Labs

- *Usermode drivers*
    - + Toggle - and change the trigger – of the Raspberry Pi builtin onboard LED
    - + GPIO
        - + Toggling an external LED via the Raspberry Pi using a simple LED circuit on a breadboard
    - DH120 temperature and humidity sensor
        - usermode I2C driver
    - SSD1306 (compatible) OLED display
        - usermode I2C driver
- *Kernel-mode drivers*
    - I2C RTC

- USB [?]
- Alberto Liberal book
    - (on Kindle) Linux Driver Development with Raspberry Pi (Practical Labs)
    - code: https://github.com/ALIBERA/linux_raspberrypi_book

---

First, ensure you read the *RPi_Zero_W_Setup_Manual* (PDF) and have the target board fully setup.

Lab 1 : Controlling Raspberry Pi Zero [W]'s ONBOARD LED from userspace – toggling it and modifying the 'trigger'

1. Login (over SHH) to the board; run as root.
2. Examine the onboard LEDs via their sysfs pseudofiles:

```
rpi # ls -l /sys/class/leds/
total 0
lrwxrwxrwx 1 root root 0 Aug 23 10:42 default-on -> ../../devices/virtual/leds/default-
on/
lrwxrwxrwx 1 root root 0 Aug 23 10:42 led0 -> ../../devices/platform/leds/leds/led0/
lrwxrwxrwx 1 root root 0 Aug 23 10:42 mmc0 -> ../../devices/virtual/leds/mmc0/
```

The Raspberry Pi Zero [W] has only one LED represented by the **`led0`** pseudofile (we'll just refer to it as 'file' from now on; understand that it's a pseudofile on sysfs), is the clearly visible LED on the board. (The later Raspberry Pi models have two LEDs – typically using one for power and one for microSD card access).

3. What does it indicate by default?
   To figure this, lookup it's **`trigger`** file:

```
rpi # cat /sys/class/leds/led0/trigger
none rc-feedback kbd-scrolllock kbd-numlock kbd-capslock kbd-kanalock kbd-shiftlock
kbd-altgrlock kbd-ctrllock kbd-altlock kbd-shiftllock kbd-shiftrlock kbd-ctrlllock kbd-
ctrlrlock timer oneshot heartbeat backlight gpio cpu cpu0 default-on input panic
```

```
[actpwr] mmc1 mmc0 rfkill-any rfkill-none rfkill0 rfkill1
```

The entry within square brackets is the default trigger – it's 'actpwr' (aka ACT); it shows whether power is applied.

4. Make it interesting – write a small script on the CLI, allowing you to test different triggers and see their effect on LED0:

```
rpi # while [ true ]
> do
> echo Enter trigger:
> read trig
> echo $trig > /sys/class/leds/led0/trigger
> done
Enter trigger:
heartbeat
Enter trigger:
cpu0
Enter trigger:
actpwr
Enter trigger:
mmc0
…
```

(Use it as a *disk activity LED*: on the RPi Zero, mmc0 is the block device for the /boot partition and mmc1 for the root partition).

5. To take over control of the LED, write `none` into the trigger file; you can now control it via the `brightness` file
    1. writing 1 to `brightness` turns it on
    2. writing 0 to `brightness` turns it off

6. *Assignment*: `led_onboard.sh`
   Write a simple bash (or other) script to toggle the onboard LED (on/off) by the interval specified as a parameter (in milliseconds)

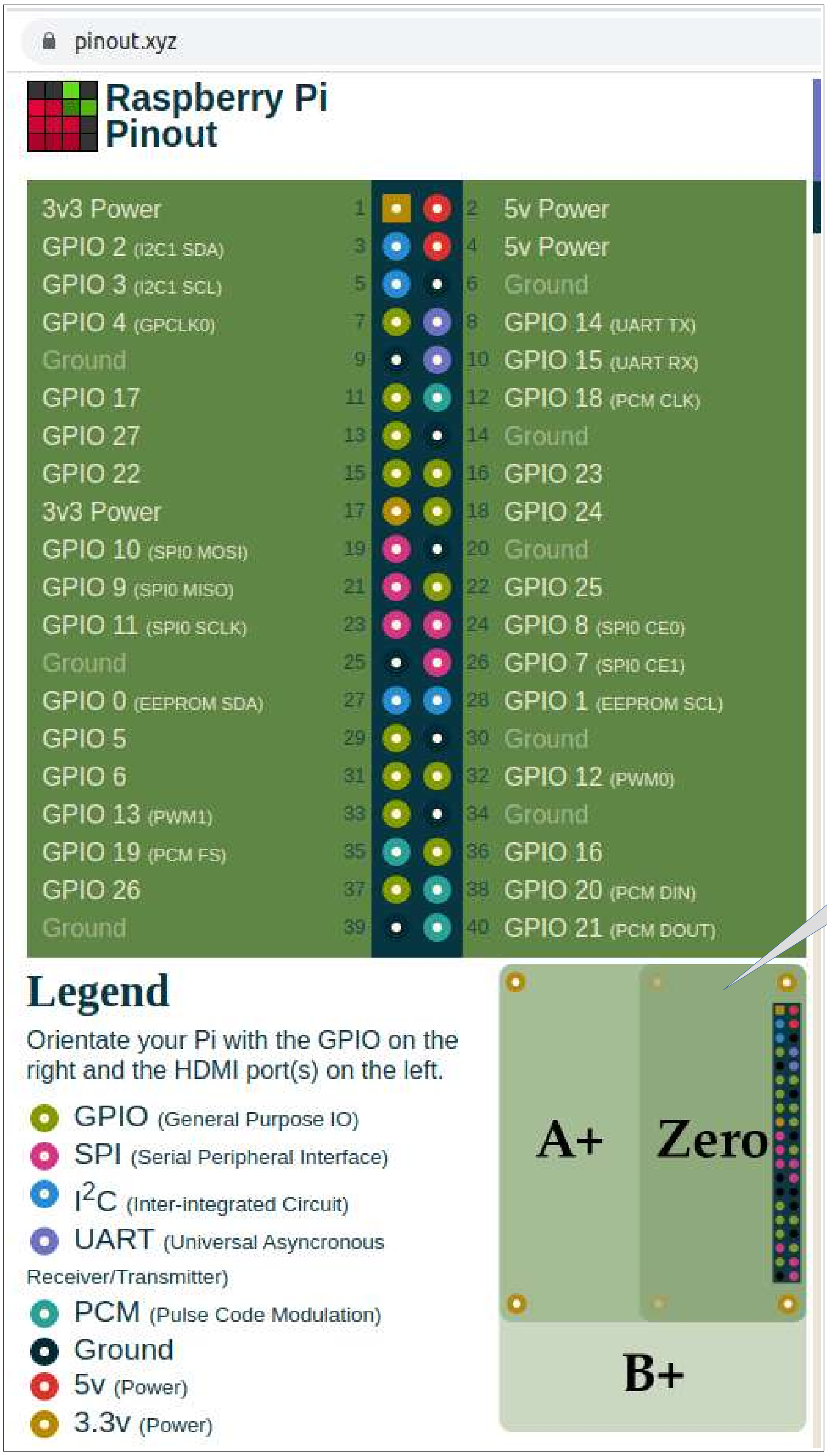
7. *Assignment*: `led_onboard.c`
   Write a C program to toggle the onboard LED (on/off) by the interval specified as a parameter (in milliseconds)

