UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Portable Thermal Printer

Electrical & Computer Engineering

Team 29

5/2/2023

Problem

HP Inc. has a need for a portable printer:

Hardware is inherently faster than software.

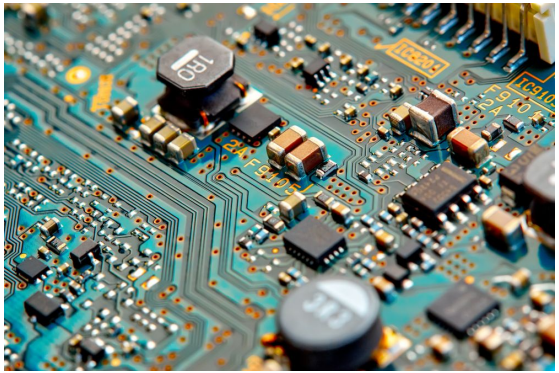
How do we speedup software approaches?

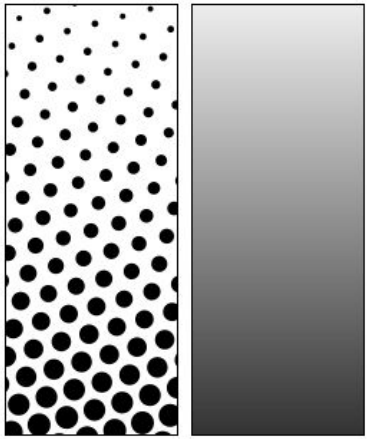
How do we make our printer portable?

Introduction

Task: Create an easy to use, portable, and fast printer.

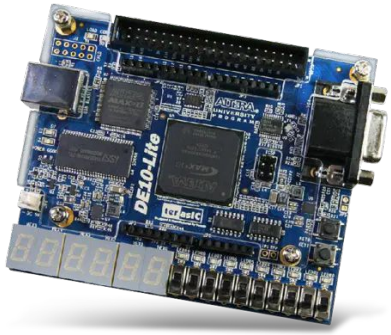
- Users can upload an image to a server for printing, anywhere, anytime.
- Taking advantage of specialized hardware acceleration to perform image processing algorithms with low scalability (similar to ASICs in the real world).





Objectives

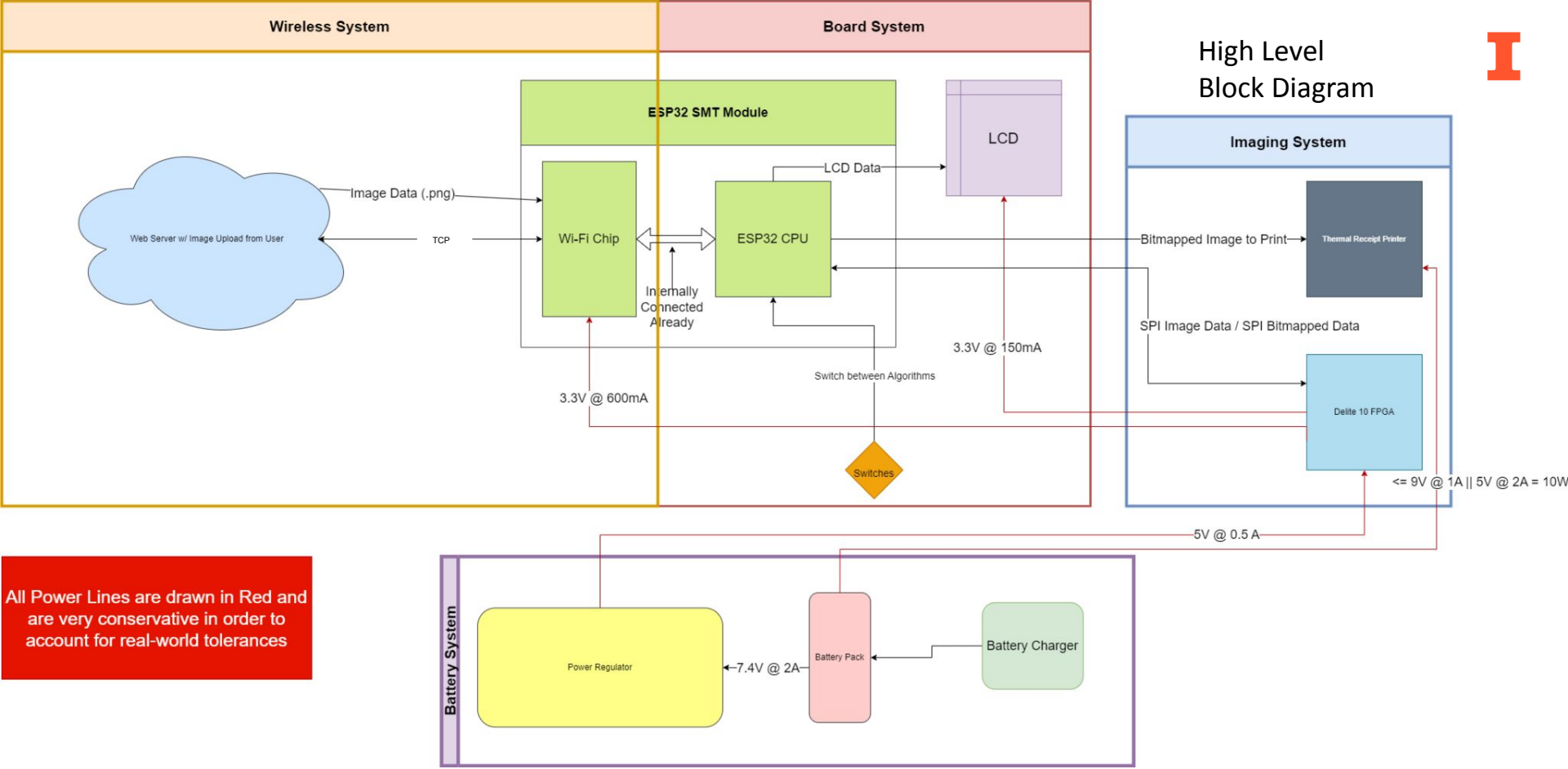
- Printers have remained relatively unchanged in the commercial printer industry.
- We intend to create a proof-of-concept portable printer that is Wi-Fi enabled, battery operated, and takes advantage of hardware acceleration (an ASIC architecture emulated using an FPGA).



