Admin

Tue is **Democracy Day!**

Assign 5 due Wed 5pm, grace til Thu 5pm

No lab Tue eve

Instead join Wed lab, come to OH W/Th/F, ask on Ed

HDMI monitors lab6/assign6 avail in lab room

Today: Graphics and framebuffers

Drawing to the screen, neat!

Honestly, though, just more practice with pointers and memory 🙃

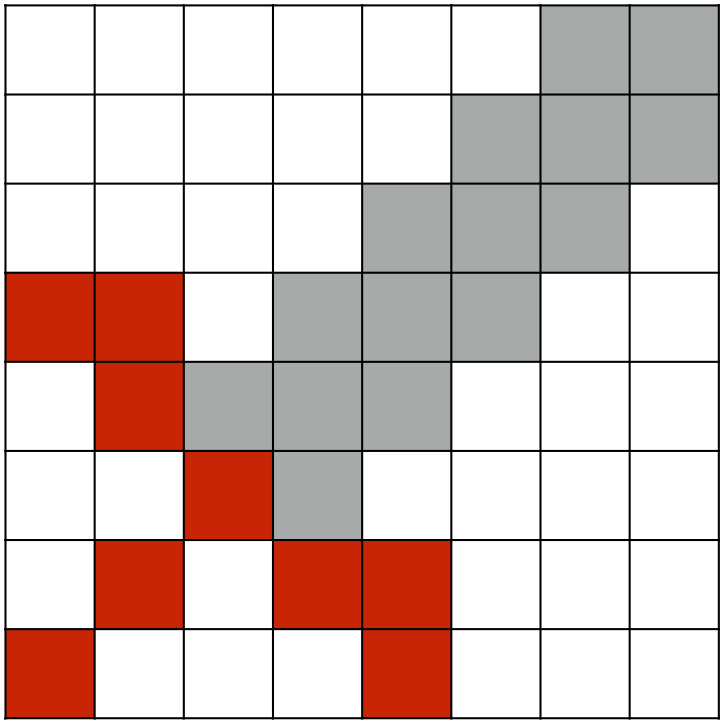
Mango Pi Display Engine pipeline

Road map

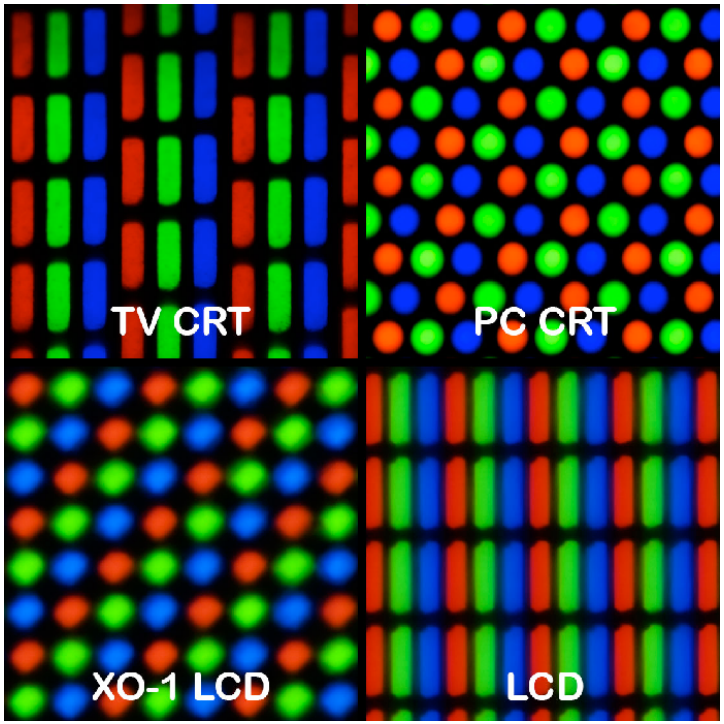
gpio
timer
uart
strings
printf
backtrace
symtab
malloc
keyboard
shell
fb
gl
console

Red-Green-Blue (RGB)

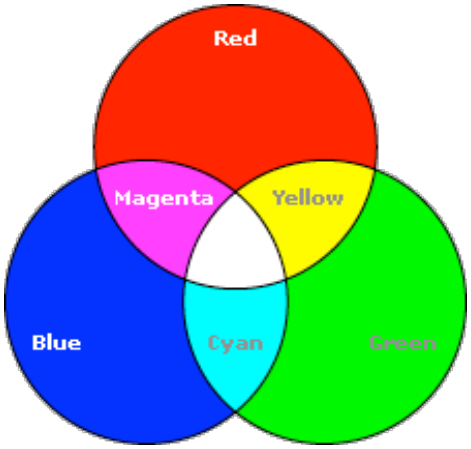
Pixels



Displays



Light



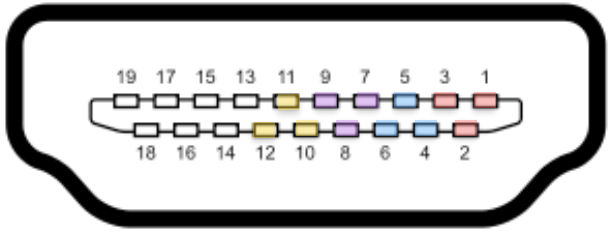
https://cs106a.stanford.edu/handouts_w2021/rgb-explorer.html

