Bitcoin Mining Node Simulation

Chris Baumler, Daniel Kurfis,
Aaron Hoelschler, Derek Harmon
Iowa State University
CPRE 588 XE Team 4
Iowa State University

***Abstract*—The recent emergence of cryptocurrency as a viable alternative to nationalized currency is posing interesting challenges to many stakeholders, including nation-states, banks, retailers, and consumers, not to mention the technological challenges surrounding the generation, distribution, storage, exchange, and security of cryptocurrencies. This paper explores some of the technological challenges related to cryptocurrencies, using Electronic System Level (ESL) modeling techniques, focusing on speed and power estimation to assist developers in making decisions regarding component selection. The paper describes the modeling of a complete mining "node", involving multiple processing elements and specialized hardware. Speed and power estimation techniques are employed to compare several devices to host a hashing algorithm, including general-purpose processors, FPGAs, and ASICs.**

I. Introduction

According to [1], a new financial technology labeled a "cryptocurrency" is emerging where digital money is treated as a currency, a global digital payment system, and a peer-to-peer electronic financial institution. They go on to describe how cryptocurrencies work by providing a public ledger for storing transactions that is distributed across all the nodes in a peer-to-peer network.

Several different cryptocurrencies have been created, with "Bitcoin" being the most recognizable name. The technology associated with the Bitcoin "system" covers many areas, including something known as Bitcoin "mining". Bitcoin mining involves specialized processing nodes designed to support validation of Bitcoin transactions through the use of 256-bit hash codes, using the NSA's standardized SHA-256 hashing algorithm.

Bitcoin "miners" monitor the Bitcoin network, looking for transactions to validate. A mining "node" bundles one or more transactions into a "block", then begins computing SHA-256 hashes to validate the transaction block. Upon successful completion, the mining node submits the hash to the Bitcoin network and receives a number of Bitcoins as a reward. This process of rewarding a successful miner with Bitcoins is the mechanism by which Bitcoins are "minted". The challenge with mining comes from the fact that many "miners" are working to solve the same problem simultaneously. Only the first miner to report a successful hash receives the reward, making speed a key objective of any mining node.

The key to profitable mining is to minimize the power consumption of the mining hardware while maximizing its speed. In their paper "*A Green Computing Based Architecture Comparison and Analysis,*" [3] Feng and Lung emphasize the importance of simulation in designing low power embedded systems. This mindset has been carried forward in the implementation of this project which attempts to model and simulate a Bitcoin mining node

Section III.A (Conceptual View) describes a notional mining node, forming the basis for the project design. Note that this conceptual view is simplified to illustrate the basic concepts of a Bitcoin mining node; additional details emerge in the Specification Model section (paragraph 3.2).

The Team 4 project involves the design of a Bitcoin mining node, with a focus on performance measurements for the SHA-256 hashing algorithm. These measurements are captured for a number of target processing elements to help identify an optimal solution.

II. Project Goals

From the outset, the project team established very high expectations. The following list summarizes key goals for the project. A brief discussion of accomplishments ensues.

- System Design and Modeling
  - Executable System Specification model developed in SpecC
    - Model implements most Bitcoin mining node features
    - Model instrumented to measure speed and power for hashing function
  - Architecture Graphing
  - Executable Architecture Model in SpecC
  - Executable Network Transaction Level Model (TLM) in SpecC
- Design Space Exploration
  - Profiling of Multiple Processing Elements (speed and power)
  - Simulation of each Processing Element to Collect Speed and Power Metrics
- Design Space Exploration
  - Specification Model Instrumented to Collect Power Consumption Metrics
  - Simulations Include Power Data for Actual Components

All of the goals stated above were accomplished. In order to complete all of the goals, some elements of the model were simplified. None of the simplifications involved the instrumentation or measurement of speed or power. The types of simplifications are summarized as follows:

- The hashing algorithm in the hardware miner does not use the entire transaction block header provided

by the mining software, but instead uses a randomizing function to mimic portions of the block header. Since those portions of the header are somewhat random in nature already, this simplification does not alter the speed or power measurements of the hashing algorithm.

- Selected data structures and representation, outside of the hardware miner, were simplified to permit abstraction of behavioral elements that were not central to the collection of speed and power metrics.
- Selected algorithms, outside of the hardware miner, were simplified to permit abstraction of behavioral elements that were not central to the collection of speed and power metrics. These include hashing functions and encryption.
- Certain interface protocols were abstracted within SpecC channels.

III. DESIGN

This section describes the design of our Bitcoin mining node. We first provide a conceptual overview that shows the design from a high level and summarizes each of the major components. We then explain each component in greater detail. We also provide diagrams of the specification model, architecture model, architecture graph, and TLM model we developed through following the system-level design methodology. The specification model was derived from the system requirements we specified for ourselves, and the other models were developed and refined from the specification model.

A. Conceptual View

At a high level our Bitcoin mining node consists of five basic components, which are summarized below. These components work together to provide a complete system, allowing a user to interact with the Bitcoin peer-to-peer network by mining blocks and sending and receiving Bitcoin.

