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 with performance measured in the number of hashes comparable to that of conventional methods while maintaining the benefit of flexibility. 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, decision tree, 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 were able to use standard laptops and consumer-grade computers, current miners must adopt specialized hardware such as ASICs 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, an application-specific integrated circuit or ASIC miner targets a specific cryptocurrency running a specific hashing algorithm. These miners are not customizable and once a design is chosen, it cannot be reprogrammed. On the other hand, GPUs are more generalized than ASICS and can switch from mining one coin to another relatively quickly.

Our solution aims to be a fully customizable miner that follows both interday and intraday price action in order to select which cryptocurrency is best to mine. This approach no longer suffers from the rigidness of ASICs while maintaining a respectable hash rate comparable to that of a GPU. CryptoHash is targeting both enterprise and consumer users by designing a system that works with as little as one FPGA or can be scaled to above ten. To support a one-in-all solution for cryptocurrency mining, 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 metrics relevant to the user such as the hash rate and current prices of mined cryptocurrencies. 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. In order for the system to choose the optimal way to distribute compute resources, the choosing algorithm will need to examine all cryptocurrency pricing data from the preceding three months. We chose three months as the target time frame because decisions made in the present should be weighted more heavily towards 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 it, 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. Consumer-grade 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 the added benefit of automatic configuration switching.

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.

One of the driving forces of cryptocurrency mining is the opportunity to generate an income from it. Users will not adopt our system if it results in a net loss every month. Thus, after factoring in energy costs, the system must be able to turn a profit.

III. Architecture and/or Principle of Operation

Our Raspberry Pi pulls the latest pricing data for our targeted cryptocurrencies using an API provided by cryptocompare which provides more accurate pricing data and other current information about each coin's blockchain. This data is then inputted into our custom machine learning model which decides the optimal percentage of computing power to allocate to each cryptocurrency. The backbone of the inference is the decision tree that is trained on historical data. The output of the decision tree serves as an input to the recurrent tendencies that dictate the spread of computing power. Other factors such as number of transactions and changes in difficulty will affect the tendency numbers before comparing against threshold values to get the ultimate spread. The historical data is obtained from Binance's API because it provides the historical data for free and allows the 10 second granularity that we desire. The entire decision making process takes 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.

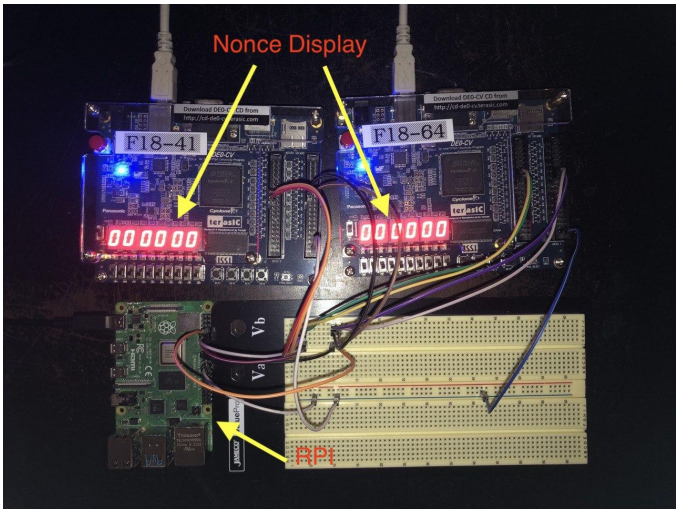


Fig. 1. Annotated Photo the system running with two FPGAs

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 communicate this spread to the FPGAs as a sort of FPGA controller, dictating which FPGAs need to change and to which mining configuration. The changes in spread are manipulated such that each change will only change one board, both to allow margin of error for the predicting process and as a restriction directly imposed by the Quartus programmer. The margin of error exists so that if the spread was changed due to a short period of time; in this scenario, only one board would suffer the overhead of switching and the loss of work done. Additionally, the Quartus programmer is unable to parallelize putting different configurations at the same time even as different processes. The communication software has another aspect where it 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 work from these pools and send this data to the FPGAs by serializing the data and sending through the aforementioned GPIO pins.

The entire system communicates using the SPI protocol which requires four wires to implement. The four wires are MOSI, where the inputs to the FPGAs will be sent , MISO, where the inputs to the Raspberry Pi will be sent, SCLK, where the clock from the Raspberry Pi is sent, and the select lines to help control the FPGAs. The Raspberry Pi software will generate clock transitions and assert select wires in accordance with this protocol. The communication software is primarily done through the spidev python library. This places a limit on the clock speed that the FPGAs ultimately can run at since the Raspberry Pi can only utilize the SPI protocol at even divisions of 125 MHz.

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.

With all this said, many things have changed from the Design Report. Most of the modes of communication we had originally planned to use were not going to work whether it was because the Raspberry Pi runs on ARM and our FPGA boards only run on Intel processors or how the nonces are sent back to the Raspberry Pi. The entire system was supposed to be stand-alone, but due to the processor dependency, it had to be connected to a separate machine to program configurations. Some of the external factors such as relative strength index never made its way into the decision making process. The APIs we used for the historical data and current data had to change because Binance's API was not providing accurate information for Ethereum despite having a valid API key.

