Portable Thermal Printer

By

Team 29

ECE 445

Jason Liu (jliu246)
Gally Huang (ghuang23)
Kevin An (kqan2)

TA: Hanyin Shao (hanyins2)
Professor: Viktor Gruev

May 2023

Abstract

This report dictates the process that we as a team took in order to complete our senior design project. Inside this report, you should find important diagrams and pictures that showcase each of our unique features and the requirements for the user if they ever want to recreate this project.

1. Introduction

1.1. Problem

One of the biggest issues frequent travelers have to face is the issue of portability. Items that are carried by travelers cannot consume too much space, weigh too much, and should not compromise on quality. One target area that has been identified by Hewlett-Packard Inc. (HP Inc.) is within the commercial printer industry, where printers have remained relatively unchanged over time with respect to other technologies that have already shifted toward more portable means. HP Inc. has proposed for us to create a proof-of-concept portable printer that will allow them to stay competitive in the printer industry.

1.2. Solution

Our solution to this challenge is to create a portable thermal printer, meeting criteria of being efficient and fast, as well as being able to function wirelessly by connecting to a server that users can upload to for printing. We will implement this solution through four subsystems as follows: the Wireless Subsystem, Imaging Subsystem, Board Subsystem, and Power Subsystem. These subsystems and their connections are illustrated in Figure 1.

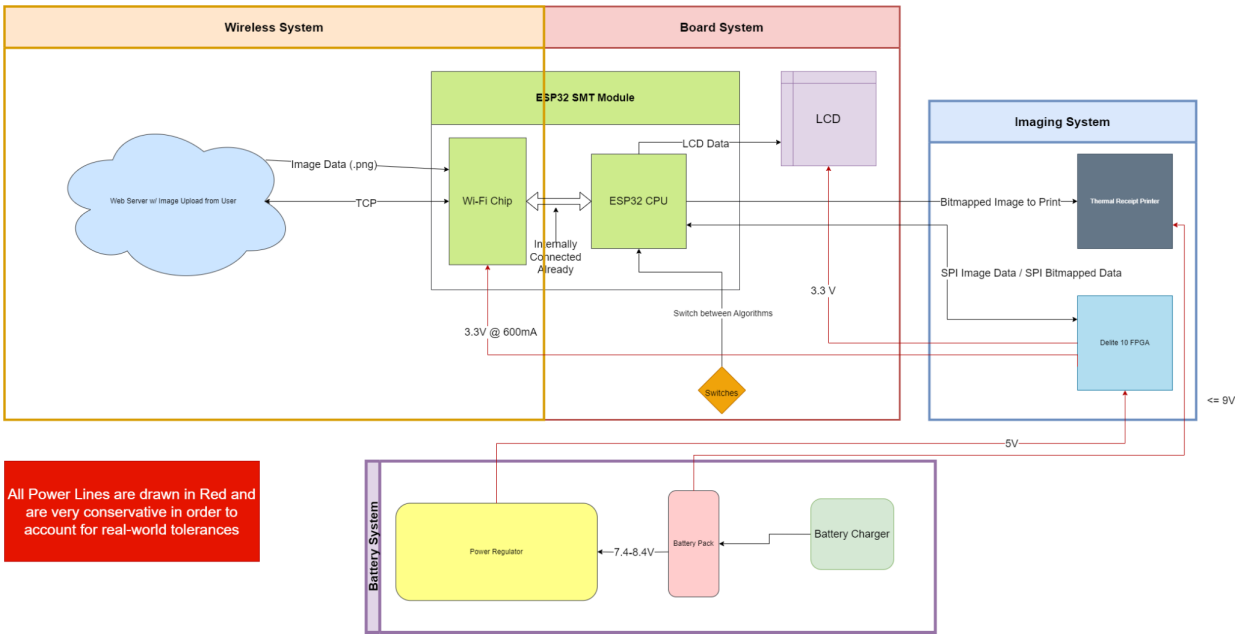

Figure 1: High-level block diagram.

1.3. High-Level Requirements

In our proposal, we outlined and met the following high-level requirements during our final demonstration and verifications:

1. The system is portable, in which the device can receive data from users wirelessly and accurately while only taking up a small footprint of fewer than 12 inches by 12 inches.
2. The start-to-end time between a user upload to the completion of a job should take no more than 20 seconds, making it fast for the user.
3. The device itself should be powered entirely by batteries, having a worst-case battery life operating time of at least 1.5 hours.

2. Design

2.1. Wireless Subsystem

2.1.1. Overview

The Wireless Subsystem is responsible for allowing a user high-level access to the printer, where they can connect through the internet to the printing server to upload (POST) an image in the PNG, JPG, and/or JPEG format to the print queue. The backend server is designed using the Python Flask framework and has a simple, easy-to-use frontend interface designed with HTML/CSS.

Upon a successful user upload, the server launches a new thread to establish a TCP socket connection with the ESP32 MCU. On this thread, the ESP32 will connect to the server, the server sends the byte-level image data to the MCU until all the image data has been received by the MCU, and the thread terminates after this step. If the MCU does not initially connect, the thread blocks until the MCU is ready to accept the connection. The original thread continues to run the server, allowing system responsiveness even while waiting for ongoing jobs in the print queue. The control flow can be seen in Figure 2.

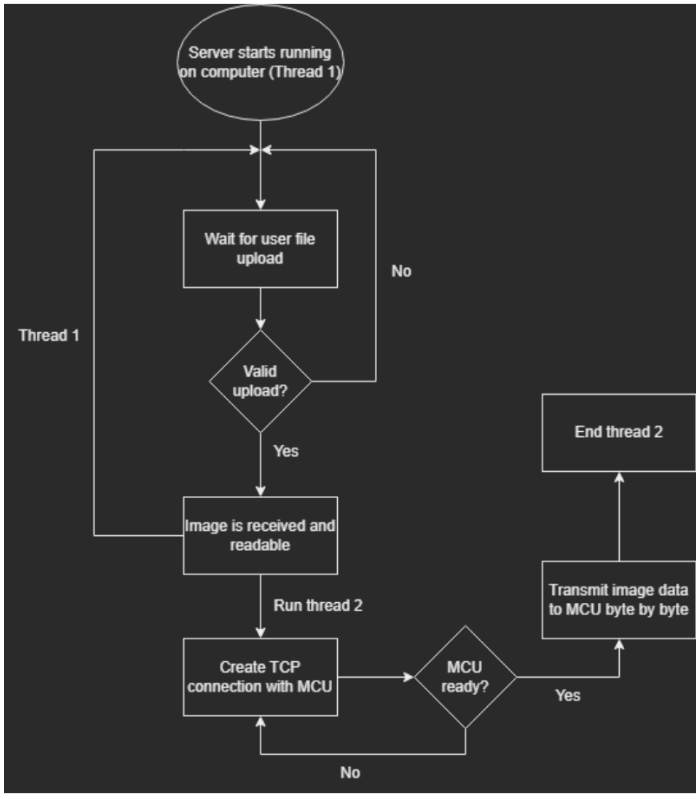

Figure 2: Server control flow diagram.

In addition, we consider the ESP32 MCU to be part of the wireless subsystem due to the part it plays by interacting wirelessly with the server, although it is physically soldered to the PCB on the printer system itself and is responsible for the printer control flow.

2.1.2. Design Decisions

The primary design decision we had for the Wireless Subsystem is that it would be fast, allowing an image of up to 10 MB to be uploaded to the server in under five seconds. In addition to being fast, we also wanted it to be lightweight, and that is one of the main benefits of choosing a backend framework such as the Flask framework in Python. There are very few overhead features while allowing for relatively fast development, and we are also well versed in programming using Python in general, thus making this framework ideal for our needs. In addition, creating the server code allows us to turn any computer we have, such as someone's laptop, into a local server to communicate with the rest of the system.

An alternative option for developing the server that was initially considered was the Django framework, but after attempting to create the server it was deemed to be too difficult to work with for a small-scale project, despite having a much richer set of backend features that Flask does not offer. In addition, we chose to host the server locally to reduce the costs of the project, as using a commercial cloud services provider such as Amazon Web Services and Google Cloud Platform all require continuous funding to maintain, and once again, considering the scope of this project we do not have any large scale implementation requirements that a local server cannot handle.