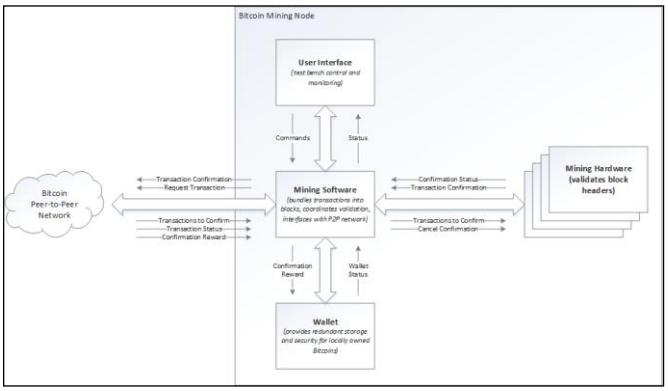

Figure 1: Elements of the high-level Bitcoin mining node in the design

- **Mining Software** - This component is an abstraction that encompasses a daemon process used to receive transactions from the Peer-to-Peer Network, and a mining application that bundles transactions into blocks, delivers block headers and other parameters to the mining hardware, and interfaces with a Bitcoin "wallet" application for the storage and retrieval of locally owned Bitcoins.
- **Mining Hardware** - This component iterates through the possible nonce values (random numbers) in the header, generates hashes, and checks if the hashes are within a valid range.
- **Wallet** - This component provides storage and retrieval of locally owned Bitcoins, receiving Bitcoin "rewards" from the Peer-to-Peer Network when successfully completing a transaction block hash before any other miners.
- **Peer-to-Peer Network** - This abstract component represents all other entities in the Bitcoin "world" that the mining node needs to interact with. Except for the data flows into and out of the Peer-to-Peer Network, modeling of this component is beyond the scope of this project.

- **User Interface** - This component represents the test bench, used to monitor the status of the mining node, monitor transactions in-progress, query for Wallet status, and report performance statistics.

B. Specification Model

The specification model we developed contains six major behaviors encapsulated within a test bench used to execute simulations. The stimulus and monitor are part of the test bench and are used to inject test inputs and monitor test outputs respectively. The four components included in the design make up the actual Bitcoin miner system. Details about each component may be found in the following subsections.

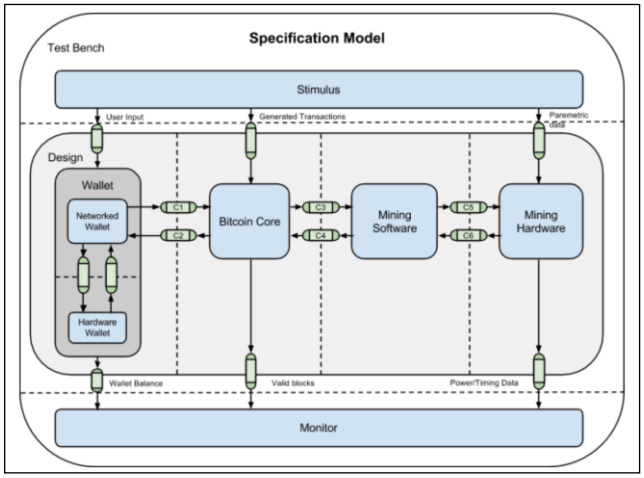

Figure 2: Specification Model

1) *Bitcoin Core(Network Daemon)*

The Bitcoin Core component represents a Bitcoin peer node that is part of the peer-to-peer network. This component is responsible for maintaining a local copy of the blockchain as well as a pool of transactions that have yet to be included in a block. As part of the peer-to-peer network, the Bitcoin Core listens for messages from other nodes and sends its own messages to the network. Received messages inform the node of new blocks to add to the blockchain and new transactions to add to the pool. Messages sent to the network include announcing transactions created by the local wallet and announcing new blocks mined by the local miner.

Since the Bitcoin Core is the central hub of information related to the Bitcoin protocol, it is also responsible for providing a set of remote procedure calls (RPCs) by which the other components get the data they need and interact with the network. A remote procedure call is a communication mechanism that allows a process to call a procedure on another process that may be executing on an entirely different processing element as if the procedure were being called on the current process. To implement this, a client-server model has been chosen. The server instances reside within the Bitcoin Core and communicate with the clients via a pair of request/response channels. Messages sent over these channels are packetized, containing various header and payload information to facilitate RPC. The end user is presented with an Application Program Interface (API) that allows various functions to be called. When this occurs, the client creates a message, sends it to the server, and blocks waiting for a reply. The server performs the desired action and responds. The following APIs are implemented as part of this modeling effort.

API	Description
getblocktemplate	Returns information used to construct a block during mining
submitblock	Broadcasts a mined block to the peer-to-peer network
getblockcount	Returns the number of blocks in the local best blockchain
Gettxoutsetinfo	Returns the set of unspent transaction outputs and details
gettxout	Returns details about a specific transaction output
createrawtransaction	Creates a transaction spending Bitcoin
signrawtransaction	Cryptographically signs a transaction
sendrawtransaction	Broadcast a transaction to the peer-to-peer network

Table 1: Implemented APIs

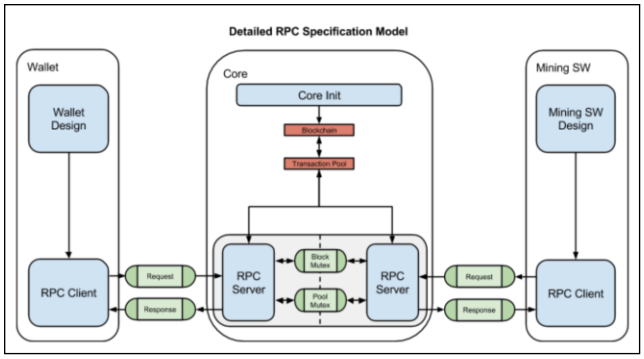

Figure 3: Detailed RPC Specification Model

2) *Mining Software (Transaction Bundling)*

As stated above, the Mining Software's primary function is a) to retrieve transactions from the peer-to-peer network and form them into a block header to present to the mining hardware for processing and b) to transmit a successfully hashed block header combined with the block to the network so it can be added to the blockchain.