Mango Pi HDMI

Figure from
High-Definition Multimedia Interface
Specification Version 1.3a

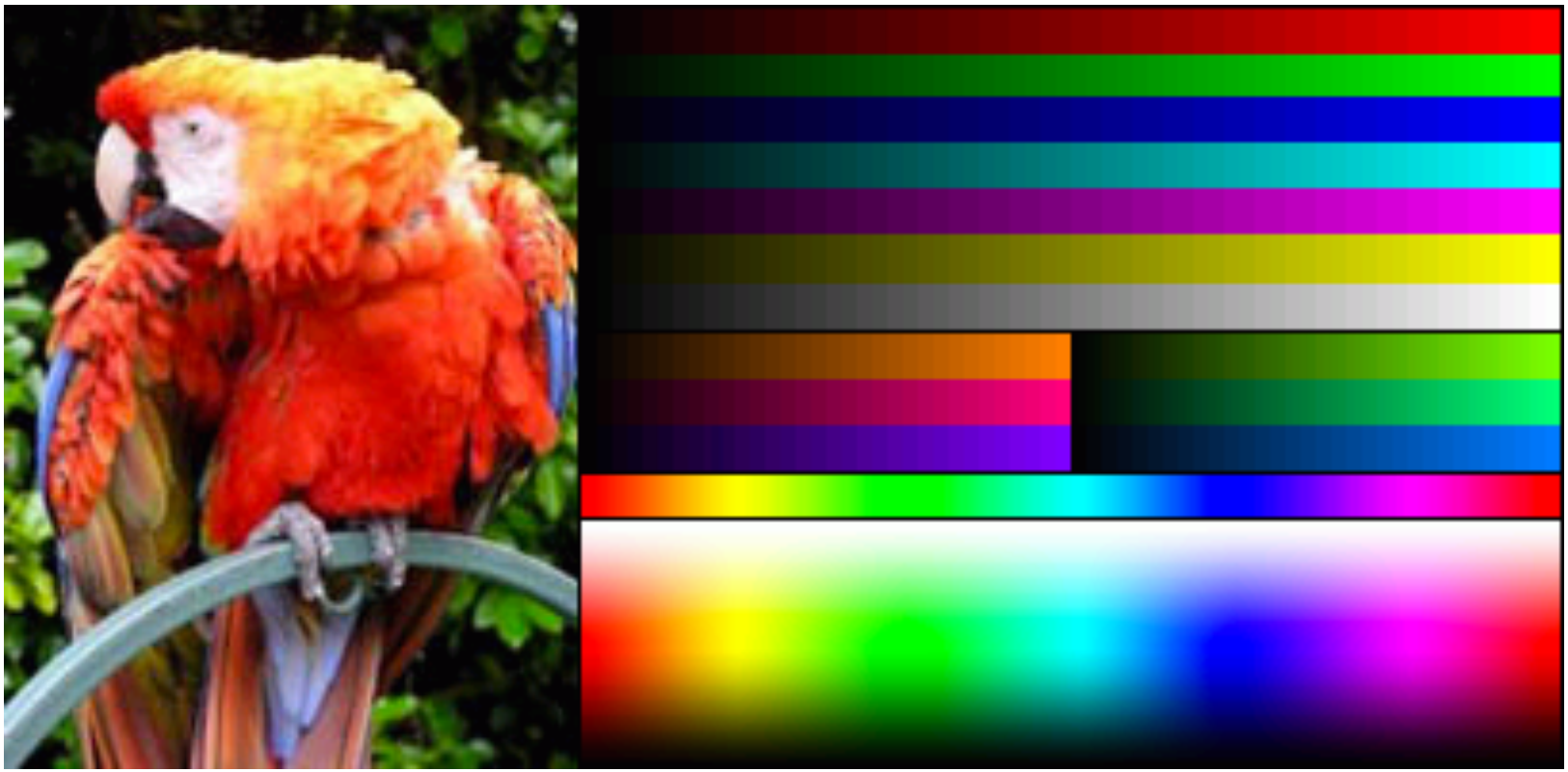
TMDS Channel	Pixel 0	Pixel 1	Pixel 2	Pixel 3	Pixel 4	…
0	B_0	B_1	B_2	B_3	B_4	…
1	G_0	G_1	G_2	G_3	G_4	…
2	R_0	R_1	R_2	R_3	R_4	…

Figure 6-1 Default pixel encoding: RGB 4:4:4, 8 bits/component



Clock
Data 0
Data 1
Data 2
Control

What is a frame buffer?

