ECE 177 – Programming I: From C Foundations to Hardware Interaction Lecture 23

Vince Weaver

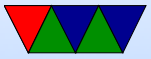
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

30 March 2026

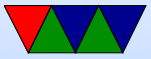
Announcements

- Lab#7 this week
- Sorry behind on things, will try to catch up
- HW#6 not posted yet, need to run through it
- Reminder midterm the 10th



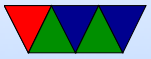
Lab7 – Graphics and Sound

- In this lab we will do some graphics on the TFT display
- We will also do some sound. Hopefully that's not too annoying.
- This is a new lab, completely different from last year (which was a calculator) so the TAs won't be as familiar with it



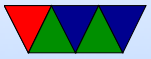
Lab7 – ST7789 TFT LCD Driver

- Sitronix ST7789
- On-chip 240x320x18 bit color 1382400 bits, ~168kB
- Can handle 12bpp (RGB=444), 16bpp (RGB=565), 18bpp (RGB=666)
- Various interfaces, we use SPI-like interface (has extra pin for command vs data)
- Can mirror data various ways (how we handle rotated)
- Can partial display or scroll
- Tearing effect (to avoid tearing?)



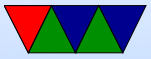
Lab7 – Init the Graphics

```
Adafruit_ST7735_gpio_init();            // initialize GPIOs
Adafruit_ST7735_initBlue();             // init our board
Adafruit_ST7735_setRotation(2);         // set portrait mode
Adafruit_ST7735_fillScreen(0xFF0F);     // makes yellow
                                        // 0xFCE070
graphics_init(240,320);                 // init graphics
graphics_setTextWrap(false);            // disable line wrap
graphics_setTextColor(0xFFFF);          // white text
graphics_setTextSize(2);                // twice size on font
                                        // for 20x20 screen
```



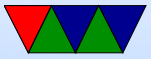
Lab7 – Framebuffer

- Picture an array, each element being a pixel
- “modern” assume 24-bit color, RGB. So you might view it as an array of 3-byte structs



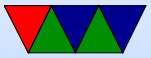
Lab7 – ST7789 Framebuffer

- the ST7789 does thing a bit differently
- Programmed via registers set via SPI
- Set a “window” you want to draw into RAM, with X,Y and width/height
- Then you write the 16-bit colors to fill that window (it wraps)



Lab7 – Colors

- Modern days might be used to 24-bit color RGB, often in hex, one byte each
- In theory LCD supports 18-bit color (RGB = 666)
- Adafruit libraries assume 16-bit color, oldschool RGB 565 bits.
  - Black is 0x0000
  - White is 0xffff
  - Red is 0xf800
  - Green is 0x07e0

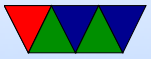


- Blue is 0x001F

`rrrr rggg gggb bbbb`

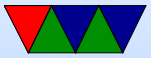
- Can convert from RGB using provided routine:

```
uint16_t Adafruit_ST7735_Color565(uint8_t r,
                uint8_t g, uint8_t b)
```



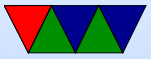
Lab7 – Adafruit Graphics Library

- Provided code that implements graphics for us
- It appears to be under a BSD license



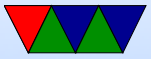
Lab7 – `graphics.c / .h`

- Line drawing (horiz, vert, Bresenham)
- drawcircle (regular and filled) (also Bresenham)
- drawRect / fillRect
- drawChar / write / cursor / font / textColor



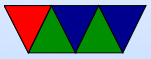
Lab7 – `Adafruit_ST7735.c / .h`

- Seems to be MIT license?
- Actually handles various types of displays, including our ST7789 which it calls the “blue” display for some reason
- SetAddrWindow sets “window” in RAM, then you write to it and it wraps.
- Has putpixel routine plus the fast rectangle / line routines



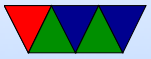
Lab7 – `tft_stdout.c / .h`

- Code by Sheaff?
- Code the overrides “stdout” with output to lcd screen
- Erases and re-draws complete screen for each write, which is why it flickers so much



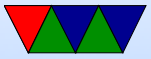
Lab7 – `glcdfont.c / gfxfont.h`

- 5x7 bitmap font



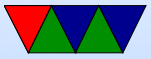
Lab7 – Drawing a Ball

- For now just drawing a rectangle
- Could we use filled-circle routine instead?
  Guessing it would be much slower, but didn't try



