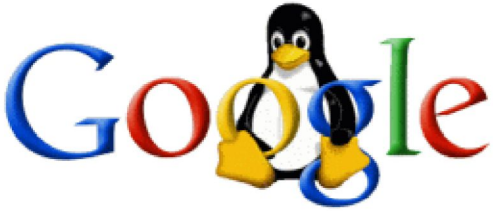
Linux.conf.au 2020



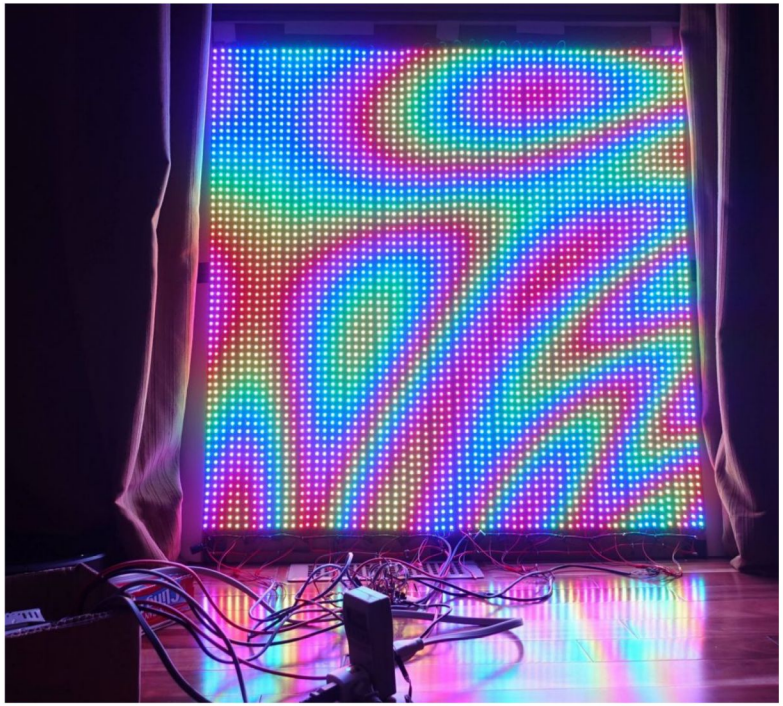
ESP32 Memory Management, Neopixels, and RGBPanels

Slides: http://marc.merlins.org/linux/talks/ESP32_Memory/

Marc MERLIN
merlin@google.com

ESP32, it was all fine when I started

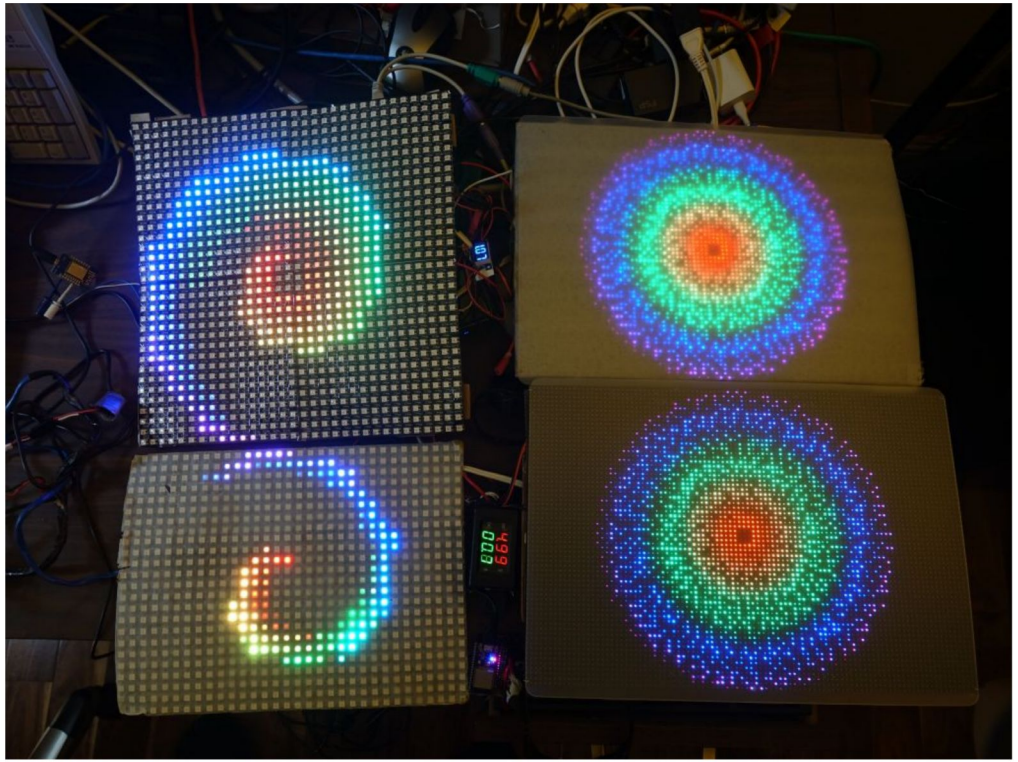
- I made a 64x64 neopixel array, 4096 pixels in 3 byte color, 12KB. Everything worked except one demo that crashed and I couldn’t find the bug in



