

Admin

Retest cycle: 51 🕸️ squashed!

Full system = **6** bonus credits

Big reward for all your modules assign2-7 working in harmony

Use of MY_MODULE_SOURCES in assignment Makefile

Retest accepted through assign7 (earlier good too!)

No lecture Monday Presidents Day

Julie at conference Wed-Sat

Lecture next Fri Interrupts ★ Ben Ruland

Today: Graphics and framebuffers

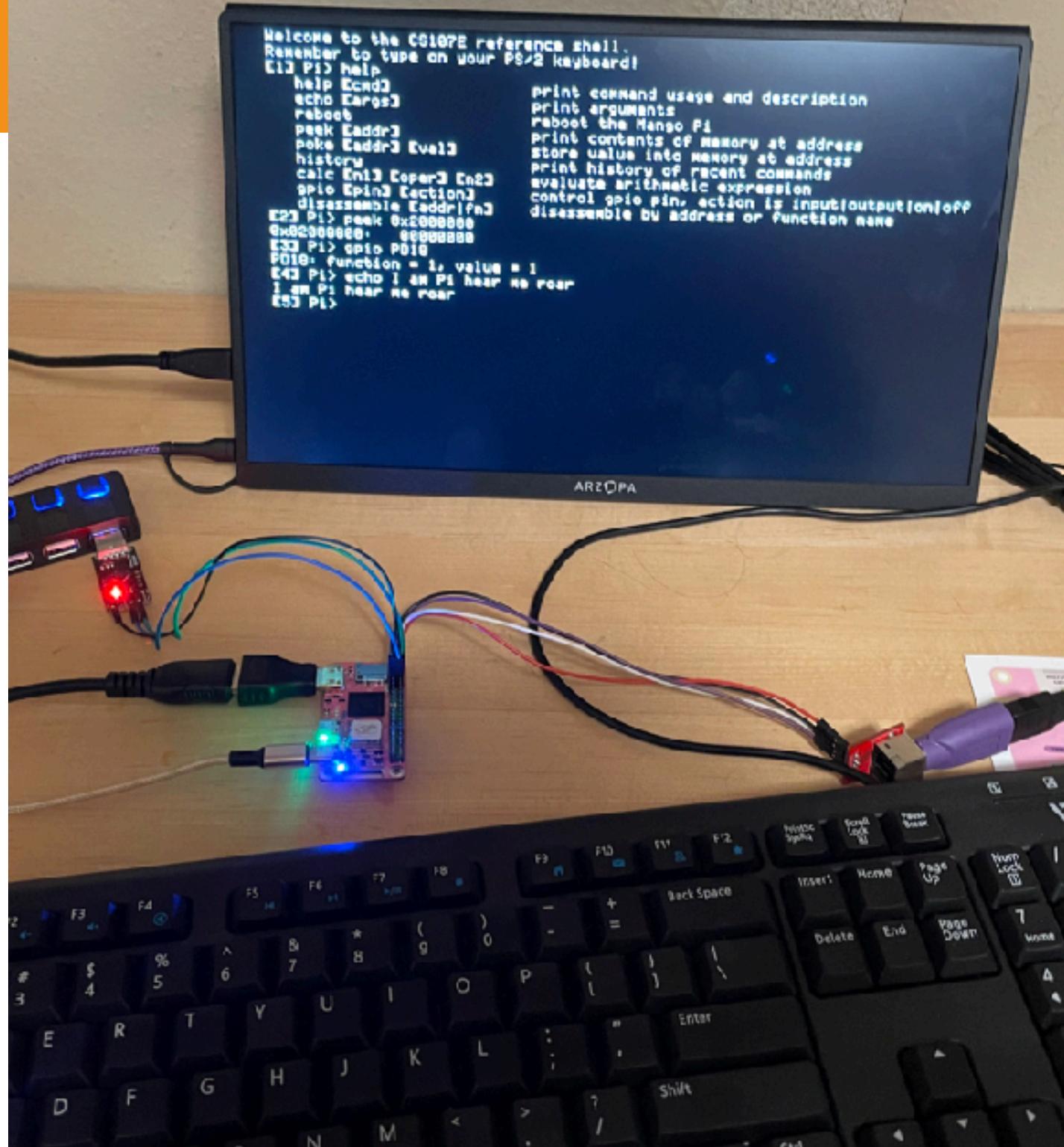
Drawing to the screen, neat!