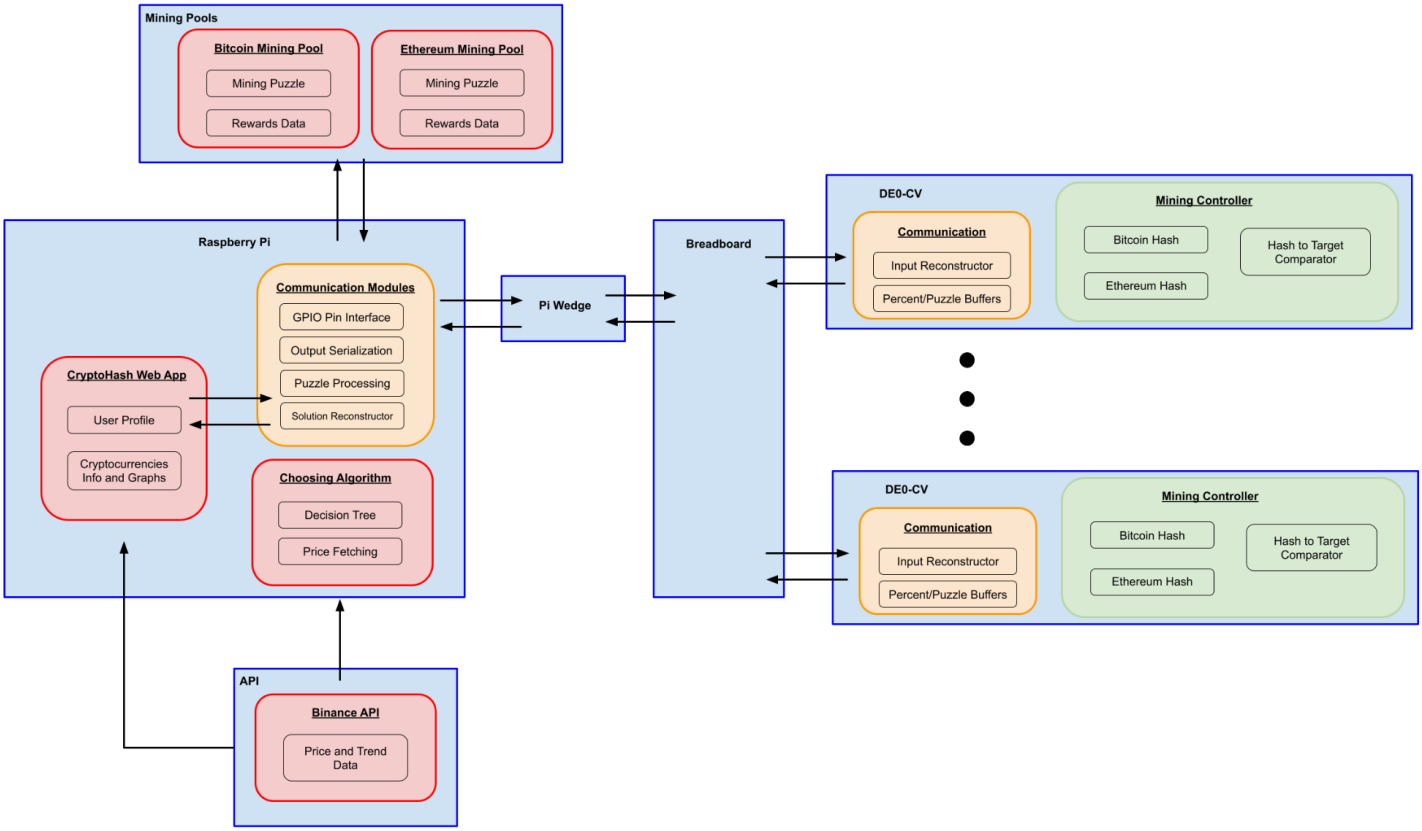


Fig. 2. Block drawing of overall system, illustrating the dataflow between various subsystem

IV. Design Requirements

The web app must be responsive to user inputs. The latency 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 less than 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 change in prices. Thus, the data fetching must happen, at the latest, every minute to ensure our system remains well-tuned to current events. 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 single board computer (SBC) which in turn broadcasts it 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 should elapse. 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 six 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 that data could become corrupted. Our system must be resilient enough such that 95% of all bits in 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 includes 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 ones between profitability and the supported coins and model complexity and accuracy. These trade-offs influence system level complexity and communication complexity. This also encapsulates the trade-offs made between hash rate and the reduced tendency to invoke switching overheads, which we will call robustness.

A. FPGA Trade-Offs

At the very core of our design, we want to meet the performance requirement of our system while still allowing for each miner to be customized. We recognize that FPGAs will not be able to achieve the same hash-rates as specialized hardware, yet we choose to use them anyways because they can be reprogrammed quickly. FPGAs are a good candidate for satisfying our requirement of a design being able to switch under 10 seconds. To close the gap between achievable hash rates of FPGAs versus ASICs, we use multiple boards to 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 when 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.

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 seeing as 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 for it was more established than for the Rock Pi and that we wanted the faster 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 Cryptocurrencies Trade-Offs

The objective 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, with 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. Thus, adding support for a coin means needing to have configurations for that algorithm implemented and compiled in the FPGA. 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 two most popular cryptocurrencies. While maximizing profit 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 greater than just making the most efficient miner that makes as much money as possible. Our focus includes that, but is also constrained under the selection of the coins. As well established coins, the protocols of how Bitcoin and Ethereum 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 selected had to run very quickly and without much computational power because we don't want the entire system to 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 low so

that computing power for the communication is not compromised. The training process for a decision tree also does not take very long either since there aren'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 containing three months of data. The largest drawback that decision trees have is that they're usually not as accurate when compared to something like a Deep Neural Network. The neural networks usually pay for this accuracy with the longer 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 three 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 one 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.

