CE-412 Digital Systems II

Secure Hash Algorithm 2 Miner in Verilog

Professor Girma Tewolde, Ph.D.
Jeremy Maxey-Vesperman
Zach Butler

2017-12-18

I. INTRODUCTION

The Internet has completely changed the word by connecting everyone. The World Wide Web represents a disruptive technology because, for the first time in history, everyone has an equal ability to disseminate information. Blockchain represents the next socially disruptive technology because, in the same way that the World Wide Web enabled the exchange information over the Internet, blockchain enables the transfer of value over the Internet. But another facet is even more profoundly disruptive than its applications in the transfer of value. Blockchain also enables decentralized trust with no third party for the first time in human history.

Blockchain-based cryptocurrencies have existed for nine years. Users are rewarded for processing others' transactions by being awarded newly minted coin in a process called "mining". In 2013, mining became extremely profitable, so hardware miners were rapidly prototyped on field programmable gate arrays (FPGAs) and sold. Application-specific integrated circuits (ASICs) quickly followed for hardware miners, making FPGA-based mining technology obsolete in year or so.

Nonetheless, implementing a hardware mining solution on an FPGA remains an excellent academic opportunity to study blockchain technology and the algorithms used for real world cryptocurrencies on the Internet. Unfortunately, due to time limitations of the Kettering school term, we have decided to focus on one part of the algorithm, the secure hash algorithm 2. SHA-256 makes bitcoin possible. SHA-256 is also used for most SSL and TLS encryption over HTTPS on the Internet.

II. THE BITCOIN BLOCKCHAIN

A blockchain is a linear data structure analogous to a singly linked list, where a block represents a list element and the chain refers to the set of all blocks. Each block contains one reverse-facing link, one block index that increments sequentially from the head, metadata, and datum. The genesis block (index zero) is the head of the linked list and the current or most recent block is the tail. The reverse-facing links contain a cryptographic digest of the previous block. This cryptographic reverse-linking makes it impossible to modify data on the blockchain without breaking continuity. Data integrity can be verified by any participant who has a copy of the entire chain by computing the hash values of each block from head-to-tail[1]. Blockchain's tamper-resistance and finality are its primary benefit. Blockchain technology was first proposed in "How to Time-Stamp a Digital Document," published in *Journal of Cryptology* by Stuart Haber and W. Scott Stornetta in 1991[2].

Today's emerging blockchain industry began Halloween 2008 when *Bitcoin: A Peer-to-Peer Electronic Cash System* was published online[3] and distributed to The Cryptography Mailing List under pseudonym Satoshi Nakamoto[4]. That paper will forever be remembered for enabling decentralized trust over the Internet through consensus. Trust is a factor of identity and permission. 'Who are you, and what are you trying to do?' Similarly, computational trust is based on authentication and authorization. Bitcoin merges three existing technologies to achieve decentralized trust: blockchain, peer-to-peer technology, and public key cryptography[5]. Bitcoin is a decentralized communication network created over the Internet using peer-to-peer technology. Asymmetric cryptography establishes identity and ownership on the network. The private key is proof of ownership, while the public key and

corresponding bitcoin address are an identity. Network participants authenticate themselves and each other through the use of digital signatures[6].

Bitcoin uses an unspent transaction output (UTXO) architecture, which is a fancy way to remind you that the datum on the bitcoin blockchain are transactions rather than account balances. Every transaction in the ledger consumes a set of inputs and produces a set of outputs. Transaction outputs can be either spent or unspent. A spent transaction output is immediately consumed as the input to another transaction, analogous to mail forwarding. Unspent transaction outputs are simply outputs that have yet to be consumed as the input to another transaction. The balance of a given bitcoin address is simply the sum of UTXOs associated with that address. A transaction can be authorized if and only if it contains a valid digital signature from the owner of every input UTXO, and the value of consumed UTXOs is equal to or greater than the value of produced UTXOs[6,7].

Mining is the process of verifying transactions, adding verified transactions to the public ledger, and minting new bitcoin[8]. Transactions broadcast on the network since the last block are disseminated by miners and verified according to the rules above. A block consists of those verified transactions, the hash of the previous block header, metadata, a nonce, and one transaction creating a predetermined amount of new bitcoin. The block digest must be calculated using the hashcash proof-of-work algorithm with two rounds of SHA-256[9]. By changing the nonce value and including new transactions, miners compete to be the first to create a candidate block whose digest begins with a run of zeroes. Miners are compensated with their newly minted bitcoin and transaction fees[6], worth over $280,000 per block at the time of this writing[10]. The required number of leading zeroes in the digest determines the difficulty[3], and is adjusted such that it is probable a bitcoin block will be mined once every ten minutes on average[6].