The Mining Software retrieves the data needed to create the block header by calling the getblocktemplate RPC (Remote Procedure Call). This call returns the following information:

1. The information necessary to construct a coinbase transaction paying the miner's wallet
2. A complete dump of the transactions the core suggests including in the block, allowing the mining software to inspect the transactions, optionally add additional transactions, and optionally remove non-required transactions.
3. Other information necessary to construct a block header for the next block: the block version, previous block hash, and bits (target).
4. The current target threshold for accepting shares.

The block header that is sent to the Mining Hardware is 80 bytes in size and consists of the following fields:

Field	Size (bytes)	Description	Comes from…
Version	4	A version number to track software/protocol upgrades	getblocktemplate
Previous Block Header Hash	32	A reference to the hash of the previous (parent) block in the chain	getblocktemplate
Merkle Root Hash	32	A hash of the root of the Merkle tree of this	Constructed from transaction IDs
Time	4	The approximate creation time of this block (seconds from Unix Epoch)	getblocktemplate
nBits	4	The proof-of-work algorithm difficulty target for this block	getblocktemplate
nonce	4	A counter used for the proof-of-work algorithm	Arbitrary number changed by mining hardware, initially 0

Table 2: Block Header Fields

The processing that the Mining Software performs consists of computing the merkle root of the transactions IDs in the block received from the getblocktemplate RPC. Merkle trees are used in Bitcoin to summarize all the transactions in a block, producing an overall digital fingerprint of the entire set of transactions, providing a very efficient process to verify whether a transaction is included in a block. A merkle tree is constructed by recursively hashing pairs of nodes until there is only one hash, called the *root*, or *merkle root*. The cryptographic hash algorithm used in Bitcoins merkle trees is SHA256 applied twice, also known as double-SHA256.

The merkle tree is constructed bottom-up. In the following example, we start with three transactions, A, B, and C, which form the leaves of the merkle tree. The transactions are not stored in the merkle tree; rather, their data is hashed (double) and the resulting hash is stored in each leaf node as H_A, H_B, H_C (32 bytes each). Consecutive pairs of leaf nodes are then summarized in a parent node, by concatenating the two hashes and hashing them together. For example, to construct the parent node H_{AB}, the two 32-byte hashes of the children are concatenated to create a 64-byte string. That string is then double-hashed to produce the parent node's hash. Because the merkle tree is a binary tree, it needs an even number of leaf nodes. If there are an odd number of transactions to summarize, the last transaction hash will be duplicated to create an even number of leaf nodes, also known as a balanced tree. This is shown in the figure below where transaction C is duplicated. The pairing-hashing process continues until there is only one node at the top, the node known as the Merkle root. That 32-byte hash is stored in the block header and summarizes all the data in all three transactions.

The same method for constructing a tree from three transactions can be generalized to construct trees of any size. In Bitcoin it is common to have several hundred to more than a thousand transactions in a single block, which are summarized in exactly the same way, producing just 32 bytes of data as the single merkle root.

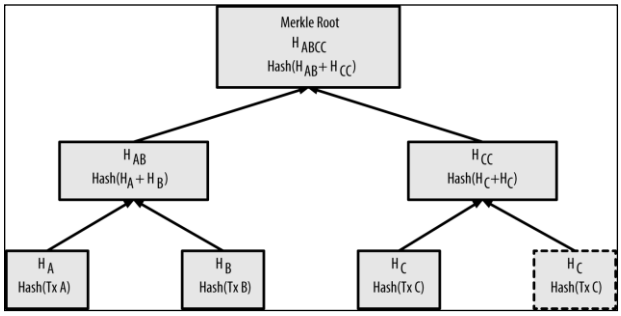

Figure 4: Merkle Root computation

After the merkle root is calculated and the remaining block header fields are filled in, the block header is sent to the mining hardware for processing (hashing). The goal is now to find a value for the nonce that results in a block header hash that is less than the difficulty target. The mining node will need to possibly test billions or trillions of nonce values before a nonce is found that satisfies the requirement.

Once the hardware finds a hash value that is less than the target value, it returns the block header with the successful nonce to the mining software combines the header with the block and sends the completed block to core node to be broadcast to the network for addition to the block chain.

If none of the hashes are below the threshold, the mining hardware gets an updated block header with a new merkle root from the Mining Software; this new block header is created by adding extra nonce data to the coinbase field of the coinbase transaction.

3) Mining Hardware (Hash Algorithm)

This section describes the Mining Hardware component which is primarily a hash generator. Note that the high level block diagram shows multiple instances of the generator. The design description presented here focuses on a single instance of the "Mining Hardware." Any use of multiple hardware miners is strictly up to the "Software Miner" component; the individual "Hardware Miner" components do not interact with each other.

a) Mining Hardware Specification Diagram

The specification diagram below expands the Mining Hardware component from the block diagram to expose behaviors Configure, Abort, Watchdog, and Hash, several global events, and global data store Config. In this specification diagram, the Stimulus and Monitor behaviors essentially represent the "User Interface" in the block diagram, and the Mining Software behavior provides simulated block headers for validation. The Wallet is not

represented in this model, because it does not interface with the Mining Hardware.

