CryptoHash

David Cheung, William Zhao, Lulu Shyr

Department of Electrical and Computer Engineering,
Carnegie Mellon University

***Abstract*—CryptoHash is a system capable of algorithmically choosing the optimal distribution of compute resources allocated to mining from a selection of different cryptocurrencies. It operates at a 10% lower hardware cost yet has performance measured in the number of hashes within 90% of conventional methods. Through the use of customized hardware, our cryptocurrency miner is able to maximize profits even with a low initial capital investment. The design is scalable and benefits from having multiple miners running in parallel.**

***Index Terms*—Cryptocurrency mining, FPGA, hashing, proof of work, Raspberry Pi, SPI protocol**

I. Introduction

The increase in retail interest in cryptocurrencies the past five years has come alongside a volatile increase in prices as well. While early miners used standard laptops and consumer-grade computers, current miners must adopt specialized hardware in order to remain profitable. Gone are the days where an individual could run a Bitcoin miner on their CPU overnight and expect anything more than a few pennies for their trouble. As their name suggests, application-specific integrated circuit miners target a specific cryptocurrency running a specific hashing algorithm. These miners are not customizable and once a design is chosen, it cannot be changed.

Our solution aims to be a fully customizable miner that follows both interday and intraday price action in order to select which cryptocurrency to mine. This approach no longer suffers from the rigidness of ASICs while maintaining a respectable hash rate. The user will also be able to view metrics relevant to the system such as current hash rate, total returns generated, and price charts for the various supported cryptocurrencies.

II. Use-Case Requirements

The web app will be displaying current metrics relevant to the user such as the hashrate and current prices of mined coins. These metrics will be updated every minute such that they are frequent enough for the user to receive real-time data but spaced out such that our communication network can support it. The choosing algorithm will also make its decision based on these minute by minute updates. The system must choose the optimal way to distribute compute resources. To make this decision, the choosing algorithm will also examine all cryptocurrency pricing data from the preceding three months. Three months is the target time frame because decisions made in the present should be weighted more heavily on what has happened recently. However, too short of a timeframe and there wouldn't be enough data to generate a meaningful trend to base our model on.

For this system to be profitable and users to purchase our system, the hash rate must be at least competitive with solutions currently on the market. Thus, we will be comparing our system with GPUs and their respective hash rate. These GPUS are more generalized than ASICS and can switch from mining one coin to another relatively quickly. This matches the intention of our system as well. Consumer-targeted GPUs have a hash rate around five million hashes per second (5 Mh/s). We are looking for at least 90% of this hashing power but with a 10% lower hardware cost.

An important component of the system is having the cryptocurrency miners react in real-time to the choosing algorithm. Thus, the system will have to keep communication overhead minimal by limiting the amount of time it takes for a command to be reflected. At the instance when the choosing algorithm makes a new decision, the miner should change its configuration to reflect these new settings in less than 10 seconds. This ensures that our system is able to keep up with market moving news.

III. Architecture and/or Principle of Operation

Our Raspberry Pi will pull the latest pricing data for our targeted cryptocurrencies using Binance's API. This data is then inputted into our custom machine learning model which decides the optimal percentage of computing power to allocate to each cryptocurrency. It will take into account a number of factors such as current price, volatility and daily volume to name a few. Our web application will also display this pricing data in an informative way for the user.

The machine learning model must predict the next best configuration based on previous pricing data and the other indicators mentioned above. In order to maximize profits, our system is predicting which cryptocurrency will outperform relatively in the next minute. If Cryptocurrency A falls by 0.10% but Cryptocurrency B falls by 0.05%, we want our system to take advantage of this spread and adjust accordingly. Depending on the training results of our model, this could mean increasing our weighting towards A because mean reversion says that A will eventually return to its long running average or our model might select B because, in the short term, it exhibits better performance.

After the choosing algorithm selects an optimal spread, the Raspberry Pi stores this data in its memory. The communication software on the Raspberry Pi then has to

construct data packets containing commands or other data necessary for operation. This software must serialize the data from the internal buffers into a bit stream that can be transmitted through the GPIO pins. Our system will connect with established mining pools that allocate work for each of the cryptocurrencies that we are interested in. The Raspberry Pi will retrieve some work from these pools and send this data to the FPGAs as well.

The entire system will be communicating using the SPI protocol which requires four wires to implement. The Raspberry Pi software will generate clock transitions and assert select wires in accordance with this protocol.

Once the data reaches the FPGA GPIO pins, it is deserialized to recreate the initial data packets. The communication modules are responsible for performing this deserialization and memory storage. The use of a Hardware Description Language allows us to design the module with minute detail and also enables us to model the system. The FPGA will output a signal back to the Raspberry Pi once these packets are received to acknowledge their full transmittal. However, in the case that data transmission is corrupted, it will inform the Raspberry Pi of this corruption and request it resend the full packet.

The hashing modules on the FPGA are responsible for taking each of the cryptocurrency proof of work puzzles and solving them. For the majority of them, this consists of a specific hashing algorithm being performed on a block of data and comparing the resultant hash to some target. The objective is to be the first to find this hash in order to receive a reward. Once a valid hash is found, it is outputted from the hashing module back to the FPGA's communication module. There, it will be serialized to be sent using the GPIO pins following the SPI protocol.

When the Raspberry Pi receives this hash, it sends it to the mining pool for final submission. The mining pool will distribute a reward to each member if the hash was the first to be accepted by the blockchain. There will be instances where we will receive a reward even when we do not submit a hash in time. This is because each member of the mining pool submitting a valid hash rewards all other members. The Raspberry Pi will keep track of our total rewards earned so far to be displayed on the web application.