One major change we had to accommodate was changing the MCU entirely, from the ESP12F to the ESP32. The reason for this was that the FPGA trigger pin we used on the ESP8266-12F, GPIO2, was documented to pulse high on startup, which was unexpected behavior until we realized that it was causing us problems dealing with the FPGA, and we did not have additional stable and reprogrammable pins to replace this with. In addition, the ESP32 has far more memory than the ESP12F, making it better suited for storing relatively large quantities of image data.

We downscaled images larger than 65,536 pixels because the FPGA SRAM is limited to 65,536 bytes and the MCU RAM is limited to around 100,000 contiguous bytes. To preserve image details, we chose to approximate the optimal downscaled width and height through binary search.

2.3. Imaging Subsystem

2.3.1. Overview

The Imaging Subsystem transforms the input image into a bitmap with image width w_b and image height h_b to print out. It encapsulates the FPGA and the thermal printer. There are communication lines between this subsystem and the MCU since we want the input image sent from the MCU to the FPGA, the processed image sent from the FPGA to the MCU, and the bitmapped image sent from the MCU to the printer.

When the MCU receives the input image through the Wireless Subsystem, it uses SPI protocol to transfer the image data to the FPGA. And to communicate with the printer, the MCU uses hardware serial UART to send over a bitmap formatted as in Figure 3 [11]. The bitmap is a buffer of bytes interpreted with h_b rows and w_b columns. Each byte d(i) in the bitmap covers at most 8 pixels, where the MSB is the leftmost pixel and the LSB is the rightmost pixel. A pixel is black if its bit value is 1 and a pixel is white if its bit value is 0. Each row has

$$n = \lceil \frac{w_b}{8} \rceil \qquad (2.3.1)$$

bitmap bytes. Intuitively, this is fitting each row of w_b bytes to the necessary amount of bitmap bytes. Extra pixels in byte dn (if w_b is not a multiple of 8) are white pixels.

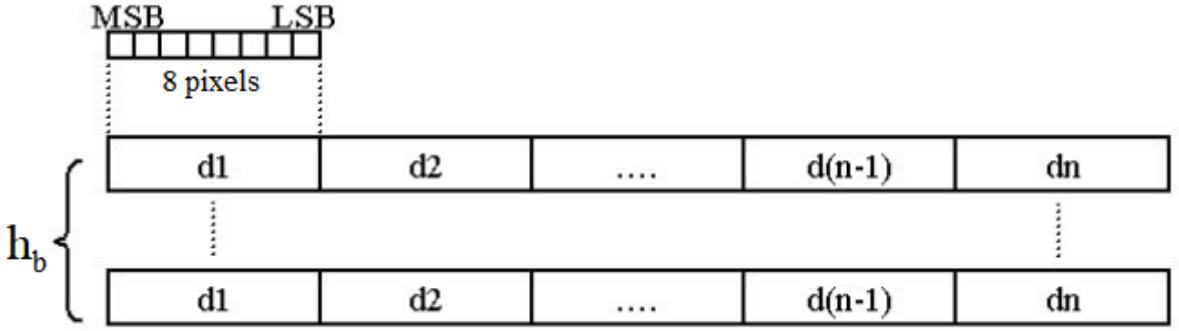

Figure 3: Bitmap structure as input to printer.

2.3.1.1. FPGA Description

The FPGA used in this project is the DE10-Lite, as we already have familiarity using the board from previous coursework and programming using SystemVerilog. Looking at the general algorithm in Figure 4 for the Floyd-Steinberg dithering algorithm, we understand that there are many inefficiencies we can target using digital hardware to speed up this process.

```
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
```

Figure 4: Floyd-Steinberg dithering algorithm pseudo-code [5].

The first observation is that a software model would have to parse through each section of the array individually, and in this software model, there are four adjacent pixels that need to be modified, meaning there would be at least four states of the software reading and writing to the memory. Additionally, the software model stores the current pixel into a new variable of oldpixel rather than just using the current pixel value directly, as well as saving the closest palette color into a new variable.

With respect to our hardware implementation, due to the issues that occur with memory bandwidth and the amount of data that could be accessed at a time, we opted for a dual-port SRAM that can access two pieces of memory at a time. As for the first part of the algorithm implemented on hardware, we compress many of the steps together in order to cut the original 5-cycle process into only 2 cycles of computation.

As shown in Figure 5, we are able to pipeline our SRAM accesses and get a speedup multiplier of 3.2x (16 cycles / 5 cycles). This allows for the already-accelerated hardware implementation to be even faster.

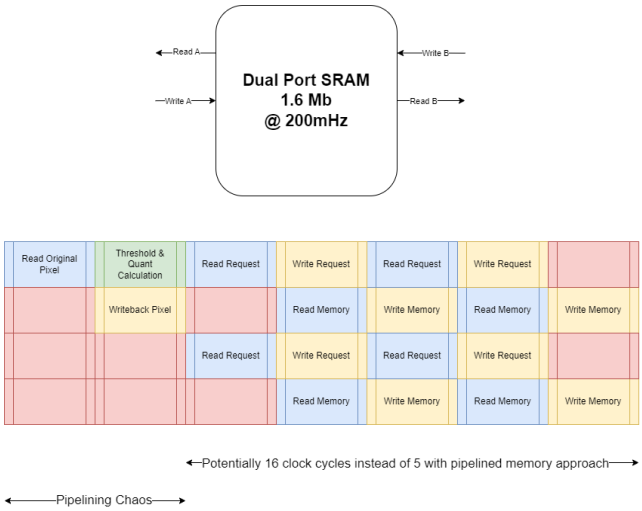

Figure 5: FPGA architecture implementation.

2.3.2. Design Decisions

Initially, we chose to use SPI instead of I2C as the communication protocol between the MCU and FPGA. If we optimized the communication, we would have faster transfers because SPI is full-duplex while I2C is half-duplex [10]. With SPI, we enable the MCU to send and receive data at the same time.

The method we use to transfer the input image from the MCU to the FPGA is correct in that data passes through each step correctly. The method is outlined in Figure 6. Step 1 involves transferring metadata about the image. Namely, the FPGA needs to know the width of each row to process the image correctly, so we send two bytes to configure the 16-bit width value.
We also use a header byte to reset the FPGA registers as well as two more bytes to configure the 16-bit height value. Finally, the last header signals the FPGA to start storing data. If we had more time, we would try to optimize it to fewer SPI clock cycles by combining steps 2 and 3. We could modify the design to send a variable amount of bytes through SPI, but this would take time and add some more complexity.

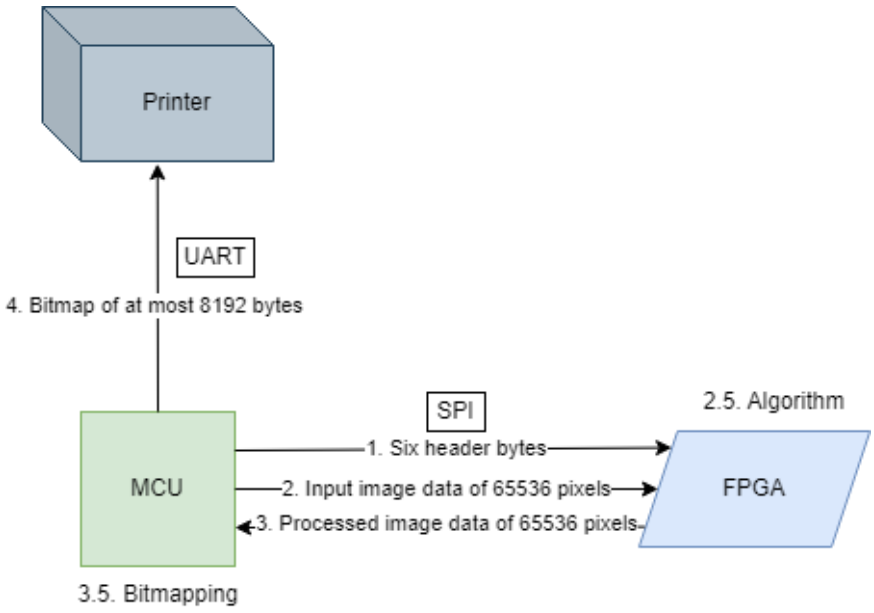

Figure 6: Communication protocol in the Imaging Subsystem.