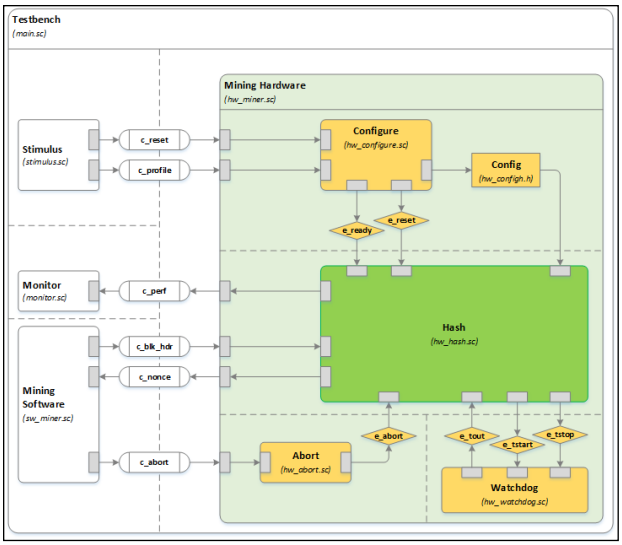

Figure 5: Mining Hardware Behavior

b) Mining Hardware Behavior

The internal Mining Hardware behaviors run in parallel with respect to each other. The Stimulus provides Configure with profile information for a given Processing Element (PE). The information used to profile a PE includes clock speed, power consumption, and cycle counts for each instruction type needed by the Hash behavior. Once the Configure behavior has received and stored the profile data, an e_ready event is issued by Configure to activate the Hash behavior. At this point, the Hash behavior is blocked, waiting for a block header on channel c_blk_hdr.

On receipt of a block header, the Hash behavior runs the hashing algorithm repeatedly, until finding a valid solution. While the algorithm is running, statistics are continuously accumulated for timing and power consumption, using the Config data provided by the Stimulus behavior. On finding a valid result, the Hash behavior reports the solution (nonce) back to the Mining Software, and also sends accumulated statistics to the Monitor. The Hash behavior then waits for another block header from the Mining Software.

A Watchdog timer behavior is used to prevent the simulation from running too long, automatically halting a hash search in-progress after a configurable period. The Hash behavior may also be aborted in-progress through the c_abort channel, which may be used by the Mining Software component in certain circumstances, such as receipt of a new transaction block to be processed or, when multiple Mining Hardware components are running in parallel, to halt all remaining algorithms as soon as one algorithm produces a valid result.

The Hash behavior represents the functionality found in an FPGA or ASIC mining node, while the other behaviors within the Mining Hardware primarily support the simulation. Therefore, all instrumentation to collect timing and power measurements is found in the Hash behavior.

c) Mining Hardware Code Struture

The Hash behavior code is written in a combination of C and SpecC. Open source examples of the hashing algorithm were found in both C and Verilog languages, which were found to be nearly identical (functionally, not structurally). An open source C implementation of the hashing algorithm was chosen for modeling [10], for ease of integration with SpecC. The SpecC portion of the Hash behavior primarily serve to create a thread within which to run the algorithm, and to provide communication and synchronization mechanisms to interact with the rest of the mining node components. Due to the complexity of this component, and "intermediate" test bench environment was established to support preliminary testing prior to integration.

4) Wallet (Bitcoin Storage)

The Wallet component is a repository for storing information about Bitcoin transactions, which are saved in a file as collection of private keys. It essentially serves as a platform for certifying transactions and also provides storage of bitcoin holdings. The Wallet is also responsible for granting the ability to access this virtual form of currency in addition to spending it. There are several types of wallets which have unique characteristics and subtle differences between one another.

Bitcoins do not actually exist. They exist in the sense that Bitcoin transactions between two addresses are recorded and stored inside the Wallet. On the most basic level, a transaction is comprised of three different elements: An input, an output, and an amount. The input is the is a record of the bitcoin address that was used to execute a transaction, whereas the output is the bitcoin address itself. The amount is the amount of Satoshis the user has, much like a bank account balance. To create a transaction, Bitcoins require an address and a private key. The private key is a secret number uniquely tied to all Bitcoin addresses generated for wallet storage. These keys are what allow Bitcoin access and enable transactions to occur. Without the key, bitcoins cannot be accessed.

Our intended design was to be built around the Signing-Only Wallet implemented as a Hardware Wallet. This type of wallet works together with a Network Wallet and interfaces with the peer-to-peer network. Since keys are what legitimize transactions and enable records saved within the wallet, the Signing-Only Wallet provides a level of increased security. The image below reveals the progression of deterministic key creation.

