

ARIZONA STATE UNIVERSITY

HONORS THESIS

Building a Mobile Device that Leverages the Power of a Desktop Computer

Author:

Dylan LATHRUM

Supervisor:

Dr. Robert HEINRICHES

*A thesis submitted in fulfillment of the requirements
for the degree of Software Engineering*

for

Barrett, The Honors College

April 3, 2022

Declaration of Authorship

I, Dylan LATHRUM, declare that this thesis titled, "Building a Mobile Device that Leverages the Power of a Desktop Computer" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

“Premature optimization is the root of all evil.”

Sir Tony Hoare

ARIZONA STATE UNIVERSITY

Abstract

Barrett, The Honors College

Software Engineering

Building a Mobile Device that Leverages the Power of a Desktop Computer

by Dylan LATHRUM

With the recent focus of attention towards remote work and mobile computing, the possibility of taking a powerful workstation wherever needed is enticing. However, even emerging laptops today struggle to compete with desktops in terms of cost, maintenance, and future upgrades. The price point of a powerful laptop is considerably higher compared to an equally powerful desktop computer, and most laptops are manufactured in a way that makes upgrading parts of the machine difficult or impossible, forcing a complete purchase in the event of failure or a component needing an upgrade.

In the case where someone already owns a desktop computer and must be mobile, instead of needing to purchase a second device at full price, it may be possible to develop a low-cost computer that has just enough power to connect to the existing desktop and run all processing there, using the mobile device only as a user interface.

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor...

Contents

Declaration of Authorship	iii
Abstract	vii
Acknowledgements	ix
List of Abbreviations	xvii
1 Introduction	1
1.1 Power and Portability	1
1.2 Purpose of this Thesis	1
1.3 Thesis Overview	1
2 Background	3
2.1 Specialization of Computers	3
2.1.1 Power of the Desktop	4
2.1.2 Convenience of the Laptop	4
2.1.3 Rise of the Gaming Laptop	5
2.2 Thin Clients	5
2.3 Application to Modern Day	5
3 State of the Art	7
3.1 Software Solutions	7
3.1.1 Remote Desktop Protocol	7
3.1.2 Virtual Network Computing	8
3.1.3 Chrome Remote Desktop	8
3.1.4 Secure Shell Protocol	8
3.2 Hardware Solutions	9
3.2.1 Thin Clients	9
3.2.2 Nvidia Shield and GameStream	9
3.3 Research Questions	10
4 Developing the Hardware	11
4.1 Requirements	11
4.1.1 Performance	11
4.1.2 Cost	12
4.2 Choosing Parts	12
4.3 Prototyping	15
4.4 Designing the Printed Circuit Board	16
4.4.1 Power	16
4.4.2 USB	18
4.4.3 HDMI	19
4.4.4 CM4 and Miscellaneous	20
4.5 Manufacturing the Circuit Board	20

4.5.1	First Attempt	21
4.5.2	Second Attempt	22
4.5.3	Third Attempt	23
5	Developing the Software	25
5.1	Requirements	25
5.2	Potential Avenues	25
5.2.1	NVIDIA GameStream	26
5.2.2	Moonlight	27
5.3	Developing for ARM	27
5.4	Developing Testing and Measurement Tools	29
6	Evaluation	31
6.1	Testing Methodology	31
6.2	Responsivity and Latency	32
6.3	Performance	32
6.4	Quality	32
6.5	Summary	32
7	Conclusion	35
7.1	Summary	35
7.2	Limitations	35
7.3	Future Research	35
A	PCB Specifications	37
A.1	PCB Schematic	37
A.2	PCB Layout	41
B	Bill of Materials	43
	Bibliography	45

List of Figures

2.1 Thin Client	5
3.1 Nvidia Shield	9
4.1 Raspberry Pi Compute Module 4	13
4.2 Raspberry Pi Compute Module 4 Pin out	13
4.3 USB Controller	16
4.4 SMD Breakout Board	16
4.5 PCB Power Circuit	17
4.6 Power Schematic	17
4.7 PCB USB Circuit	18
4.8 USB Schematic	18
4.9 The PCB's HDMI circuit	19
4.10 The circuitry surrounding the CM4's two mezzanine connectors	19
4.11 USB Schematic	20
4.12 PCB Stencil	21
4.13 Hand-soldered mezzanine connector	22
4.14 Two versions of the QFN-20-1EP_4x4mm package	22
4.15 Assembled PCB	23
6.1 Input Delay Data	33
6.2 Color Delay Data	33
6.3 Color Delay Loss Data	34
A.1 Top Level Schematic	37
A.2 Power Schematic	38
A.3 CM4 Schematic	39
A.4 USB Schematic	40
A.5 PCB Front Layout	41
A.6 PCB Back Layout	42

List of Tables

4.1 Selected PCB Components	15
B.1 Bill of Materials	44

List of Abbreviations

BOM	Bill Of Materials
CM4	Raspberry Pi Compute Module 4
CPU	Central Processing Unit
CRD	Chrome Remote Desktop
CSV	Comma Separated Values
FFC	Flat Flexible Cable
FPS	Frames Per Second
GPU	Graphics Processing Unit
IDE	Integrated Development Environment
PC	Personal Computer
PCB	Printed Circuit Board
RDP	Remote Desktop Protocol
RTC	Real Time Clock
SBC	Single Board Computer
SMD	Surface Mounted Device
SSH	Secure Shell (Protocol)
VNC	Virtual Network Computing

For/Dedicated to/To my...

Chapter 1

Introduction

1.1 Power and Portability

TODO

1.2 Purpose of this Thesis

TODO

1.3 Thesis Overview

TODO

Chapter 2

Background

Computers have a long history of being built to serve a particular purpose, often compromising some aspects of the system to bolster others. Chapter 2.1 discusses the history of how computers were built for a particular function, and how the advancement of technology has brought the “Personal Computer” seen know today. Chapter 2.1.1 discusses the benefits of a desktop computer, Chapter 2.1.2 discusses the benefits and drawbacks of a laptop computer, and Chapter 2.1.3 discusses the rise of the gaming laptop and the closing gap between power and portability at the cost of increased price. Chapter 2.2 briefly discusses thin clients, a corporate solution for remote access to a computer. Chapter 2.3 introduces current world wide context that has an impact on the computing world and the need for power and portability.

2.1 Specialization of Computers

In the age of punch cards and monolithic computers the size of entire rooms, it was expected of a computer to only be capable of serving a single function. Before the technology could be miniaturized, every single computer had to be specialized for the job at hand. As time passed and technology progressed, a single computer could be produced that served multiple functions as well as be adaptable programmable to handle new tasks that were not considered when the machine was originally built. This gave rise to what was known as “Autonomic Computing”, a system of characteristics that are built into a computer to help it self-manage its resources and adapt to its administrator’s requirements without the need to be redesigned for its new purpose [18]. Autonomic Computing was designed to combat the exponential complexity crisis that came from the widespread availability of computers in different disciplines, since now anyone who could afford a computer could program it for any purpose. As new use cases were found for computers in day-to-day life and new technologies were being developed to interface with the world around us, the need for componentization became ever more apparent. Researchers at IBM knew that the best way to address the looming problem of runaway complexity in computers was to develop a way for the computers to automatically interface with any new components that are installed, and to configure itself to only use what is necessary for the job at hand [18, p. 43]. This allowed hardware and software developers to focus on building their products to work with a common standard rather than having to manually integrate their products with every computer.

Once the idea of modularity began to take hold in the computer industry, attention was turned once again towards specializing individual computers. Now that a computer’s physical footprint can be minified, and peripheral components can be added and removed without a complete reengineering of the device, a single computer can be optimized to handle a single task without the overhead cost of

building a monolithic computer [4]. Now a computer can be specialized to serve a particular purpose or set of purposes while keeping development time comparatively short. The world has seen this realized through numerous applications such as cell phones, designed to be user-friendly portable devices, computing clusters, built for high-performance computing, or even internet routers, which are built to be a plug-and-play solution for a problem posed to users of all levels of familiarity with computers. Without this idea of Autonomic Computing, each of these devices would have to be reengineered from the ground up every time a new use case was developed.

This leads into the specialization of home computers in from the perspective of a consumer. It used to be that a “home computer” was a device that was bought off the shelf as-is and served it’s purpose. Now a home computer can be a desktop PC that can be upgraded with time and seldom moves from it’s place, or it could be a laptop that is portable and easy to cary around. These specializations bring more choices for the consumer to pick a computer that best suits their needs, but they often come with compromises.

2.1.1 Power of the Desktop

The classical manifestation of a personal computer is the desktop computer. Known for it’s componentization, user repairability, and direct connections to power and the network, desktop computers are the best option for a user looking for a workhorse system. Pieces of the computer could be bought separately, and single parts could be upgraded or replaced by anyone will a little hardware knowledge. Even though they aren’t very portable, desktop computers allow consumers to access greater processor power for a lower cost compared to portable devices [21]. Due to this, a desktop PC is often seen at one’s home or place of work hardwired to the wall where it remains unmoving unless it needs upgrading, cleaning, or repairs. Adding on the fact the desktop is modular, it can be easily customized to fit any user’s needs, as well as enable particular parts to be replaced or upgraded without needing to rebuild the entire system.

While desktops were the staple of personal computers for decades, advancing technology and the rising need of portability and flexibility has led to the growth of laptops.

2.1.2 Convenience of the Laptop

As the world moved towards a more mobile lifestyle, the laptop shifted from being a luxury to being a necessity. People began to need computing power on the go, whether it be for working remotely, attending classes and following along, or simply needing more power than their phone can offer. Companies began to take note too and started building applications to work fully within a web browser, such as Google’s implementation of Google Docs, allowing users to edit documents on the go without having to install software like Microsoft Word directly to their computers. This shift towards mobility provided the general public the ability to be more flexible with how they used computers, no longer requiring them to dedicating a spot in their house to be taken up by a desktop computer.

Reducing the barrier to entry for computers, such as the knowledge to buy the computer the best fits their needs and the restrictions that come with setting up and keeping a desktop, led to the widespread adoption of laptops. In fact, by 2021, laptops accounted for 80% of total computer sales [28]. For most consumers, the

laptop is all they need; it may not have the raw power of a desktop, but it handles everything they need it to do. The biggest limiting factor then becomes technology, with manufacturers constantly trying to minimize existing desktop technology into the footprint of a laptop while keeping energy efficiency high enough to be feasible in a mobile device.

2.1.3 Rise of the Gaming Laptop

All computer products are a balance, it simply isn't possible to have the perfect combination of portable, powerful, serviceable, and cheap. While desktops focus on power and serviceability, laptops often focus portability at the expense of other attributes. As mobility becomes ever more important, gaming laptops have become more popular to bring power to the portable form factor. This understandably comes with a cost, with the literal cost of a gaming laptop being the most obvious compromise. The componentization of such a laptop also often takes a hit with parts being directly soldered to the motherboard, reducing a user's options for repairs or upgrades. It is more expensive, and often less user-serviceable, to buy a gaming laptop that has the same performance of a desktop computer – but given that it is an all-in-one device, it's portability gives it a niche that a desktop cannot hope to rival. The compromise presented by a gaming laptop helps close the gap between power and portability, but it still falls short of being an all around good solution.

2.2 Thin Clients

On the corporate side of computing, the solution for portability has been the thin client. Rather than integrating the power of a desktop to a portable machine, a thin client will connect to a powerful server and provide a thin interface to the user. This allows the user to utilize the power of the host machine from a cheap computer that usually doesn't have enough power to be used as a standalone computer. Because of its low power, the thin client cannot be used as a standalone computer, but its low cost and portability make it a great solution for a corporate user. While these clients work well for simple interfacing tasks such as data entry or system management, they are not designed to be used for complex tasks that require quick response times or graphical fidelity.



FIGURE 2.1: A thin client next to a traditional desktop computer [11].

2.3 Application to Modern Day

In 2022, the world is currently going through some chaotic times that demand flexibility and adaptability like it hasn't faced before. From the COVID-19 pandemic to massive chip shortages and supply line disruptions, people are now more than ever looking for a way to be flexible in their work and personal lives now and in the future. The COVID-19 pandemic has forced people to be able to operate remotely, whether that be for work, school, or communicating with friends and family [19]. Many office workers used to working on a desktop computer onsite are now needed

to work from home, or classes usually taught in person are now held remotely [7]. This shift to remote operation has fueled the need for mobile computers, many of which need to be powerful enough to handle the workload of a desktop computer from anywhere in the world. The global chip shortage is also causing issues for the purchase of new devices [16]. Parts which are usually readily available for purchase are out of stock with months of delay on back order. Resellers are taking advantage of the situation by hiking prices to two or three times the original listing price, and it's become harder than ever to build a new PC [29]. All of these factors combined lead to a very difficult situation for anyone looking to purchase a new computer or parts for their existing computer.

It doesn't make sense to build or purchase a powerful PC for use at home and another power laptop to handle working on the go, especially with rising prices, so many people are moving to exclusively powerful laptops or looking for other solutions.

Chapter 3

State of the Art

This chapter seeks to provide a summary of existing solutions for remote computer access. It will not cover specific applications, but rather the protocols that are used to communicate with a remote computer. Generally speaking, there are two types of solutions that have been developed, software solutions, which are covered in Chapter 3.1 and hardware solutions, which are covered in Chapter 3.2. Various protocols and implementations are covered in the subsections of those two categories.

3.1 Software Solutions

While there are many applications in existence that allow a user to remotely control a computer from another location, few of them offer the capabilities required to stream high-performing applications to a remote device. The following subsections will introduce existing solutions to this problem as well as drawbacks that come with each implementation.