Actually just more practice with pointers and memory 😊

Mango Pi Display Engine pipeline

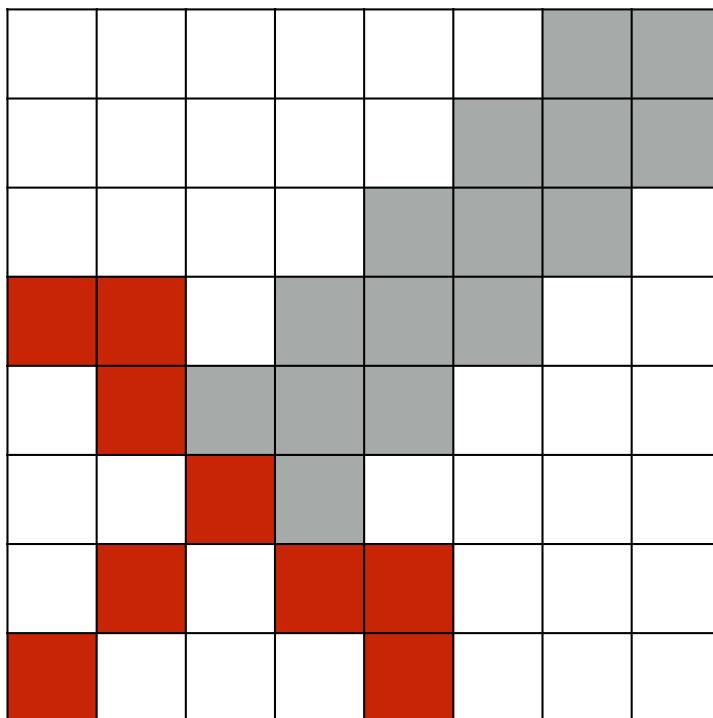
libmango

gpio	✓
timer	✓
uart	✓
strings	✓
printf	✓
malloc	✓
keyboard	✓
shell	✓
fb	
gl	
console	
interrupts	

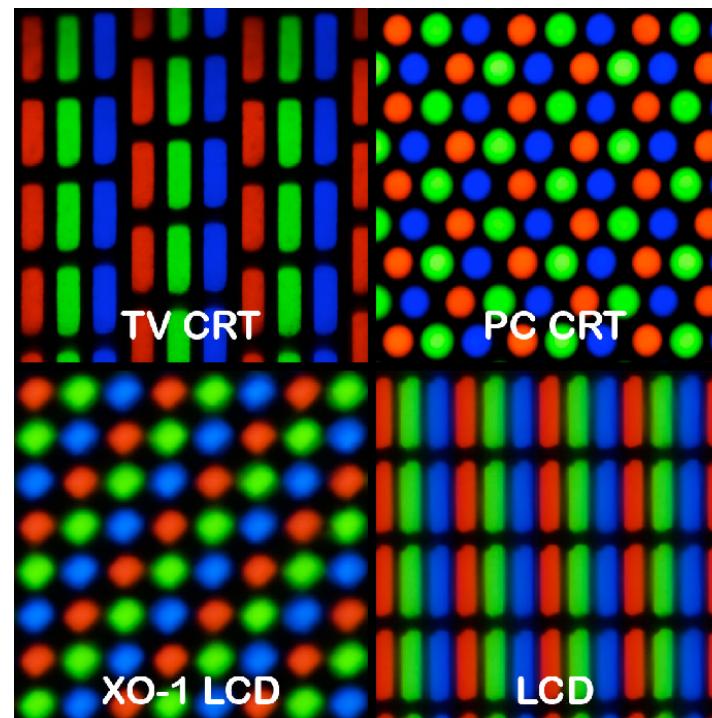


Red-Green-Blue (RGB)

