5-3 SERIAL PERIPHERAL INTERFACE SPI

DOING SERIAL *FAST*

DAY 5-3

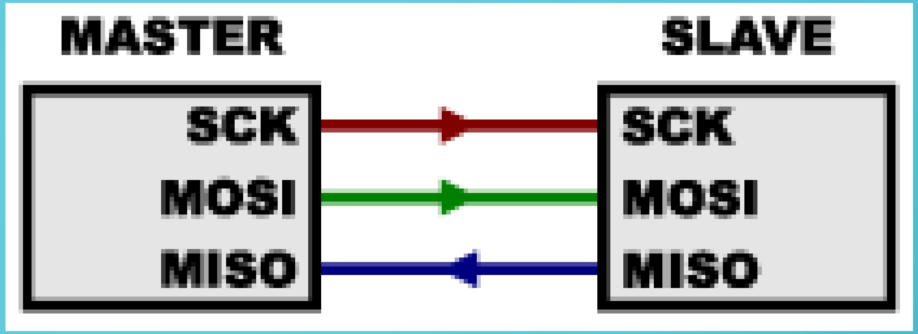
ASSIGNMENT:

- HW 02, Due Wednesday
- HW 03, Due Friday
- HW 04, Due Thursday

TODAY’S TOPICS:

- Projects
- SPI
- LCD

SPI: INTRO



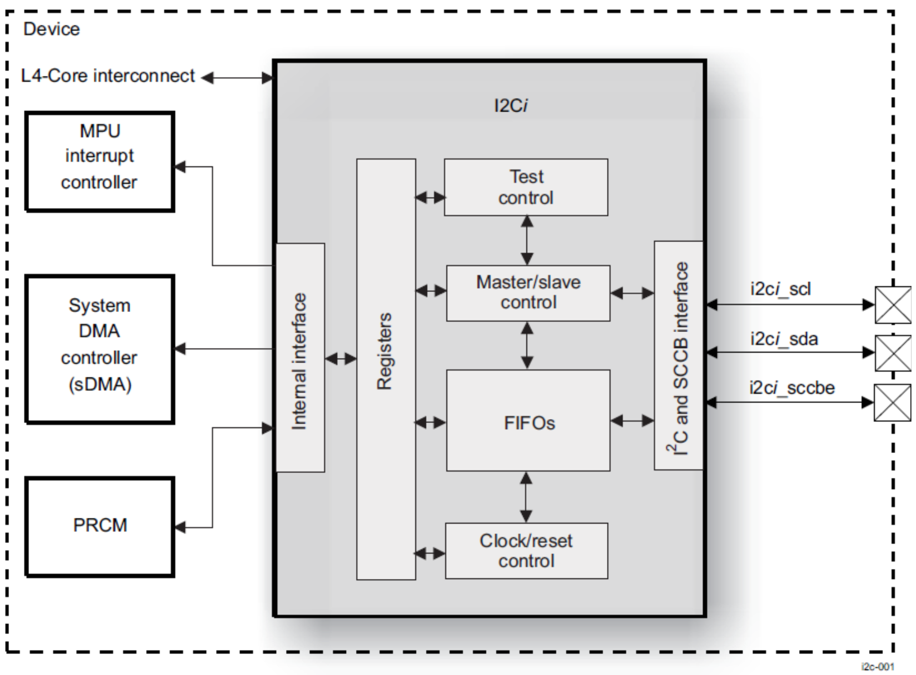
- Serial Peripheral Interface (SPI) is an interface bus commonly used to send data between microcontrollers and small peripherals
  - (shift registers, sensors, and SD cards)
- It uses separate clock and data lines, along with a select line to choose the device you wish to talk to.

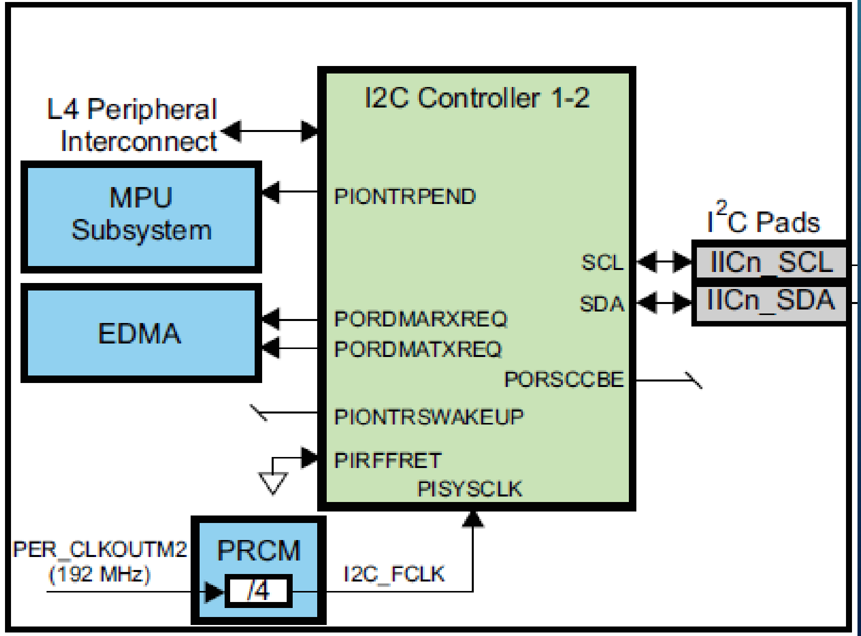
From: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi

I^{2}C - FLASHBACK

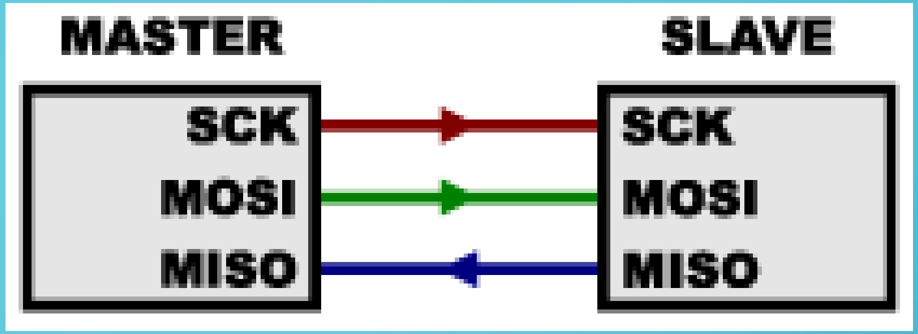
- “two-wire interface” standard
- Used to attach low-speed peripherals to embedded systems

