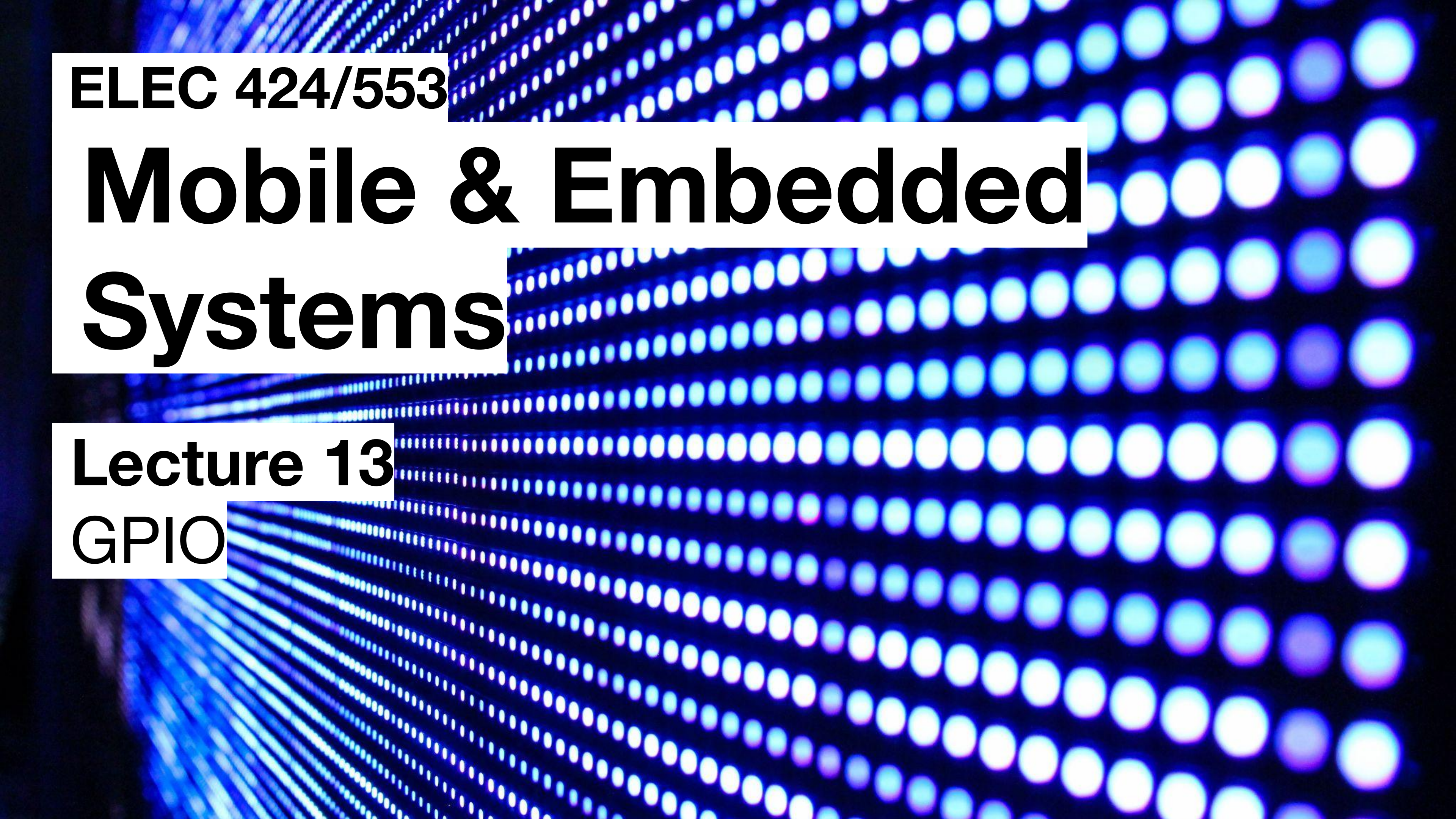


ELEC 424/553

Mobile & Embedded Systems

Lecture 13

GPIO

In Past Recent News

ars TECHNICA

RISC-Y BUSINESS —

A history of ARM, part 1: Building the first chip

In 1983, Acorn Computers needed a CPU. So 10 people built one.

