

# **ECE 598 – Advanced Operating Systems Lecture 18**

Vince Weaver

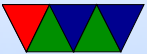
`http://web.eece.maine.edu/~vweaver`

`vincent.weaver@maine.edu`

5 April 2018

# Announcements

- Homework #9 will be posted (graphics)



# Graphics Interface History

- Teletypes
- Vector Displays
- CRTs
- LCD displays



# Video Display Technology

- Atari 2600 – Racing the Beam
  - 4k ROM, 128 bytes RAM, 40-pixel (5 byte) framebuffer
  - 3 sprites
  - all calculation done during the retraces
- SNES Tile/Sprite Based
  - RAM getting cheap enough can have framebuffers, but bandwidth still not that great.
  - Use tiles, that let you split the display into tiles, with each large tile specified by a single byte.

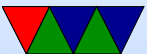


# Video Adapters – Framebuffers

- Just an array of bytes that get displayed on the screen.
- Bits per pixel
  - 1 – monochrome
  - 4 – 16 colors
  - 8 – 256 colors (usually palette)
  - 15 – rgb 555
  - 16 – rgb 565 “true color”
  - 24 – rgb 888
  - 32 – rgba



- Can be large:  $1024 \times 768 \times 24 \text{bpp} = 2.4 \text{MB}$ , to update at  $60 \text{Hz} = 141 \text{MB/s}$
- Bit-planes
- Palette

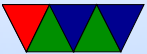


# Video Adapters – GPUs

- Draw lots of triangles, really fast
- Can divide screen into small sections, and calculate massively parallel.
- OpenGL/Direct3d
- Triangles
- Textures
- Z-buffers
- Shaders
- Can make a card with no framebuffer? Text just written



to texture and scaled to fill screen?





# VGA Display example

- VGA text
- Memory mapped, IO ports
- Mode setting
- VESA BIOS
- “Mode X”
- Bitplanes
- Colors
- Loadable fonts



# Linux graphic interface

- originally, none. VGA Text only  
X11 drove software directly.
- Attempt at GGI/KGI, Linus nixed it
- Framebuffer devices got in. Why? Well some machines had no textmode without it
- Gradually the DRI interface (Direct Rendering Interface) started providing abstractions needed for modern video



cards.

DRI1/DRI2/DRI3

DRM – event queueing?

KMS – kernel mode setting

GEM/TTM – memory allocation

MESA3D – handles OpenGL translation



# Higher Level

- X11 – client/server, network transparent  
MIT, 1984
- Wayland – Compositing Manager is mandatory  
Draw to an offscreen buffer, window manager copy to screen  
Can have 3d compositor, fancy effects



# Even Higher

- Libraries like Qt, Gtk, (historically Motif)
- Desktops like KDE, GNOME, XFCE



# Raspberry Pi Framebuffer

- Pi *can* do advanced 3D GPU graphics.  
Not documented well (but getting better)  
But it is complex, more than we need for a simple OS
- The GPU firmware does provide for a simple flat framebuffer mode if you ask it nicely



# Pi GPU interface

<http://petewarden.com/2014/08/07/how-to-optimize-raspberry-pi-code-using-its-gpu/>

<https://github.com/raspberrypi/firmware/wiki/Mailboxes>



# Raspberry Pi Mailbox Interface

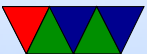
- How the ARM CPU communicates with the GPU that really run things
- Mailbox channels

MAILBOX_POWER	0
MAILBOX_FRAMEBUFFER	1
MAILBOX_VIRT_UART	2
MAILBOX_VCHIQ	3
MAILBOX_LED	4
MAILBOX_BUTTONS	5
MAILBOX_TOUCHSCREEN	6
MAILBOX_PROPERTY_TO_VC	8
MAILBOX_PROPERTY_FROM_VC	9
- Property tags contains a lot of the stuff we get from devicetree





- Temperature
- Clocks
- DMA
- Graphics



# Raspberry Pi Mailbox Interface

- Two mailboxes starting at 0x3f00b880. As Pi, always read/receive on Mailbox 0 and write/send on Mailbox1

Mbox0 Offset	Mbox1 Offset	Size	Name	Description	R/ W
0x00	0x20	4	Read/Write	Get/Send Mail	R/W
0x10	0x30	4	Poll/Peek	Check mail	R
0x14	0x34	4	Sender	Sender info	R
0x18	0x38	4	Status	Infor	R
0x1c	0x3c	4	Config	Settings	RW

- to send to a mailbox:
  - sender waits until the Mailbox1 Status field has a 0 in the MAIL\_FULL bit
  - sender writes to Write such that the lowest 4 bits are the channel to write to, and the upper 28 bits are the



message to write.