E. Number of Board Trade-Offs

Due to the number of logic elements present in the DE0-CV boards, and the innate need of multiple boards to switch between, our project utilizes multiple FPGAs. The innate need lies in the fact that the proof of work mining process is deterministic in the nonces that it cycles through to determine a correct nonce; switching configurations would require the process to start over from its programmed beginning and saving where the board left off of is unnecessary added complexity that may not be relevant if the mining puzzle is now different. We were able to avoid the bottleneck of the limited number of GPIO pins with the Pi Wedge, but connecting a large number of FPGA boards to the Pi Wedge still had a space bottleneck. We requested a total of 10 FPGAs from the ECE inventory, so we tested configurations out based on our supply as well.

The setups we tested were two boards, six boards, and ten boards. Our ultimate goal at the beginning was to utilize ten boards at the same time to produce our desired hash rate. The two board setup would be our base case where we would ultimately do most of our testing. The six board setup was decided seeing as six is the midpoint between two and ten. With more boards, there comes increased hash rates because there is simply more processing power and more logic elements. However, more boards also means that there is more communication overhead and the thresholds for the spread that the decision making process outputs are closer together. If the thresholds are closer together, then there will be more switching that happens resulting from very temporary gains. This results in an overall decrease in the robustness of the entire system where a few predictions of one coin can waste time and resources switching from one configuration back to itself, ultimately having to start from scratch for the proof of work process. This switching overhead is compounded by the time necessary for Quartus to load the mining configuration onto the board and wait for a reset signal to begin mining again.

From testing with the three different number of boards, we found that having fewer boards in the system allowed for a lot of robustness, but the hashrate for either coin was poor given our unoptimized hashing algorithms and that FPGAs are not streamlined for hashing like ASICs can be. The opposite results were observed when testing the system on ten boards, where the overall system, even when it entered the phase where it heavily favors one coin, was experiencing a few bouts where it decided to mine the other coin for one or two iterations. Having ten FPGA boards was able to significantly multiply the hash rate. The six board configuration was the setup we decided to use in the final video and demo since it was able to strike a balance while also lessening the load on the shared communication buses.

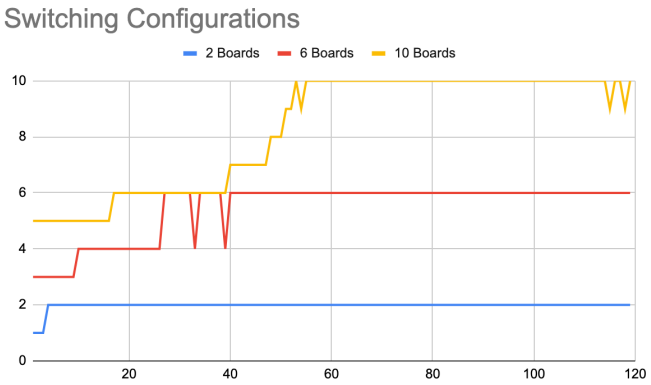


Fig. 3. Chart depicting the switching for different board configurations

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 Binance's API.

We will also be displaying the current spread ratio the miners are using based on the decision from our choosing algorithm and the revenue made 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.

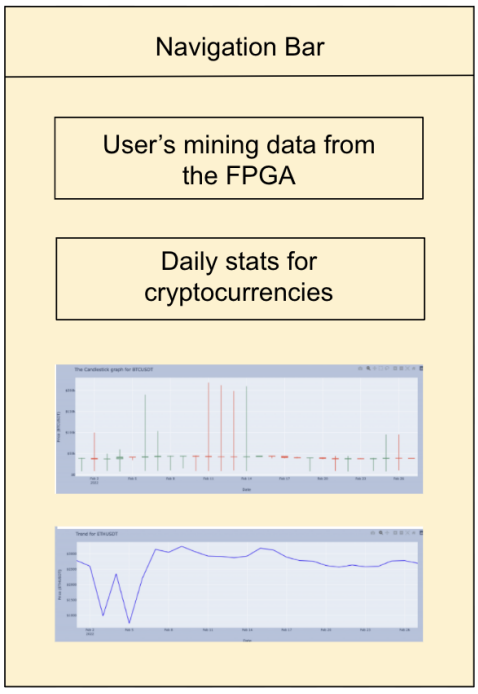


Fig. 4. Diagram of Screenshot of Web Application

B. *Raspberry Pi and Mining Pool Communication*

Acting as the communication center, the Raspberry Pi needs to communicate with the mining pool for each 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 alone would be much higher, we decided not to attempt this so that we do not have to compete with miners sporting 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. This topic will be revisited in the future work section.

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 allows it to receive new puzzles when the puzzle changes for the currency.

The communication between the Raspberry Pi and mining pools is most commonly encapsulated in the Stratum TCP protocol that runs on JSON RPC. The mining client on the Raspberry Pi has threads that house miners for each currency each of which connect to the respective mining pools and communicate with the appropriate responses. The replies and parameters from the mining pools are also different based on the coins and are treated as such in the respective miners.