Commercial Time: Framebuffer::GFX and SmartMatrix::GFX

I wanted to re-use all my code written for FastLED::NeoMatrix, on top of SmartMatrix

So I wrote SmartMatrix::GFX
Then I wrote a base class,
Framebuffer::GFX
+A multi API wrapper for Adafruit::GFX,
FastLED, and LEDMatrix



Commercial Time: FastLED_SPITFT::GFX

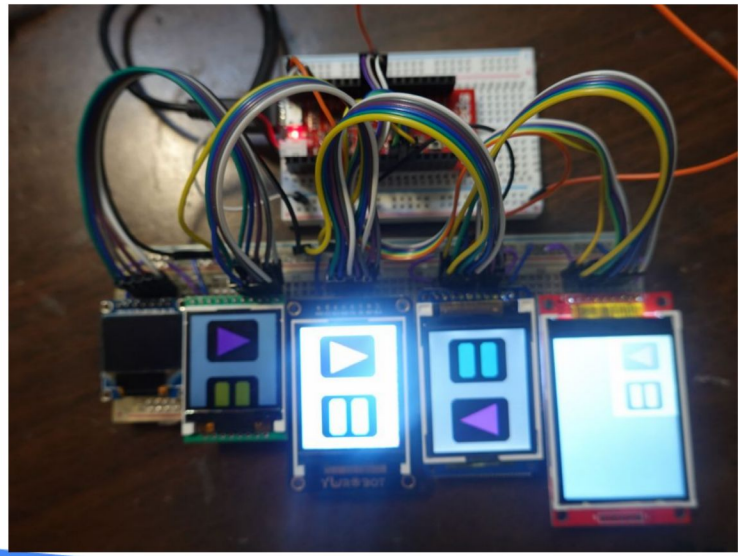
Once I had Framebuffer::GFX, I wrote other driver frontends