3.1.1 Remote Desktop Protocol

Microsoft's Remote Desktop Protocol (RDP), is a protocol that defines communication between a terminal server and terminal server client for multimedia purposes [31]. Coming pre-installed on every Windows machine since Windows XP and with clients available for Windows, Mac, and Linux, RDP is an easy to use solution for remotely accessing Windows machines graphically.

Although it is the built-in solution for Windows machines, it isn't without its drawbacks. Firstly, only one graphical session is active at one time while using windows. This means if a user is logged into the host computer and someone initiates a connection with RDP, the user using the host computer will be logged out. While this isn't always an issue, sensitive programs that don't take well to being logged out may run into issues. Secondly, RDP does not support relative mouse movement [5]. This means every mouse input is sent to the host computer as an absolute position on the screen, rather than as a relative distance from its previous location. This means any program that moves the user's mouse for them, such as 3D applications with a virtual camera, will be sent the absolute position of the client's mouse. This will cause the application to detect a large movement of the mouse instantly, which can result in erratic camera movement as the program interprets the mouse moving a large distance every time the cursor is reset to the center of the screen. Again, while this isn't always an issue, tasks such as 3D modeling and animation or game development cannot be controlled properly.

3.1.2 Virtual Network Computing

Virtual Network Computing (VNC), is a platform-independent system originally developed by Olivetti & Oracle Research Labs and later bought and shelved by AT&T [30]. While the original implementation is no longer used in a wide capacity, the protocol it was developed on has been expanded and improved to become one of the most flexible yet simple ways to control a computer remotely. VNC can run on any modern operating system, and even a web browser can serve as a VNC client. It was originally built as the simplest form of remote computer control over LAN, allowing system administrators to control almost any kind of computer with a simple, robust, and extremely compatible system. However, since its use has gradually outgrown its original scope, VNC isn't usually the right tool for any job greater than remote system management. Because of its simplicity and focus on being as straightforward as possible, VNC is often found to be lacking in terms of security and speed [30]. It is not recommended to use VNC over the internet without some secondary form of security, such as SSL or a secure VPN.

3.1.3 Chrome Remote Desktop

Should this chapter be called Chromoting?
"Chrome Remote Desktop" is the much more recognizable name though

Google Chromoting, implemented publicly as Chrome Remote Desktop, is an open source protocol that enables remote desktop control through any web browser running on the Chromium engine [6]. It prioritizes low bandwidth usage and ease of access, allowing easy server install on every major desktop operating system, and enables any web browser or mobile device to act as a client. By employing the VP8 compression format (Most commonly seen in internet video streaming), Chromoting is able to keep network traffic low by only sending a few full images of the server's screen as keyframes and transmitting motion with compressed partial frames [13]. These partial frames are calculated by the server using the difference between the previous and current frame, allowing the server to only send the difference between the two frames.

This save in bandwidth and processing done by the client makes it the perfect solution for Google's ChromeOS laptops coming out around the same time. These low powered machines were built to be laptops running on the power of a web browser with a link back to more powerful computers when the tasks were too great [32]. While the protocol works well for remote access, it is limited in certain applications by its conservative approach to bandwidth usage and reliance on the web browser.

3.1.4 Secure Shell Protocol

The Secure Shell Protocol (SSH), is a protocol for securely transmitting data over an insecure network [33]. Usually used for remote server management, SSH has become the de facto standard for connecting to a remote computer when nothing more than terminal access is needed. Paired with the associated SSH file transfer (SFTP) or secure copy (SCP) protocols, SSH provides an incredible amount of flexibility for working with remote computers. Its largest and most obvious drawback is limited graphical support. While not a problem for its traditional use-case, it does limit what high-intensity applications can be used with it.

One recent innovation in remote computing has been the introduction of Remote Development using SSH through the Visual Studio Code IDE. By opening multiple SSH connections to the host machine at once, Visual Studio Code is able to run the IDE's User interface on the client's machine, and simply send all the written code and commands through to the host machine, utilizing its power to compile

code, run applications, and debug software [9]. This enables a development experience comparable to working on a local machine by only transmitting the data that is needed over the internet and constructing the visual user interface on the client. Because of this, high-intensity computational programs such as Artificial Intelligence training and simulation can be run on a powerful host machine and only need to transmit the output to the lower-powered client, but graphical applications are still limited.

3.2 Hardware Solutions

While Hardware solutions are not as popular or widespread as software solutions, there has been some innovation in the field that is worth considering. Hardware solutions usually focus on building a device that has just enough power to connect and stream data to a remote computer in order to leverage its power for the user to use.

3.2.1 Thin Clients

As discussed in Section 2.2, Thin Clients are a family of devices commonly seen in the corporate world to allow employees to access data centers and computing clusters from their desks. Due to their low cost and ease of deployment, they are often the device of choice to enable employees to access the full power of the business' computing infrastructure without needed to purchase a powerful device for each employee. However, since a typical thin client is a headless device, meaning it doesn't come with a monitor, keyboard, mouse, or other peripherals, they are often less portable as one might think. A thin client is usually set up once, and then left in that place for as long as a desktop would stay. Especially with laptop sales on the rise, thin clients do not have the benefit of being more portable than the alternatives.

3.2.2 Nvidia Shield and GameStream

The Nvidia Shield game console was Nvidia's first foray into the realm of remote streaming. Though it was focused on gaming and media streaming, it was one of the first attempts at using a portable device to stream intensive applications such as games from a host computer [2]. At the steep price point of \$349, it definitely was enthusiast hardware for a device that focused on mobile and desktop gaming over running console games or demanding titles locally. But while the physical device was praised for performance, battery life, and the experience of streaming games to the console, it didn't perform too well in terms of sales. It was still widely regarded as a technological breakthrough though, and Nvidia continued to develop the technology into a new device called the Nvidia Shield TV [8]. Focused on bringing the power of a gaming desktop to a TV, the Nvidia Shield TV dropped the idea of portability in favor of filling a different market focused on comfort. While this is no longer a valid device for the purposes of



FIGURE 3.1: An open Nvidia Shield displaying the home page [3].

this paper, the developments of the proprietary GameStream technology that powered the Shield and the Shield TV proved to be even more exciting than the physical devices themselves.

Though the GameStream technology that powers the Nvidia Shield devices is closed source and only officially compatible with the products Nvidia releases, its enticing power and potential drew a community to reverse engineer the protocol. In 2013, a group of students at Case Western Reserve University developed Moonlight, an open source implementation of the GameStream protocol [22]. This open source implementation allows the development of GameStream compatible clients that can run on other devices, from other computers to embedded ARM devices. This stands as a good starting place to develop the solution described in this paper.

3.3 Research Questions

While there are many existing solutions for accessing a computer remotely, many struggle to be performant enough or efficient enough to stream demanding applications to the client without compromising on the user's experience. This thesis will seek to build a solution that enables the use the power of a desktop computer from a mobile device in a way that is performant and responsive in response to the following questions:

1. **Hardware Feasibility:** *What hardware is needed in order to power a mobile device capable of acting as a client?*
2. **Hardware Cost:** *Is it feasible to construct such a device at a lower or comparable cost to existing solutions?*
3. **Communication Protocol:** *Does a protocol exist that is efficient enough to stream demanding applications to a client?*
4. **Software Application:** *Can a software solution be built to utilize this protocol in a manner that performant enough?*

Chapter 4

Developing the Hardware

There are a lot of factors that go into designing and building an electrical system that fits the needs for this project. This process begins by outlining the requirements in Section 4.1, followed by describing the process and methodology of choosing parts in Section 4.2. Then, the feasibility of prototyping such as system is discussed in Section 4.3, and finally the system is designed and manufactured in Sections 4.4 and 4.5 respectively.

4.1 Requirements

This section will explain the different constraints taken into account while developing the hardware portion of this project. First, the hardware must be performant enough to provide a positive user experience (4.1.1), and secondly the product must be cheap enough to justify it over similar alternatives (4.1.2).

4.1.1 Performance

When considering how much power needs to be incorporated in to the miniature computer, it is important to consider a number of factors, such as: having enough power to stream audio and video over the internet, not consuming too much power so as to require an expensive battery, have a fast enough connection to the internet to deliver data with the least possible latency, and having a reasonable footprint to be used as a mobile device.

Immediately, the device must be capable of decoding and rendering video that is streamed over the internet. Such video processing is a relatively expensive process that cannot easily be done with something such as a micro controller or even a single core CPU seen on Single Board Computers (SBC) such as the Raspberry Pi Zero due to the lack of working memory and processing power [12]. Though at the same time, it is important to not simply build a powerful system akin to a laptop that will consume more power. It makes no sense to built a powerful mini-computer when all it will ever be used to is to connect to another computer, and the added power draw would require a larger battery in order to stay operational for decent periods of time. Instead it should be cut down and distilled into something that can do it's job as efficiently as possible.

The second major performance requirement of the hardware is the ability to stream this data over the internet as quickly as possible. Because all of the data being sent to the device is live information from the host machine, the video and audio streams cannot simply be buffered in advance and played when it's ready like a recorded video. Instead, the data must be displayed as it is received so that the user can have a smooth experience without having to wait for their keystrokes or mouse movements to be registered. While Wi-Fi is growing ever quicker and more

reliable, and should absolutely be included as a way to connect to the internet, the best way to ensure a stable and speedy connection is to use an Ethernet Cable. This will help the device work on a consistent connection when used at a desk, while also providing a way to connect while on-the-go.

This leads to the desire for a portable design that can be used both in a conducive environment such as a desk or while mobile. The hardware should have the ability to be powered by a chord plugging into the device, or by an internal battery. This alone presents a number of difficult concerns which will be addressed in Section 4.3. Such a requirement leads to a clear benefit of integrating crucial computers such as a battery and a screen into the hardware design, while leaving as many ports open such as ethernet and USB for the user to take advantage of.

4.1.2 Cost

Because this project consists of both a Hardware and Software component to make up a complete product, there must be reason for both to be developed. While the software has the benefit of being the foundation for the connection between host and client machines, there are already hardware solutions that would theoretically be able to run such software. Since the goal of this project is to save money by not needing to purchase an expensive mobile device to connect to a powerful desktop computer, there must be benefit in the hardware to warrant its existence over repurposing something like an old laptop. The most obvious metric of this is the actual cost of the hardware. If the hardware can be produced at a high enough quality at a low enough cost to bring value over just a software solution, then it should be considered moving forward. Otherwise it may prove that the development of the software should focus on compatibility so the hardware can be provided by the end user.

Generally speaking, the goal for the hardware of this project is to cost approximately \$100-\$200 to produce a single unit. This comes with the understanding that hardware production in bulk often costs significantly less than assembling a single product. The reasoning for this cost comes down to feasibility: the price of the final product should be less than what it would cost to purchase or refurbish an existing computer to do the job of this project. Otherwise, as stated before, the software of this project may prove more valuable than the hardware as long as compatibility is kept in the forefront of development.

4.2 Choosing Parts

When starting to choose parts for the hardware, there are many difficult questions to face. First and foremost, the hardware had to be built with the time and resources available for this project. Specifically, building a Single Board Computer from scratch would be completely infeasible, but the goal was also not to simply purchase an off the shelf SBC that was not tailored to the task at hand such as a traditional Raspberry Pi. Instead, a better suited solution would be to utilize a *Raspberry Pi Compute Module 4 (CM4)* which has “The power of Raspberry Pi 4 in a compact form factor for deeply embedded applications” [25]. Boasting an incredibly small size and streamlined I/O, the only way to communicate with the Compute Module was through two tiny mezzanine connectors as shown in Figure 4.2. With no other pin outs for anything such as power, HDMI, or USB, everything had to be built on a custom Printed Circuit Board (PCB).

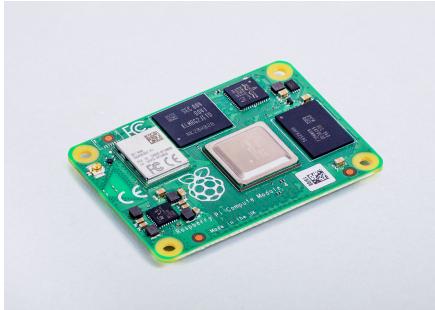


FIGURE 4.1: Raspberry Pi Compute Module 4, sizing
55 x 40mm

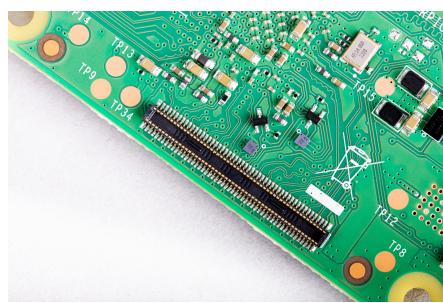


FIGURE 4.2: Close-up of the pin out, with a 0.04mm pitch between pins

This PCB would have to be designed so that the CM4 could be plugged into the board using the mezzanine connectors and rely on it for all inputs and outputs. This, however, is only the beginning of what is required to develop the product. Since the project would not use an off the shelf SBC like a regular Raspberry PI, the project would require only designing the PCB but also picking out all the components that would go onto the PCB, such as the USB ports themselves, the USB driver chip, and related components needed for operation. Because the CM4 is designed to be as low-powered and compact as possible, some features that would usually be expected of such a device, like the USB ports, are disabled by default and must instead be built by the board designer. While this helps greatly by removing extraneous components and features that aren't needed for this project, it also provides a great opportunity to learn exactly what parts are being used in the hardware.