Majority consensus is used to verify mined blocks. Other miners verify the transactions and proof-of-work for a candidate block. Accepted blocks become the tail of the blockchain, and miners start the process over using the candidate block hash as input to their proof-of-work. To reject a block, miners simply ignore it and continue to mine using the previous block header. It is possible to temporarily have two valid branches off the main blockchain (a fork) if there is a network disruption, a lack of consensus, or two miners find valid blocks at the same time. The bitcoin protocol always considers the longest chain to be the correct one and will work to extend it[3]. By definition, the majority consensus will have greater computing power than the minority and will be able to extend their chain faster. Once the majority chain outpaces the minority chain, consensus is achieved and the minority chain is abandoned. Mining rewards incentivize miners to mint honest blocks, confirm valid candidate blocks broadcast by others, and remain on the same fork. Time spent mining the wrong chain consumes energy and risks future mining rewards[3,6].

III. NABC ANALYSIS

3.1 Need

The security of the bitcoin network relies upon having as many full nodes, or independent miners, as possible. Therefore, any bitcoin user with capital on the network has a vested interest in relaying and processing transactions for other users. Probability of finding a block is directly proportional to the number of hashes your hardware can perform per second, so bitcoin miners in search of block rewards need hardware-accelerated hashing solutions. Lastly, since this industry is so new, the academic community is only just now preparing curriculum and creating blockchain-oriented degree programs. They need a way to study the hashcash-SHA-256^2 algorithm in practice.

3.2 Approach

The authors have implemented the United States National Security Agency's 256-bit Secure Hash Algorithm 2 in Verilog, using the Intel EP2C35F672C6 field programmable gate array as a target. The FPGA can be connected to a general-purpose computer via serial and used to hardware-accelerate existing bitcoin mining software. The mining application would connect to the bitcoin network, collect and relay transactions, receive and transmit candidate blocks, and store the blockchain. It would offload hashing calculations to the FPGA.

3.3 Benefits per Cost

FPGA hardware is affordable and readily available. Verilog allows our implementation to be compiled to any supported FPGA, reducing reliance upon a single manufacturer or fabrication facility. Commonly available mining software already has extensive support for hardware-acceleration using FPGAs in the way we describe. Compared to application-specific integrated circuits, we do not need to wait for fabrication to take place between design and implementation. The reprogrammability of FPGAs gives us the ability to distribute a software update if we need to make corrections or improvements to the hardware, or if bitcoin one day adopts SHA 3. This reprogrammability also empowers miners to switch between coins which use different algorithms. This is useful because mining is incredibly competitive and it may be most profitable to mine a different cryptocurrency or token at any given point in time. Software already exists which will "automine" the most profitable coin, and our FPGA solution could be integrated with this kind of software to switch algorithms along with it. Lastly, most Universities which seek to create blockchain curricula already own FPGAs.

3.4 Competition

Unfortunately, application-specific integrated circuits have forced FPGA bitcoin mining solutions into obsolescence. FPGA bitcoin mining is now relegated to academia alone. On the contrary, this also means there is little active development on FPGA mining algorithms. This could enable us to corner the academic market.

IV. SYSTEM ARCHITECTURE

To design an FPGA-based mining solution, three major hardware modules needed to be designed and implemented in Verilog. This included a way to input chunk data, a SHA256 hasher, and a way to output the digest information. The authors' chose to implement UART as the means to receive chunk data and transmit the digest. This protocol was chosen as it was readily available on the FPGA board and one of the designers was familiar with the protocol.

4.1 Main Control Unit

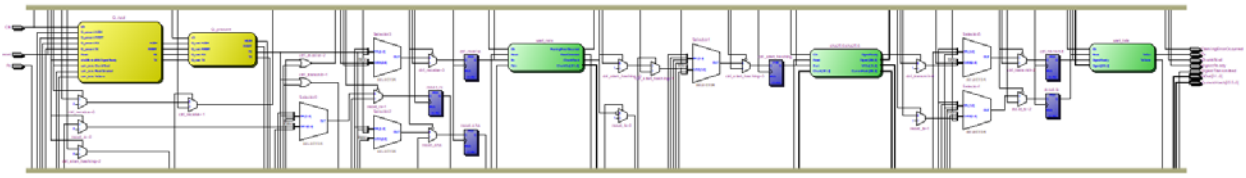

Figure 4.1.1 - SHA256 Hasher Control Unit Module

The authors' attempted to modularize the design as much as possible. Therefore, the SHA256 hasher was implemented in with one main Control Unit (CU) that regulated the operation of the hasher. This control unit was tasked with regulating three additional CUs; a SHA256 hasher CU, UART Tx CU, and UART Rx CU.