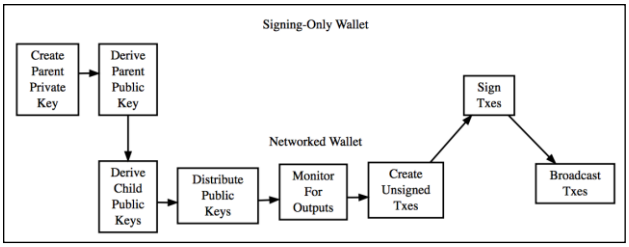

Figure 6: Wallet Behavior

As stated above the Wallet component is comprised of two behaviors that work in conjunction with one another to sign and store Bitcoin transactions. These behaviors are the Hardware Wallet and Network Wallet. The Network Wallet interacts with the peer-to-peer network, whereas the Hardware Wallet has the main duty of signing transactions. When transactions are created, the following steps take place:

1. Look into the block-chain and identify new transactions (Network Wallet)
2. Generate a parent public key (hard-coded for simplicity) and point it to the address (Hardware Wallet)
3. Generate a parent private key and point it to the address (Hardware Wallet)
4. Use parent public key to derive child public key (Network Wallet)
5. Attach the parent chain code, parent public key, and index number to the hash (Network Wallet)
6. Monitor for outputs spent to the public keys (Network Wallet)
7. Create unsigned transaction spending those outputs (Network Wallet)
8. Sign the transactions (Hardware Wallet)
9. Broadcast the signed transactions to the network (Network Wallet)

To limit scope for this project, a simplified version of the Wallet has been implemented that ignores the complexity of producing private and public keys and actually cryptographically signing the transactions.

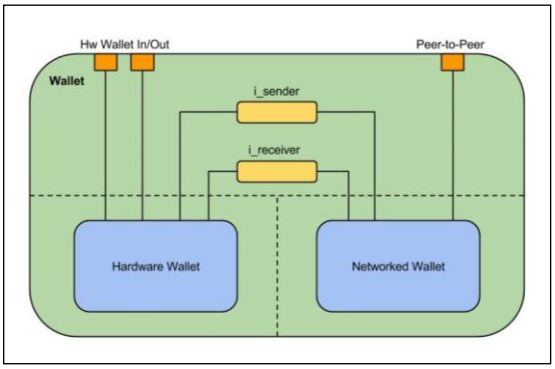

Figure 7: Wallet Behavioral Specification Model

C. *Architecture Model*

The following architecture model is derived from the specification model. It maps the Bitcoin Core and Mining Software components to one PE, the Mining Hardware to one PE, and the Network and Hardware Wallets to separate PEs. The Hardware Wallet is mapped to a custom IP that provides greater security in managing keys. Likewise, the Mining Hardware is mapped to a hardware PE designed for speed and power optimization such as an FPGA or an ASIC.

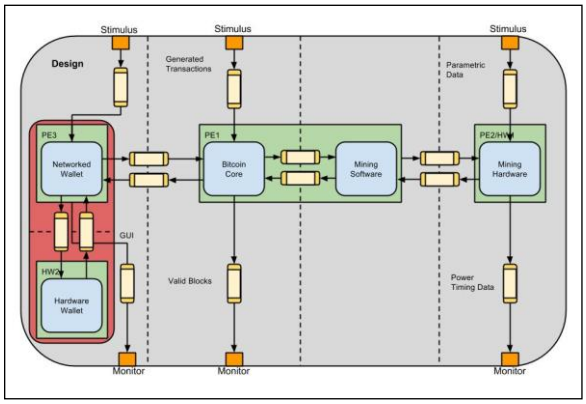

Figure 8: Architectural Model

1) *Task Graph*

The following task graph shows how the major tasks within the Bitcoin mining node interact. Essentially, the Bitcoin core communicates with the peer-to-peer network (test bench) and passes relevant information to the Network Wallet and the Mining Software. The Network Wallet passes transactions to be signed to the Hardware Wallet and receives them back and then sends the transactions to the Core to be broadcast to the network. The Mining Software gets block details from the Bitcoin Core and passes block headers to the Mining Hardware. It also submits blocks to the network via the Bitcoin Core. The Mining Hardware receives headers and returns valid nonces found by hashing the headers.

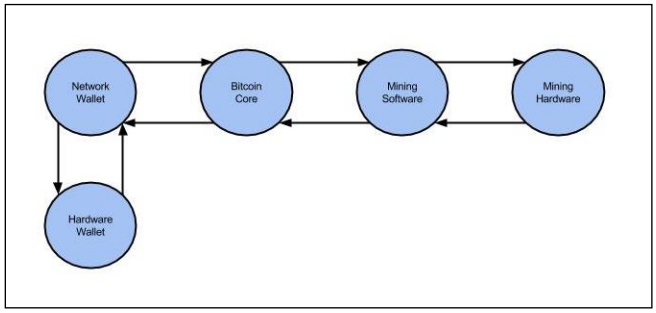

Figure 9: Task Graph

2) *Architectural Graph*

The following architecture graph illustrates a preferred arrangement of processing elements, hardware elements,

and buses. This graph is not intended to suggest that only one arrangement of components is possible, but rather that a pre-defined (common) architecture has already been established, and is suitable for this application.

The heart of the mining node rests with PE1, where the Bitcoin Core and Mining Software tasks are mapped. This PE1 has access to two independent buses, one to a networked PE3 (e.g. Ethernet), and another bus to PE2/HW1 (e.g. USB). The Network Wallet is mapped to the networked PE3, which has a separate, dedicated bus to HW1 (e.g. USB). The Hardware Wallet task operates on HW1, with the intention that it may be physically removed for safe-keeping. Finally, the Mining Hardware task is mapped to PE2/HW1. This component may be a general purpose processor (PE2), or specialized hardware (HW1), running the SHA-256 hashing algorithm to search for a valid transaction block nonce.

