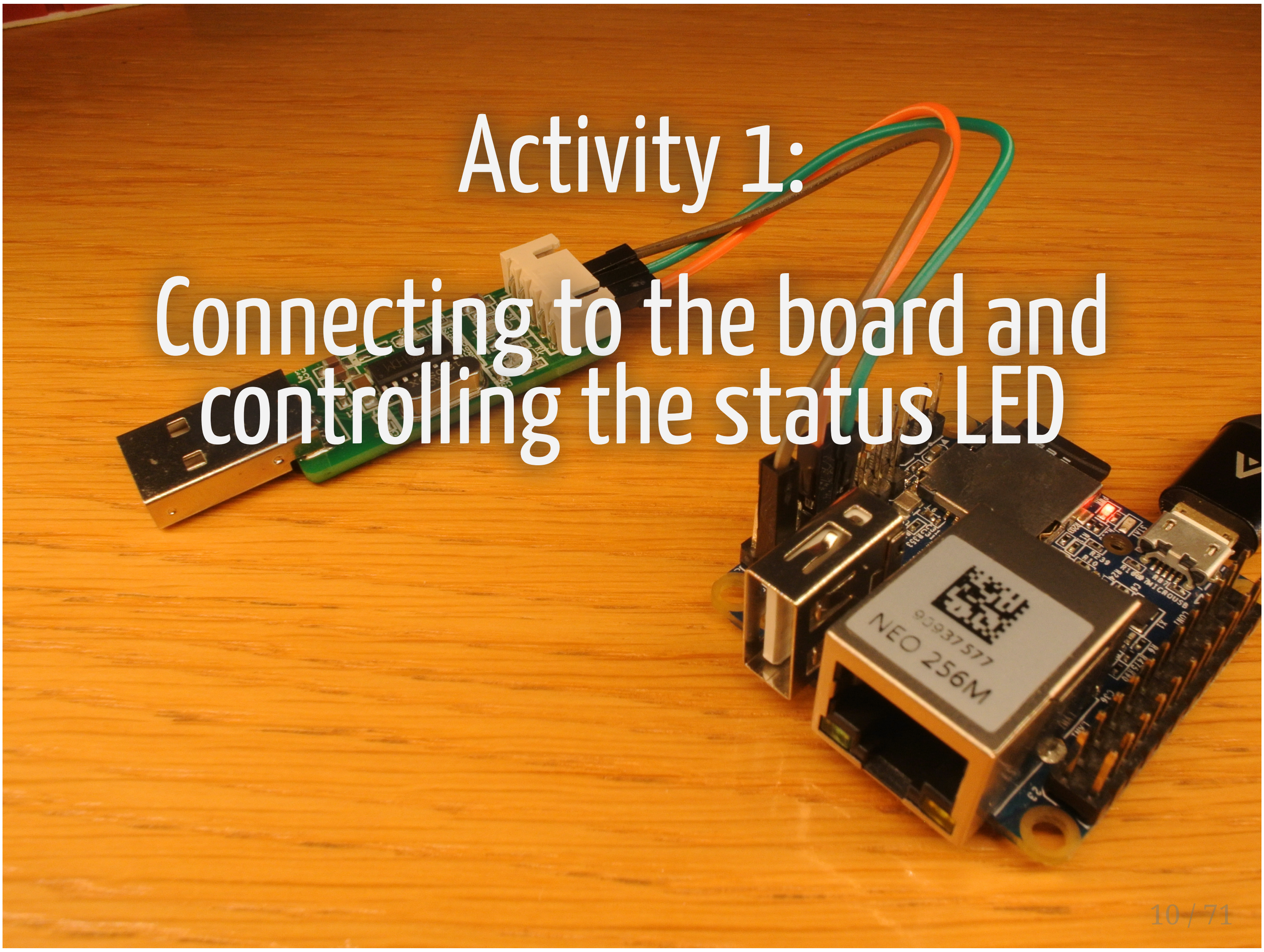
https://wiki.friendlyarm.com/wiki/index.php/NanoPi_NEO

FriendlyElec

https://friendlyarm.com/

- FriendlyElec make a wide range of Single Board Computers (SBC)
- Support FreeBSD developers with hardware
- Really are very friendly

- Thank you FriendlyElec


Activity 1:
Connecting to the board and controlling the status LED
NEO 256M

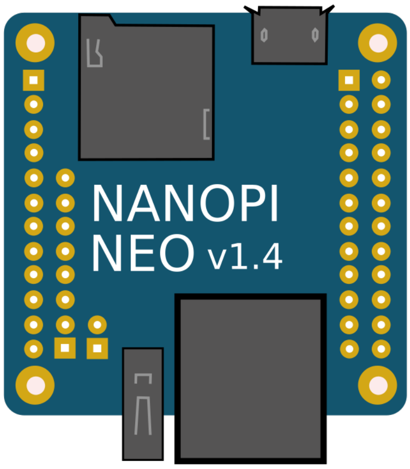
Activity 1:

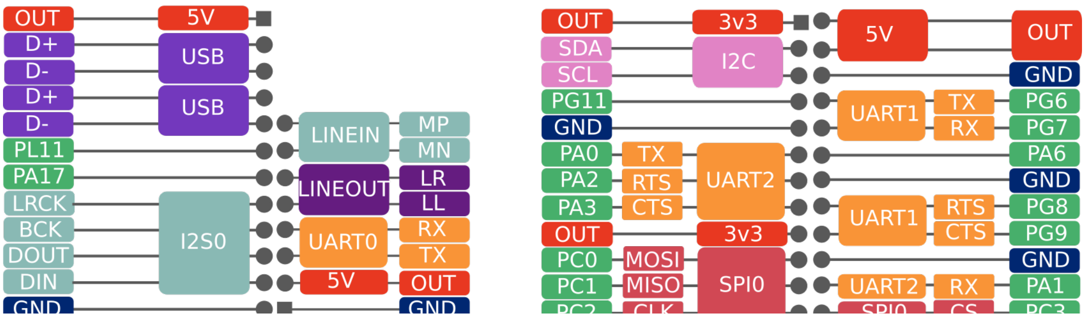
Connecting to the board and controlling the status LED

We are going to:

- Wire the NanoPi to the USB Serial board
- Turn on the NanoPi
- Connect to the NanoPi
- List out the GPIO
- Turn on the Status LED

Activity 1:


NANOPI
NEO v1.4


OUT
5V
D+
D-
USB
D+
D-
USB
PL11
PA17
LRCK
BCK
DOUT
DIN
GND
I2S0
LINEIN
MP
MN
LINEOUT
LR
LL
UART0
RX
TX
5V
OUT
GND
OUT
3v3
5V
OUT
SDA
SCL
I2C
GND
PG11
UART1
TX
PG6
GND
RX
PG7
PA0
TX
UART2
PA6
PA2
RTS
GND
PA3
CTS
UART1
RTS
PG8
CTS
PG9
OUT
3v3
PC0
MOSI
SPI0
GND
PC1
MISO
UART2
RX
PA1

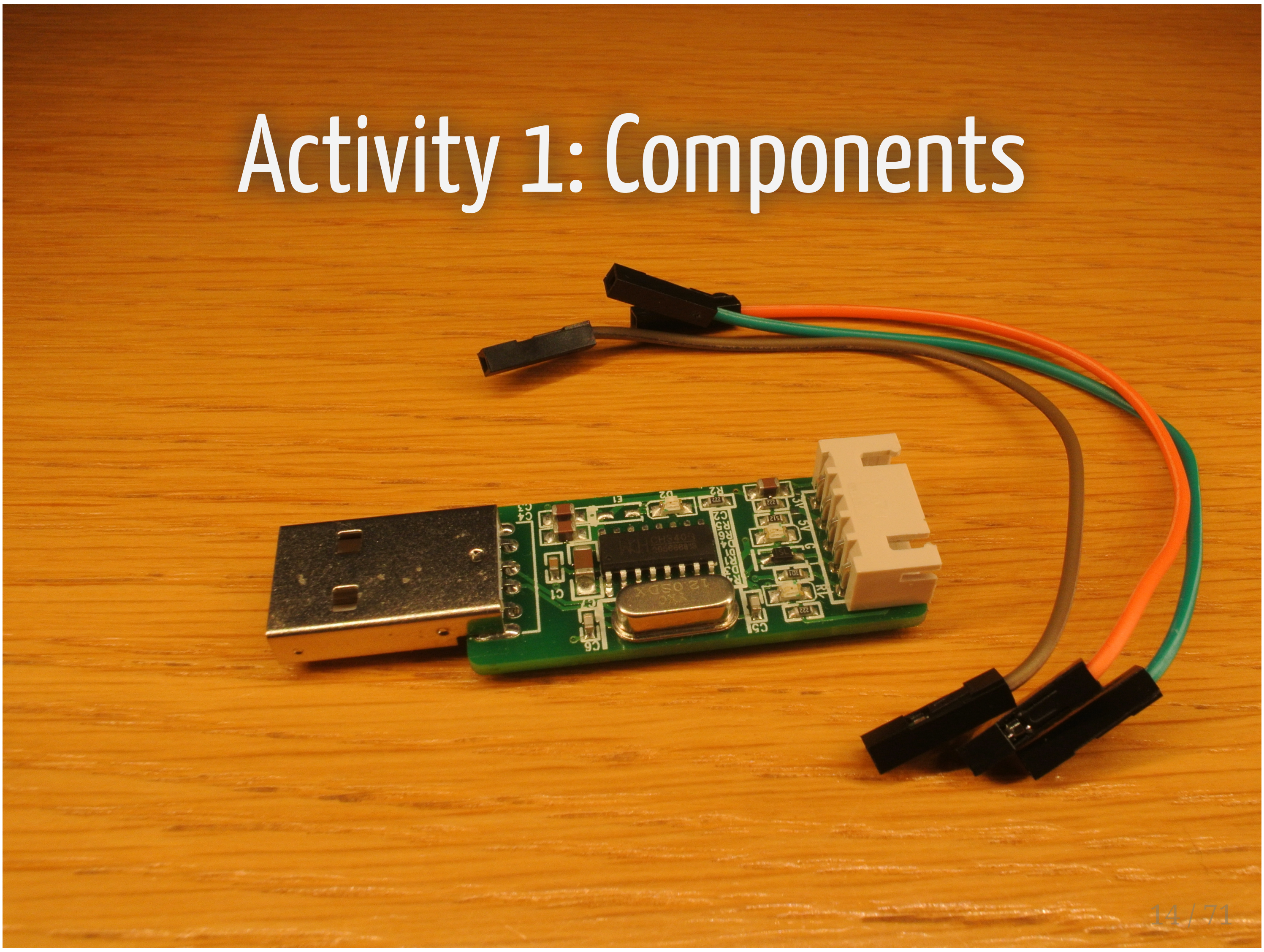
Activity 1a:

Connecting to the board

- We are going to use a CP2102 USB Serial Adapter board
- It connects on USB and as presents a serial device
- If you are not on FreeBSD or Linux, you will need drivers (I am sorry)
  - If you didn't install before, please install now
  - https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

13 / 71

Connecting to the board can be the hardest part of any day


Activity 1: Components
14 / 71

Activity 1a:

Connecting to the board

Finding the serial port

- dmesg
- FreeBSD USB serial devices connect as /dev/ttyUX
- Linux USB serial devices connect as /dev/ttyUSBX
- Mac or Windows event log
- On Windows you have to look in device manager
- I find looping an ls is handy on Mac OS:

```
$ while true; do; ls /dev/tty.*; sleep 0.5; done
/dev/tty.Bluetooth-Incoming-Port
/dev/tty.Bluetooth-Incoming-Port
/dev/tty.Bluetooth-Incoming-Port /dev/tty.usbserial-1420
```

Activity 1a:

Connecting to the board

```
user@laptop $ sudo cu -l /dev/tty.usbserial-1420 -s 115200
Connected.
```

Activity 1a:

Connecting to the board

```
$ sudo cu -l /dev/tty.usbserial-1420 -s 115200
Connected.

U-Boot SPL 2019.04 (Jul 11 2019 - 14:13:51 +0000)
DRAM: 256 MiB
Trying to boot from MMC1


U-Boot 2019.04 (Jul 11 2019 - 14:13:51 +0000) Allwinner Technology

CPU:   Allwinner H3 (SUN8I 1680)
Model: FriendlyARM NanoPi NEO
DRAM:  256 MiB
MMC:   mmc@1c0f000: 0
Loading Environment from FAT... *** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   phy interface0

Error: ethernet@1c30000 address not set.
eth-1: ethernet@1c30000
```

Activity 1a:

eventually

```
Creating and/or trimming log files.
Starting syslogd.
Clearing /tmp (X related).
Updating motd:.
Mounting late filesystems:.
Performing sanity check on sshd configuration.
Starting sshd.
Starting cron.
Starting background file system checks in 60 seconds.

Fri Jan  1 00:00:47 UTC 2010

FreeBSD/arm (generic) (ttyu0)

login:
```

default FreeBSD creds are: **root:root** and freebsd:freebsd

Activity 1b:

Controlling an LED

First check if we actually have a GPIO Controller:

```
root@generic:~ # dmesg | grep gpioc
gpioc0: <GPIO controller> on gpio0
gpioc1: <GPIO controller> on gpio1
```

```
root@generic:~ # gpioctl -f/dev/gpioc0 -l | head
pin 00: 0       PA0<>
pin 01: 0       PA1<>
pin 02: 0       PA2<>
pin 03: 0       PA3<>
pin 04: 0       PA4<>
pin 05: 0       PA5<PU>
pin 06: 0       PA6<>
pin 07: 0       PA7<>
pin 08: 0       PA8<>
pin 09: 0       PA9<>
```

Activity 1b:

Controlling the Status LED

- NanoPi NEO-LTS has a green Status LED next to the red power LED
- Status LED is connected to `PA10`.

  root@generic:~ # gpioctl -f /dev/gpioc0 -l | grep PA10 pin 10: 0 PA10

Turn the LED on:

```
root@generic:~ # gpioctl -f /dev/gpioc0 10 1
```

`gpioctl` uses gpioc0 as the default, so we could also have done:

```
root@generic:~ # gpioctl 10 1
```

```
root@generic:~ # sh ./scripts/gpio-blink.sh 0 10
```

Activity 1:

It doesn't work!

Check your wiring

- check that `GND` is connected to `GND`
- check that `TX` is connected to `RX`
- check that `RX` is connected to `TX`

If that still doesn't work, try again

- Rewire everything
  - Seriously it works more often than you would like

If that still doesn't work:

- Ask for help (less applicable after this tutorial)

What are these GPIO things?

- General Purpose Input/Output
- IO on the processor that is left for application specific use or can be reconfigured so
- On NanoPi diagram interfaces like SPI and I2C also have GPIO names

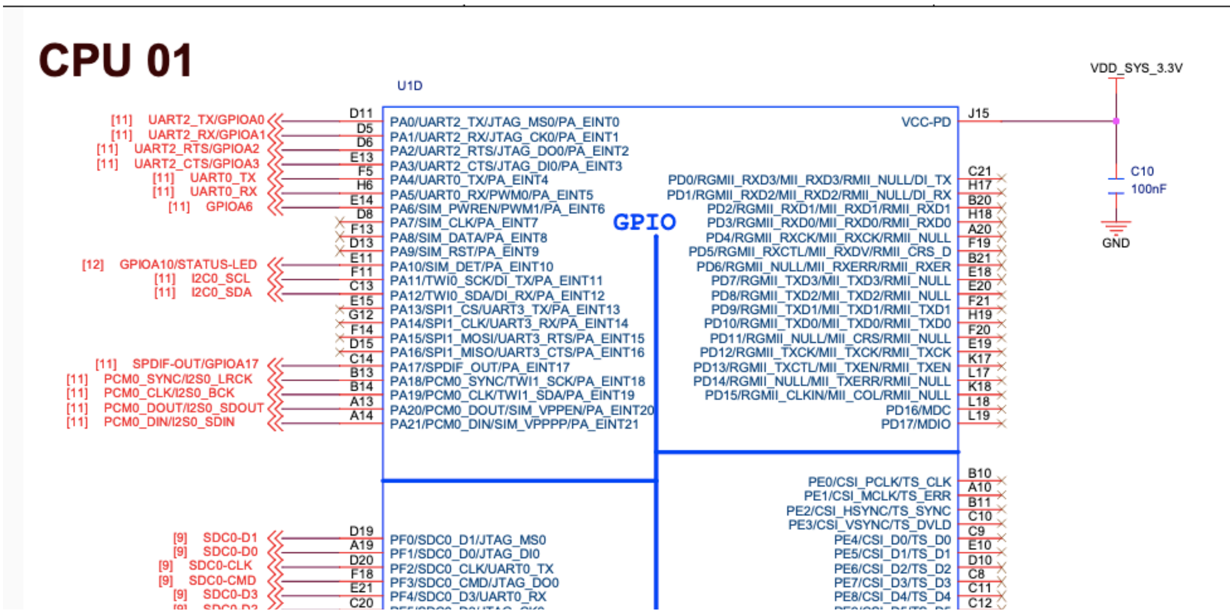
- GPIO are left for **us** to use
- All over the place on all architectures

22 / 71

Tom,

How did you know where the status LED was?

FriendlyElec helpfully host the schematics for their boards on their wiki



23 / 71

What does this output mean

```
root@generic:~ # gpioctl -f/dev/gpioc0 -l
pin 79: 0       PF6<IN,PU>
pin 80: 0       PG0<>
pin 81: 0       PG1<>
pin 82: 0       PG2<>
pin 83: 0       PG3<>
pin 84: 0       PG4<>
pin 85: 0       PG5<>
pin 86: 0       PG6<>
pin 87: 0       PG7<>
pin 88: 0       PG8<>
pin 89: 0       PG9<>
pin 90: 0       PG10<>
pin 91: 1       PG11<OUT>
pin 92: 0       PG12<>
pin 93: 0       PG13<>
```

What does this output mean

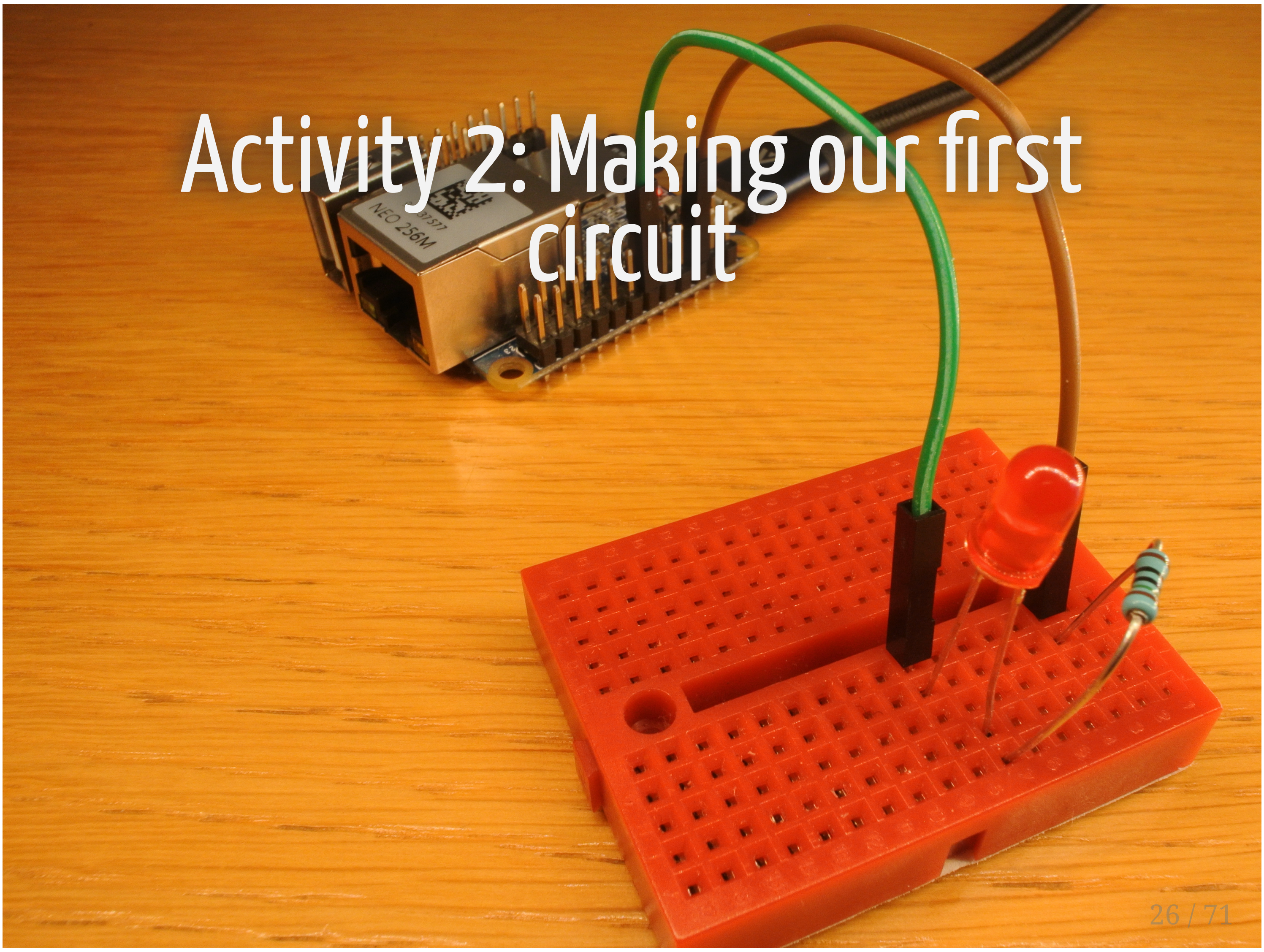
```
pin 79: 0       PF6<IN,PU>
pin 80: 0       PG0<>
pin 91: 1       PG11<OUT>
```