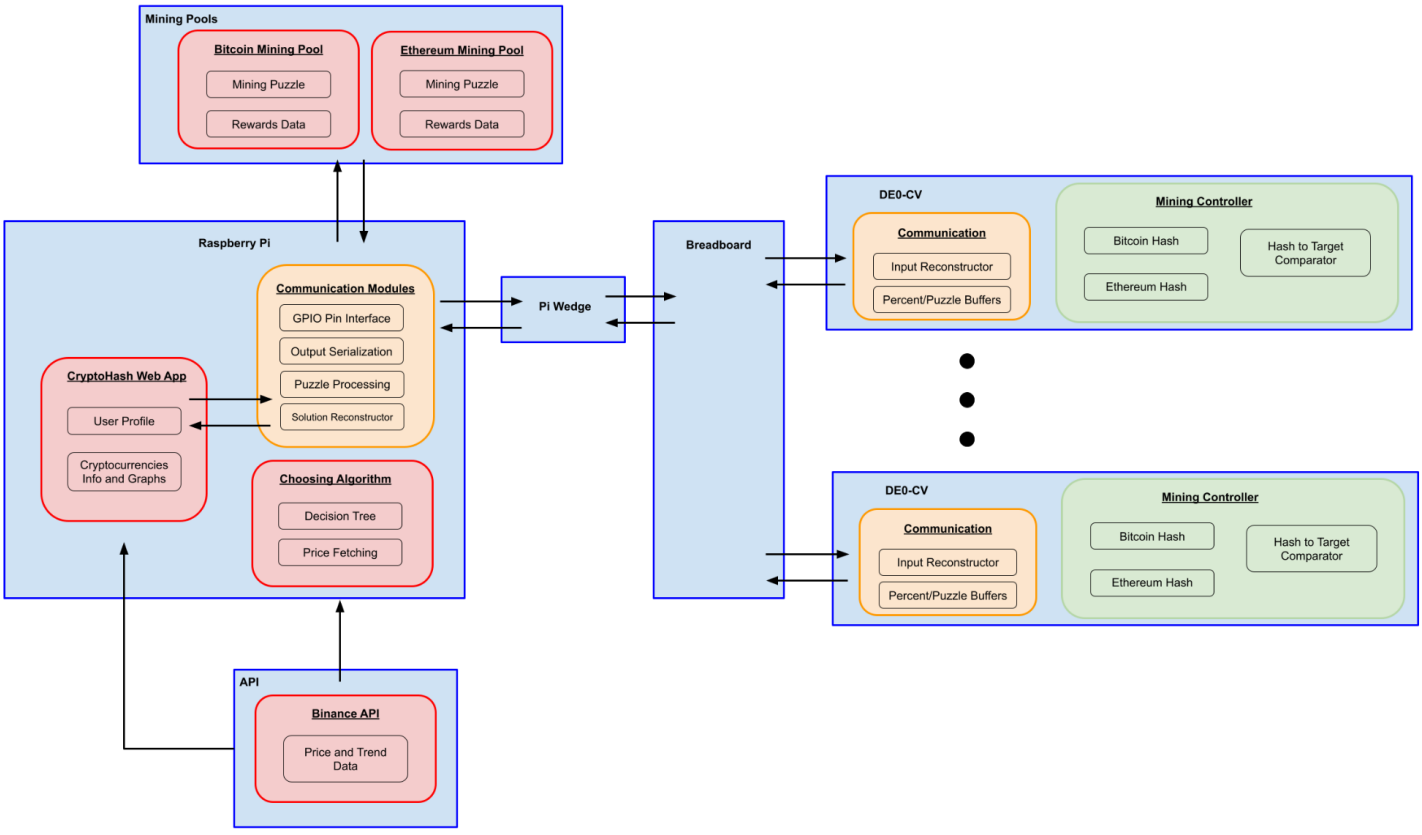


Fig. 1. Block drawing of overall system, illustrating the dataflow between various subsystems

IV. Design Requirements

The web app must be responsive to user inputs. The time when a user switches a graph chart from one time frame to another must be less than 100ms. This should be the case whether the user goes from a month view to a week view or from a year view to a day view. When the user changes the graph to display another cryptocurrency, this understandably takes longer to clear the current graph, fetch the new data using Binance's API, and create the new graph with this fresh data. We are limiting the refresh time for this use case to be 200ms.

Our choosing algorithm must fetch data frequently in order to make an informed decision on the optimal distribution of computing resources. In order to maximize profits, we want to be aware of any sudden changes in prices. Thus, the data fetching must happen at the latest, every minute to ensure our system remains optimal. The model we create will also be expected to have 70% accuracy when predicting the next best move. Some of this inaccuracy will come from the fact that cryptocurrencies are inherently volatile. We must also consider the fact that our model cannot predict when a certain piece of news will impact one cryptocurrency more than another.

Once the choosing algorithm has an optimal configuration, it must send this information to the SBC which in turn passes it along to each miner. From the time that the choosing algorithm makes a decision to when the first miner receives this configuration, no more than 10 seconds must pass. This keeps our communication overhead minimal and dedicates compute resources to the sole task of mining.

Communication is a key component of our system. We will be running five miners in parallel each with a set of wires connected to the SBC. Any time there is a physical connection to a wire or from a wire, there is the possibility that bits will be dropped or the data could become corrupted. Our system must be resilient enough such that 95% of packets are sent properly.