Additional reading:

- Controlling PWR and ACT LEDs on the Raspberry Pi, Jeff Geerling, Mar 2015
- https://forums.raspberrypi.com/viewtopic.php?t=321025
- Blinking an LED via GPIO in Python

Raspberry Pi Zero W pinout

https://pinout.xyz/



Raspberry Pi Pinout

3v3 Power	1	2	5v Power
GPIO 2 (I2C1 SDA)	3	4	5v Power
GPIO 3 (I2C1 SCL)	5	6	Ground
GPIO 4 (GPCLK0)	7	8	GPIO 14 (UART TX)
Ground	9	10	GPIO 15 (UART RX)
GPIO 17	11	12	GPIO 18 (PCM CLK)
GPIO 27	13	14	Ground
GPIO 22	15	16	GPIO 23
3v3 Power	17	18	GPIO 24
GPIO 10 (SPI0 MOSI)	19	20	Ground
GPIO 9 (SPI0 MISO)	21	22	GPIO 25
GPIO 11 (SPI0 SCLK)	23	24	GPIO 8 (SPI0 CE0)
Ground	25	26	GPIO 7 (SPI0 CE1)
GPIO 0 (EEPROM SDA)	27	28	GPIO 1 (EEPROM SCL)
GPIO 5	29	30	Ground
GPIO 6	31	32	GPIO 12 (PWM0)
GPIO 13 (PWM1)	33	34	Ground
GPIO 19 (PCM FS)	35	36	GPIO 16
GPIO 26	37	38	GPIO 20 (PCM DIN)
Ground	39	40	GPIO 21 (PCM DOUT)

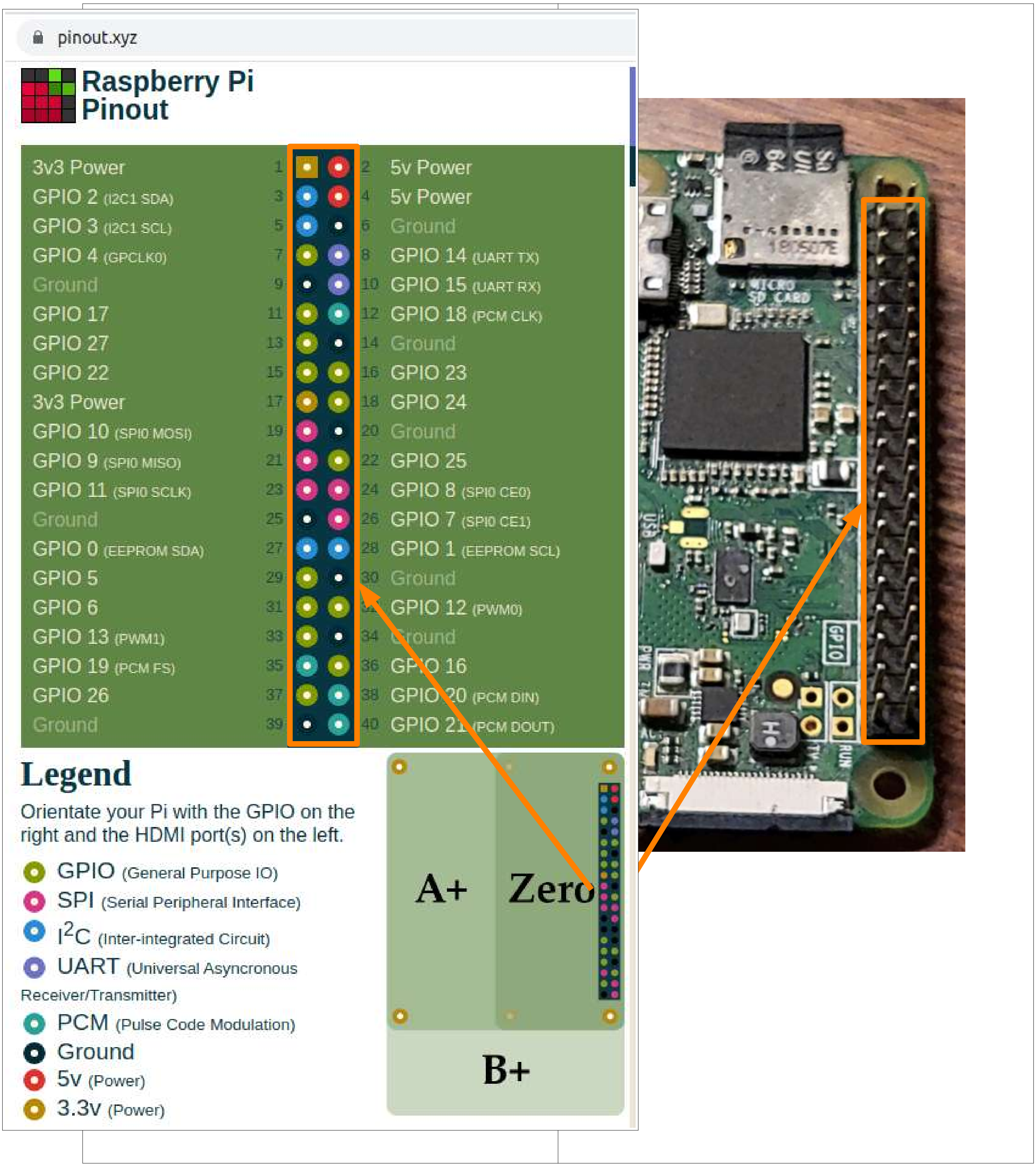
Legend

Orientate your Pi with the GPIO on the right and the HDMI port(s) on the left.

