ELEC 424/553

Mobile & Embedded Systems

Lecture 14

GPIO (2)

Picture URL: https://paltonico.com/embedded/driving-2-leds-with-only-2-gpio/

In (Old) Recent News

theverge.com

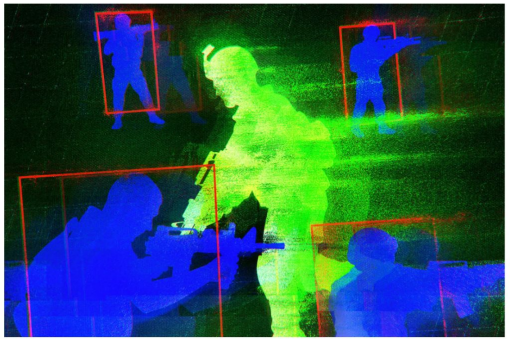
The Verge

Menu

ENTERTAINMENT / GAMING / ESPORTS

EA announces kernel-level anti-cheat system for PC games
/ EA's new system starts with FIFA 23 this fall

By TOM WARREN / @tomwarren
Sep 13, 2022, 4:51 PM CDT
12 Comments / 12 New



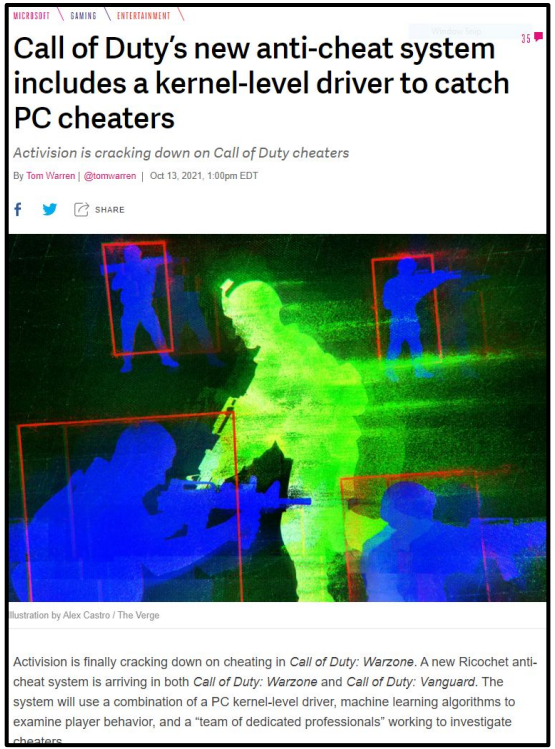
EA's new anti-cheat arrives in FIFA 23 *first*. Illustration by Alex Castro / The Verge

Electronics Arts (EA) is launching a new kernel-level anti-cheat system for its PC games. The EA AntiCheat (EAAC) will debut first in *FIFA 23* later this fall and is a custom anti-cheat system developed in-house by EA developers. It's designed to protect EA games from tampering and cheaters, and EA says it won't add anti-cheat to *every* game and treat its implementation on a case-by-case basis.

"PC cheat developers have increasingly moved into the kernel, so we need to

https://www.theverge.com/2022/9/13/23351900/ea-kernel-level-anti-cheat-system-fifa-23

News From 2021 - Call of Duty's Driver



MICROSOFT \ GAMING \ ENTERTAINMENT \

Call of Duty's new anti-cheat system includes a kernel-level driver to catch PC cheaters

Activision is cracking down on Call of Duty cheaters

By Tom Warren | @tomwarren | Oct 13, 2021, 1:00pm EDT

SHARE

Illustration by Alex Castro / The Verge

Activision is finally cracking down on cheating in *Call of Duty: Warzone*. A new Ricochet anti-cheat system is arriving in both *Call of Duty: Warzone* and *Call of Duty: Vanguard*. The system will use a combination of a PC kernel-level driver, machine learning algorithms to examine player behavior, and a "team of dedicated professionals" working to investigate cheaters.

"'Cheaters aren't welcome. There's no tolerance for cheaters, and soon you'll know what we mean.'"
[from Call of Duty Twitter account here]

"The system will use a combination of a PC **kernel-level driver**, machine learning algorithms to examine player behavior, and a "team of dedicated professionals" working to investigate cheaters."

"PC games are increasingly using **kernel-level drivers** to detect sophisticated cheating, but since they **run at such a high level** in Windows, there are always **privacy concerns** surrounding such an approach."

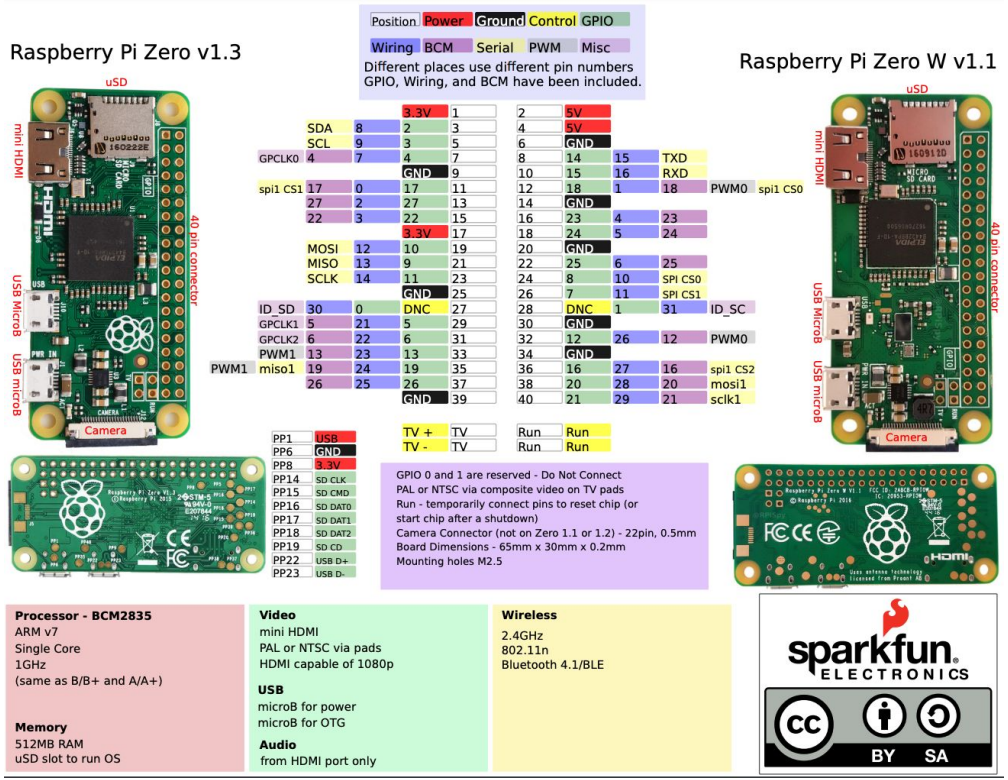
"The **driver itself will monitor processes** interacting with Warzone to **see if they're trying to inject code or manipulate the game**, and report the results back."

https://www.theverge.com/2021/10/13/22724037/call-of-duty-ricochet-anti-cheat-system-kernel-level-driver

Assignment 2

- Use libgpiod with GPIO pins
  - Button, LED
- Lecture today will further help with this project
- Due Saturday
- Any questions?