As mentioned in the Use Case Requirements, our system will need a competitive hashing rate in order to be a viable alternative to GPUs. Hashing a block of 512 bits with SHA-256 takes 64 clock cycles. For a 50 MHz clock, this equates to one hash every 1.28 microseconds or ~781 kh/s. We will need to modularize our hashing controller and generate multiple smaller subsystems in order to improve this performance. To achieve a hash rate comparable to GPUs of 5 Mh/s, our hashing modules will need to be compact enough such that each miner will be able to support multiple copies running in parallel.

Each of the hashing modules will also need a long enough critical path such that they are able to complete a substantial amount of work yet stay within one clock cycle. The design must have the critical path of the system reside in the hashing modules rather than communication. This allows the system to take advantage of each clock cycle to the fullest extent possible. The latency of the hashing module will be within 80% of the clock period.

V. Design Trade Studies

The trade-offs for our design lie primarily in the choice of the hardware. At the most fundamental level, the hardware we use include a Raspberry Pi to facilitate communications and FPGAs so that we can configure the system. The main driving factors that influence what hardware we select are the performance and the component cost. Other trade-offs include one between profitability and the supported coins and model complexity and accuracy. These trade-offs influence system level complexity and communication complexity.

A. FPGA Trade-Offs

At the very core of our design, we want to preserve a relatively low hardware cost while still allowing for the customization requirement and still meeting the performance requirement we set on ourselves. We recognize that FPGAs will not be the most efficient miners and will not be viable in mining popular coins by themselves simply because they won't be able to achieve the same hash-rates as specialized hardware will. However, to create a system that can be reprogrammed quickly, we choose to use FPGAs because of our relative familiarity with them. To realize the gap between achievable hash rates, we use multiple boards to very crudely model a multicore system. In this essence, we are also limited by our allocated budget that needs to be shared with other components. This fundamentally pushed us to use the components already listed in the Inventory, where we ultimately made a decision between the DE0-CV and DE2-115 boards. The DE0-CV differ primarily in the number of logic units and the price. DE0-CV boards are relatively cheap as compared to DE2-115 boards, but they have much fewer logic elements. We ultimately chose to go with the DE0-CV boards since they're smaller and multiple of these boards would be easier to assemble. The boards are also cheaper, which ties in nicely with the idea of lower hardware cost. The relatively fewer logic units problem is resolved by having multiple FPGAs as long as it stays within budget.

B. SBC Trade-Offs

To make the design portable and have the connections be more easily understood, the communication command center is implemented as a single board computer. In particular, our design uses a Raspberry Pi Model 4. Once again respecting the allocated budget, we wanted to take a look at the inventory first before making any purchases. The Raspberry Pi Model 3 was not worth the price for the small amount of computing power it would be able to provide. The Raspberry Pi Model 4 also came in differing sizes of RAMs, which struck a tradeoff between performance and cost, but we didn't need that much computational strength from the single board computer, we

settled with 4 GB of RAM. The largest tradeoff analysis came down to the Raspberry Pi Model 4 and the RockPi. We compared the cores, the I/O, and the operating system/documentation. The Raspberry Pi has a quad-core while the RockPi has a six-core, but there are 4 main things the SBC needs to continuously do, so any number of cores greater than or equal to 4 would suffice. Both Pi's were capable of supporting GPIO I/O, which is the direct communication between the FPGA and the SBC. The largest factors that played into choosing the Raspberry Pi was that the documentation was more established than for the Rock Pi and that we wanted the quicker cores on the Raspberry Pi so that tasks would not get scheduled onto the slower cores on the Rock Pi for no benefit.

C. Supported Coin Trade-Offs

The beauty of the project is being able to mine multiple cryptocurrencies at the same time and have a choosing algorithm that can choose between the coins to maximize profit. However, there exists a plethora of coins, only a subset of which can be mined with a proof-of-work system such as ours. The coins that are proof-of-work also tend to not share the same algorithms, where adding support for a coin using means needing to have configurations for that algorithm in the FPGA side of things. This system allows for easy scalability, but the modules have a significant overhead in data size and communication because the headers would have to change. For example, adding functionality for a currency like Litecoin, would require implemented Scrypt modules for the FPGA, which would also necessitate recompilation.

Our implementation focuses on Bitcoin and Ethereum, the most popular cryptocurrencies. While maximizing profits is one of our project's requirements, we needed to make a tradeoff where instead of mining some altcoin that could have some more potential to give us higher returns, we mine a coin that is well established. Again, our primary focus is not to make the most efficient miner that makes as much money as possible. Our focus is to do that, but also constrained under the selection of the coins. As well established coins, the protocols of how they work are more understood and there exists more documentation and research on mining these coins. It also serves as a more welcoming setup where people can mine the currencies they are more familiar with.

D. Selection Model Trade-Offs

The choosing algorithm is modeled as a decision tree that is trained every month and inferred from every minute. There are a few trade-offs that exist here, in choosing the type of the model and the hyperparameters.

In choosing the decision tree, the model we chose had to be run very quickly and without much computational power because we don't want the entire system to always constantly be under great stress. The model is trained and inferred on the Raspberry Pi, which also has to deal with all of the communication coming from the webapp, mining pools, and the FPGAs. It is imperative to keep other computation relatively low so that computing power for the communication is not compromised much. The training process for a decision tree also does not take very long either since there isn't any weight updates in the form of back propagation or gradient descents. This allows for the model to be retrained quickly on a large data set such as the 3 months of data. The largest drawback that decision trees usually have is that they're usually not that accurate when compared to something like a Deep Neural Network. The neural networks usually pay for this accuracy with the long time it takes to converge. Due to the nature of cryptocurrency volatility, it is hard to predict the behavior especially since it is so sensitive to current events, or other events that price and price trends tend to not explain very well.