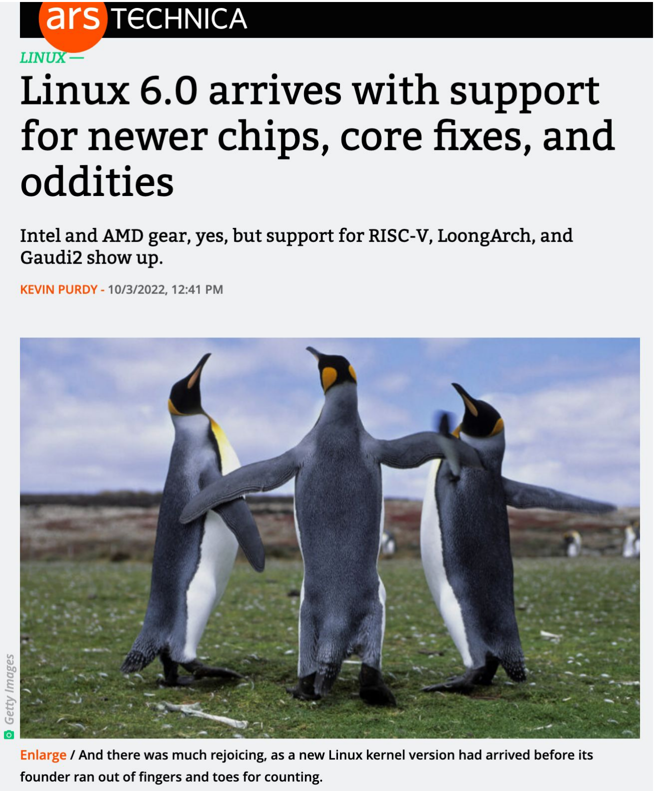
JEREMY REIMER - 9/23/2022, 10:47 AM

Aurich Lawson / Getty Images

Enlarge

Ars Technica. URL: https://arstechnica.com/gadgets/2022/09/a-history-of-arm-part-1-building-the-first-chip/

In Past Recent News



LINUX —

Linux 6.0 arrives with support for newer chips, core fixes, and oddities

Intel and AMD gear, yes, but support for RISC-V, LoongArch, and Gaudi2 show up.

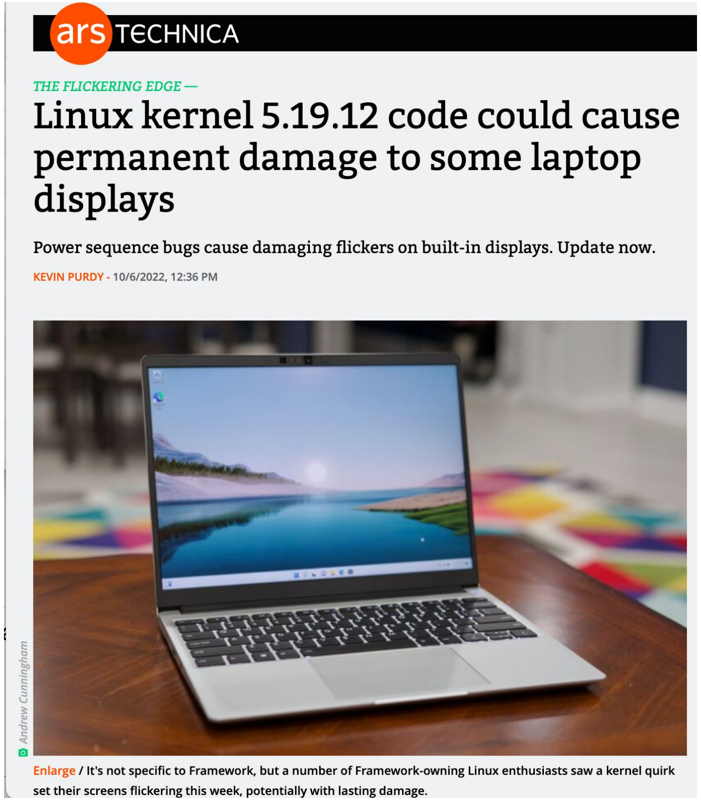
KEVIN PURDY - 10/3/2022, 12:41 PM

Getty Images

Enlarge / And there was much rejoicing, as a new Linux kernel version had arrived before its founder ran out of fingers and toes for counting.

Ars Technica. URL: https://arstechnica.com/gadgets/2022/10/linux-6-0-arrives-with-support-for-newer-chips-core-fixes-and-oddities/

In Past Recent News



ars TECHNICA

THE FLICKERING EDGE —

Linux kernel 5.19.12 code could cause permanent damage to some laptop displays

Power sequence bugs cause damaging flickers on built-in displays. Update now.