GPIO: General-Purpose Input/Output



GPIO Pins

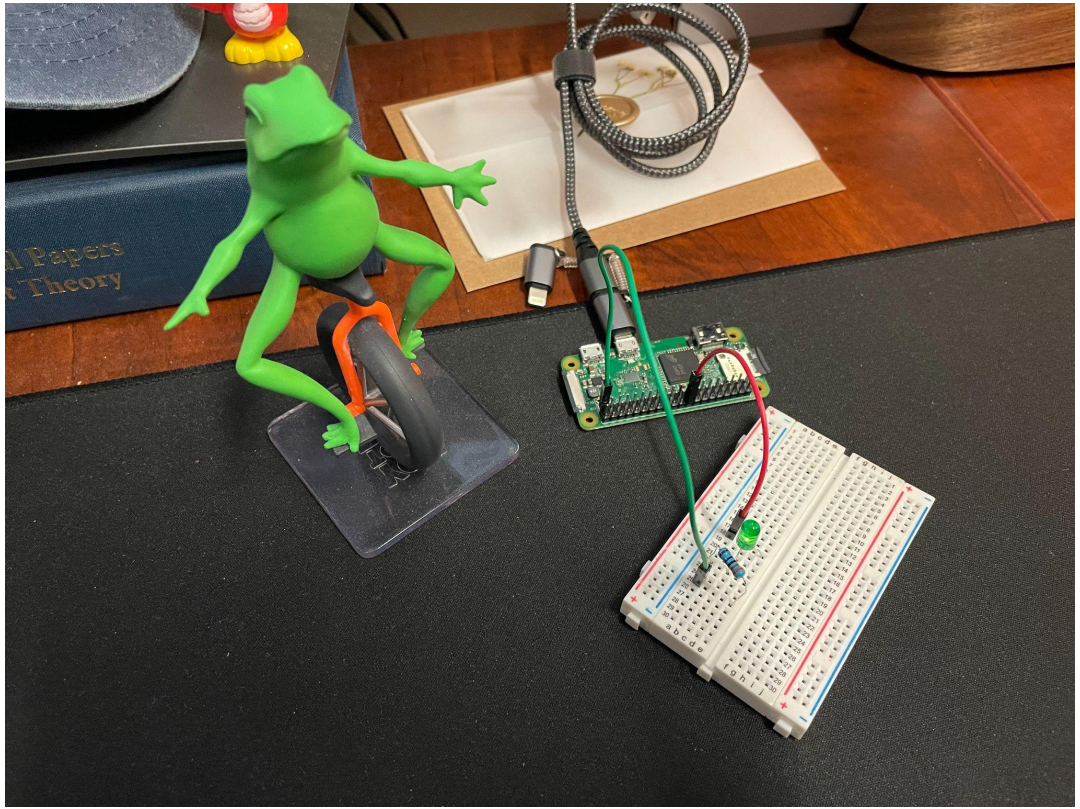
"behavior (including whether it is an input or output pin) can be controlled / programmed by the user at run time."

"GPIO", SUNXI. URL: https://linux-sunxi.org/GPIO

Can trigger interrupts

Diagram from SparkFun. License: CC BY-SA (link). No changes made. URL: https://cdn.sparkfun.com/assets/learn_tutorials/6/7/6/PiZero_1.pdf

Cute Demo of LEDs Turning On From RPi



Assignment 2: Important Note

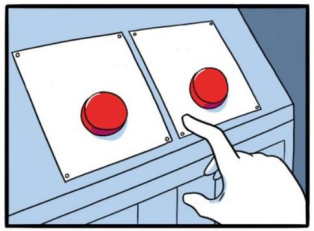
Be very careful what you connect

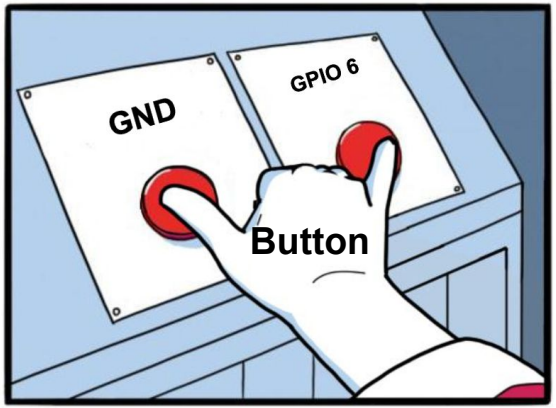
What does this mean for your logic?

Why does having the button connect GPIO 6 to GND work?

Do not directly connect 3.3V, 5V, and GND to each other

Bad, bad, bad things will happen





Doge picture from Dogecoin. URL: https://dogecoin.com/
Bottom middle meme from Imgflip LLC. URL: https://imgflip.com/memegenerator/Two-Buttons
Right meme from Piñata Farms. URL: https://www.pinatafarm.com/memegenerator/89499743-d4aa-4133-975e-e5ef9bfc2141

What did we just do?

Insightful Student Questions

“For *trigger.sh*, a shell type, this is a file that executes something. For *10-module.rules* , this is a file located in the root. This is a file that executes if something happens (in this case, *insmod hello.ko*)

How come trigger.sh is stored in /local/bin ? Can shell files be stored in any location that is not the root? Where can I learn more about different possibilities for this? For example, how could I make an LED on breadboard turn on for *sudo insmod hello.ko*, then LED turn off for *sudo insmod hello.ko*? Is a file in a root the appropriate way?

I guess what I'm wondering: What are shell types typically used for? What file type is 10-module.rules and what is it typically used for? What are some fun applications of this?”

Install git on your Pi

More Time For Exercise 11
Access code: fire
Solution code: chocolate

Install git on your Pi during this time:

```
sudo apt update
sudo apt install git -y
```

Shell Scripting Example - `sudo nano /usr/local/bin/trigger.sh`

```
#!/bin/bash
```

<- From reference at bottom

```
/bin/date >> /tmp/udev.log
```

Note: > overwrites
>> appends

Seth Kenlon, “An introduction to Udev: The Linux subsystem for managing device events”. URL: https://opensource.com/article/18/11/udev

Shell Scripting Example - Execute

```
$ sudo ls -al /usr/local/bin

$ sudo chmod +x /usr/local/bin/trigger.sh
```

<- From reference at bottom

```
$ sudo ls -al /usr/local/bin

$ /usr/local/bin/trigger.sh

(in another window): $ sudo tail -f /tmp/udev.log
```

Seth Kenlon, “An introduction to Udev: The Linux subsystem for managing device events”. URL: https://opensource.com/article/18/11/udev

Let’s Try Out udev With Our Fake Device

- Switch to root: **`$ sudo su -`**
- **`$ udevadm monitor`**
- Insert the hello.ko module, see what happens
- Remove the hello.ko module, see what happens
- Make new file in **`/etc/udev/rules.d`**
  - E.g. **`10-module.rules`**
  - It should contain one line: **`SUBSYSTEM=="mes", ACTION=="add", RUN+="/usr/local/bin/trigger.sh"`**
  - What do we use for subsystem? Stack Exchange
- **`$ udevadm control --reload`**
- Insert and remove module while **`tailing`** log in **`/tmp/udev.log`**

Modified from: Seth Kenlon, “An introduction to Udev: The Linux subsystem for managing device events”. URL: https://opensource.com/article/18/11/udev