Figure 17-1. HS I²C Controllers Overview Block Diagram





SPI: INTRO

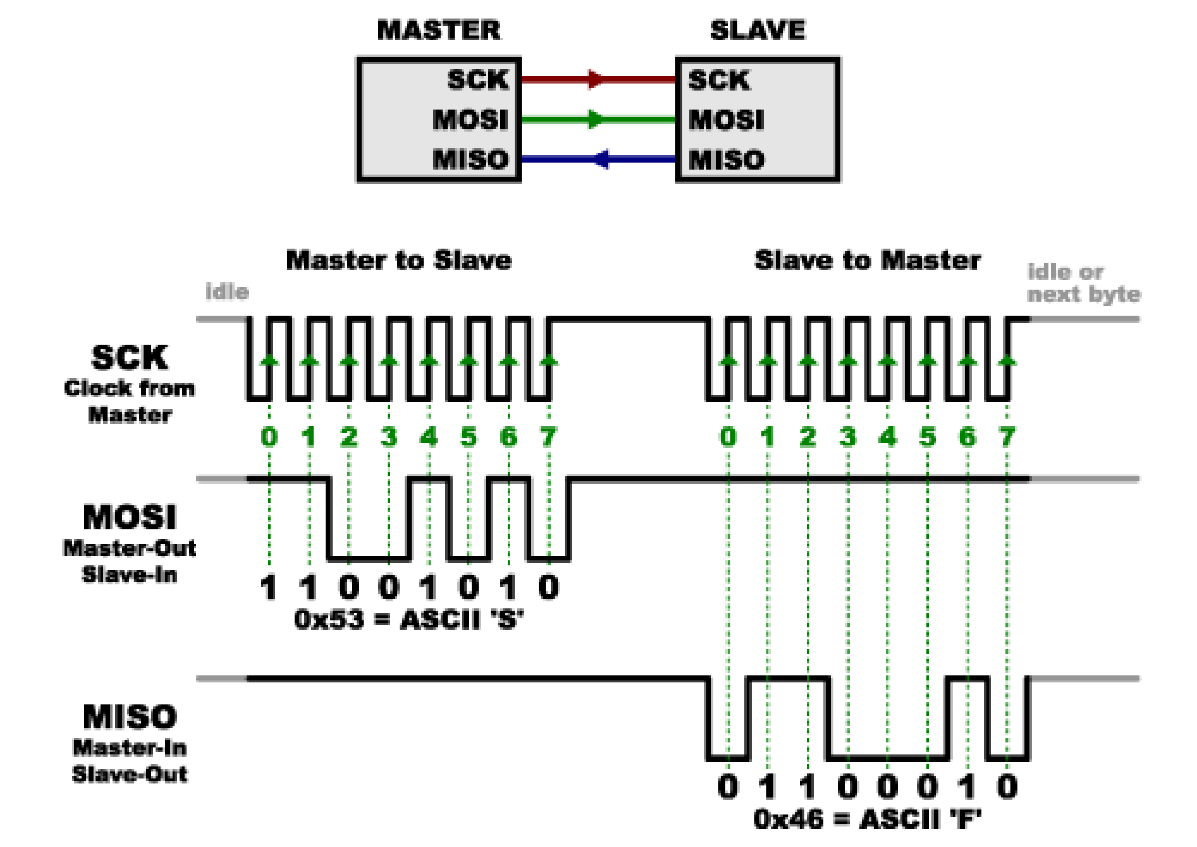


- Serial Peripheral Interface (SPI) is an interface bus commonly used to send data between microcontrollers and small peripherals
  - (shift registers, sensors, and SD cards)
- It uses separate clock and data lines, along with a select line to choose the device you wish to talk to.