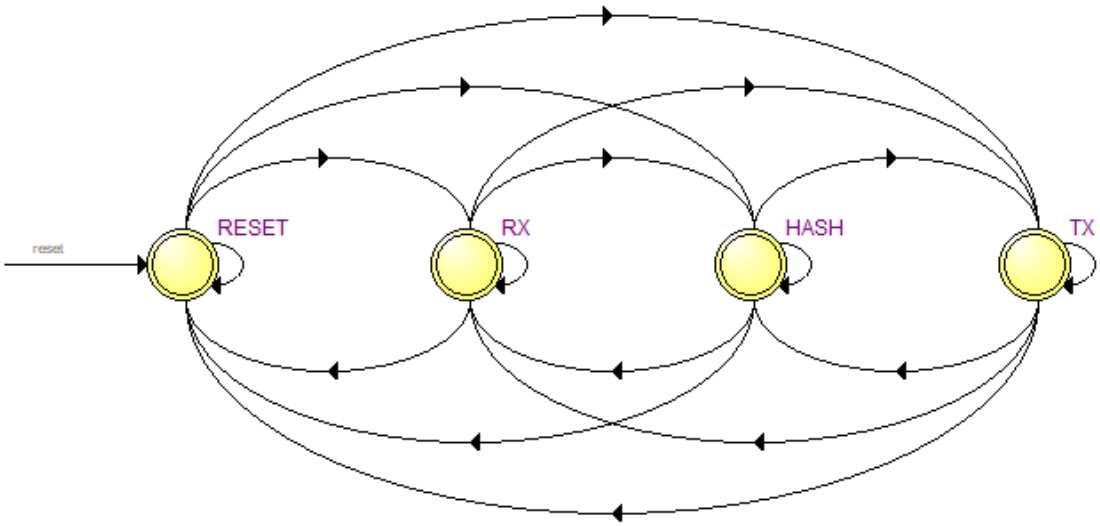

Figure 4.1.2 - Main Control Unit Finite State Machine

On reset / power-on, the main CU defaults to enabling the UART Rx CU. The device will continue to listen for data and receive data until it either receives a full chunk, a reset control signal via UART

(ASCII ‘r’ / ‘R’ in this implementation), or an asynchronous reset from a key on the FPGA board. If it has received a chunk, it will de-assert the control signal for the UART Rx module, (current implementation does not accept reset input until hashing and transmission is complete) and assert an enable signal for the SHA256 hasher CU. It will stay in this state until hashing is complete (indicated with an assertion of a digest ready flag). Afterwards, the CU will transition to its Tx state. The Tx CU will be enabled and the SHA256 hasher CU will be disabled. The system will stay in this state until transmission is complete. Finally, it will transition to the Reset state where it will reset all control signals before reentering Rx State.

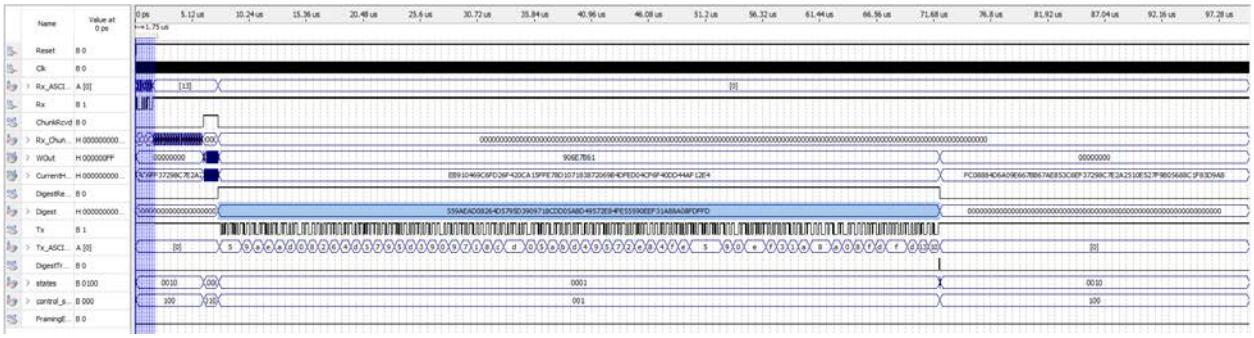

Figure 4.1.3 - Complete Functional Simulation of SHA256 hasher with UART chunk input of ‘A’, SHA256 hashing, and UART transmission of digest.

4.2 UART Rx Control Unit

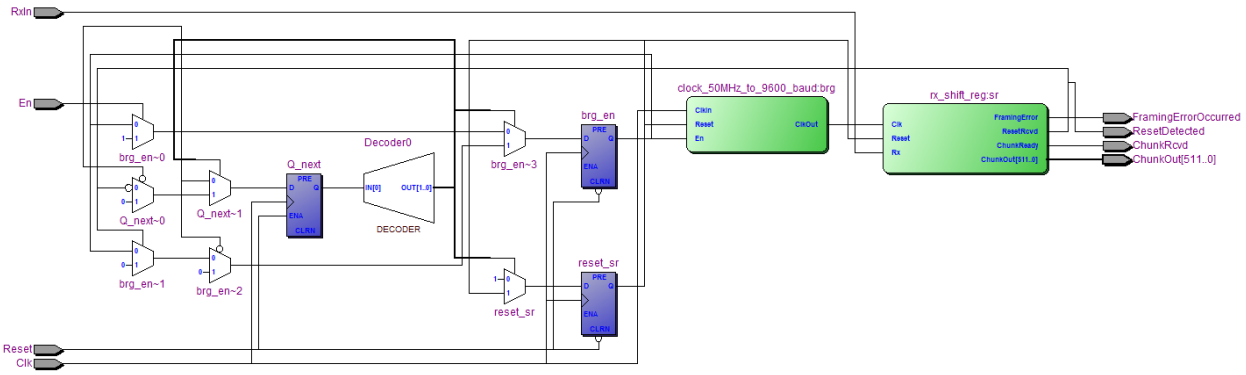

Figure - 4.2.1 UART Rx Control Unit Module

The UART Rx CU regulates the operation of a baud rate module and receiver shift register designed specifically for chunk data. Baud rate clock signal generation is regulated by enable and reset control signals. Likewise, control signals for reset and enable were implemented for the shift register.