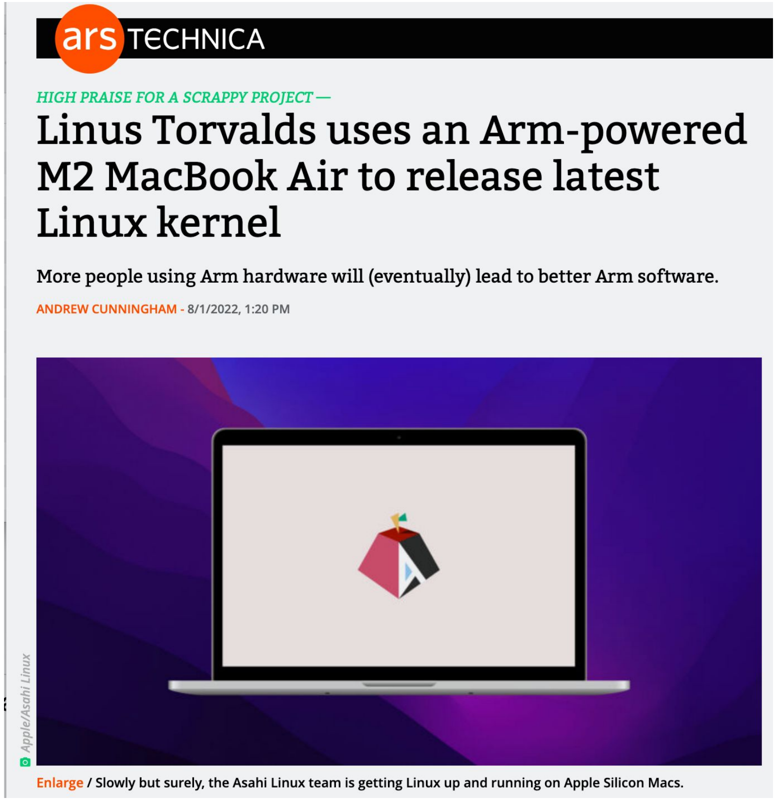
KEVIN PURDY - 10/6/2022, 12:36 PM

Andrew Cunningham

Enlarge / It's not specific to Framework, but a number of Framework-owning Linux enthusiasts saw a kernel quirk set their screens flickering this week, potentially with lasting damage.

Ars Technica. URL: https://arstechnica.com/gadgets/2022/10/linux-5-19-2-kernel-can-flicker-and-damage-screens-on-some-intel-gpu-laptops/

In Past Recent News



ars TECHNICA

HIGH PRAISE FOR A SCRAPPY PROJECT —

Linus Torvalds uses an Arm-powered M2 MacBook Air to release latest Linux kernel

More people using Arm hardware will (eventually) lead to better Arm software.

ANDREW CUNNINGHAM - 8/1/2022, 1:20 PM

Apple/Asahi Linux

Enlarge / Slowly but surely, the Asahi Linux team is getting Linux up and running on Apple Silicon Macs.

Ars Technica. URL: https://arstechnica.com/gadgets/2022/08/linus-torvalds-uses-an-arm-powered-m2-macbook-air-to-release-latest-linux-kernel/

The Linux Device Model

- 2.6 kernel (Dec. 2003) introduced **unified device model**
  - "Linux Device Model" (another resource)
- Enabled easy view of devices & device hierarchy
- Driver & device association (both ways)
- Cluster devices according to class
  - E.g. "input device"
- But really, why?
  - Power management: Need to know what to shut off first
  - USB Mouse -> USB Controller -> PCI Bus

Robert Love, *Linux Kernel Development*, 3rd Edition

`dev` and `sysfs`

- `dev`
  - `/dev`
  - Focused on accessing devices
- `sysfs`
  - `/sys`
  - **Virtual file system**
  - Files realized on demand
  - In-memory
  - Focused on device management
  - Way for user to view & modify **kernel objects**
  - User view of Linux Device Model (another resource) - see later slide
- UNIX philosophy: “Everything is a file”

https://www.facebook.com/itsfoss/photos/well-everything-is-actually-a-file-in-unix-/1337947193012711/

/sys Directories (text below copied from reference at bottom)

“The kernel provides a representation of its model in userspace through the sysfs virtual file system. It is usually mounted in the /sys directory and contains the following subdirectories:

- **block** - all block devices available in the system (disks, partitions)
- **bus** - types of bus to which physical devices are connected (pci, ide, usb)
- **class** - drivers classes that are available in the system (net, sound, usb)
- **devices** - the hierarchical structure of devices connected to the system
- **firmware** - information from system firmware (ACPI)
- **fs** - information about mounted file systems
- **kernel** - kernel status information (logged-in users, hotplug)
- **module** - the list of modules currently loaded
- **power** - information related to the power management subsystem”

“Linux Kernel Teaching”. URL: https://linux-kernel-labs.github.io/refs/heads/master/labs/device_model.html