From: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi

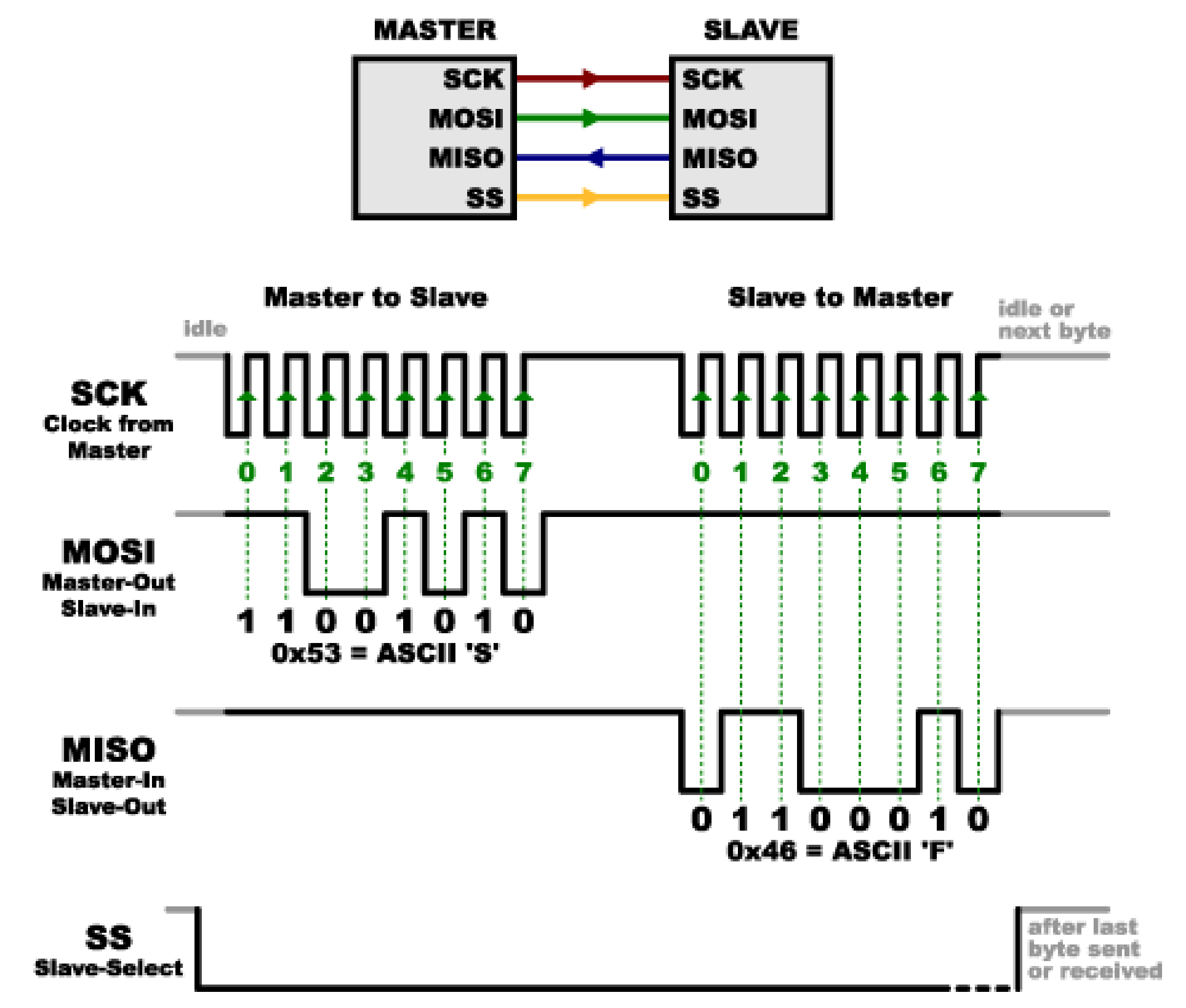
SPI - CLOCK

- One side generates the clock
- Called the *master* (only one)
- Other side is called the *slave*
- There are two data lines
  - MOSI – Master Out / Slave In
  - MISO – Master In / Slave Out



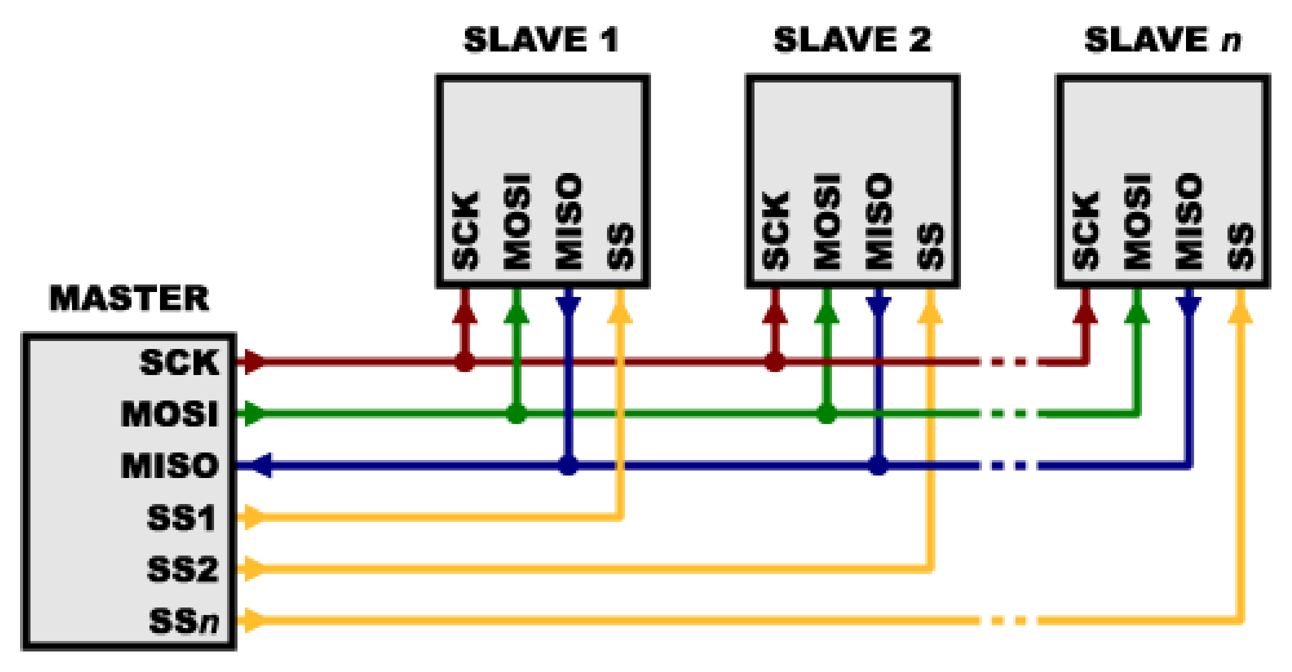
CHIP SELECT

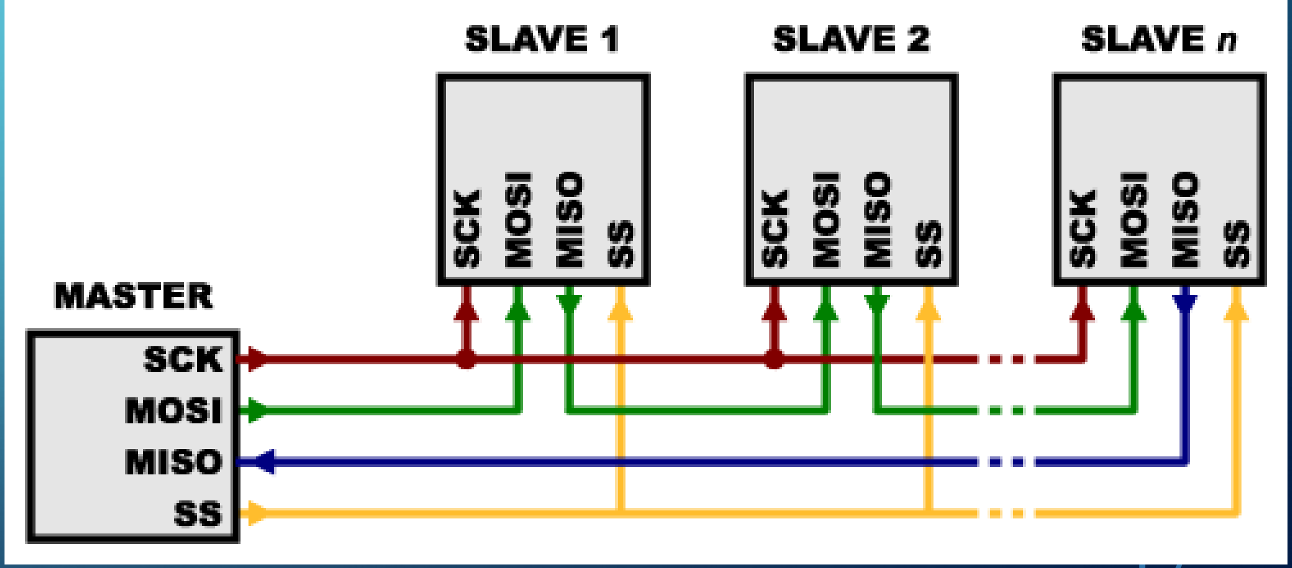
- Tells slave it should wake up.
- Used if multiple slaves
- CS or SS (slave select)