The hyperparameters for the model include the 3 months of data for training, the frequency that the model or inference is updated, and the history of mining pool rewards. For the training data, we have access to a lot of data from the Binace API, but if we train with too much data, then the model will be very flawed where it tries to use very old data to predict the future. However, we can't settle on too little since it wouldn't be enough to paint a good picture of the climate of the prices due to the inherent price volatility. The monthly model update was a trade-off between performance and weight updating. The model needs to update its weights so that it doesn't become obsolete. Updating it every month is also a way to amortize the cost of training. The 1 minute switching allows the FPGAs to have some leeway in switching between the cryptocurrencies. We also want to track the more recent rewards from the miners to try and implement the greedy algorithm, biasing the decision based on which mining pools provided more taste.

VI. System Implementation

A. Web Application

The web application will be built using the Django framework in python and will be deployed onto the cloud using the Amazon EC2 instance. Django provides an authentication system which handles user accounts, groups, and permissions. We will use this to implement our login and registration page. When the user first accesses the web application, they will be greeted by our login page, they will then input their username and password to log in. If they do not have an account already, they can click registration and be redirected to the registration page to register for an account. Later on, if they are already logged in, they will be redirected to the home page. On the top of every page, there will be a navigation bar where users can choose to logout or go to a specific page.

On the home page, the user will see a table of the current cryptocurrency prices, price and percentage changes within the last 24 hours, high and low prices, and quote volume. In addition to the statistics displayed in the table, the user will also be able to see the candlestick charts and the line charts for the cryptocurrencies to get a better sense of the trends in cryptocurrency. The statistics for the cryptocurrencies will be obtained by using the Binance API.

By default, a machine learning algorithm will choose how much to mine on each cryptocurrency (BTC and ETH), but on the bottom of the page there's a dropdown menu where the user can choose the ratio of how they want to mine their cryptocurrency if they wish to change it. We will also be displaying the amount of cryptocurrency it has mined and the hashrate of mined coins from the FPGA on the top of the home page in a profile section. The web application will refresh every minute such that the metrics will be updated frequently enough for the user to receive real-time data.

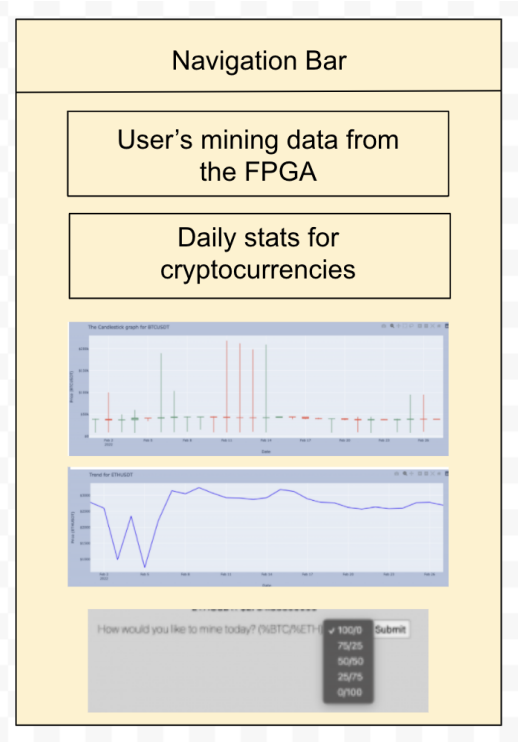


Fig. 2. Diagram of Screenshot of Web Application

B. *Raspberry Pi and Mining Pool Communication*

Acting as the communication center, the Raspberry Pi needs to communicate with the mining pool for the respective cryptocurrency. The Raspberry Pi will obtain work, or the puzzles, from the mining pools primarily through the stratum tcp protocol that modern mining pools use. While the reward for mining solo would be much higher, we do not configure it as such so that we do not have to compete with miners sporting ridiculously high hash rates that we can not achieve. Connecting and communicating with the respective blockchains also incurs unnecessary overhead in communication and resources that is beyond the core concept of our project.

The information needed to connect to different mining pools, including different usernames and passwords will be stored in a configuration file. The Raspberry Pi will read the file and connect to the mining pools and start to receive work from the mining pools. The work that the mining pool provides will have to be processed and placed into a PUZZLE packet to be sent off to all of the FPGAs. The FPGAs are configured to perform the check themselves and will only send the Raspberry Pi their solution in a SOLUTION packet when they have a valid solution. Since the packets are assumed to be valid, the Raspberry Pi will parse the packet and send the solution to the mining pool as soon as possible to maximize the rewards. Therefore, the Raspberry Pi will have to continuously stay in connection with the mining pools, which also includes detecting whether the pool has died.

The communication between the Raspberry Pi and mining pools is most simplistically captured with the getwork function and HTTP protocol. However, getwork has since been deprecated, necessitating the Stratum Mining Proxy to bridge getwork with the stratum mining protocol.