`/sys` (sysfs)

- Device list & information
- User space sees devices via **sysfs**
  - **udev** uses sysfs
  - “Kernel events” trigger **udev** when devices inserted/detached
- Parameters for modules visible in sysfs
- **`ls /sys/module/hello`**
  - **`parameters`** folder will be there
  - Can modify parameters
  - *In class demonstration as reminder*

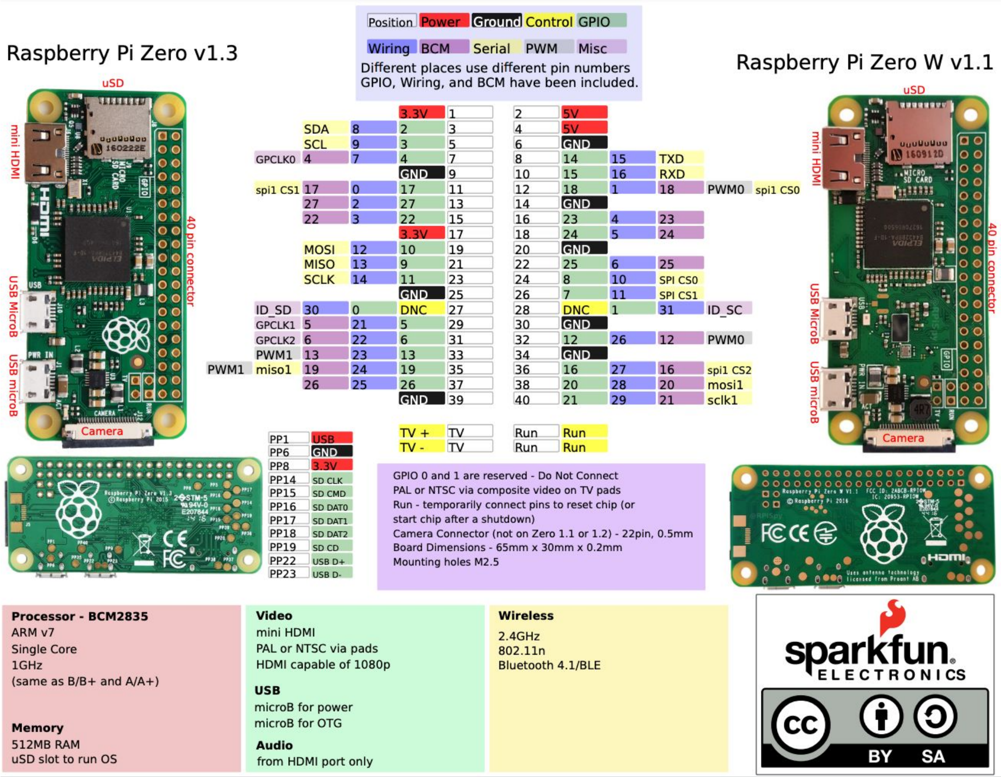
Accessing GPIO

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

Accessing GPIO

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

GPIO: General-Purpose Input/Output



Position Power Ground Control GPIO

Wiring BCM Serial PWM Misc

Different places use different pin numbers GPIO, Wiring, and BCM have been included.

				3.3V	1	2	5V				
		SDA	8	2	3	4	5V				
		SCL	9	3	5	6	GND				
	GPCLK0	4	7	4	7	8	14	15	TXD		
				GND	9	10	15	16	RXD		
	spi1 CS1	17	0	17	11	12	18	1	18	PWM0	spi1 CS0
		27	2	27	13	14	GND				
		22	3	22	15	16	23	4	23		
				3.3V	17	18	24	5	24		
		MOSI	12	10	19	20	GND				
		MISO	13	9	21	22	25	6	25		
		SCLK	14	11	23	24	8	10	SPI CS0		
				GND	25	26	7	11	SPI CS1		
	ID_SD	30	0	DNC	27	28	DNC	1	31	ID_SC	
	GPCLK1	5	21	5	29	30	GND				
	GPCLK2	6	22	6	31	32	12	26	12	PWM0	
	PWM1	13	23	13	33	34	GND				
PWM1	miso1	19	24	19	35	36	16	27	16	spi1 CS2	
		26	25	26	37	38	20	28	20	mosi1	
				GND	39	40	21	29	21	sclk1	

TV +	TV	Run	Run
TV -	TV	Run	Run

PP1	USB
PP6	GND
PP8	3.3V
PP14	SD CLK
PP15	SD CMD
PP16	SD DAT0
PP17	SD DAT1
PP18	SD DAT2
PP19	SD CD
PP22	USB D+
PP23	USB D-