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 three month's worth of historical data from Binance's API. It will perform inference, and thus switch between cryptocurrencies, conducting inference on a ten second basis, which does not equate to switching between configurations every ten seconds, but checking to switch every ten seconds. The model takes as input the current price of the cryptocurrencies, the current price trend for the past ten minutes, and the recent rewards from the different mining pools, among other parameters. The model will have a greedy algorithm that will bias the model to want to mine more of the cryptocurrency that has higher payouts. 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 fundamental 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 three months of historical data from Binance so that the decision tree can keep up with the most recent data.. Throughout the month, the model will run inference every ten seconds 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 ten 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 and FPGA Communication

The system uses 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.

The Raspberry Pi uses four GPIO pins to output to the FPGAS. The first is a data wire that sends the mining pool puzzles bit by bit. Next is a data ready wire that is asserted when there is valid data being sent. The last two wires send the clock signal and the reset signal respectively. The Raspberry Pi clock operates at a higher frequency than the FPGAs so we need to sync both components to a common clock signal. One issue we ran into was having the Raspberry Pi use a clock signal sent from the FPGA. The SPI protocol requires specific clock frequencies so this approach did not work. One of the ways to fix the issue was instead of having the FPGA send in the clock, the Pi would be the one that sends in the clock signal. There will also be an asynchronous first in first out (FIFO) buffer that stores the nonces that need to be sent into the Pi. The nonce will be stored into the buffer with the FPGA clock signal. When the Pi is requesting a nonce, it will read from the buffer using the Pi's clock so that it is outputting at a rate that the Pi can process. In the end, the design was too big and it wasn't able to fit on the DE0-CV board that we choose when compiling on quartus. In the future, we can choose a bigger board or refine our whole design so that it can fit on the board

After the choosing algorithm on the Raspberry Pi determines that a configuration switch is necessary, the FPGA must receive a reset signal in order to initialize internal buffers and begin mining. All cryptocurrency configurations on the FPGAs will be expecting a single type of data packet from the Raspberry Pi, namely a PUZZLE packet. This contains an eight bit header and the proof of work puzzle received from the mining pool. The header has five bits which indicates the FPGA the packet information is meant for, and three bits for the cryptocurrency the puzzle is for. 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 the Ethereum block header is 64 bytes. This puzzle is stored in the Raspberry Pi and serialized in order to transmit as a single bit through the GPIO pins.

When an FPGA finds a valid nonce from mining, it couples this nonce with an eight bit header to send to the Raspberry Pi. Similar to the previous header, it has five bits for the FPGA the packet information came from, and three bits for the cryptocurrency the nonce is for. Bitcoin uses a 32 bit nonce while Ethereum has a 64 bit nonce. The Raspberry Pi will need to be able to parse the input packet and determine how to submit this nonce to the mining pool.

E. FPGA Hashing Module

We designed custom FPGA mining controllers to handle the puzzle input and distribute work to the numerous hashing modules. For Bitcoin mining, there are eight of these hashing modules. Each of them takes in the 80 byte block header and replaces the last 32 bits with a 32 bit nonce. Hashing module one starts with a nonce value of 0x0, module two starts at 0x5555555, module three at 0xAAAAAAA, and etc.

Bitcoin uses SHA-256 as its hashing algorithm which we implemented in SystemVerilog. The block header with the replaced nonce is inputted into SHA-256 which pads the data such that it is a multiple of 512 bits. The output hash of this first round is inputted to a second SHA-256 module which performs a hash of a hash. The hashing module takes this second hash and compares it with a hash target.

We now describe how this hash target is calculated. Each Bitcoin block header contains a 32 bit difficulty field. The first 24 bits of this field is left shifted by the value of the last 8 bits of this field. This creates a 256 bit hash target to compare against. If the output of the second hash from SHA-256 is above this target, the nonce used for the block header is incremented by one and the process repeats. If, however, the second hash is below the target, this means that the nonce is valid and the FPGA sends this nonce to the Raspberry Pi. With 2^{32} and 2^{64} number of nonces to try for Bitcoin and Ethereum respectively, this hashing process represents the proof of work process of mining.

Ethereum uses a completely different hashing process to find a valid nonce. One of the fields inputted in the 64 byte block header is the current block number. The FPGA mining controller takes the block number and generates a seed from it by hashing 0x0 with Keccak-256. The output of this is hashed again with Keccak-256 in a process whose length is dependent on the block number. This seed is then used to create a cache of random values. The first value of the cache is the Keccak-512 hash of the seed with subsequent values of the cache being the Keccak-512 hash of the previous value.

Once the cache is filled, the hashing module then begins the proof of work hashing process. The block header, a 64 bit nonce, and the cache is combined and hashed together. Due to the relatively obscure nature of Keccak, we were unable to find good documentation outlining its hashing process. Thus, we referenced an already implemented SystemVerilog version and used that in our Ethereum mining controller.

As soon as either the Bitcoin or Ethereum hashing modules finds a valid nonce, it will communicate this back to their respective mining controllers. The controller will take the 32

bit or 64 bit nonce and serialize it in order to send to the Raspberry Pi.

VII. Test, Verification and Validation

A. *Results 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 Binance's 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. *Results 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 previous requirement was set to 70% which is better than the 50% from randomly guessing. This accuracy number was constructed with the understanding that it 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. However, as our project and our understanding of our project's fundamental goals developed, we developed a systematic process to determine a metric for the accuracy for our decision making process.

To test the accuracy of the algorithm, the system was run for 200 iterations where the pricing data for each coin and the spread output were logged. Without looking at the predicted spread and only looking at the pricing changes, an ideal switching metric is produced. This testing process simplifies the switching process to only consider the price change and the price trend but with the added benefit of being able to consider the changes in hindsight. After multiple iterations of testing on a two board configuration, which is the simplest configuration to reason about accuracy, the best run is depicted below with an accuracy of 85.5%. This allows our decision making process to boast at least 85% accuracy.