2.4. Board Subsystem

2.4.1. Overview

The Board Subsystem offers users the ability to interact with the entire system. For this, we use the FPGA's switches to map a switch configuration to an algorithm the Imaging Subsystem will follow. This subsystem also offers diagnostic information about the current job. Namely, there is a 128 by 32 pixel LCD that displays the current state of the job.

2.4.2. Design Decisions

We chose to use the FPGA's switches over using a switch box because they were not in use. With ten switches, we could theoretically implement 2^{10} algorithms.

We chose to implement four algorithms called Floyd-Steinberg dithering, thresholding, Sierra dithering, and an alternative Floyd-Steinberg dithering as in Figure 7 because they are all similar in structure. Only the numbers are different [6], so we implement this as a multiplexer in Verilog. If we had more time, we would add additional similar algorithms.

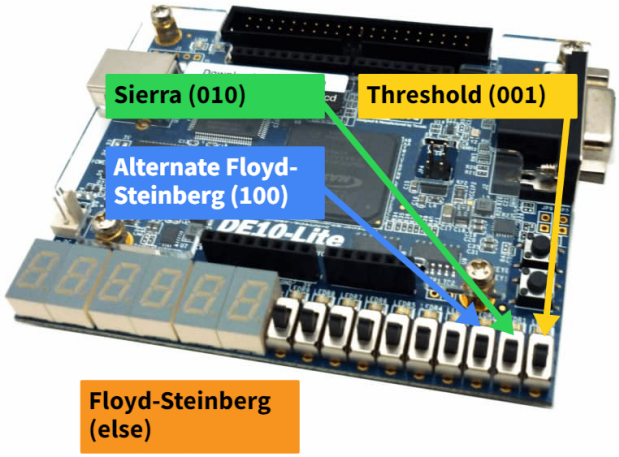

Figure 7: FPGA switches mappings to algorithms.

We chose to display four states on the LCD: "Ready", "Processing", "Printing", and "Completed". They should offer sufficient information about the printing status, covering when a user can upload an image to print ("Ready"), when the Imaging Subsystem is processing the image or transmitting data ("Processing"), when the thermal printer is printing the imager ("Printing"), and when the entire printing job is finished ("Completed"). Should the printer fail to print due to lack of paper, its LED would turn on, so a "Failed" state is not necessary. And if the MCU cannot connect to the server or internet, no state would be displayed.

We used an LCD that uses I2C protocol so that we did not have to handle SPI with multiple slaves, which may cause timing issues or complexity.

2.5. Power Subsystem

2.5.1. Overview

This subsystem is responsible for power regulation and delivery for the rest of the subsystems to function, with the exception of the Wireless Subsystem server. To power the DE10-Lite FPGA using external power, we need a stable 5 V DC power supply [7], and the ESP32 MCU requires a DC voltage range between 3.0-3.6 V [3] throughout the operation. The LCD controller (SSD1306) also requires an operating voltage between 1.65-3.3 V for operation [2].

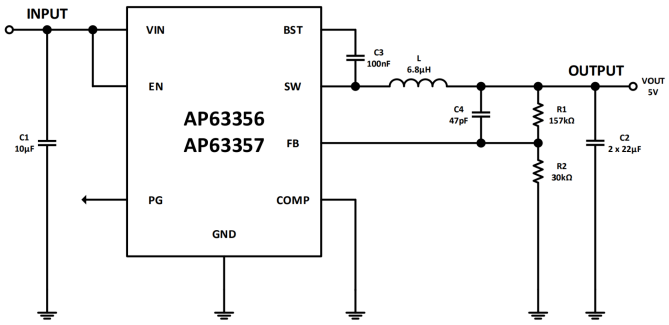

Figure 8: Recommended 5 V application circuit [4] for AP63356.

We use a high-capacity lithium polymer battery rated at 7.4 V, 2000 mAh to power the printer system. This subsystem is extremely important due to the nature of the electronics (notably the MCU and FPGA) we are handling and entrusting to the battery to operate for sustained periods. We create an application circuit on the PCB similar to the one provided above to obtain a continuous 5 V output that we can safely power the FPGA with. By powering the FPGA, we have access to an onboard 3.3 V DC line (explained below) that we can use to directly power the MCU and LCD, both of which are relatively light load on the FPGA's 3.3 V line. We also connect the printer guts input voltage lines to the battery itself, as the printer allows a wide range between 5-9 V for successful operation, and it draws no power while no data comes in.

The battery also comes with a safety circuit that disconnects the battery once its voltage is detected to be below or above a certain threshold, preventing potential damage to the cells. The threshold for a battery of this type is expected to be 6.4 V (consisting of two 3.7 V cells), which means we anticipate being able to use all 2000 mAh above the A63356 input voltage range before the batteries stop providing power.

2.5.2. Design Decisions

We decided to use lithium polymer battery packs with built-in charging/discharging protection because Professor Gruev recommended them for safety purposes. These batteries have a built-in safety circuit in order to ensure that they do not go beyond certain thresholds. Given this, we did experience firsthand that the batteries are able to malfunction if the ends are shorted, and they will not work as intended and start smoking once you connect them to a working

application circuit. We also decided to choose 7.4 V batteries specifically with a high current output capacity.

For the buck converter, we chose a buck converter that was well documented and would be able to step down a relatively high DC voltage battery to 5 V. The AP63365 fits this criterion due to its low cost and plentiful documentation on its usage. This buck converter is able to take in any input voltage ranging from 6 V up to 32 V, which gives it a very high voltage ceiling on the input voltage - making it very flexible for our needs, and with the proper application circuit it outputs a steady DC 5 V that could be used to power the FPGA.

We decided to not use an additional buck converter to step down the 5 V that would be feeding into the FPGA since the FPGA has a buck converter onboard already that is capable of stepping down voltage by itself to a regulated 3.3 V output line, which is the right voltage we need to power the microcontroller. The 3.3 V output by the FPGA is also rated to output up to 3 A, which is about 5x the absolute maximum current required by the microcontroller and allows for a lot of flexibility.

3. Design Verification

3.1. Wireless Subsystem

Let t_{upload} be the time taken to upload the expected image.

We make assumptions to bound t_{upload} to below 5 seconds. Namely, we use IllinoisNet as the network the server operates on and each user has a 50% chance of actively using bandwidth. Assuming 100 people are connected to a local IllinoisNet access point at 240 Mb/s with fair bandwidth allocation per active user, we allow a tolerance of up to 14 other active users for a 10 MB image to be uploaded within 5 seconds. The probability of simultaneous active users exceeding 14 at any given moment is

$$P = \sum_{k=0}^{14} \binom{100}{k} (0.5)^{k} (0.5)^{(100-k)} \quad (3.1.1)$$

where k steps through all possible numbers of simultaneous active users from 0 to 14. The final probability computed is 1.183×10^{-29}. We have recorded some sample upload times in Table 1 to demonstrate some common case application results.

Table 1: Sample upload times.

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	0.005952
PNG	412.1	3	0.001757
JPG	77.1	3	0.0009966
JPG	22.7	3	0.001949
JPEG	212.2	3	0.001304

Under these assumptions and the result of equation (3.1.1), we can upper bound the expected time needed for a user to upload their image by

$$\mathbb{E}[t_{upload}] \leq 4.99\,s \times (1 - 1.183 \times 10^{-29}) + 5\,s \times (1.183 \times 10^{-29}) \quad (3.1.2)$$

where we upper bound the time taken to upload an image when IllinoisNet is not busy by 4.99 seconds and the time taken to upload an image when IllinoisNet is busy but functional by 5 seconds. The expectation is that t_{upload} is bounded by 4.99 seconds.