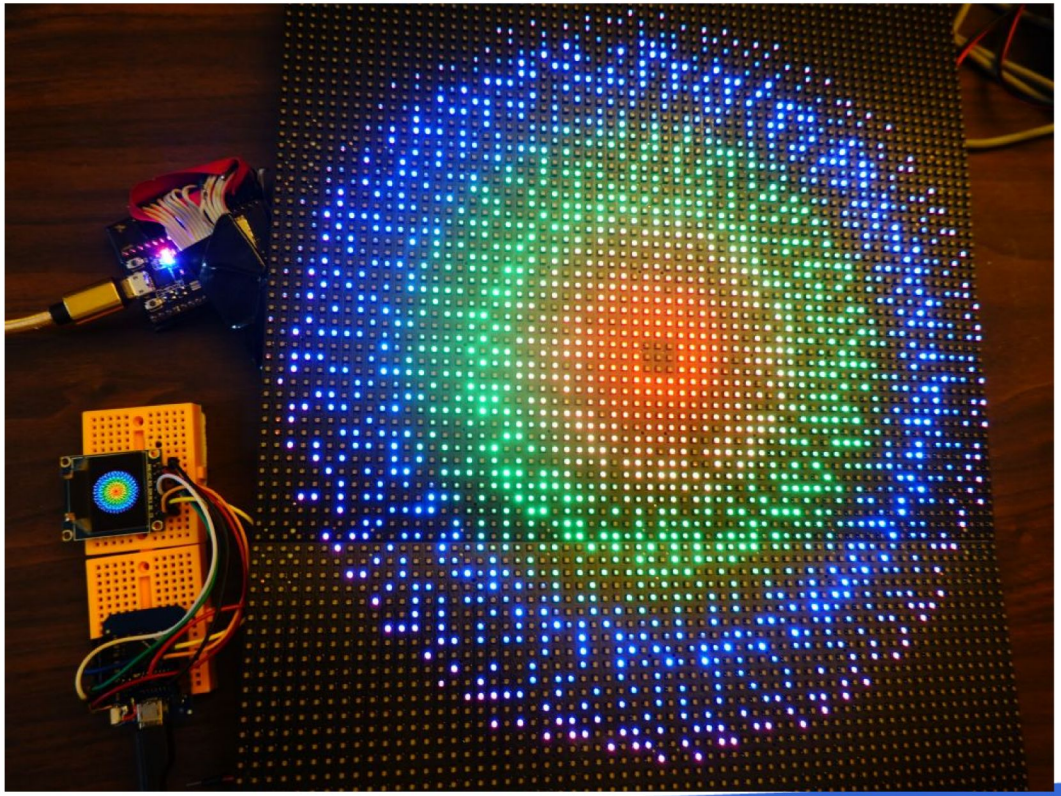
Including one for 3 types of TFTs

SSD1331 (96x64)

ST7735 128x128, ST7735 128x160

ILI9341 with a full 320x240





Then I switched to RGBPanels, they need a lot more RAM

My new 96x64 RGBPanel outfit uses RGBPanels, which require multiple pre-computed bitmap planes that are used for DMA

Suddenly I started getting all kinds of weird memory errors. ESP32 has 520KB, right?

ESP32, 520KB of RAM, really?

- RGBPanels 96x64 and TFTs at 320x240 blew up memory on ESP32
- 128*160*3 = 32KB, 320*240*3 = 225KB => only works on teensy
- You’re actually running a task
- You don’t get to see/use all the memory
- The memory isn’t contiguous
- Some allows 8bit access, some only allows 32bit access (IRAM, not usable for C arrays or basic malloc)
- And then there is PSRAM on some chips, which can be allocated via malloc, but not C arrays

You really get 2x 160KB (not quite, even)

https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/system/mem_alloc.html

- “Due to a technical limitation, the maximum statically allocated DRAM usage is 160KB” => this means global arrays
- “The remaining 160KB (for a total of 320KB of DRAM) can only be allocated at runtime as heap.” => malloc
- “At runtime, the available heap DRAM may be less than calculated at compile time, because at startup some memory is allocated from the heap before the FreeRTOS scheduler is started (including memory for the stacks of initial FreeRTOS tasks).” => not even 160KB

IRAM and non contiguous RAM

https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/system/mem_alloc.html

- Some memory in the ESP32 is available as either DRAM or IRAM. If memory is allocated from a D/IRAM region, the free heap size for both types of memory will decrease.

```
I (252) heap_init: Initializing. RAM available for dynamic allocation:
I (259) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (265) heap_init: At 3FFB2EC8 len 0002D138 (180 KiB): DRAM
I (272) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM
I (278) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (284) heap_init: At 4008944C len 00016BB4 (90 KiB): IRAM
```

Global Arrays can barely access 160KB

https://github.com/espressif/esp-idf/issues/3497 “Statically allocated DRAM is limited to 160KB”

https://github.com/espressif/esp-idf/issues/1934 “Internal RAM 520, allowed for allocation 301KiB ? Missing 219?”