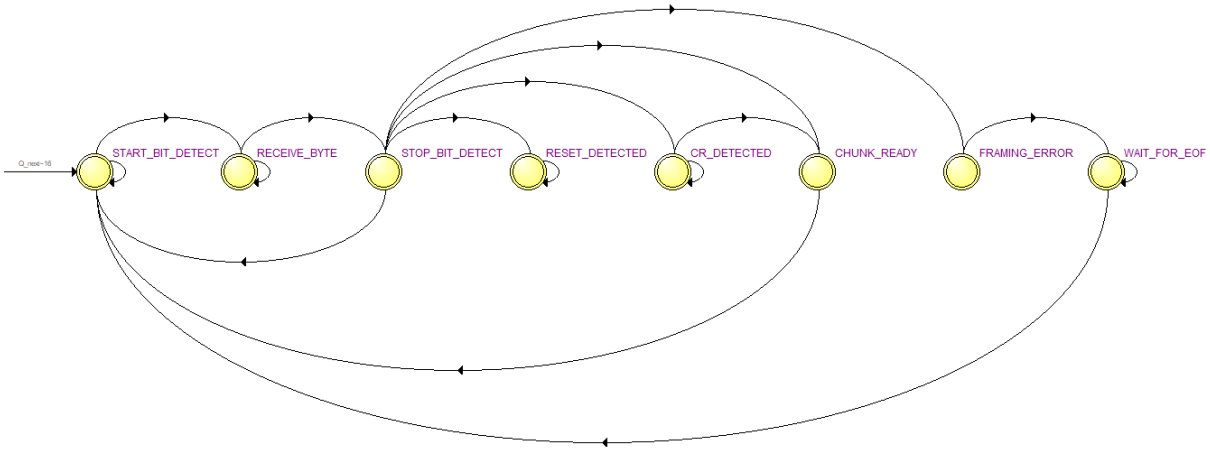

Figure 4.2.2 - UART Rx CU Finite State Machine

On reset / power on, the control unit enters a start bit detect state. In this state, the receiver listens for data on the UART Rx line. Upon transitioning to LOW (START bit), the receiver enters the Receive Byte state. In this state, the shift register module shifts data into an 8-bit ASCII input register every clock cycle (driven by baud rate clock signal) and increments a counter to determine when the end of data has been reached. Once this occurs, a transition to Stop Bit Detect state occurs. Here, the unit checks for a STOP bit (HIGH).

If this is not detected, a framing error has occurred. The unit transitions to a Framing Error state and discards the data in the ASCII in register as it is not considered valid input. A Framing Error flag is asserted to indicate the occurrence and then transitions to Wait For EOF. In this state, the receiver attempts to get realigned with the transmission frames. To accomplish this, it waits for silence on the receiver line for the duration of one UART frame; sampling occurs every cycle to ensure that it has waited long enough to ensure that it will not start listening for a START bit in the middle of a frame.

However, if the STOP bit is detected, the control unit looks at the ASCII input data to determine if it is a reset signal (‘r’ or ‘R’). If this is the case, the unit transitions to a Reset Detected state where it asserts a flag and waits in this state for the Main CU to handle the input. Otherwise, the Rx CU looks to see if the Chunk Ready flag has already been set. If so, it goes back to Start Bit Detect. However, if not it then compares the current frame count to the maximum allows frames per chunk

If this is not the case, it then looks to see if the input is a Carriage Return as this indicates the end of chunk data input. If not, it shifts the data into a 512-bit chunk register, increments a message length (in bits) counter (necessary for appending the message length at the end of the chunk; this is part of the preprocessing for the hasher algorithm), increments the frame counter, and transitions back to the Start Bit Detect state. Conversely, if the input data is a Carriage Return, a transition to CR Detected state occurs. Here, a 1 bit is appended to the end of the message, it is padded with 0’s, the length of the message is appended to the chunk, and it moves to the Chunk Ready state. Likewise, if the maximum frames have been reached, the 1 bit is appended along with the message length before switching states

to Chunk Ready; these steps are necessary for preprocessing the message data for the hashing algorithm.

In the Chunk Ready state, the unit sets a Chunk Ready flag so the Main CU can handle it and transitions back to the Start Bit Detect state to listen only for control inputs from the user until it is reset by the Main CU.