More GPIO

Accessing GPIO

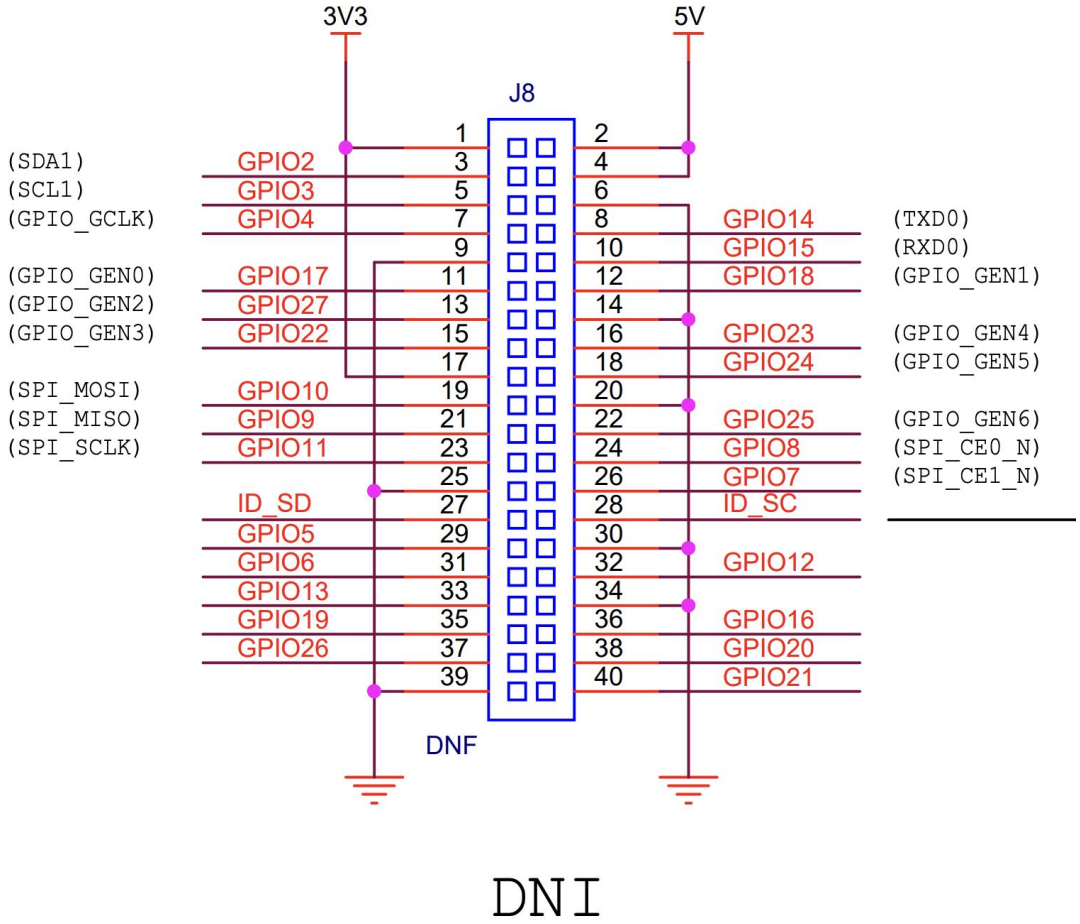
- **Kernel space**
  - `gpio`
  - **`gpiod`**
- **User space**
  - **`sysfs`**
  - `Char dev` [Kernel 4.8]
    - **`libgpiod`**
      - Command line & C program

Example: sysfs

Where In The World Is gpio6?

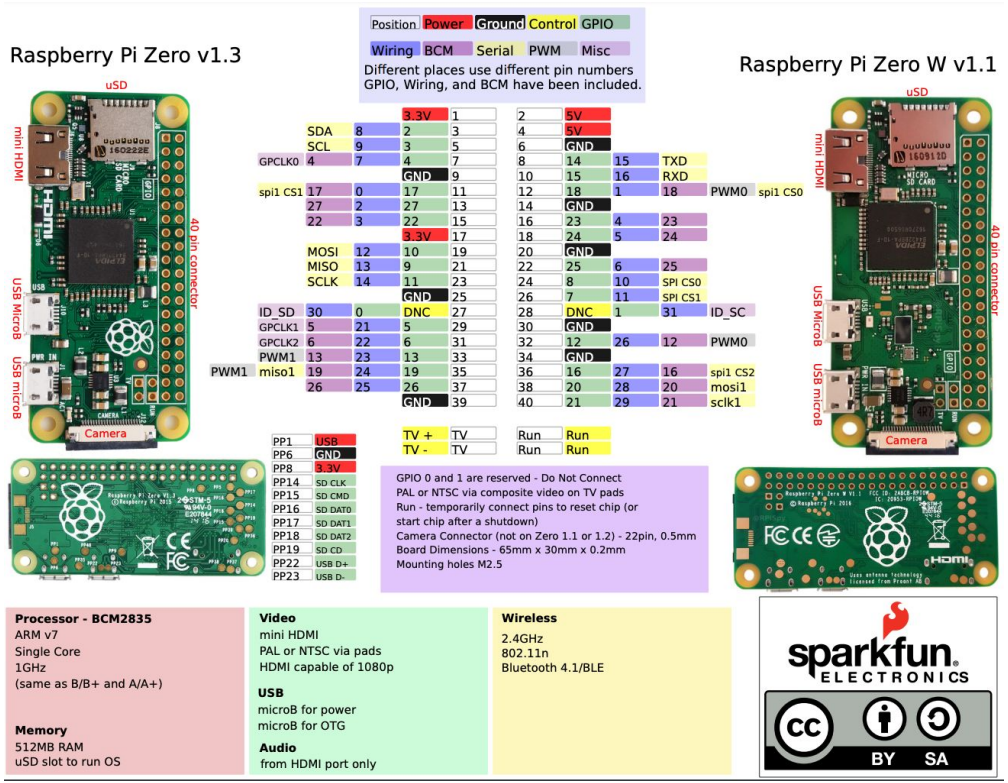
```
debian@beaglebone:~$ cd /sys/class/gpio
debian@beaglebone:/sys/class/gpio$ ls
export   gpio114  gpio15  gpio3   gpio36  gpio46  gpio60  gpio68  gpio74  gpio80  gpiochip0
gpio10   gpio115  gpio2   gpio30  gpio37  gpio47  gpio61  gpio69  gpio75  gpio81  gpiochip32
gpio11   gpio116  gpio20  gpio31  gpio38  gpio48  gpio62  gpio7   gpio76  gpio86  gpiochip64
gpio110  gpio117  gpio22  gpio32  gpio39  gpio49  gpio63  gpio70  gpio77  gpio87  gpiochip96
gpio111  gpio12   gpio23  gpio33  gpio4   gpio5   gpio65  gpio71  gpio78  gpio88  unexport
gpio112  gpio13   gpio26  gpio34  gpio44  gpio50  gpio66  gpio72  gpio79  gpio89
gpio113  gpio14   gpio27  gpio35  gpio45  gpio51  gpio67  gpio73  gpio8   gpio9
```