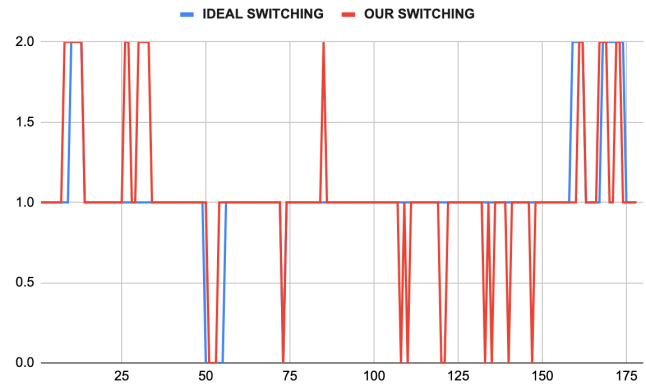


Fig. 5. Graph depicting differences between our spread and an ideal one

C. *Results for Configuration Switching Overhead*

One of our requirements was to be able to switch FPGA configurations within ten seconds of the command being inputted. We tested for this by having the choosing algorithm run for a period of time based on pre-trained data. It generated a set of spreads that was known ahead of time to a system of six FPGAs. Once the choosing algorithm indicated that a switch was necessary, the Python code that loads the configuration file started a Python timer. It then instructed Quartus Programmer to load the configuration and idled until this task was completed. When Quartus finished, the timer stopped and we recorded this result for multiple runs. For a single board, we had an average configuration switching time of 6.38 seconds. For multiple boards, this number grew proportionally with the number of boards. Even still, we considered the requirement to be satisfied because the choosing algorithm will only command one FPGA board to switch at a time.

D. *Results for Packet Bit Errors*

Although our system uses two-way communication between the Raspberry Pi and the FPGAs, we decided to test the FPGA to Raspberry Pi packet drop rate and extrapolate this to two-way communication. On the FPGA, we designed a module that would send a zero bit on one clock cycle, a one bit on the next clock cycle and repeat this process. The Raspberry Pi took in this bitstream and recorded any instances when it observed two zeroes or two ones in a row. There were issues performing this test because the communication software was not completed until the very late stages of the project. Thus, the tests we performed both before and after the software was completed gave mixed results. Before the issues were resolved, our system repeated one bit every eight bits for a resiliency of 87%. After we implemented a fix for the communication issues, we still observed some packet bit errors but with an improvement to a 93% error rate.

E. Results for Hash Rate

The hash rate of our system is the clearest indicator of the performance of our system. To test for the Bitcoin hash rate, we found the Bitcoin genesis block online which specified the 80 byte block header and the nonce used to find a valid hash. This information was hard coded onto an FPGA running our Bitcoin mining controller instead of receiving the puzzle input from the Raspberry Pi. The Raspberry Pi sent a reset signal to the FPGA and started a timer when it did so. As soon as the FPGA found the known valid nonce, it sent this back to the Raspberry Pi which then stopped the timer.

Using this testing methodology, we recorded a hash rate of 3.8 kH/s for a Bitcoin mining controller with one hashing module and a hash rate of 3.04 MH/s with eight hashing modules. With a system of six FPGAs, we had a hash rate of 18.24 MH/s.

F. Results for Profitability

Because we weren't able to successfully mine any cryptocurrencies during the testing process, our profitability results rely on what our potential earnings could have been. Using the total hash rate of our system, we take it as a percentage of the global hash rate and multiply this figure by the mining rewards and the current cryptocurrency price. Bitcoin has a mining reward of 6.25 Bitcoin and a block time of around ten minutes which means the rewards are issued every ten minutes. Ethereum mining rewards are variable and the average reward is four Ethereum every 13 seconds.

From this, we calculated our expected returns to be $\$5.67 \times 10^{-11}$ per second mining Bitcoin. Using Synopsys, we found that the power consumption of each FPGA board mining Bitcoin was 24.8 mW. After subtracting the energy costs from our revenue, each board has a loss of $\$6.57 \times 10^{-9}$ per second.

VIII. Project Management

A. Schedule

See page 12 for the schedule.

B. Team Member Responsibilities

We split the responsibilities for each team member based on their strengths and courses taken. Lulu primarily worked on the web application and a portion of the communication between the Pi and the FPGAs on the Pi side, specifically only the sending and receiving bitstream functions. David worked on the communication between the mining pool and the RPi and the FPGAs, which includes creating modules to communicate both ways with the mining pools and parsing the relevant information to then send to the FPGAs. He also worked on implementing and testing the choosing algorithm including acquiring and processing all of the data that it requires. William wrote the SystemVerilog modules that handled the communication between the RPi and FPGA on the FPGA side and the hashing modules on the FPGAs. For communication, this meant writing modules that reconstructed the proof of work puzzle from the input bitstream along with serializing the valid nonce found and sending that to the Raspberry Pi. He also researched the mining process for Bitcoin and Ethereum and wrote both mining controllers for the two cryptocurrencies. In addition, William and David worked together on the configuration switch on the FPGAs using a remote server in addition to finding solutions to all the various problems that arose and conducting all of the necessary research.

C. Bill of Materials and Budget

See page 13 for the bill of materials for the project.