4.3 UART Tx Control Unit

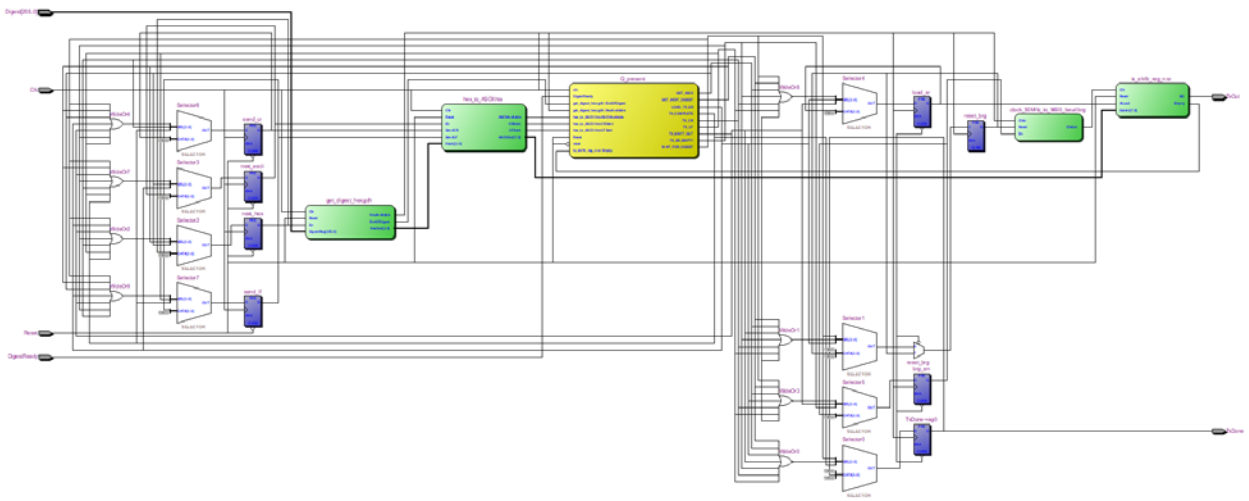

Figure 4.3.1 - UART Tx Control Unit Module

The UART Tx CU regulates the transmission of digest data back to the host. It regulates the operation of a baud rate clock generator module, a module that fetches the next hex value from the digest register, a hex to ASCII converter module, and a Tx Shift register module.

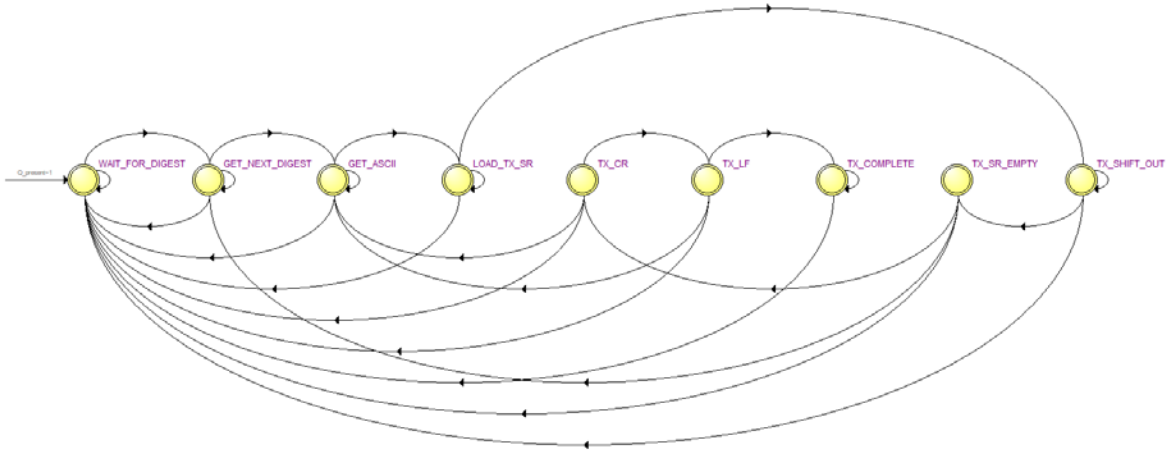

Figure 4.3.2 - UART Tx Control Unit Finite State Machine

On reset / power on, the unit enters a Wait for Digest state. Here it waits for the assertion of a control signal indicating that the digest is ready for transmission. Once this is the case, it fetches the next hex value from the digest.

At this point, a flag will be asserted if the last hex value has been retrieved from the digest register. The CU will then transition to a Get ASCII state. Here it will either assert a control signal to the ASCII conversion module to produce a Carriage Return or Line Feed ASCII value or convert the input hex value to ASCII depending on flag registers set in various other states.

Afterwards, the Tx control unit transitions to the Load Tx Shift Register state. In this state, the control unit asserts a load signal to the Tx shift register module and parallel loads the ASCII value for transmission. Next, the data is clocked out until the shift register is empty. At this point, the Tx Shift Register Empty state is entered. The CU looks to see if the last hex value retrieved flag is set. If so, it transitions to Tx CR. Here, it again looks at a flag to determine if a carriage return needs to be generated or if it already has been. It moves either to Get ASCII and asserts a CR flag or Tx LF based on this flag. Afterwards, it determines if a Line Feed has been sent. If not, it generates one by asserting a flag and entering Get ASCII state. Otherwise, a transition to Tx Complete occurs where it asserts a flag indicating this to the Main Control Unit. Finally, the module stays in this state until reset by the Main CU.