MULTIPLE SLAVES

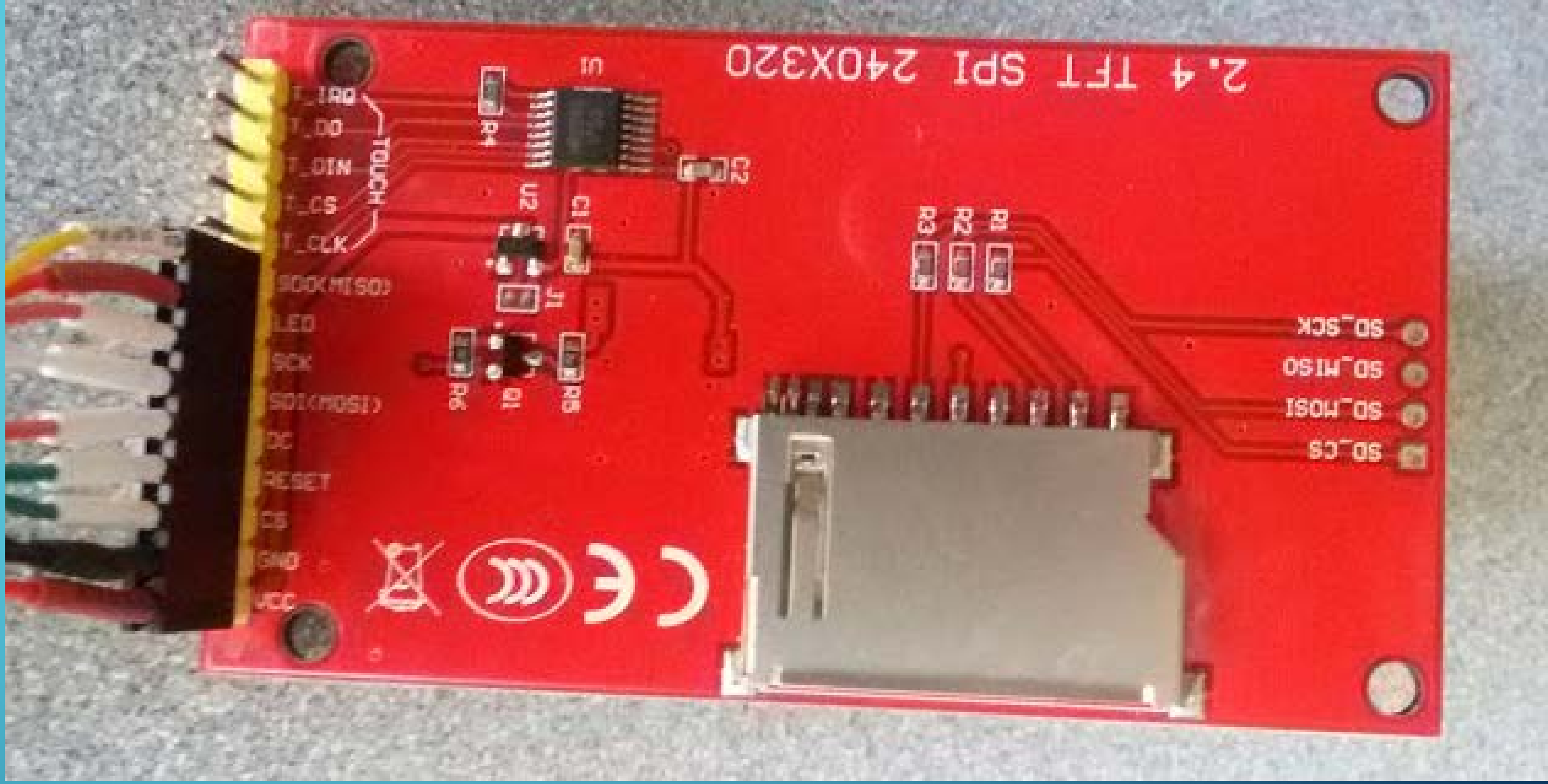
- Use multiple Chip Selects
  - SP1.1 and SP1.2 on Blue
- Daisy-chain
  - Used with addressable LEDs