`gpioctl` prints

```
- pin number on the GPIO controller
- current value
- configuration flags
```

```
IN          Input pin
OUT         Output pin
OD          Open drain pin
PP          Push pull pin
TS          Tristate pin
PU          Pull-up pin
PD          Pull-down pin
II          Inverted input pin
IO          Inverted output pin
```

25 / 71


Activity 2: Making our first circuit
NEO 256M
26 / 71

Activity 2:

Making our first circuit

We are going to:

- Wire up a LED and resistor on breadboard
- Reconfigure a GPIO as an output
- Control the LED

Activity 2:

How to break things

- Draw too much current
- Expose too high a voltage

10 ways to kill an arduino

Please do none of these things today

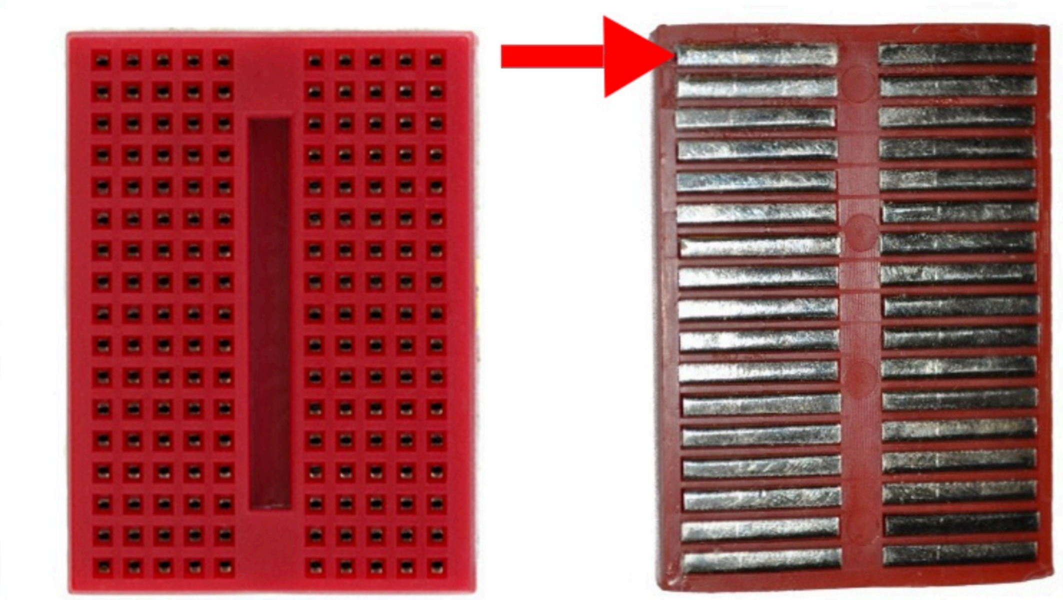
- Short VCC to Ground
  - My NanoPi just turned off when I did it
  - **DO NOT TRY**
  - If you must try this wait until the end of the tutorial
- Connect 5v to a Pin without a resistor
- Attempt to drive an LED without a resistor

- If you kill **your** board I will tell you off and try to get you another
  - if you kill **someone else's** board they will probably be quite cross

28 / 71

Activity 2:

Breadboards

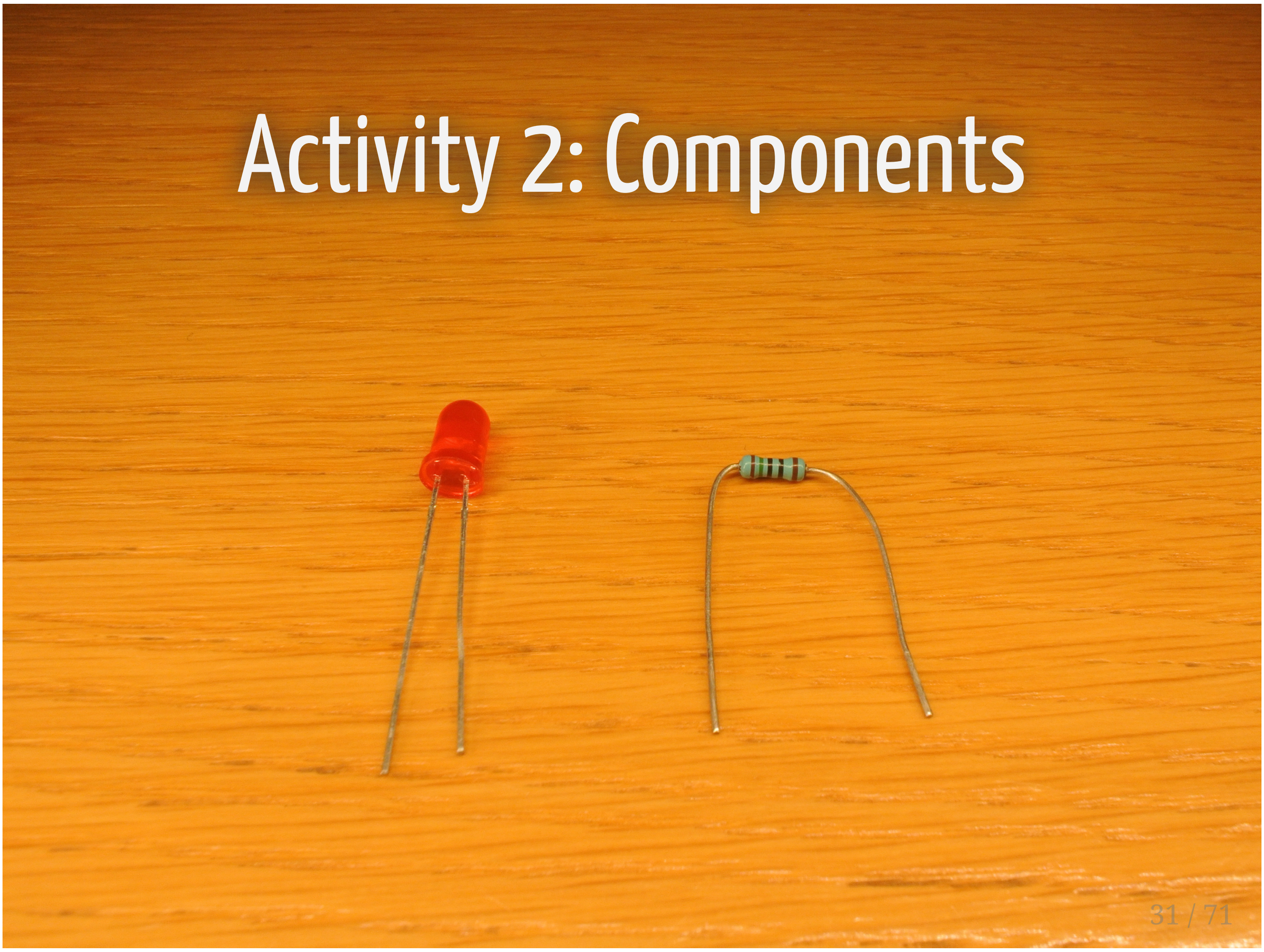


29 / 71

Activity 2:

Jumper Wires

- Also known as Dupont connectors
- 3 types
    - male to male
    - male to female
    - female to female
- Collective noun 'mess'


Activity 2: Components
31 / 71

Activity 2:

Components

Too much current kills things

We choose the resistor based on Ohms Law:

```
V = IR    (voltage, current (I), resistance)
```

```
where V = 3.3 and I = 10mA (milli Amps)
```

H3 SOC datasheet says we can draw 40mA in total, we'll only allow 10mA for this LED. This is still plenty power to illuminate the bulb.