4.4 SHA256 Hasher Control Unit

The final control unit implemented in this design was the SHA256 Hasher Control Unit. This module controls the rounds of the hashing algorithm, calculation of w (the stirred up input data based on previous w values), and fetching of k constants for the algorithm module.

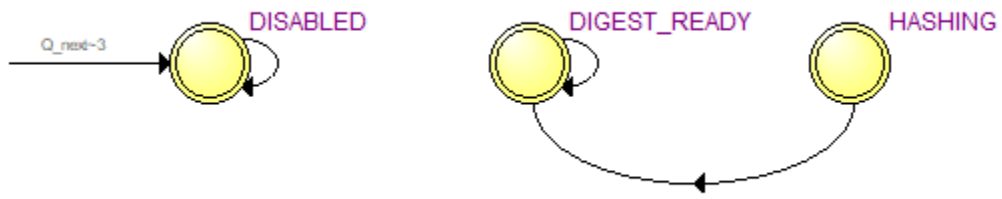

Figure 4.4.1 – SHA256 Hasher Control Unit Finite State Machine

The FSM for this CU is relatively straightforward. The unit enters a disable mode upon reset or power on. In this state, the initial round constants are loaded into the 32-bit word round registers (Ain – Hin) and waits for a control signal to be asserted from the main CU indicating that the preprocessed chunk is ready for hashing. On this transition, the W array register set it initialized with the chunk data from the UART Rx Module. From here, hashing rounds begin. On each positive clock edge, the round is incremented and (if this is round 16 or later) the oldest w value is shifted out of the register set and the

newest w value is stored into the register set. Additionally, the next k constant is selected and hashing is performed. On the falling edge, the round constants from the previous hashing round are stored into the round input registers for the next cycle. In this manner, each round of hashing is able to be performed in one clock cycle. Once the final round of hashing takes place, the unit transitions to a Digest Ready state. In this state, a flag is set for the Main CU to handle, the intermediate digest value is calculated by adding the final round's output word values to the initial round's input word values, and all words are concatenated together in one 256-bit digest register.

4.4.1 SHA256 Algorithm module

The hasher algorithm module implements the SHA256 algorithm:

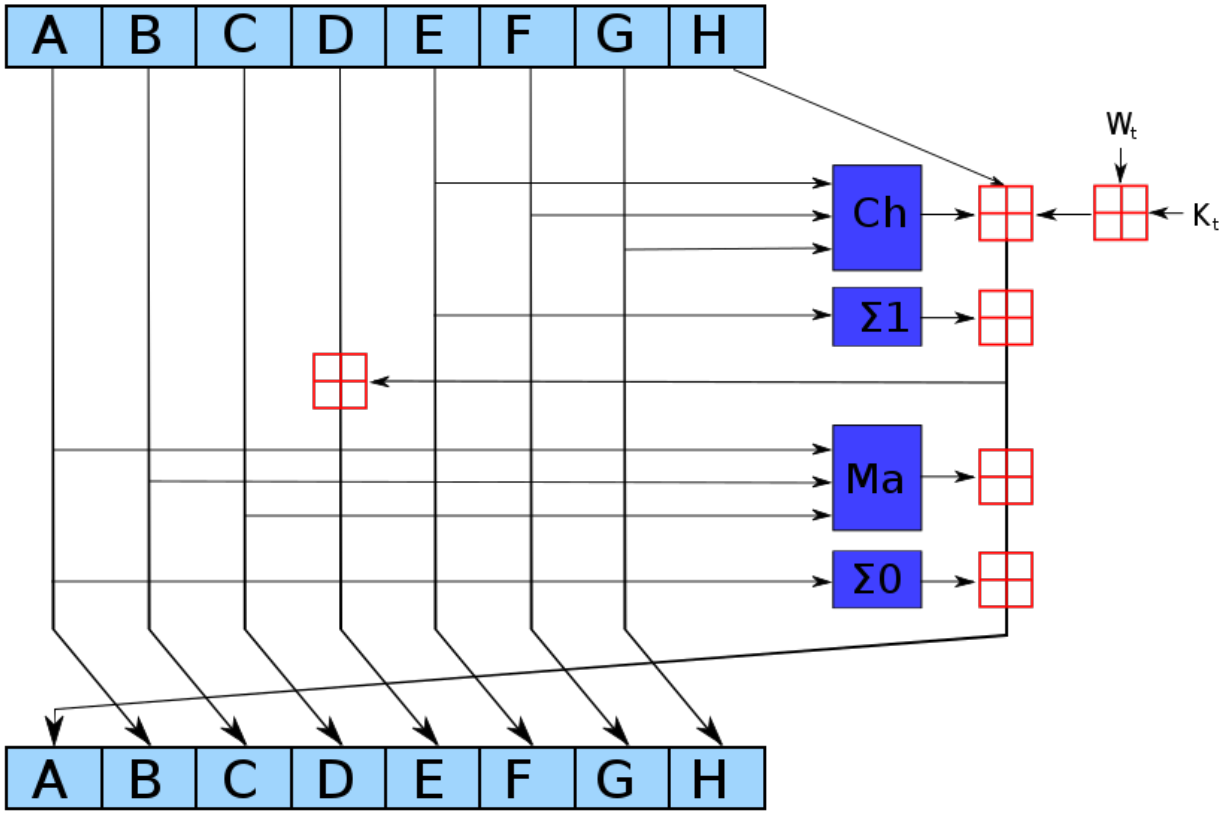

Figure 4.4.2 – Hashing algorithm performed each round (Retrieved from Wikipedia)