Thought must also be given to the peripherals that will be included in the design. Most importantly, if a screen is going to be included in the design, what kind of screen will be used? For projects of this scale, the usual candidates for a relatively performant screen are HDMI screens or DSI screens. While HDMI screens are much more common and recognizable, DSI screens are a new development that promises 60Hz performance in low cost and low power applications such as embedded systems. However, when looking into existing DSI screens, there are a couple of issues that prevent it from being the best choice for this project. Even though DSI screens are more power efficient and have built in touch support, it is a closed format which greatly restricts manufacturers from producing their own screens [14]. Due to this, there is only one screen available for purchase outside of the official implementation, and both have a lower resolution than what is possible with HDMI screens of the same size. So even though HDMI screens are more expensive at this size, they are a more reliable choice compared to the still-developing DSI standard.

Another potential feature to consider is whether to integrate a Real Time Clock (RTC) into the design. An RTC is a component commonly found within laptop and desktop computers that is used to keep track of time even though the computer is powered off. This is done with a small button battery powering an RTC chip integrated into the motherboard to keep the system's clock in sync with the real world. While this is almost never necessary in embedded systems, it is worth considering for this project because of its use as a mobile computer. However, when considering the primary use case of connecting to another computer over the internet, and the potential of needing a linux kernel patch in order to add support for the RTC [17], it would be much simpler to tell the operating system to simply resync its clock with an internet clock every time it connects to a network.

An invaluable resource while designing the PCB was the Official I/O board for the CM4 [26]. Though the PCB board would still have to be custom built and components would need to be replaced due to the current ongoing chip shortage, it provides a great foundation instead of starting from scratch. Another massive benefit from working within the Raspberry Pi Ecosystem is their incredible documentation and community surrounding the boards [27]. Around the time this project was being started, the CM4 was just beginning to get into the hands of makers and developers, and only a couple custom boards had actually been made by anyone other than Raspberry Pi themselves. Even though this leaves the community with less of a bulk of knowledge to work with, it has provided a sort of comradery with everyone helping each other learn the new Compute Module together. Members of the CM4 engineering team would even assist those on the forums to help them pick parts and design their own boards [1, 17].

Following the example put forward by the existing boards, PCB parts were chosen to be integrated into the circuit as safely as possible. Thankfully due to most parts following international standards, many parts were chosen simply because they followed the same standard as the CM4 (such as USB 2.0, or 1PORT 1000 BASE-T Ethernet) and were in stock. What follows is a table of some components that were used in the custom PCB that may differ from those used by the CM4's official board, and the reason for their selection. A full Bill of Materials (BOM) can be found in Appendix B.

Purpose	Manufacturer's Number	Reason
USB-C Power in	USB4125-GF-A by "GCT"	Most projects of this scale are powered by micro usb, but with USB-C gaining popularity, this component was picked for its low price and exclusive ability to transfer power without transferring data.
Ethernet Port	ARJM11D7-502-AB-EW2 by "Abracon LLC"	One of the few in-stock ethernet ports that supports 1000 BASE-T usage. The CM4 recommends the <i>MagJack-A70-112-331N126</i> produced by "LINK-PP", but that part is out of stock for at least another year as of writing.
USB Port	SS-52100-001 by "Stewart Connector"	Single flat port that supports USB 2.0.
HDMI Port	SS-53000-001 by "Stewart Connector"	Single flat port that supports HDMI 1.4.
Flat Flexible Cable Connector	FFC3B07-20-T by "GCT"	A generic Flat Flexible Cable port that can be connected to anything. In this case, one is connected to the built-in screen's HDMI port, and another is connected to the screen's touch/power USB port.

2-Pin Through Hole Header	<i>PREC002SAAN-RC</i> by "Sullins Connector Solutions"	A generic 2-pin header than can be used to bypass the power circuit as will be discussed in Section 4.3.
HDMI Power Regulator	<i>RT9742ENGJ5</i> by "Richtek USA Inc."	The CM4 recommends the <i>RT9742SNGV</i> from the same manufacturer, but it is out of stock. This part has slight differences in functionality that require adjustments from the official I/O board design.
SD-Card Power Regulator	<i>RT9742VGJ5</i> by "Richtek USA Inc."	The CM4 recommends the <i>RT9742GGJ5F</i> from the same manufacturer, but it is out of stock. This part has slight differences in functionality that require adjustments from the official I/O board design.
Battery Voltage Regulator and Boost Converter	<i>TPS61090RSAR</i> by "Texas Instruments"	Used to balance power usage from the USB-C port or the attached battery, favoring the external power when available.
Li-Po Battery Charge Management Controller	<i>MCP73871-2CCI_ML</i> by "Microchip Technology"	Used to charge the attached battery when attached to external power through the USB-C port.
4 Port USB Hub Controller	<i>USB2504A-JT</i> by "Microchip Technology"	Used to control the four USB 2.0 ports and send the data to the CM4. The CM4 recommends the <i>USB2514B-I/M2</i> by the same manufacturer, however it and all the parts in it's family are out of stock with at least a 52-week back order. Out of all the components, this has the highest potential to be problematic.

TABLE 4.1: A non-exhaustive list of components that were chosen for this project.

4.3 Prototyping

Whenever working with electrical projects such as this, it is always a good idea to build up prototypes before committing to a final design that will be produced at the highest quality. Usually breadboards and some breakout boards for Surface Mounted Devices (SMD) are a great way to develop a circuit without creating a full PCB; but due to the nature of the complex microchips required for this project as well as the dependence on the CM4's tiny mezzanine connectors, building a prototype using a breadboard is impractical given the time and resources available.

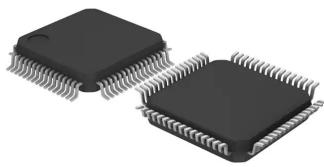


FIGURE 4.3: The USB controller with 0.5mm between pins

For example, the USB controller chip only has 0.5mm of space between each of its 64 pins (Figure 4.3), which would require a breakout board at least 7 times larger than the chip so that each pin can be easily accessed by hand. Not to mention the difficulty in wiring up that many pins along with the other related components which would need their own breakout boards either purchased or custom-designed.

Instead, the first board was built out in a piecewise fashion, allowing each section of the board a way to fail without interrupting the rest of the board. This allows each function of the board to be

tested individually and in isolation so that changes could be made incrementally without needing to worry about their integration with the rest of the board. For example, in order to build a system that would allow the board to be powered via USB or an internal battery, as well as charge the battery simultaneously, a complex circuit would have to be built that managed Lithium Polymer (LiPo) batteries which could cause fires if mishandled. Since power is obviously a critical component of the board, A secondary way of feeding power directly to the board was built to bypass all that circuitry in case it fails. In doing so, the board can still be powered and its other features can be tested while a fix for that portion of the circuitry is developed for the next iteration.



FIGURE 4.4: A breakout board similar to what would be needed for the USB controller used on the custom PCB

4.4 Designing the Printed Circuit Board

Because of the lack of prototyping options discussed in Section 4.3, and ultimate decision to build the board in a way that could safely fail in any way, it became clear that multiple versions of the project were going to need to be designed and produced. Since the time and budget for this project was limited, the amount of revisioning was minimized by running the circuit by electrical engineering students and professors to catch any potential flaws before the board was manufactured.

4.4.1 Power

Development started with arguably the most important part of the board: the power supply (schematic in Figure 4.6). If the CM4 could not receive power, none of the project would work. A feature of modern devices that has become ubiquitous as much as it has been taken for granted is the ability to charge a battery at the same

This is that tricky sentence

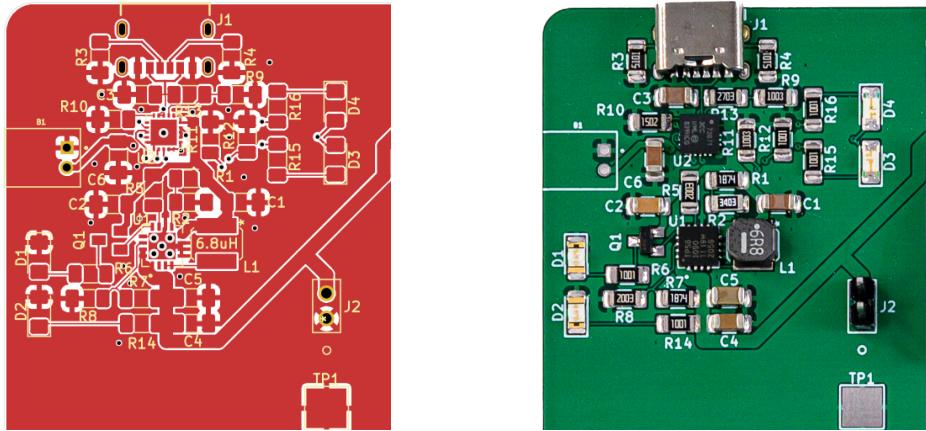


FIGURE 4.5: The front side of the PCB's power circuit

time it is in use by the device. This enables the device to be used portably until the battery runs out and then charge the device without needing to stop using it before moving again. Though it proves to be slightly more complicated than simply connecting the two power supplies. As seen in Figure 4.5, the source of external power, the USB-C port labeled *J1*, has to travel through two integrated circuits before reaching the CM4 through the trace on the right side. The first chip, *U2*, detailed in Table 4.1 as *MCP73871-2CCI_ML*, is used to charge the internal battery attached to the left side of the board through the connector labeled *B1*. The second chip, *U1*, also detailed in Table 4.1 as *TPS61090RSAR*, is used to balance the power draw from the USB-C port or the attached battery, favoring the external power when the device is plugged in. Thankfully, due to the sensitivity of the two chips, their provided datasheets offer specific configurations that they should be designed in to ensure proper functionality. Although very dense, these blueprints promised individual functionality for each chip, all that was left was connecting them together. The two of these chips must work in tandem for power to be successfully delivered to the

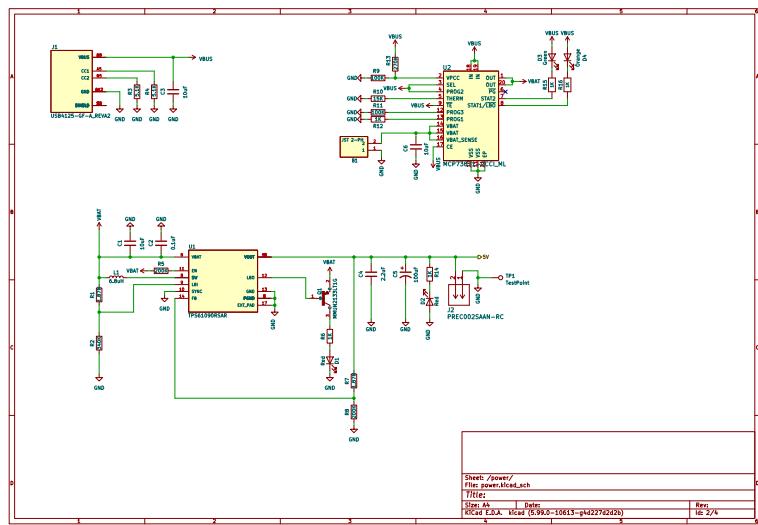


FIGURE 4.6: Schematic layout for the power circuit, including battery management. Expanded in Appendix A.2

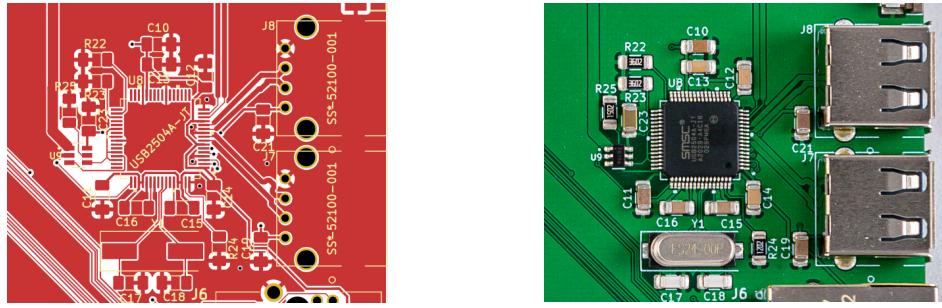


FIGURE 4.7: The front side of the PCB's USB circuit

CM4, if either failed, none of the other parts of the board could be tested. In the event this happened, a secondary method of powering the board directly was added in the form of the generic two-pin header *J2*. These two pins are simply 5v input and ground through-hole pins that bypass all the power circuitry just described. Once the CM4 could be powered, the rest of the I/O could be designed.

4.4.2 USB

The next prominent circuit on the board is the section that controls the USB ports. As seen in Figure 4.7 (schematic in Figure 4.8), the dense circuit is mostly comprised of the USB controller chip, *U8*, its associated components, and the four USB ports, two of which are pictured as *J7* and *J8*. Similarly to the power circuit, the USB controller chip is a very sensitive component that must be arranged in relation to other components in a certain way in order to ensure it will function properly. This includes surrounding the chip with numerous capacitors to normalize any small power fluctuations, using a specific load balancing power distribution switch (*U9*), and an oscillating crystal to provide a stable clock for the chip. The chip connects to each of the four downstream USB ports and handles all of the digital signal processing before sending the data to the upstream CM4.