C. *Coin Choosing Algorithm*

The choosing mechanism will automatically switch between what cryptocurrency our setup mines for. It is also the default configuration that seeks to optimize which cryptocurrencies are mined in an effort to maximize profit. At a high level, it is a trained model that is trained every month with the past 3 month's worth of data from the Binance API. It will perform inference, and thus switch between cryptocurrencies, on a minute basis. The model takes as input the current price of the cryptocurrencies, the current price trend for the past 10 minutes, and the recent rewards from the different mining pools. The model will have a greedy algorithm that will bias the model to want to mine more of the cryptocurrency that has higher payouts. Another factor that isn't explicitly an input is the relative strength index, which provides a picture of how the price momentum is swinging. The model will calculate the relative strength index with the Binance data from the past 10 minutes. The model strikes a balance between the price and trend where if a coin grows in price with a favorable trend, it will attempt to bias its weights to switch some portion of mining power to that coin.

The exact algorithm that the choosing mechanism is based on is a decision tree, where we use each input and calculated input as nodes. Every month, the model will be trained with the 3 months of data from Binance. Throughout the month, the model will run inference every minute where it runs through the model with the current price, current trend, and earnings history from the mining pools. The decision tree result is stored and compared with the next time to generate test results that will later be used as accuracy checks for the testing and

validation portions. The decisions will also have some threshold value that changes every 10 minutes, which is also the time frame of data we keep track of as inputs. The most recent earnings from the different mining pools will also be stored in addition to the fractional share of the mining power corresponding to the time of the earnings.

D. Raspberry Pi to FPGA Communication

There will be two types of data packets that the Raspberry Pi will send to the FPGA: UPDATE packets and PUZZLE packets.

An UPDATE packet will include an eight bit header followed by 32 bits of data. The header has five bits which indicates the FPGA the packet is targeted towards, one bit to distinguish an UPDATE packet from PUZZLE, and two bits for the number of data bytes in the packet. The 32 bits of data are separated into four equal eight bit portions. Each of these eight bits contains a numerical value which tells the FPGA how much of that particular cryptocurrency to mine. A data packet of {8'd10, 8'd25, 8'd40, 8'd25} tells the FPGA to use 25% mining power for coin one, 40% for coin 2, etc. These percentages are automatically generated by our choosing algorithm.

The other type of packet is the PUZZLE packet. This contains the proof of work puzzle received from the mining pool. Different coins have different puzzle sizes so we need to handle variable length packets. For the two coins we have planned, the Bitcoin block header is 80 bytes and for Ethereum it is 508 bytes. This puzzle is stored in the Raspberry Pi and serialized in order to transmit as a single bit through the GPIO pins.

We are using a Pi Wedge with one end inserted into the 40 GPIO pins found on the Raspberry and the other end into a breadboard. The FPGAs will receive their inputs by connecting jumper wires from their GPIO pins into the breadboard. Using a Pi Wedge breadboard implementation allows us to scale the system without being limited by the amount of pins available on the Raspberry Pi.

E. FPGA Hashing Module

We will be creating our own mining controller to handle the puzzle input and distribute work to the numerous hashing modules. It first receives a serialized input through its GPIO pins that it must recreate into the UPDATE or PUZZLE packet. After a distinction is made, either the percentages or the proof of work puzzle is stored into internal buffers we create in the FPGA. Simultaneously, the FPGA will continue to hash the previous puzzle until a new one is fully received.

To generate a valid hash to return to the mining pool, the hashing module on the FPGA will cycle through a series of numbers to hash along with the inputted puzzle. Although we now have a hash, it must be validated before it can be returned to the mining pool. To do so, there will be a separate module whose job is to compare this resultant hash with a target number inputted along with the puzzle. The hash is accepted only if it is less than the target number.

Once the hashing module finds a valid hash, it will communicate this back to the mining controller. The controller will take the 256 bit hash and serialize it in order to send this to the Raspberry Pi. The total communication packet will be 264 bits because there is the inclusion of an 8 bit header field to inform the Raspberry Pi which FPGA sent the hash and for which cryptocurrency the hash is for.

The process of outputting data to the Raspberry Pi follows a similar process to inputting data. The FPGA sends the data as a single bit using the GPIO pins on the board. This data travels through a wire to the breadboard, back to the Pi Wedge, and finally to the Raspberry Pi's GPIO pins.

VII. Test, Verification and Validation

A. Tests for Web Applications

The most basic validation method for the user interface on the web application will be clicking through each button and tab making sure the page loads according to user input. For example, clicking on the logout tab and submitting the registration information should bring the user back to the login page and inputting the correct username and password on the login page should bring the user to the home page. We can print out the information stored in the profile model in the terminal to make sure all the information is stored correctly. On the home page there will be five options for the mining ratio that the user can choose from. After hitting the submit button, the page should refresh and we can utilize a print statement that prints out the option selected to make sure the data is stored correctly and ready to send to the FPGA. For the table and graphs that are generated from the Binance API, we can compare the numbers to the values displayed on the official Binance website to make sure we are getting the correct data from the API database and parsing it correctly on the web application. Similarly with the graphs, we can compare them with the graphs on the official website to make sure the general trend is correct.

B. Tests for Model Accuracy

Validation of the choosing algorithm is difficult to quantify and even more difficult to create a highly accurate predictor of the price fluctuations. Our requirement is set to 70% which is better than the 50% from randomly guessing. The accuracy isn't very high because price and price trends don't paint a complete picture of how cryptocurrency prices fluctuate. The prices are very volatile and can be easily swayed by public figures, current events, or even just random buyer hesitancy.

These are all difficult to quantify and virtually impossible to predict in advance.