GPIO 0 and 1 are reserved - Do Not Connect
PAL or NTSC via composite video on TV pads
Run - temporarily connect pins to reset chip (or start chip after a shutdown)
Camera Connector (not on Zero 1.1 or 1.2) - 22pin, 0.5mm
Board Dimensions - 65mm x 30mm x 0.2mm
Mounting holes M2.5

Processor - BCM2835
ARM v7
Single Core
1GHz
(same as B/B+ and A/A+)

Memory
512MB RAM
uSD slot to run OS

Video
mini HDMI
PAL or NTSC via pads
HDMI capable of 1080p

USB
microB for power
microB for OTG

Audio
from HDMI port only

Wireless
2.4GHz
802.11n
Bluetooth 4.1/BLE

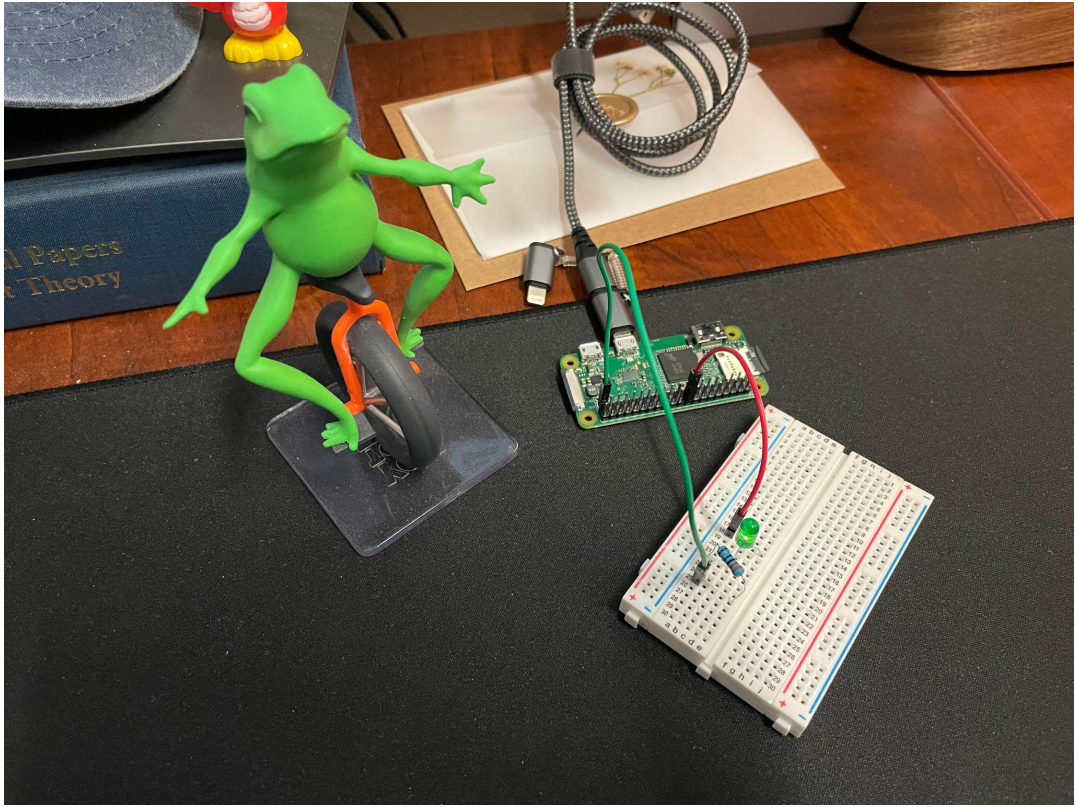
GPIO Pins
“behavior (including whether it is an input or output pin) can be controlled / programmed by the user at run time.”