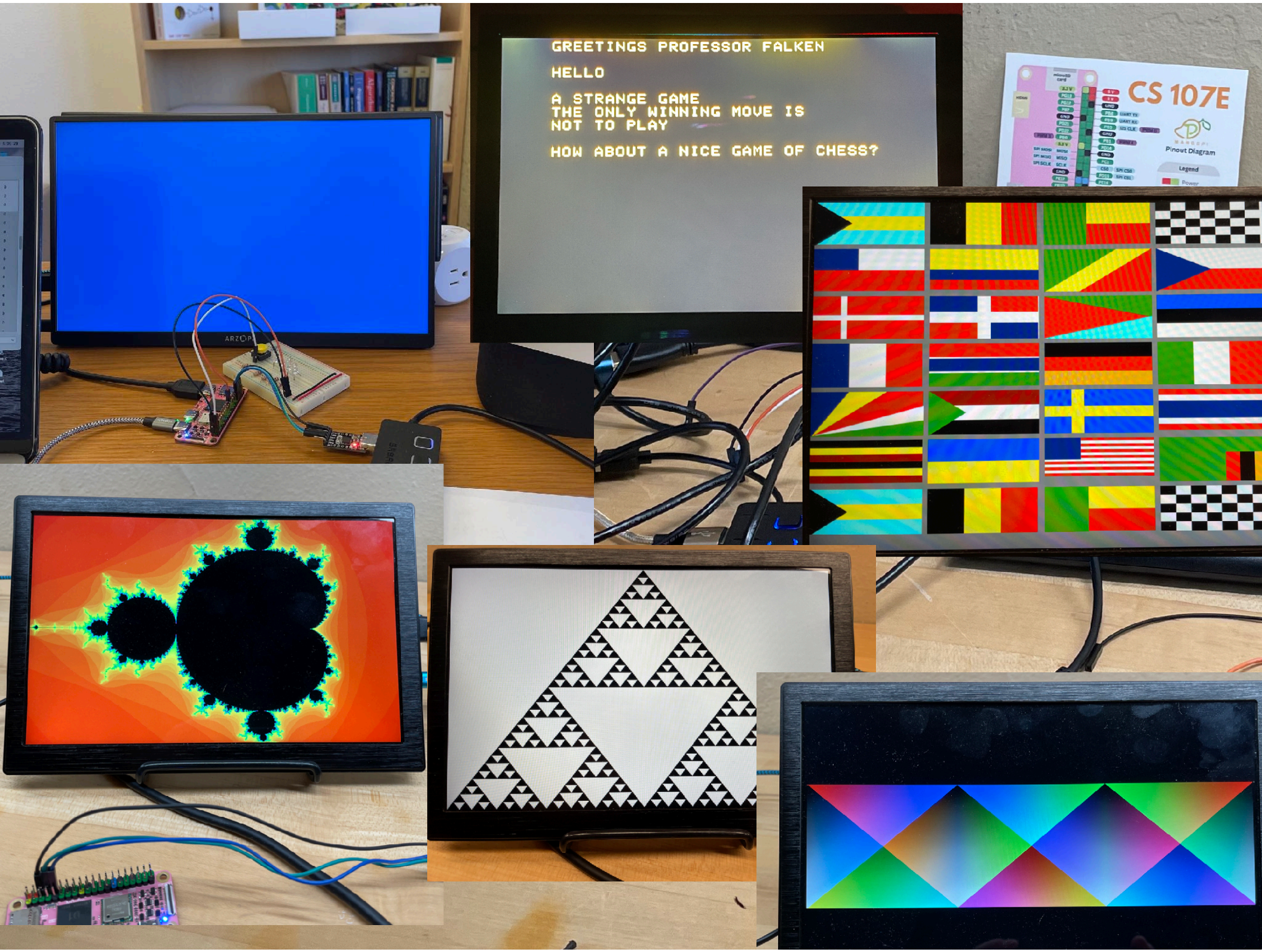


Framebuffer memory stores image

Image is a 2-D array of pixels

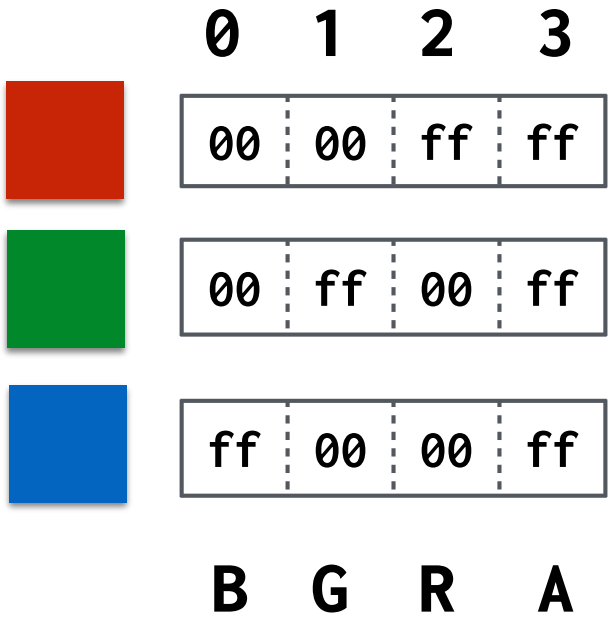
RGBA pixel depth 32-bit

8 bits each of red, green, blue, alpha


GREETINGS PROFESSOR FALKEN
HELLO
A STRANGE GAME
THE ONLY WINNING MOVE IS
NOT TO PLAY
HOW ABOUT A NICE GAME OF CHESS?
CS 107E
Pinout Diagram
Legend
Power

A pixel stores a color

32-bit RGBA (BGRA) pixel is four bytes
one byte for each component



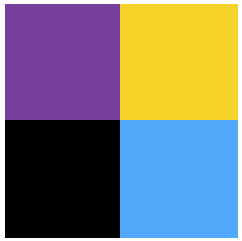
Take note: blue is first in memory (least significant byte)

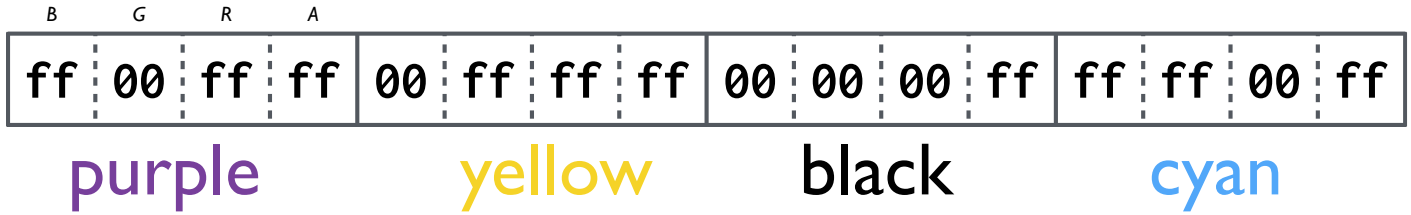
Framebuffer is an array

```
#define DEPTH 4
#define WIDTH 2
#define HEIGHT 2
```

Allocate memory using malloc!

```
unsigned char *fb = malloc(WIDTH*HEIGHT*DEPTH);