D. Risk Management

In terms of the potential design risks, we had the concern that during the communication between the Pi and the FPGAs we will drop bits or read extra bits which we did observe when testing the communication modules. This was due to the difference in clock frequency between the Pi and the FPGAs. After consulting with a TA, we were able to implement an asynchronous first in first out buffer that solves the issues. Another design risk that we encountered was the parameters, such as cryptocurrencies prices, used for training the choosing algorithm was imported incorrectly. This was due to the Ethereum prices from Binance's API not being updated. After pulling data from another API we were able to solve the issue.

Midway through the project, we realized that the idea we had for configuration switching used a lot of wasted hardware space. Our initial design placed both the Bitcoin and Ethereum hashing modules on the same FPGA and enabled either one depending on the current configuration. After realizing the inefficiencies of this, we pivoted to using Quartus Programmer to load different configurations onto the FPGAs externally. One issue that arose was the fact that Quartus is an Intel program that only runs on x86 based computers while the RPI runs on an ARM CPU. Luckily, we were able to use a VM that already had Quartus installed to reprogram the FPGAs with configuration files.

In terms of schedule, we were behind most of the time. In the beginning of the semester, since our delivery for the Pi came late we were behind on tasks related to the Pi. We were able to catch up by working on them during the spring break. After updating the schedule for the design review report, we tried to make sure we had enough time in the end for integration. In the end, since our communication subsystem still had bugs we didn't integrate the system until the last week of the class. We made time during final's week to try to fix the issues.

In terms of budget, we didn't face many constraints since we got our FPGAs from inventory and we only needed a Raspberry Pi which was the most expensive part we bought for our project.

IX. Ethical Issues

Our intended audience is anyone who wants to make money from mining cryptocurrencies without any prior knowledge in

this field. Since our product will choose the optimal spread for mining different cryptocurrencies, the users only need to power up the devices then they can start making money. Mining requires usage of the internet and powerful hardware that are able to do the computations fast enough in order to make a profit. Hence the main ethical issue for our product is environmental impact. Running these hardwares requires enormous amounts of power, which has the potential to negatively impact our environment such as global warming. Although our project uses the FPGAs as our main hardware, which does not consume as much energy as ASICs, energy consumed can still be enormous if the user uses multiple devices at the same time, or enough users use our device at the same time.

Other possible edge cases of harm can include wires wearing down or the Pi overheating which can result in an electric hazard. The user would be adversely affected in these situations. One of the approaches to mitigate these adverse impacts is to have an emergency stop button that stops the whole system when the user realizes the system is overheating. Another approach is to have the system shutdown and sleep for a period after running for a long time to avoid the overheating situation. The best way to induce a sleep period would be to implement a thermistor temperature sensor that would notify the system to shut down once the temperature exceeded a certain threshold and sleep until the temperature of the system reached a safe minimum.

X. 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]

XI. Summary

Our system was able to meet most of our design requirements but failed to generate a profit. After amortizing the initial setup time, the FPGAs maintained less than a 10 second configuration switching time and performed quite well with a hash rate of 18.24 Mh/s for a six board system. Although this exceeded our requirement of being competitive with GPUs, the power consumption of the system was such that the costs exceeded the revenue. Our choosing algorithm performed quite well with results indicating that it could predict the next best coin with an accuracy of up to 85%.

A. Future work

One improvement that we can make is to adjust the Bitcoin hashing algorithm such that it reduces the number of SHA-256 hashes needed. When hashing the 80 byte block header, the data size is greater than the 512 bit input so the FPGA has to hash the first half followed by the second half. However, the nonce resides in the second half of the block header and thus, the hash of the first half does not change. An improvement that would greatly improve our hash rate would be to calculate the hash of the first half of the block header and save this for reuse without needing to hash it again.

There is a lot of potential that exists in terms of the coins that can be used for the project. The number of coins can be scaled up both in terms of the number of coins that can be supported for mining and the number of coins that can be switched around. Adding support for more coins that can be mined is just a matter of creating the mining configuration for the FPGA and adding the module to connect to the respective mining pool. Adding support to switch between more coins is a slightly more tricky issue in changing the decision making process both in how it is trained and how the spread thresholds are calculated. Another improvement is to have the entire mining system connect to the blockchains of the coins themselves to mine coins without the middle-man of a mining pool. Mining pools are very helpful for very popular coins such as Bitcoin and Ethereum where the probability of mining those coins as a solo miner is negligible. However, there are a plethora of other altcoins that can be mined where having complete ownership of the coins is a desirable characteristic. This also can remove one of the costs that decreases the overall revenue of the system: mining pool fees. Each mining pool will charge the relevant miners a fee after the miner has obtained a certain amount of rewards; mining solo can avoid this surcharge.