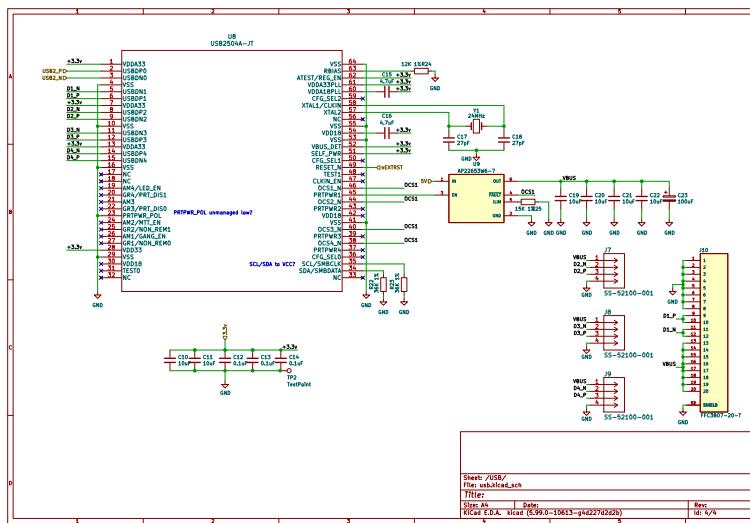


FIGURE 4.8: Schematic layout for the USB hub and ports. Expanded in Appendix A.4

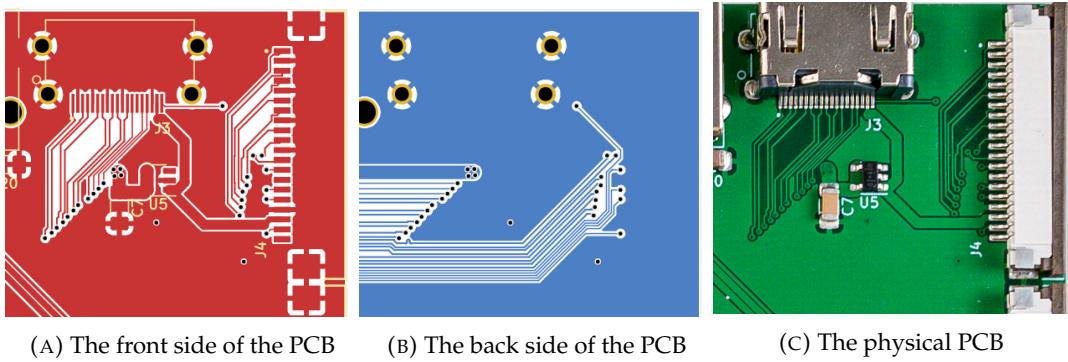


FIGURE 4.9: The PCB's HDMI circuit

4.4.3 HDMI

Another vital part of the board is the two HDMI ports provided by the device. Thankfully, unlike the USB ports, HDMI ports send information in only one direction, so an extra chip is not necessary to handle the data as an intermediary. All that is necessary is to connect the two HDMI ports to the CM4 in the right order and the device will be able to output the rendered video to a display. Because this device is intended to be a standalone machine, it makes sense to have essential components like a screen integrated as part of the design. The only issue with simply attaching an HDMI cable inside of the device is that the ports are relatively bulky and inflexible, meaning that another solution must be created to fit in the tight space. This is accomplished with the Flat Flexible Cable Connector, seen on the right side of Figure 4.9a as J4. Flat Flexible Cables (FFC) are generic bendable cables that can be used in tight spaces. Even though their connectors are completely generic, meaning that they have no set pinout, it is possible to use them given that the receiving screen will take in the FFC in the same order as a HDMI cable. Using this knowledge, an internal FFC can be wired for the device's primary screen, while a second traditional HDMI port can be exposed to the user in case they want to use an external monitor. This, along with the FFC used for one of the four USB ports, allows a screen to be hooked up to the CM4 internally without needing to expose anything to the user.

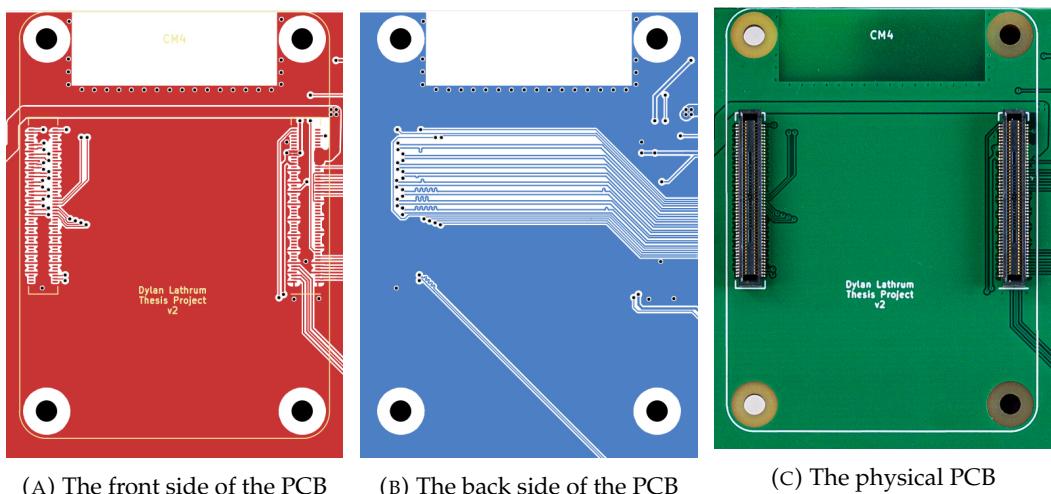
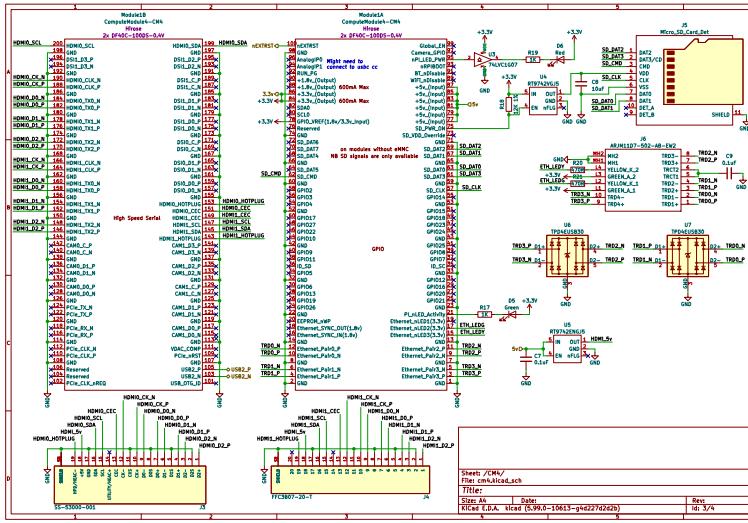


FIGURE 4.10: The circuitry surrounding the CM4's two mezzanine connectors



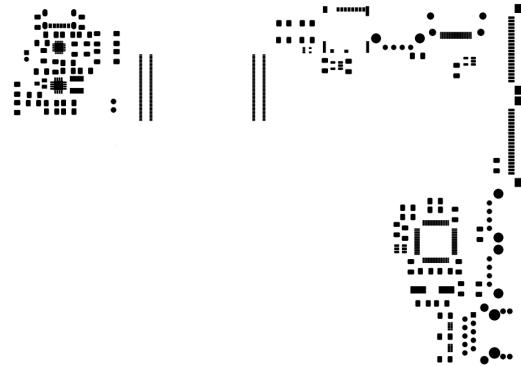


FIGURE 4.12: The PCB's stencil. Areas colored black are where solder paste is applied to solder SMD parts to the board

by recommendation of a professor, the printing was done through an external company that provides quick turnaround and low costs for printing PCBs [24]. A set of five boards was ordered, allowing for multiple boards to be assembled in case of error, as well as a stencil. A PCB stencil is a thin sheet of metal with holes cut in it that line up with the pads on the PCB board. This enables the easy application of solder paste onto all of the pads on the board as once, including the difficulty small pads of the CM4's mezzanine connectors. While the PCB was being printed, all of the parts found in the Bill of Materials (found in Appendix B) were ordered from sites such as Digi-Key and Mouser [10, 23]. In order to account for the production of multiple prototype boards and the possibility of broken parts, enough parts were ordered to produce five complete boards.

4.5.1 First Attempt

Once the PCB and its component parts arrived, the board could be assembled. The first board was assembled by hand using the PCB stencil and a handheld heat gun. The stencil, shown as an image in Figure 4.12, is used by securing it to the PCB board so that the holes line up with the pads, and spreading solder paste across it allowing the paste to cover the exposed pads through the holes. The stencil was then removed, and each of the surface mounted components were then placed onto the board using a set of tweezers. For tricky parts such as the CM4's mezzanine connectors and other small parts with the pads below the component itself, a microscope was used to assist in lining up the pads with the components. Once the parts were placed, a heat gun was used to melt the solder paste and fuse the components to the board. Following the SMDs, the through-hole components, such as the ports lining the side of the board, were then placed and soldered using a hand-held soldering iron.

The first board ran into two issues that prevented it from working. Firstly, due to improper use of the heat gun, most of the LEDs had popped, resulting in all hardware-level diagnostic indicators being inoperable. Secondly, the CM4 connectors were not properly connected to the board, and in an attempt to repair it, the connector components were irreparably damaged and the board itself was damaged slightly, seen in Figure 4.13. Instead of attempting to assemble another board by hand, a local third-party company was contacted to assemble the board professionally. After delivering the board, stencil, and parts, the second board was assembled.

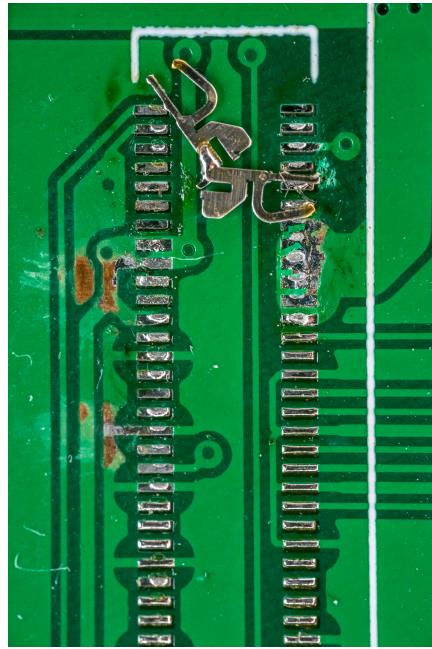


FIGURE 4.13: The leftovers of the broken-off mezzanine connector from the hand-soldered first board. Notice the exposed copper and warped pads.

4.5.2 Second Attempt

The second board was assembled mostly successfully, and allowed the first stage of testing to begin. The only issue apart from some ports being too close together with real-world cables was one component having the wrong footprint on the board. Unfortunately, the component *U2 (MCP73871-2CCI_ML)*, is described as using the *QFN-20-1EP_4x4mm* footprint in its datasheet, but actually uses a revised version of the footprint, shown in Figure 4.14. This discrepancy meant that that component could not be soldered, and its associated circuit would not function properly. In this case, the component was a part of the battery charging circuit, causing the entire primary power supply to be unusable. Thankfully, as described in Section 4.3, the secondary method of powering the board could be used to test the remaining features of the board. After making the necessary fixes, a second round of five PCB boards and a stencil was ordered. Because there was still enough components to assemble three more boards, and no components needed to be changed, no more

Does this need to be cited?
Can you cite a datasheet
like this?



(A) *QFN-20-1EP_4x4mm*



(b) *QFN-20-1EP_4x4mm_RevA*

FIGURE 4.14: The two footprints for components with the *QFN-20-1EP_4x4mm* package. The original footprint is shown in Figure 4.14a, while the revised footprint is shown in Figure 4.14b

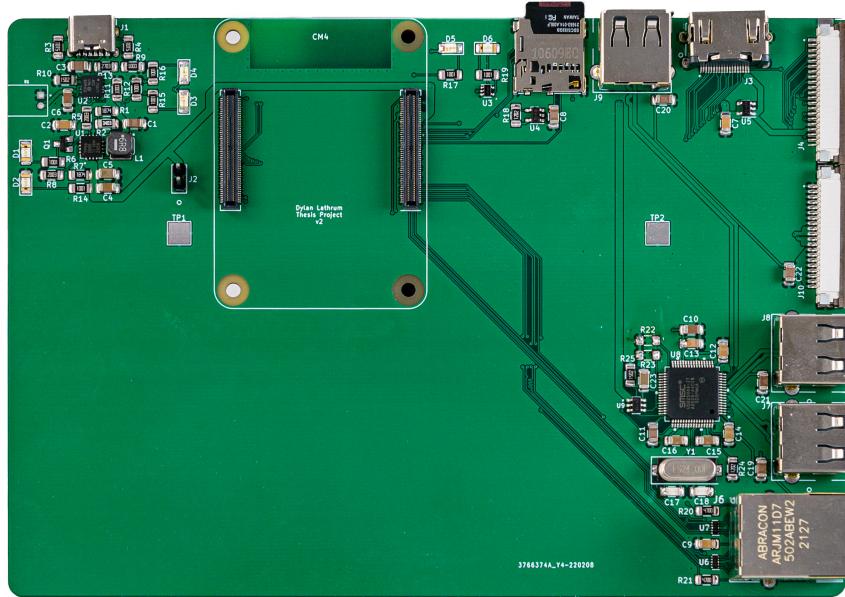


FIGURE 4.15: The assembled PCB

components were ordered at this stage. Once the boards were delivered, the revised board was again sent to be professionally assembled.