```





`0,0` is upper left corner of display

Order of pixels is `y` then `x`, within a pixel components are `bgra`

`fb` **module**

https://cs107e.github.io/header#fb

Coordinates with display hardware peripheral to configure for display size

Allocates memory for framebuffer (malloc!)

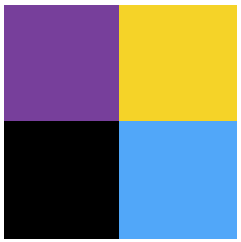
Tells hardware peripheral to display framebuffer

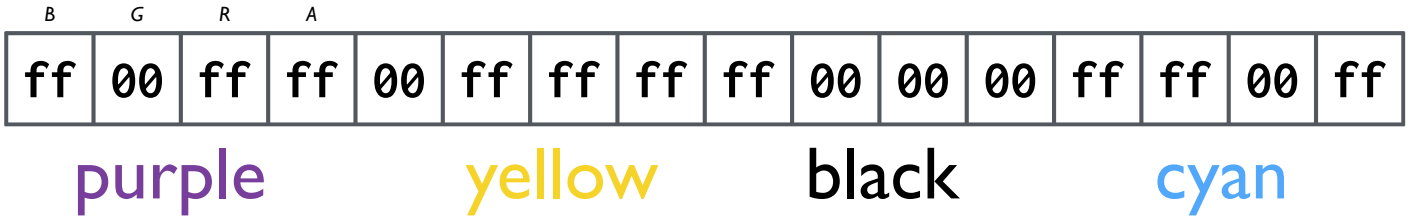
Review demo code: `code/clear`

fb as array of unsigned char

```
unsigned char *fb = malloc(WIDTH*HEIGHT*DEPTH);

   // fb[DEPTH*(WIDTH*y+x)+bgra] = component
fb[0] = 0xff; // x=0, y=0, blue component
fb[1] = 0x00; // x=0, y=0, green
fb[2] = 0xff; // x=0, y=0, red
fb[3] = 0xff; // x=0, y=0, alpha
fb[4] = 0x00; // x=1, y=0, blue
...
```