“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



Example: Use `sysfs` to turn on built-in LED

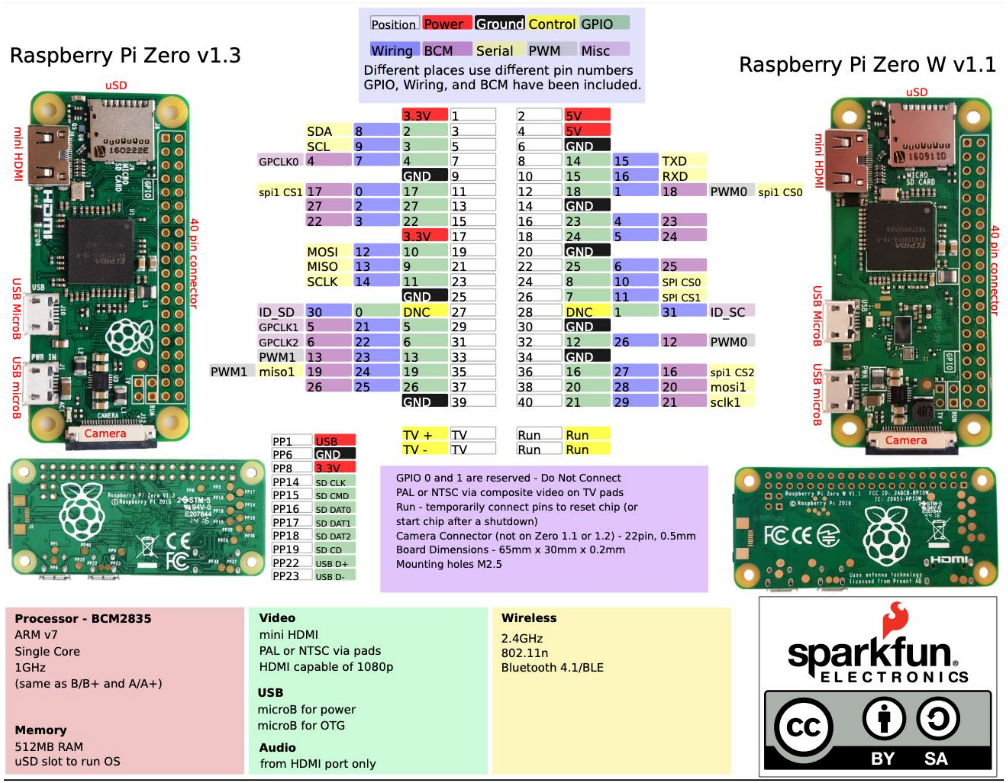
```
pi@raspberrypi:~ $ sudo su -
root@raspberrypi:~# cd /sys/class/leds/mmc0
root@raspberrypi:/sys/class/leds/mmc0# ls
brightness  max_brightness  power  subsystem  trigger  uevent

root@raspberrypi:/sys/class/leds/mmc0# cat 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

root@raspberrypi:/sys/class/leds/mmc0# echo none > trigger
root@raspberrypi:/sys/class/leds/mmc0# cat 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

root@raspberrypi:/sys/class/leds/mmc0# echo 1 > brightness
root@raspberrypi:/sys/class/leds/mmc0# echo 0 > brightness
```

Where In The World Is gpio24?



Position Power Ground Control GPIO
Wiring BCM Serial PWM Misc

Different places use different pin numbers
GPIO, Wiring, and BCM have been included.

				3.3V	1	2	5V				
		SDA	8	2	3	4	5V				
		SCL	9	3	5	6	GND				
	GPCLK0	4	7	4	7	8	14	15	TXD		
				GND	9	10	15	16	RXD		
	spi1 CS1	17	0	17	11	12	18	1	18	PWM0	spi1 CS0
		27	2	27	13	14	GND				
		22	3	22	15	16	23	4	23		
				3.3V	17	18	24	5	24		
		MOSI	12	10	19	20	GND				
		MISO	13	9	21	22	25	6	25		
		SCLK	14	11	23	24	8	10	SPI CS0		
				GND	25	26	7	11	SPI CS1		
	ID_SD	30	0	DNC	27	28	DNC	1	31	ID_SC	
	GPCLK1	5	21	5	29	30	GND				
	GPCLK2	6	22	6	31	32	12	26	12	PWM0	
	PWM1	13	23	13	33	34	GND				
PWM1	miso1	19	24	19	35	36	16	27	16	spi1 CS2	
		26	25	26	37	38	20	28	20	mosi1	
				GND	39	40	21	29	21	sclk1	

TV +	TV	Run	Run
TV -	TV	Run	Run

PP1	USB
PP6	GND
PP8	3.3V
PP14	SD CLK
PP15	SD CMD
PP16	SD DAT0
PP17	SD DAT1
PP18	SD DAT2
PP19	SD CD
PP22	USB D+
PP23	USB D-

GPIO 0 and 1 are reserved - Do Not Connect
PAL or NTSC via composite video on TV pads
Run - temporarily connect pins to reset chip (or start chip after a shutdown)
Camera Connector (not on Zero 1.1 or 1.2) - 22pin, 0.5mm
Board Dimensions - 65mm x 30mm x 0.2mm
Mounting holes M2.5

Processor - BCM2835
ARM v7
Single Core
1GHz
(same as B/B+ and A/A+)