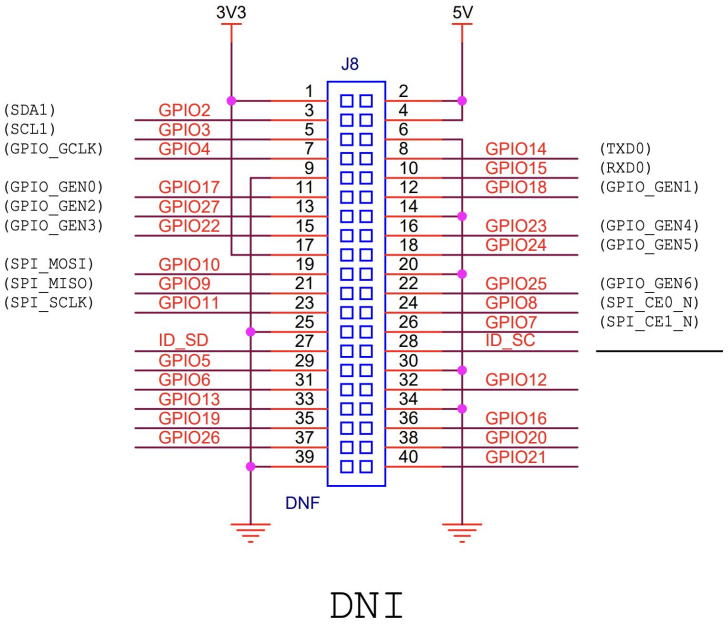
Where In The World Is gpio6?



Raspberry Pi Zero W schematic, from Raspberry Pi. URL: https://datasheets.raspberrypi.com/rpizero/raspberry-pi-zero-w-reduced-schematics.pdf

Where In The World Is gpio6?





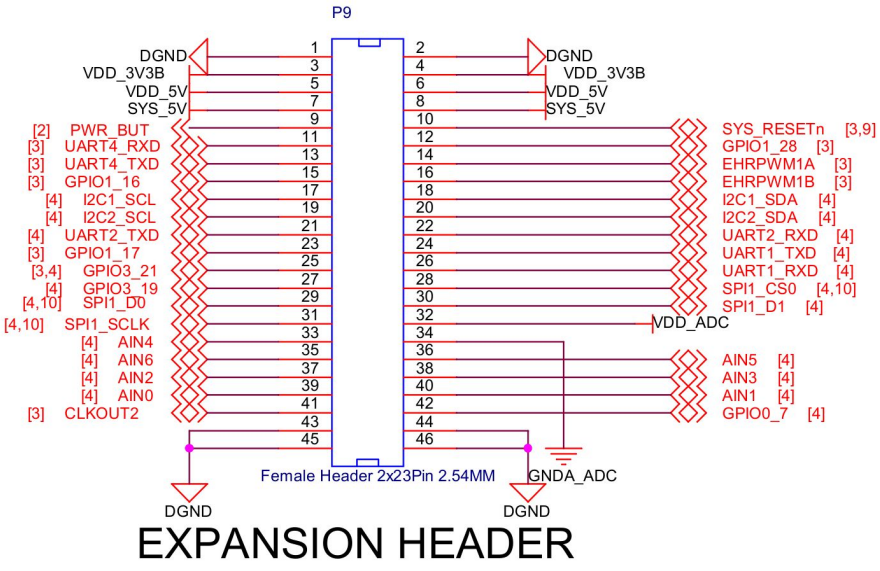
Left diagram from SparkFun. License: CC BY-SA (link). No changes made. URL: https://cdn.sparkfun.com/assets/learn_tutorials/6/7/6/PiZero_1.pdf
Right Raspberry Pi Zero W schematic, from Raspberry Pi. URL: https://datasheets.raspberrypi.com/rpizero/raspberry-pi-zero-w-reduced-schematics.pdf

Note: Mapping Between Two GPIO Naming Systems For BeagleBone Black

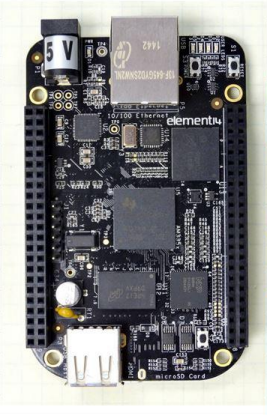
= (GPIO_CHIP*32) + GPIO_PIN

This is because each GPIO chip (controller) has 32 pins (lines)

E.g., for **GPIO1_28**: **GPIO_60** = (1*32) + 28



Beaglebone Black Pinout Diagram



P9

Function	Physical Pins		Function
DGND	1	2	DGND
VDD 3.3 V	3	4	VDD 3.3 V
VDD 5V	5	6	VDD 5V
SYS 5V	7	8	SYS 5V
PWR_BUT	9	10	SYS_RESET
UART4_RXD	11	12	GPIO_60
UART4_TXD	13	14	EHRPWM1A
GPIO_48	15	16	EHRPWM1B
SPIO_CSO	17	18	SPIO_D1
I2C2_SCL	19	20	I2C_SDA
SPIO_DO	21	22	SPIO_SLCK
GPIO_49	23	24	UART1_TXD
GPIO_117	25	26	UART1_RXD
GPIO_115	27	28	SP11_CSO
SP11_DO	29	30	GPIO_112
SP11_SCLK	31	32	VDD_ADC
AIN4	33	34	GND_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	ECAPWMO
DGND	43	44	DGND
DGND	45	46	DGND

LEGEND
Power, Ground, Reset
Digital Pins
PWM Output
1.8 Volt Analog Inputs
Shared I2C Bus
Reconfigurable Digital

P8

Function	Physical Pins		Function
DGND	1	2	DGND
MMC1_DAT6	3	4	MMC1_DAT7
MMC1_DAT2	5	6	MMC1_DAT3
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
EHRPWM2B	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
EHRPWM2A	19	20	MMC1_CMD
MMC1_CLK	21	22	MMC1_DAT5
MMC1_DAT4	23	24	MMC1_DAT1
MMC1_DAT0	25	26	GPIO_61
LCD_VSYNC	27	28	LCD_PCLK
LCD_HSYNC	29	30	LCD_AC_BIAS
LCD_DATA14	31	32	LCD_DATA15
LCD_DATA13	33	34	LCD_DATA11
LCD_DATA12	35	36	LCD_DATA10
LCD_DATA8	37	38	LCD_DATA9
LCD_DATA6	39	40	LCD_DATA7
LCD_DATA4	41	42	LCD_DATA5
LCD_DATA2	43	44	LCD_DATA3
LCD_DATA0	45	46	LCD_DATA1

BeagleBone Black Schematic. URL:
https://github.com/beagleboard/beaglebone-black/blob/master/BBB_SCH.pdf

"PINOUT FOR BEAGLEBONE BLACK".
URL: https://toptechboy.com/beaglebone-black-lesson-1-understanding-beaglebone-black-pinout/beaglebone-black-pinout/