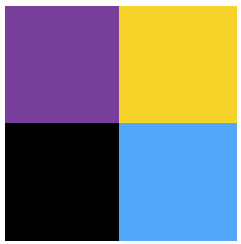


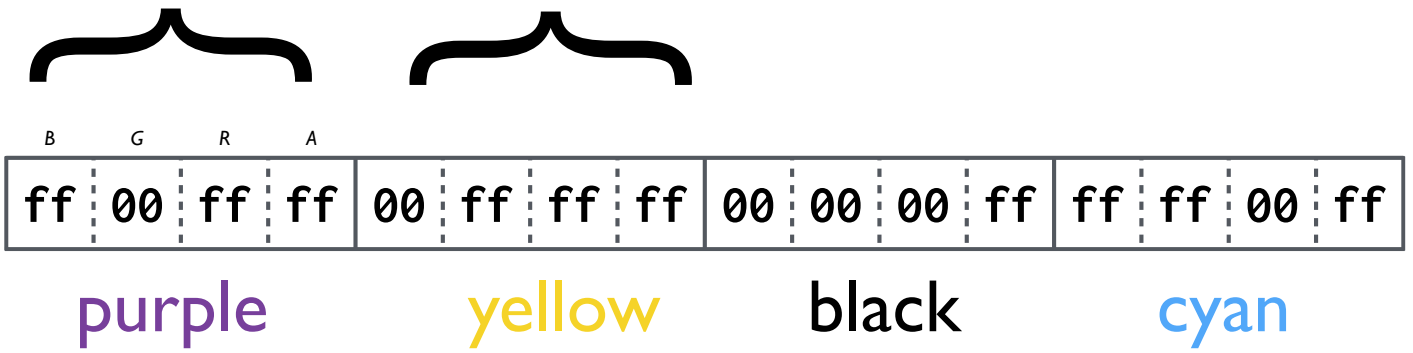


fb as array of unsigned int

```
unsigned int *fb = malloc(WIDTH*HEIGHT*DEPTH);

    // fb[WIDTH*y+x] = color
fb[0] = 0xffff00ff; // x=0, y=0
fb[1] = 0xffffff00; // x=1, y=0
fb[2] = 0xff000000; // x=0, y=1
fb[3] = 0xff00ffff; // x=1, y=1
```





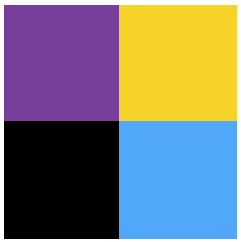
Memory is same size, contains same bytes as previous

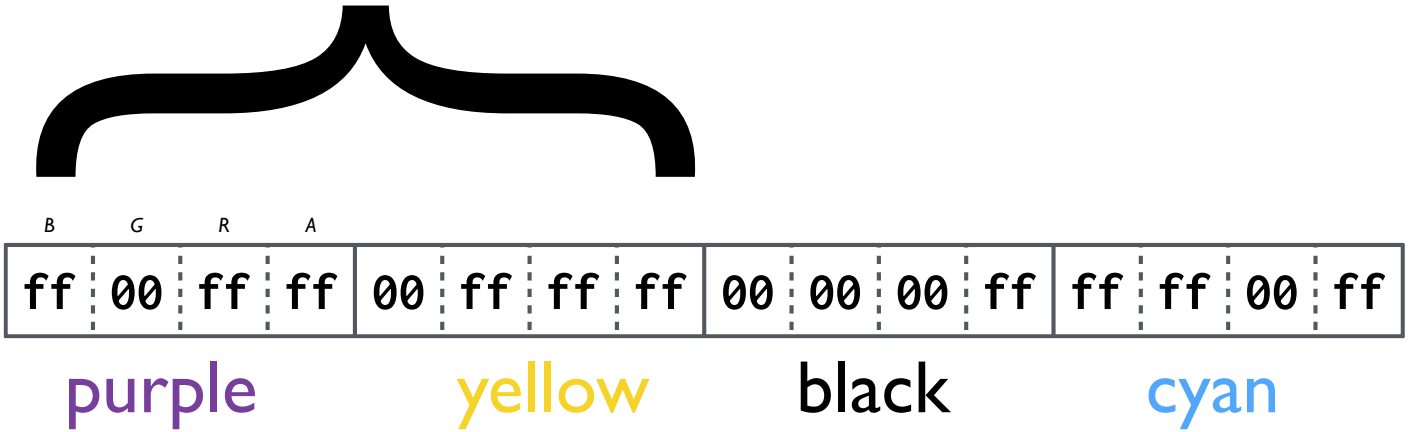
fb as 2-D array of `unsigned int`

```
unsigned int (*fb)[WIDTH] = malloc(WIDTH*HEIGHT*DEPTH);

    // fb[y][x] = color
fb[0][0] = 0xffff00ff;
fb[0][1] = 0xffffff00;
fb[1][0] = 0xff000000;
fb[1][1] = 0xff00ffff;
```

https://cdecl.org





Memory is same size, contains same bytes as previous

`gl` module

https://cs107e.github.io/header#gl

Layers on `fb` module, provides high-level graphics primitives

```
gl_clear()
gl_draw_pixel()
gl_draw_rect()
gl_draw_string()
```