Memory
512MB RAM
uSD slot to run OS

Video
mini HDMI
PAL or NTSC via pads
HDMI capable of 1080p

USB
microB for power
microB for OTG

Audio
from HDMI port only

Wireless
2.4GHz
802.11n
Bluetooth 4.1/BLE

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

Now Let’s Try Turning On External LED

dev and sysfs

- dev
  - /dev
  - Focused on accessing devices
- sysfs
  - /sys
  - **Virtual file system**
  - Files realized on demand
  - In-memory
  - Focused on device management
  - Way for user to view & modify **kernel objects**
  - User view of Linux Device Model (another resource) - see later slide
- UNIX philosophy: “Everything is a file”

https://www.facebook.com/itsfoss/photos/well-everything-is-actually-a-file-in-unix-/1337947193012711/

`udev` - User Space **`/dev`**

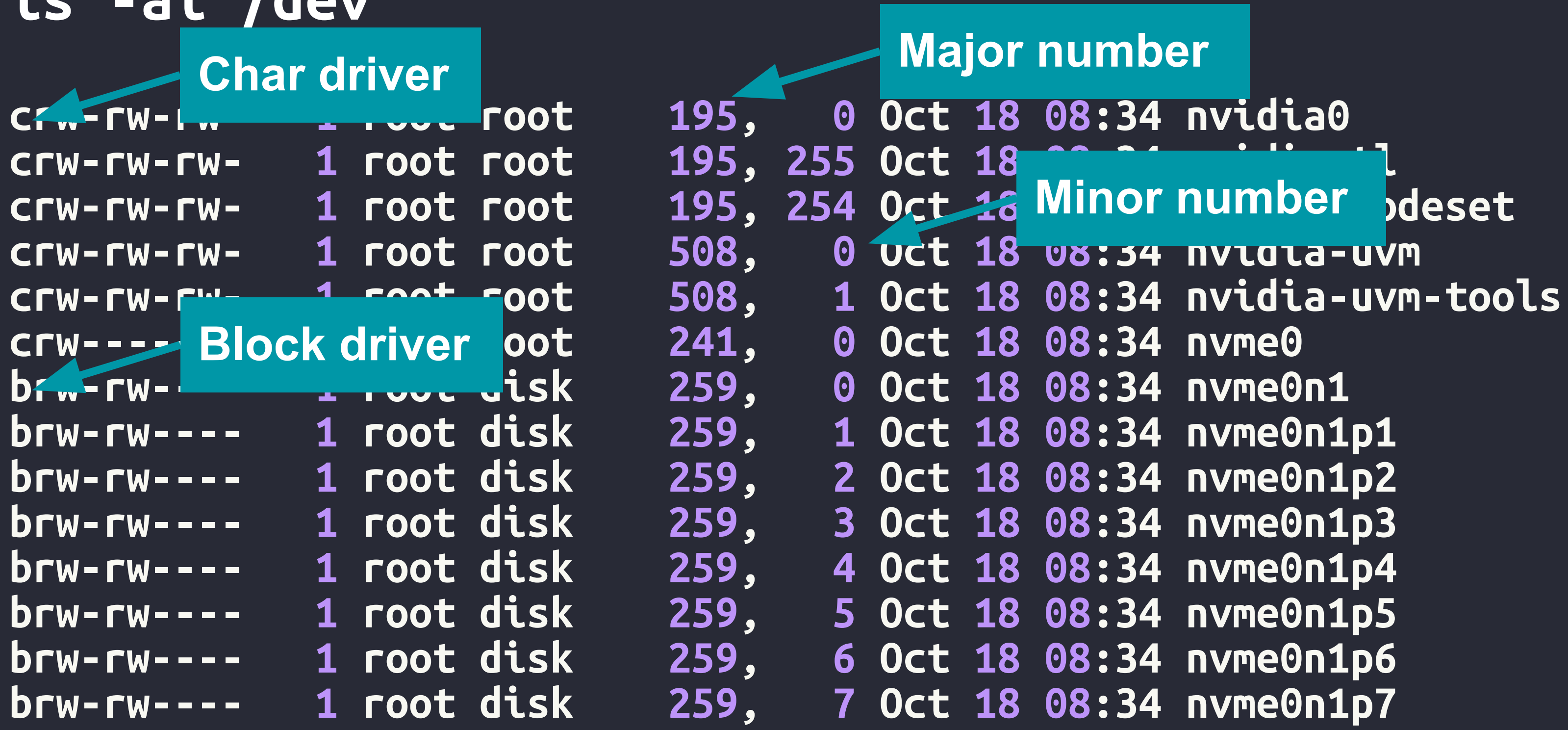
- Device manager - adds and deletes **`/dev`** device nodes
  - **`/dev`** was dead (static) before this!
  - How do you think **`/dev`** was managed before?
- Operates in user space
  - Part of systemd 💀 [can list using: `systemctl --type=service`]
- Creates device events
- Bunch of daemons
- **`udev`** scripts - link
- Device recognition
  - Serial number, Manufacturer, Vendor/Product ID