Example: sysfs

```
debian@beaglebone:~$ cd /sys/class/gpio
debian@beaglebone:/sys/class/gpio$ ls
export   gpio114  gpio15  gpio3   gpio36  gpio46  gpio60  gpio68  gpio74  gpio80  gpiochip0
gpio10   gpio115  gpio2   gpio30  gpio37  gpio47  gpio61  gpio69  gpio75  gpio81  gpiochip32
gpio11   gpio116  gpio20  gpio31  gpio38  gpio48  gpio62  gpio7   gpio76  gpio86  gpiochip64
gpio110  gpio117  gpio22  gpio32  gpio39  gpio49  gpio63  gpio70  gpio77  gpio87  gpiochip96
gpio111  gpio12   gpio23  gpio33  gpio4   gpio5   gpio65  gpio71  gpio78  gpio88  unexport
gpio112  gpio13   gpio26  gpio34  gpio44  gpio50  gpio66  gpio72  gpio79  gpio89
gpio113  gpio14   gpio27  gpio35  gpio45  gpio51  gpio67  gpio73  gpio8   gpio9
debian@beaglebone:/sys/class/gpio$ ls gpio45
active_low  device  direction  edge  label  power  subsystem  uevent  value
debian@beaglebone:/sys/class/gpio$ cat gpio45/direction
in
debian@beaglebone:/sys/class/gpio$ echo out > gpio45/direction
debian@beaglebone:/sys/class/gpio$ cat gpio45/direction
out
debian@beaglebone:/sys/class/gpio$ echo 1 > gpio45/value
```

Accessing GPIO

- **Kernel space**
  - `gpio`
  - `gpiod`
- **User space**
  - `sysfs`
  - `Char dev` [Kernel 4.8]
    - `libgpiod`
      - Command line & C program

User Space: `libgpiod`

- Char device instead sysfs
- Gpiochips (GPIO controllers)
  - For BBB, each chip has 32 lines/pins
  - Look at **`/dev`**
    - **`ls -al /dev/gpio*`**
- Can’t use echo/cat
- d stands for device, not daemon

libgpiod - Command Line

Descriptions copied from: https://git.kernel.org/pub/scm/libs/libgpiod/libgpiod.git/about/

- **gpiodetect** - list all gpiochips present on the system, their names, labels and number of GPIO lines
- **gpioinfo** - list all lines of specified gpiochips, their names, consumers, direction, active state and additional flag
- **gpioget** - read values of specified GPIO lines
- **gpioset** - set values of specified GPIO lines, potentially keep the lines exported and wait until timeout, user input or signal
- **gpiofind** - find the gpiochip name and line offset given the line name
- **gpiomon** - wait for events on GPIO lines, specify which events to watch, how many events to process before exiting or if the events should be reported to the console

`libgpiod` - C Library (C++, Python, & More Supported)

Core functions (text below copied & modified from ISC - see reference below):

- Open GPIO chip via a **`gpiod_chip_open`** function
  - **`gpiod_chip_open_by_name`**()
  - Returns a **`gpiod_chip struct`** which used by subsequent API calls
- Open GPIO line(s)
  - **`gpiod_chip_get_line()`**
  - **`gpiod_chip_get_lines()`**
  - Returns **`gpiod_line struct`**
- Request use of line as input/output
  - **`gpiod_line_request_input()`**
  - **`gpiod_line_request_output()`**
- Read value of input
  - **`gpiod_line_get_value()`**
- Set output
  - **`gpiod_line_set_value()`**

“GPIO Programming: Exploring the libgpiod Library”. ICS. URL: https://www.ics.com/blog/gpio-programming-exploring-libgpiod-library

`libgpiod` - Command Line

In-Class Demo

`libgpiod` - C Library (C++, Python, & More Supported) (2)

Core functions (text below copied & modified from ISC - see reference below):

- Release line(s)
  - **`gpiod_line_release()`**
- Release chip(s)
  - **`gpiod_chip_close()`**
- Other APIs for more advanced functions
  - Setting pin modes for pullup/pulldown resistors
  - Defining callback function for when an event occurs (e.g., level of an input pin changing)

```
#include <gpiod.h>
```


“GPIO Programming: Exploring the libgpiod Library”. ICS. URL: https://www.ics.com/blog/gpio-programming-exploring-libgpiod-library

libgpiod - A Workflow (Modified from ICS - See Reference Below)

```
*chipname = "gpiochipx";
chip = gpiod_chip_open_by_name(chipname);
line = gpiod_chip_get_line(chip, v);
gpiod_line_request_output(line, "something", z);
gpiod_line_set_value(line, t)
gpiod_line_set_value(line, y)
gpiod_line_release(line);
gpiod_chip_close(chip);
```


“GPIO Programming: Exploring the libgpiod Library”. ICS. URL: https://www.ics.com/blog/gpio-programming-exploring-libgpiod-library

User Space: `libgpiod` vs. `sysfs`

- sysfs (deprecated) straightforward to use from shell
  - Not so much for C and Python
  - Latency
  - Lacking pullup/down feature
- libgpiod to the rescue
  - Faster (adafruit: 400Khz pin toggling on Pi 3 via C)
  - Easy functions in C, Python, and other languages

Accessing GPIO

- **Kernel space**
  - `gpio`
  - `gpiod`
- **User space**
  - `sysfs`
  - `Char dev` [Kernel 4.8]
    - `libgpiod`
      - Command line & C program

Kernel Space: `gpio`

- **`gpio_set_value(pin, value)`**
- **`linux/gpio.h`**
- (In class) Make it so segment of display toggles on and off in **`device_write`** and **`device_read`** (respectively) of module by using **`gpio_set_value()`**
- Deprecated