3.2. Imaging Subsystem

In order to verify that the FPGA algorithm was correct before even connecting to the printer, we wanted to feed in some data and detect if the pipeline was able to process data correctly. This was done by writing a separate verification testbench module to accompany the FPGA image processing modules. The TestBench allowed for the processing unit to act as a DUT. There were some initial issues with reading and writing as well as attempting to parallelize memory accesses and computation, but eventually, the end result proved that we were able to modify an initial image to be processed through the Floyd-Steinberg algorithm as shown in Figure 9.

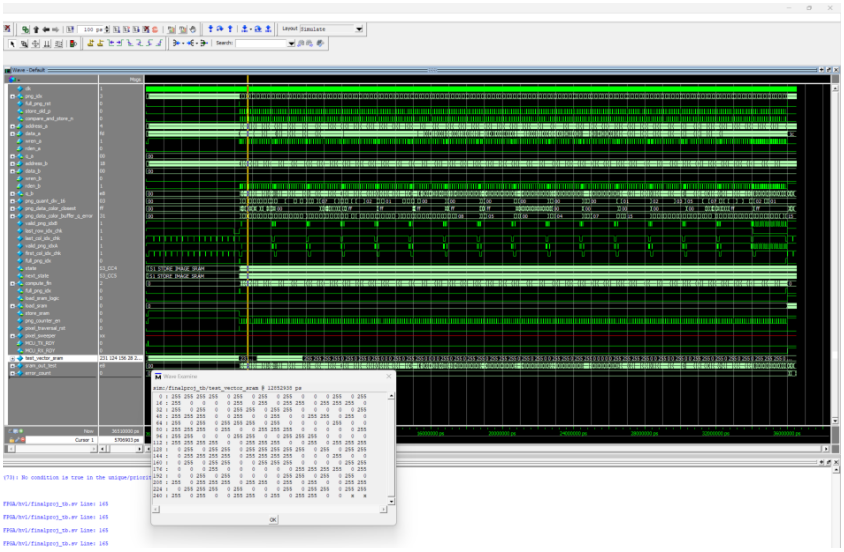

Figure 9: Verification of image data through FPGA pipeline.

Additionally, the Imaging Subsystem manages to process an imaging algorithm faster than software when the image size is large.

Note that we only have the software process the necessary number of pixels, while the FPGA iterates through *all* 65,536 pixels, shown in Figure 6.

We had the MCU perform Floyd-Steinberg dithering on images, where we recorded the time spent performing the function with timing function millis() and other times in Table 2.

Since we had the MCU and FPGA send data back and forth, we have to consider the time taken for the communication protocols to finish. The time taken to transfer the full image data back and forth is

$$(6\ header\ bytes\ +\ 2\ \times\ 65536\ pixel\ bytes)\ \times\ \frac{8\ bits}{1\ byte}\ \times\ \frac{1\ clock\ cycle}{1\ bit}\ \times\ \frac{1\ s}{20{\times}10^{6}\ clock\ cycles} \qquad (3.2.1)$$

where the final result is 52.4132 ms.

The time taken for the FPGA to process the full image data is

$$t_{FPGA}\ \leq\ 65536\ pixels\ \times\ \frac{8\ clock\ cycles}{1\ pixel}\ \times\ \frac{1\ s}{50{\times}10^{6}\ clock\ cycles} \qquad (3.2.2)$$

where the final result is 10.48576 ms.

Table 2: Comparison of MCU process time to Imaging Subsystem process time.

Image Size (Width by height in pixels)	MCU measured mean time (ms)	FPGA calculated time t_{FPGA} (ms)	SPI protocol time t_{SPI} (ms)
8 by *8* (total of 64)	≈ 0	10.48576	52.4132
64 by *16* (total of *1,024*)	12.1	10.48576	52.4132
64 by *64* (total of *4,096*)	48.5	10.48576	52.4132
x by *y* (Total of *65,536*)	779.2	10.48576	52.4132

In conclusion, the software implementation is faster than the Imaging Subsystem for small images only. For image sizes above around 5,500 pixels, the Imaging Subsystem is faster.

The printing times compose the time it takes for the printer to shift up a dot and the time it takes for the MCU to send over all of the bitmap data through serial UART.

The printer shifts up all dots in

$$t_v = 30000 h_b \, \mu s \tag{3.2.3}$$

where h_b is the height of the input image in pixels.

The MCU sends over all of the bitmap data in

$$t_h = \frac{w_b h_b \, bytes}{\frac{8 \, bytes}{1 \, bitmap \, byte}} \times \frac{11 \, bits \times 1000000 \, \mu s + \frac{9600 \, bits/s}{2}}{9600 \, bits/s} \tag{3.2.4}$$

where w_b and h_b are the width and height of the input image respectively in pixels, state bits add three bits per byte, a tolerance of a second is used to bound each byte, and the MCU uses a baud rate of 9600 bits/s. If the input image is maximal at 65,536 bytes, t_h is approximately 9.53 seconds, which is a valid upper bound.

Combining equations (3.1.2), (3.2.1), (3.2.2), (3.2.3), and (3.2.4), we approximate the total time that a job takes and find that jobs with “short” input images with h_b less than 183 pixels can be completed within approximately 20 seconds as the corresponding print time would be bounded by about 15 seconds.

3.3. Board Subsystem

During our final demonstration, we showed that the LCD was very responsive and helped the user pinpoint what stage the job was at, depicted in Figure 10, Figure 11, and Figure 12. The video demonstration [14] shows the LCD's responsiveness as well.

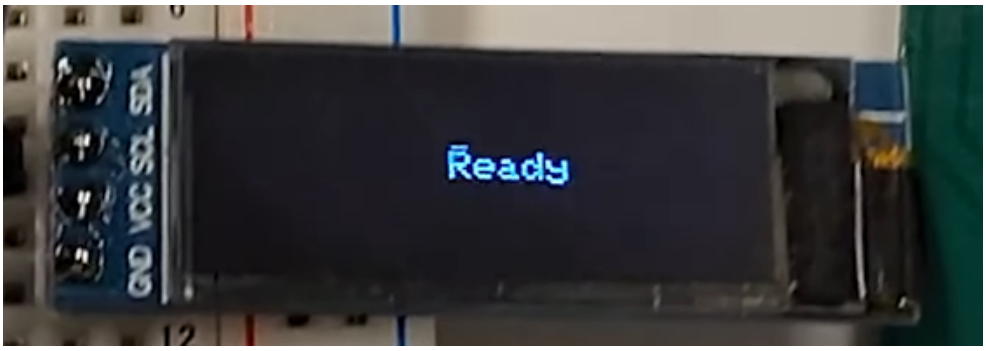

Figure 10: "Ready" status string displayed.

Figure 11: "Printing" status string displayed.

Figure 12: "Completed" status string displayed.

We implemented multiple algorithms by using the switch combination as a selector in a multiplexer, where the inputs are the final pixel values based on the algorithm implementation. We showcased the algorithms during the final demonstration as well.

3.4. Power Subsystem

To check for the functionality of the power subsystem, we first tested if it could step down the 7.4 V from a power supply that could limit its current. This way if there were any issues with the power that came out of the battery, we would know that as it was different from the stable power supply. Upon the first test, it appeared the power subsystem was capable of regulating its voltage and power at a steady pace. Its voltage was a constant 5.05 V which is enough for the FPGA. The next thing to do was to test if the voltage coming out of the battery was enough for the buck converter. At first, it appeared that it was capable of powering the entire project, however, we did notice that there was some dimming that occurred when the printer and everything were running in sync. Overall, it did not appear that there was any difference in the print quality when we plugged the system into the wall versus having it be portable.

The voltages for all of the components in our system are shown in Table 3.

Table 3: Average measured voltages of components.

Component Name	Voltage relative to GND
Printer	7.5 V
FPGA	5.05 V
Microcontroller	3.3 V
LCD	3.3 V
Battery	8.1 V (when full)

We fall short of the high-level requirement of lasting over 1.5 hours in our final design, and we can model this mathematically using the assumption used in the Power Subsystem that the cutoff voltage for a single 7.4 V lithium polymer battery in our system is 6.4 V (when each cell measures 3.2 V), and therefore can power the entire printer system until (printer within 5.0-9.0 V range and AP65336 within 6.0-32.0 V) it reaches this cutoff and all 2000 mAh of energy have been consumed.