Through just periodic runs, the model will accumulate a running log of how it predicts the price to change and the actual price change. The model in addition to outputting the ideal spread will also keep track of how it feels the price will be in the next run. On the next run, this prediction will be compared to the real result and its correctness will be saved, which ultimately becomes the accuracy for that run. The longer that the model runs for, the more accurate the accuracy rating will be, but to a certain extent. The longer it runs, the more outdated the training data for the model will be. So although the results might be a little skewed in the size of the data set, we can validate more quickly by choosing to check the accuracy for a short time period, currently set at 100 results.

C. Tests for Configuration Switching

One of our requirements was to be able to switch FPGA configurations within ten seconds of the command being inputted. We will test for this by having the choosing algorithm select a pre-programmed configuration to send to the FPGA. The Raspberry Pi will start an internal timer once the configuration has begun sending. After the FPGA receives this input, parses it, and performs the necessary updates to reflect these new changes, it will output a signal back to the Raspberry Pi. Once the Raspberry Pi receives this signal, it will stop its timer. We will run several of these configurations and inspect whether any of these instances caused the response time to be greater than ten seconds.

D. Tests for Packet Bit Errors

There are two ways of physical communication between the components of our system, from the Raspberry Pi to the FPGA and from the FPGA to the Raspberry Pi. We can test both ways at once by having the FPGA pass its input through to its output. The Raspberry Pi initiates the process by sending randomized packets selected in a provided test file. These test inputs follow the normal path through the Raspberry Pi GPIO pins, Pi Wedge, breadboard, and FPGA GPIO pins. Once it reaches the FPGA, the inputs are simply returned as outputs where it follows the same path back to the Raspberry Pi. The Pi will examine this output and compare it with the randomly generated test input. A shortened output is the result of dropped bits and flipped bits is due to data corruption. Each error instance is tallied and the overall ratio of incorrect bits to total bits is recorded. This process is run multiple times through the use of the randomized tester to generate an accurate percentage of dropped bits.

E. Tests for Hashing Rate

The hash rate of our system will be the clearest indicator of the performance of our system. There will be a rigorous process to determine the average hash rate for each cryptocurrency being mined. First, we will generate randomized inputs through the use of SystemVerilog testbenches to simulate the execution of the hashing module. This will be allowed to run for 5 minutes such that the number has time to converge. While the testbench is running, it will output the current calculated hash rate every 30 seconds. We will use a software tool developed by Synopsys called VCS for simulation.

After retrieving the simulation hash rate, we will also need to synthesize the design onto an FPGA to test the design on a physical board. Synthesis will be done using a program called Quartus. Once the design is on the FPGA, it will be connected to the Raspberry Pi which will generate the randomized inputs for testing. Similar to before, we are allowing it to run for 5 minutes for convergence. The FPGA will output the number of completed hashes back to the Raspberry Pi where it will be displayed every 30 seconds.

If there is a divergence between the simulation numbers and the synthesis numbers, we are inclined to lean more towards the synthesis hash rate because it reflects how the design performed on a physical board.

F. Tests for Critical Path and Latency

When synthesizing our design, the configuration file takes in a clock period as an argument. It then generates several reports, one of which being the timing report for our design. We know that the DE0-CV FPGA supports a 50 MHz clock. Thus, by providing this as our clock period argument, we can see if our design will meet this timing requirement. The timing report will indicate whether our design's latency fits within this period along with the critical path of our design. Although the gates and flip flops are not always descriptively named, it can still be deciphered to determine whether the critical path of our system lies within the hashing module.

VIII. Project Management

A. Schedule

By the middle of the week 3/28, we hope to have achieved a basic miner that's able to mine cryptocurrency and show the communication between each component, such as between RPi and the web application, FPGA and the RPi, and mining pool and the RPi. This will serve as the basic framework when we want to scale up our project in the future. Our goal is to finish the project in the week of 4/18, approximately a week in advance. For the detailed schedule, see page 9 for the complete Gantt Chart.

B. Team Member Responsibilities

We split the responsibilities for each team member based on their strengths and courses taken. Lulu will primarily work on the web application and helps with implementing the machine learning algorithm that will do the coin choosing. David will be establishing the communication between the mining pool and the RPi. William will be writing modules that handle the communication between RPi and FPGA and the hashing modules on the FPGAs.

C. Bill of Materials and Budget

See page 10 for the bill of materials for the project.

D. Risk Mitigation Plans

Our most significant risk right now is allocating time for integration. Since the RPis came in later than we expected, we were unable to do tasks and testing related to the RPi. This includes implementing communication protocol between the RPi and mining pool, and testing communication between the FPGAs and the RPi. In an attempt to mitigate the risk, David and William are going to each to take an RPi to work on these tasks during spring break in order to meet our deadline for the interim demo.

In terms of potential design risks, we are concerned about how we are going to compare the hashing rate against the GPU miner. Since one of our design metrics requires our hashing rate to reach at least 90% of GPU's hashing rate, it's important for us to get an accurate hashrate number to ensure we meet our design requirement. We decided on 5 Mh/s for the GPU's hashrate because it was the most reasonable number that we found.