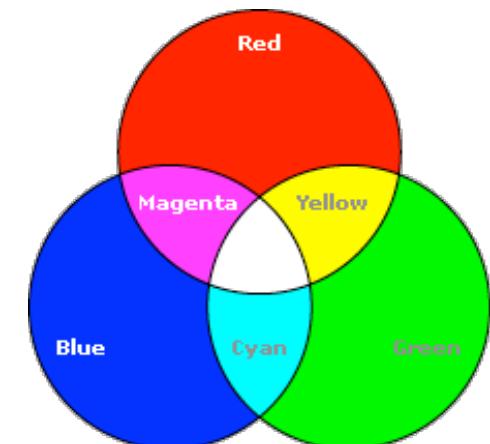
Pixels



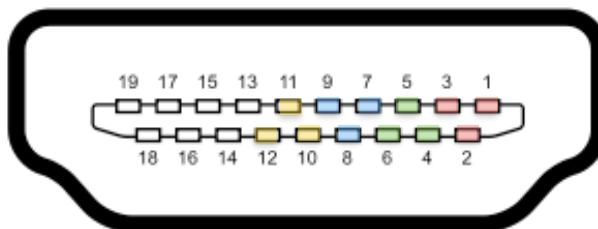
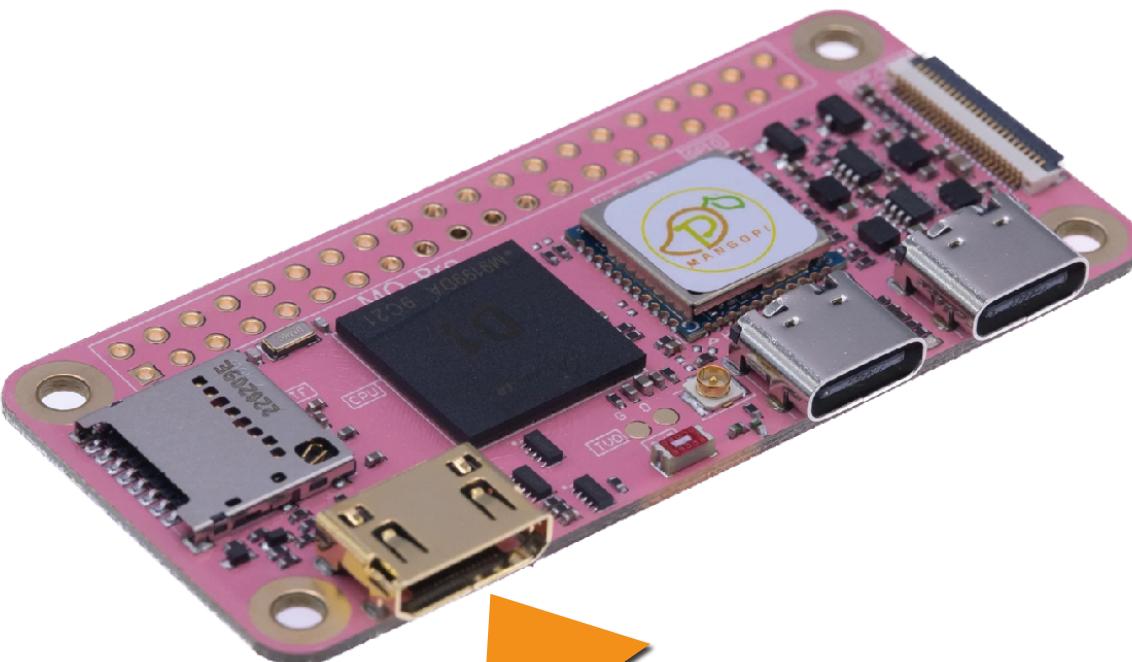
Displays



Light



Mango Pi HDMI



Clock
Data 0
Data 1
Data 2
Control

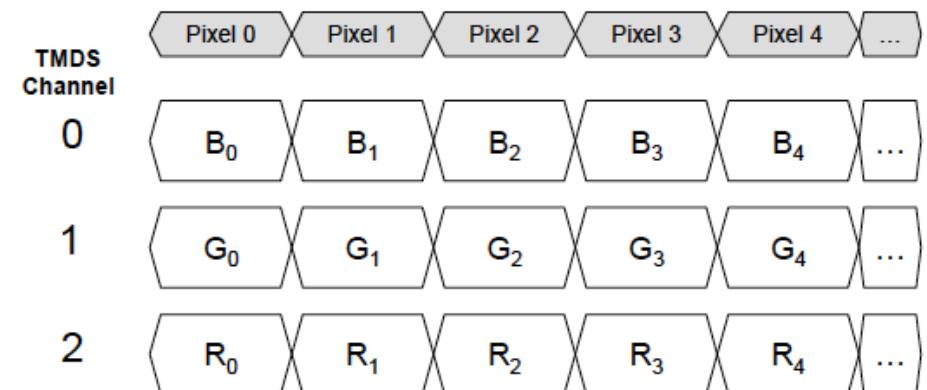
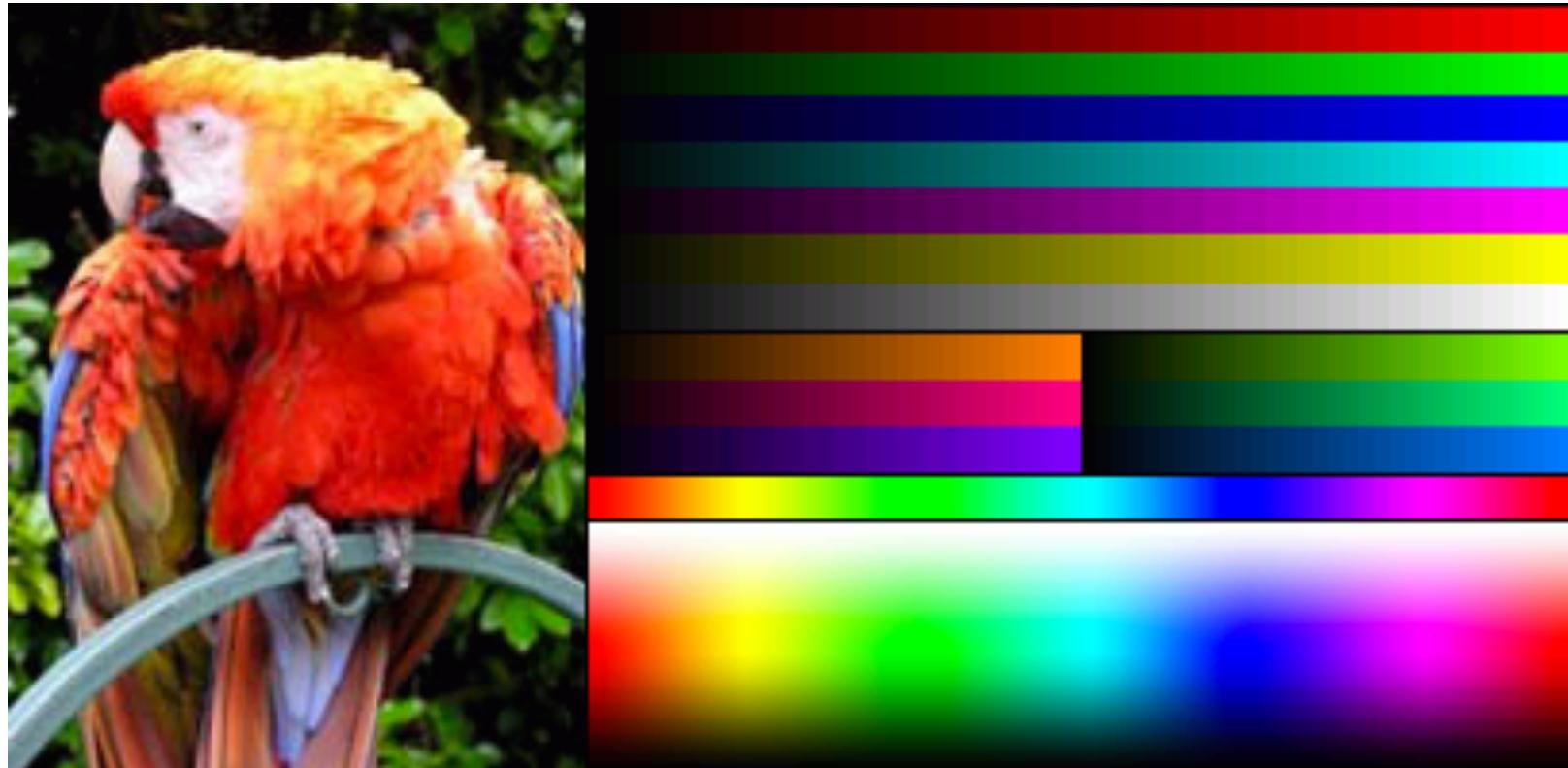