/dev

- Device access from user space
- Consider our **test.c** file

```
int main(){
    int fd;
    int ret;
    char stringToSend[256];
    printf("Warm it up.exe\n");
    fd = open("/dev/meschar", O_RDWR);    // Capital o, not zero
    printf("Do you know anything about the Chamber of Secrets?\n");
    scanf("%[^\n]%*c", stringToSend);
    ret = write(fd, stringToSend, strlen(stringToSend));
    printf("HP, I'll repeat what you said if you hit enter");
    getchar();
    ret = read(fd, receive, 256);
    printf("REPEAT OF MESSAGE: %s\n", receive);
    return 0;
}
```

ls -al /dev



Char driver

Major number

Minor number

Block driver

```
crw-rw-rw-   1 root root    195,   0 Oct 18 08:34 nvidia0
crw-rw-rw-   1 root root    195, 255 Oct 18 08:34 nvidiactl
crw-rw-rw-   1 root root    195, 254 Oct 18 08:34 nvidia-modeset
crw-rw-rw-   1 root root    508,   0 Oct 18 08:34 nvidia-uvm
crw-rw-rw-   1 root root    508,   1 Oct 18 08:34 nvidia-uvm-tools
crw-------   1 root root    241,   0 Oct 18 08:34 nvme0
brw-rw----   1 root disk    259,   0 Oct 18 08:34 nvme0n1
brw-rw----   1 root disk    259,   1 Oct 18 08:34 nvme0n1p1
brw-rw----   1 root disk    259,   2 Oct 18 08:34 nvme0n1p2
brw-rw----   1 root disk    259,   3 Oct 18 08:34 nvme0n1p3
brw-rw----   1 root disk    259,   4 Oct 18 08:34 nvme0n1p4
brw-rw----   1 root disk    259,   5 Oct 18 08:34 nvme0n1p5
brw-rw----   1 root disk    259,   6 Oct 18 08:34 nvme0n1p6
brw-rw----   1 root disk    259,   7 Oct 18 08:34 nvme0n1p7
```

```
ls -al /dev
```

```
crw-rw-rw-   1 root root    195,   0 Oct 18 08:34 nvidia0
crw-rw-rw-   1 root root    195, 255 Oct 18 08:34 nvidiactl
crw-rw-rw-   1 root root    195, 254 Oct 18 08:34 nvidia-modeset
crw-rw-rw-   1 root root    508,   0 Oct 18 08:34 nvidia-uvm
crw-rw-rw-   1 root root    508,   1 Oct 18 08:34 nvidia-uvm-tools
crw-------   1 root root    241,   0 Oct 18 08:34 nvme0
brw-rw----   1 root disk    259,   0 Oct 18 08:34 nvme0n1
brw-rw----   1 root disk    259,   1 Oct 18 08:34 nvme0n1p1
brw-rw----   1 root disk    259,   2 Oct 18 08:34 nvme0n1p2
brw-rw----   1 root disk    259,   3 Oct 18 08:34 nvme0n1p3
brw-rw----   1 root disk    259,   4 Oct 18 08:34 nvme0n1p4
brw-rw----   1 root disk    259,   5 Oct 18 08:34 nvme0n1p5
brw-rw----   1 root disk    259,   6 Oct 18 08:34 nvme0n1p6
brw-rw----   1 root disk    259,   7 Oct 18 08:34 nvme0n1p7
```

Owner, Group, Others

What Problem Did udev Solve?

- Persistent Naming
- Remember when you had a substitute teacher and roll call happened?
- Example from RedHat:
  - “A disk fails to power up or respond to the SCSI controller. This results in it not being detected by the normal device probe. The disk is not accessible to the system and subsequent devices will have their major and minor number range, including the associated sd names shifted down. For example, **if a disk normally referred to as sdb is not detected, a disk that is normally referred to as sdc would instead appear as sdb**.”

Shell Scripting 101

- Can make a file that will run bash commands
- Link for more info
- The power of the `which` command

Exercise 11

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

```
#!/bin/bash

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

<- From reference at bottom

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

$ sudo ls -al /usr/local/bin

$ /usr/local/bin/trigger.sh

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

<- From reference at bottom

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`**
- What do we use for subsystem? Stack Exchange
- **`SUBSYSTEM=="mes", ACTION=="add", RUN+="/usr/local/bin/trigger.sh"`**
- **`$ 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