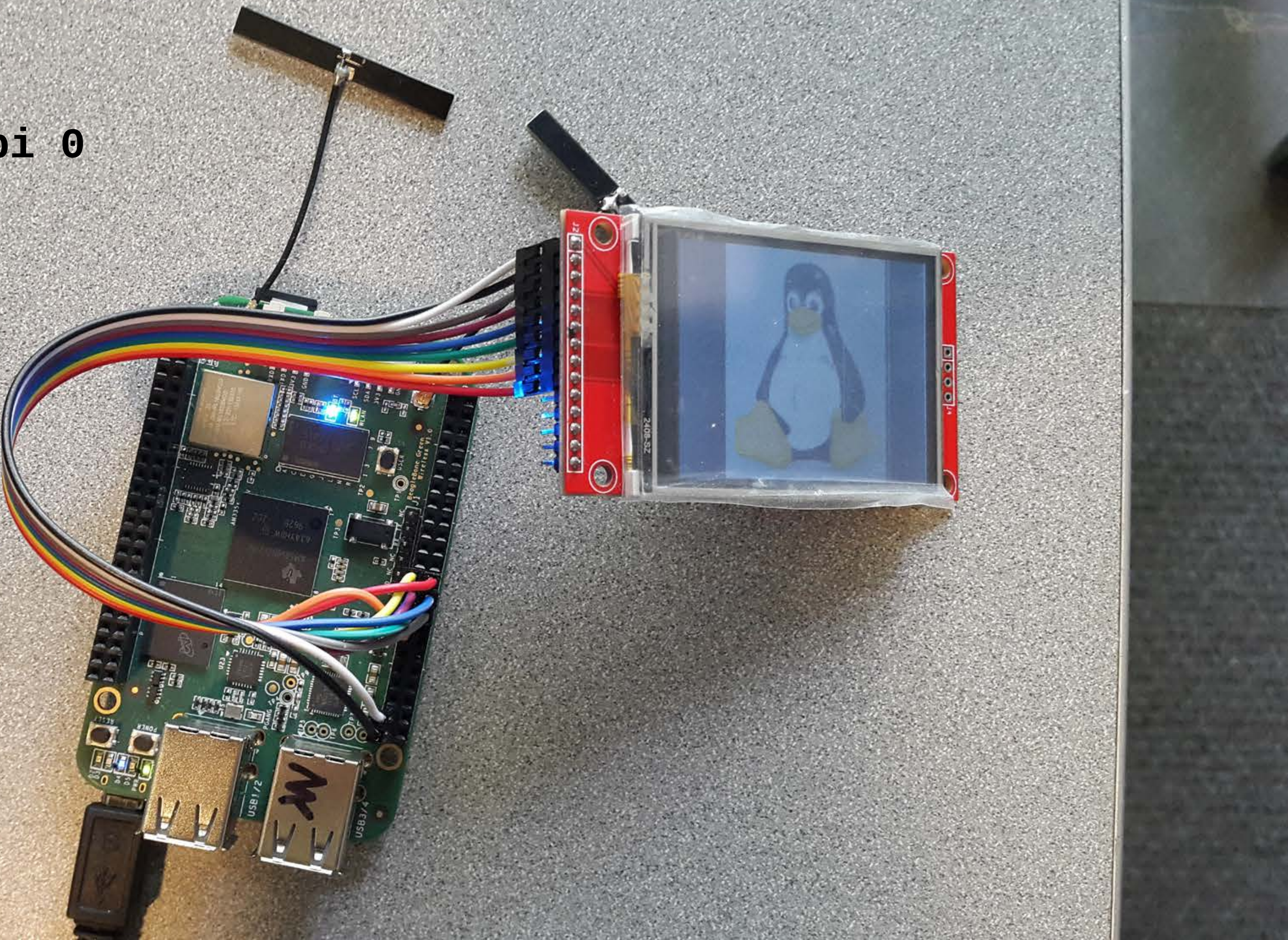


LCD DISPLAY

- MISO
- LED
- SCK
- MOSI
- DC
- Reset
- CS
- GND
- VCC



```
# For using spi 0
MISO    P9_21
LED     P9_16
SCK     P9_22
MOSI    P9_18
D/C     P9_19
RESET   P9_20
CS      P9_17
GND     P9_2
VCC     P9_4
```

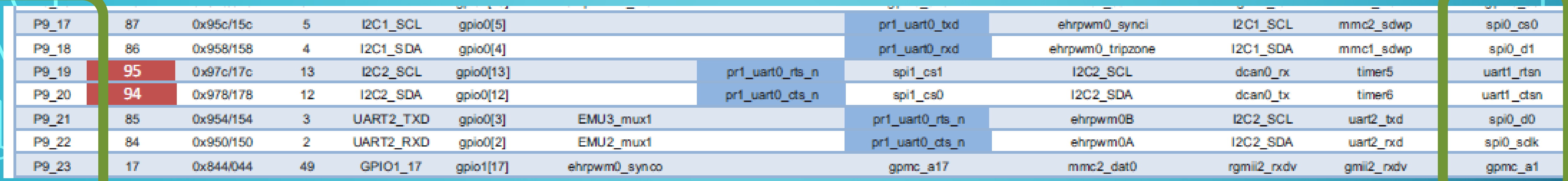


LCD SOFTWARE

```
bone$ cd exercises/displays/ili9341
bone$ ./on.sh
bone$ ./off.sh
bone$ ./reset.sh
```