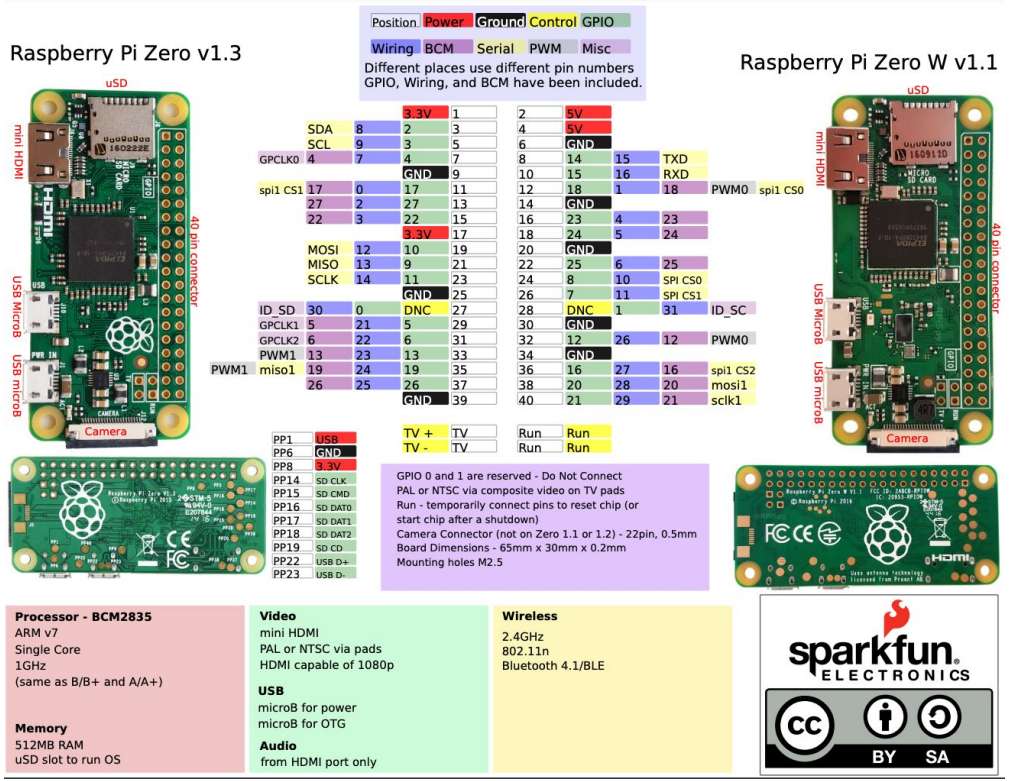
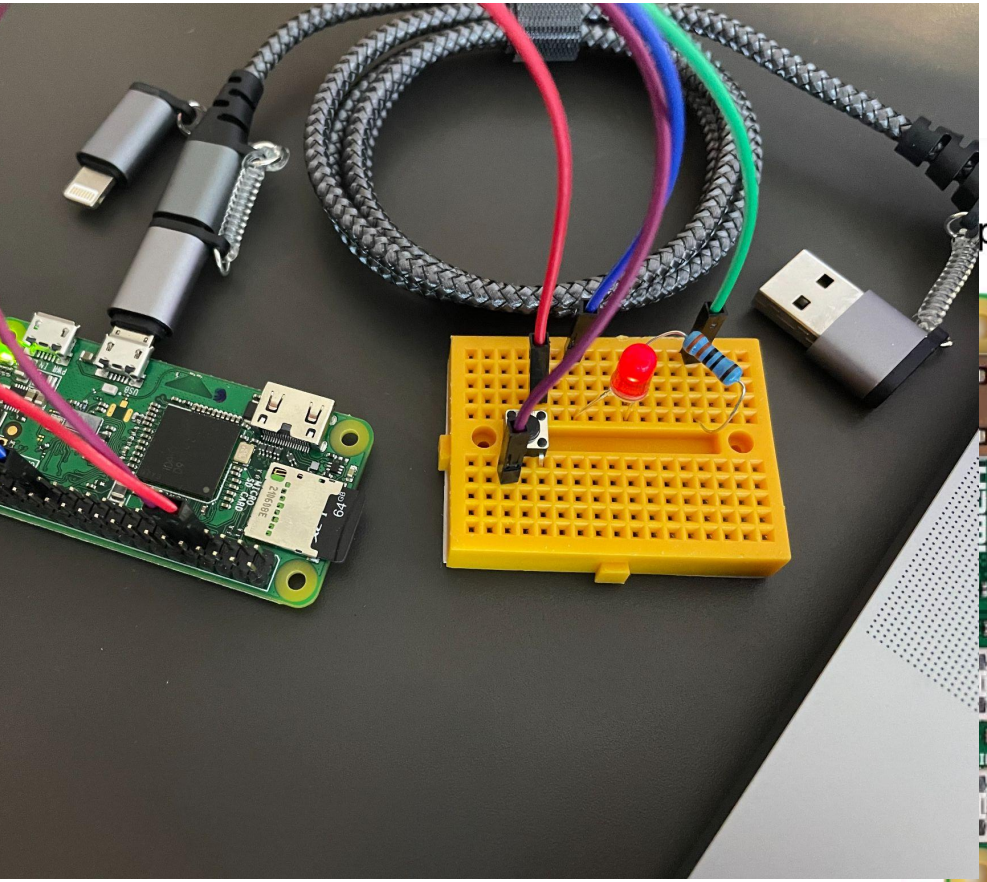


Diagram from SparkFun. License: CC BY-SA (link). No changes made. URL: https://cdn.sparkfun.com/assets/learn_tutorials/6/7/6/PiZero_1.pdf



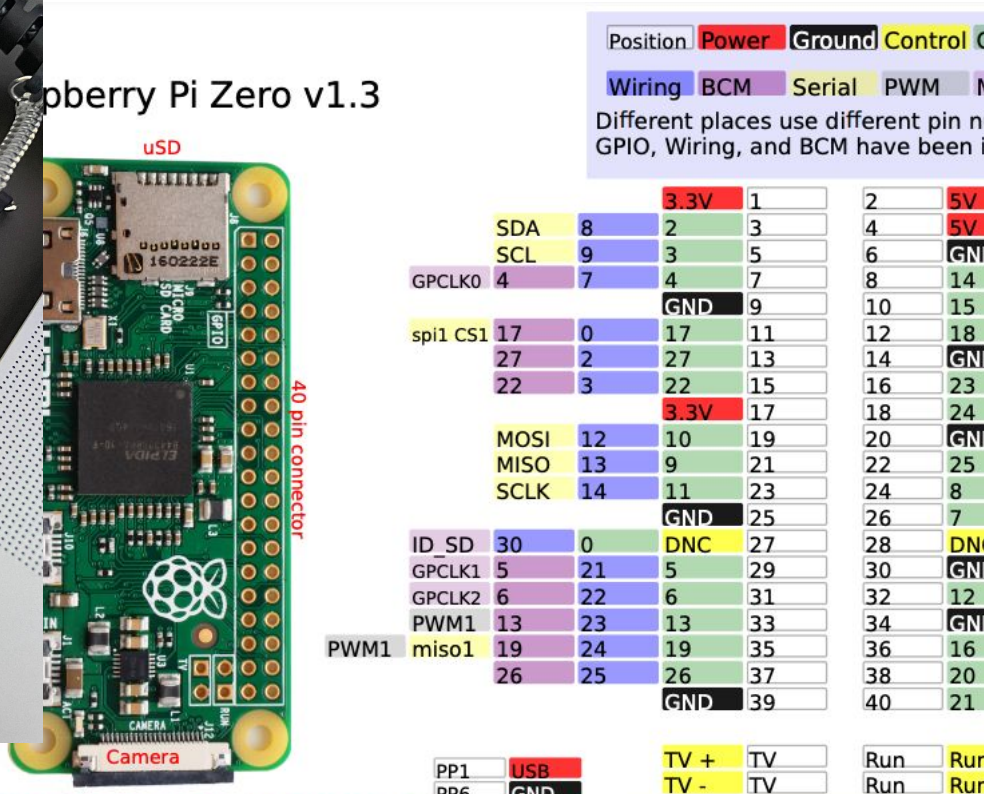


Diagram from SparkFun. License: CC BY-SA (link). No changes made. URL: https://cdn.sparkfun.com/assets/learn_tutorials/6/7/6/PiZero_1.pdf

Kernel Space: `gpio` - Interrupts

```c
// From: Johannes4Linux
// https://github.com/Johannes4Linux/Linux_Driver_Tutorial/blob/main/11_gpio_irq/gpio_irq.c

#include <linux/module.h>
#include <linux/init.h>
#include <linux/gpio.h>
#include <linux/interrupt.h>

/* Meta Information */
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Johannes 4 GNU/Linux");
MODULE_DESCRIPTION("A simple LKM for a gpio interrupt");

/** variable contains pin number o interrupt controller to which GPIO 17 is mapped to */
unsigned int irq_number;

/**
 * @brief Interrupt service routine is called, when interrupt is triggered
 */
static irq_handler_t gpio_irq_handler(unsigned int irq, void *dev_id, struct pt_regs *regs) {
   printk("gpio_irq: Interrupt was triggered and ISR was called!\n");
   return (irq_handler_t) IRQ_HANDLED;
}
```