Design

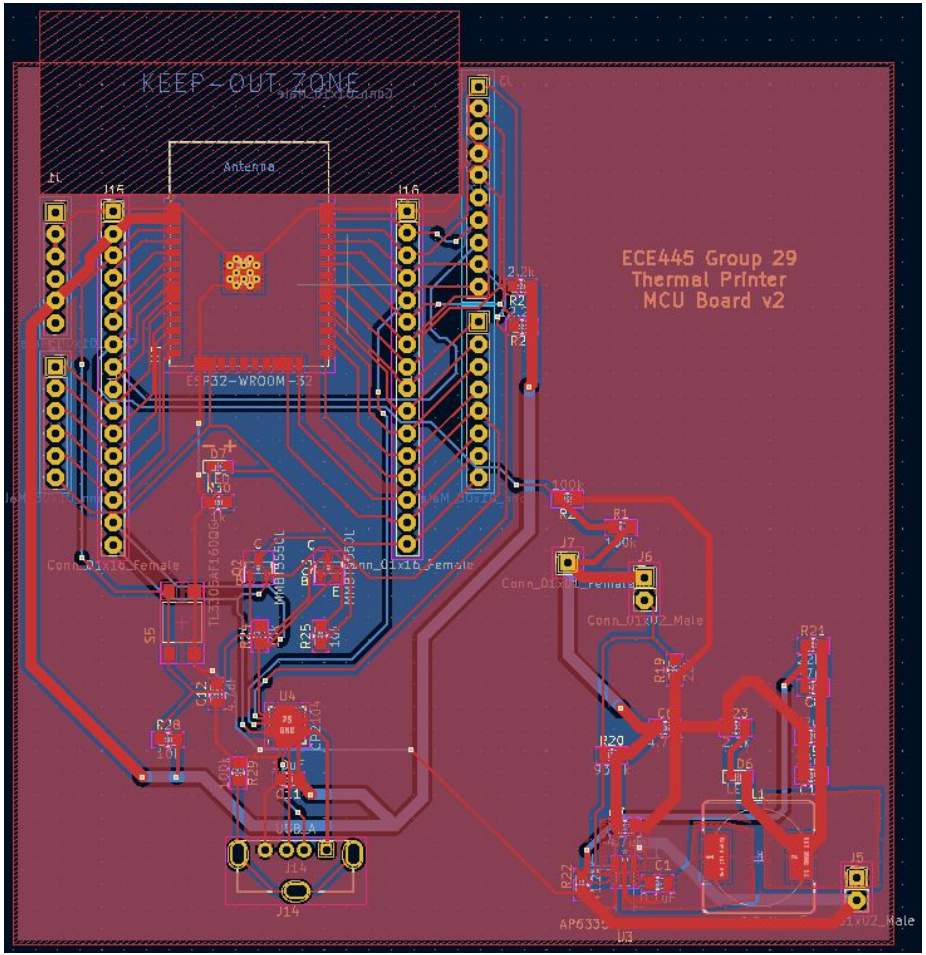
Wireless, board, power, and imaging subsystems

High Level Block Diagram



All Power Lines are drawn in Red and are very conservative in order to account for real-world tolerances

PCB Schematic



Wireless Subsystem

- Grants users high level access to printing system, allowing them to connect through the internet and upload an image to the print queue
- Shields the low level details of the printer implementation from the user.
- Simple, easy to use interface allowing file uploads of PNG, JPG, and JPEG.
- Printing server designed with Flask, Python, HTML/CSS.

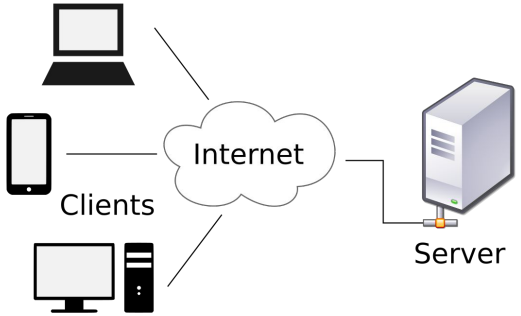
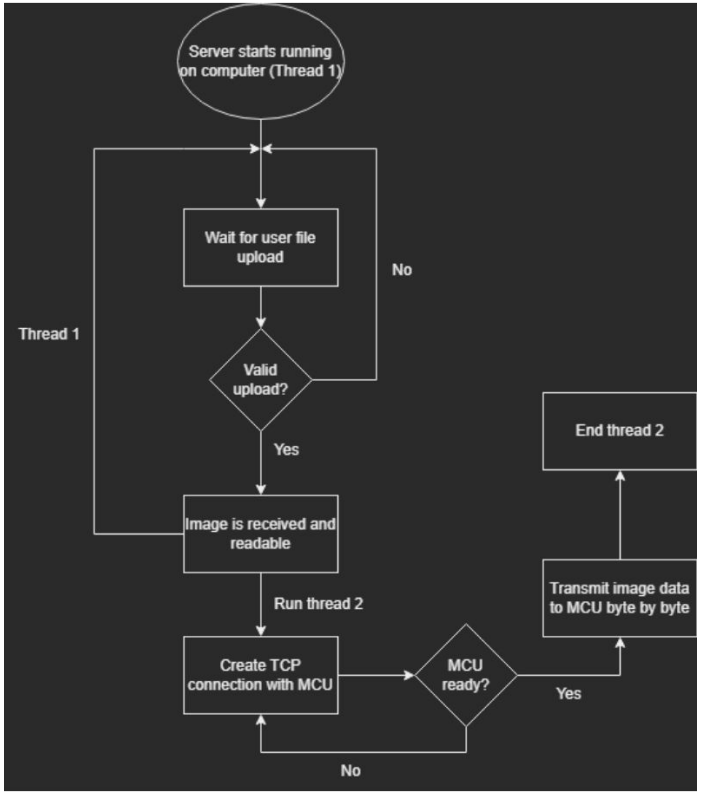


Image courtesy of https://wikipedia.org

Wireless Subsystem Cont'd

- Server, upon user upload, launches thread to establish TCP socket for ESP32 MCU to connect to.
- TCP transport protocol ensures reliable and in-order delivery of uploaded image byte data.
- New thread blocks transfer of data until MCU accepts connection as client, while original thread continues to run the server.