- If your global array is too big, the links tells you “xtensa-esp32-elf/bin/ld: region `dram0_0_seg’ overflowed by 12040 bytes”

Your program may compile, but crash at runtime

https://github.com/espressif/arduino-esp32/issues/2567 “Compiler/linker should do a better job handling global arrays instead of crashing at runtime before setup() runs”

- https://github.com/espressif/esp-idf/issues/3211 “ESP32 code can be compiled with global static arrays that cause FreeRTOS crash at runtime”
- If your array is not too big, but just a little too big, the linker works, and then the code crashes at runtime because FreeRTOS tries to allocate memory too, and fails:

```
rst:0xc (SW_CPU_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
0x40088960: invoke_abort at /Users/ficeto/Desktop/ESP32/ESP32/esp-idf-public/components/esp32/panic.c line 140
0x40088b63: abort at /Users/ficeto/Desktop/ESP32/ESP32/esp-idf-public/components/esp32/panic.c line 149
0x40082b72: start_cpu0_default at /Users/ficeto/Desktop/ESP32/ESP32/esp-idf-public/components/esp32/cpu_start.c line 381
0x40082d04: call_start_cpu0 at /Users/ficeto/Desktop/ESP32/ESP32/esp-idf-public/components/esp32/cpu_start.c line 213
```

Only 8KB of stack

https://github.com/espressif/arduino-esp32/issues/2567#issuecomment-475057527

- Code that naively defined an array inside a function worked fine, until I grew the display size. Then, it started to crash nonsensically.
- Same code worked fine on teensy
- Took me a while to learn that it was because of stack smashing and the ESP32 stack is allocated for each task and is smaller (many chips only have one task and grow the stack down and the heap up, so there is more room for both)

Converting arrays to malloc

- Whether you’re dealing with stack smashing crashes
- Or code that won’t compile/run because of your big global array, you’re better off allocating via malloc in setup()
- It’s annoying when you use existing arduino code and libraries that work fine on other chips that have less RAM than the ESP32
- It’s not fun when you are porting code using multi dimensional arrays which have to be rewritten as single dimension once converted to malloc
- I also got nailed by sizeof(array) returning the array size before and pointer size after conversion to malloc

Converting arrays to mallocs()

The most annoying parts are: having to change working code, checking for malloc success, and dealing with multi dimensional arrays.

```
Dot *gDot;          // Creates an object named gDot of type Dot class
//Dot gDot[MAX_SHELLS];
Dot *gSparks;       // Creates an array object named gSparks of type Dot class
//Dot gSparks[MAX_SHELLS*MAX_SPARKS];

void fireworks_setup() {
    gDot    = (Dot *) malloc(MAX_SHELLS * sizeof(Dot));
    gSparks = (Dot *) malloc(MAX_SHELLS * MAX_SPARKS * sizeof(Dot));
    while (gDot == NULL || gSparks == NULL) {
     Serial.println("fireworks_setup malloc failed");
    }
    memset(gDot,    0, MAX_SHELLS * sizeof(Dot));
    memset(gSparks, 0, MAX_SHELLS * MAX_SPARKS * sizeof(Dot));
}

// gSparks[a][b].Move();
gSparks[a*MAX_SPARKS+b].Move();
```

Multi dimensional arrays with malloc and memory saved

If you don’t want to rewrite multi dimensional arrays, you can use a malloc loop

- This is better as it allows use of non contiguous memory segments
- But it’s also worse because you can’t do pointer operations on what used to be a contiguous block of memory

```
There is a more tedious way to allocate 2 dimentional arrays so that tempMatrix[i][j] still works
for (uint16_t i=0; i < MATRIX_WIDTH+1; i++) {
    tempMatrix[i] = (uint8_t *) malloc(MATRIX_HEIGHT+1);
    while (tempMatrix[i] == NULL) { Serial.println("tempMatrix[i] malloc failed"); }
    memset(tempMatrix[i], 0, MATRIX_HEIGHT+1);
}

Before, with arrays
32-bit Memory Available: 179392 bytes total, 86640 bytes largest free block
8-bit Accessible Memory Available: 92752 bytes total, 36576 bytes largest free block

After malloc cleanup, 39KB of static memory saved
32-bit Memory Available: 212344 bytes total, 86640 bytes largest free block
8-bit Accessible Memory Available: 125704 bytes total, 64624 bytes largest free block
```