Figure 6-1 Default pixel encoding: RGB 4:4:4, 8 bits/component

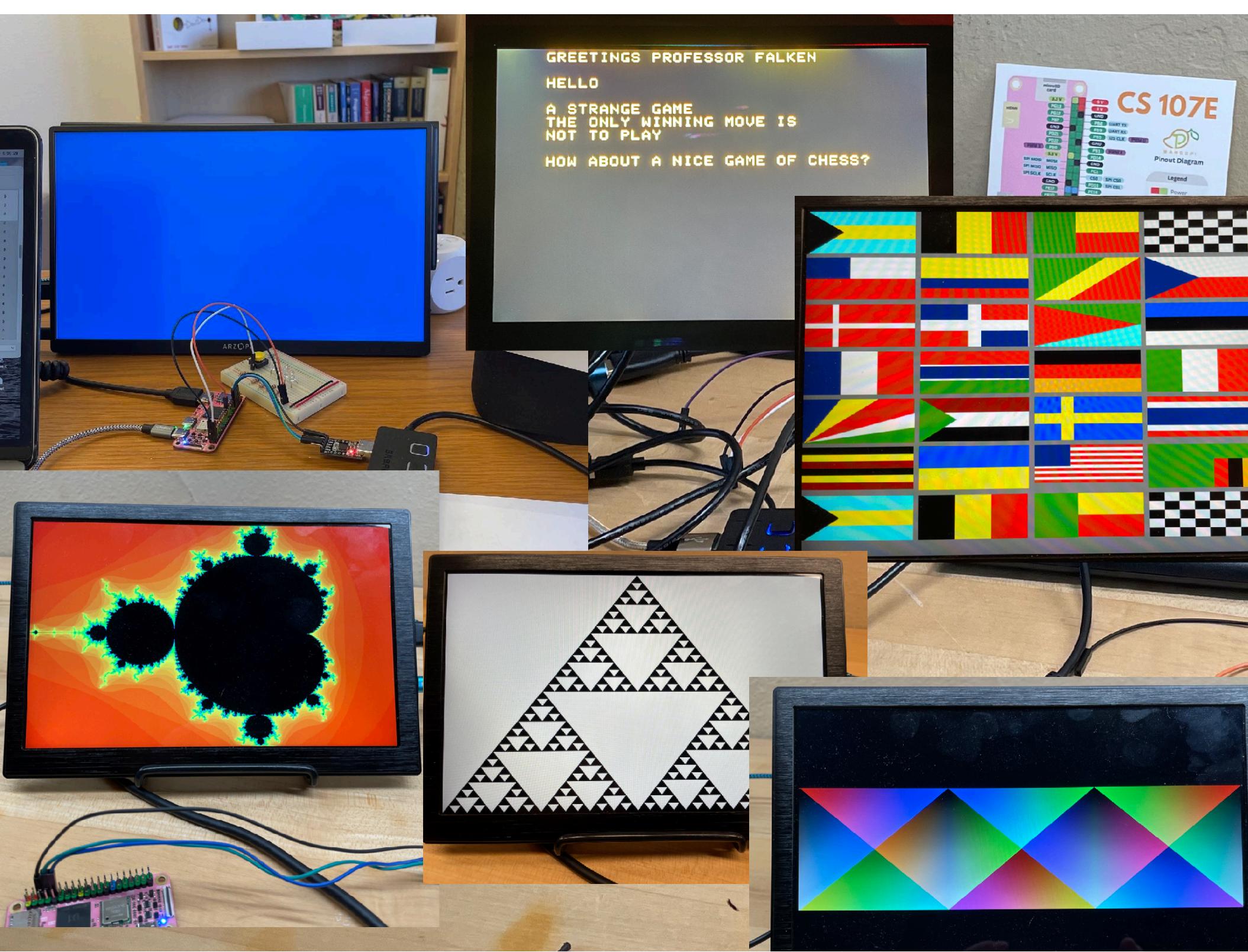
Figure from
High-Definition Multimedia Interface
Specification Version 1.3a