As an absolute lower bound estimate on the battery life, let us assume an upper bound on the maximum current consumption of the printer system. The printer itself peaks at 2.0 A of current draw throughout printing, the FPGA requires 500 mA maximum for whole board operation, and the FPGA 3.3 V line powers the LCD and ESP32 MCU. The LCD requires an average of 20 mA, and the ESP32 requires 100 mA for WiFi receive and an additional 240 mA for WiFi transmission/scan at its greatest range setting, but we assume the current for these are provided by the FPGA under its 500 mA usage. If the fully charged battery needs to supply all these components simultaneously without stopping, we can solve for the battery life as follows:

$$Battery\ Life\ =\ 2000\,mAh\ \div\ (2.0\,A\ +\ 0\,.50\,A)\ =\ 0.80\,hours \qquad (3.4.1)$$

While our printer system is indeed operated solely by batteries, this lower bound on the battery life falls well short of our initial expectation of 1.5 hours in our proposal, as we had originally combined the total energy (2000 mAh per battery x 2 battery) from the two batteries we had ordered, but we were only able to use one of the batteries in our final design due to a battery failure. This does not mean the printer can operate for 1.5 hours under optimal conditions, or even standard use, rather, this is following our high level requirement exactly per specification of “worst-case battery life operating time of at least 1.5 hours”, as outlined in equation (3.4.1).

4. Costs

4.1. Parts

Table 4: Parts costs list.

Item Name	Item Amount	Item Unit Price	Item Total Price	Did *not* use My.ECE funds?
USB C PD Boards	1	$9.99	$9.99	
LED Green	10	$0.13	$1.28	
LED Red	10	$0.13	$1.28	
ESP32 MCU	2	$3.95	$7.90	
ESP32 MCU Breakout Boards	Pack of 5	$19.99	$19.99	✓
ESP8266-12F MCU	Pack of 5	$9.88	$9.88	
ESP8266-12F MCU Breakout	Pack of 5	$12.99	$12.99	✓
AP63356 Buck Converter	3	$0.87	$2.61	
22 kΩ Resistor	30	$0.02	$0.63	
MMBT222A BJT	10	$0.11	$1.15	
CP2104 USB UART Bridge	3	$2.91	$8.73	
93.1 kΩ Resistor	10	$0.04	$0.36	
10 µF Capacitor	10	$0.07	$0.67	
22 µF Capacitor	10	$0.08	$0.84	
6.8 µH Inductor	3	$1.15	$3.45	
100 kΩ Resistor	100	$0.04	$3.80	
4.7 µF Capacitor	50	$0.02	$1.05	
USB A Female	2	$1.33	$2.66	

220 kΩ Resistor	5	$0.12	$0.60	
47 pF Capacitor	10	$0.08	$0.84	
Adafruit Thermal Printer	1	$49.99	$49.99	
7.4 V RC Battery	Pack of 2	$19.99	$19.99	
1 µF Capacitor	50	$0.04	$1.78	
0.1 µF Capacitor	10	$0.04	$0.44	
10 V, 3 A Schottky Diodes	4	$0.50	$2.00	
Orange LEDs	10	$0.05	$0.46	
STUSB4500 USB Chip	2	$3.49	$6.98	
10 kΩ Resistor	100	$0.01	$1.15	
2.2 kΩ Resistor	100	$0.01	$1.15	
1 kΩ Resistor	100	$0.03	$2.97	
Tactile SMD Switches	10	$0.32	$3.21	
Zener Diodes for ESD	10	$0.41	$4.08	
USB C Connector	3	$0.81	$2.43	
I2C OLED LCD	Pack of 2	$7.99	$7.99	✓
DE10-Lite FPGA	1	$0.00	$0.00	✓
Total	N/A	$147.58	$195.32	N/A

4.2. Labor

The amount of time spent on the project varies depending on the portion.

According to the Grainger College of Engineering, the average starting salary for an electrical engineering graduate is $80,296 [13], so if we take this number and assume they have a 52-week year and work 40 hours a week, we can determine the average hourly salary to be $38.60.

Table 5: Estimated labor costs at $38.60/h.

Person(s)	Task	Hours Spent
Gally	PCB Design	50
Jason	SPI Controller + MCU program	60
Gally	Floyd-Steinberg Accelerator	50
Kevin Jason	Web Server	80 combined
Jason Gally Kevin	MCU Design	25 combined
Kevin	Soldering	15
Gally Jason Kevin	Writing and Bookkeeping	150 combined
Total	N/A	430

Based on Table 5, we expect the total labor costs to be

$$\frac{\$38.60}{h} \times 430\,h \times 2.5 = \$41,495$$

5. Conclusions

5.1. Accomplishments

We were able to present a functional, relatively fast printer system that is portable in that it fits within a small box. During the final demonstration, arbitrary users connected to IllinoisNet were able to upload arbitrary images of format JPG, PNG, or JPEG to the server, which were printed out on thermal paper shortly after. Although we cannot mathematically guarantee meeting the high level requirement of having a worst-case system battery life of 1.5 hours, we certainly meet the requirements of taking up a footprint less than 12 inches by 12 inches and spending under 20 seconds for a job, and above all we assert that our design is functional in operation as a printer should be.

5.2. Ethical Considerations

Our project aims to create a convenient and speedy printing solution for consumers. As such, we must ensure that the design of the printer system is safe for users to handle and interact with on a consistent basis. While many of the components listed are found in everyday objects and are consumer-grade, there are some components that may pose risks in their usage. According to the IEEE Code of Ethics I.1 [1], we must remain transparent with our design process and disclose the factors that might endanger the public and/or environment.

One concern is the use of battery packs. These batteries are lithium-ion rechargeable batteries, which, due to their energy density, makes them susceptible to explosion when in contact with high temperatures or manufacturing defects. This was demonstrated in the Samsung Note 7 smartphone case study [8], where the battery packs caught on fire due to a defect in the phone design that allowed the battery leads to touch and short circuit. We followed the safety manual [12] for guidance.

5.3. What We Learned

We should decide what components to use before we actually start physically attaching them in the lab. We wasted a lot of time and money attempting to migrate our old MCU to the new one. The smallest of changes may require an overhaul of the entire project.

Additionally, we should be *extremely* wary of shorts, wires, and batteries. This caused one of our laptops to break and one of the batteries that came in the case to explode.

Lastly, engineering is hard but rewarding!

Appendix A. Requirement and Verification Tables

Table 6: Wireless Subsystem RV table.

Requirement	Verification
● Connected users can upload an image successfully quickly to the server. ● Images will need to be delivered to the server within 5 seconds of upload to reduce the amount of overall delay for the user.	● Starting with activating the server, at the time of user upload, we will manually check the time using a stopwatch (i.e., with a cellphone) to determine that the data was received on the server within 5 seconds. ● Upon a successful user upload, the image that is delivered to the server should appear in the host computer's local storage, as in the directory that the server code is active on. This will allow us to verify that the necessary data has been received in full within the given time and that the custom API is successful in this requirement. ● We will repeat this for many scenarios, such as having multiple users on the server simultaneously, varying image sizes and resolutions, and at different distances from the host computer.
● The ESP32-WROOM microcontroller requires 3.0-3.6 V continuously from the power subsystem for operation.	● Ensure the power system is active without having the MCU soldered in place. We will use an oscilloscope to measure the voltage across the Vin and GND pins on the breakout PCB to ensure that the voltage range is safe for the MCU operation. ● Record the voltages throughout many different points of operation of the battery life to ensure that the Vin does not fluctuate outside of the 3-3.3 range. Any higher than 3.6 V will damage the MCU.
● The ESP32-WROOM microcontroller can receive asynchronous image data from a WebSocket client.	● The server sends the image data as bytes to the MCU through a TCP connection.

Table 7: Imaging Subsystem RV table.