32 / 71

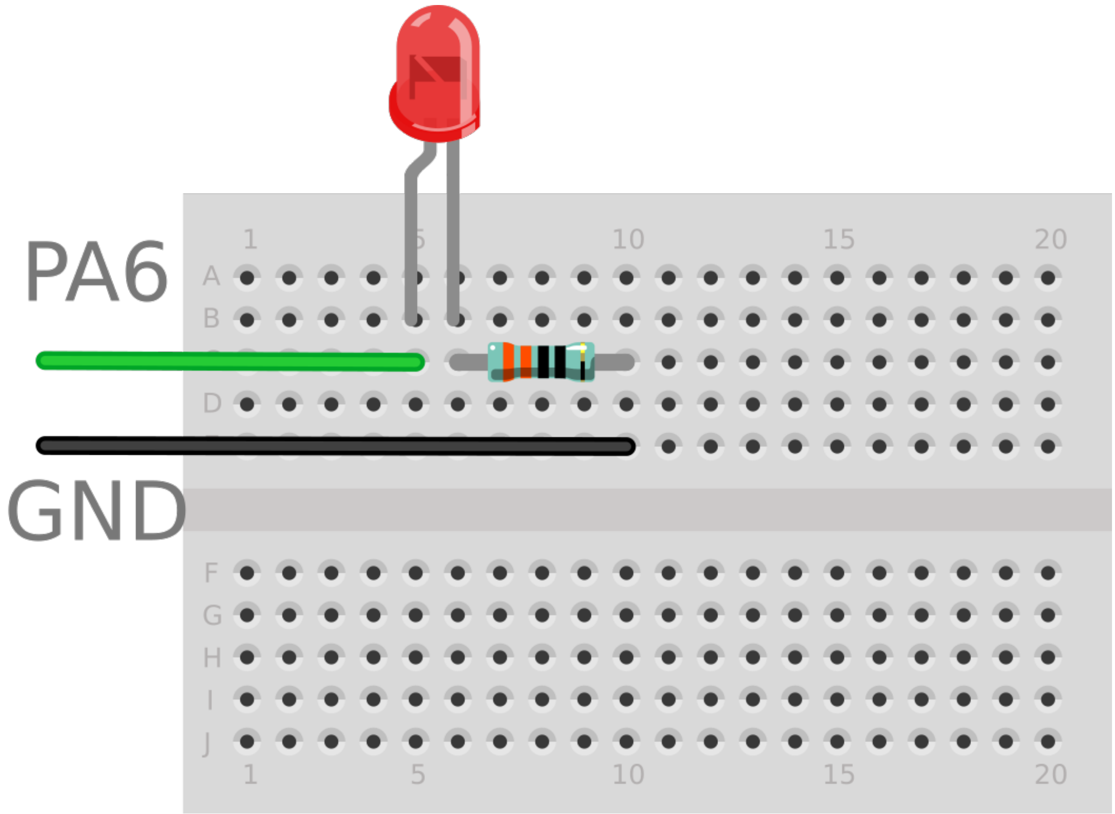
H3 Max ratings

Table 9-1. Absolute Maximum Ratings

Symbol	Parameter	MIN	Max	Unit
$I_{I/O}$	In/Out current for input and output	-40	40	mA
AVCC	Power Supply for Analog part	-0.3	3.4	V
EPHY_VCC	Power Supply for EPHY	-0.3	3.8	V
EPHY_VDD	Power Supply for EPHY	-0.3	1.4	V
HVCC	Power Supply for HDMI	-0.3	3.6	V
V33_TV	Power Supply for TV	-0.3	3.6	V
VCC_IO	Power Supply for Port A	-0.3	3.6	V
VCC_PD	Power Supply for Port D	-0.3	3.6	V
VCC_PG	Power Supply for Port G	-0.3	3.6	V
VCC_PLL	Power Supply for system PLL	-0.3	3.6	V
VCC_RTC	Power Supply for RTC	-0.3	3.6	V
VCC_USB	Power Supply for USB	-0.3	3.6	V
VCC-DRAM	Power Supply for DRAM	-0.3	1.98	V
VDD_CPUS	Power Supply for CPUS	-0.3	1.4	V
VDD_CPUX	Power Supply for CPU	-0.3	1.5	V
VDD_EFUSE	Power Supply for EFUSE	-0.3	3.6	V
VDD_SYS	Power Supply for System	-0.3	1.4	V
T_{STG}	Storage Temperature	-40	125	℃

33 / 71

Activity 2a:



34 / 71

Activity 2b:

Configuring GPIO PA6 as an Output

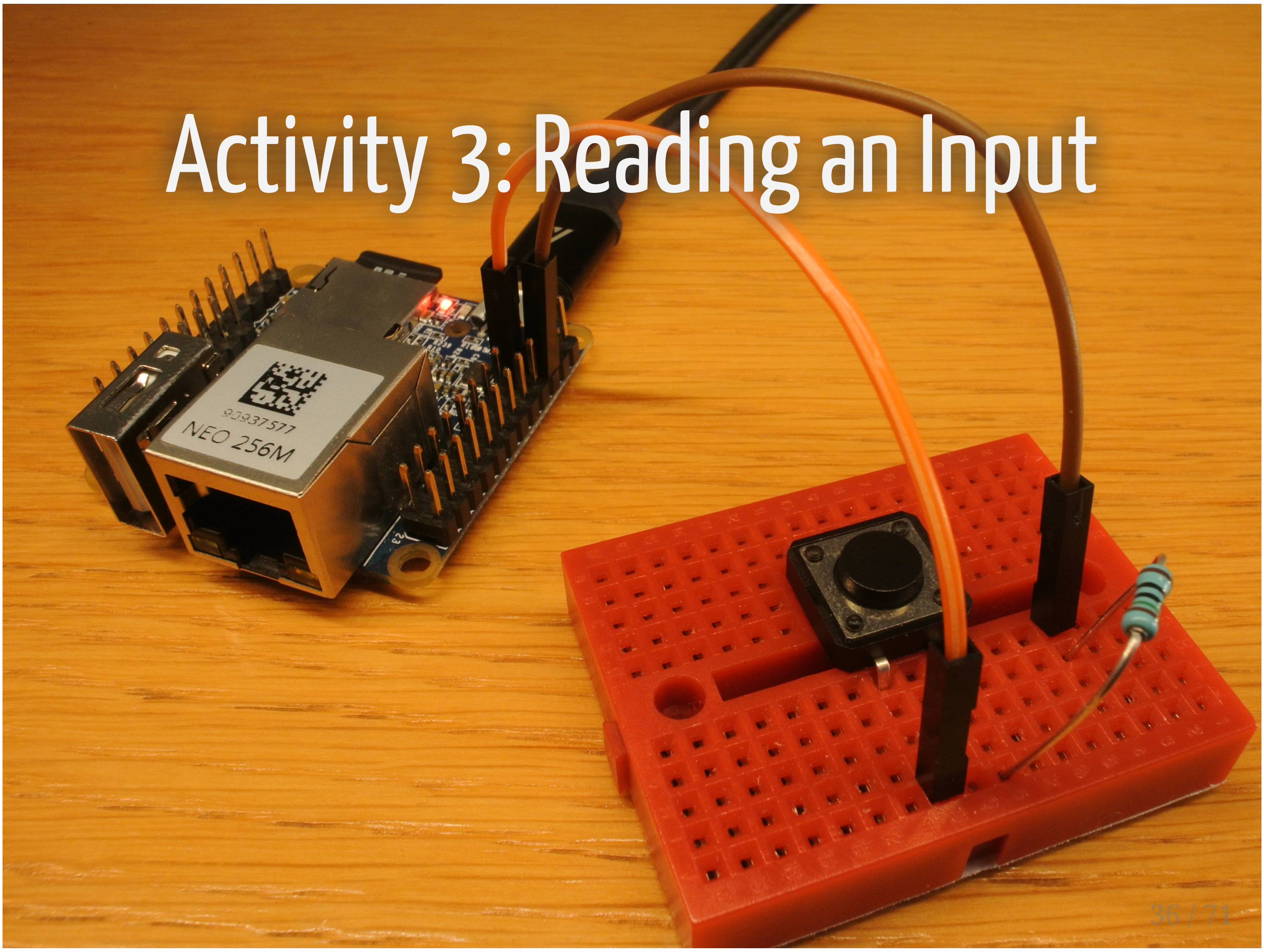
We need to configure the GPIO this time before we can use it, we will use GPIO `PA6`.

```
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>
```

Once configured we can use it:

```
root@generic:~ # gpioctl 6 1
root@generic:~ # gpioctl 6 0
root@generic:~ # gpioctl -t 6
root@generic:~ # gpioctl -t 6
root@generic:~ # sh ./scripts/gpio-blink.sh 0 6
```

35 / 71


Activity 3: Reading an Input
90937577
NEO 256M

Activity 3: Reading an Input

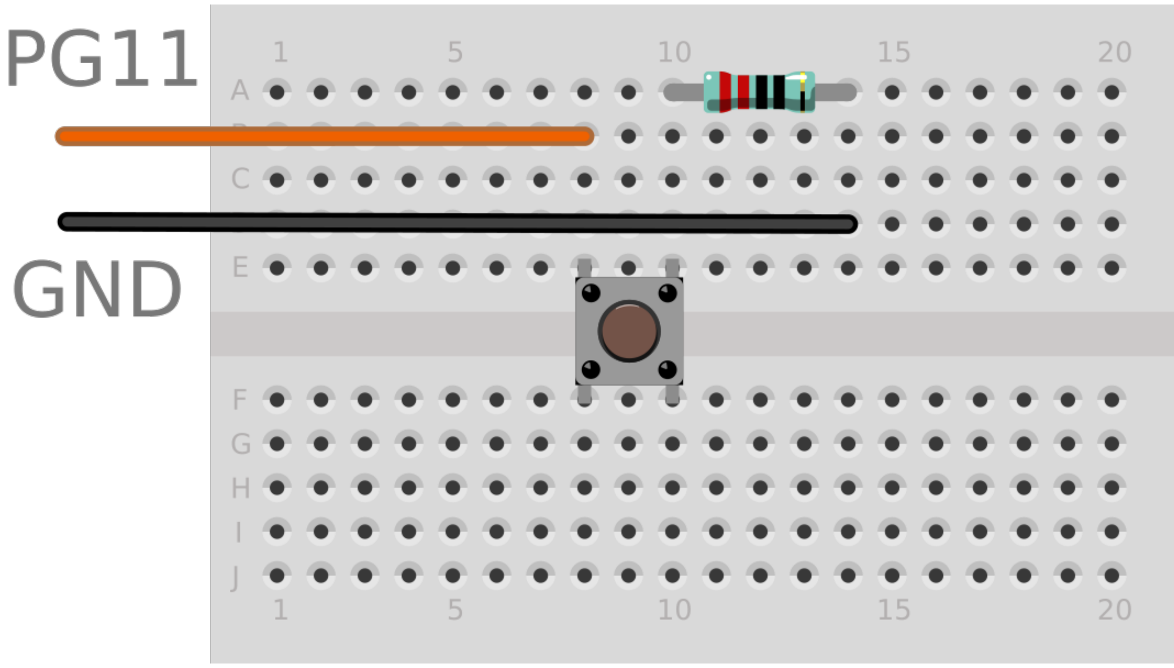
We are going to:

- Wire up a button and a resistor
- Configure a GPIO as an input
- Read the status of the button


Activity 3: Components
38 / 71

Activity 3a:

Wiring the Circuit



39 / 71

Activity 3b:

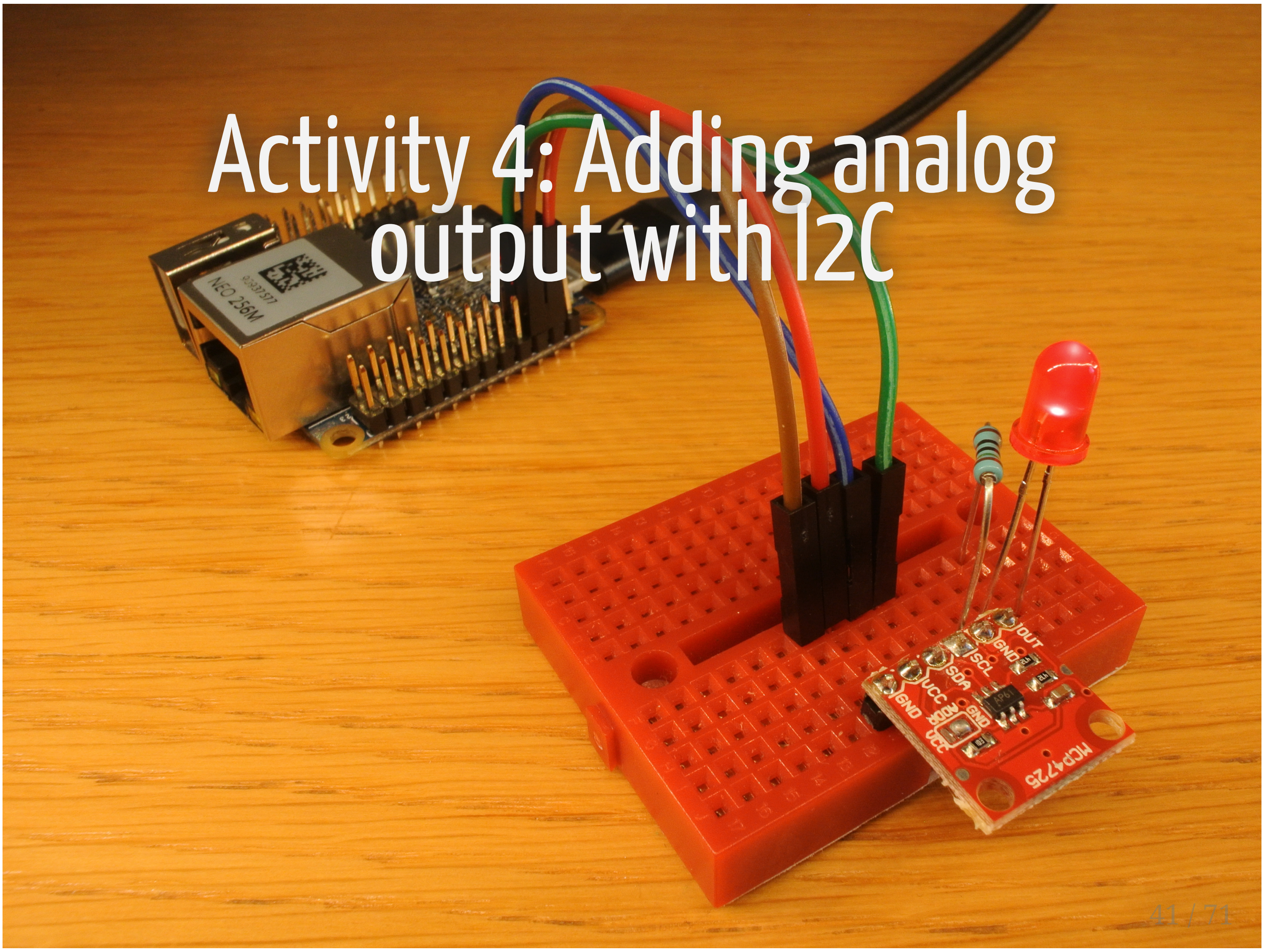
Configure GPIO PG11 as an Input

```
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>
```

Read the GPIO

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

```
root@generic:~ # sh ./scripts/gpio-button.sh
```


Activity 4: Adding analog output with I2C
NEO 256M
GND
VCC
SDA
SCL
GND
OUT
ADDR
MCP4725
41 / 71

What is I2C?

- Inter IC Connect (IIC, I^2C or I2C)
- 2 wire serial protocol for connection low speed devices
- Master slave protocol with multi master capability
- Addressable (and searchable) 7-bit bus
- Used everywhere
  - lots of sensors, interfaces, IO expanders and even screens with I2C support

What is I2C?

- I2C uses clock and a data line
  - commonly labelled SDA (serial data) and SCL (serial clock)
  - SDA connects to SDA
  - SCL connects to SCL
- Master drives the clock line
- Master sends data and the slave drives the Data line to respond with data and to ack

43 / 71

A Diversion into Flattened Device Tree

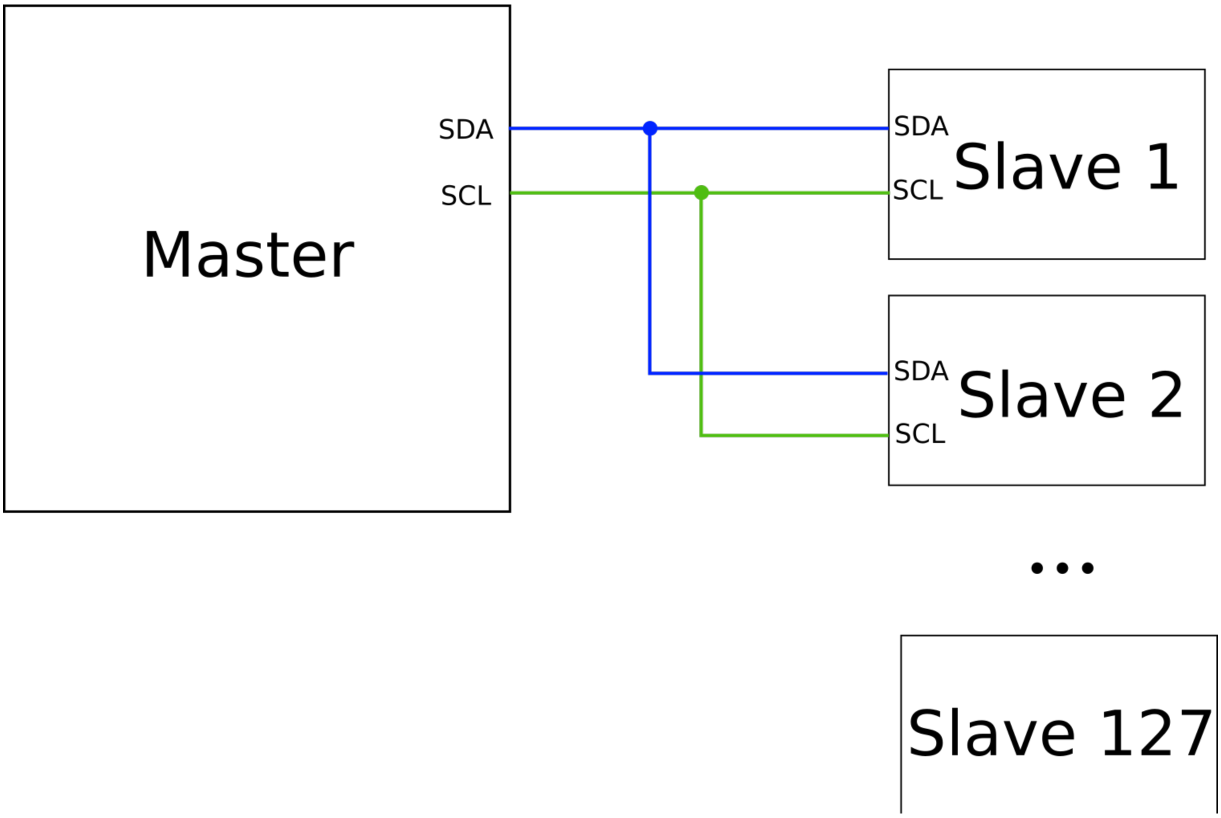
- x86 uses ACPI to discover the devices in the system
- typically ARM platforms use Flattened Device Tree (FDT)
- FDT has a long history and an odd syntax
- FDT can be augmented with overlays
    - add or remove device definitions from the system
- More reading: https://wiki.freebsd.org/FlattenedDeviceTree
- specified in a Device Tree Source (DTS) which is compiled to a Device Tree Blob (DTB)

Activity 4:

The MPC4725 I2C DAC

- MicroChip MPC4725
  - Datasheet
- Digital Analog Converter
- Connected over I2C
- Converts a digital input, over I2C to an analog voltage
- 12 bits of resolution
- 4096 brightness levels for our LED
- has on board memory to maintain value between power cycles