Kernel Space: `gpio` - Interrupts (continued)

```
1  // @brief This function is called, when the module is loaded into the kernel
2  static int __init ModuleInit(void) {
3      printk("qpio_irq: Loading module... ");
4
5      /* Setup the gpio */
6      if(gpio_request(17, "rpi-gpio-17")) {
7             printk("Error!\nCan not allocate GPIO 17\n");
8             return -1;
9      }
10
11     /* Set GPIO 17 direction */
12     if(gpio_direction_input(17)) {
13            printk("Error!\nCan not set GPIO 17 to input!\n");
14            gpio_free(17);
15            return -1;
16     }
```

Kernel Space: gpio - Interrupts (continued - 2)

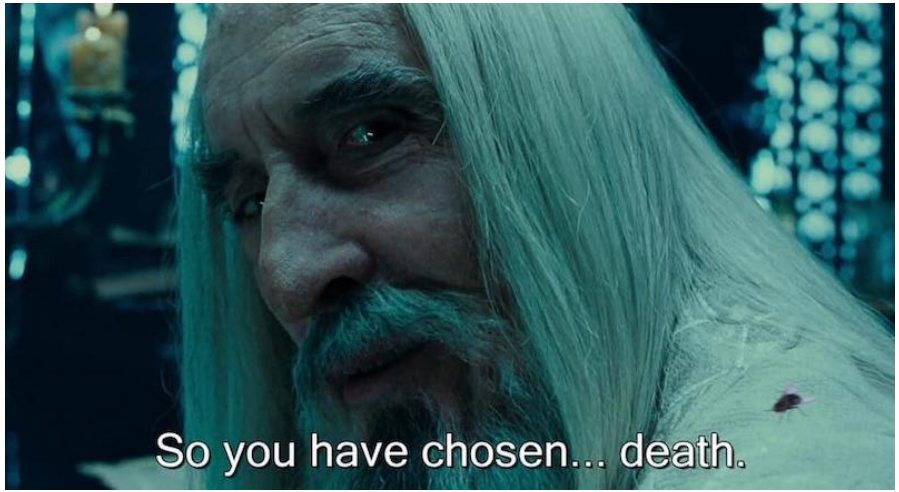
```
    /* Setup the interrupt */
    irq_number = gpio_to_irq(17);

    if(request_irq(irq_number, (irq_handler_t) gpio_irq_handler, IRQF_TRIGGER_RISING, "my_gpio_irq", NULL) != 0){
            printk("Error!\nCan not request interrupt nr.: %d\n", irq_number);
            gpio_free(17);
            return -1;
    }

    printk("Done!\n");
    printk("GPIO 17 is mapped to IRQ Nr.: %d\n", irq_number);
    return 0;
}

// @brief This function is called, when the module is removed from the kernel
static void __exit ModuleExit(void) {
    printk("gpio_irq: Unloading module... ");
    free_irq(irq_number, NULL);
    gpio_free(17);
}
```

Kernel Space: `gpio` - Interrupts (continued - 3)

```
1 module_init(ModuleInit);
2 module_exit(ModuleExit);
```

Accessing GPIO

- **Kernel space**
  - `gpio`
  - **`gpiod`**
- **User space**
  - `sysfs`
  - `Char dev` [Kernel 4.8]
    - **`libgpiod`**
      - Command line & C program



Meme from *The Lord of the Rings: The Fellowship of the Ring (2001)* [New Line Cinema and WingNut Films]. Found on Meming Wiki, URL: https://en.meming.world/wiki/File:So_you_have_chosen..._death..jpg/

Kernel Space: `gpio`

- Old functions (**`gpio`**) were **gpio_*** (**`gpio.h`**) - link to code shown in Bootlin Elixir screenshot below. Deprecated now.

/ include / linux / gpio.h All syml Search Identifier

```
1  /* SPDX-License-Identifier: GPL-2.0 */
2  /*
3   * <linux/gpio.h>
4   *
5   * This is the LEGACY GPIO bulk include file, including legacy APIs. It is
6   * used for GPIO drivers still referencing the global GPIO numberspace,
7   * and should not be included in new code.
8   *
9   * If you're implementing a GPIO driver, only include <linux/gpio/driver.h>
10  * If you're implementing a GPIO consumer, only include <linux/gpio/consumer.h>
11  */
12 #ifndef __LINUX_GPIO_H
13 #define __LINUX_GPIO_H
14
15 #include <linux/errno.h>
16
17 /* see Documentation/driver-api/gpio/legacy.rst */
18
```

Kernel Space: gpiod

- New functions (**gpiod**) are **gpiod_*** (**consumer.h**) - link to code shown in Bootlin Elixir screenshot below. Deprecated now.


```
64  /* Acquire and dispose GPIOs */
65  struct gpio_desc *__must_check gpiod_get(struct device *dev,
66                                           const char *con_id,
67                                           enum gpiod_flags flags);
68  struct gpio_desc *__must_check gpiod_get_index(struct device *dev,
69                                                 const char *con_id,
70                                                 unsigned int idx,
71                                                 enum gpiod_flags flags);
72  struct gpio_desc *__must_check gpiod_get_optional(struct device *dev,
73                                                    const char *con_id,
74                                                    enum gpiod_flags flags);
75  struct gpio_desc *__must_check gpiod_get_index_optional(struct device *dev,
76                                                          const char *con_id,
77                                                          unsigned int index,
78                                                          enum gpiod_flags flags);
79  struct gpio_descs *__must_check gpiod_get_array(struct device *dev,
80                                                  const char *con_id,
81                                                  enum gpiod_flags flags);
82  struct gpio_descs *__must_check gpiod_get_array_optional(struct device *dev,
83                                                           const char *con_id,
84                                                           enum gpiod_flags flags);
85  void gpiod_put(struct gpio_desc *desc);
86  void gpiod_put_array(struct gpio_descs *descs);
87
88  struct gpio_desc *__must_check devm_gpiod_get(struct device *dev,
89                                                const char *con_id,
```

What Does `gpiod` Solve?

Text from this commit on kernel.org (2013):

This patch exports the gpiod_* family of API functions, a safer alternative to the legacy GPIO interface. Differences between the gpiod and legacy gpio APIs are:

- **gpio works with integers**, whereas **gpiod operates on opaque handlers** which cannot be forged or used before proper acquisition
- gpiod get/set functions are **aware of the active low state of a GPIO**
- gpio consumers should now include **<linux/gpio/consumer.h>** to access the new interface, whereas chips drivers will use <linux/gpio/driver.h>

The legacy gpio API is now built as inline functions on top of gpiod.

Signed-off-by: Alexandre Courbot <acourbot@nvidia.com>
Signed-off-by: Linus Walleij <linus.walleij@linaro.org>

Implementing `gpiod` Must Be Fun Right?