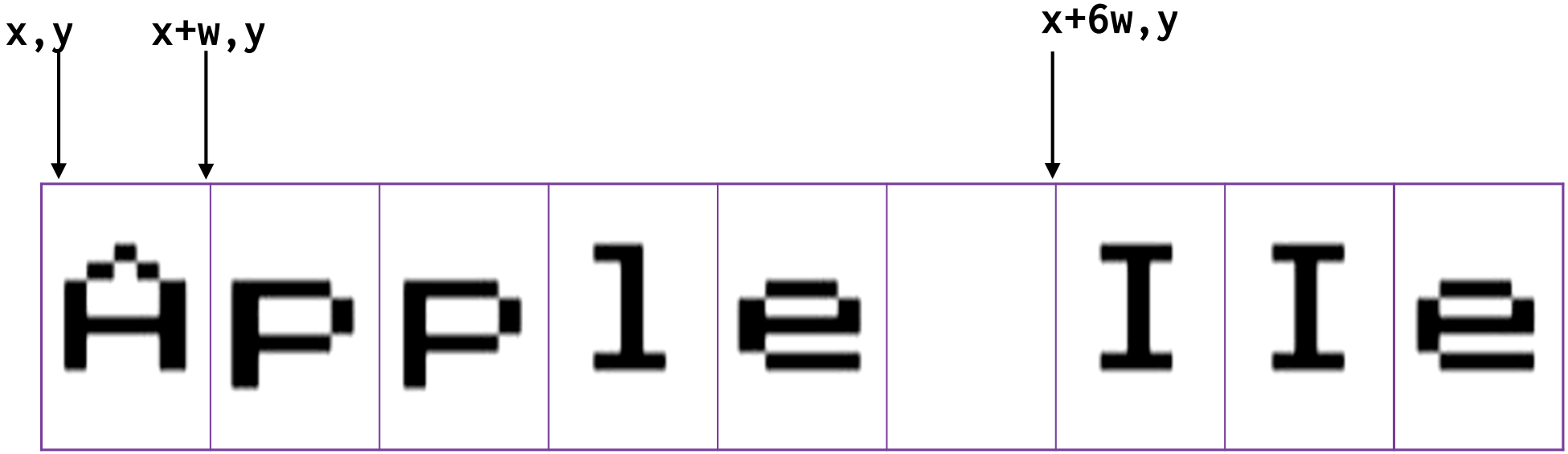
Review demo code: `code/draw`

Drawing text

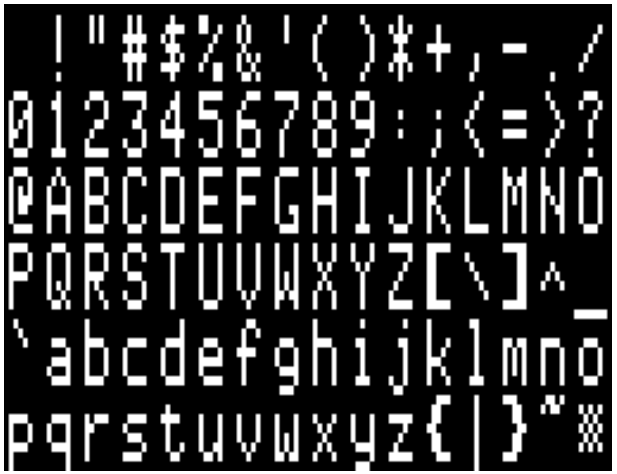
Font is a set of *glyphs*, one for each character

Our font is monospace, every glyph has same fixed width (vs proportional)

Our glyphs stored as bitmap image (vs raster/vector)



Apple II bitmap font



Original glyphs 7x8 pixels

Our font doubled to 14x16 pixels

All font glpyhs in one long, skinny image

https://cs107e.github.io/src#font

Single and Double Buffering

`code/doublebuffer`

Single Buffer

Draw directly to framebuffer while data is being output to display

Changed pixels immediately update (good for debugging!)