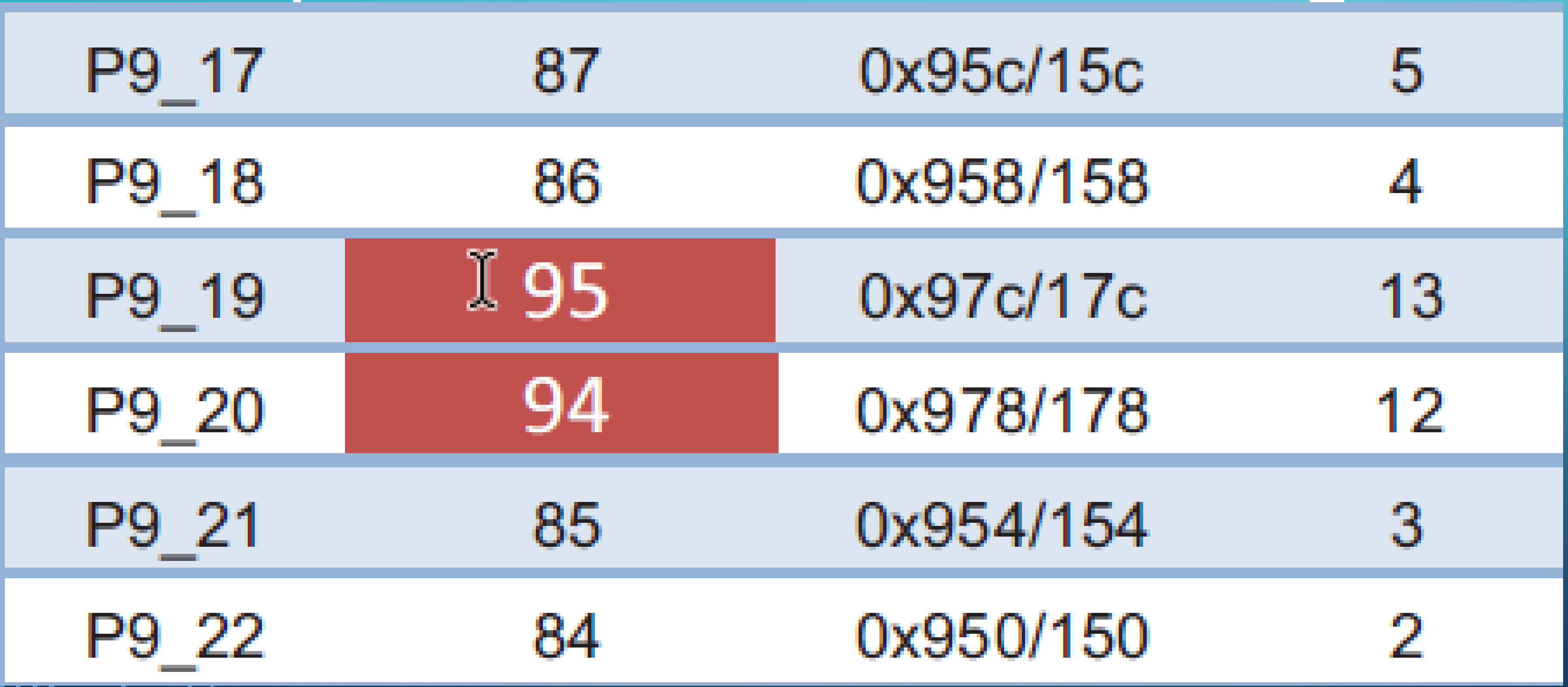
ON.SH

```
export LED=51       # P9_16
# This is for the Black SPI 0
export RESET=12     # RESET - P9_20
export DC=13        # D/C   - P9_19
export CS=5         # CS    - P9_17
```

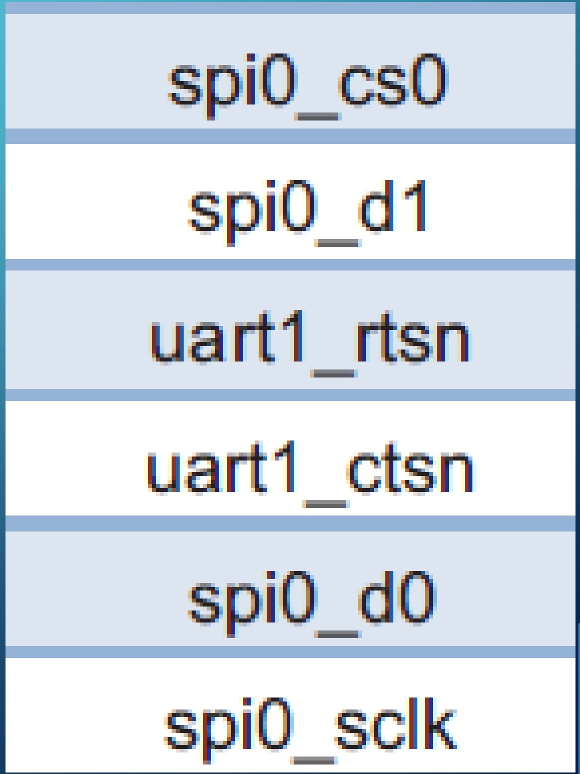


P9_17	87	0x95c/15c	5	I2C1_SCL	gpio0[5]			pr1_uart0_txd	ehrpwm0_synci	I2C1_SCL	mmc2_sdwp	spi0_cs0
P9_18	86	0x958/158	4	I2C1_SDA	gpio0[4]			pr1_uart0_rxd	ehrpwm0_tripzone	I2C1_SDA	mmc1_sdwp	spi0_d1
P9_19	95	0x97c/17c	13	I2C2_SCL	gpio0[13]		pr1_uart0_rts_n	spi1_cs1	I2C2_SCL	dcan0_rx	timer5	uart1_rtsn
P9_20	94	0x978/178	12	I2C2_SDA	gpio0[12]		pr1_uart0_cts_n	spi1_cs0	I2C2_SDA	dcan0_tx	timer6	uart1_ctsn
P9_21	85	0x954/154	3	UART2_TXD	gpio0[3]	EMU3_mux1		pr1_uart0_rts_n	ehrpwm0B	I2C2_SCL	uart2_txd	spi0_d0
P9_22	84	0x950/150	2	UART2_RXD	gpio0[2]	EMU2_mux1		pr1_uart0_cts_n	ehrpwm0A	I2C2_SDA	uart2_rxd	spi0_sclk
P9_23	17	0x844/044	49	GPIO1_17	gpio1[17]	ehrpwm0_synco		gpmc_a17	mmc2_dat0	rgmii2_rxdv	gmii2_rxdv	gpmc_a1

```
export RESET=12        # RESET - P9_20
```



P9_17	87	0x95c/15c	5
P9_18	86	0x958/158	4
P9_19	95	0x97c/17c	13
P9_20	94	0x978/178	12
P9_21	85	0x954/154	3
P9_22	84	0x950/150	2