Activity 4: I2C Bus



46 / 71

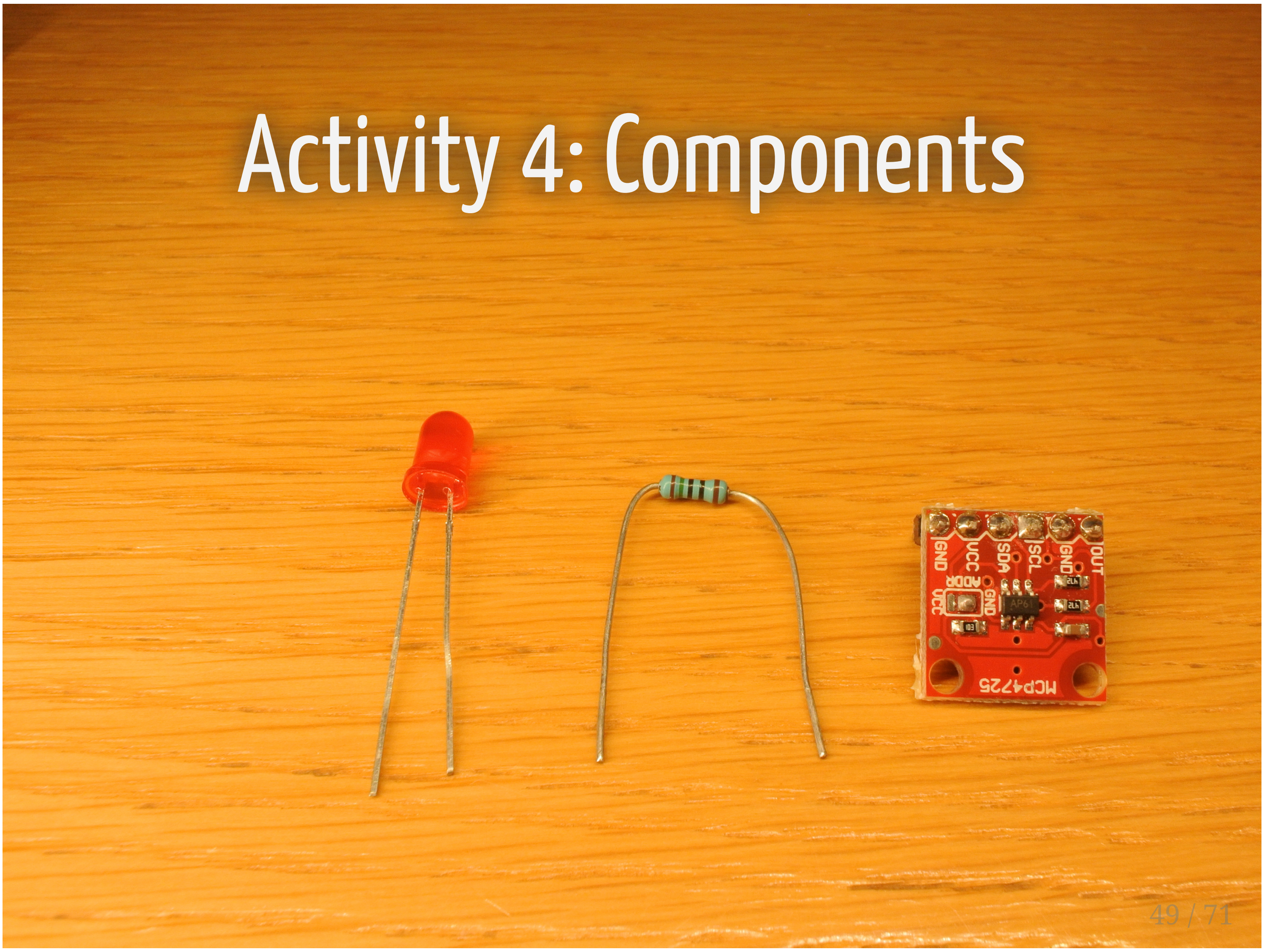
Activity 4:

A dive into the datasheet

Activity 4:

We are going to:

- Add a FDT overlay to enable the I2C
- Read the datasheet for the MPC4725
- Connect the MPC4725 to
  - the NanoPi
  - and an LED
- Fade the LED with the MPC4725


Activity 4: Components
GND
VCC
SDA
SCL
GND
OUT
ADDR
GND
VCC
MCP4725
49 / 71

Activity 4a:

Enable I2C

There is an overlay to enable I2C on H3 in: /boot/dtb/overlays

```
root@generic:~ # ls /boot/dtb/overlays/
spigen-rpi-b.dtbo       spigen-rpi2.dtbo        sun8i-a83t-sid.dtbo
sun8i-h3-i2c0.dtbo      sun8i-h3-sid.dtbo
```

The overlay can be enabled in loader

This is documented in loader.conf

Add to /boot/loader.conf:

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

Activity 4a:

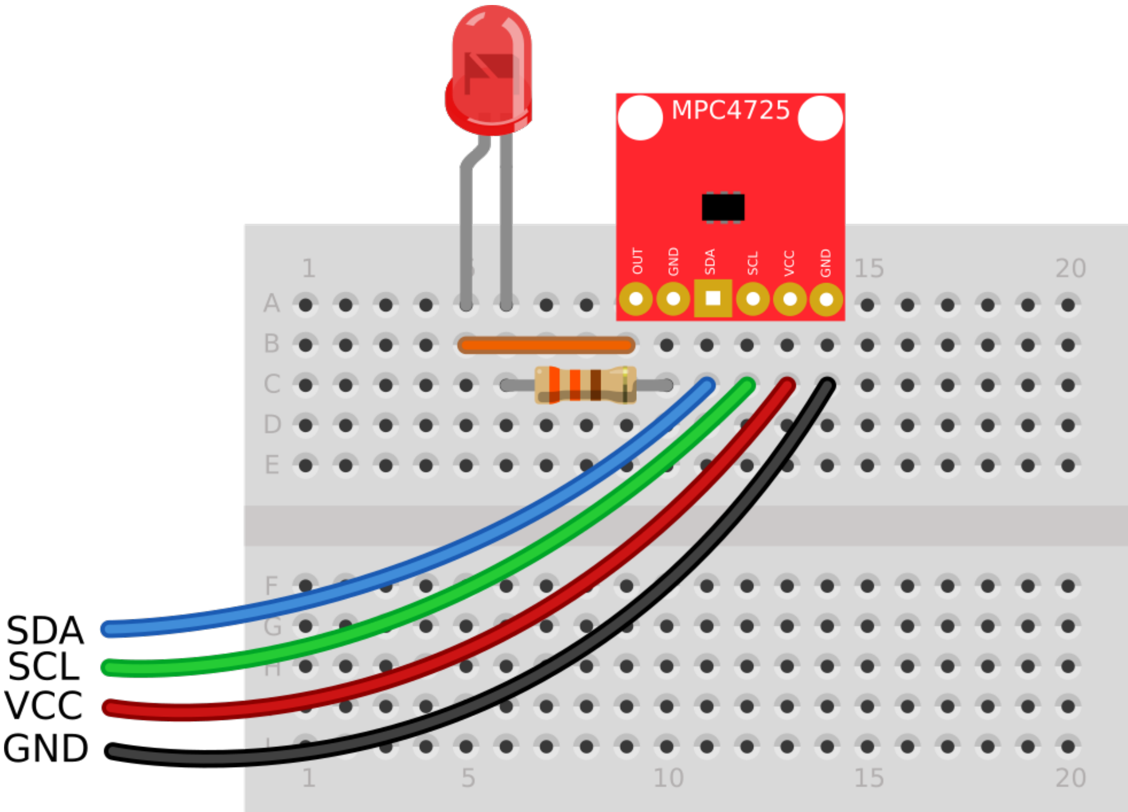
Enable I2C

```
root@generic:~ # ls /dev/iic*
ls: No match.

root@generic:~ # reboot

root@generic:~ # ls /dev/iic*
/dev/iic0
```

Activity 4b: Wiring the Circuit



52 / 71

Activity 4b:

Using the I2C Bus

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

Activity 4b:

Speaking to the MPC4725

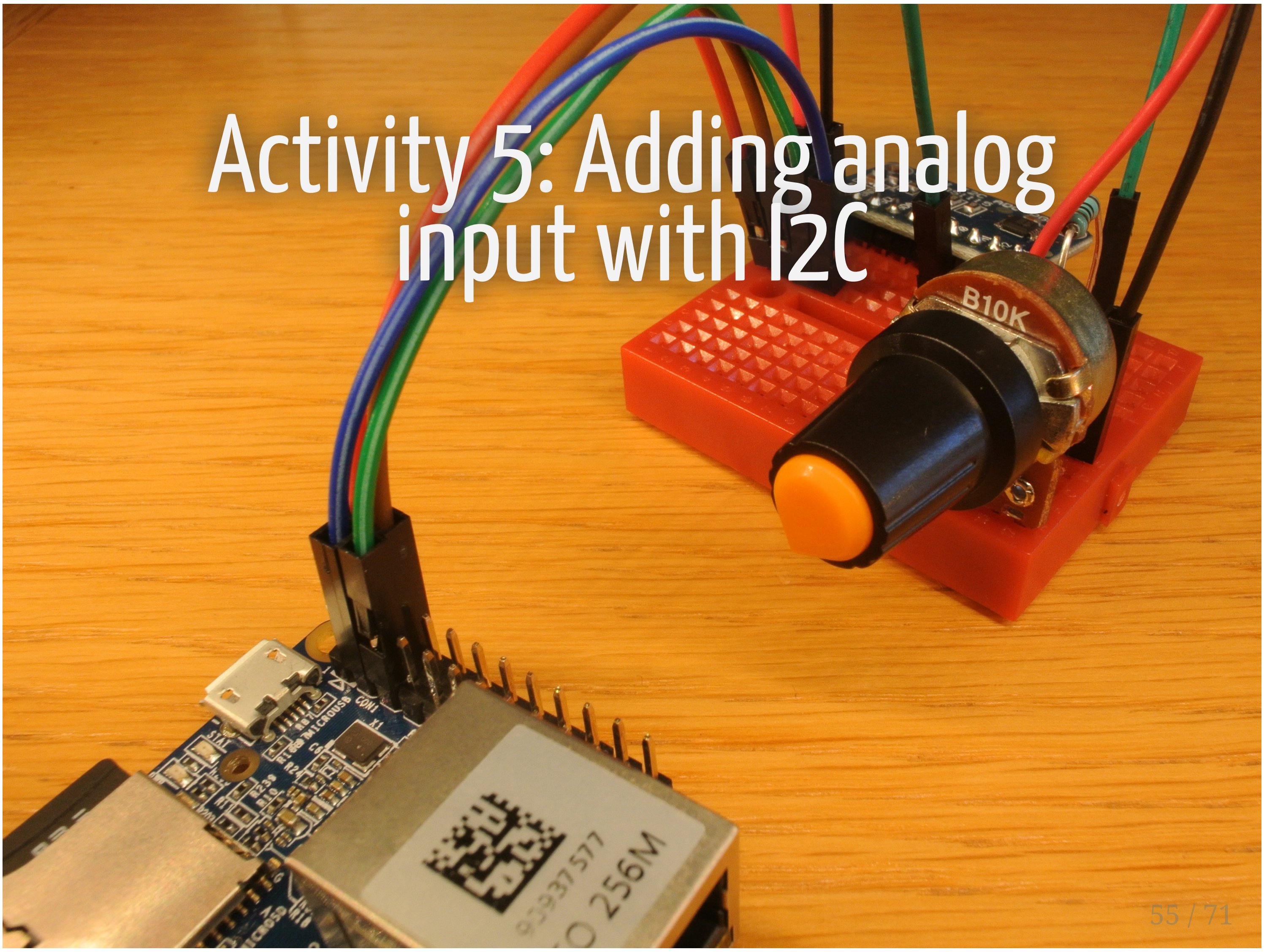
```
root@generic:~ # printf "\17\377" | i2c -a 0x62 -d w -c 2 -m tr
root@generic:~ # printf "\0\0" | i2c -a 0x62 -d w -c 2 -m tr

# printf "%o\n" 0x0f
17
```