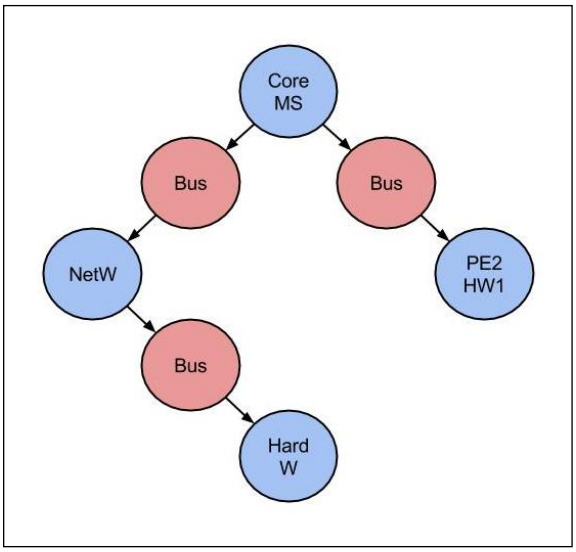

Figure 10: Architectural Graph

D. Transaction-Level Model

The following Transaction Level Model (TLM) is derived from the architecture model. This is where the overall topology of the architecture model can be seen with the respective final communication network. The system specification has been refined to map processes, variables, and channel behaviors onto designated PEs and buses of the system platform, which can be seen below. The TLM is bound to architecture components in multiple ways, providing a set of design points, which can be analyzed using Pareto-optimal techniques. Also, each binding of a TLM to an architecture may be simulated to generate timing information.

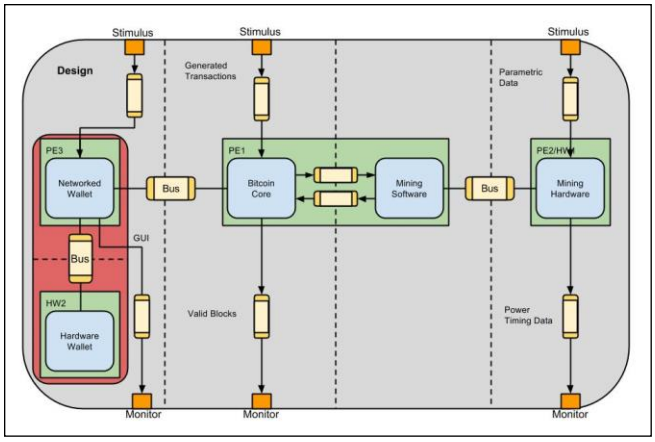

Figure 11: Transaction Level Model (TLM)

IV. Performance Data Collection Strategy

Some of the most useful results produced by system simulation are metrics to support decisions about architecture, behavior mapping, and component selection. This project uses clock speed, instruction cycle counts, and power dissipation for a number of processor types and models, to illustrate how metrics are captured in a simulation, and how the data might be used.

A. Processing Element Profiles

A Processing Element (PE) is profiled in terms of clock speed, pipeline depth, parallel execution paths, power consumption, and cycle counts for each instruction type needed by the Hash behavior. In this context, the term "PE" is being used to represent any component capable of executing the Hash algorithm (CPU, FPGA, ASIC, etc.). This profile data is provided to the Mining Hardware by the Stimulus behavior at the start of a simulation run. The following profile parameters have been defined:

PE Performance:

t_clock	Hertz
t_period	Seconds, 1.0 / t_clock

Memory Operations:

t_mread	Cycle Count, memory read
t_mwrite	Cycle Count, memory write

Composite Operations:

t_call	Cycle Count, function calls

SHA-256 Instruction Set:

t_isum	Cycle Count, int add/sub
t_imul	Cycle Count, int multiply
t_idiv	Cycle Count, int divide
t_shft	Cycle Count, shift, per bit
t_rot	Cycle Count, rotate, per bit
t_band	Cycle Count, bitwise AND
t_bor	Cycle Count, bitwise OR

t_bnot Cycle Count, bitwise NOT
t_bxor Cycle Count, bitwise Ex-OR
t_comp Cycle Count, int compare

Watchdog Timer:
t_timeout Seconds, hash search limit

Power Consumption:
p_pwr_static Watts, idle
p_pwr_dynamic Watts, active

The table below identifies seven processing elements (PEs) that might serve as suitable hosts for the Hardware Miner behavior. The clock, instruction, and power data have been derived from published materials, increasing the fidelity of the simulation metrics.

NOTE: While much of the data in the table below is taken directly from manufacturer specifications or design notes, some of the data points were not clearly defined, and sometimes even conflicted between two sources. While care has been taken to capture the most accurate information possible, these data points should be regarded as "representative" of the processor types and models listed.