What is a frame buffer?



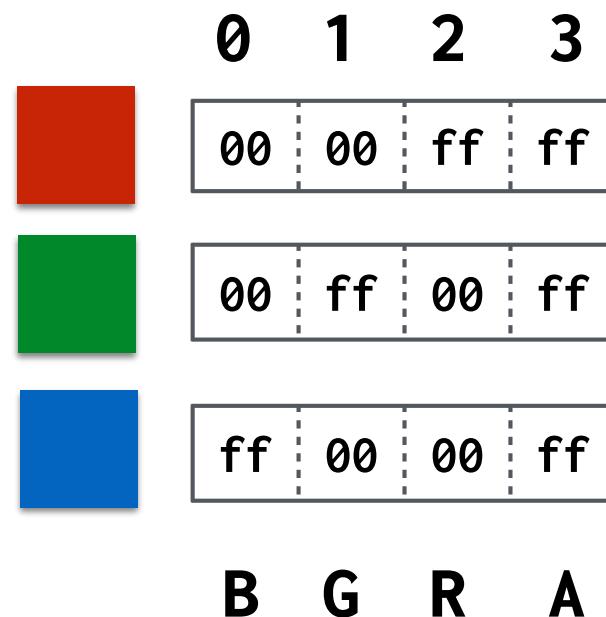
Framebuffer memory stores image
Image is a 2-D array of pixels

RGBA pixel depth 32-bit
8 bits each of red, green, blue, alpha



A pixel stores a color

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



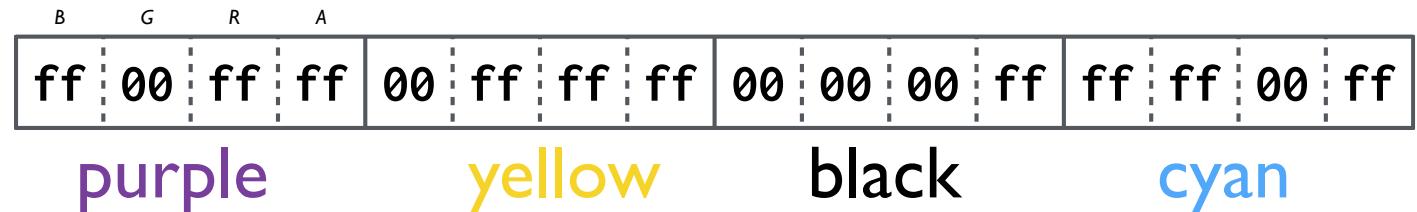
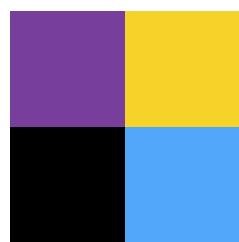
Take note: blue is first in memory (least significant byte)

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!)