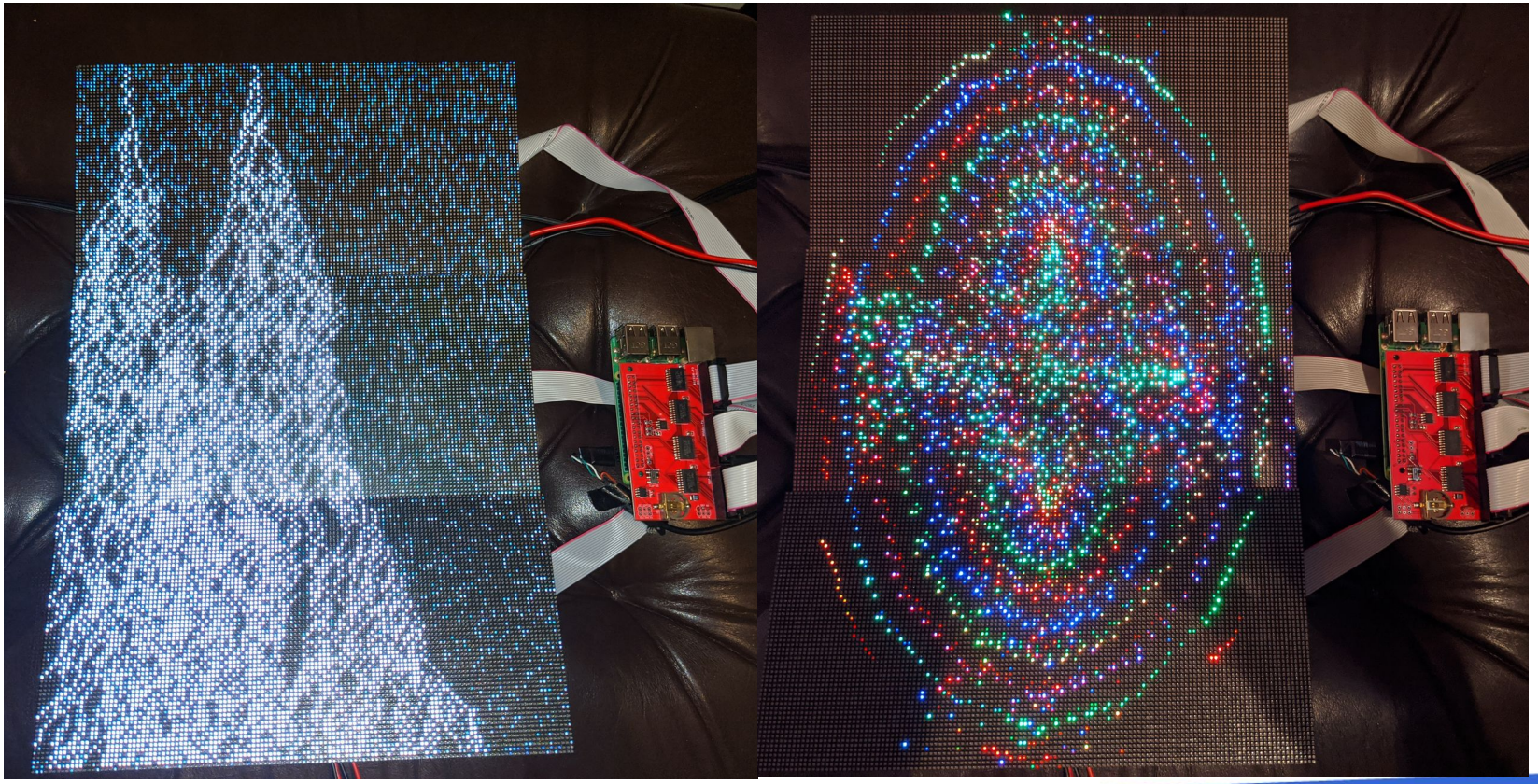
Converting arrays to malloc: the gift that keeps on giving