- ○ GPIO (General Purpose IO)
- ○ SPI (Serial Peripheral Interface)
- ○ I^{2}C (Inter-integrated Circuit)
- ○ UART (Universal Asyncronous Receiver/Transmitter)
- ○ PCM (Pulse Code Modulation)
- ○ Ground
- ○ 5v (Power)
- ○ 3.3v (Power)

How you align the board

Pinout side-by-side with the actual board



pinout.xyz

Raspberry Pi Pinout

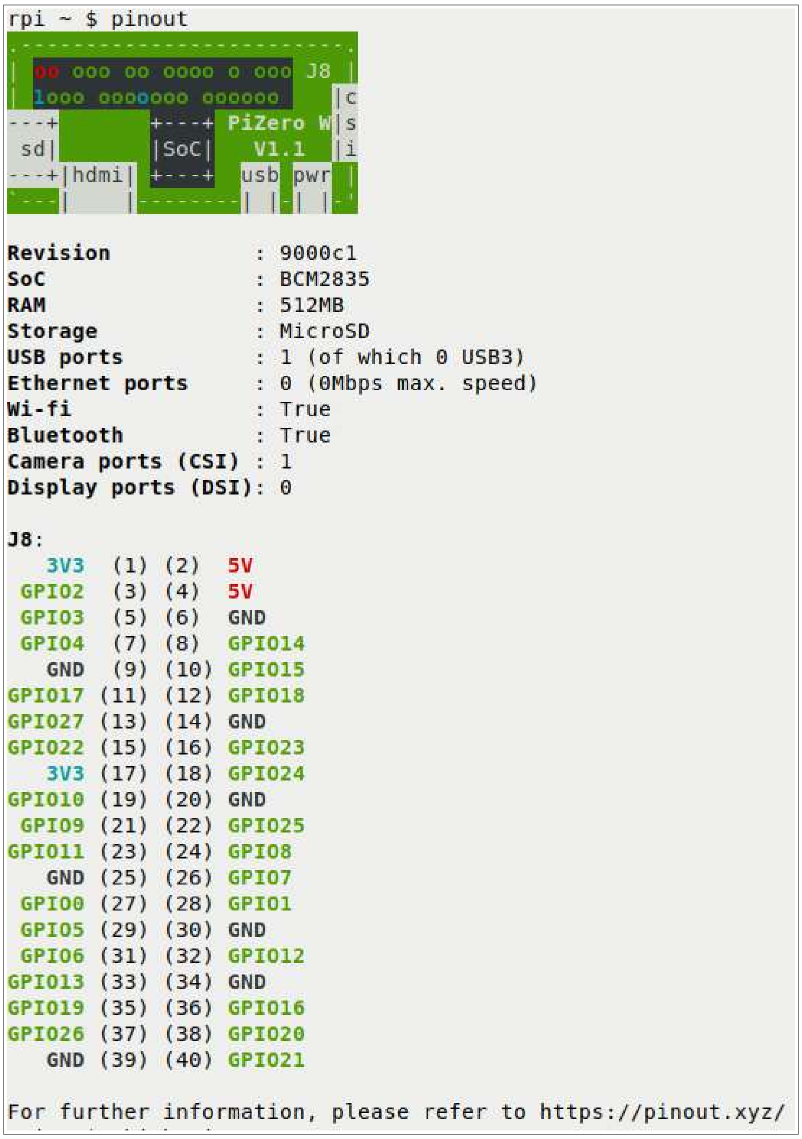
3v3 Power	1	2	5v Power
GPIO 2 (I2C1 SDA)	3	4	5v Power
GPIO 3 (I2C1 SCL)	5	6	Ground
GPIO 4 (GPCLK0)	7	8	GPIO 14 (UART TX)
Ground	9	10	GPIO 15 (UART RX)
GPIO 17	11	12	GPIO 18 (PCM CLK)
GPIO 27	13	14	Ground
GPIO 22	15	16	GPIO 23
3v3 Power	17	18	GPIO 24
GPIO 10 (SPI0 MOSI)	19	20	Ground
GPIO 9 (SPI0 MISO)	21	22	GPIO 25
GPIO 11 (SPI0 SCLK)	23	24	GPIO 8 (SPI0 CE0)
Ground	25	26	GPIO 7 (SPI0 CE1)
GPIO 0 (EEPROM SDA)	27	28	GPIO 1 (EEPROM SCL)
GPIO 5	29	30	Ground
GPIO 6	31	32	GPIO 12 (PWM0)
GPIO 13 (PWM1)	33	34	Ground
GPIO 19 (PCM FS)	35	36	GPIO 16
GPIO 26	37	38	GPIO 20 (PCM DIN)
Ground	39	40	GPIO 21 (PCM DOUT)

Legend

Orientate your Pi with the GPIO on the right and the HDMI port(s) on the left.

- ● GPIO (General Purpose IO)
- ● SPI (Serial Peripheral Interface)
- ● I^{2}C (Inter-integrated Circuit)
- ● UART (Universal Asyncronous Receiver/Transmitter)
- ● PCM (Pulse Code Modulation)
- ● Ground
- ● 5v (Power)
- ● 3.3v (Power)

Tip: Run the awesome *pinout* Python script !



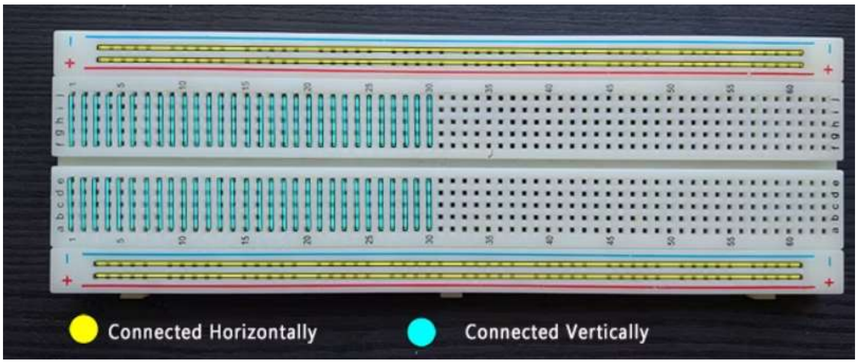
```
rpi ~ $ pinout
.-------------------------.
| oo ooo oo oooo o ooo J8 |
| 1ooo ooooooo oooooo    |c
---+      +---+ PiZero W|s
 sd|      |SoC|   V1.1  |i
---+|hdmi| +---+  usb pwr |
`---|    |--------| |-| |-'

Revision           : 9000c1
SoC                : BCM2835
RAM                : 512MB
Storage            : MicroSD
USB ports          : 1 (of which 0 USB3)
Ethernet ports     : 0 (0Mbps max. speed)
Wi-fi              : True
Bluetooth          : True
Camera ports (CSI) : 1
Display ports (DSI): 0

J8:
   3V3  (1) (2)  5V
 GPIO2  (3) (4)  5V
 GPIO3  (5) (6)  GND
 GPIO4  (7) (8)  GPIO14
   GND  (9) (10) GPIO15
GPIO17 (11) (12) GPIO18
GPIO27 (13) (14) GND
GPIO22 (15) (16) GPIO23
   3V3 (17) (18) GPIO24
GPIO10 (19) (20) GND
 GPIO9 (21) (22) GPIO25
GPIO11 (23) (24) GPIO8
   GND (25) (26) GPIO7
 GPIO0 (27) (28) GPIO1
 GPIO5 (29) (30) GND
 GPIO6 (31) (32) GPIO12
GPIO13 (33) (34) GND
GPIO19 (35) (36) GPIO16
GPIO26 (37) (38) GPIO20
   GND (39) (40) GPIO21

For further information, please refer to https://pinout.xyz/
```