IX. Related Work

There is a senior project from California Polytechnic State University's electrical engineering department that also used an FPGA for bitcoin mining. But the miner is only mining bitcoin and it doesn't have the self choosing algorithm that we hope to implement. This project mainly focused on learning the mining algorithm and the hashing algorithm, SHA-256, and ways to improve the miner's hash rate. [1]

Another paper that we looked at was from Lebanese University, where the main goal of their project was to implement communication between an FPGA and RPi using SPI protocol. This is similar to what we are trying to achieve with our RPi, as we want to be able to control multiple FPGAs at the same time using SPI protocol. [2]

X. Summary

We hope to create an FPGA miner that's flexible and cost friendly in comparison to the GPU and ASICS miner. Most importantly, the miner will be able to choose the optimal ratio using a machine learning algorithm to mine different cryptocurrencies in order to achieve maximum profit. Users can make profit without analyzing the trend of cryptocurrencies. The upcoming challenge in the implementation will be keeping the communication overhead between each component to a minimum so that the miner can reflect the changes in cryptocurrency prices in real time.

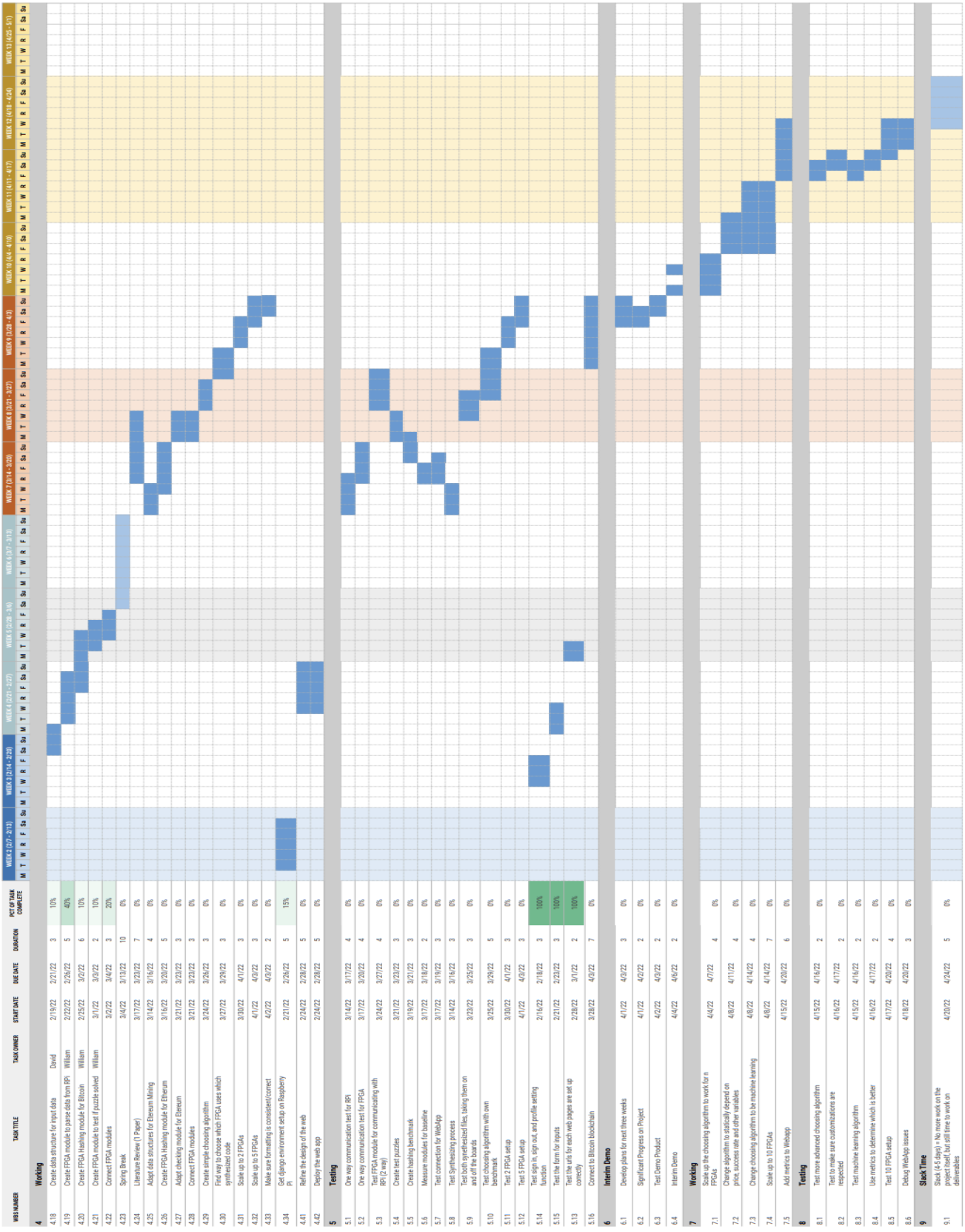
Glossary of Acronyms

API – Application Programming Interface
ASIC – Application-Specific Integrated Circuit
CPU – Central Processing Unit
FPGA – Field Programmable Gate Arrays
GPU – Graphics Processing Unit
RPi – Raspberry Pi
SBC – Single Board Computer
SHA-256 – Secure Hash Algorithm 256
SPI – Serial Peripheral Interface

References

[1] Dotemoto, P. (2014, June). *FPGA based Bitcoin mining*. DigitalCommons@CalPoly. Accessed on March 4, 2022. Available: https://digitalcommons.calpoly.edu/eesp/268

[2] Hajjar, H., & Mourad, H. (2019). *Implementation of an FPGA - Raspberry Pi SPI Connection*. ThinkMind(TM) Digital Library. Accessed on March 4, 2022. Available: https://www.thinkmind.org/articles/cenics_2019_1_20_50029.pdf



WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
4	**Working**					
4.18	Create data structure for input data	David	2/19/22	2/21/22	3	10%
4.19	Create FPGA module to parse data from RPi	William	2/22/22	2/26/22	5	40%
4.20	Create FPGA Hashing module for Bitcoin	William	2/25/22	3/2/22	6	10%
4.21	Create FPGA module to test if puzzle solved	William	3/1/22	3/3/22	2	10%
4.22	Connect FPGA modules		3/2/22	3/4/22	3	20%
4.23	Spring Break		3/4/22	3/13/22	10	0%
4.24	Literature Review (1 Paper)		3/17/22	3/23/22	7	0%
4.25	Adapt data structures for Ebersum Mining		3/14/22	3/16/22	4	0%
4.26	Create FPGA Hashing module for Ethersum		3/16/22	3/20/22	5	0%
4.27	Adapt checking module for Ebersum		3/21/22	3/23/22	3	0%
4.28	Connect FPGA modules		3/21/22	3/23/22	3	0%
4.29	Create simple choosing algorithm		3/24/22	3/26/22	3	0%
4.30	Find way to chose which FPGA uses which synthesized code		3/27/22	3/29/22	3	0%
4.31	Scale up to 2 FPGAs		3/30/22	4/1/22	3	0%
4.32	Scale up to 5 FPGAs		4/1/22	4/3/22	3	0%
4.33	Make sure formatting is consistent/correct		4/2/22	4/3/22	2	0%
4.34	Get django environment setup on Raspberry Pi		2/21/22	2/26/22	5	15%
4.41	Refine the design of the web		2/24/22	2/28/22	5	0%
4.42	Deploy the web app		2/24/22	2/28/22	5	0%
5	**Testing**					
5.1	One way communication test for RPi		3/14/22	3/17/22	4	0%
5.2	One way communication test for FPGA		3/17/22	3/20/22	4	0%
5.3	Test FPGA module for communicating with RPi (2 way)		3/24/22	3/27/22	4	0%
5.4	Create test puzzles		3/21/22	3/23/22	3	0%
5.5	Create hashing benchmark		3/19/22	3/21/22	3	0%
5.6	Measure modules for baseline		3/17/22	3/18/22	2	0%
5.7	Test connection for WebApp		3/17/22	3/19/22	3	0%
5.8	Test Synthesizing process		3/14/22	3/16/22	3	0%
5.9	Test both synethesized files, taking them on and off the boards		3/23/22	3/25/22	3	0%
5.10	Test choosing algorithm with own benchmark		3/25/22	3/29/22	5	0%
5.11	Test 2 FPGA setup		3/30/22	4/1/22	3	0%
5.12	Test 5 FPGA setup		4/1/22	4/3/22	3	0%
5.14	Test sign in, sign out, and profile setting function		2/16/22	2/18/22	3	100%
5.15	Test the form for inputs		2/21/22	2/23/22	3	100%
5.13	Test the urls for each web pages are set up correctly		2/28/22	3/1/22	2	100%
5.16	Connect to Bitcoin blockchain		3/28/22	4/3/22	7	0%
6	**Interim Demo**					
6.1	Develop plans for next three weeks		4/1/22	4/3/22	3	0%
6.2	Significant Progress on Project		4/1/23	4/2/23	2	0%
6.3	Test Demo Product		4/2/22	4/3/22	2	0%
6.4	Interim Demo		4/4/22	4/6/22	2	0%
7	**Working**					
7.1	Scale up the choosing algorithm to work for n FPGAs		4/4/22	4/7/22	4	0%
7.2	Change algorithm to statically depend on price, success rate and other variables		4/8/22	4/11/22	4	0%
7.3	Change choosing algorithm to be machine learning		4/8/22	4/14/22	4	0%
7.4	Scale up to 10 FPGAs		4/8/22	4/14/22	7	0%
7.5	Add metrics to Webapp		4/15/22	4/20/22	6	0%
8	**Testing**					
8.1	Test more advanced choosing algorithm		4/15/22	4/16/22	2	0%
8.2	Test to make sure customizations are respected		4/16/22	4/17/22	2	0%
8.3	Test machine learning algorithm		4/15/22	4/16/22	2	0%
8.4	Use metrics to determine which is better		4/16/22	4/17/22	2	0%
8.5	Test 10 FPGA setup		4/17/22	4/20/22	4	0%
8.6	Debug WebApp issues		4/18/22	4/20/22	3	0%
9	**Slack Time**					
9.1	Slack (4-5 days) + No more work on the project itself, but still time to work on deliverables		4/20/22	4/24/22	5	0%

Figure 3: Gantt Chart

Table 1: Bill of Materials

Item	**Description**	**Model #**	**Manufacturer**	**Quantity**	**Cost**
FPGA	Cyclone V 5CEBA4F23C7N Device, 49K Programmable Logic Elements, 3080 Kbits embedded memory, 4 Fractional PLLs	DE0-CV	Terasic Inc.	10	$0
RPi 4	1.5 GHz 64-bit quad core ARM Cortex-A72 processor, on-board 802.11ac Wi-Fi, Bluetooth 5, full gigabit Ethernet, two USB 2.0 ports, two USB 3.0 ports, 2–8 GB of RAM	B	Raspberry Pi Foundation	1	$167.48
Pi Wedge	Adapts the GPIO header on the RPi to a standard solderless breadboard	None	SparkFun	1	$20.85
Breadboard	Breadboard Kit Solderless, Large	None	Digilent	1	$0
Jump Wires	Male to Female 4 and 8 Inch Solderless Ribbon Dupont-Compatible Jumper Wires	None	GenBasic	1	$13.56
Total Cost					$201.89