4.5.3 Third Attempt

Once the third board was assembled, the board could be tested in full. While each part of the board was at least partially functional, there were still issues preventing it from working completely. In particular, USB devices plugged into any of the ports on the board were able to receive power from the board but were unable to transfer data in either direction.

I am currently going through and diagnosing this issue

Chapter 5

Developing the Software

The second half of the project is composed of software that is used to connect the client machine to a host computer. Designing the software begins with identifying its requirements in Section 5.1. Then, potential avenues for development are discussed in Section 5.2 so that the software does not need to be built from scratch. Following that, special considerations must be made while developing the software for the CM4 running on an *ARM* processor in Section 5.3, and the development of the suite of tools used to record data is detailed in Section 5.4.

5.1 Requirements

Much like the hardware requirements in section 4.1, the primary requirement of the software is to be performant enough to provide the user with a positive experience. This is broken down to include being able to stream and decode video over the internet with as little latency as possible, stream inputs from the user to the remote computer, and retain visual fidelity while the host machine is under heavy load. While much of the latency between the host and client is due to the network both devices are connected to, there is still some optimizations that can be made to reduce the time between data being send and images being displayed on the screen. Similarly, while the rate at which data is transferred between the host and client cannot exceed the speed of the slowest internet connection of the two devices, there can still be a balance between the bandwidth-saving but lower quality streaming seen in Chrome Remote Desktop (3.1.3), and the local network but higher quality Nvidia Shield (3.2.2). To accomplish this, a compression method similar to CRD will be used, but it will not be bound by the traditional web browser which limit CRD due to the software solution being developed as a native application.

As a secondary objective, it would also be beneficial for the software to be able to run on a variety of different hardware platforms in the event the hardware is determined to be infeasible to develop, as discussed in Section 4.1.2.

5.2 Potential Avenues

As with any software project, there are many potential ways to solve the problem. In many cases, the easiest place to start is to begin by ruling out potential avenues that are not feasible. For example, as evident by the number of existing solutions that attempt to solve this problem (Chapter 3), taking into special account Nvidia's recent foray into remote gaming with the Shield (3.2.2), it is not reasonable to attempt to create a brand new software solution from scratch with the time and resources available. It has taken Nvidia, a company worth hundreds of billions of dollars, years to

get to the point they are at now [2]. Developing a rival protocol or new software platform is simply not possible for this project. Instead, given that Nvidia's GameStream technology is the current leader in the realm of high-performance remote computing, the best way to build a software product that meets the requirements laid out in Section 3.3 is to build upon what already exists.

5.2.1 NVIDIA GameStream

Should this subsection and the following subsection be top level sections? The organization of the headers is a little strange, but the content flows pretty well

Unfortunately, the Nvidia GameStream protocol is both proprietary and a closed source project. Very little is officially known about the protocol, other than it uses specific video encoders built into Nvidia Graphics cards from the 600 series from 2012 and newer [15]. Apart from that, everything has been researched and reverse engineered by the community. A GameStream session is started by first connecting to a host computer which has enabled the GameStream service through Nvidia's GeForce Experience application. Once the service is enabled, the computer will listen on port 47989 for the following web requests:

- Should this be italicized?
quoted? plaintext?
- /serverinfo
 - /pair
 - /unpair
 - /applist
 - /launch
 - /resume
 - /cancel

Once a session has been started with a */launch* request, raw data covering video, audio, and input is passed back and forth over a number of ports until the session is terminated. An end-to-end example of the protocol's usage is as follows:

1. A user knows that a capable host computer is ready to stream at the address 192.168.1.123.
 - (a) This can be confirmed by sending a HTTP GET request */serverinfo* to the host computer on port 47989. For example:
http://192.168.1.123:47989/serverinfo?uniqueid=1234
2. The user can make a request to */pair* with a number of parameters to authorize itself with the host.
 - (a) Pairing is a two step-process. First, a request to pair is made, at which point the host machine will display a four digit code on its screen. Then the client must send that code back to the host as a second form of authentication.
 - (b) The authentication created by this pairing process lasts until either the client makes an */unpair* request with its unique ID or a user on the host machine unpairs the client.

3. The user can make a request to */applist* to get a list of applications that the host is able to stream.
4. The user makes a request to */launch* with a number of parameters to instruct the host to create a session with the given settings and launch the specified application.
 - (a) The configuration options given include the application to launch, the video resolution to stream, audio setup, input mapping, and multiple forms of authentication.
5. If the user unexpectedly or accidentally leaves a session without closing it, they can make a request to */resume* to instruct the host to reopen and resume the session the previous session.
6. Once the user is finished, they make a request to */cancel* to instruct the host to close the session.
7. If the user is completely finished with the host, they can return it to the state they found it in by making a request to */unpair* with their unique ID, unauthorizing themselves with the host.

The most difficult part of this process is handling the transfer of data after a session has begun. Once a session has launched, data is sent directly between the host and client over a number of ports without going through an easy to understand web protocol. This is where the community project Moonlight steps in.

5.2.2 Moonlight

Moonlight is an unofficial third-party open-source client for the Nvidia GameStream protocol [22]. The project is structured as a core library written in C with a number of community built clients built on top of it for various platforms. In essence, it turns the process described in the previous section (5.2.1) into a usable programming interface. By reverse engineering the protocol originally implemented by the Nvidia Shield, Moonlight enables the development of a GameStream client for almost any operating system. The Moonlight community has already developed fantastic clients for many platforms such as Windows, Mac, Linux, Android, iOS, Web, and even other gaming consoles. All that is needed for the purposes of this project is to make a fork of the project that tweaks it to best suit the CM4 that drives the hardware and develop a suite of tools to test its functionality.

5.3 Developing for ARM

The CM4 runs on a 64-bit ARM Cortex-A72 processor [25], meaning that applications must be compiled for the ARM architecture in order for the CPU to be able to run it. If a developer were to write a program on their *x86* architecture Windows desktop computer, they could compile their program for any other computer running on the same *x86* architecture, but such a program would be incompatible with an ARM processor. Similarly, if the program were compiled for ARM, that program could be run on an ARM computer but not one running *x86*.

In order to run the code for the project, given that the CM4 is the only ARM computer available for this project, the code must either be compiled on the CM4 or built

in a cross-compilation environment on an *x86* machine. Using a cross-compilation environment would be overall quicker than that developing on the CM4 itself since it can use the full power of a desktop PC. Thankfully, setting up a cross-compilation environment for projects written in C is straightforward. *CMake*, a common compilation tool for C, has built-in support for compiling for different target architectures with easy to use libraries. What isn't as straightforward however, is building a toolchain to compile the rest of the code for *ARM*.

In order to create an out-of-the-box experience for someone using the hardware for this project, a custom operating system image is created that has all the dependencies, libraries, and applications pre-installed and pre-configured. Manjaro provides good tools for building and managing custom images of the Manjaro operating system for *ARM* architectures [20], with the only drawback being that it is that it must be run on an *ARM* processor, and is much more difficult to run on another architecture. To make development easier, a toolchain was built that could be run on the CM4 to build the code that is written on other computers. All the developer needs to do is to pull the git repository of the project, run a set up script to install dependencies, and then run the build script to compile everything into an immediately deployable OS image. This build script will automatically compile all the necessary code using *cmake*, package all the necessary files using *makepkg*, and build a fresh OS image that is ready to flash. The following excerpt details the three functions used to make this work.

Excerpt from */scripts/build.sh*

```

50 runMake()
51 {
52     mkdir ./moonlight/build # Create build directory
53     cd ./moonlight/build # Go to build directory
54     cmake ../ # Run cmake
55     make # Build the project
56     sudo make install # Install the project
57     sudo ldconfig /usr/local/lib # Link libraries
58     cd "$parent_path" # Go back to original execution location
59 }
60
61 runPackage()
62 {
63     cd ./pkgbuild # Go to pkgbuild directory
64     tar -cf moonlight.tar ../moonlight/ # Create tar archive of the
       project
65     makepkg -sLfc # Build the package
66     rm moonlight.tar # Remove the tar archive
67     cd "$parent_path" # Go back to original execution location
68 }
69
70 runBuild()
71 {
72     pkgname=$(sed -n 's/^pkgname=//p' ./pkgbuild/PKGBUILD)
73     pkgver=$(sed -n 's/^pkgver=//p' ./pkgbuild/PKGBUILD)
74     pkgrel=$(sed -n 's/^pkgrel=//p' ./pkgbuild/PKGBUILD)
75     # Copy local package to cache to be installed into new image
76     sudo cp ./pkgbuild/$pkgname-$pkgver-$pkgrel-aarch64.pkg.tar.zst /var/
       cache/pacman/pkg/
77     # Build image
78     sudo buildarmimg -d rpi4 -e thesis -i $pkgname-$pkgver-$pkgrel -
       aarch64.pkg.tar.zst
79     read -s -n 1 -p "Image built, press a key to continue (for sudo)"
80     # Remove local package from cache

```

```

81  sudo rm /var/cache/pacman/pkg/$pkgname-$pkgver-$pkgrel-aarch64.pkg.
82    tar.zst -f
83  # Move build image to build folder
84  sudo mv /var/cache/manjaro-arm-tools/img/* ./build/
85  cd "$parent_path" # Go back to original execution location
}

```

This script starts by building the forked version of moonlight using *cmake* and *make* and links its libraries using *ldconfig*. Once the application is built, it then builds it into an installable package using *makepkg*, before installing it into the newly created image. Finally, the image is moved to the project's build folder where it is ready to be flashed and installed like any other operating system.

5.4 Developing Testing and Measurement Tools

In order to evaluate the performance of the project, a suite of tools must be developed to test both the newly developed application and the existing solutions. To do this, two sets of data will be collected: First, data is recorded directly from the forked version of moonlight that logs statistics such as the number of frames received, dropped, or rendered, and data that is recorded from a separate application that logs the amount of time it takes for the client machine to receive updates after it sends data to the host machine. The first set of data is important to track the raw performance of the application, with statistics like Frames Per Second (FPS) and decoding time which are difficult to record otherwise. These statistics provide insight into how performant the application is from a technical perspective. The following is a structure that holds the statistics recorded over a session of connecting to a host computer. The data recorded is discussed in Chapter 6.

Excerpt from *moonlight/src/video/stats.h*

```

23  typedef struct _VIDEO_STATS {
24    uint32_t receivedFrames;
25    uint32_t decodedFrames;
26    uint32_t renderedFrames;
27    uint32_t totalFrames;
28    uint32_t networkDroppedFrames;
29    uint32_t totalDecodeTime;
30    uint32_t totalRenderTime;
31    uint32_t lastRtt; // Rtt = Round Trip Time
32    uint32_t lastRttVariance;
33    float receivedFps;
34    float decodedFps;
35    float renderedFps;
36    uint32_t measurementStartTimeStamp;
37 } VIDEO_STATS;

```

The second set of data is recorded by measuring the total amount of time it takes for the host to receive data from the client, and for the client to receive updates back from the host. This is done using the following steps once the client is connected to the host:

1. When a user hits a key on the client, record which key was pressed and at what timestamp.
2. When the host receives the instruction to press the key from the client, record which key and the timestamp.

3. Change the color of the host's screen to a different color.
4. When the client detects that the color of the screen has changed, record the current timestamp.

This records two points of data, the time it takes for the host to implement the actions done by the user, and the time that the user sees the result of their actions. Because this method of data collection runs as a separate application on both the client and the server, it can be used to measure the performance of any of the remote desktop applications for comparison. This is again discussed in Chapter 6.

Both the application running on the client and the server do the same thing, every time a key is pressed, record which key was pressed and at what time, and every time the pixel in the center of the screen changes color, record the new color and at what time it changed. This data is written to a Comma Separated Values (CSV) file for later processing. To change the color of the screen, a simple web page is used that changes color any time a key is pressed. Its source in its entirety follows:

/datalogger/web/index.html

```

1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <script>
5       let nextColor = "green";
6       document.addEventListener("keydown", () => {
7         document.body.setAttribute("style", "background-color:" +
8           nextColor);
9         nextColor = nextColor === "green" ? "red" : "green";
10      });
11    </script>
12  </head>
13  <body></body>
</html>
```

This is sufficient to change the color of the screen back and forth between easily distinguishable colors for both the client and the host to record. With both of these datasets recorded, the next step is to analyze the data and evaluate the performance of the application.

Chapter 6

Evaluation

6.1 Testing Methodology

In order to ensure that the data collected for analysis is accurate and directly comparable, the testing environment must be kept exactly the same between trials and between different applications. This gives each application the exact same starting conditions, and the best chance to demonstrate it's unabated performance. For these tests, the host machine is a desktop computer running Windows 10 with a AMD Ryzen 9 5900X CPU and a Nvidia RTX 3070 GPU running the latest GeForce Experience version 3.25.0.84. This configuration does not change throughout testing. The client machine is the CM4 board attached to the PCB developed in Chapter 4, running the custom Manjaro linux distribution and forked version of Moonlight developed in Chapter 5. This configuration also does not change throughout testing. Each test is performed with both the host computer and the client machine running on the same network, with both devices connected to the same router using an ethernet cable. This ensures that the speed of the network is not a factor in the tests. The tests are performed multiple times under the following conditions:

- Ideal Conditions: No other applications are running during testing.
- CPU Stressed: The CPU is constantly at 100% load during the entire test.
- GPU Stressed: The GPU is constantly at 100% load during the entire test.