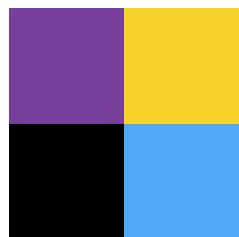
Tells hardware peripheral to display framebuffer

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  
...  
...
```



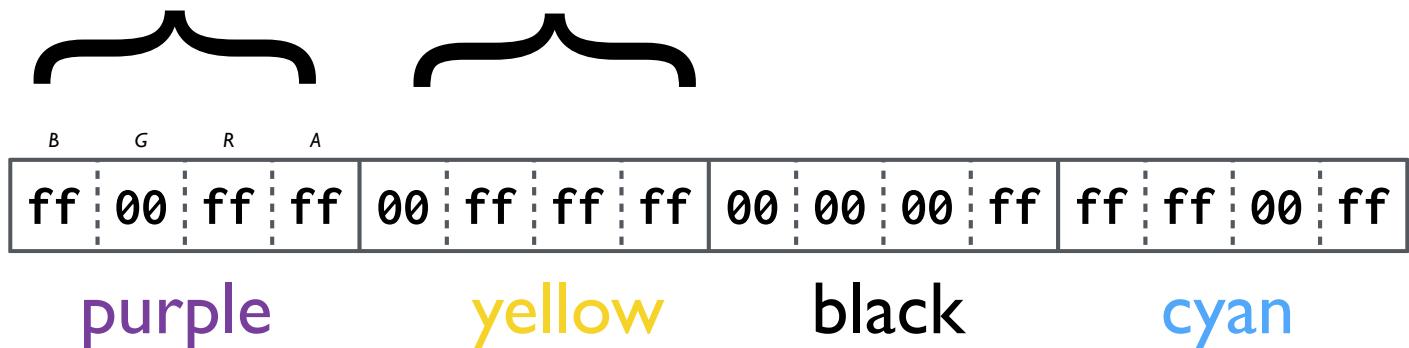
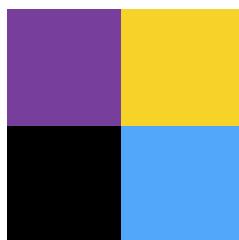
B	G	R	A	ff	00	ff	ff	00	ff	ff	ff	ff	ff	00	00	00	ff	ff	00	ff

purple yellow black cyan

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] = 0xffffffff00; // x=1, y=0  
fb[2] = 0xff000000; // x=0, y=1  
fb[3] = 0xff00ffff; // x=1, y=1
```



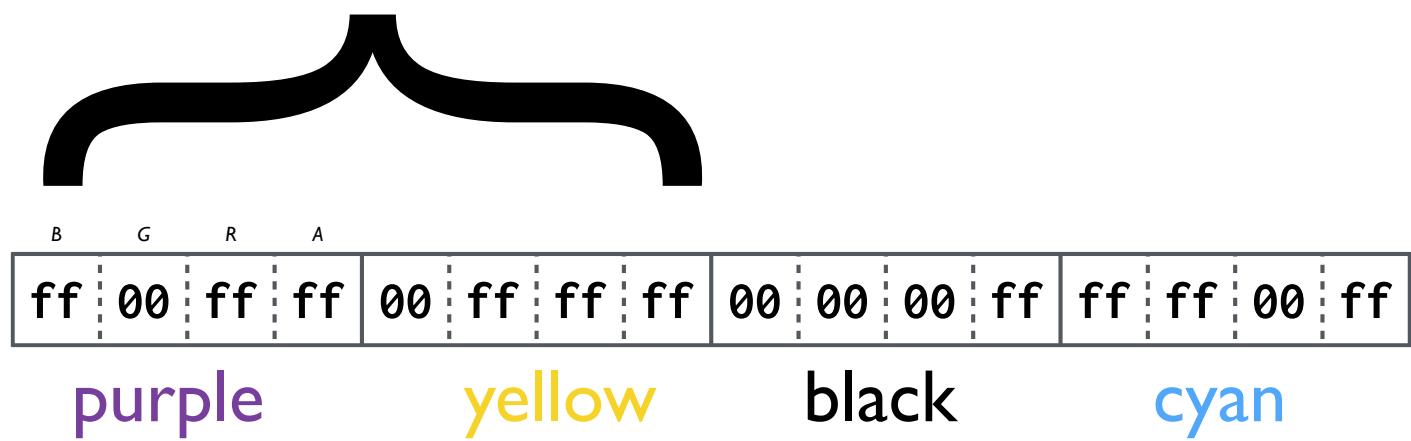
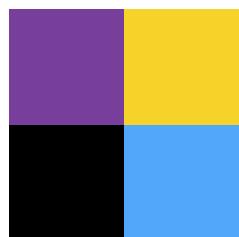
Memory is same size, contains same bytes as previous

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] = 0xffffffff00;  
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()