Wireless Subsystem Cont'd



Server control flowchart

Wireless Subsystem Cont'd

- Verifications:
  - Automated timing to ensure runtime of user upload to server is within 5 seconds every time.
  - Varying image formats, sizes, load on server.

Image Type	Image Size (KB)	Number of Devices Concurrently Connected to Server	Upload Time (s)
JPG	32.0	3	0.0009999
JPG	1602.4	3	.005952
PNG	412.1	3	.001757
JPG	77.1	3	.0009966
JPG	22.7	3	.001949
JPEG	212.2	3	.001304

- **Binomial distribution calculation practically guarantees likelihood of large image arriving on IllinoisNet is still within allowed time**

Assuming 100 people are connected to local IllinoisNet access point at 240 Mbps with fair bandwidth allocation per active user, we allow a tolerance of up to 14 other active users for a 10MB image to be uploaded within 5 seconds. The probability of simultaneous active users exceeding 14 at any given moment is 1.183×10^{-29}, given that a user has a .5 probability of being active, and not active.

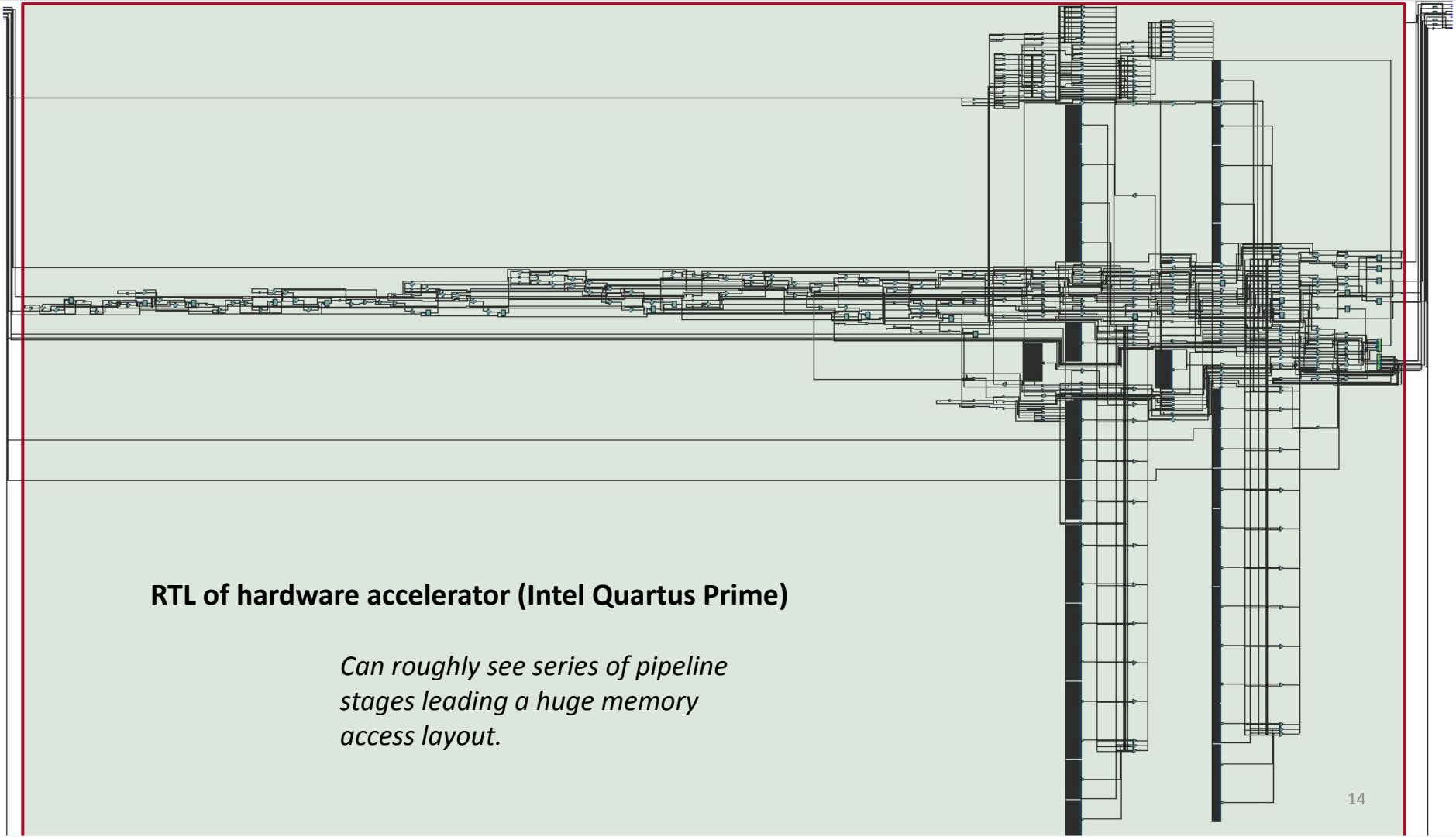
$$P = \sum_{k=0}^{14} \binom{100}{k} * .5^{k} * .5^{100-k}$$

Binomial distribution representation of problem statement

Imaging Subsystem

- Performs an algorithm (e.g., dithering, threshold) on the image data in hardware (DE-10 Lite FPGA).
- What does the hardware actually look like?

https://thi.ng/pixel-dither, Karsten Schmidt; image on right



RTL of hardware accelerator (Intel Quartus Prime)

Can roughly see series of pipeline stages leading a huge memory access layout.