Lab 2 : Controlling EXTERNAL LEDs with the Raspberry Pi Zero [W]’s onboard GPIO pins from userspace

1. The hardware part: what you’ll need:

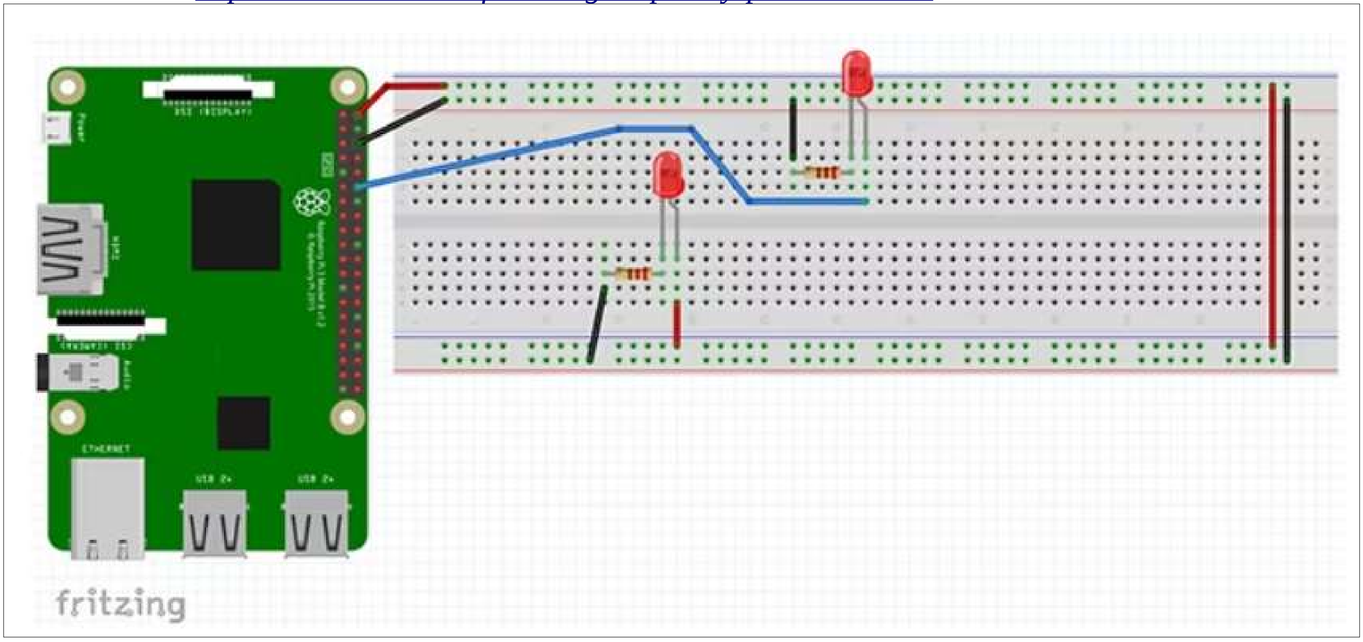
   ○ The SBC – Raspberry Pi Zero W (or any other model, for that matter)
   ○ A breadboard
   ○ 2 x LEDs
   ○ 2 x resistors (anything from 220 Ohm to 1 kOhm)
   ○ Hookup cables

2. Prerequisites:
   Read these:
   - *How to Program Your Raspberry Pi to Control LED Lights, Ian Buckly, July 2018*
   - *IMP: What Is a Breadboard and How Do You Use One?*



3. The Circuit:

   Src: https://www.makeuseof.com/tag/raspberry-pi-control-led/



Connect as shown.

REALLY Important – when working with hardware

- Ensure the power supply ratings are the right and official one for the board; more problems are caused by incorrect / cheap power supplies than anything else!
- Using a breadboard is nice for convenience and “trying things out” but can be a huge pain when jumper cables / wires are of poor quality and/or the connections aren’t good or stable (soldering is superior but requires a PCB and more effort/skill)
- When connecting stuff, always power down the board, remove the power cable and then work on it (using a grounding wire on the lab table is a good idea too)
- When it doesn’t work, check, and then recheck, all your wiring. Is it loose? Did you read the (GPIO/other) pins right (Board #s vs BCM GPIO #s, etc).

4. Login (over SHH) to the board; run as root

5. The GPIOs can be exposed; they’re seen as sysfs files (technically, the GPIO sysfs interface is deprecated in favour of a kernel-level char driver API interface; we can still use the sysfs interface for a while though...):

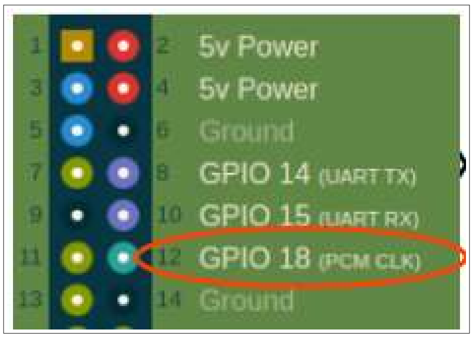
```
# ls -l /sys/class/gpio/
total 0
--w--w---- 1 root gpio 4096 Sep  9 16:21 export
lrwxrwxrwx 1 root gpio    0 Sep  9 16:21 gpiochip0 ->
../../devices/platform/soc/20200000.gpio/gpio/gpiochip0/
--w--w---- 1 root gpio 4096 Sep  9 16:21 unexport
#
```

‘Export’ the required GPIO pin, making it ‘appear’ here, by writing the (Broadcom) GPIO pin # into the export pseudofile:

```
# echo 18 > /sys/class/gpio/export
# ls -l /sys/class/gpio/
total 0
--w--w---- 1 root gpio 4096 Sep 10 12:44 export
lrwxrwxrwx 1 root root    0 Sep 10 12:44 gpio18 -> 
../../devices/platform/soc/20200000.gpio/gpiochip0/gpio/gpio18/
lrwxrwxrwx 1 root gpio    0 Sep  9 19:13 gpiochip0 ->
../../devices/platform/soc/20200000.gpio/gpio/gpiochip0/
--w--w---- 1 root gpio 4096 Sep  9 19:13 unexport
#
```

NOTE! CAREFUL!

The GPIO pin # to use is Not the physical pin number (seen on the pinout), rather it’s the number GPIOn. So, here, when we connect the wire to physical pin 12 it corresponds to GPIO 18! That’s the number to use. (It’s called the Broadcom (BCM) number as opposed to the physical board number; we’re using the BCM #).