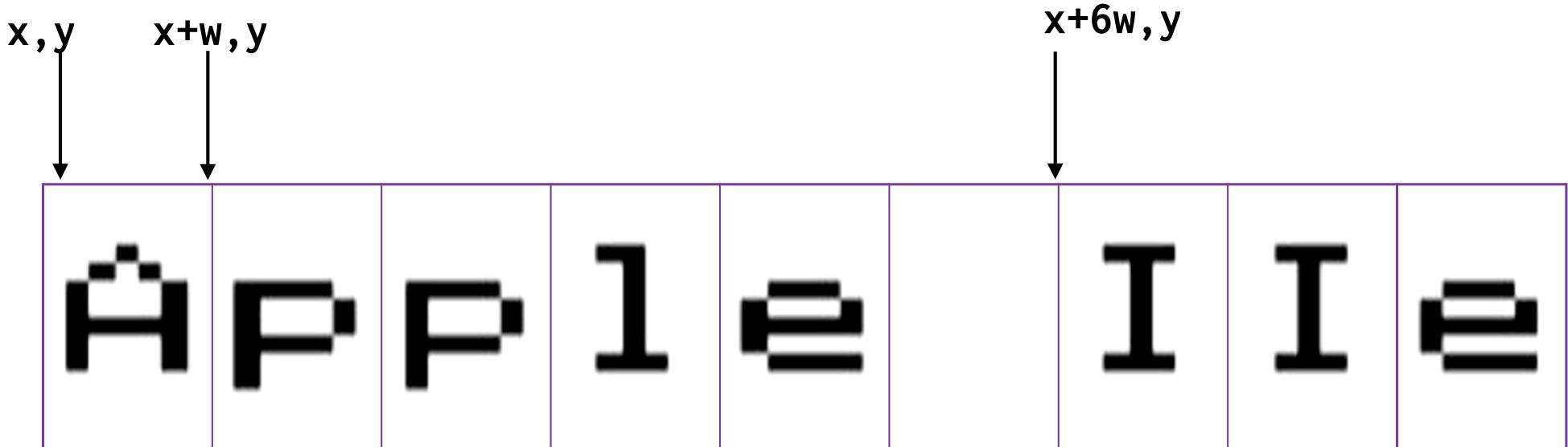
Review demo code: code/draw

Drawing text

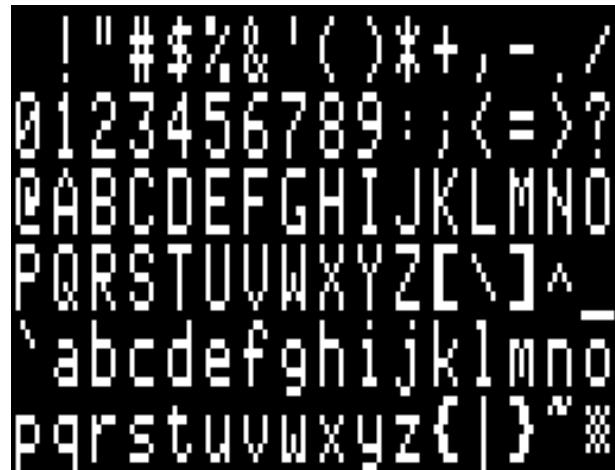
Font is a set of *glyphs*, one for each character

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



! "#\$%& ' () * + , - /
0123456789 : ; < = > ?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ [\] ^ _
' abcdefghijklmnpo
pqrstuvwxyz { | } " *

Original glyphs 7x8 pixels

Our font doubled to 14x16 pixels



= > ? @ A B C D E F

!"#\$%&'()*+,-./0123456789:; < = > ? @ ABCDEFGHIJKLMNOPQRSTUVWXYZ [\] ^ _ ' abcdefghijklmnpoqrstuvwxyz { | } " *

All font glyphs 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



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

front back



Display pipeline

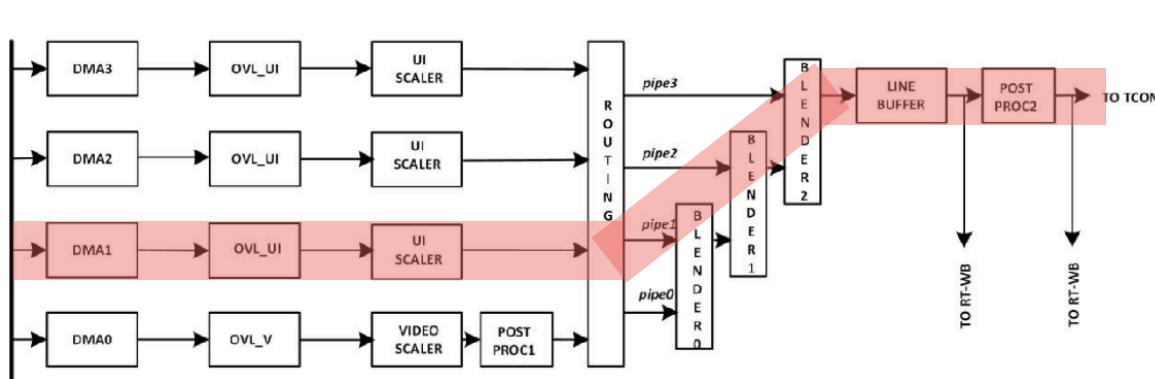
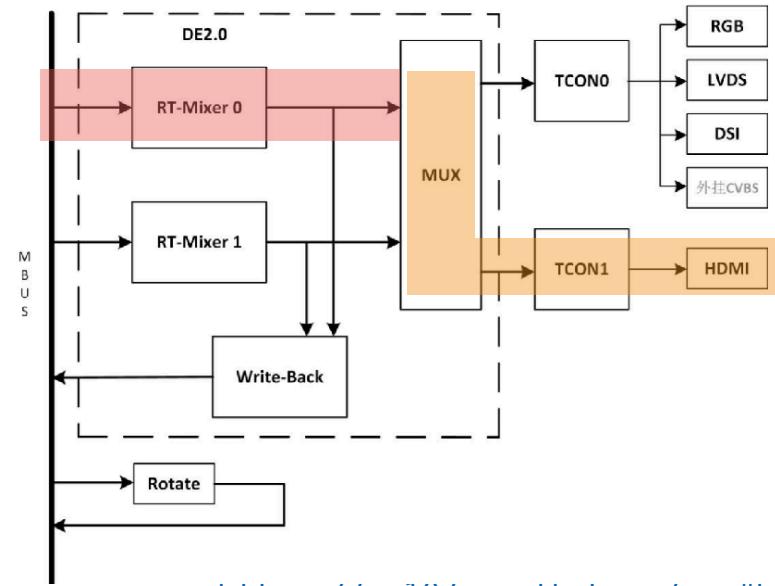