Click for spooky content

This requires knowledge of the device tree…

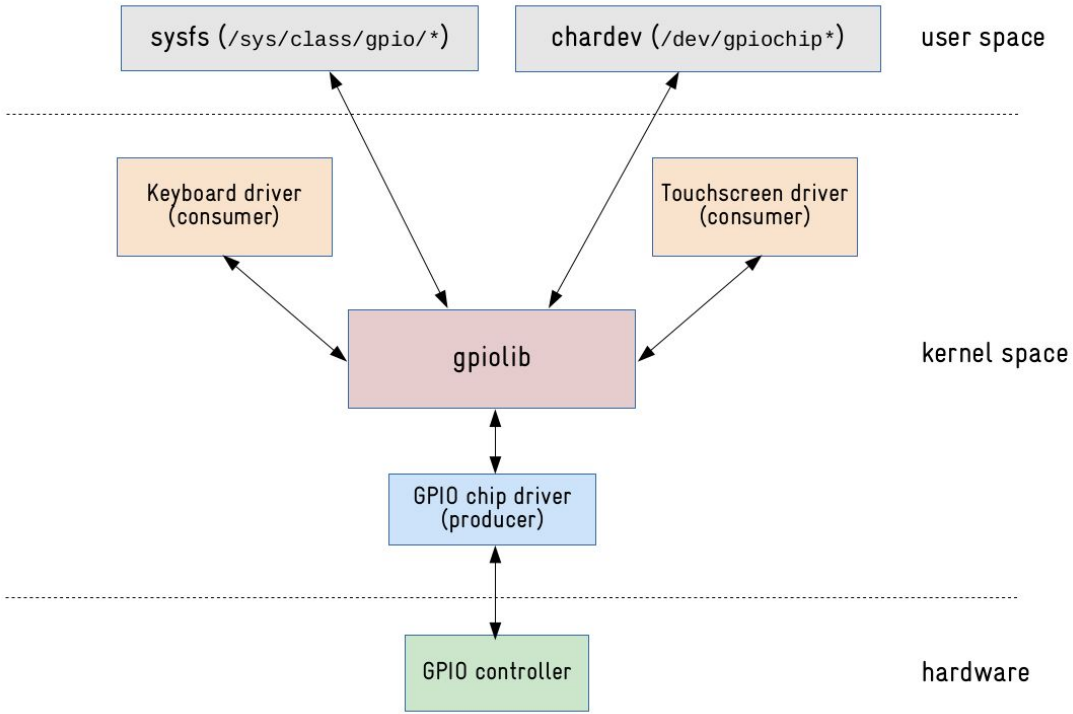
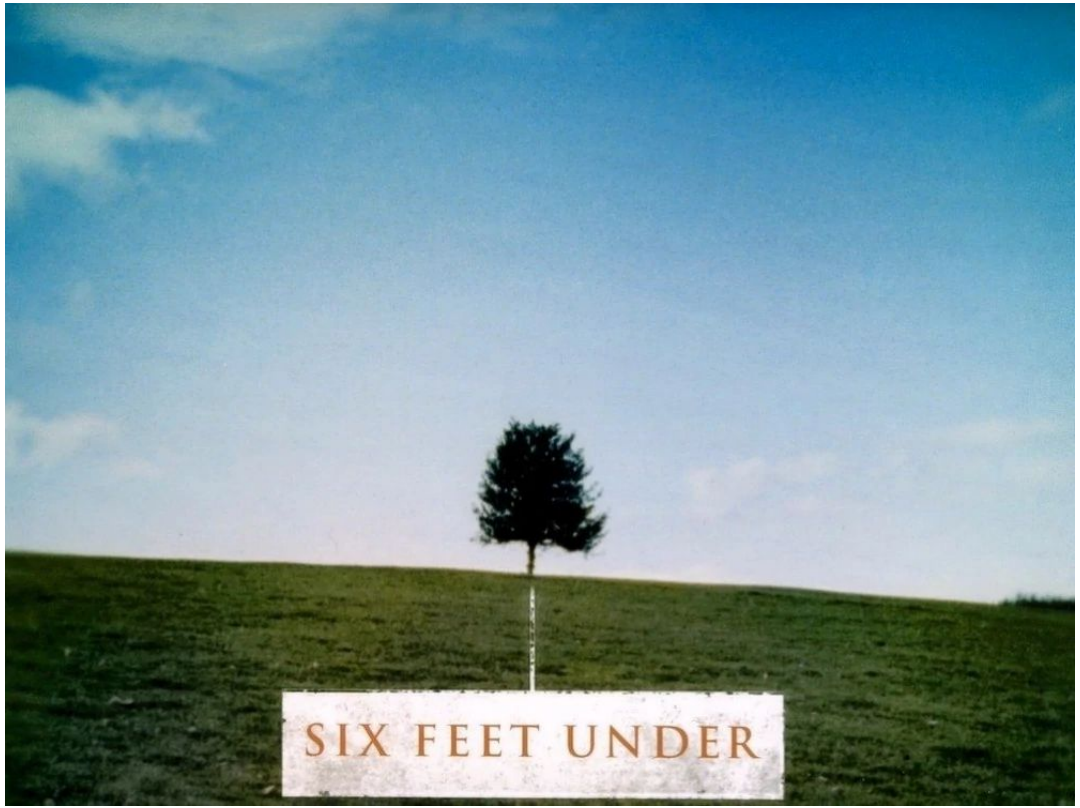


Diagram: “Linux kernel GPIO user space interface”. #embeddedbits. URL: https://embeddedbits.org/new-linux-kernel-gpio-user-space-interface/

The Device Tree



From HBO's *Six Feet Under*. URL: https://sixfeetunder.fandom.com/wiki/Six_Feet_Under?file=Six_feet_under.jpg

The Device Tree

“The **devicetree** is a **data structure** for **describing hardware**. Rather than hard coding every detail of a device into an operating system, many aspects of the hardware can be described in a data structure that is **passed to the operating system** at boot time.”

- https://www.devicetree.org

“can now configure his platform without having to recompile the kernel” - https://docs.google.com/document/u/1/d/17P54kZkZO_-JtTjrFuVz-Cp_RMMg7GB_8W9JK9sLKfA/pub

Did you ever hear the tragedy of Board Files the Unwise?

From *Star Wars: Episode III – Revenge of the Sith* (2005). URL: https://starwars.fandom.com/wiki/The_Tragedy_of_Darth_Plagueis_the_Wise

Current BeagleBone Linux source code here

Board Files

- In kernel 3.2
  - `arch/arm/mach-omap2/board-am335xevm.c`
  - More than 4,000 lines of code
  - Get a sense of this by looking at this file
  - Good luck getting the kernel to compile with this mess
    - Consider capes

https://imgflip.com/memetemplate/294137091/Man-behind-man

What Is The Motivation For Device Tree Appearing In Linux?

From Linus Torvalds <>
Date Thu, 17 Mar 2011 19:50:36 -0700
Subject Re: [GIT PULL] omap changes for v2.6.39 merge window

On Thu, Mar 17, 2011 at 11:30 AM, Tony Lindgren <tony@atomide.com> wrote:
>
> Please pull omap changes for this merge window from:

Excerpts from Linus' response (modified for language):

Gaah. Guys, this whole ARM thing is a x'ing pain in the x.

You need to stop stepping on each others toes. There is no way that your changes to those crazy clock-data files should constantly result in those annoying conflicts, just because different people in different ARM trees do some x renaming of some random device. Seriously.

That usb_musb_init() thing in arch/arm/mach-omap2/usb-musb.c also seems to be totally insane. I wonder what kind of insanity I'm missing just because I don't happen to see the merge conflicts, just because people were lucky enough to happen to not touch the same file within a few lines.

Somebody needs to get a grip in the ARM community. I do want to do these merges, just to see how screwed up things are, but guys, this is just ridiculous.

And stop the crazy renaming already! Just leave it off. Don't rename boards and drivers "just because", at least not when there clearly are clashes.

Somebody in the ARM community really needs to step up and tell people to stop x'ing around.

Enough Is Enough: Device Tree

- To be continued