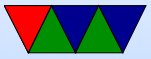
Lab7 – Sprites/Blitting

- What if you want an arbitrary shape?
- You can have a bitmap that is an array holding the colors for a shape you want to draw
- Sometimes these are called “sprites” and drawing them is called “blitting” (aside, some old computers could do this in hardware and people to this day get annoyed if you call the software version sprites)
- Generally to blit a sprite you’d just have two loops, x and y, to walk through the array and draw the pixels in



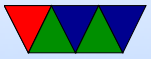
a rectangle

- What if you want transparency? Usually you pick a color that means transparent, and have an if statement, if it matches the color don't draw it so the background shows through
- What if your graphics hardware has more than one pixel per byte? Then you might need to have a separate mask and sprite, and use AND with the mask to clear out the background data and OR to put the sprite in place. bitwise masking and bit manipulation
- If boring black background could use XOR instead



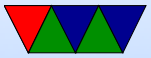
Lab7 – Erasing the Ball

- Since we have a plain background, just draw a rectangle of the background color to the same co-ordinates
- (aside, if doing sprites and you used xor to draw, you can xor again to restore things)



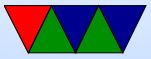
Lab7 – Moving a Ball

- Things to track
  - x and y location
  - xsize and ysize of your object
  - xadd, yadd – speed, how fast it moves, but also direction
- Each frame add xadd to x, and yadd to y
- How do you reverse direction?
- Just flip sign of xadd and yadd

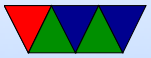


Lab7 – structs

- To give you practice with structs I have you use a struct to hold the ball information
- Why are structs good here?
  - Group together all the info for the ball so easier to know they belong together
  - If you have some other object later that also needs x, y, etc you don't have to worry about variable name conflict/ confusion
  - If you need to pass the ball info to a function you can

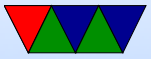


just pass the struct rather than all the individual items



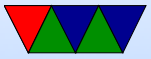
Lab7 – Collision Detection

- Complicated
- Often ends up being a lot of compares
- There are some fancier ways to do it (hardware? etc?) but Pi Pico is fast enough the lots-of-compares method works
- Hitboxes are used if you have more complex sprites you are drawing



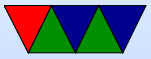
Lab7 – Wall Collision

- We're just doing walls here
  - So if `x<0` it means you hit the wall to the left, so collide and flip X direction
  - If `x>=XSCREEN-XSIZE` then you hit the wall on the right
  - Do same for Y (top/bottom of screen)



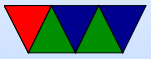
Lab7 – Framerate

- Do you want to run at maximum possible speed? Why not?
- Frame limit. Limit to 60fps or 30fps. How?
- For example, want 50fps. so $1/50$. Is 0.020s, or 20ms.
- So we can just use `sleep_ms(20);` and the max possible speed you can run is 50fps, no matter what platform you run on
- Notice if you are doing a lot during this frame it can actually be slower than this.

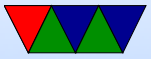


Lab7 – Unlimited Framerate Aside

- How can you have arbitrary frame rate but not have game run too fast too slow?
- Adjust speed on the fly
- Each frame, calculate how long it took
- Them when adding, instead of xspeed being 1 or -1, scale it by how long it actually took
- This is more complex, also needs floating/fixed point
- Also if your adjusted speed is too big can “glitch” through walls and such if the speed is wider than the

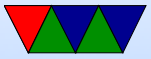


hitbox of enemies and walls. You can adjust for this too but it's more work

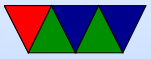


Lab7 – Flicker/Screen Updates

- How could you avoid flicker?
- Wait until refresh (tearing) to only draw when not being updated
- Only erase for minimal amount of time (our big problem here as we do a decent amount of stuff between erase/draw. We could move the erase until after the calcs but then we'd have to save the old x/y coords which is an extra step)
- page-flipping / double buffering– ideally device would

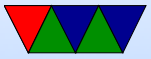


have two pages of RAM and you can draw to the off-screen one, then instantly flip. more resources to manage



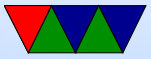
Lab7 – Sound

- How do you make sound?
- Motion of the air
- Speakers, often magnet and a coil pulling paper back and forth, can make the complex sound waves in the air
- The ones on our board are sort of weird, maybe piezo speakers? Not actually sure

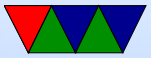


Lab7 – Generating Sound

- Generally you use a digital analog converter, need to convert digital data in a computer to analog
- Sounds on computers are sampled, take the complex wave forms, sample them every so often (44kHz? Nyquist Theorem?) with a digital-analog converter (DAC) and then record what they are at 8-bits, 12-bits, 16-bits, etc
- You usually want to compress them because 16-bits (maybe 8-bit audio in stereo) times 44,000 samples a

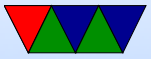


second adds up fairly quickly



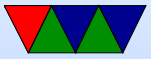
Lab7 – What if no DAC?