How can you make the address of the message have the bottom 4 bits be zero? (align directives)

- To read a mailbox:
  - receiver waits until the Mailbox0 Status field has a 0 in the MAIL\_EMPTY
  - receiver reads from Read.
  - receiver confirms the message is for the correct mailbox, and tries again if not.
- Talk to GPU through this mailbox interface. Lots of things set in it (the GPU is in control on Pi). Things



like power, clock enables, etc.



# Raspberry Pi Framebuffer Interface

- You can send it an address to a piece of memory to use as a framebuffer and it will draw it to the screen over HDMI.

- ```
struct frame_buffer_info_type {  
    int phys_x,phys_y;           /* IN: Physical Width / Height*/  
    int virt_x,virt_y;         /* IN: Virtual Width / Height */  
    int pitch;                 /* OUT: bytes per row */  
    int depth;                 /* IN: bits per pixel */  
    int x,y;                   /* IN: offset to skip when copying fb */  
    int pointer;               /* OUT: pointer to the framebuffer */  
    int size;                  /* OUT: size of the framebuffer */  
};
```

- Write the address of FrameBufferInfo + 0xC0000000 to mailbox 1 (C0000000 is a mirrored part of the address



space that is not cached)

Read the result from mailbox 1. If it is not zero, we didn't ask for a proper frame buffer.

GPU firmware returns a framebuffer you can write to.

Copy our images to the pointer, and they will appear on screen!

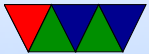


# Using a Framebuffer

- How big is it?
- Why might it not just be  $X*Y*(bpp/8)$  bytes big?  
Alignment issues? Powers of two? Weird hardware reasons?
- Things like R/G/B order, padding bits, bits grouped together (on Apple II groups of 7 bytes), etc
- Otherwise it's just an exercise in calculating start address and then copying values



- How do you calculate colors?





# Putting a Pixel

- Depends a bit on the graphics mode you request
- For simplicity, request 800x600x24-bit
- Get back pointer, size, pitch
- Each X row has B,G,R bytes repeated for each pixel
- To get to next row increment by pitch value (bytes per row)  
$$fb[(x*3)+(y*pitch)]=b$$



$$\text{fb}[(x*3)+(y*\text{pitch})+1]=g$$

$$\text{fb}[(x*3)+(y*\text{pitch})+2]=r$$

- pitch returned by the GPU. Normally it would just be  $(\text{maxy}*\text{bpp})/8$ , but it can vary depending on how the hardware arranges the bits.



# Drawing a Gradient

- Just draw a horizontal line, incrementing the color for each line



# Console Display

- Font / VGA Fonts
- console framebuffer. Color?
- scrolling
- backspace
- ANSI emulation



# Bitmapped Font

- Each character an 8x8 (or 8x16, or similar) pattern

```
● unsigned char smiley[8]={
    0x7e,  /*      * * * * *      */
    0x81,  /*     *           *   */
    0xa5,  /*    * * * * *    */
    0x81,  /*     *           *   */
    0xa5,  /*    * * * * *    */
    0x99,  /*     *  **      *   */
    0x81,  /*     *           *   */
    0x7e,  /*      * * * * *      */
};
```

```
void put_smiley(int xoff, int yoff, int color) {
    for(y=0;y<8;y++) {
        for(x=0;x<8;x++) {
            if (simley[y]&(1<<(7-x))) {
                putpixel(color,x+xoff,y+yoff);
            }
        }
    }
}
```



}  
}  
}  
}

- Can find source of fonts online, VGA fonts. Just a binary set of bitmapped characters indexed by ASCII code.
- Usually 8x16 though; the custom font used in the homework is a hand-made 8x8 one