Redraw is "live" causes screen flicker/tearing

fb

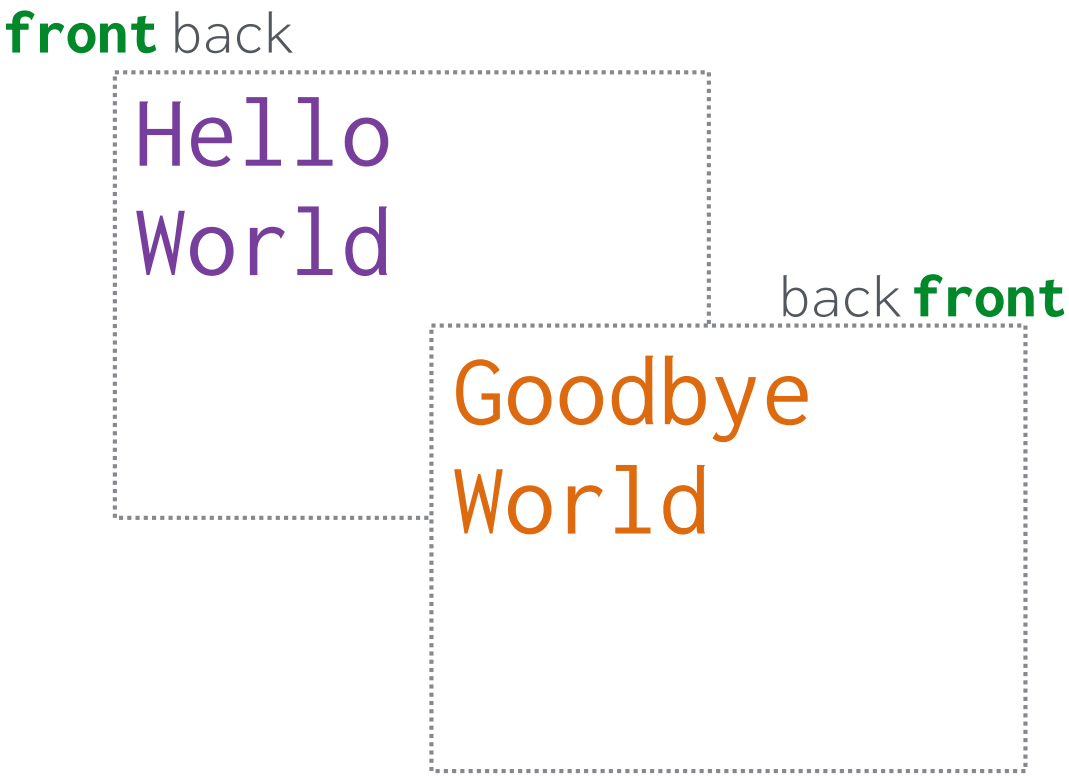
Double Buffer

Allocate two framebuffers: one for on-screen ("front"), other off-screen ("back")

Display front buffer

Draw updates in back buffer

Swap front and back when done with updates, frame comes on-screen in one smooth update



Display pipeline

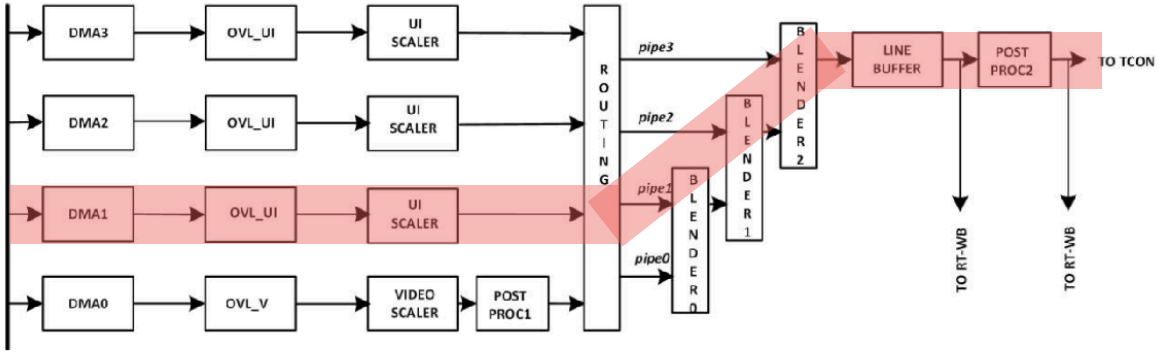
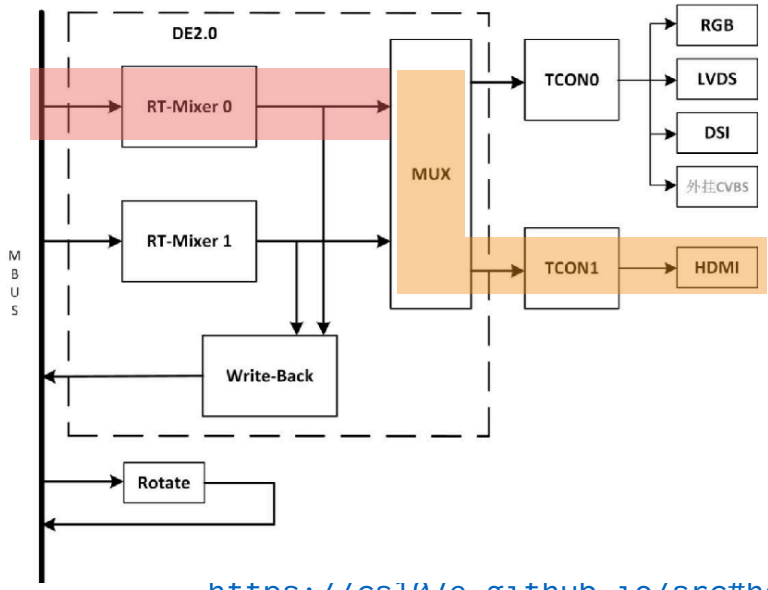