	Processing Elements (6)						
	General Purpose Processors (1)(15)			FPGA (7)(8)(17)			ASIC (10)
Operation	AMD Athlon 64	Intel Core i7	VIA Nano L3050	Xilinx Artix-7	Xilinx Virtex-7	Xilinx Kintex-7	Generic (14)
pe_clock (9)	2050	3400	2000	258	377	385	375 (11)
pe_mread (5)	1	1	1	1	1	1	1
pe_mwrite (5)	1	1	1	1	1	1	1
pe_call (3)(4)	7	9	8	6	6	6	4 (12)
pe_isum (3)	1	1	1	1	1	1	1
pe_imul	3	5	3	1	1	1	3
pe_idiv (2)(3)	39	22	22	32	32	32	22 (13)
pe_shft (3)(18)	1	1	1	1	1	1	1
pe_rot (3)(18)	1	1	1	1	1	1	1
pe_band (3)	1	1	1	1	1	1	1
pe_bor (3)	1	1	1	1	1	1	1
pe_bnot (3)	1	1	1	1	1	1	1
pe_bxor (3)	1	1	1	1	1	1	1
pe_comp (3)	1	1	1	1	1	1	1
pe_pwr_static	25W	84W	1W (16)	0.5 W	7W	2W	2W (11)
pe_pwr_dynamic	190 W	184 W	20W (16)	5W	17 W	8W	12W (11)

Table 3: Processing Element Comparison

(1) Source for timing data: [4]
(2) Averaged, using 32 bit operation, (min + max) / 2
(3) Assuming optimized for register/register operations (minimizing memory read/write ops)
(4) Assuming "near" call; includes sum of CALL, PUSH, POP, and RETURN
(5) Assuming one clock cycle for each memory transfer for all devices
(6) Pipelining is not accounted for; all numbers reflect serial execution
(7) Source: [5]
(8) FPGAs are running Xilinx MicroBlaze Core Processor
(9) MHz
(10) Source: [6]
(11) Clock speed and power derived from measurements reported for an actual Bitcoin miner, source: [7]
(12) Not listed in reference; assume four operations (CALL, PUSH, POP, RETURN), all taking one cycle
(13) Division had not been implemented in OpenRISC 1200 at the time the reference document was published; Division has since been implemented, but no performance numbers have been found; Assume the operation could be done at least as fast as any other PE's evaluated
(14) Assume any ASIC supplier could achieve the same results using standard cells
(15) Source for power data: [8]
(16) The reference material did not provide direct comparison of identical VIA processors; The VIA power numbers are for a VIA C7 Esther processor
(17) Source for power data: [9]
(18) Shift and rotate cycle counts are per bit

B. Code Instrumentation

The project is interested in comparing the performance of various types of processing elements used to host the hashing algorithm, therefore the primary focus of metrics collection is the **Hash** behavior. The basic algorithm is derived from an open source bitcoin mining application, written in C. The C code was adapted to the SpecC environment, then analyzed to determine where to insert metrics "instrumentation" code. This resulted in the identification of "code blocks", within which the types of instructions were counted manually. SpecC "waitfor" statements were then inserted for each code block, using the manual instruction counts and the configurable PE profile data. An example from the "hw_hash.sc" file is illustrated below, showing a simple code block containing two bitwise Exclusive OR operations and one bitwise AND:

```
u32 Ch(u32 x, u32 y, u32 z) {
// Begin Instrumentation Block
cycles = t_bxor*2 + t_band;
waitfor(cycles);
power_meter(p_pwr_dynamic, cycles);
// End Instrumentation Block
return z ^ (x & (y ^ z));
}
```

Simulation time accumulates with each call to "waitfor". How this accumulated time is used to extract useful metrics is described in the paragraphs that follow.

C. Simulation Time

SpecC simulation time is basically unitless, and a PE "cycle" is a function of the configurable clock speed. All "waitfor" statements are based on cycle counts, which by themselves have no relation to time. This actually simplifies the collection of metrics, allowing time to be accumulated in a generic manner, with the actual time values only computed when needed, as:

```
real_time = cycles * t_period
```

where t_period is the reciprocal of the clock speed.

D. Time Metrics

A simple time accumulator does not produce useful metrics. Instead, current simulation time must be recorded at key points in the code, and then the differences between recorded times may be used as "elapsed time" for a given operation. Consider the example below:

```
start_proc = now();
g_nonce.status = hash_it_to_pieces();
proc_time = (double)(now() - start_proc) * t_period;
```

In this example, the current simulation time is recorded in "start_processing", and then after returning from the hashing function an elapsed time is computed using the delta simulation time.

E. Power Metrics

Using techniques described in [2], the same code blocks used to collect time are also used to accumulate power. The paper describes two types of power characteristics: static and dynamic. Static power represents PE consumption "at rest" (powered-on but idle). The dynamic power represents PE consumption while the PE is actively hashing a block header. Both cases use elapsed time in a given code block, multiplied by a constant (one constant for static, one for dynamic). Real-world power profiles are of course more complex, but this simplified view of power is a very convenient way to obtain useful estimates for component selection decisions.

Re-visiting the code example from paragraph 4.2, a "power_meter" function is called from within each code block, with a parameter to indicate static or dynamic power. The example below shows a version of the function that uses clock cycles to represent time. A separate version of the function exists to accumulate power for cases where the use of actual elapsed time is more convenient. Both versions of the power_meter function increment a common accumulator.

```
u32 Ch(u32 x, u32 y, u32 z) {
  // Begin Instrumentation Block
  cycles = t_bxor*2 + t_band;
  waitfor(cycles);
  power_meter(p_pwr_dynamic, cycles);
  // End Instrumentation Block
  return z ^ (x & (y ^ z));
 }
```

Static power is applied to the single case where the hashing algorithm is blocked, awaiting a new transaction block header to process, as shown below:

```
start_idling = now();
// Wait for a block header to process (idle time)
c_blk_hdr.receive(&g_blk_hdr,
sizeof(g_blk_hdr));
idle_time = (double)(now() - start_idling) * t_period;
```

Any waitfor statements executed outside of the algorithm will contribute to this "static elapsed time". Dynamic power is applied to all other processing performed by the algorithm. Note that code designed only to support the simulation framework (test bench) is not instrumented at all, avoiding measurement bias.