- Pi Picos (as well as real Pis) don't have DACs
- How can you play sound?
- You can definitely play square waves with GPIOs. What do those sound like?
- Aside: AY-3-8910 chiptune music made of 3 square waves



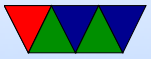
Lab7 – What if you want more than just square wave?

- Can use PWM to approximate actual notes
- Essentially by varying how long it is on, the "average" signal is from 0-100% so works out sort of like a DAC would
- Depends a bit on the capacitance of speaker to smooth it out



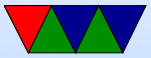
Lab7 – PWM

- Pulse-width modulation
- You could turn the GPIO on and off carefully timed 40k times a second
- Really hard to do exactly in software, especially on modern CPUs
- If there's a hardware timer you can offload this (Especially if it's a fixed tone) and the hardware will toggle the pin for you so you don't have to
- You just need to set it up and tell it when to start and



stop

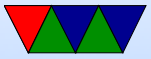
- You'll see this more in ECE271



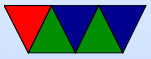
Lab7 – PWM on pico

- Good reference on this here: https://www.i-programmer.info/programming/hardware/14849-the-pico-in-c-basic-pwm.html
- I provide code so all you need to do is turn on/off
- Pi-Pico has 8 PWM generators, each with two channels

```
/* init */

/* Set GPIO18 to be used as PWM */
gpio_set_function(28,GPIO_FUNC_PWM);

/* GPIO28 is slide 6A (output 6, channel 0) */
/* we can use this routine to look that up automatically *
slice_number=pwm_gpio_to_slice_num(28);
```



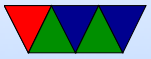
```
/* Pi Pico input clock is 125MHz */
/* This divides this by 255 (to 488kHz */
pwm_set_wrap(slice_number,255);

/* The above makes it count to 255 and start over */
/* These make the output channel go from 0 to 1 once */
/* the counter hits 128, so giving a duty cycle of 50% */
pwm_set_chan_level(slice_number,PWM_CHAN_A,128);
pwm_set_chan_level(slice_number,PWM_CHAN_B,128);

/* Finally this divides the 488kHz clock by 255 */
/* Which gives 1914Hz.  Closest note is B6 (1975Hz) */
pwm_set_clkdiv_int_frac(slice_number,255,0);
```



Lab7 – Sound Disabling

- Does the sound get annoying after a while?
- Ideally would have a keypress to turn off. Next Lab. How? Have a global "sound-enabled" variable and set it to 0 if want off, then in the `play_sound()` routine just return early if 0
- Can also right now temporarily disable with

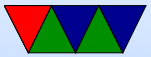
```
#define SOUND_DISABLED 1     // have at top

#if !SOUND_DISABLED
    the sound code
#endif
```



Lab7 – Routine to turn on/off

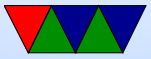
- I have you put this inside a function

```
pwm_set_enabled(slice_number,true);
sleep_ms(length);
pwm_set_enabled(slice_number,false);
```



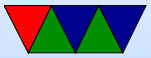
Lab7 – Pre-processor

- We have settings for `XMAX` and `YMAX`
  Can change them easily if move to another board
- We have settings for `XSIZE` and `YSIZE`
  We will use them to change size of ball
- By having this central defines, and consistently using them, if we want to change things we can without having to search/replace for hard-coded constants



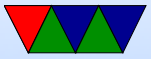
Lab7 – Function Prototypes

- Pre-declare the functions
- We can declare our function after
- I have you set up a function and call it



Lab7 – Static Functions

- What does static mean in this case?
- It's not about holding value across function calls, which is what it means for variables
- It means that this function is *only* called in this C file
- Compiler can optimize better because if knows no one else calls it
- Also can get warnign if you define a function but don't use it



Lab7 – Fun Extra Stuff

- Once you are done with the lab and submit it, there are other things you could do if you're enjoying the lab
- Have the ball change colors after each wall hit
- Have the ball change size after each wall hit (would have to make xsize and ysize variables instead of defines)
- Change the frequency of the sound, or have multiple sound variants
- Play music