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
VM - Virtual Machine

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

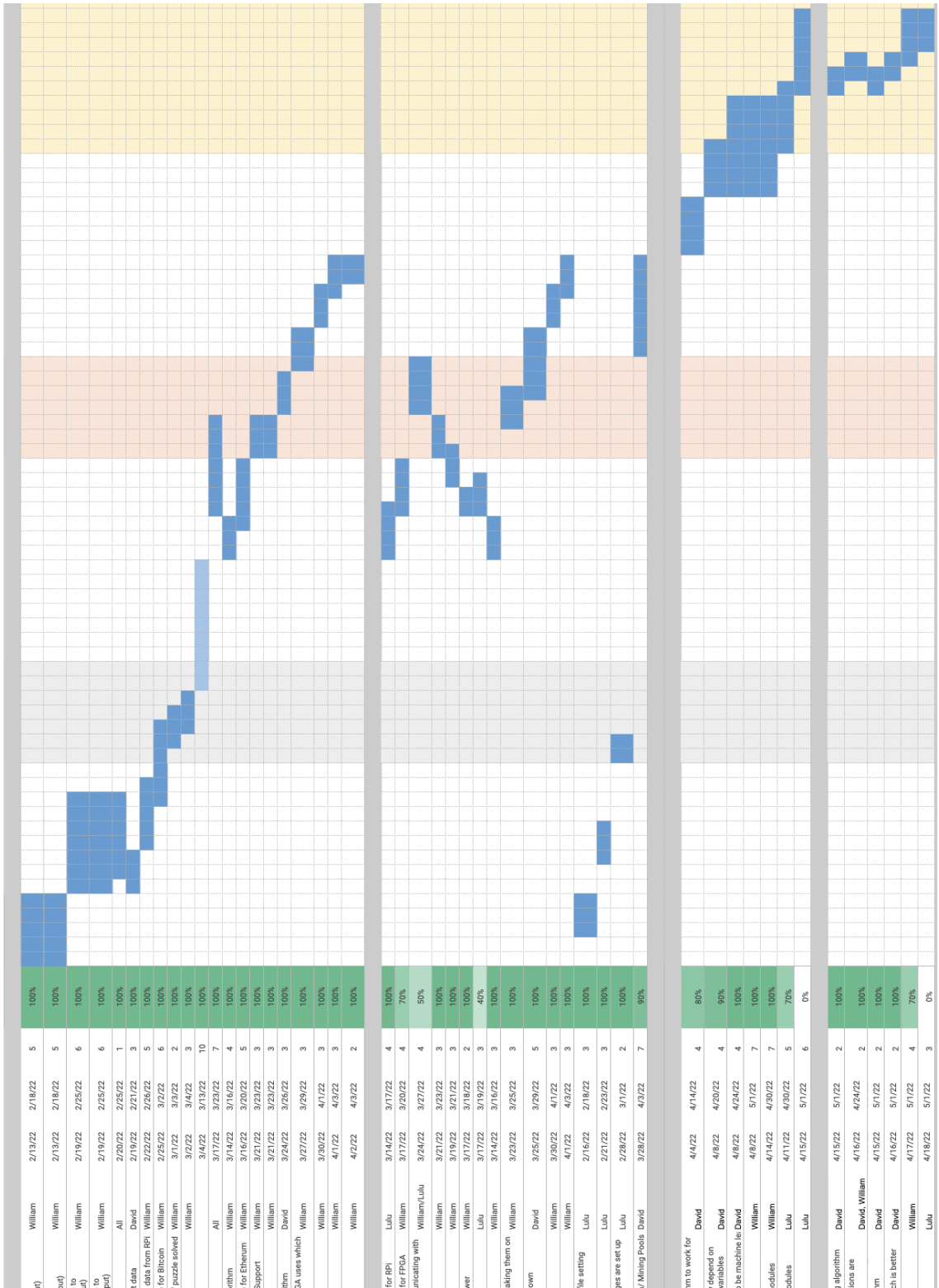
[3] Moles, Josh. "A Verilog (Specifically, System Verilog) Implementation of the Not-Yet-Finalized SHA-3 Winner, Keccak." *Keccak-Verilog*, Github, 12 Apr. 2021, https://github.com/jmoles/keccak-verilog.

[4] Ward, Chris. "Ethash." *Ethash*, Ethereum Wiki, 11 June 2020, https://eth.wiki/en/concepts/ethash/ethash.

[5] *Cryptocurrency API, historical & real-time Market Data*. CryptoCompare. (n.d.). Retrieved May 7, 2022, from https://min-api.cryptocompare.com/documentation?key=Price&cat=SingleSymbolPriceEndpoint

[6] Binance. (2022, April 26). *Binance-spot-API-docs/rest-api.md at master · Binance/Binance-spot-API-docs*. GitHub. Retrieved May 7, 2022, from https://github.com/binance/binance-spot-api-docs/blob/master/rest-api.md

[7] Ricmoo. (n.d.). *Ricmoo/nightminer: Simple python cryptocurrency mining client*. GitHub. Retrieved May 7, 2022, from https://github.com/ricmoo/nightminer