V. RESUTS

As mentioned earlier, seven different processing elements (PEs) were "profiled" to support metrics collection. The most significant metrics for Bitcoin mining are "million-hashes per joule" (MH/j), representing efficiency, and "million-hashes per second (MH/s), representing speed. These metrics, coupled with per-unit cost, provides mining node developers some very important information to aid in component selection decisions.

The table below summarizes the simulation results for seven PEs. Interestingly, it appears that all general purpose processors outperform all FPGA and ASIC devices, contrary to expectations. The answer is simple: the code used to model the algorithm was written for a general purpose processor, avoiding the advantages offered by FPGA and ASIC devices. In fact, the open source code used to model the algorithm is not time-optimized even for a general purpose processor. The code contains many loops and function calls, which could be "unrolled" to improve timing performance (at the expense of memory space).

Using existing CPU core definitions for FPGA (MicroBlaze) and ASIC (OpenRISC 2000), the hashing algorithm is applied to these devices as if they were general purpose processors. Given their lower clock speeds, it is not surprising that they appear to be slower.

	Processing Elements						
	General Purpose Processors			FPGA			ASIC
Measurement	AMD Athlon 64	Intel Core i7	VIA Nano L3050	Xilinx Artix-7	Xilinx Virtex-7	Xilinx Kintex-7	Generic
MH/joule	0.000274636	0.000437324	0.00245289	0.000401467	0.00058664	0.00127311	0.000897246
MH/second	0.0521298	0.0804372	0.049033	0.0068208	0.0099669	0.0101782	0.0107608

Table 4: PE Power Result Comparison

In addition to the timing results that might be skewed in favor of general purpose processors due to the choice of algorithm, one final consideration is important: cost. Commercially-available mining operations tend to be built upon multiple instances of the hashing component, allowing parallel searches for a valid result, thus increasing the likelihood of success. The unit cost of ASIC devices is a fraction that of general purpose processors, when purchased in quantity. The cost of FPGA devices falls in the middle. These facts suggest that a more rigorous analysis might be useful. The team did examine Verilog code designed specifically for FPGA/ASIC devices, and it appears that code would easily outpace a true general purpose processor. A follow-up effort to explore the modeling of the Verilog code would be an interesting activity to pursue.

VI. References

[1] Alqassem, I.; Svetinovic, D., "Towards Reference Architecture for Cryptocurrencies: Bitcoin Architectural Analysis," Internet of Things(iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing(CPSCom), IEEE , vol., no., pp.436,443, 1-3 Sept. 2014 doi: 10.1109/iThings.2014.78

[2] Samei, Y.; Domer, R., "Automated estimation of power consumption for rapid system level design," *Performance Computing and Communications Conference (IPCCC), 2014 IEEE International* , vol., no., pp.1,8, 5-7 Dec. 2014 doi: 10.1109/PCCC.2014.7017085

[3] Zhong, B.; Ming Feng; Chung-Horng Lung, "A Green Computing Based Architecture Comparison and Analysis," Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom) , vol., no., pp.386,391, 18-20 Dec. 2010 doi: 10.1109/GreenCom-CPSCom.2010.110

[4] Agner Fog, Technical University of Denmark, retrieved 3 May 2015, URL: "http://www.agner.org/optimize/instruction_tables.pdf"

[5] MicroBlaze Processor Reference Guide, Xilinx, retrieved 3 May 2015, URL: "http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf"

[6] OpenRISC 2000 IP Core Processor Specification, Damjan Lampert, opencores.org, retrieved 3 May 2015, URL: "http://www.isy.liu.se/en/edu/kurs/TSEA44/OpenRISC/or1200_spec.pdf"

[7] CryptoMining Blog, retrieved 3 May 2015, URL: "http://cryptomining-blog.com/tag/bitcoin-asic-power-usage/"

[8] Wikipedia, retrieved 3 May 2015, URL: "http://en.wikipedia.org/wiki/List_of_CPU_power_dissipation_figures#Pentium_Dual-Core", and CPU Benchmarks web site by PassMark Software, URL: "https://www.cpubenchmark.net/power_performance.html"

[9] Xilinx Power Efficiency web site, retrieved 3 May 2015, URL: "http://www.xilinx.com/products/technology/power.html"

[10] FIPS 180-2 SHA-224/256/384/512 implementation, 02/02/2007, Con Kolivas <kernel@kolivas.org>, Olivier Gay olivier.gay@a3.epfl.ch, retrieved 22 Apr 2015: URL: "https://github.com/blockerupter/cgminer"

The following source material is acknowledged for providing significant insight and guidance on Bitcoin history, architecture, and code development techniques, though not explicitly cited in the report.

[11] Bitcoin Developer Documentation Websites. Developer-Documentation and Developer-Reference. Bitcoin Foundation 2015, Web. 5 May, URL: "https://bitcoin.org/en/developer-documentation" and "https://bitcoin.org/en/developer-reference"

[12] Bitcoin: A Peer-to-Peer Electronic Cash System by Satoshi Nakamoto