ECE 531 – Advanced Operating Systems
Lecture 21

Vince Weaver

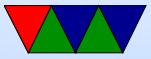
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

14 November 2023

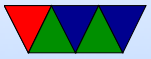
Announcements

- Homework #7 will be posted
- Project Progress Report Due (extended until Monday the 20th)
- Midterm after Thanksgiving, tentatively moved to Thursday November 30th



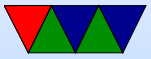
Notes on Homeworks

- They've been delayed due to a few problems
  - elf2bflt was broken (since fixed)
  - Pi4 interrupt support (since fixed)
  - Mysterious problem on Pi4 with icache enabled (still working on)
  - Pi4 mailbox/graphics support also broken
- I will post some updated code that people can use for projects



Graphics Interface History

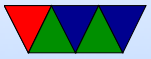
- Teletypes
- Vector Displays
- CRTs
- LCD displays



Video Display Technology

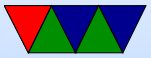
- Atari 2600 – Racing the Beam
  4k ROM, 128 bytes RAM, 40-pixel (5 byte) framebuffer
  3 sprites
  all calculation done during the retraces

- 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.

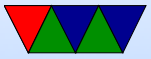


Video Adapters – Framebuffers

- 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

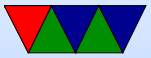


- Can be large: 1024x768x24bpp = 2.4MB, to update at 60Hz = 141MB/s
- Bit-planes
- Palette



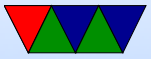
VGA Display example

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



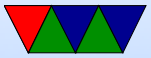
Video Adapters – GPUs

- 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?

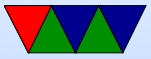


Modern GPUs

- Massively Parallel
- Instead of fixed pipelines, programmable shaders
- 3D shaders
  - Vertex shader, called on each vertex, can calc texture, convert to 2D, change position
  - Geometry shader, can make new vertices
  - Tesselation – makes meshes
  - Ray Tracing
- 2D shaders



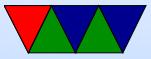
- Pixel (Fragment) Shaders - called on each pixel, can do things based on co-ords, nearby values
- Compute
  - GPGPU – general purpose code can be offloaded to GPU



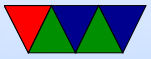
VideoCore GPUs

- In Pi. Old ones IV, newer VI?
- OpenGL ES 2.0 V3D
- Output to HDMI, DSI, DPI, composite
- In theory can do compute shader jobs, but no Linux interface
- QPU – 16-way SIMD (suited for calculating quad (x4) data

  In theory can use mailbox to submit QPU jobs
- Can run general purpose code unlike some GPUs

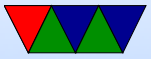


On Pi runs ThreadX, full OS by itself



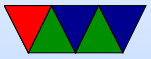
Programming VideoCore

- Broadcom eventually released enough info for a full open-source Linux driver
- V3Dlib (?)
- videocoreiv on github, code to write baremetal simple OS running only on GPU



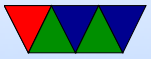
Linux graphic interface

- 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

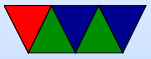
- DRI1/DRI2/DRI3
- DRM – event queueing?
- KMS – kernel mode setting
- GEM/TTM – memory allocation
- MESA3D – handles OpenGL translation



Higher Level

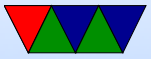
- X11 – client/server, network transparent
  MIT, 1984

- Wayland – Compositing Manager is mandatory
  Draw to an offscreen buffer, window manager copy to screen
  Can have 3d compositor, fancy effects



Even Higher

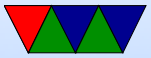
- Libraries like Qt, Gtk, (historically Motif)
- Desktops like KDE, GNOME, XFCE



Raspberry Pi Framebuffer

- Pi *can* do advanced 3D GPU graphics.
  But it is complex, more than we need for a simple OS

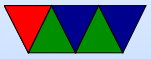
- The GPU firmware does provide for a simple flat framebuffer mode if you ask it nicely



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

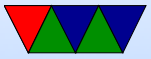


Raspberry Pi Mailbox Interface

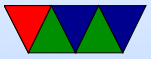
- How the ARM CPU communicates with the GPU that really run things
- Mailbox channels

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

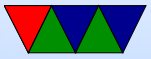


Raspberry Pi Mailbox Interface

- Two mailboxes in MMIO space (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

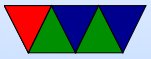
- to send to a mailbox:
  - sender waits until the Mailbox1 Status field has a 0 in the `MAIL_FULL bit`
  - sender writes to Write such that the lowest 4 bits are



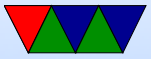
the channel to write to, and the upper 28 bits are the message to write.

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

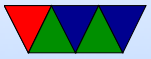


things set in it (the GPU is in control on Pi). Things like power, clock enables, etc.



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

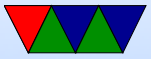


space that is not cached)

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

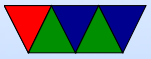
GPU firmware returns a framebuffer you can write to.

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



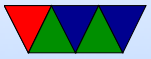
Using a Framebuffer

- 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 is calculating start address and then copying values
- How do you calculate colors?



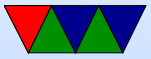
Putting a Pixel

- 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 (the ordering changed at some point with a firmware release)
- To get to next row increment by pitch value (bytes per row)



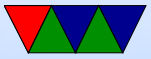
```
fb[(x*3)+(y*pitch)]=b
fb[(x*3)+(y*pitch)+1]=g
fb[(x*3)+(y*pitch)+2]=r
```

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



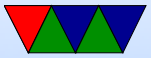
Drawing a Gradient

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



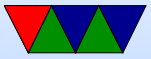
Console Display

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



Fonts

- How do you convert an ASCII character to a pixel pattern?
- Fonts!
- Fancy fonts have vectors, spacing info
- Can have full programming language to calculate this as well as to help scale for different sizes
- We want something simpler, just a pure bitmap font

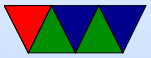


Bitmapped Font

- 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