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 – for the DHT2x temperature+humidity sensor chip (refer separate doc)

- 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

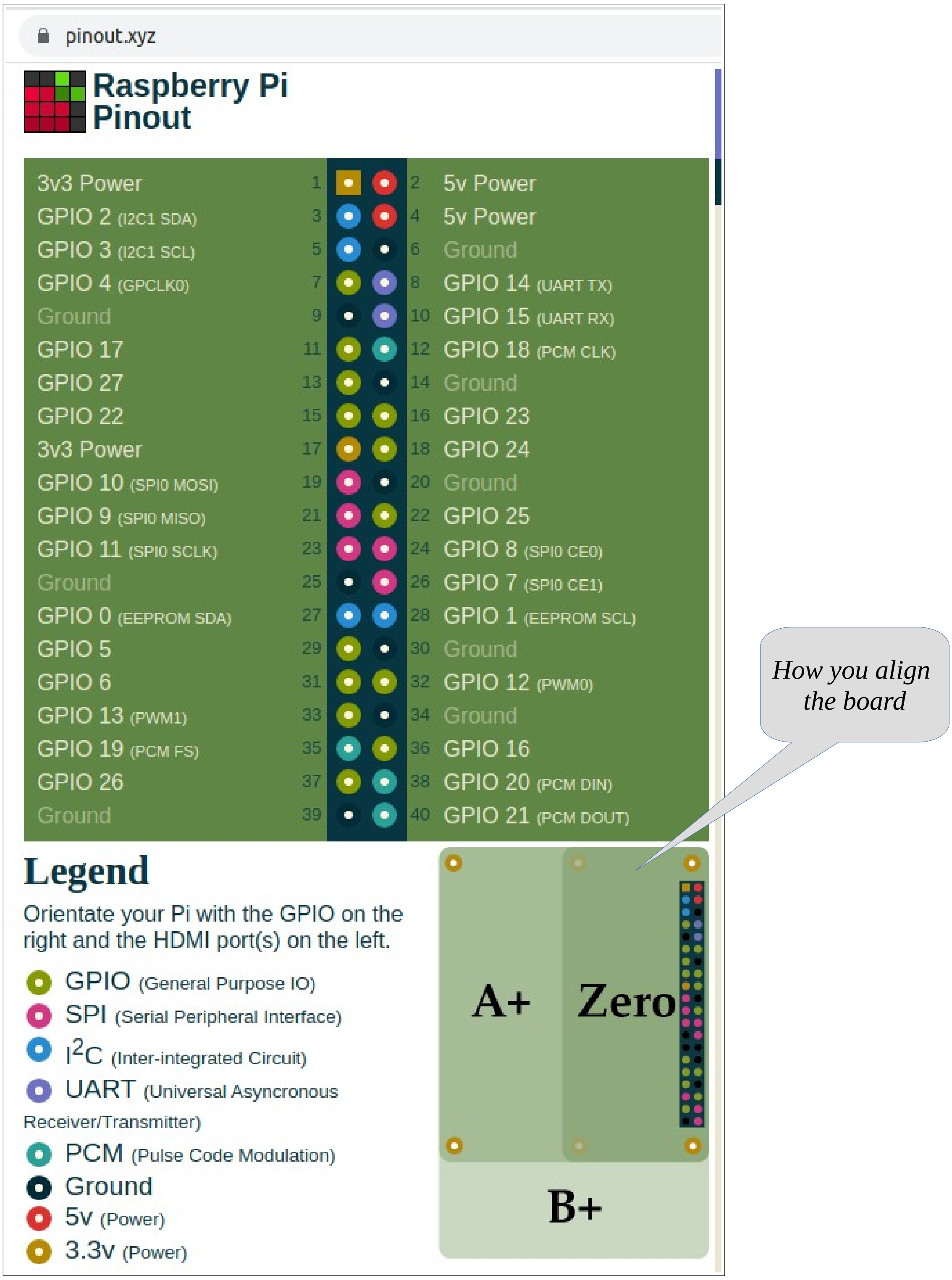
6. *Assignment*: `led_onboard.sh`
   Write a simple bash (or other) script to toggle the onboard LED (on/off) by the interval specified as a parameter (in milliseconds)

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)