For each test, the following steps are performed:

1. Both the host and client computers are fully rebooted to ensure no other application is running.
2. The host computer starts up a single browser window with the web page described in Section 5.4.
3. If the test calls for the CPU or the GPU to be stressed, the application "Blender" is started on the host machine, and a benchmark is performed for the CPU or the GPU. This benchmark will run for longer than the duration of the test, and will keep the CPU or GPU at 100% load for the entire duration of the benchmark.
4. Both devices start up their respective data collection tool that will record key-strokes and changes in the screen's color.
5. The client machine connects to the host computer using the application being tested.

6. The user presses the “0” key on the client’s numpad to synchronize the two devices’ data recording tools.
7. The user begins by typing on the client’s keyboard slowly, allowing both computers to record their data.
8. The user then types on the client’s keyboard quickly to ensure that quick successive inputs are handled correctly and no data is lost or duplicated.
9. Once typing is complete, both the client and the host computers close their data recording tools and the client ends the streaming session.

This process is repeated for each of the following applications:

- The forked version of Moonlight developed for this project.
- Chrome Remote Desktop (CRD, Section 3.1.3).
- Microsoft’s Remote Desktop Connection (RDP, Section 3.1.1).
- Virtual Network Computing (VNC, Section 3.1.2).

Once all the data is collected, the data from the host and client machines for a single trial are organized into a single CSV file for processing. In order to ensure that the data is comparable, all extraneous factors must be removed from the data. Namely, the network latency and difference in computer clocks must be removed. This is partially accounted for in step 6 of the above process, and the remaining inaccuracy can now be removed by subtracting the time difference between the when the client presses the button and the time the host executes that command. This ensures that any inaccuracies in each computer’s internal clock will not impact the data.

After the data is cleaned up, the data can be processed by matching each key-stroke and change in the screen’s color detected by the client and host machines and calculating how much time elapsed between the events. This reveals the amount of delay the user experiences between when they press a key and when the host executes the command, and between when they press a key and can visually see the response.

6.2 Responsivity and Latency

TODO

6.3 Performance

TODO

6.4 Quality

TODO

6.5 Summary

TODO

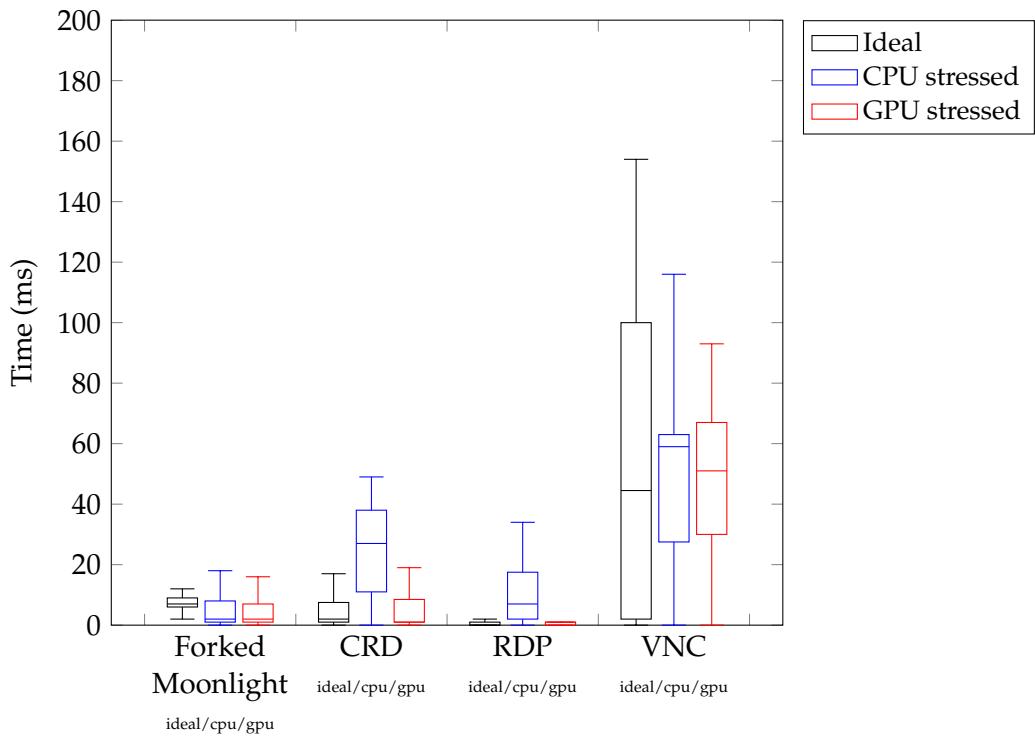


FIGURE 6.1: Time for host to press key from client.

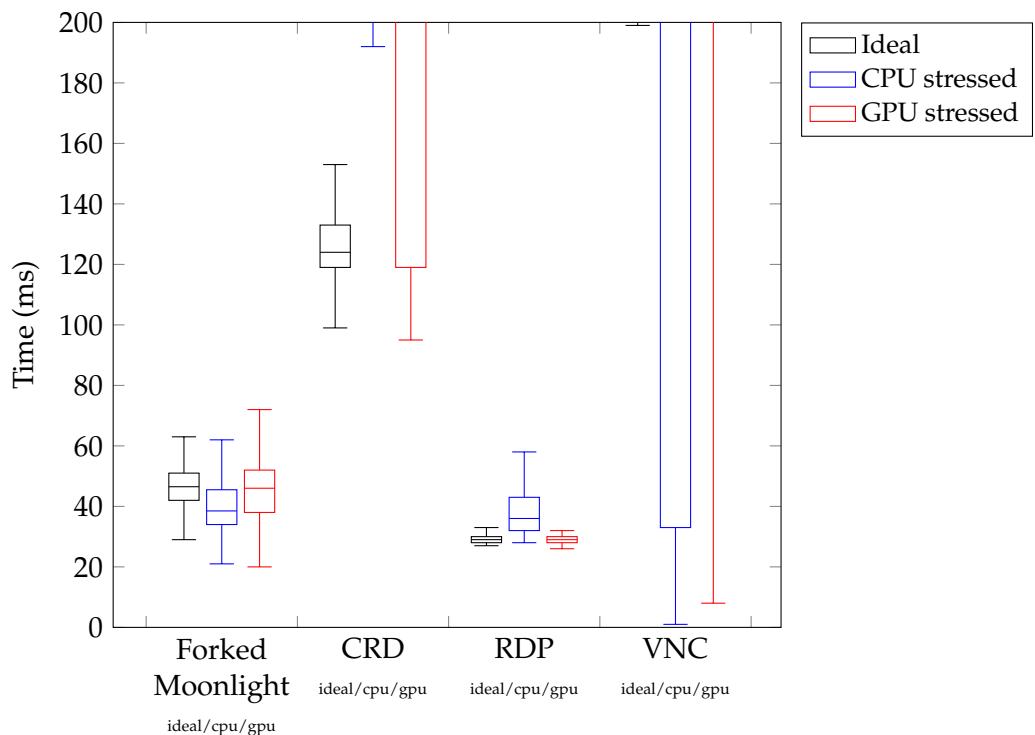


FIGURE 6.2: Time for client to receive update to screen.

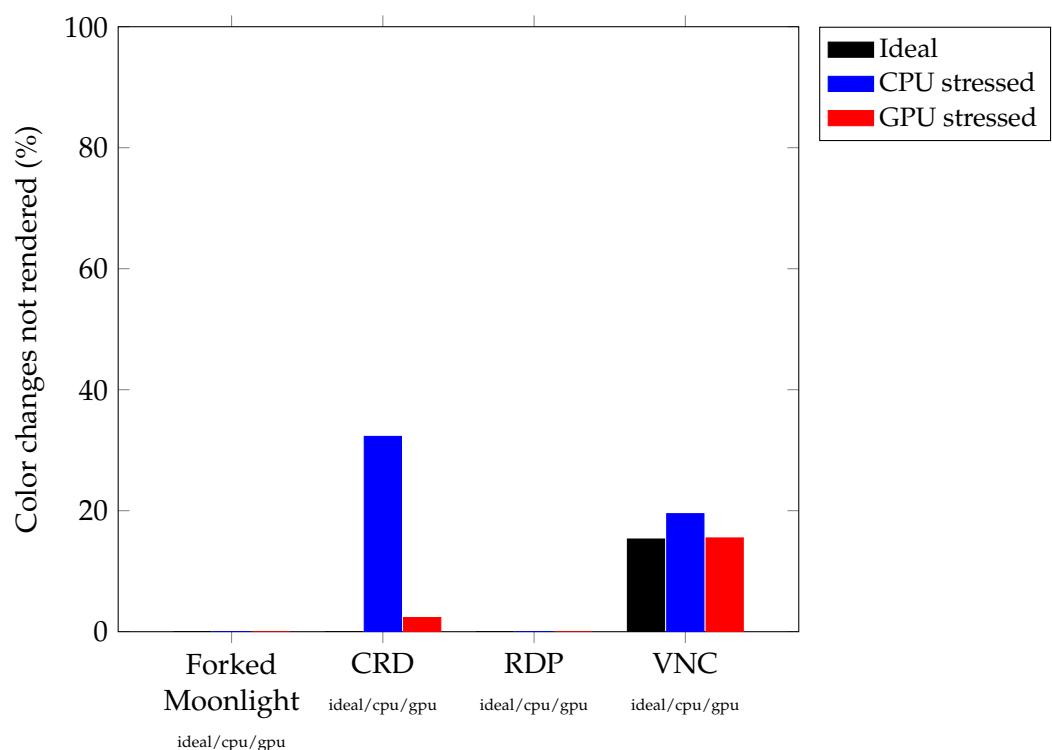


FIGURE 6.3: Percentage of color changes not rendered.