- LEDMatrix defined its matrix array in its constructor
- This broke on ESP32 as the array got bigger
- So I changed it to a malloc
- And later it broke again because object was created at global scope and malloc couldn’t find enough memory before setup() ran.
- So I had to further modify the library to stop creating the array at creation time. I had to break the API to require calling a method to feed a malloc’ed area after the object was created
- The upstream maintainer will never accept this API breaking change that is there just because of ESP32 memory shenanigans

My future isn’t on ESP32 anymore

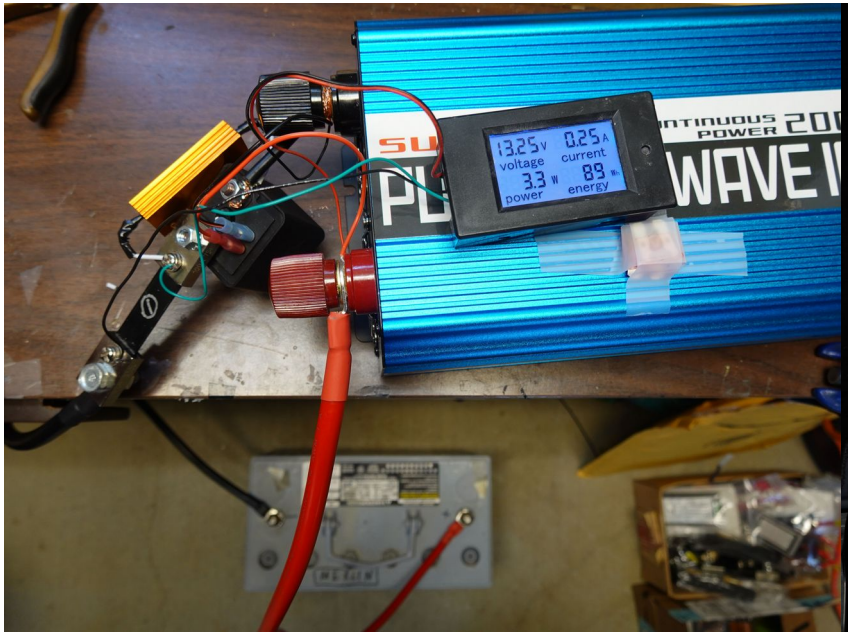
- Higher resolution TFTs would fit in ESP32 PSRAM, but SPI line pushes prefer DMA
- RGBPanels require DMA memory, so not PSRAM
- Also, ESP32 starts running out of steam for refreshing RGBPanels at 128x128
- While I just learned that a future ESP32 chip will allow DMA out of PSRAM and that it should be possible to get arduino-esp32 to allocate arrays automatically in PSRAM, for now, I had to use other methods.
- This is where rPi comes in and https://github.com/hzeller/rpi-rgb-led-matrix . So much more RAM and ability to run 3 RGBPanel strings in parallel.
- I forked https://github.com/marcmerlin/ArduinoOnPc-FastLED-GFX-LEDMatrix to allow building arduino code on linux (hence rPi too)
- I then wrote a glue driver between arduino and linux rpi-rgb-led-matrix: https://github.com/marcmerlin/FastLED_RPIRGBPanel_GFX
- Arduino code running on RGPPanels in high resolutions that wouldn’t fit on ESP32 or teensy or other arduino API chips: Profit!

https://github.com/marcmerlin/FastLED_RPIRGBPanel_GFX



Not even remotely related, powering my house from my Tesla

- Had to find a direct tap into the 400V-12V DC-DC converter
- And build a relay/resistor slow ramp up to prevent converter shutdown on powerup due to the massive capacitor that takes hundreds of amps



Q&A

Slides:
http://marc.merlins.org/linux/talks/ESP32_Memory/

Marc Merlin
merlin@google.com