There is a nice fading led program in programs/mpc4725-fade.c

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

54 / 71

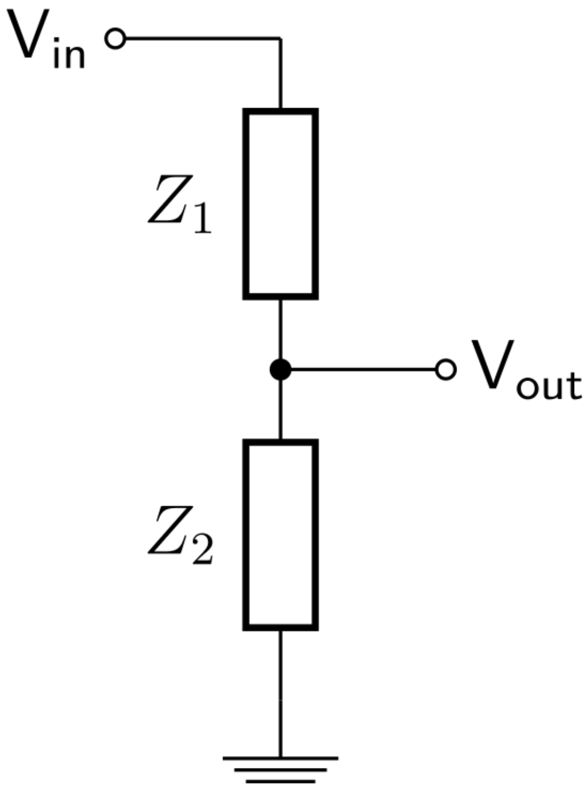

Activity 5: Adding analog input with I2C
B10K
256M
55 / 71

Activity 5:

The ADS1115 I2C ADC

- Texas Instruments ADS1115
    - Datasheet
- Analog Digital Converter
- Connected over I2C
- Converts an analog input, over I2C to digital reading
- 16 bits of resolution
- 4 Channels

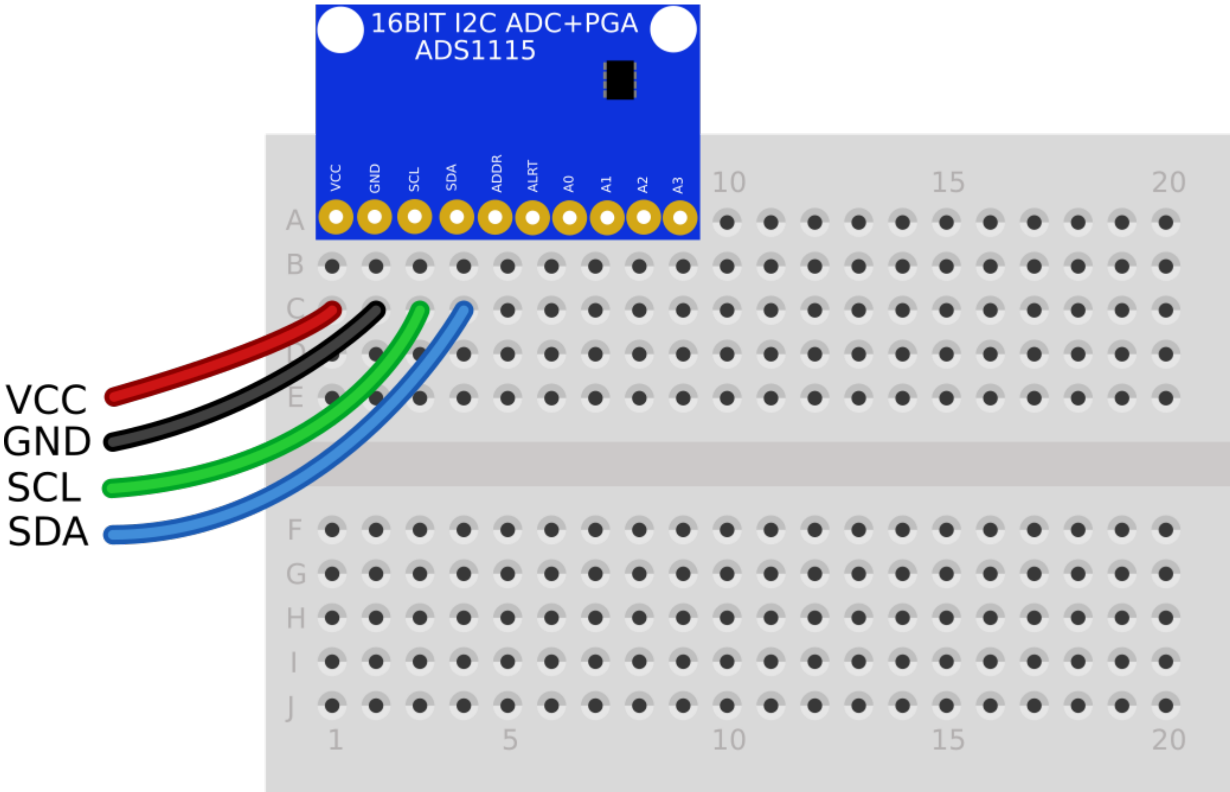
Voltage Dividers



Voltage Dividers

$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$

Activity 5: Analog Input



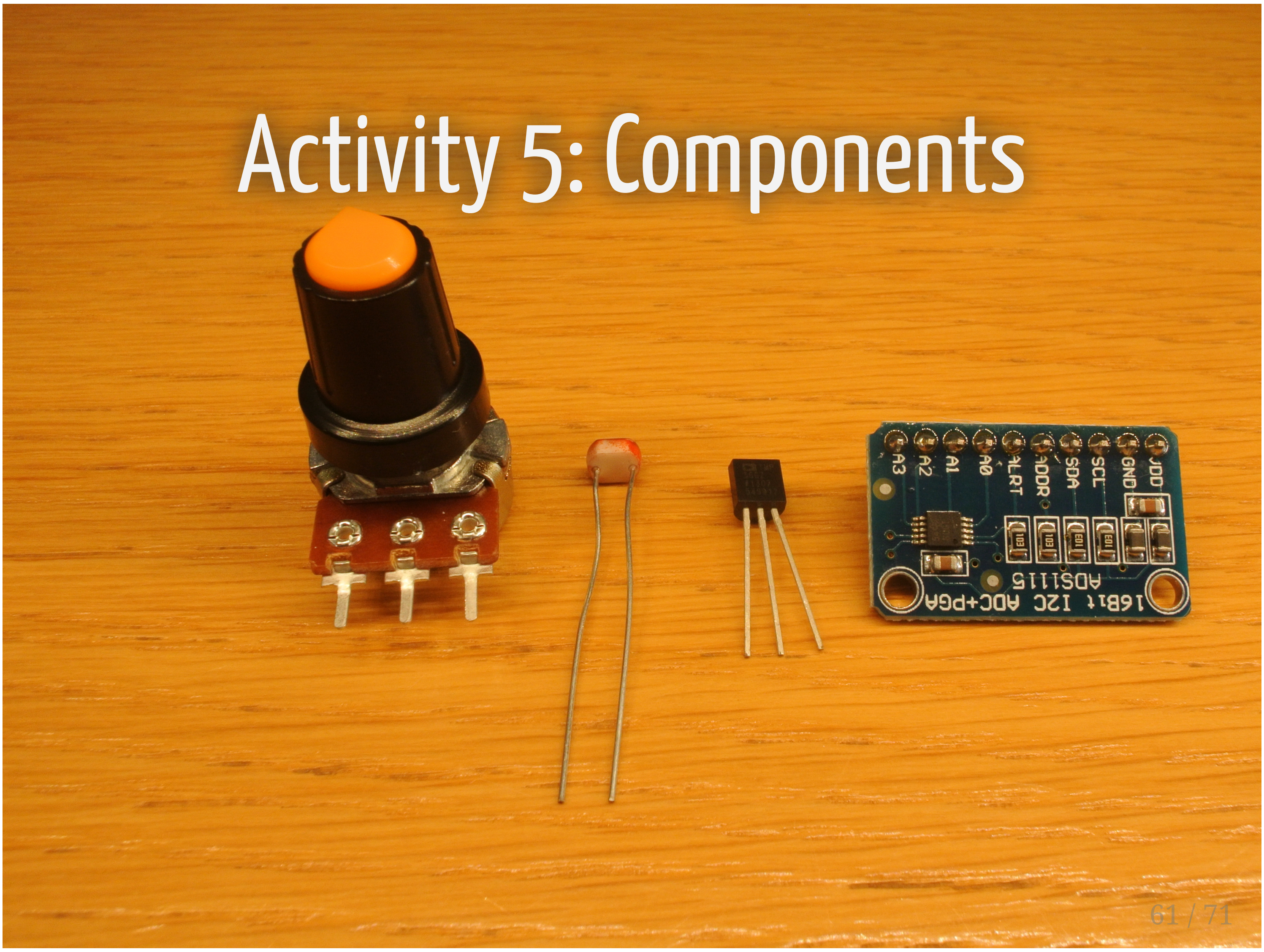
59 / 71

Activity 5:

We are going to:

- Connect the ADS1115
- Add a FDT overlay to add the ADS1115 as a child to the I2C bus
- Read from the ADC using sysctl
- Connect:
  - LDR Voltage Divider
  - Potentiometer Voltage Divider
  - TMP36 Temperature Sensor