Chapter 7

Conclusion

7.1 Summary

TODO

7.2 Limitations

TODO

7.3 Future Research

TODO

Appendix A

PCB Specifications

A.1 PCB Schematic

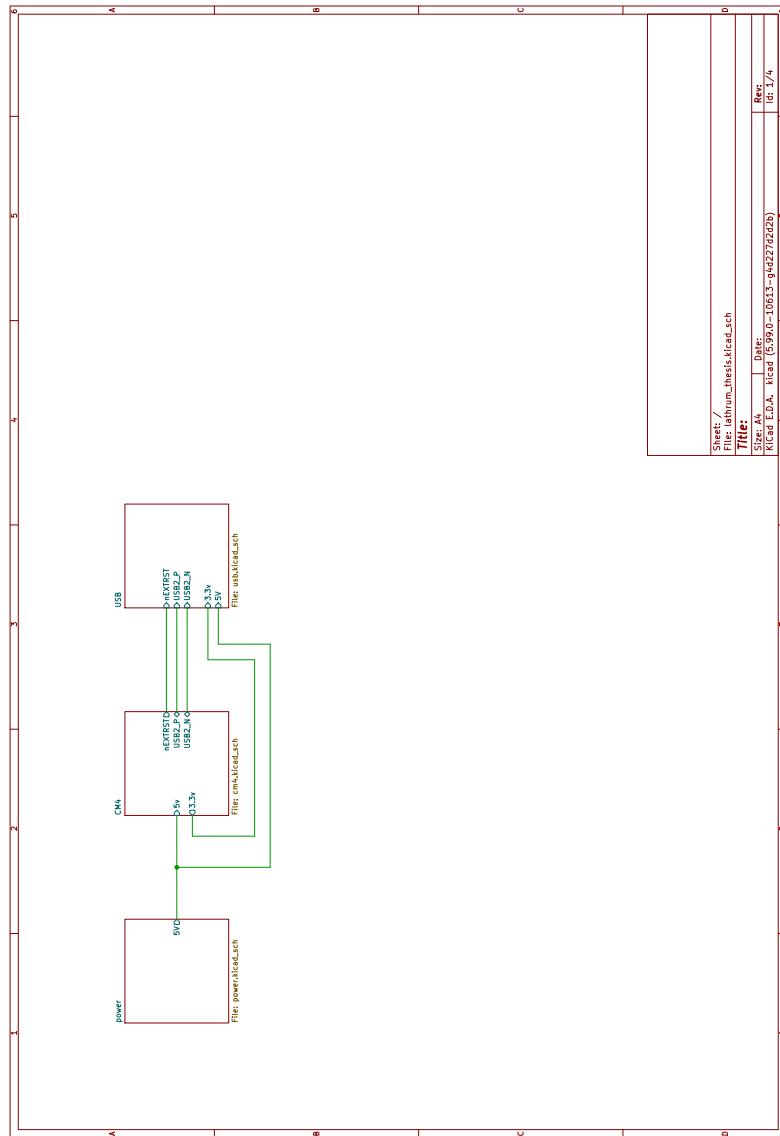


FIGURE A.1: The top level of the PCB board schematic

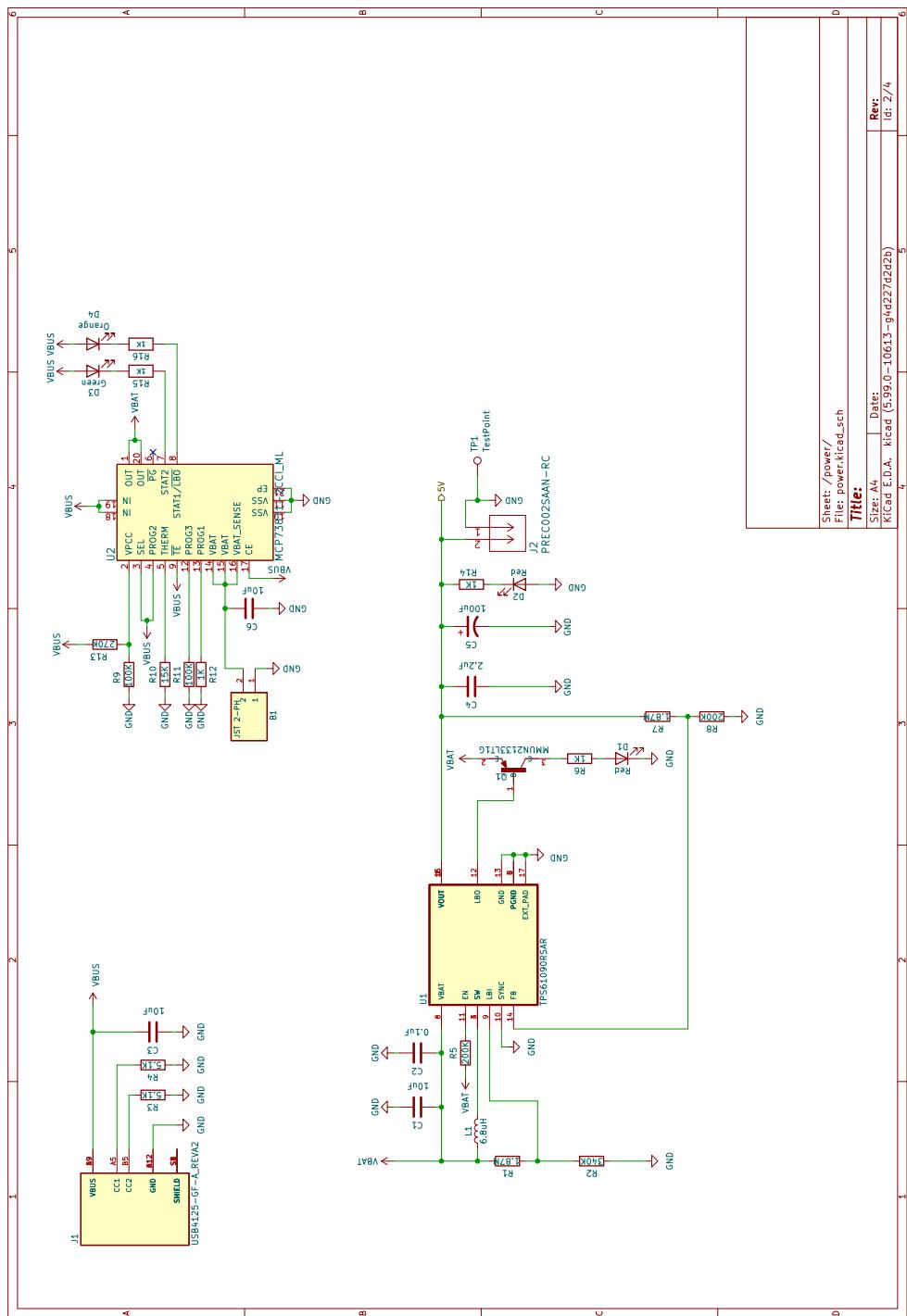


FIGURE A.2: Schematic layout for the power circuit, including battery management

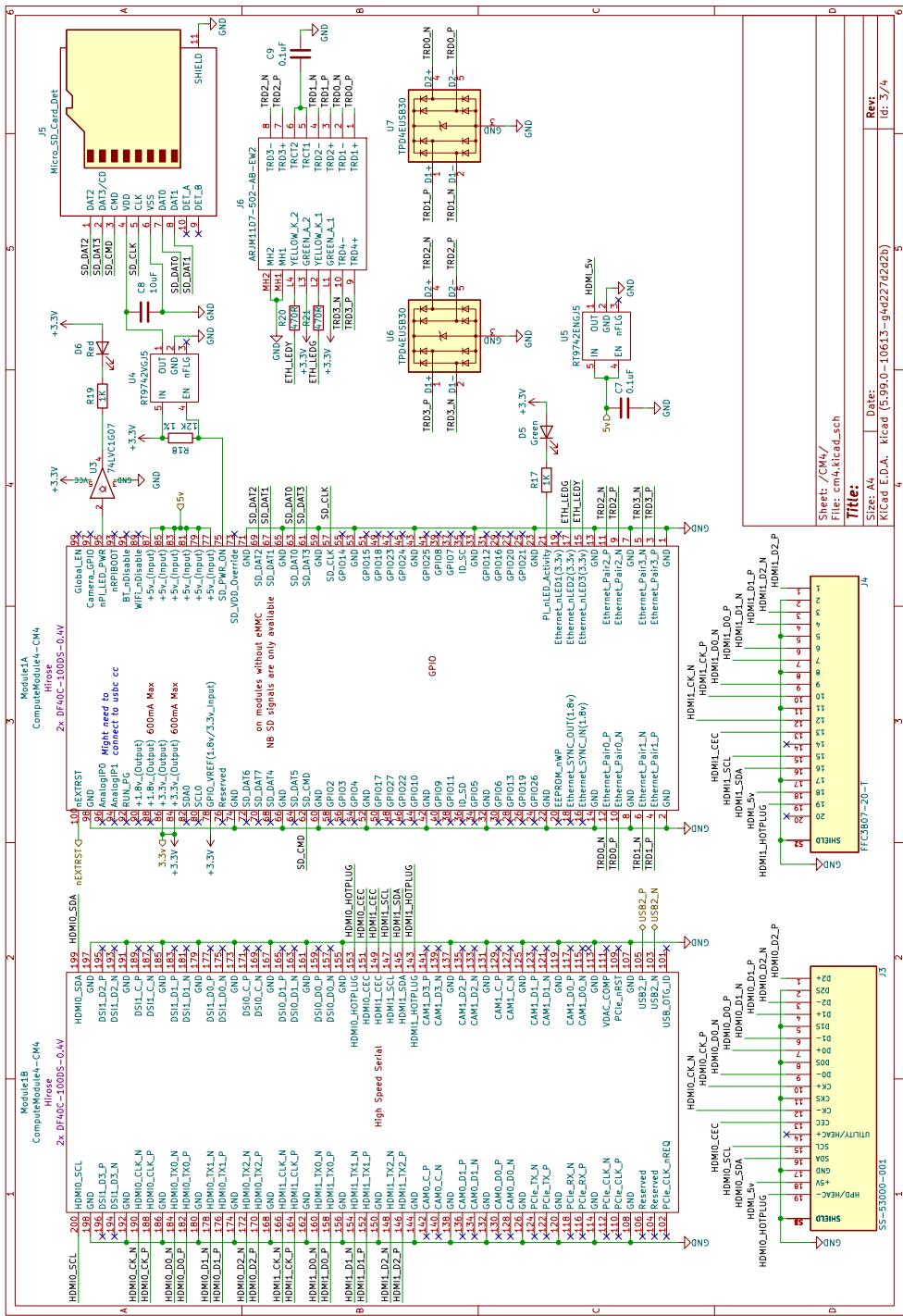


FIGURE A.3: Schematic layout for interfacing with the Compute Module 4

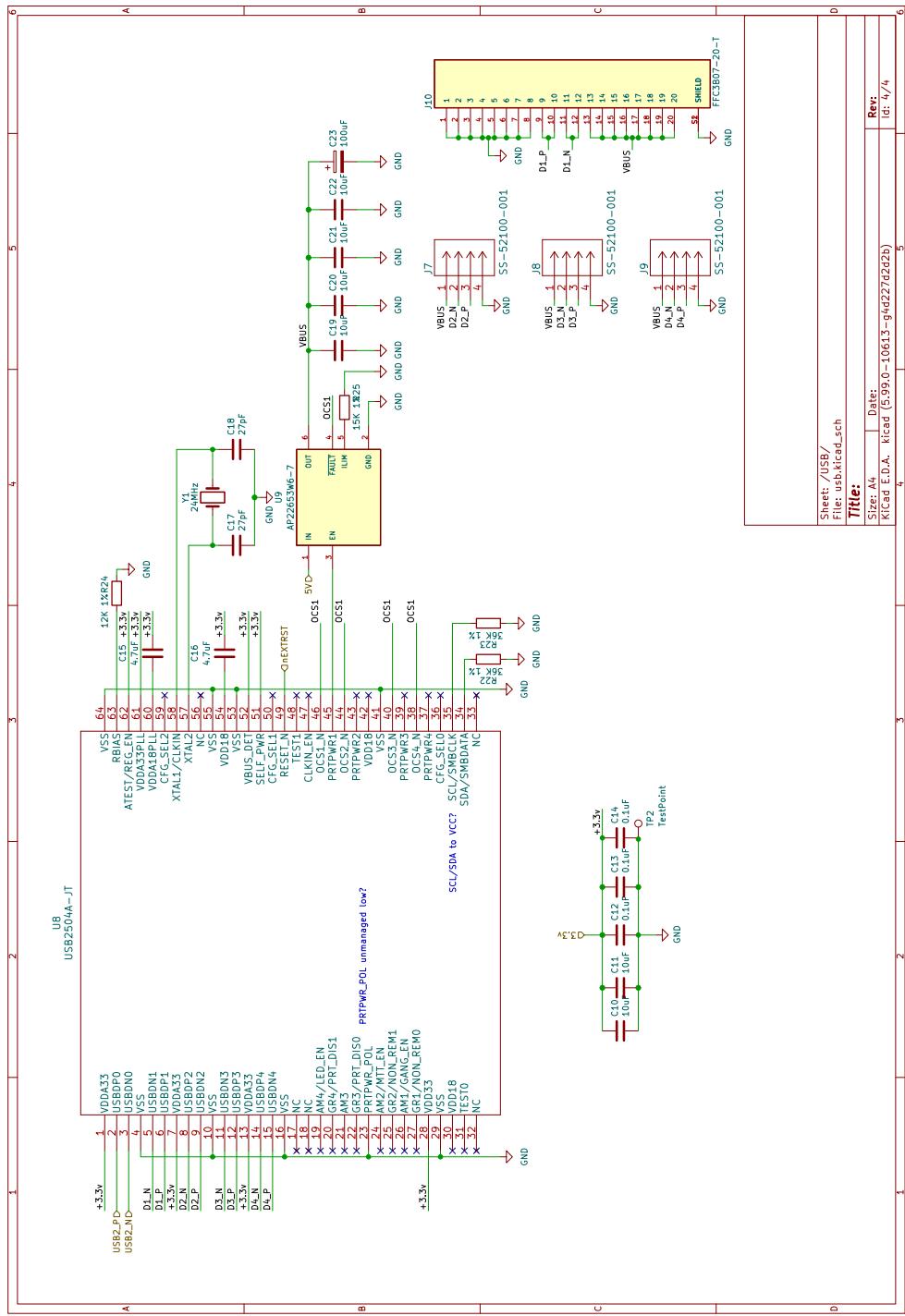


FIGURE A.4: Schematic layout for the USB hub and ports

A.2 PCB Layout

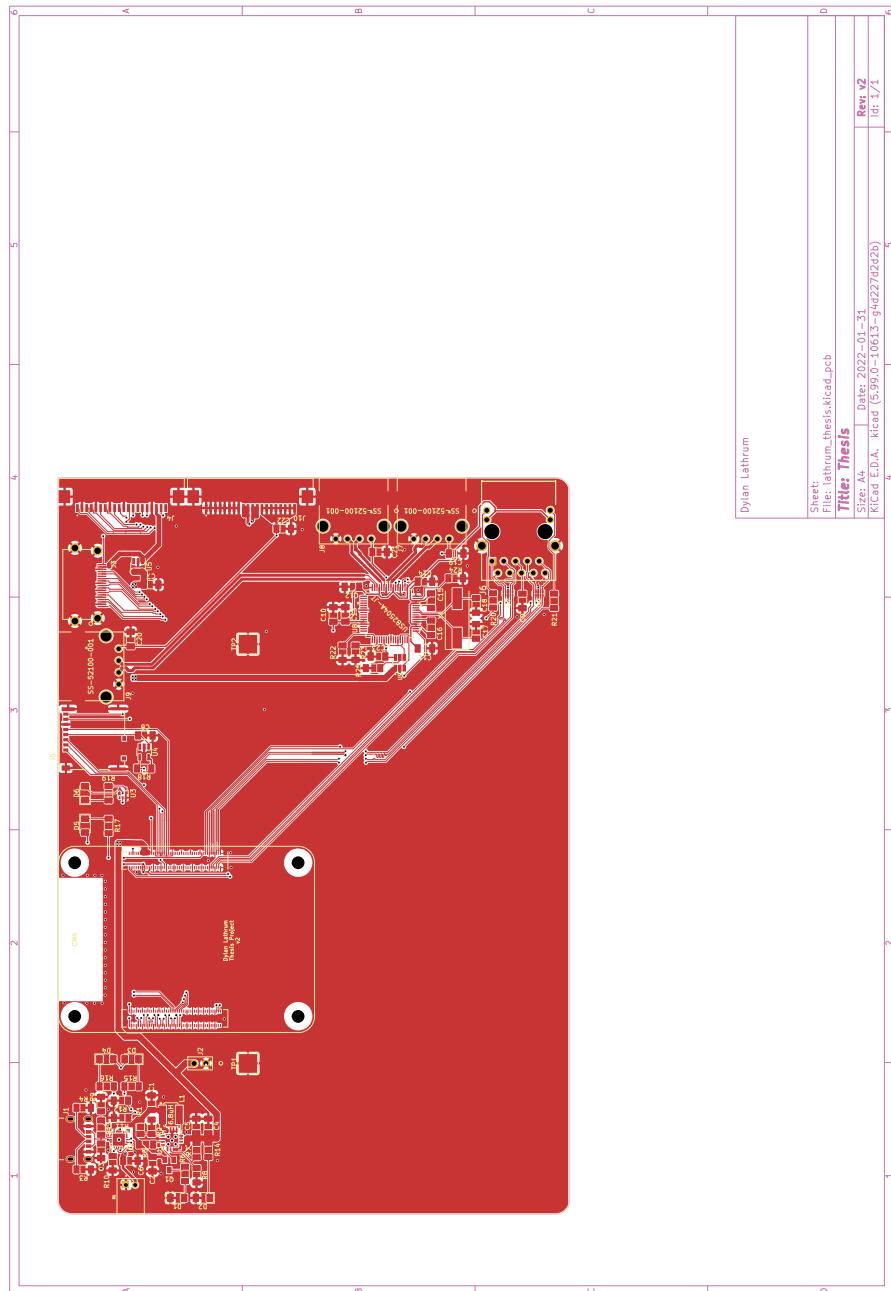


FIGURE A.5: Physical board layout for the front side of the PCB

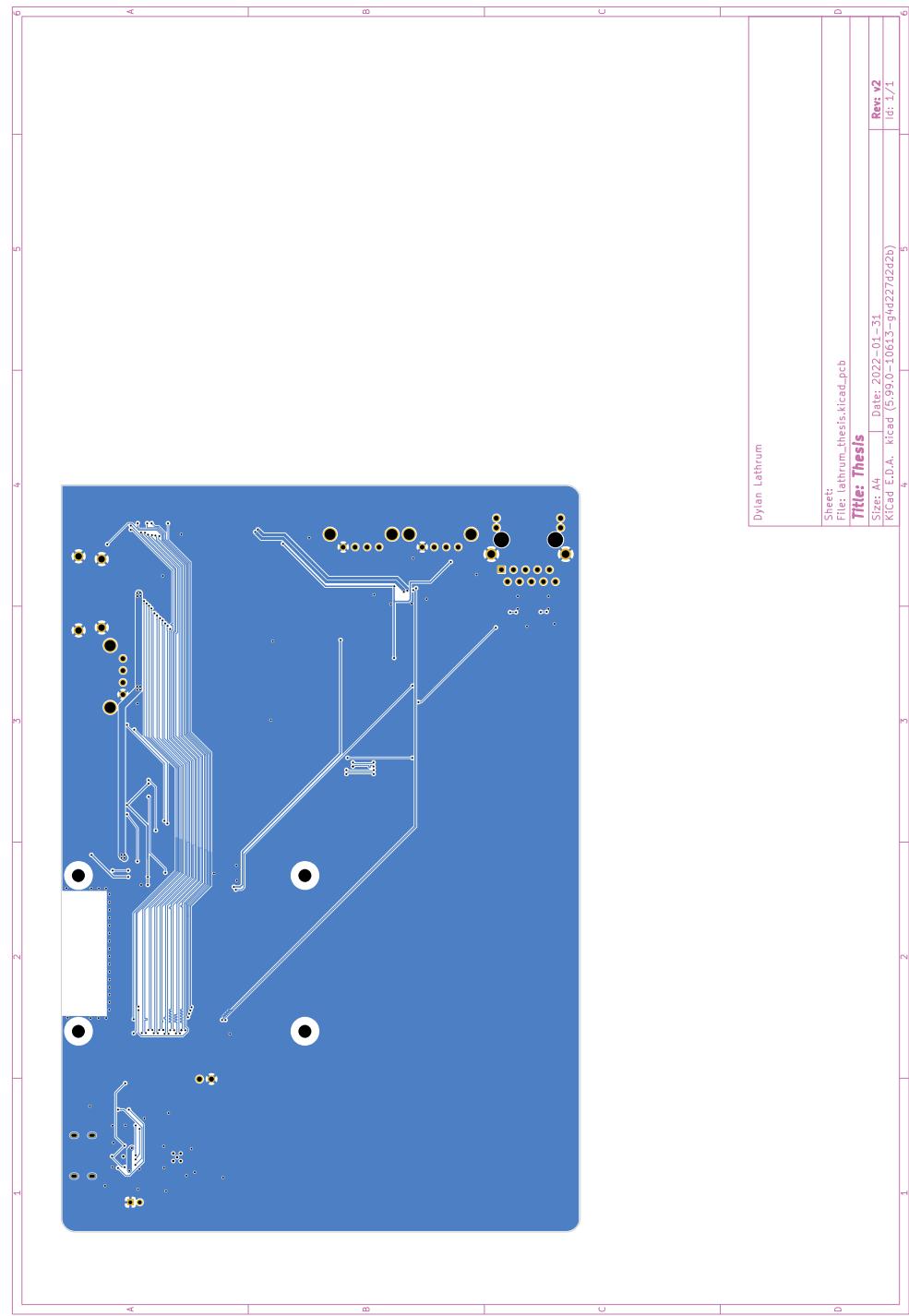


FIGURE A.6: Physical board layout for the back side of the PCB

Appendix B

Bill of Materials

Reference Designator	Count	Manufacturer's Number	Value
B1	1	B2B-PH-K-S(LF)(SN)	JST 2-PH
C1 C3 C6 C8 C10 C11 C19 C20 C21 C22	10	CL31B106MOHNNNE	10uF
C15 C16	2	CL31B475KAHNFNE	4.7uF
C17 C18	2	C1206C270K5GAC7800	27pF
C2 C7 C9 C12 C13 C14	6	CL31B104KBCNFNC	0.1uF
C4	1	CL31B225KOHNNE	2.2uF
C5 C23	2	GRM31CR60G107ME39L	100uF
D1 D2 D6	3	156120RS75300	Red
D3 D5	2	150120GS75000	Green
D4	1	150120AS75000	Orange
J1	1	USB4125-GF-A	USB4125-GF-A_REVA2
J2	1	PREC002SAAN-RC	PREC002SAAN-RC
J3	1	SS-53000-001	SS-53000-001
J4 J10	2	FFC3B07-20-T	FFC3B07-20-T
J5	1	5033981892	Micro_SD_Card_Det
J6	1	ARJM11D7-502-AB-EW2	ARJM11D7-502-AB-EW2
J7 J8 J9	3	SS-52100-001	SS-52100-001
L1	1	NRS5030T6R8MMGJ	6.8uH
Module1	1	CM4104000	ComputeModule4-CM4
Q1	1	MMUN2133LT1G	MMUN2133LT1G
R1 R7	2	RC1206FR-071M87L	1.87M
R10	1	RC1206FR-0715KL	15K

R13	1	RC1206FR-07270KL	270K
R18 R24	2	RC1206FR-0712KL	12K 1%
R2	1	ERJ-8ENF3403V	340K
R20 R21	2	RC1206FR-07470RL	470R
R22 R23	2	RC1206FR-0736KL	36K 1%
R25	1	RC1206FR-0715KL	15K 1%
R3 R4	2	RMCF1206FT5K10	5.1K
R5 R8	2	RC1206FR-07200KL	200K
R6 R12 R14 R15 R16 R17 R19	7	RC1206FR-071KL	1K
R9 R11	2	RMCF1206FT100K	100K
U1	1	TPS61090RSAR	TPS61090RSAR
U2	1	MCP73871-2CCI/ML	MCP73871-2CCI_ML
U3	1	74LVC1G07SE-7	74LVC1G07
U4	1	RT9742VGJ5	RT9742VGJ5
U5	1	RT9742ENGJ5	RT9742ENGJ5
U6 U7	2	TPD4EUSB30DQAR	TPD4EUSB30
U8	1	USB2504A-JT	USB2504A-JT
U9	1	AP22653W6-7	AP22653W6-7
Y1	1	FC4SDCBMF24.0-T1	24MHz

TABLE B.1: Bill of Materials for a single PCB Board.

Bibliography

- [1] *AnalogIP0 AnalogIP1 pins on CM4*. URL: <https://forums.raspberrypi.com/viewtopic.php?t=289344#p1748915>.
- [2] Eric Brown. *Nvidia Shield game console runs Android on tegra 4*. 2013. URL: <https://linuxgizmos.com/nvidia-shield-android-game-console-shipping-soon/>.
- [3] Eric Brown. *Nvidia Shield: Shipped, praised, critiqued, dissected*. 2013. URL: <https://linuxgizmos.com/nvidia-shield-reviews-and-teardown/>.
- [4] Steve Burbeck. “Complexity and the Evolution of Computing : Biological Principles for Managing Evolving Systems”. In: *Complexity and the Evolution of Computing : Biological Principles for Managing Evolving Systems*. 2007. URL: <https://evolutionofcomputing.org/Complexity%20and%20Evolution%20of%20Computing%20v2.pdf>.
- [5] Bryan S Burgin. *RelativeMouseMode*. 2013. URL: <https://social.microsoft.com/Forums/en-US/ae516842-1e3d-4943-8144-1b225e74684e/relativemousemode>.
- [6] *Chromoting Build Instructions*. 2020. URL: https://chromium.googlesource.com/chromium/src/+/refs/tags/82.0.4058.2/docs/old_chromoting_build_instructions.md.
- [7] *Covid-19 announcements archive*. URL: <https://eooss.asu.edu/health/announcements/coronavirus/announcements-archive>.
- [8] Chris Daniel. *Shield TV brings the smart home to a TV near you: Nvidia blog*. 2017. URL: <https://blogs.nvidia.com/blog/2017/01/04/shield-ai-home/>.