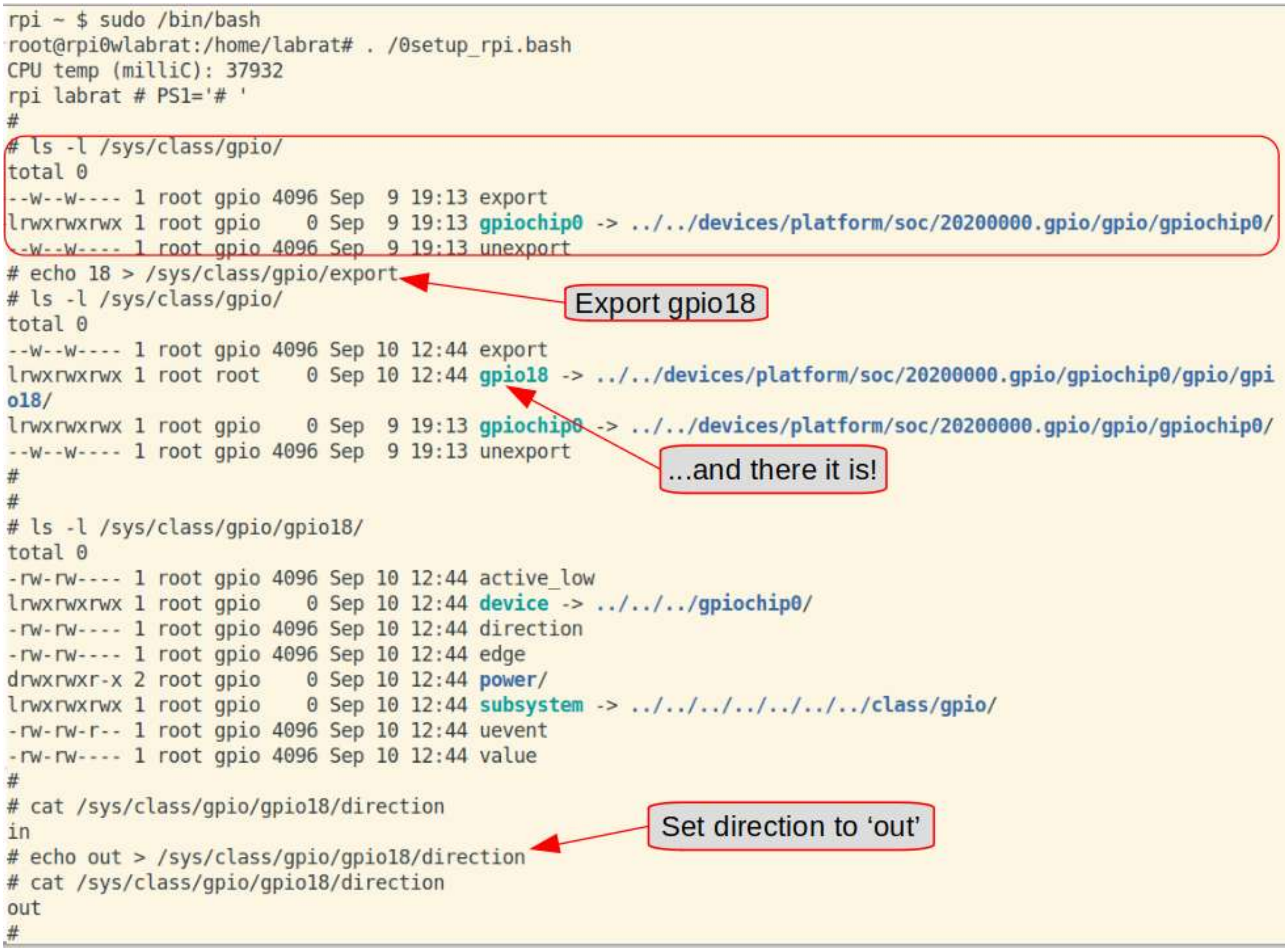


The (truncated) output of the *pinout* app shows the physical board # in brackets and the Broadcom # as GPIOn:

```
[...]  GND  (9) (10) GPIO15
    GPIO17 (11) (12) GPIO18
    GPIO27 (13) (14) GND    [...]
```

Ref: *Difference between BCM and BOARD pin numbering in Raspberry Pi*

The screenshot shows how we set it up:



```
rpi ~ $ sudo /bin/bash
root@rpi0wlabrat:/home/labrat# . /0setup_rpi.bash
CPU temp (milliC): 37932
rpi labrat # PS1='# '
#
# ls -l /sys/class/gpio/
total 0
--w--w---- 1 root gpio 4096 Sep  9 19:13 export
lrwxrwxrwx 1 root gpio    0 Sep  9 19:13 gpiochip0 -> ../../devices/platform/soc/20200000.gpio/gpio/gpiochip0/
--w--w---- 1 root gpio 4096 Sep  9 19:13 unexport
# echo 18 > /sys/class/gpio/export
# ls -l /sys/class/gpio/
total 0
--w--w---- 1 root gpio 4096 Sep 10 12:44 export
lrwxrwxrwx 1 root root    0 Sep 10 12:44 gpio18 -> ../../devices/platform/soc/20200000.gpio/gpiochip0/gpio/gpi
o18/
lrwxrwxrwx 1 root gpio    0 Sep  9 19:13 gpiochip0 -> ../../devices/platform/soc/20200000.gpio/gpio/gpiochip0/
--w--w---- 1 root gpio 4096 Sep  9 19:13 unexport
#
#
# ls -l /sys/class/gpio/gpio18/
total 0
-rw-rw---- 1 root gpio 4096 Sep 10 12:44 active_low
lrwxrwxrwx 1 root gpio    0 Sep 10 12:44 device -> ../../../gpiochip0/
-rw-rw---- 1 root gpio 4096 Sep 10 12:44 direction
-rw-rw---- 1 root gpio 4096 Sep 10 12:44 edge
drwxrwxr-x 2 root gpio    0 Sep 10 12:44 power/
lrwxrwxrwx 1 root gpio    0 Sep 10 12:44 subsystem -> ../../../../../../../class/gpio/
-rw-rw-r-- 1 root gpio 4096 Sep 10 12:44 uevent
-rw-rw---- 1 root gpio 4096 Sep 10 12:44 value
#
# cat /sys/class/gpio/gpio18/direction
in
# echo out > /sys/class/gpio/gpio18/direction
# cat /sys/class/gpio/gpio18/direction
out
#
```

Export gpio18

...and there it is!

