An Introduction to Hardware Hacking with FreeBSD

Controlling the World with FreeBSD

Tom Jones
tj@enoti.me
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
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
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 Equipment

- 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.
The NanoPi-NEO LTS
NanoPi NEO-LTS

- 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 version)
- 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"

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
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:
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

Connecting to the board can be the hardest part of any day
Activity 1: Components
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:

```bash
$ 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.ubserial-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


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.

```bash
root@generic:~ # gpioctl -f /dev/gpioc0 -l | grep PA10
```

Turn the LED on:
```bash
root@generic:~ # gpioctl -f /dev/gpioc0 10 1
```

`gpioctl` uses gpioc0 as the default, so we could also have done:
```bash
root@generic:~ # gpioctl 10 1
```
```bash
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
Tom,

How did you know where the status LED was?

FriendlyElec helpfully host the schematics for their boards on their wiki
What does this output mean

```
root@generic:~ # gpioctl -f /dev/gpiochip0 -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 number on the GPIO controller
- current value
- configuration flags

<table>
<thead>
<tr>
<th>GPIO Pin</th>
<th>Configuration Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>PF6&lt;IN,PU&gt;</td>
</tr>
<tr>
<td>80</td>
<td>PG0&lt;&gt;</td>
</tr>
<tr>
<td>91</td>
<td>PG11&lt;OUT&gt;</td>
</tr>
</tbody>
</table>

- 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
Activity 2: Making our first circuit
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
Activity 2:

Breadboards
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
Activity 2:

Components

Too much current kills things

We choose the resistor based on Ohms Law:

\[ V = IR \]  
\[ \text{where } V = 3.3 \text{ and } I = 10\text{mA} \]  

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.
# H3 Max ratings

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>MIN</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_{IO}$</td>
<td>In/Out current for input and output</td>
<td>-40</td>
<td>40</td>
<td>mA</td>
</tr>
<tr>
<td>AVCC</td>
<td>Power Supply for Analog port</td>
<td>-0.3</td>
<td>3.4</td>
<td>V</td>
</tr>
<tr>
<td>EPHY_VCC</td>
<td>Power Supply for EPHY</td>
<td>-0.3</td>
<td>3.8</td>
<td>V</td>
</tr>
<tr>
<td>EPHY_VDD</td>
<td>Power Supply for EPHY</td>
<td>-0.3</td>
<td>1.4</td>
<td>V</td>
</tr>
<tr>
<td>HVCC</td>
<td>Power Supply for HDMI</td>
<td>-0.3</td>
<td>3.6</td>
<td>V</td>
</tr>
<tr>
<td>V33_TV</td>
<td>Power Supply for TV</td>
<td>-0.3</td>
<td>3.6</td>
<td>V</td>
</tr>
<tr>
<td>VCC_IO</td>
<td>Power Supply for Port A</td>
<td>-0.3</td>
<td>3.6</td>
<td>V</td>
</tr>
<tr>
<td>VCC_PD</td>
<td>Power Supply for Port D</td>
<td>-0.3</td>
<td>3.6</td>
<td>V</td>
</tr>
<tr>
<td>VCC_PG</td>
<td>Power Supply for Port G</td>
<td>-0.3</td>
<td>3.6</td>
<td>V</td>
</tr>
<tr>
<td>VCC_PLL</td>
<td>Power Supply for system PLL</td>
<td>-0.3</td>
<td>3.6</td>
<td>V</td>
</tr>
<tr>
<td>VCC_RTC</td>
<td>Power Supply for RTC</td>
<td>-0.3</td>
<td>3.6</td>
<td>V</td>
</tr>
<tr>
<td>VCC_USB</td>
<td>Power Supply for USB</td>
<td>-0.3</td>
<td>3.6</td>
<td>V</td>
</tr>
<tr>
<td>VCC_DRAM</td>
<td>Power Supply for DRAM</td>
<td>-0.3</td>
<td>1.98</td>
<td>V</td>
</tr>
<tr>
<td>VDD_CPUS</td>
<td>Power Supply for CPUS</td>
<td>-0.3</td>
<td>1.4</td>
<td>V</td>
</tr>
<tr>
<td>VDD_CPUX</td>
<td>Power Supply for CPU</td>
<td>-0.3</td>
<td>1.5</td>
<td>V</td>
</tr>
<tr>
<td>VDD_EFUSE</td>
<td>Power Supply for EFUSE</td>
<td>-0.3</td>
<td>3.6</td>
<td>V</td>
</tr>
<tr>
<td>VDD_SYS</td>
<td>Power Supply for System</td>
<td>-0.3</td>
<td>1.4</td>
<td>V</td>
</tr>
<tr>
<td>$T_{STD}$</td>
<td>Storage Temperature</td>
<td>-40</td>
<td>125</td>
<td>°C</td>
</tr>
</tbody>
</table>
Activity 2a:
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:~ # sh ./scripts/gpio-blink.sh 0 6
```
Activity 3: Reading an Input
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
Activity 3a:

Wiring the Circuit

PG11

GND

https://adventurist.me/tutorial/presentation.html#1
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
What is I2C?

- Inter IC Connect (IIC, I²C 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
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

Master

Slave 1

Slave 2

...
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
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

[Diagram of a circuit board with labels: SDA, SCL, VCC, GND]
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
Activity 5: Adding analog input with I2C
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

\[ V_{\text{in}} \]

\[ Z_1 \]

\[ V_{\text{out}} \]

\[ Z_2 \]
Voltage Dividers

\[ V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2} \]
Activity 5: Analog Input
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
Activity 5: Components
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:

```bash
# 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:

```bash
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`
root@generic:~ # dc -e "4 k $uv 1000000 / p"
```

[64 / 71]
Activity 5c:

LDR Voltage Divider
Activity 5c:

Potentiometer Voltage Divider
Activity 5c:

TMP36 Temperature Sensor
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
Activity 1a:

Serial data