Figure 1-2. RT-MIXER0 Block Diagram



https://cs10/e.github.io/src#hdmi

`printf` create formatted string in memory, pass data to `uart`
-> `uart` peripheral outputs data, controls timing/protocol

`fb/gl/de` create pixel array in memory, pass data to `hdmi`
-> `hdmi` peripheral outputs data, controls timing/protocol

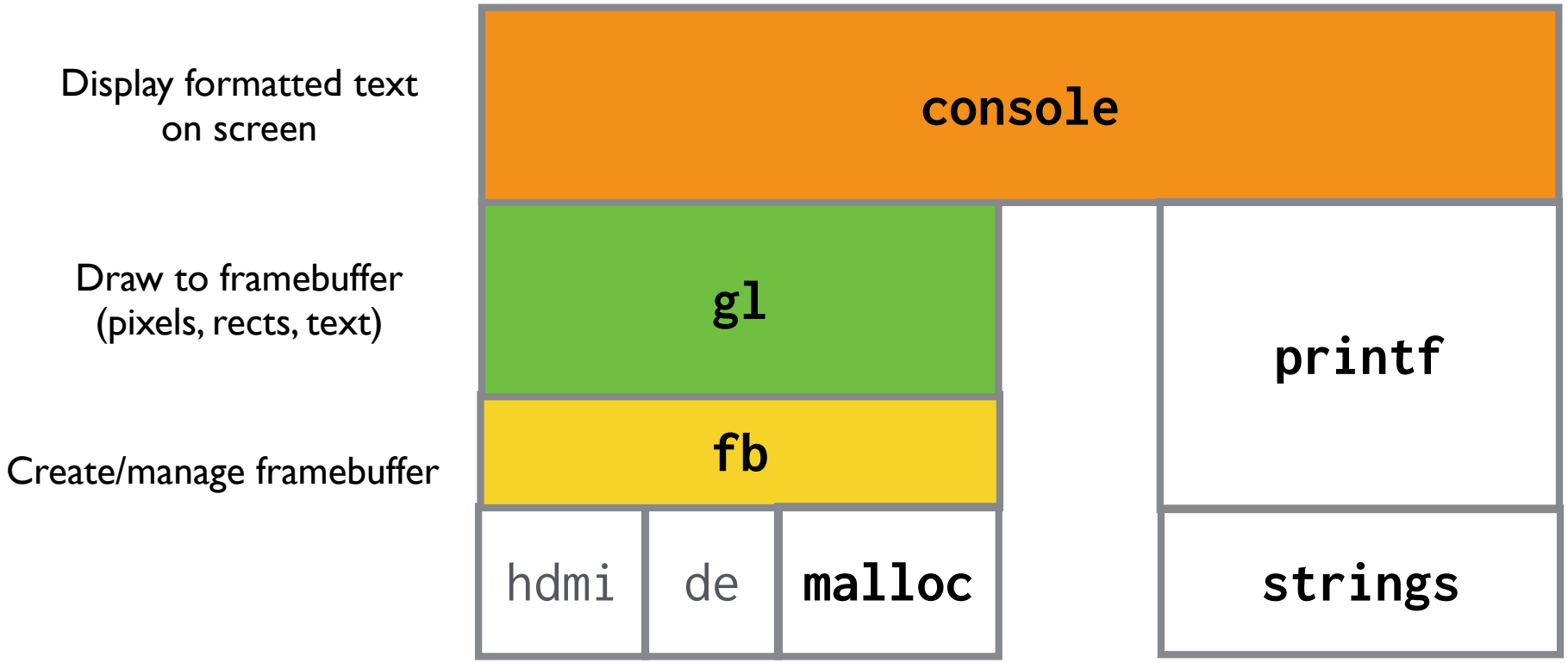
Display Engine

Mango Pi `DisplayEngine` peripheral

- Composes frames to be displayed
- Blend multiple channels, overlays, compositing
- Scale up/down
- Post-processing, enhancement, sharpening, contrast
- Video codec

https://cs107e.github.io/readings/Allwinner_DE2.0_Spec_V1.0.pdf

Layered abstractions



+ shell + keyboard = Apple IIe