Each round, most of the 32-bit words are shifted over one position to the right except for A and E. A new A value is calculated using a variety of compression functions, round constants, stirred-up input data, and 32-bit addition steps. The Choose function (Ch) is essentially an array of 32 2:1 muxes. Each bit of E is used as the select signal for the input bit of F or G. Likewise, the Majority function (Ma) is essentially an array of 32 3-input, 2-bit adders (Where the upper bit of each input is tied to Logic 0 and the lower bit is A, B, and C). The second bit of the sum is the output of the function for each bit. The final

two functions, Sigma 0 and Sigma 1, perform very similar operations to A and E. The input to the function is split into two different right-rotated and one logical right-shifted versions. These are then bitwise XOR'ed together to produce the output of the function.

During the round, various 32-bit sums are also calculated to determine the output of A and E. The first addition step adds the round's k constant to the w input. This is then added together with H and the output of Ch. Next, this sum is added to the output of Sigma 1. At this point, E is calculated by summing the previous sum together with input D. Additionally, the output is used in another addition step with the output of Ma. Finally, A is calculated from the output of the previous sum with Sigma 0.

V. MINING RETURNS

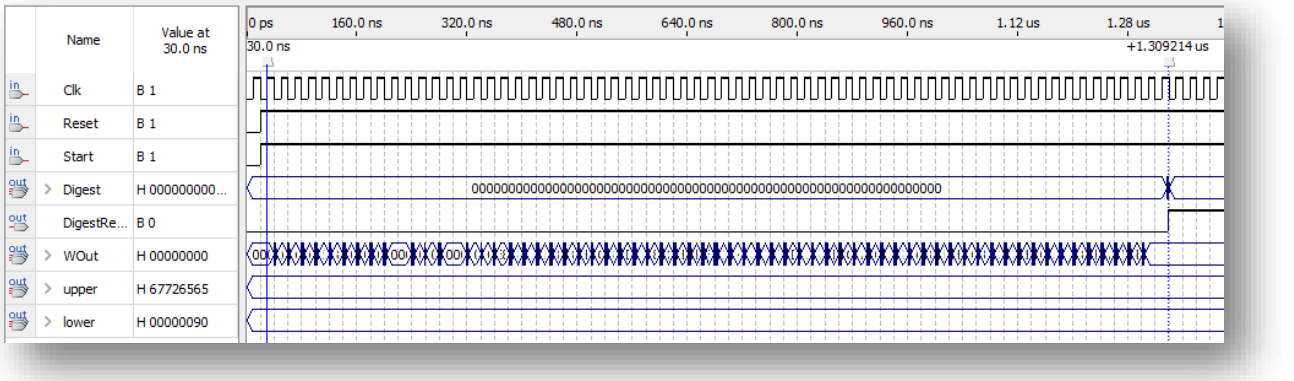

Figure 1: Algorithm Timing Analysis

Using our combinatorial algorithm, the Intel EP2C35F672C6 field programmable gate array returns one round of SHA-256 in 1,279.214 nanoseconds. Since bitcoin's hashcash protocol uses two rounds, this chip takes 2.558 microseconds to generate one bitcoin hash – a rate of 390,865 hashes per second. We can estimate the yield of this FPGA based on the current network hash rate of 11.3 exahashes per second and a 12.5 BTC block reward. This FPGA would generate 2E-8 BTC per year, or about two Satoshi (the smallest divisible unit of a bitcoin)11. At a current market rate of $17,518.37 per bitcoin, this comes to four hundredths of a cent per year. Compare this to its power consumption of 300 milliwatts, or about thirty cents per year12. Compare this to the BitFury Antminer S9 ASIC, capable of 14 terahashes per second at 1.372 kilowatts13. This ASIC isn't new by industry standards, and yet it would still generate 807.7 mBTC per year which is about $14,160. I think it is safe to say, you will not have to worry about students stealing your FPGA development boards to mine bitcoin at home.