Set direction to 'out'

6. The actual state of the GPIO pin can be seen / read and changed via it's `value` sysfs file:

i. Set the GPIO pin high by writing 1 into `value` (thus turning the LED on):

```
echo 1 > /sys/class/gpio/gpio18/value
```

ii. Delay a bit...

```
sleep 1
```

iii. Set the GPIO pin low by writing 0 into `value` (thus turning the LED off):

```
echo 0 > /sys/class/gpio/gpio18/value
sleep 1
```

Done.

8. *Assignment*: `ledblink.sh`
   Write a simple bash script to toggle an external LED on a breadboard circuit on/off by the interval specified as a parameter (in milliseconds)

9. *Assignment*: `ledblink.c`
   Write a C program to toggle an external LED on a breadboard circuit on/off by the interval specified as a parameter (in milliseconds).

Ref:

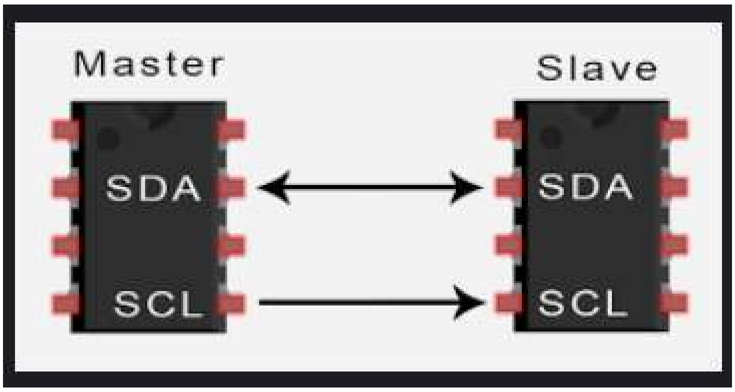
- *GPIO Programming: Using the sysfs Interface*
- *GPIO Sysfs Interface for Userspace – kernel documentation;* explains the layout and meaning of the pseudo-files under sysfs relating to gpio; also stresses that one should NOT abuse this interface when a proper / official kernel driver exists for the hardware being driven!
- https://pimylifeup.com/raspberry-pi-gpio/

I2C and the DHT2x temperature + humidity sensor

First, learn about the very popular widely used I2C (Inter-Inter Connect) 2-wire protocol that drives chip like this one!

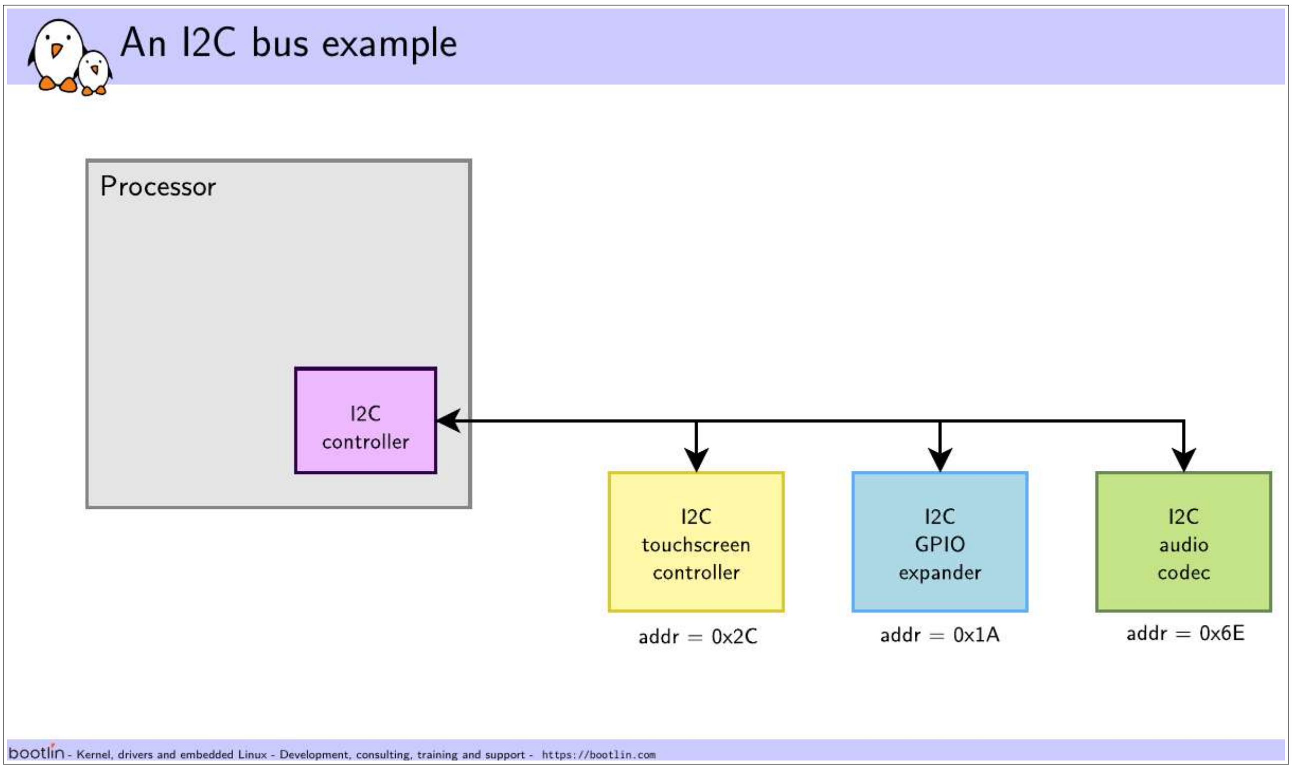
Refer:
Linux Kernel and Driver Development, Bootlin (aka Free electrons): linux-kernel-and-driver-dev.pdf : pg 172 – 185

Ref (for notes following):
https://learn.sparkfun.com/tutorials/i2c/all



<Master/Controller> **<Client chip or peripheral>**

- simple and efficient; only 2 wires (SCL – Serial CLock, SDA – Serial DaTa)
- master / controller initiates communication with the slave or client chip
- I2C controller is usually part of the SoC or processor
- 'Each slave device is identified by a unique I2C address. Each transaction initiated by the master contains this address, which allows the relevant slave to recognize that it should reply to this particular transaction.'



An I2C bus example

bootlin - Kernel, drivers and embedded Linux - Development, consulting, training and support - https://bootlin.com

I2C Address List

- Supports
  - multiple controllers (unlike SPI)
  - multiple clients - up to 1008 peripherals (client chips)!
  - clock speeds
    - 0 to 5 Mhz (original I2C)
    - 10 KHz to 100 KHz (Intel's *System Management Bus (SMBus)* version; more controlled protocol)
- 7 bit addresses for addressing clients; implies client addresses range from 0 to 127 (0x0 to 0x7F).