4	**Working**						
4.8	Implement FPGA module for communicating with RPi (input)	William	2/13/22	2/18/22	5	100%	
4.9	Implement FPGA module for communicating with RPi (output)	William	2/13/22	2/18/22	5	100%	
4.10	Outline Raspberry Pi software to communicate with FPGA (input)	William	2/19/22	2/25/22	6	100%	
4.11	Outline Raspberry Pi software to communicate with FPGA (output)	William	2/19/22	2/25/22	6	100%	
4.17	Literature Review (1 Paper)	All	2/20/22	2/25/22	1	100%	
4.18	Create data structure for input data	David	2/19/22	2/21/22	3	100%	
4.19	Create FPGA module to parse data from RPi	William	2/22/22	2/26/22	5	100%	
4.20	Create FPGA Hashing module for Bitcoin	William	2/25/22	3/2/22	6	100%	
4.21	Create FPGA module to test if puzzle solved	William	3/1/22	3/3/22	2	100%	
4.22	Connect FPGA modules	William	3/2/22	3/4/22	3	100%	
4.23	Spring Break		3/4/22	3/13/22	10	100%	
4.24	Literature Review (1 Paper)	All	3/17/22	3/23/22	7	100%	
4.25	Research Ethereum hash algorithm	William	3/14/22	3/16/22	4	100%	
4.26	Create FPGA Hashing module for Ethenum	William	3/16/22	3/20/22	5	100%	
4.27	Add Configuration Switching Support	William	3/21/22	3/23/22	3	100%	
4.28	Connect FPGA modules	William	3/21/22	3/23/22	3	100%	
4.29	Create simple choosing algorithm	David	3/24/22	3/26/22	3	100%	
4.30	Find way to choose which FPGA uses which synthesized code	William	3/27/22	3/29/22	3	100%	
4.31	Scale up to 2 FPGAs	William	3/30/22	4/1/22	3	100%	
4.32	Scale up to 5 FPGAs	William	4/1/22	4/3/22	3	100%	
4.33	Make sure data formatting is consistent/correct	William	4/2/22	4/3/22	2	100%	
5	**Testing**						
5.1	One way communication test for RPi	Lulu	3/14/22	3/17/22	4	100%	
5.2	One way communication test for FPGA	William	3/17/22	3/20/22	4	70%	
5.3	Test FPGA module for communicating with RPi (2 way)	William/Lulu	3/24/22	3/27/22	4	50%	
5.4	Create test puzzles	William	3/21/22	3/23/22	3	100%	
5.5	Measure hash rate	William	3/19/22	3/21/22	3	100%	
5.6	Measure module area and power	William	3/17/22	3/18/22	2	100%	
5.7	Test connection for WebApp	Lulu	3/17/22	3/19/22	3	40%	
5.8	Test Synthesizing process	William	3/14/22	3/16/22	3	100%	
5.9	Test both synethesized files, taking them on and off the boards	William	3/23/22	3/25/22	3	100%	
5.10	Test choosing algorithm with own benchmark	David	3/25/22	3/29/22	5	100%	
5.11	Test 2 FPGA setup	William	3/30/22	4/1/22	3	100%	
5.12	Test 5 FPGA setup	William	4/1/22	4/3/22	3	100%	
5.14	Test sign in, sign out, and profile setting function	Lulu	2/16/22	2/18/22	3	100%	
5.15	Test the form for inputs	Lulu	2/21/22	2/23/22	3	100%	
5.13	Test the urls for each web pages are set up correctly	Lulu	2/28/22	3/1/22	2	100%	
5.16	Connect to Bitcoin blockchain/ Mining Pools	David	3/28/22	4/3/22	7	90%	
6	**Interim Demo**						
7	**Working**						
7.1	Scale up the choosing algorithm to work for n FPGAs	David	4/4/22	4/14/22	4	80%	
7.2	Change algorithm to statically depend on price, success rate and other variables	David	4/8/22	4/20/22	4	90%	
7.3	Change choosing algorithm to be machine le	David	4/8/22	4/24/22	4	100%	
7.4	Scale up to 10 FPGAs	William	4/8/22	5/1/22	7	100%	
7.5	Complete Ethereum mining modules	William	4/14/22	4/30/22	7	100%	
7.6	Debug RPi communication modules	Lulu	4/11/22	4/30/22	5	70%	
7.7	Add metrics to Webapp	Lulu	4/15/22	5/1/22	6	0%	
8	**Testing**						
8.1	Test more advanced choosing algorithm	David	4/15/22	5/1/22	2	100%	
8.2	Test to make sure customizations are respected	David, William	4/16/22	4/24/22	2	100%	
8.3	Test machine learning algorithm	David	4/15/22	5/1/22	2	100%	
8.4	Use metrics to determine which is better	David	4/16/22	5/1/22	2	100%	
8.5	Test 10 FPGA setup	William	4/17/22	5/1/22	4	70%	
8.6	Debug WebApp issues	Lulu	4/18/22	5/1/22	3	0%	

Fig. 6. Full Schedule

TABLE I. 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	$167.12
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	197.07
Memory Card	Ultra 64GB UHS-I/Class 10 Micro SDXC Memory Card With Adapter	None	San Disk	1	$17.89
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
Jumper Wires	Male to Female 4 and 8 Inch Solderless Ribbon Dupont-Compatible Jumper Wires	None	GenBasic	1	$13.56
USB C to Micro SD Adapter	Anker USB C Hub, 5-in-1 USB C Adapter, with 4K USB C to HDMI, SD and microSD Card Reader, 2 USB 3.0 Ports	None	Anker	1	$38.10
USB Hub	Powered 10-to-1 USB 3.0 Data Hub Splitter	None	Atolla	1	$43.54
Ethernet Cable	Cat 8 Ethernet Cable, 20ft Heavy Duty High Speed Internet Network Cable, Professional LAN Cable, 26AWG, 2000Mhz 40Gbps with Gold Plated RJ45 Connector, Shielded in Wall	None	Vabogu	1	$23.5
Ethernet Cable	Cat8 Ethernet Cable, Outdoor & Indoor, 6FT Heavy Duty High Speed 26 AWG Cat8 LAN Network Cable 40Gbps, 2000Mhz with Gold Plated RJ45 Connector	None	DbillionDa	1	$22.98
USB C to Ethernet Converter	USB C 6 In 1 Hub	None	UtechSmart	1	$31.47
Power Strip	GE 6-Outlet Power Strip with 2 Ft Extension Cord, (Ended up not using this.)	None	General Electric	2	$11.90
Total Cost					$587.98