- [9] *Developing on remote machines using SSH and Visual Studio Code*. 2021. URL: <https://code.visualstudio.com/docs/remote/ssh>.
- [10] *DigiKey Electronics Home*. URL: <https://www.digikey.com/>.
- [11] Djajah. *Clientron U700*. 2008. URL: <https://commons.wikimedia.org/wiki/File:ClientronU700.jpg>.
- [12] *Everything about raspberry pi zero 2 W*. 2021. URL: <https://picockpit.com/raspberry-pi/everything-about-raspberry-pi-zero-2-w/>.
- [13] *Google Chromoting - remote desktop management*. URL: <https://www.miniorange.com/google-chromoting-remote-desktop-management>.
- [14] *HDMI vs DSI, Pros and Cons?* 2018. URL: <https://raspberrypi.stackexchange.com/questions/88075/hdmi-vs-dsi-pros-and-cons>.
- [15] *How GameStream Works*. URL: <https://www.nvidia.com/en-us/support/gamestream/how-gamestream-works/>.
- [16] *How long will the chip shortage last?: J.P. Morgan Research*. 2021. URL: <https://www.jpmorgan.com/insights/research/supply-chain-chip-shortage>.
- [17] *How to make the PCF85063AT work with CM4?* 2020. URL: <https://forums.raspberrypi.com/viewtopic.php?t=293632>.

- [18] J.O. Kephart and D.M. Chess. "The vision of autonomic computing". In: *Computer* 36.1 (2003), pp. 41–50. DOI: 10.1109/MC.2003.1160055. URL: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1160055>.
- [19] Gad Levanon. "Remote work: The biggest legacy of covid-19". In: *Forbes* (2020). URL: <https://www.forbes.com/sites/gadlevanon/2020/11/23/remote-work-the-biggest-legacy-of-covid-19/?sh=699b0767f590>.
- [20] *manjaro-arm-tools*. URL: <https://gitlab.manjaro.org/manjaro-arm/applications/manjaro-arm-tools/-/tree/master/>.
- [21] L. Meyer. "5 Reasons Schools Still Need Desktop Computers: Despite the Growth of Mobile Learning, Desktops Still Play Important Roles in the 21st Century Classroom". In: *T.H.E. Journal Technological Horizons in Education* 41 (2014), p. 20.
- [22] *Moonlight*. URL: <https://moonlight-stream.org/>.
- [23] *Mouser Electronics - electronic components distributor*. URL: <https://www.mouser.com/>.
- [24] *PCB prototype & PCB fabrication manufacturer*. URL: <https://jlpcb.com/>.
- [25] Raspberry Pi. *Compute Module 4*. URL: <https://www.raspberrypi.com/products/compute-module-4/>.
- [26] Raspberry Pi. *Compute Module 4 IO Board*. URL: <https://www.raspberrypi.com/products/compute-module-4-io-board/>.
- [27] *Raspberry pi documentation*. URL: <https://www.raspberrypi.com/documentation/computers/compute-module.html>.
- [28] *Solid notebook demand in consumer and education marks 2021 Q1 as the fourth consecutive quarter of double-digit growth in EMEA PC market, says IDC*. 2021. URL: <https://www.idc.com/getdoc.jsp?containerId=prEUR147632021>.
- [29] Maxim Tamarov. *Chip shortage driving up PC prices, Wait Times*. 2021. URL: <https://www.techtarget.com/searchmobilecomputing/news/252499402/Chip-shortage-driving-up-PC-prices-wait-times>.
- [30] *The VNC family of Remote Control Application*. URL: http://ipinfo.info/html/vnc_remote_control.php.
- [31] *Understanding remote desktop protocol (RDP) - windows server*. URL: <https://docs.microsoft.com/en-us/troubleshoot/windows-server/remote/understanding-remote-desktop-protocol>.
- [32] Linus Upson and Caesar Sengupta. *Next step in the chrome OS journey*. 2012. URL: <https://blog.google/products/chromebooks/next-step-in-chrome-os-journey/>.
- [33] Tatu Ylonen. *RFC 4251 - the secure shell (SSH) protocol architecture*. Ed. by ChrisEditor Lonvick. 2006. URL: <https://datatracker.ietf.org/doc/html/rfc4251>.