60 / 71


Activity 5: Components
A3 A2 A1 A0 ALRT ADDR SDA SCL GND VDD
ADS1115
16Bit I2C ADC+PGA
61 / 71

Activity 5a:

Enabling the ADS1115

There is DTS source file in `~/overlays`, we need to compile it using `dtc` and copy it to the `/boot/overlays` directory:

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

Change the fdt_overlays line in /boot/loader.conf to:

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

reboot the NanoPi

Activity 5a:

Check for the ADS1115

```
# dmesg | grep ADS1115

root@generic:~ # sysctl dev.ads111x

root@generic:~ # sysctl -d dev.ads111x.0.4.voltage
dev.ads111x.0.4.voltage: sampled voltage in microvolts

root@generic:~ # sysctl hw.dev.ads111x.0.4.voltage
```

Activity 5b:

Using the ADS1115

Now we have the ADC we can look at all the values it reports:

```
root@generic:~ # sh -c 'for x in 4 5 6 7; do sysctl dev.ads111x.0.$x.voltage; done;'
```

The driver reports the voltage in microvolts

```
root@generic:~ # sysctl -d dev.ads111x.0.4.voltage
dev.ads111x.0.4.voltage: sampled voltage in microvolts
```

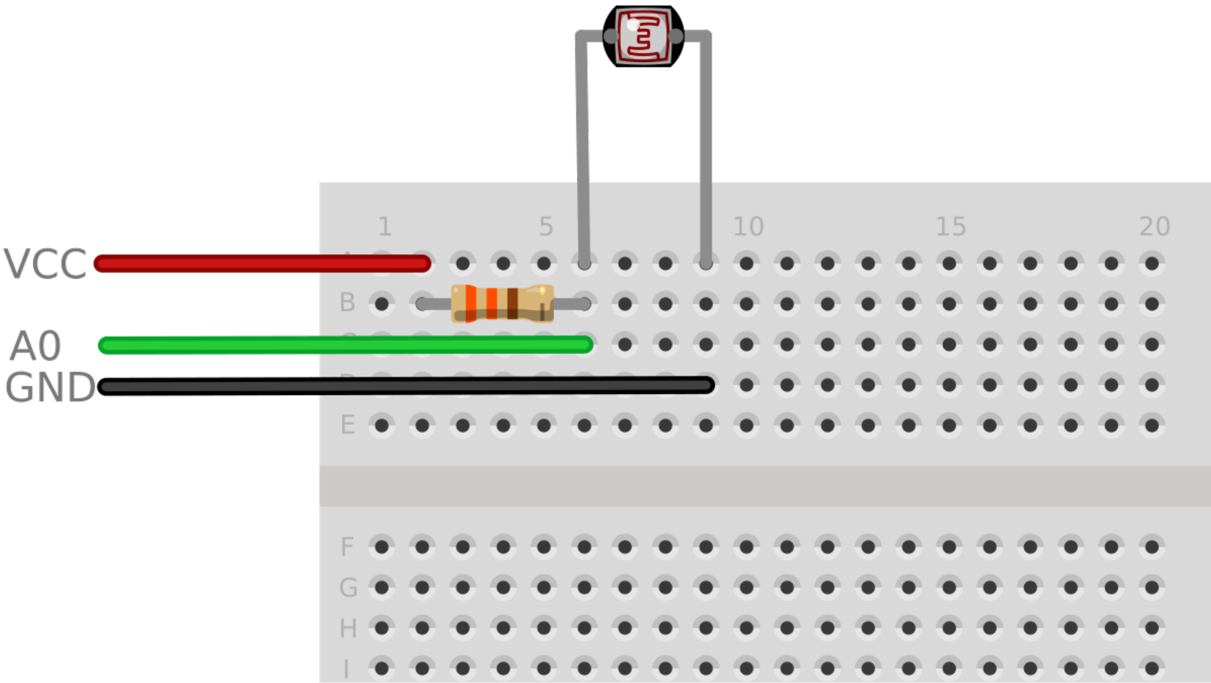
This means we get giant looking numbers for even a small voltage. A microvolt is a 10^-6 volts, to get voltage from the ADS1115 driver we need to divide by 1000000.

I like to use dc for maths on the command line, we can convert the voltage:

```
root@generic:~ # uv=`sysctl -n dev.ads111x.0.$1.voltage`
```

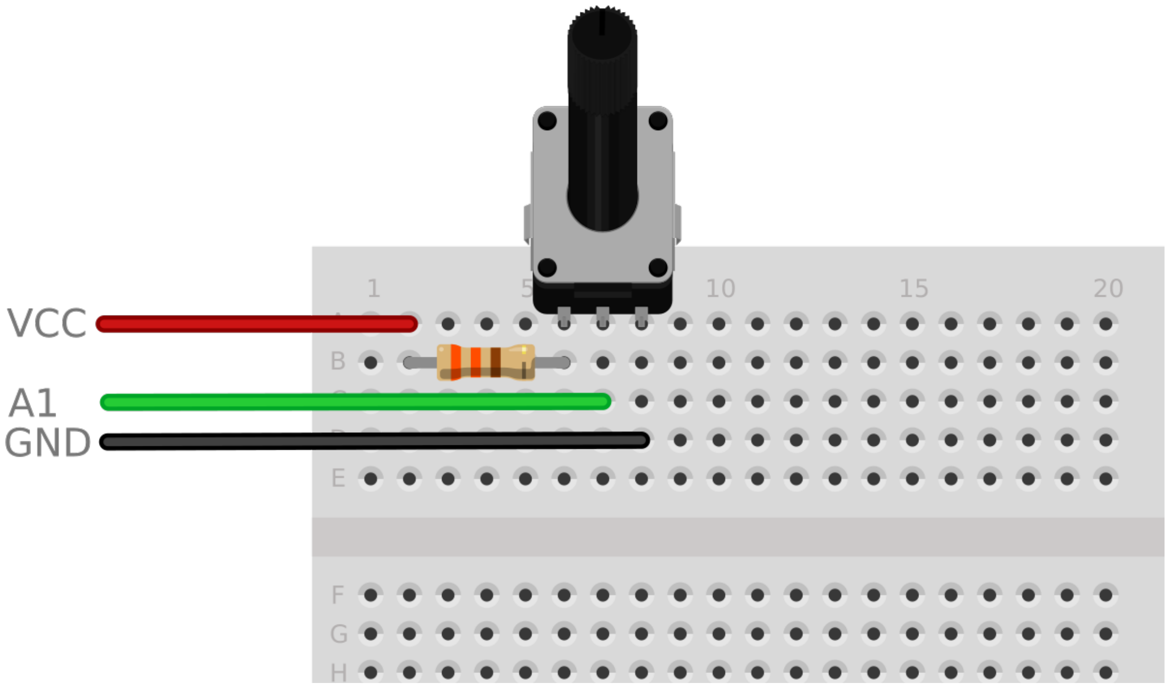
Activity 5c:

LDR Voltage Divider



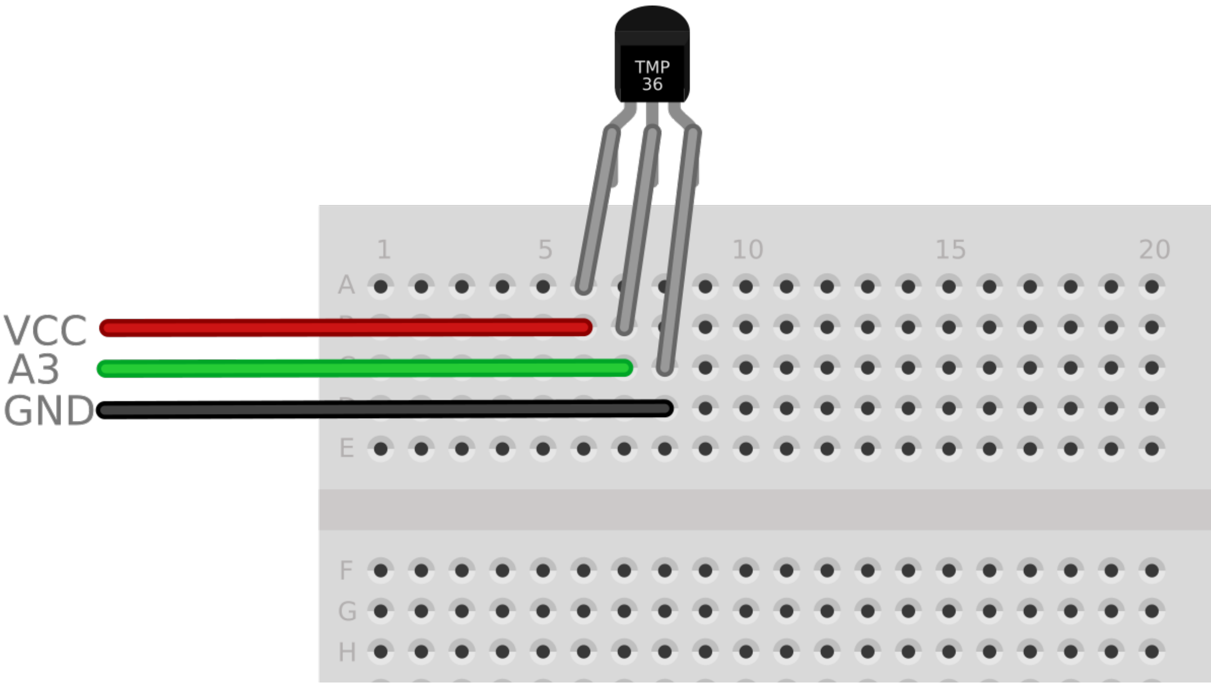
Activity 5c:

Potentiometer Voltage Divider



Activity 5c:

TMP36 Temperature Sensor



67 / 71

Acknowledgements

- FriendlyElec
- manu@
- ganbold@
- ian@
- beta testers

What projects do you want to build?

Extra Slides

Tools I have that you might want

- Multimeter
- Oscilloscope
- Logic Analyser

- Bench power supply

- Making things more permanent

    - Soldering iron
    - Helping hands
    - Solder
    - Protoboard
    - Bits of wire
    - Pin Headers and Pin Sockets

70 / 71

Activity 1a:

Serial data