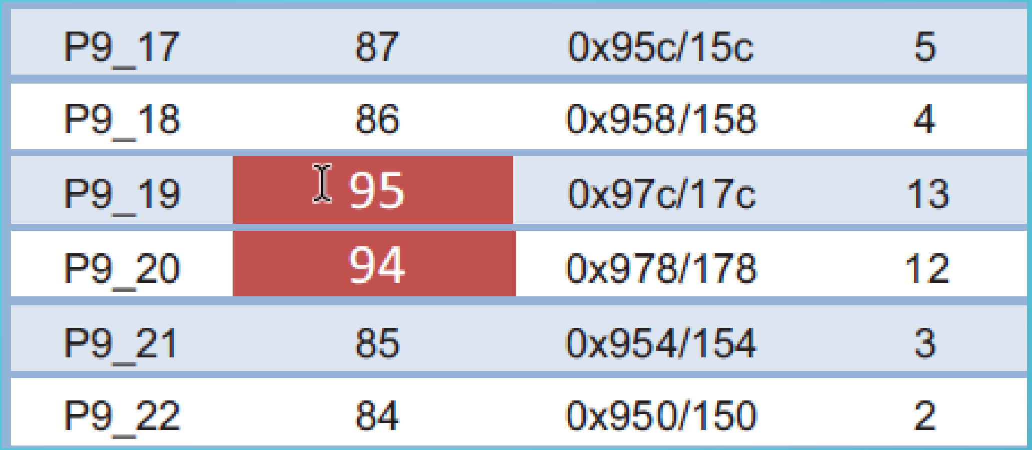
spi0_cs0
spi0_d1
uart1_rtsn
uart1_ctsn
spi0_d0
spi0_sclk

ON.SH - 2

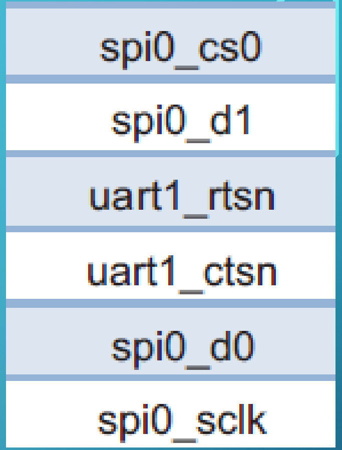
There are some 53 modules running on the current image.

```
sudo bash << EOF
# Remove the framebuffer modules
 if lsmod | grep -q 'fbtft_device ' ; then rmmod fbtft_device;       fi
 if lsmod | grep -q 'fb_ili9341 '   ; then rmmod --force fb_ili9341; fi
 if lsmod | grep -q 'fbtft '        ; then rmmod --force fbtft;      fi
```

ON.SH - 3



P9_17	87	0x95c/15c	5
P9_18	86	0x958/158	4
P9_19	95	0x97c/17c	13
P9_20	94	0x978/178	12
P9_21	85	0x954/154	3
P9_22	84	0x950/150	2



spi0_cs0
spi0_d1
uart1_rtsn
uart1_ctsn
spi0_d0
spi0_sclk

```
sudo bash << EOF
…
    # Set the pinmuxes for the display
    config-pin P9_19 gpio    # D/C
    config-pin P9_20 gpio    # RESET
    config-pin P9_18 spi     # spi 0_d1 MOSI
    config-pin P9_21 spi     # spi 0_d0 MISO
    config-pin P9_22 spi_sclk # spi 0_sclk
    config-pin P9_17 spi_cs # spi 0_cs0
```

ON.SH - 4

```
sudo bash << EOF
…
    # LED pin, turn on
    ./backlight.py
    sleep 0.1
    # Insert the framebuffer modules
    modprobe fbtft_device name=adafruit28 busnum=1
       rotate=90 gpios=reset:$RESET,dc:$DC cs=0
EOF
```

Why not bus 0?

MODINFO

```
bone$ modinfo fbtft_device
filename:       /lib/modules/4.14.58-ti-
r65/kernel/drivers/staging/fbtft/fbtft_device.ko.xz
license:        GPL
author:         Noralf Tronnes
description:    Add a FBTFT device.
depends:        fbtft
staging:        Y
intree:         Y
name:           fbtft_device
vermagic:       4.14.58-ti-r65 SMP preempt mod_unload modversions ARMv7 p2v8
```

MODINFO

```
parm:   name:Devicename (required). name=list => list all supported devices.
(charp)
parm:   rotate:Angle to rotate display counter clockwise: 0, 90, 180, 270 (uint)
parm:   busnum:SPI bus number (default=0) (uint)
parm:   cs:SPI chip select (default=0) (uint)
parm:   speed:SPI speed (override device default) (uint)
parm:   mode:SPI mode (override device default) (int)
parm:   gpios:List of gpios. Comma separated with the form: reset:23,dc:24 (when
overriding the default, all gpios must be specified) (charp)
parm:   fps:Frames per second (override driver default) (uint)
parm:   gamma:String representation of Gamma Curve(s). Driver specific. (charp)
```

MODINFO

```
parm: txbuflen:txbuflen (override driver default) (int)
parm: bgr:BGR bit (supported by some drivers). (int)
parm: startbyte:Sets the Start byte used by some SPI displays. (uint)
parm: custom:Add a custom display device. Use speed= argument to make it a SPI
device, else platform_device (bool)
parm: width:Display width, used with the custom argument (uint)
parm: height:Display height, used with the custom argument (uint)
parm: buswidth:Display bus width, used with the custom argument (uint)
parm: init:Init sequence, used with the custom argument (array of int)
parm: debug:level: 0-7 (the remaining 29 bits is for advanced usage) (ulong)
parm: verbose:0 silent, >0 show gpios, >1 show devices, >2 show devices before
(default=3) (uint)
```

ON.SH - 4

```
sudo bash << EOF
…
 # LED pin, turn on
    ./backlight.py
    sleep 0.1
    # Insert the framebuffer modules
    modprobe fbtft_device name=adafruit28 busnum=1
       rotate=90 gpios=reset:$RESET,dc:$DC cs=0
EOF
```