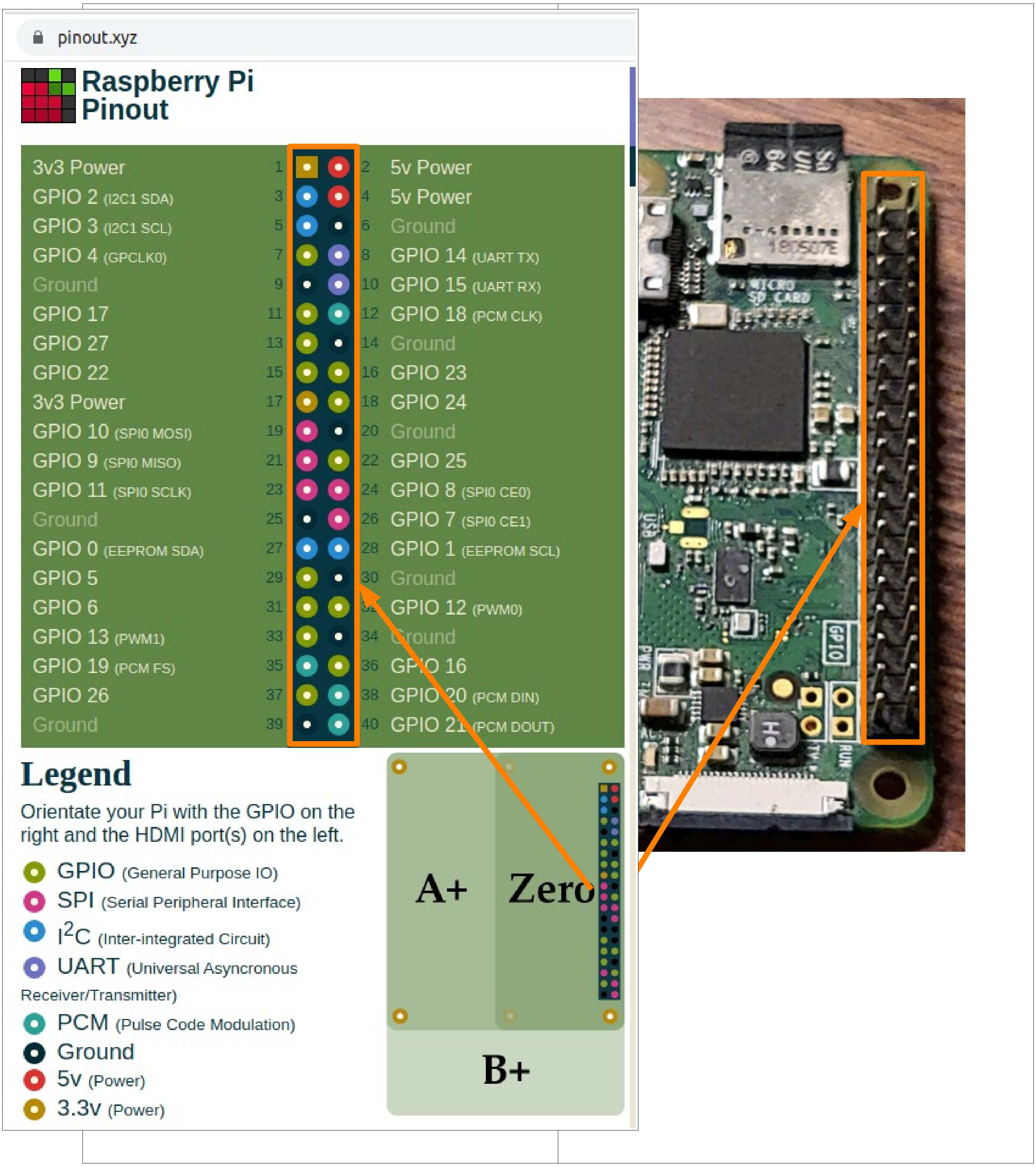
Additional reading:
- [Controlling PWR and ACT LEDs on the Raspberry Pi, Jeff Geerling, Mar 2015]
- [https://forums.raspberrypi.com/viewtopic.php?t=321025]
- [Blinking an LED via GPIO in Python]

Raspberry Pi Zero W pinout

https://pinout.xyz/



Pinout side-by-side with the actual board



Tip: Run the awesome *pinout* Python script !