Requirement	Verification
● Processing an image with the FPGA should be faster than processing it with the microcontroller.	● We will run a timing test between the microcontroller (software) and the FPGA (hardware). By doing this, we will be able to clock the time it takes to process the image using the FPGA and the amount of time it takes to process the image using the microcontroller. The FPGA time should be relatively faster than the MCU's implementation of the image processing to demonstrate hardware acceleration. o The MCU records timings with the millis() function, which returns milliseconds elapsed. o Then we will be able to figure out the time on the FPGA by multiplying the inverse clock (50 MHz) by the amount of clock cycles needed to process each pixel by the amount of pixels in the image.
● The FPGA should be able to function by itself as soon as it is turned on without having to be flashed again by an external source – simulating a commercial printer.	● Upon the plugging in of all the components, we should check to see if the original FPGA ROM is loaded into the FPGA. If it is not (the LEDs are not blinking rapidly), then we would know that the FPGA has successfully loaded in our correct .pof file (loaded to flash ROM).
● The temperature of the surface of the enclosed system (with the imaging subsystem) must not exceed 120°F.	● The temperature of the printer system as a whole is important for user safety, in which it cannot exceed a recommended temperature limit of 120°F [9] for handling. ● During the startup, idle, and active printing stages of operation we will measure the temperature of many surfaces of the printer enclosure using an infrared thermometer. ● We will record the temperatures of critical regions like the printer. The printer has a built-in function to print out a test page that has the internal temperature displayed on it.

Table 8: Board Subsystem RV table.

Requirement	Verification
● The diagnostic LCD must be highly responsive if the system status changes.	● Start at a known system state where the diagnostic LCD is outputting a steady state. ● Change the system state by changing the state. At this point, start recording time elapsed on a stopwatch. ● Stop the stopwatch when the diagnostic LCD updates its display to the correct status. ● Record the start-stop readings and the time taken. Repeat for all possible system status changes.
● The “algorithms” switches controls the image processing algorithm that the FPGA uses.	● Start with all switches in the OFF position. Set the switches to a valid algorithm mapping. ● We will write a TestBench file that allows us to analyze the value per bit after the FPGA processes the image. ● Since the algorithms are deterministic, we can verify the final image with a C++ script that performs the same algorithm on the same image.
● At startup, the diagnostic LCD status should read “Ready” before printing.	● Check that there is enough paper in the printer beforehand, and that the system is idle (i.e., just starting up). ● Power the system on by plugging in the battery pack to power and ground. ● The microcontroller should be programmed to display “Ready” on the LCD once connected to the internet and server WebSocket.
● While the system is on, start a printing job with the paper loaded. The status displayed should be “Completed” when the thermal paper finishes printing (the image is visible on paper and there are no printing noises).	● Check that there is enough paper in the printer beforehand, and that the system is idle (i.e., just starting up). ● Power the system on by plugging in the battery pack to power and ground. ● Upload any valid image to the server. ● Upon completion of the printing job, we will manually verify if the LCD displays the word “Completed”.

Table 9: Power Subsystem RV table.

Requirement	Verification
● This subsystem must generate sufficient power to the entire subsystem.	● We will measure the voltages across other components with a multimeter (insert the leads into the pin and ground). The voltage ranges accepted for verification are: o ESP32-WROOM MCU 3.0 - 3.3 V o Thermal printer 5.0 - 9.0 V o FPGA 5.0 V ± 5% o LCD 3.3 V ± 5%
● The subsystem is safe in that it does not exceed the hard limit of 20 W.	● While the system is active, ensure that there is no power overage by measuring the voltage across each component and the current through each component. It should not exceed our cap of 20 W.
● The subsystem must be able to provide power to the entire system for 1.5 hours from full charge without recharging during this time.	● Verify that the batteries are fully charged by measuring the voltage drop across it with a multimeter or oscilloscope. This voltage drop should be at least 7.4 V. If not, then charge the batteries until this condition is true. ● Start a stopwatch as a user starts the first job since the system turned on by uploading an image to the local server. ● Have the user constantly upload images as the printing jobs are completed. ● If the stopwatch records 1.5 hours and the system is still active, this requirement has been verified. Otherwise, this requirement has failed.

Appendix B. PCB Design

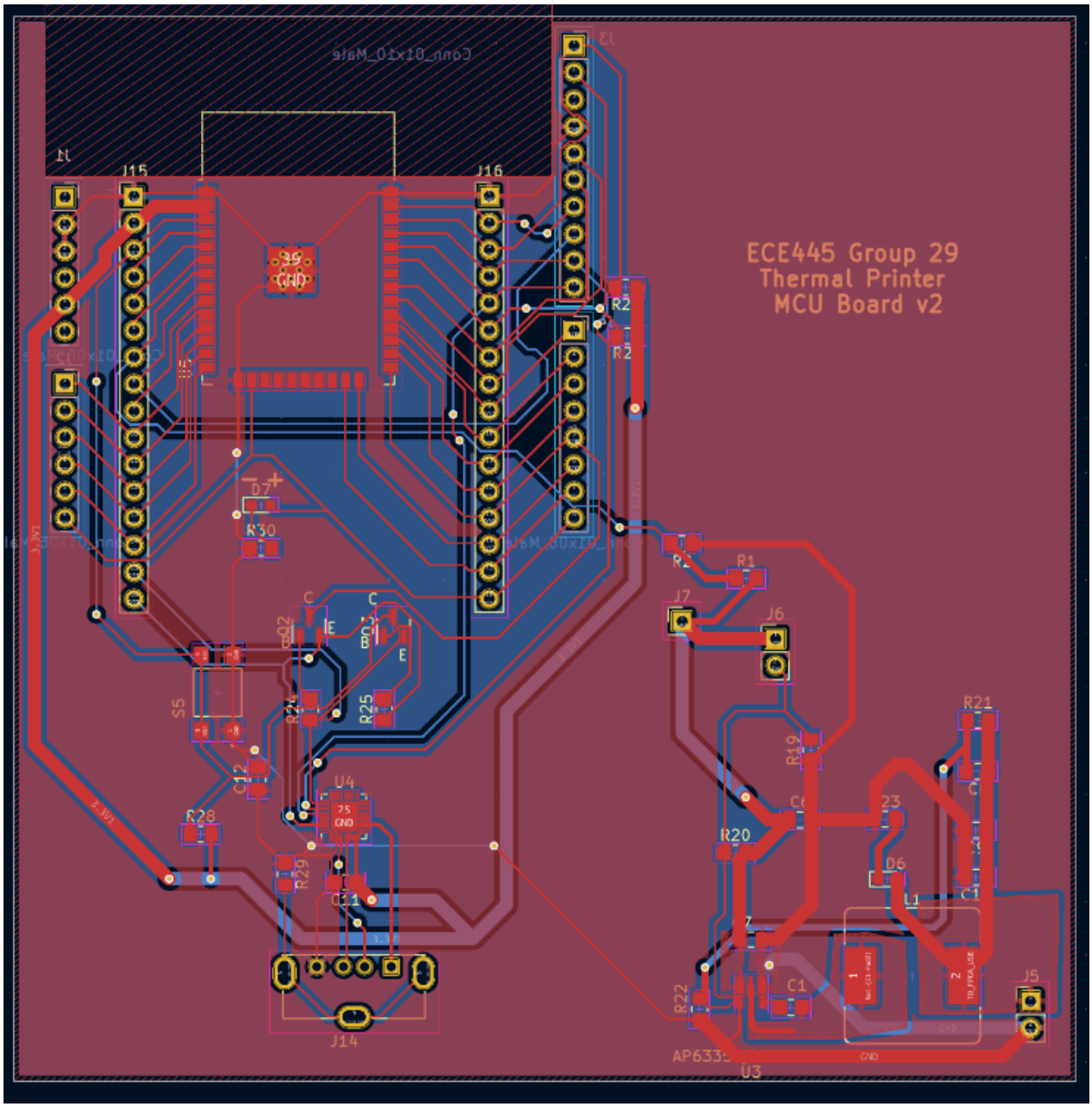

Figure 13: MCU PCB.