Acceleration of Image Processing Algorithm (e.g., Floyd-Steinberg Dithering)

```
for each y from top to bottom do
    for each x from left to right do
        oldpixel := pixels[x][y]
        newpixel := find_closest_palette_color(oldpixel)
        pixels[x][y] := newpixel
        quant_error := oldpixel - newpixel
        pixels[x + 1][y    ] := pixels[x + 1][y    ] + quant_error × 7 / 16
        pixels[x - 1][y + 1] := pixels[x - 1][y + 1] + quant_error × 3 / 16
        pixels[x    ][y + 1] := pixels[x    ][y + 1] + quant_error × 5 / 16
        pixels[x + 1][y + 1] := pixels[x + 1][y + 1] + quant_error × 1 / 16
```

Wikipedia on Floyd-Steinberg Dithering pseudo-code.

Image courtesy of https://wikipedia.org

Imaging Subsystem: Expectations

- WANT: Use hardware over software to speed up process.
- What we got:
  - For large images, hardware finishes the process faster.
  - **`millis()`** to measure software time (MCU).

Image Size (Width by height in pixels)	MCU measured mean time (Milliseconds)	FPGA calculated time (Milliseconds)	SPI protocol time (Milliseconds)
8 by *8* (total of 64)	*≈ 0*	*10.48576*	*52.4132*
64 by *16* (total of *1024*)	*12.1*	*10.48576*	*52.4132*
64 by *64* (total of *4096*)	*48.5*	*10.48576*	*52.4132*
x by *y* (Total of *65536*)	*779.2*	*10.48576*	*52.4132*

FPGA dithers pixel at 50 MHz with safe upper bound of 8 clock cycles.

SPI sends bits at 20 MHz.

Constant-sized buffers of 65536 bytes.

Imaging Subsystem: Expectations Cont'd.

- **Let t_{FPGA} be the time taken for the FPGA to run its algorithm**

$$t_{FPGA} \leq 65536 \text{ pixels} \times \frac{8 \text{ clock cycles}}{\text{pixel}} \times \frac{1 \text{ second}}{50e6 \text{ clock cycles}} = 0.01048576 \text{ seconds}$$

- **Let t_{SPI} be the time taken in the SPI protocol**

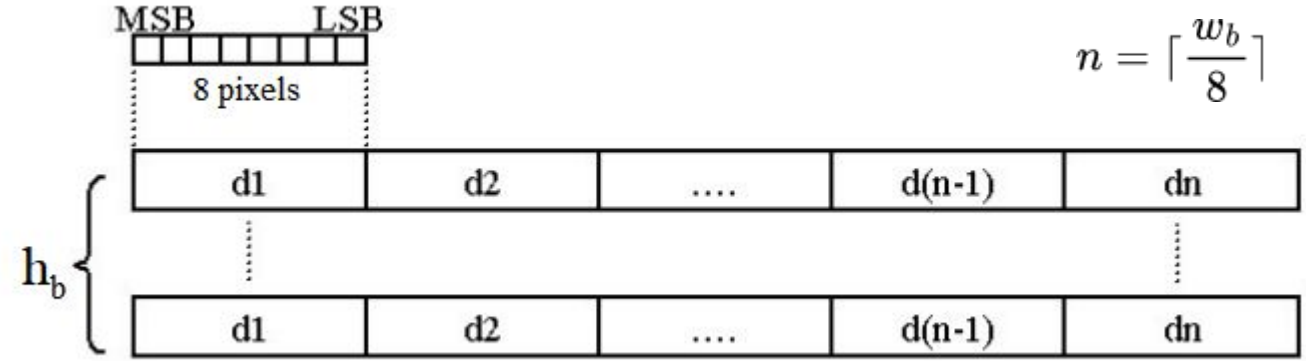
$$t_{SPI} = (6 \text{ header bytes } + 2 \times 65536 \text{ pixel bytes }) \times \frac{8 \text{ bits}}{1 \text{ byte}} \times \frac{1 \text{ clock cycle}}{1 \text{ bit}} \times \frac{1 \text{ second}}{20e6 \text{ clock cycles}} = 0.0524312 \text{ seconds}$$

- ONLY constraints on bitmapped image (may be downscale of original) with width w_b and height h_b:

$$1 \leq w_b \leq 384$$

$$1 \leq h_b$$

$$w_b \times h_b \leq 65536$$



Imaging Subsystem: Printing Times

- **Vertical scan total t_v:**

$$t_v = 30000 h_b \text{ microseconds}$$

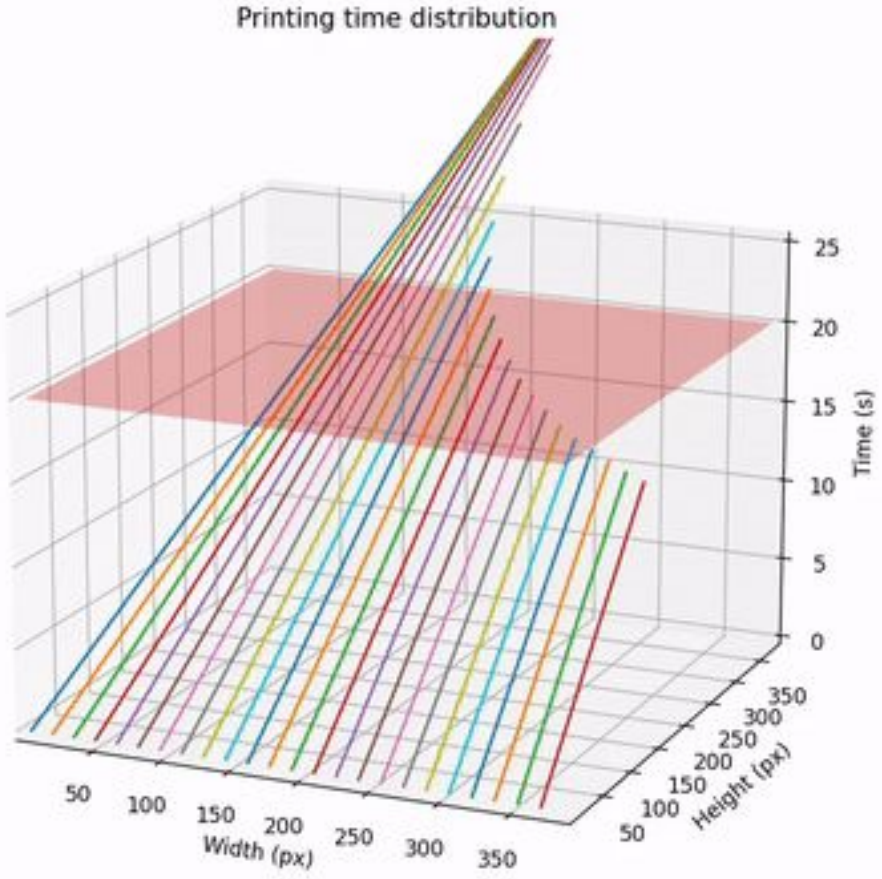
- **Horizontal scan total t_h:**

$$t_h = \frac{w_b h_b \text{ bytes}}{\frac{8 \text{ bytes}}{1 \text{ bitmap byte}}} \times \frac{11 \text{ bits} \times 1000000 + \frac{9600 \frac{\text{bits}}{\text{second}}}{2}}{9600 \frac{\text{bits}}{\text{second}}}$$

  - **MAXIMUM t_h: 9.53 seconds**
- **Conclusion:**
  - **Bounded by h_b, dependent on how tall the input is**

Imaging Subsystem: Printing

- WANT: Relatively fast printing.
- What we got:
  - Limited printing speed largely dependent on image height
  - < 15 seconds for “short” images where h_b< 183 pixels
  - < 20 seconds for images where h_b< 348 pixels
  - Based on downscaling scheme, want aspect ratio (w : h) to be greater than 2.0 if input image is large


Printing time distribution
Time (s)
0
5
10
15
20
25
Width (px)
50
100
150
200
250
300
350
Height (px)
50
100
150
200
250
300
350

Timing Conclusions: Summary

How long does it take for a printing job to finish?

1. Uploading image t_{upload}
2. Hardware accelerator process time t_{hw}
    a. SPI protocol time t_{SPI}
    b. FPGA algorithm time t_{FPGA}
3. Printing operation t_{print}

Timing Conclusions: Piecing it together

$$\mathbb{E}[t_{upload}] \leq 4.99 \text{ seconds } \times (1 - 1.183 \times 10^{-29}) + 5 \text{ seconds } \times (1.183 \times 10^{-29}) \approx 4.99 \text{ seconds}$$

$$t_{hw} = t_{SPI} + t_{FPGA} \leq (0.0524312 + 0.01048576) \text{ seconds} = 0.06291696 \text{ seconds}$$

$$t_{print} \leq 15 \text{ seconds if } h_b < 183 \text{ pixels}$$

Conclusion:

If IllinoisNet is slow but usable (e.g., many students, DoS attack) and image file <= 10 MB and image aspect ratio (*w* / *h*) is > 2.0, then…

Our project prints it within around 20 seconds (the summation of above).

Thermal printer temperature

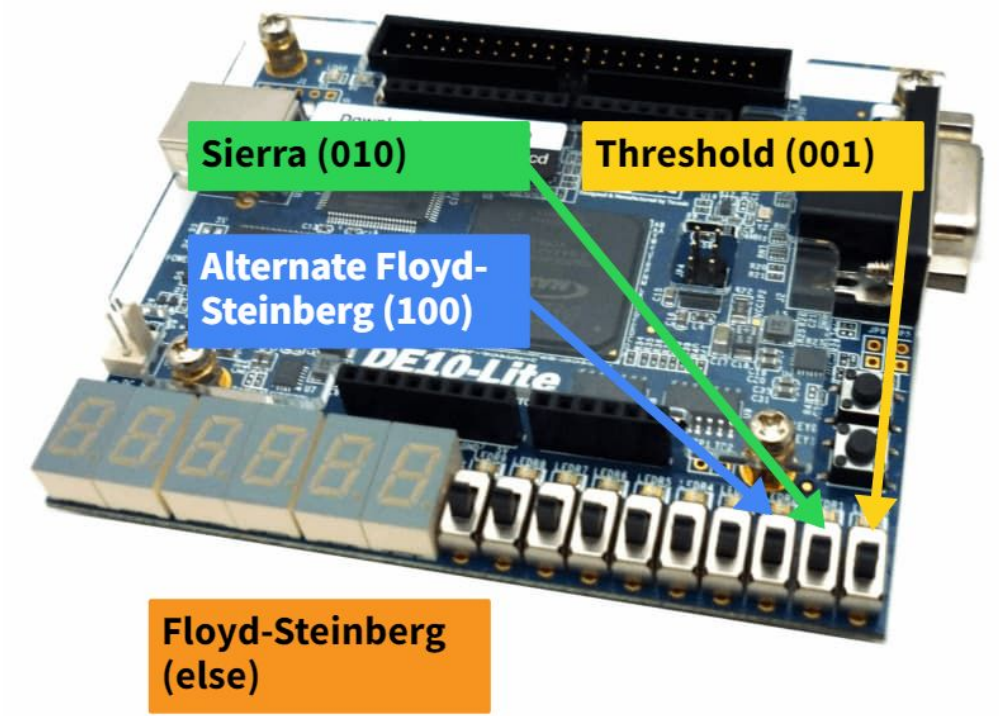
- WANT: Printer temperature below 120°F or 48.8889°C
  - WHY?:
    - Higher temperatures cause distortions in printout.
    - Safe to the touch for users.
- Printer has ability to compute temperature
  - Minimum was 24°C
  - Maximum was 40°C

Board Subsystem

- On-printer subsystem which allows for user interaction and system observation.
- 128x32 pixel LCD informs user of current printer state, from ready at startup, printing, and completed states.
- Switchbox allows users to specify dithering algorithm to better suit their needs for print job (i.e., choose between Floyd-Steinberg, thresholding, etc.).