FRAMEBUFFERS

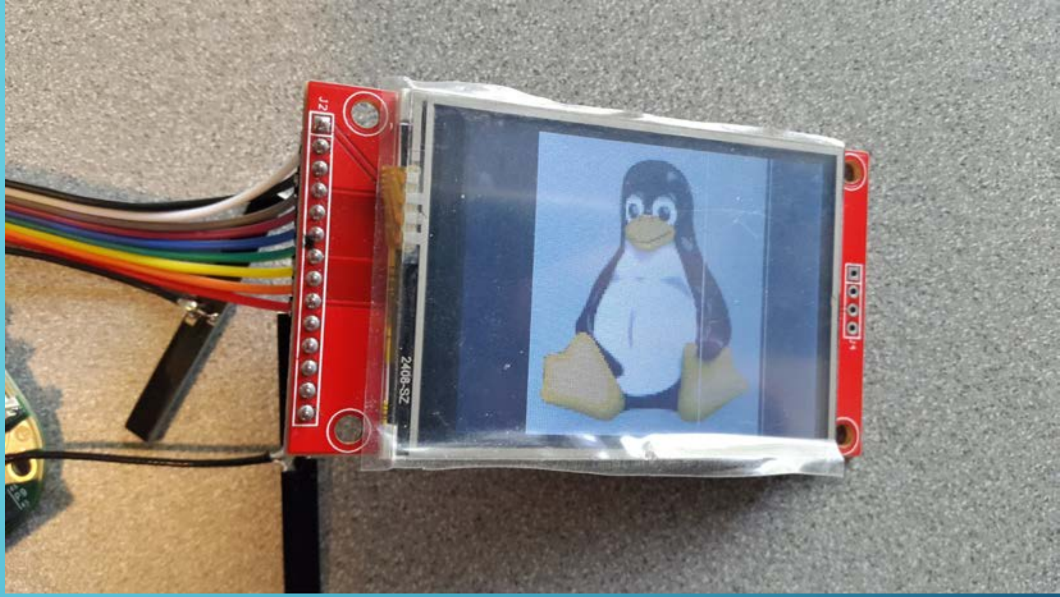
- Once the modprobe is run
- **`/dev/fb0`** appears

```
bone$ ls -ls /dev/fb0
0 crw-rw---- 1 root video 29, 0 Sep 12 13:44 /dev/fb0
bone$ grep video /etc/group
video:x:44:debian
```

FRAMEBUFFERS

- Once the modprobe is run
- **`/dev/fb0`** appears
- You can read and write to it.
- `bone$ cat /dev/fb0 > /tmp/backup`
- `bone$ cat /tmp/backup > /dev/fb0`

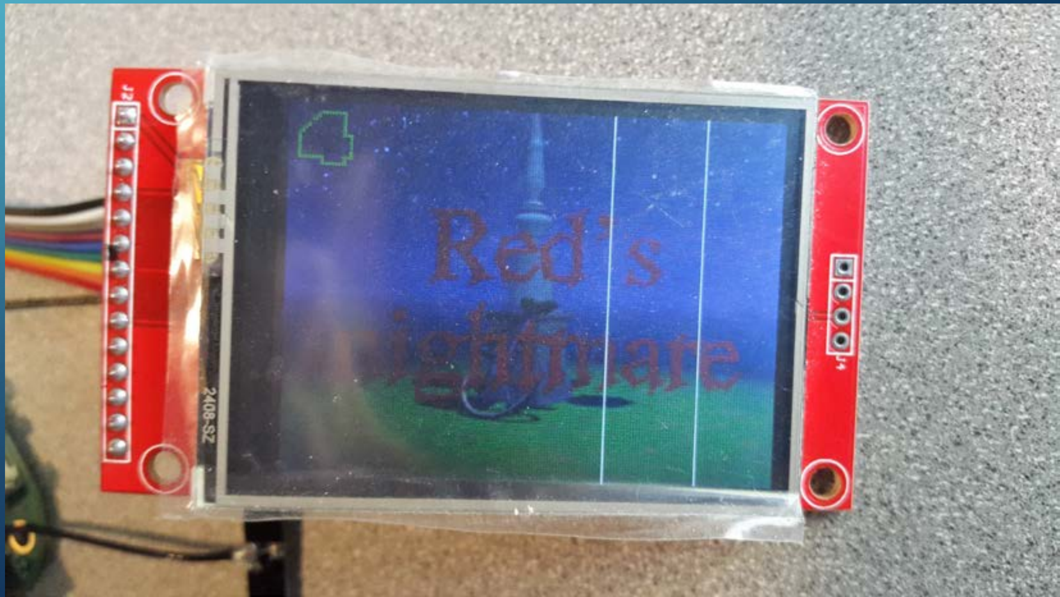
PROGRAMS USE IT



- Display an image

```
bone$ fbi -noverbose -T 1 -a boris.png
```

- Play a movie
- `bone$ mplayer RedsNightmare.mpg`
- pygame!



PYGAME

```
bone$ cd exercises/displays/ili9341/fb
bone$ ./on.sh
bone$ cd pygame
bone$ ./install.sh
bone$ ./clockWeather.py
```

Wait an hour

You may have to run it twice.

MMAP

- You can open it with mmap

FRAMEBUFFER.C

```
// Open the file for reading and writing
fbfd = open("/dev/fb0", O_RDWR);
if (fbfd == -1) {
     perror("Error: cannot open framebuffer device");
     exit(1);    }
printf("The framebuffer device was opened successfully.\n");
```

FRAMEBUFFER.C - 2

```
 // Figure out the size of the screen in bytes
screensize = vinfo.xres * vinfo.yres * vinfo.bits_per_pixel / 8;
// Map the device to memory
fbp = (char *) mmap(0, screensize, PROT_READ | PROT_WRITE, MAP_SHARED,
fbfd, 0);
if ((int)fbp == -1) {
        perror("Error: failed to map framebuffer device to memory");
        exit(4);
    }
printf("The framebuffer device was mapped to memory successfully.\n");
```

FRAMEBUFFER.C - 3

```
// Figure out where in memory to put the pixel
location = (x+vinfo.xoffset) * (vinfo.bits_per_pixel/8) +
           (y+vinfo.yoffset) * finfo.line_length;
int r = 0;     // 5 bits
int g = 0;     // 6 bits
int b = 31;    // 5 bits
unsigned short int t = r<<11 | g << 5 | b;
*((unsigned short int*)(fbp + location)) = t;
```

SCREEN SIZE

```
 // Get fixed screen information
if (ioctl(fbfd, FBIOGET_FSCREENINFO, &finfo) == -1) {
        perror("Error reading fixed information");
        exit(2);
    }
    // Get variable screen information
if (ioctl(fbfd, FBIOGET_VSCREENINFO, &vinfo) == -1) {
        perror("Error reading variable information");
        exit(3);
    }
```

FOLLOW UP

- You’ll get to play with this on the next homework