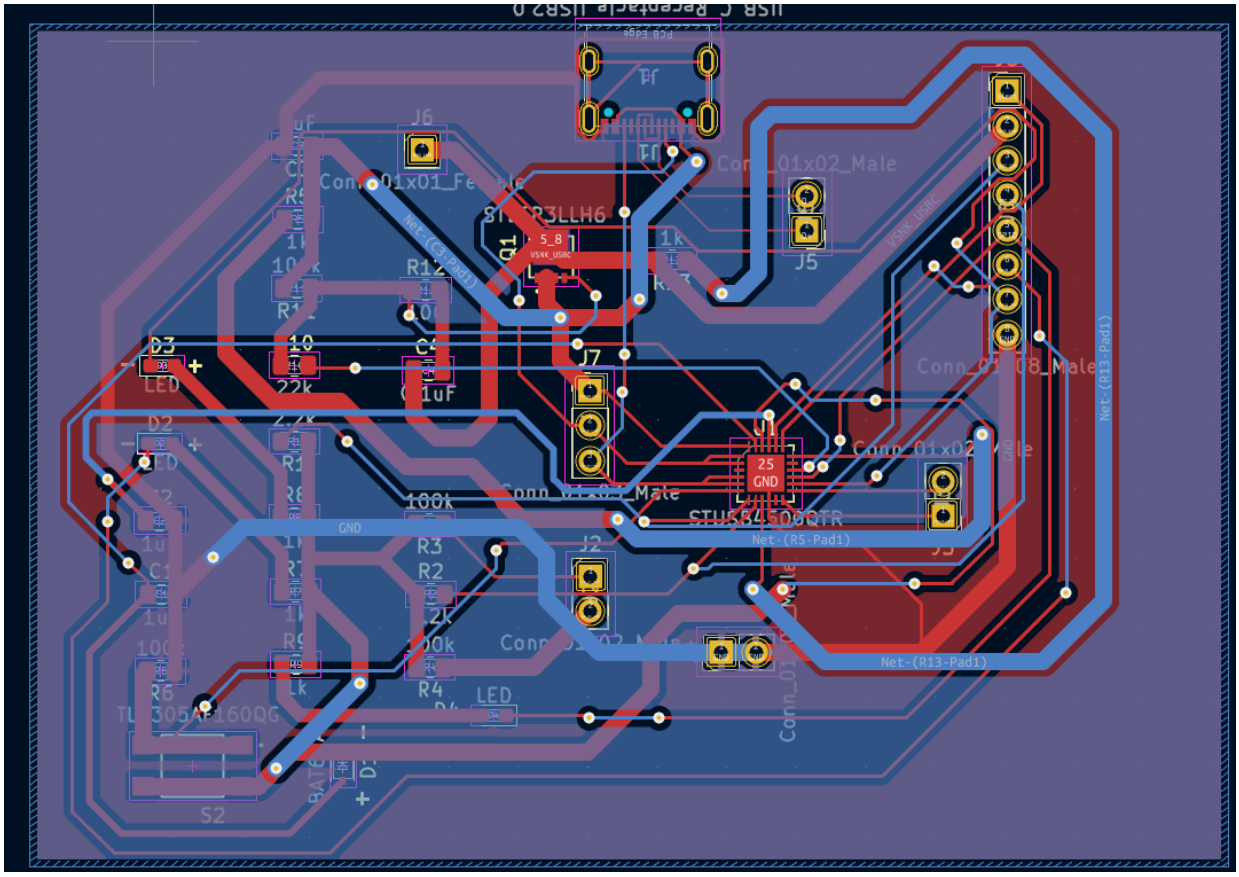

Figure 14: USB C PD PCB.

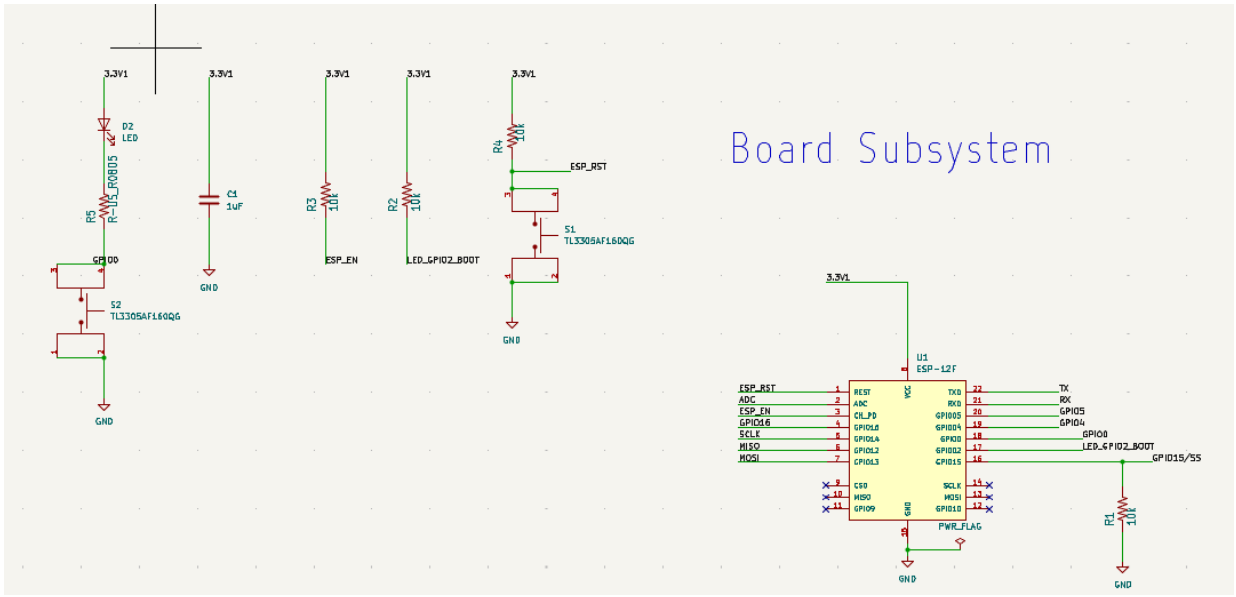

Figure 15: Board Subsystem of MCU board.