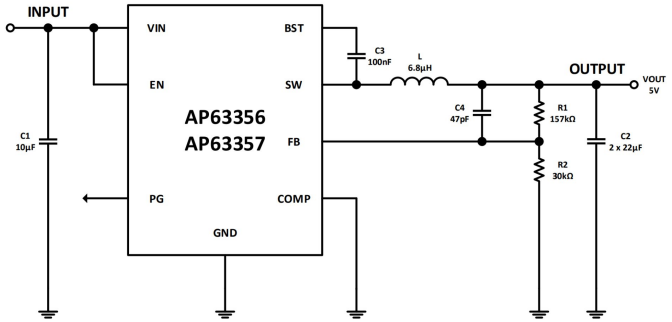
Board Subsystem Cont'd

- **LCD sample: link**
- **Video: link**



Power Subsystem

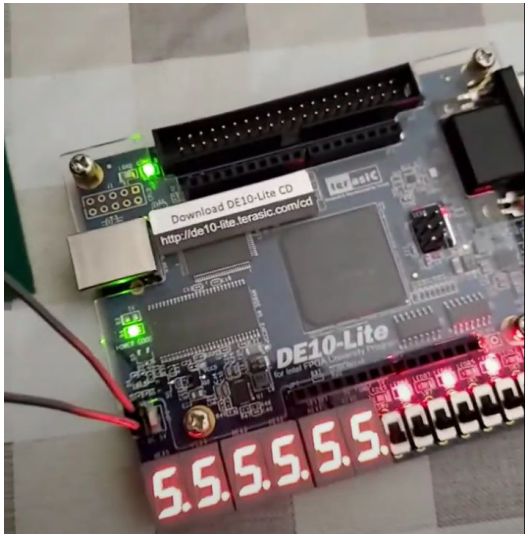
- Responsible for powering the entire printer system
- Uses a 2000mAh 7.4v Lithium Polymer battery
- PCB buck converter (AP63356) steps down 7.4v to 5v for powering FPGA, and FPGA has embedded buck converter to step down to 3.3v for MCU



Recommended buck converter application circuit directed by documentation
@ https://www.diodes.com/assets/Datasheets/AP63356-AP63357.pdf

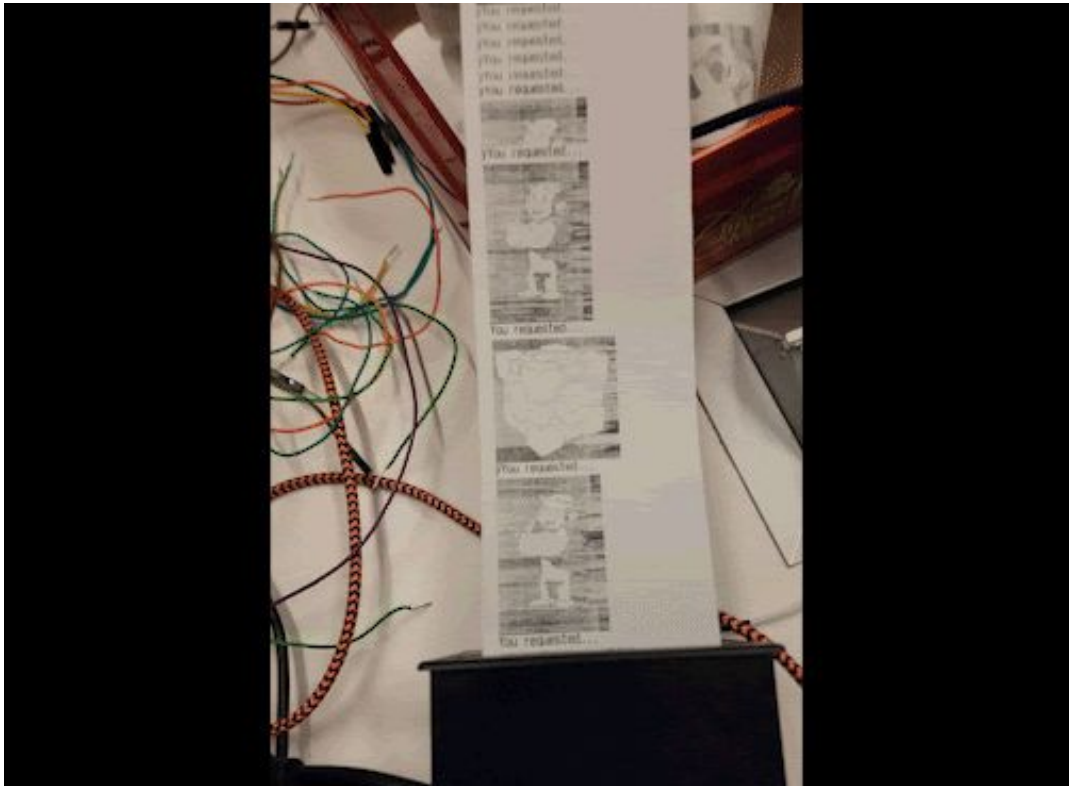
Power Subsystem Cont'd

- WANT: Every component to be powered sufficiently
- Averages with battery recorded @ 8.1v:
  - ESP 32
    - 3.2 - 3.3V
  - Thermal printer
    - 7.8 - 8 V
  - FPGA
    - 4.8 - 4.9 V
  - LCD
    - 3.2 - 3.3V



Powering the FPGA using only regulated 5v DC, arguably the most important aspect of the project to have working

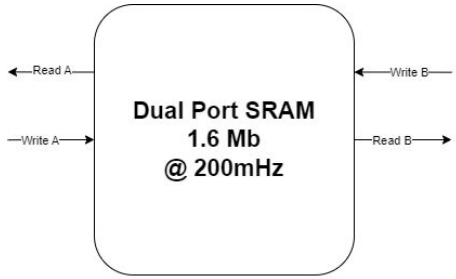
Sample (left: user-uploaded image, right: Floyd-Steinberg dithered printout)