Figure 1-2. RT-MIXERO Block Diagram



<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

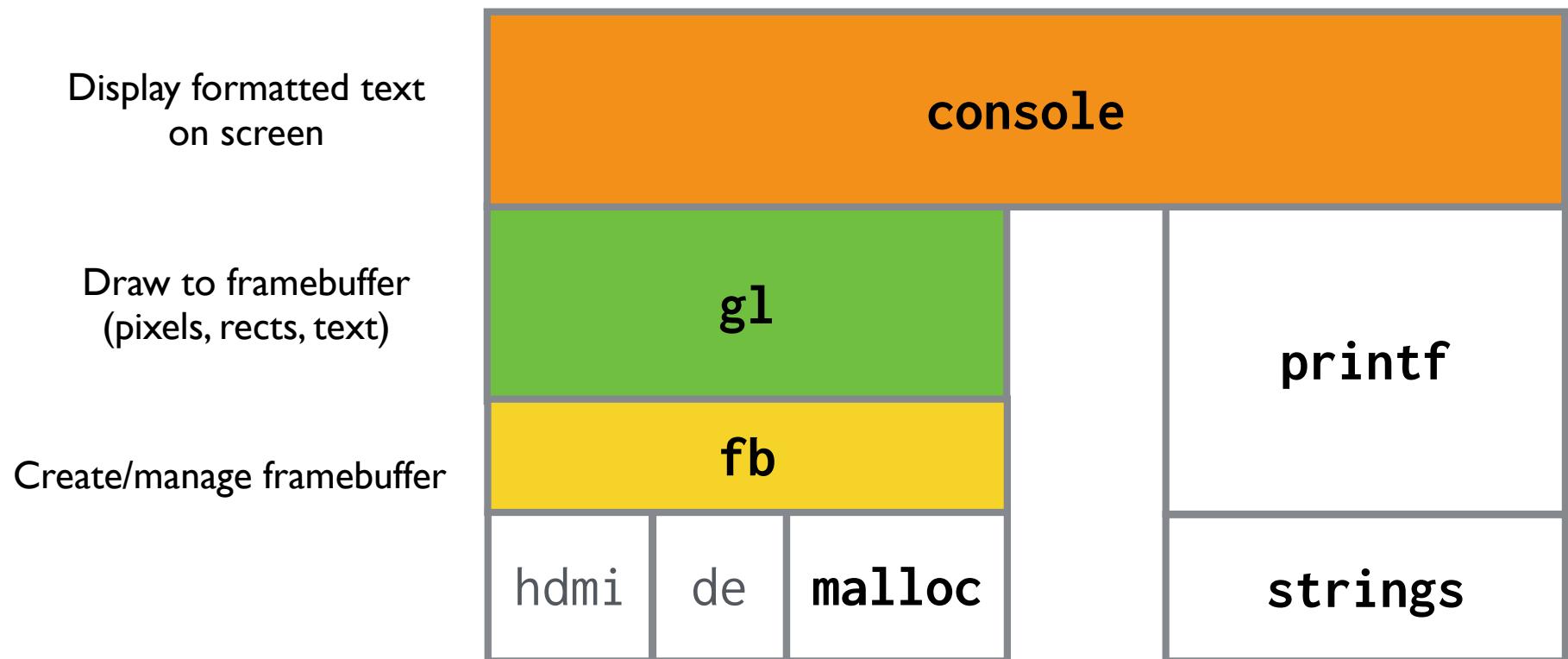
Scale up/down

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**