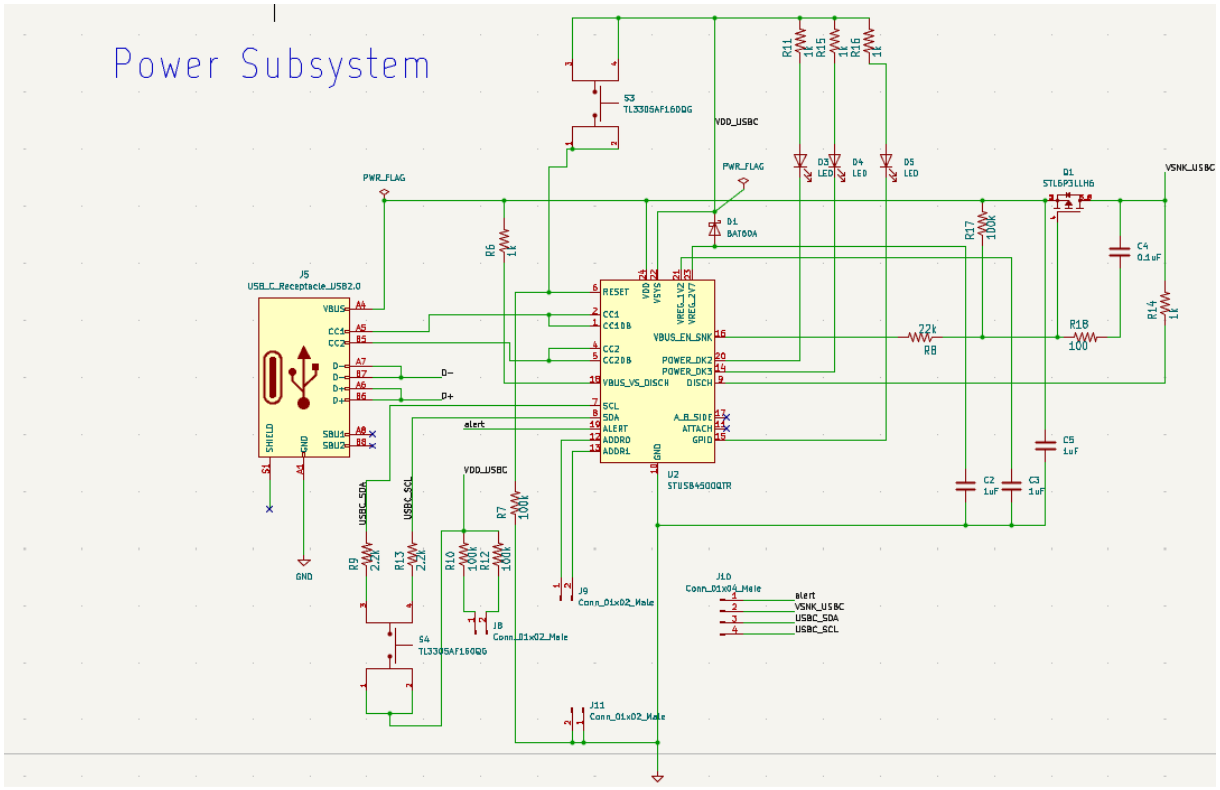

Figure 16: Power Subsystem of MCU board.

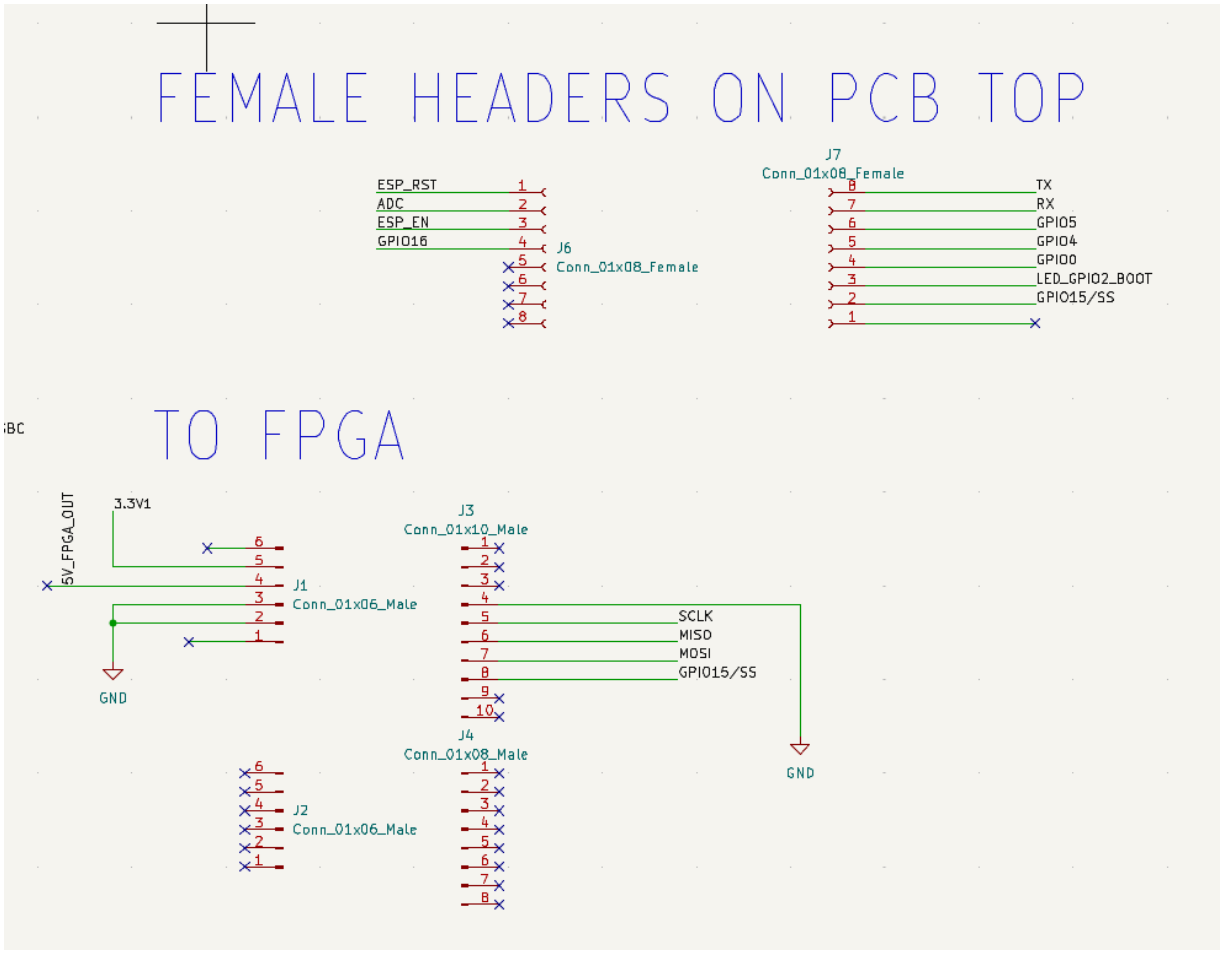

Figure 17: Female headers to and from the FPGA on MCU board.

References

[1] *IEEE Policies, Section 7 - Professional Activities (Part A - IEEE Policies*), IEEE Code of Ethics 2020.

[2] “Adafruit Industries,” *SSD1780*, Apr-2008. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf. [Accessed: 04-May-2023].

[3] Expressif Systems, “ESP32 series - Espressif,” *ESP32 datasheet*, Jan-2023. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. [Accessed: 04-May-2023].

[4] “Diodes incorporated - analog, discrete, logic, mixed-signal,” *AP63356, AP63357*, Aug-2019. [Online]. Available: https://www.diodes.com/assets/Datasheets/AP63356-AP63357.pdf. [Accessed: 04-May-2023].

[5] “Floyd–Steinberg dithering,” *Wikipedia*, 24-Apr-2023. [Online]. Available: https://en.wikipedia.org/wiki/Floyd%E2%80%93Steinberg_dithering. [Accessed: 03-May-2023].

[6] T. Helland, “Image dithering: Eleven algorithms and source code,” *tannerhelland.com*, 28-Dec-2012. [Online]. Available: https://tannerhelland.com/2012/12/28/dithering-eleven-algorithms-source-code.html. [Accessed: 03-May-2023].

[7] Terasic, "DE10-Lite User Manual," Jun. 05, 2020.

[8] R. Rox, "Samsung Galaxy S7 explodes during charge," Notebookcheck, Sep. 03, 2017. [Online]. https://www.notebookcheck.net/Samsung-Galaxy-S7-explodes-during-charge.246242.0.html [Accessed: 08-Feb-2023].

[9] Johns Manville, "Too Hot to Handle?," www.jm.com, Feb. 25, 2015. https://www.jm.com/en/blog/2015/february/too-hot-to-handle/ (accessed Feb. 08, 2023).

[10] When your project involves interfacing peripherals with a microcontroller, “SPI vs I2C protocols - pros and cons,” *Arrow.com*, 04-Feb-2019. [Online]. Available: https://www.arrow.com/en/research-and-events/articles/spi-vs-i2c-protocols-pros-and-cons. [Accessed: 03-May-2023].

[11] Adafruit, “A1 micro panel printer user manual.doc”. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/A2-user%20manual.pdf [accessed 23-Feb-2023].

[12] ”Safe Practice for Lead Acid and Lithium Batteries”, General Battery Safety, Senior Design Spring 2016 Course Staff, Apr. 2016. [Online]. Available: https://courses.grainger.illinois.edu/ece445/documents/GeneralBatterySafety.pdf

[13] Grainger Engineering Office of Marketing and Communications, “Electrical Engineering,” *The Grainger College of Engineering | UIUC*. [Online]. Available: https://grainger.illinois.edu/academics/undergraduate/majors-and-minors/electrical-engineering. [Accessed: 03-May-2023].

[14] J. Liu, *ECE 445 [Team 29] Thermal Portable Printer*. 2023.