VI. USER MANUAL

On reset / power on, the system defaults to a listening mode where it accepts user input via UART serial communication. The SHA256 Miner can be interfaced with using any Serial Terminal. It uses the following configuration settings for UART:

UART Configuration	
Setting	**Value**
Speed	9600 baud
Data bits	8
Stop bits	1
Parity	None
Flow Control	None

Table 6.1 – SHA256 Miner UART Configuration

The system accepts ASCII-encoded values. There are a few special commands. Chunk data can be any ASCII value except the following:

SHA256 Miner Commands				
Command	**ASCII value**	**Base-16**	**Base-10**	**Description**
Carriage Return (Enter)	CR	0xD	13	Signals the end of Chunk input
Reset	'R'	0x52	82	Perform system reset
Reset	'r'	0x72	114	Perform system reset

Table 6.2 – SHA256 Miner Commands List

To run the SHA256 Miner on the Altera DE2 Development Board:

1. Power on the system
2. Connect a cable between the RS-232 port on the DE2 development board and the PC
3. Open Putty
4. Configure Putty with the settings in Table 6.1 with the appropriate COM port selected
5. Enter chunk data
6. Press Enter to hash the input data
7. Miner will automatically calculate the digest and transmit it back to the terminal
8. Repeat, system will begin listening for chunk data after completing transmission

NOTE: Press 'r' / 'R' at any time while entering chunk data to clear the register and restart system

VII. KNOWN ISSUES

There are a few known issues with the current implementation of the Secure Hash Algorithm 2 Miner.

7.1 Chunks Containing 'r' or 'R' Triggers System Reset

Currently, any input chunk data will cause the system to reset. This is due to a design issue caused by a change in the planned implementation of the UART receiver. Originally, the designers planned to require preprocessing on the PC's end to convert all ASCII data into hexadecimal representation before passing to the Miner. However, this was changed to allow for any ASCII input

after the reset feature was implemented to eliminate the preprocessing requirement. A fix for this issue is trivial as it only requires a change to the literal compared to the ASCII shift register.

7.2 Intermittent Issue with UART Receiver Missing Input

There is an issue with the UART receiver that intermittently causes it to miss input. The root cause is most likely due to the way sampling occurs. The receiver begins sampling at the falling edge of the START bit. Considering that UART is asynchronous and has a large tolerance, it is likely that the receiver is not always sampling the data bits at the correct time. This would result in an incorrect character being registered. A fix for this would be to design the receiver such that it samples the start bit at a much faster rate (say 16x) than the configured baud rate. By doing so, the receiver could center sampling over the START bit and make the system more accurate.

NOTES

[1] Et Al, "Blockchain," *Wikipedia, The Free Encyclopedia*, 2017-Dec-09, https://en.wikipedia.org/wiki/Blockchain (2017-Dec-03).

[2] Stuart Haber and W. Scott Stornetta, "How to Time-Stamp a Digital Document," J. Cryptol., **3** (2), 99-111 (1991).

[3] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," *bitcoin.org*, 2008-Oct-31, https://bitcoin.org/bitcoin.pdf (2017-Aug-17).

[4] Satoshi Nakamoto, "Bitcoin P2P e-cash paper," *The Cryptography Mailing List*, 2008-Oct-31, http://www.metzdowd.com/pipermail/cryptography/2008-October/014810.html (2017-Dec-09).

[5] Nolan Bauerle, "What is Blockchain Technology?," *A Beginner's Guide to Blockchain Technology*, https://www.coindesk.com/information/what-is-blockchain-technology/ (2017-Dec-09).

[6] Phil Champagne, The Book Of Satoshi: The Collected Writings of Bitcoin Creator Satoshi Nakamoto, (e53 Publishing, Austin, TX, 2014).

[7] Vitalik Buterin, "Thoughts on UTXOs by Vitalik Buterin, Co-Founder of Ethereum," *ConsenSys Blog*, 2016-Mar-09, https://medium.com/@ConsenSys/thoughts-on-utxo-by-vitalik-buterin-2bb782c67e53 (2017-Dec-03).

[8] Investopedia, "Bitcoin Mining," *investopedia.com*, 2014, https://www.investopedia.com/terms/b/bitcoin-mining.asp (2017-Dec-09).

[9] Et Al, "Hashcash," *Bitcoin Wiki*, 2017-Mar-25, https://en.bitcoin.it/wiki/Hashcash (2017-Dec-15).

[10] BitInfoCharts, "Bitcoin (BTC) price stats and information," *BitInfoCharts*, 2017-Dec-09, https://bitinfocharts.com/bitcoin/ (2017-Dec-15).

[11] CryptoCompare, "Bitcoin Mining Profitability Calculator," *cryptocompare.com*, https://www.cryptocompare.com/mining/calculator/btc?HashingPower=390.865&HashingUnit=KH%2Fs&PowerConsumption=0.3&CostPerkWh=0 (2017-Dec-15).

[12] Intel Corporation, "Cyclone II Power Comparison—1/2 the Power of Competing 90-nm Low-Cost FPGAs," *Intel FPGA*, https://www.altera.com/products/fpga/cyclone-series/cyclone-ii/features/cy2-power-compare.html (2017-Dec-15).

13 Jordan Tuwiner, "Bitcoin Mining Hardware," What is Bitcoin Mining?, 2017-Jun-27, https://www.buybitcoinworldwide.com/mining/hardware/ (2017-Dec-15).