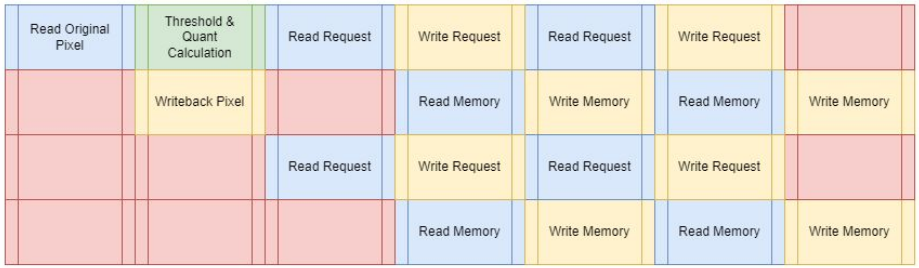


Challenges

- Microcontroller swap
- Failed superscalar attempt at architecture
- FPGA Pipelining Chaos
- 7.4 Battery Explosion & RIP Laptop Incident
- Free 3D Printing

FPGA Pipelining Chaos & Failed Superscalar attempt





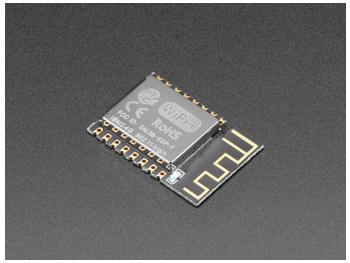
←Potentially 16 clock cycles instead of 5 with pipelined memory approach→

←Pipelining Chaos→

All reads are blue
All writes are yellow
Thresholding & Quant Calculation is done in green
Else everything else is red

- **Relatively simple calculations.**
- **MEMORY LIMITED!!!**
- Overall speedup of 3.2x with memory access!
- Did not successfully implement any parallel execution/superscalar approach toward the quantization error spreading & write back pixels.

Microcontroller Issues

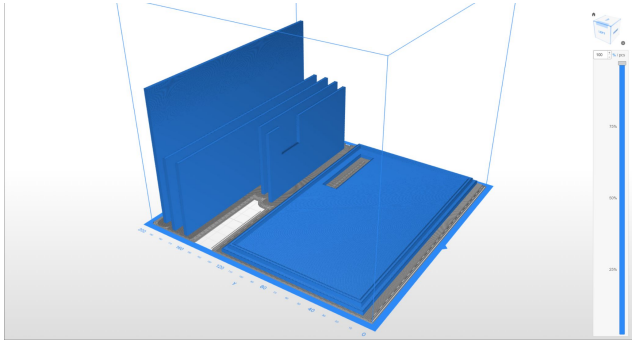


- At startup, the ESP8266 MCU would send garbage to the sensitive inputs on the FPGA and trigger the image processing, resulting in the garbage being processed and ready for read.
- In-depth read of documentation reveals that our FPGA trigger (GPIO2) actually pulses at startup
- MCU also has low on-chip memory (80KiB) and few additional pins we could program.
- Needed an MCU with at least more (stable and programmable) pins and more memory to work with: the ESP32.

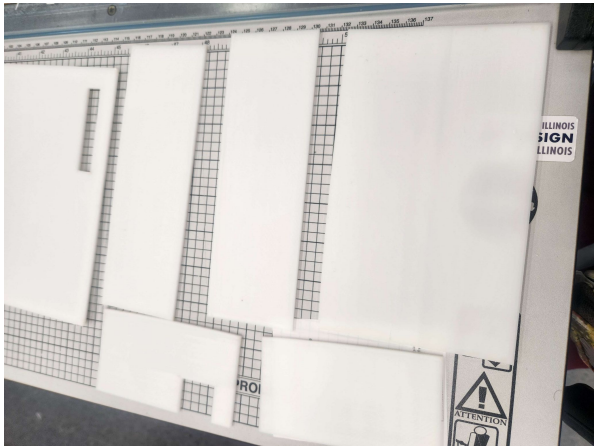
- Required PCB redesign, ordering of new MCU's, reading documentation, etc.

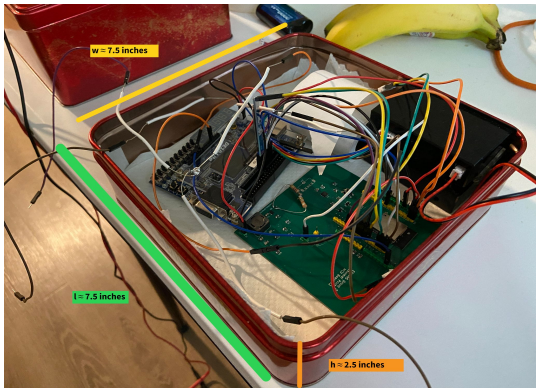
3D Printing on Campus

- Created a 7.5”x4”x2.5” model for the printer container, only for the print job to have an extremely long delay.
- Actual 3D printed components ready *after* the demo:



CAD model





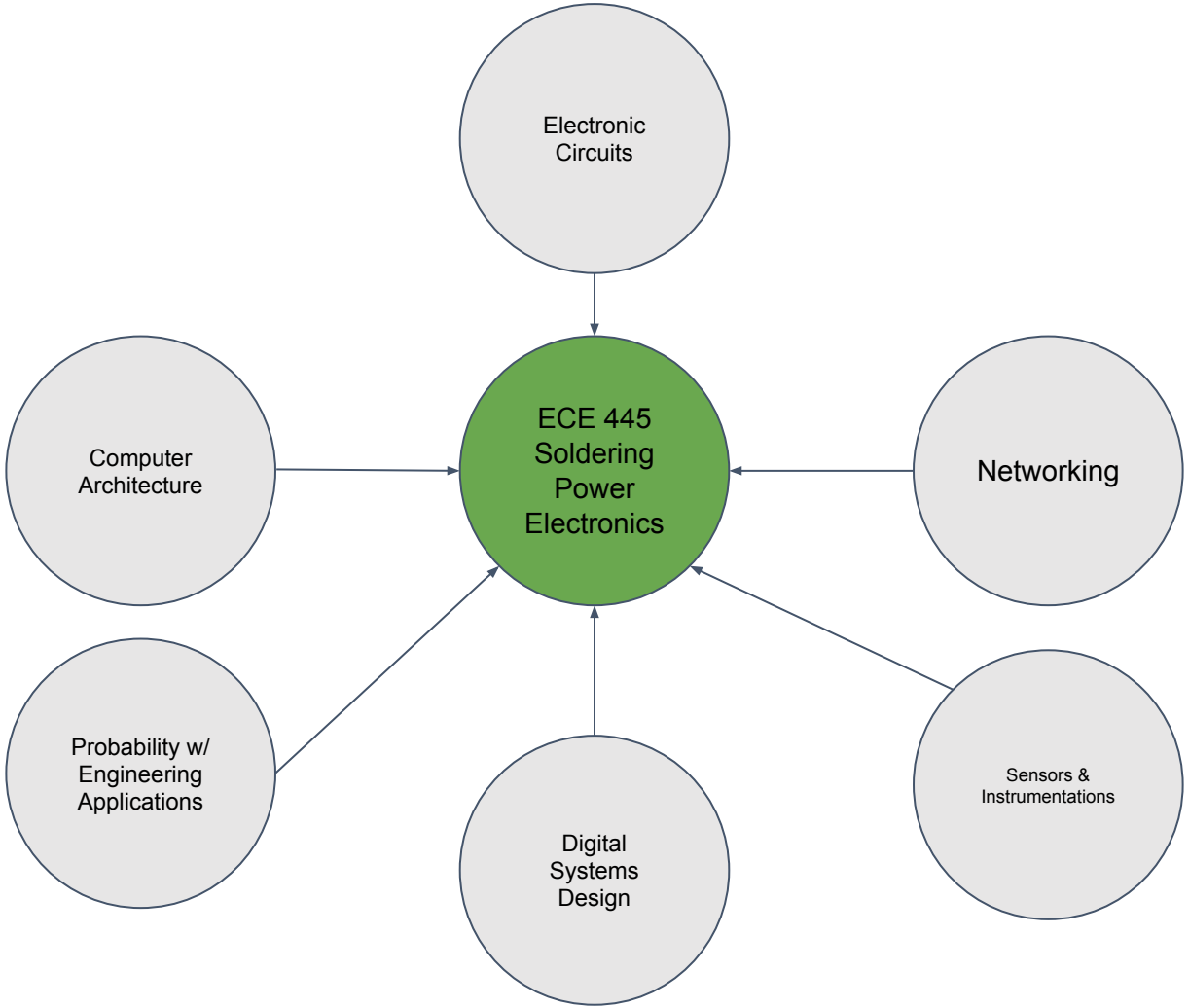
Printer at time of demo

Conclusions

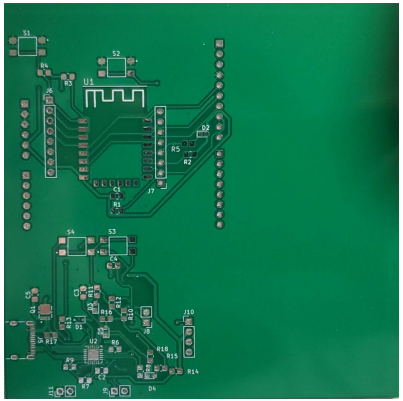
What we learned?

What can be improved?

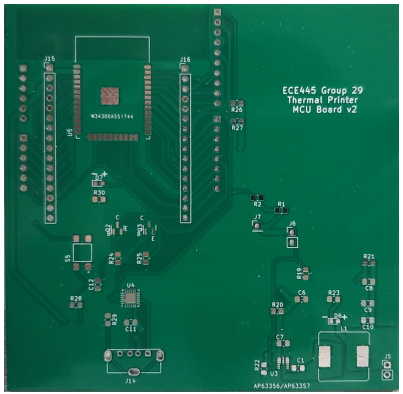
Outlets for improvement?


Electronic Circuits
Computer Architecture
ECE 445 Soldering Power Electronics
Networking
Probability w/ Engineering Applications
Digital Systems Design
Sensors & Instrumentations

What We Learned



PCB v1



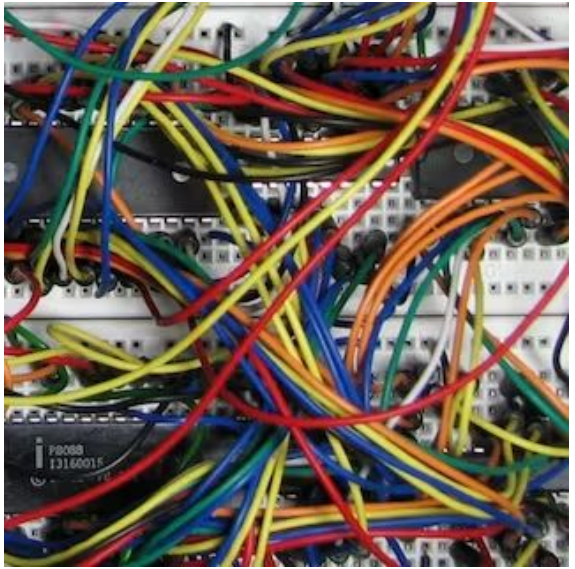
PCB v2

- Decide important components after extensive research.
- Be *extremely* wary of shorts, wires, and batteries.
- The smallest of changes may require overhaul of entire project.

- **ENGINEERING IS HARD BUT REWARDING!!**

What We Could Have Changed

- Redesign and implement even more algorithms on hardware.
- Different architecture approach to remedy memory access limitations.
- Resolder PCB pins for direct FPGA connection to reduce footprint size and improve board organization (accidentally soldered pins on wrong side).
- Chosen the ESP32 from the start (initially seen as “overkill” instead of “safe” option).



Rat’s nest due to poor organization

Outlets For Future Improvements

- Can extend concept to a different type of printer guts (ie. ink based), allowing more than just B/W printing.
- Allow users to customize print resolution, size, etc., through wireless subsystem.
- Commercial cloud-hosted server for connectivity anywhere, not just locally.

Image courtesy of https://wikipedia.org