Ref:

- I2C protocol basics- 'How to use I2C in STM32F103C8T6? STM32 I2C Tutorial'
- Linux Kernel and Driver Development, Bootlin (aka Free electrons): linux-kernel-and-driver-dev.pdf : pg 172 – 185
- *Kernel doc : Implementing I2C device drivers*
- *Kernel doc: Implementing I2C client drivers in user-space*
- *Raspberry Pi SPI and I2C Tutorial*
- *Interfacing with I2C Devices (eLinux)*

With the DHT11 sensor chip
(Kernel drv: drivers/iio/humidity/dht11.c)

For all code / docs refer the GitHub repo here:
https://github.com/kaiwan/labrat_drv/tree/main/dht2x_temp_humd_i2c_driver

1. Set up a connection – via a USB-to-serial console cable or SSH (preferred) – to your hardware board. Here, we assume it's a Raspberry Pi (in particularly, am working with a Raspberry Pi 4 Model B; relevant for the DTS / DTB!)
   Login to the board.

2. Ensure that I2C is enabled (`sudo raspi-config`)

3. Ensure the following packages are installed:
   ```
   sudo apt install -y i2c-tools libi2c-dev python3-smbus
   ```

4. Obtain the board's DTS, edit it to include the DHT2x I2C sensor chip, compile it to the DT blob (DTB file), set the new DTB as the one used at boot.
   1. In the repo, here's the edited DTS.. The stanza added:
      ```
      ...
      i2c@7e804000 {
         pinctrl-names = "default";
         #address-cells = <0x01>;
         [...]

         /* KNB: added node for DHT2x temp/humd sensor chip, to match my driver
          * Also note it's added in the right place, under the relevant I2C node
          */
         dht2x: dht22@0 {
               compatible = "knb,dht2x";
               reg = <0x38>;
               pinctrl-names = "default";
               /*       pinctrl-0 = <&dht11_pins>; */
               status = "okay";
               #clock-cells = <1>;
         };
      };
      [...]
      ```
   2. Compile DTS:
      ```
      $ dtc my_rpi4b_dht2x.dts -o my_rpi4b_dht2x.dtb 2>&1 |cut -d: -f2-| grep -i dht2
      256.19-263.6: Warning (i2c_bus_reg): /soc/i2c@7e804000/dht22@0: I2C bus unit address format error, expected "38"
      ```

      Looks like we can ignore this warning...

   3. Copy the new DTB into /boot, then edit /boot/config.txt and add this line
      `device_tree=my_rpi4b_dht2x.dtb`
      in order to override the default DTB with ours...
      (Ensure you keep the original DTB (here, it's */boot/bcm2711-rpi-4-b.dtb*) intact!)
      1. Ref:
         1. Raspberry Pi /boot/config.txt :
            https://www.raspberrypi.com/documentation/computers/config_txt.html
         2. *Raspberry Pi DTBs, overlays and config.txt* :
            https://www.raspberrypi.com/documentation/computers/configuration.html#part3.1
      2. Test by rebooting; if all okay, the board reboots correctly, and you can see your new entry for the DHT2x within /proc/device-tree !
         NOTE- if your board does not reboot, it's likely because the DTB isn't good; remove

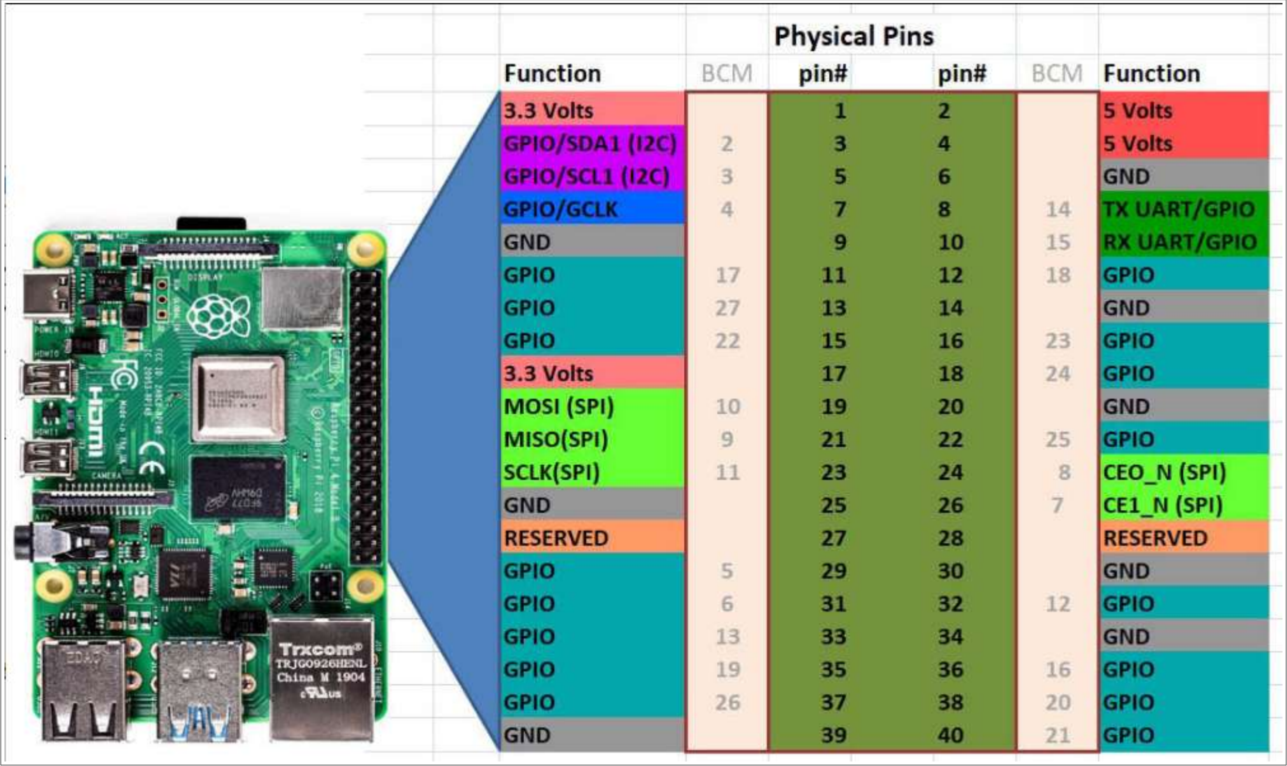
the ‘device_tree=<...>’ line in config.txt (IOW, let it use the original DTB), reboot, fix the issues and retry...