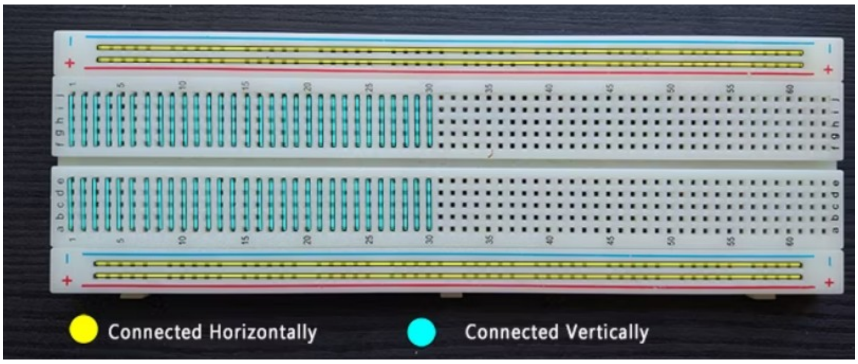
```
rpi ~ $ pinout
.-------------------------.
|  oo ooo oo oooo o ooo J8 |
|  1ooo oooooooo oooooo    |c
---+       +---+ PiZero W|s
 sd|       |SoC|   V1.1  |i
---+|hdmi| +---+  usb pwr |
`---|    |--------| |-| |-'

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:

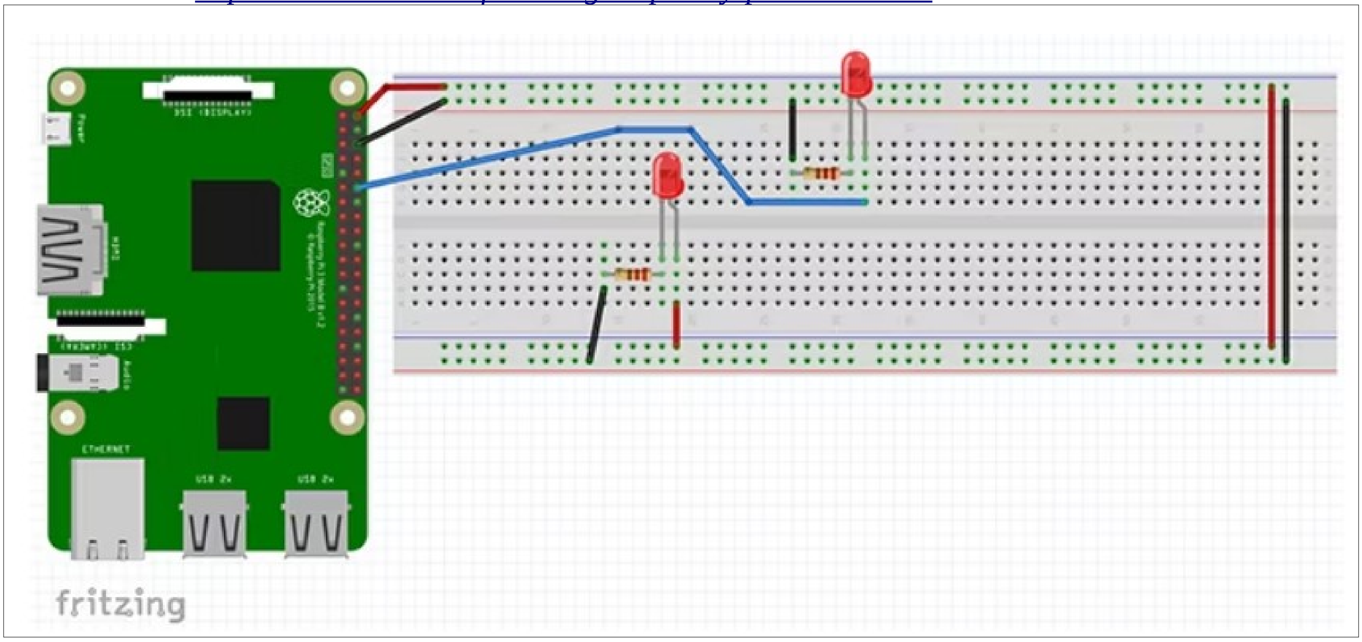
   - The SBC – Raspberry Pi Zero W (or any other model, for that matter)
   - A breadboard
   - 2 x LEDs
   - 2 x resistors (anything from 220 Ohm to 1 kOhm)
   - Hookup cables

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
#
```


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/