Rebooted with the new DTB; can see it’s ok:

```
$ ls /proc/device-tree/soc/i2c@7e804000/dht22@0/
'#clock-cells'   compatible   name   pinctrl-names   reg   status
```

4. Shutdown, power off, and connect the DHT2x sensor to your board:

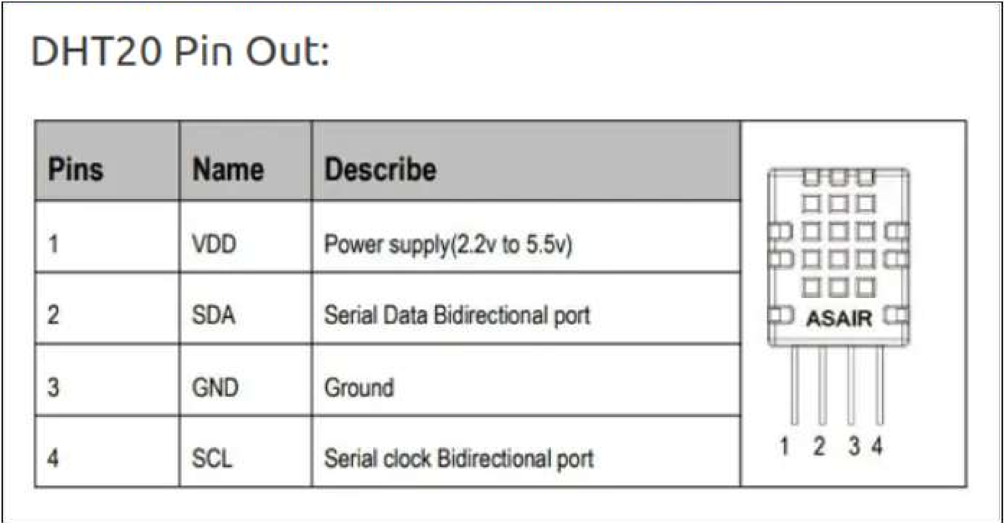
Raspberry Pi 4 (and Pi 3 / Pi 0W) pinout:



		Physical Pins			
Function	BCM	pin#	pin#	BCM	Function
3.3 Volts		1	2		5 Volts
GPIO/SDA1 (I2C)	2	3	4		5 Volts
GPIO/SCL1 (I2C)	3	5	6		GND
GPIO/GCLK	4	7	8	14	TX UART/GPIO
GND		9	10	15	RX UART/GPIO
GPIO	17	11	12	18	GPIO
GPIO	27	13	14		GND
GPIO	22	15	16	23	GPIO
3.3 Volts		17	18	24	GPIO
MOSI (SPI)	10	19	20		GND
MISO(SPI)	9	21	22	25	GPIO
SCLK(SPI)	11	23	24	8	CEO_N (SPI)
GND		25	26	7	CE1_N (SPI)
RESERVED		27	28		RESERVED
GPIO	5	29	30		GND
GPIO	6	31	32	12	GPIO
GPIO	13	33	34		GND
GPIO	19	35	36	16	GPIO
GPIO	26	37	38	20	GPIO
GND		39	40	21	GPIO

DHT2x pinout

ref: https://proto-pic.co.uk/product/humidity-and-temperature-sensor-dht20/



DHT20 Pin Out:

Pins	Name	Describe
1	VDD	Power supply(2.2v to 5.5v)
2	SDA	Serial Data Bidirectional port
3	GND	Ground
4	SCL	Serial clock Bidirectional port

VDD (power) to +3.3V (pin 1 on the Raspberry Pi)

5. After starting up with it attached, i2cdetect should show it:

```
$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- 38 -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Perfect – it’s on I2C bus 1, device address 0x38, as expected!

6. Write / edit the device driver, build it, test, install it:

(I found that without the `sudo depmod` in the Makefile, the driver gets installed but not detected and loaded up at boot, even with putting it into /etc/modules-load.d/<name>.conf !

```
rpi 5.15.76-v8+ # grep -i dht2x *
grep: build: Is a directory
grep: extra: Is a directory
grep: kernel: Is a directory

rpi 5.15.76-v8+ # depmod
[...]
rpi 5.15.76-v8+ # grep -i dht2x *
grep: build: Is a directory
grep: extra: Is a directory
grep: kernel: Is a directory
modules.alias:alias i2c:knb,dht2x dht2x_kdrv
modules.alias:alias of:N*T*Cknb,dht2xC* dht2x_kdrv
modules.alias:alias of:N*T*Cknb,dht2x dht2x_kdrv
grep: modules.alias.bin: binary file matches
modules.dep:extra/dht2x_kdrv.ko.xz:
grep: modules.dep.bin: binary file matches
rpi 5.15.76-v8+ #
```

)

The driver code and Makefile is *in the GitHub repo*...

[Though unnecessary here, in general, to have the kernel auto-load the driver, create this file:

```
$ cat /etc/modules-load.d/dht2x_kdrv.conf
# The DHT2x temp+humd I2C sensor chip
dht2x_kdrv
$
```

Here, the kernel bus driver, detecting that the DHT2x chip’s present, auto-loads (via the udev mechanism) the driver!

7. Reboot; the driver should now be auto-loaded; the (I2C) bus driver pairs it with the sensor chip and the probe() method gets called. Great!

```
$ lsmod |grep dht
dht2x_kdrv             16384  0
$ dmesg |grep -i dht2x
[    3.199212] dht2x_kdrv: loading out-of-tree module taints kernel.
[    5.017455] dht2x 1-0038: dht2x_probe(): hey, in probe! name=dht2x addr=0x38
[    5.349181] dht2x 1-0038: dht2x_probe(): chip status (0x1c): calibration[b3]:
0x8   busy[b7]: 0x0
[    5.349218] dht2x 1-0038: dht2x_probe(): chip found
$
```

Moreover, the sysfs hooks are setup and ready to use; a quick demo shows its working just fine:

```
$ ls -l /sys/bus/i2c/devices/1-0038/
total 0
-r--r--r-- 1 root root 4096 Nov 25 14:28 dht2x_humd
-r--r--r-- 1 root root 4096 Nov 25 14:28 dht2x_temp
lrwxrwxrwx 1 root root    0 Nov 25 14:01 driver ->
../../../../../../bus/i2c/drivers/dht2x/
-r--r--r-- 1 root root 4096 Nov 25 14:28 modalias
-r--r--r-- 1 root root 4096 Nov 25 14:01 name
lrwxrwxrwx 1 root root    0 Nov 25 14:28 of_node ->
'../../../../../../firmware/devicetree/base/soc/i2c@7e804000/dht22@0'/
drwxr-xr-x 2 root root    0 Nov 25 14:28 power/
lrwxrwxrwx 1 root root    0 Nov 25 14:01 subsystem -> ../../../../../../bus/i2c/
-rw-r--r-- 1 root root 4096 Nov 25 14:01 uevent
$

$ cat /sys/bus/i2c/devices/1-0038/dht2x_humd
77507$
$ cat /sys/bus/i2c/devices/1-0038/dht2x_temp
23427$
$
```

Realize, of course, that the

- humidity value is in milli-percentage points (so humidity is currently 77.507%)
- similarly, temperature is expressed in millidegrees Celsius (so temperature is currently 23.427 degC)

Success!