Indian Institute of Technology,
Bombay

VLSI Design Lab

EE705

Accelerated Bitcoin Miner

Authors:
Siddhant Singh Tomar (213079010)
K Akhilesh Rao (213079018)
Dhruva S Hegde (213079022)

Course Instructor:
Dr. Sachin B Patkar

January 7, 2023

1 Objective

The primary objective of this project is to design and implement a fully independent bitcoin mining hardware unit.

The entire project can largely be divided into 2 stages.

1. First step is to design a custom circuit that hashes 4 different input strings using the SHA-256 algorithm with a common data-path in a pipelined manner. This way, multiple hash functions can be obtained at one shot using the same circuit, which increases its throughput and hence accelerate the process of bitcoin mining.

2. Second step is to use this circuit as a unit cell in a pipelined manner to implement a time efficient bitcoin mining unit.

SHA-256 algorithm is arguably the most widely used algorithm in the world today due to the rise in popularity of crypto-currency.

Therefore, the main motivation for this project is to learn about the blockchain, the process of bitcoin mining and how the SHA-256 algorithm works, and to implement the same efficiently using digital hardware modeling techniques.

2 Theory

This sections gives a brief overview of the background details regarding the blockchain, bitcoin mining process and the SHA-256 algorithm.

2.1 What is a Blockchain?

The blockchain can be thought of as a file that contains a list of every bitcoin transaction ever made. Everyone on the bitcoin network shares a copy of this file, and it gets updated regularly with the latest transactions.

Every node on the bitcoin network shares information about new transactions. They store these transactions in their memory pool. Each node also has the option to try and "mine" the transactions in their memory pool in to the blockchain.

2.2 How to mine a Bitcoin?

The information regarding a new transaction in a memory pool is stored in a block header. In order to the add transaction data (or block) from the memory pool to the blockchain, a node has to hash this block header. The node that performs this task will be rewarded with bitcoins.

Block header format

The block headers have a total length of 80 bytes (640 bits) which are divided into 6 fields which provide details of the block.
Figure 1 shows the 6 fields present inside a block header, along with their sizes and types.

Field	Size	Type
Version	4	int32_t
Previous Block Hash	32	char[32]
Merkle Root	32	char[32]
Timestamp	4	uint32_t
Difficulty Target	4	uint32_t
Nonce	4	uint32_t

Figure 1: Block header fields and sizes (in bytes)

The details regarding each of these fields are not essential for the project. However, most of them are self-explanatory. The important field for mining is the Nonce field, which is the only variable field in the block header.

Essentially, a bitcoin miner needs to get this block data and hash it using SHA-256 algorithm twice. If the hash value meets the given constraint (which is usually that the hash should contain 'x' number of leading zeros i.e the magnitude of the hash should be less than the difficulty target), then the block will be successfully added to the blockchain and the miner will earn bitcoins.

Since the hashed output of SHA-256 algorithm is impossible to predict, the only way to obtain 'x' number of leading zeros is by brute force i.e by trying different Nonce bits. This is why an accelerator that can compute SHA-256 hash for multiple inputs really fast is essential in order to add blocks to the blockchain and mint bitcoins.

2.3 SHA-256 Algorithm

SHA-256 is a member of the SHA-2 cryptographic hash functions designed by the National Security Agency (NSA).
SHA stands for Secure Hash Algorithm.

This algorithm works with 512 bits of data at a time. It produces a 256 bit hash code for the given input data. Any input message longer than 512 bits needs to be split (and padded with zeros) and fed into the algorithm sequentially.

Figure 2 represents the SHA-256 hashing for an input message whose length is greater than 512 bits (and less than 1024 bits).

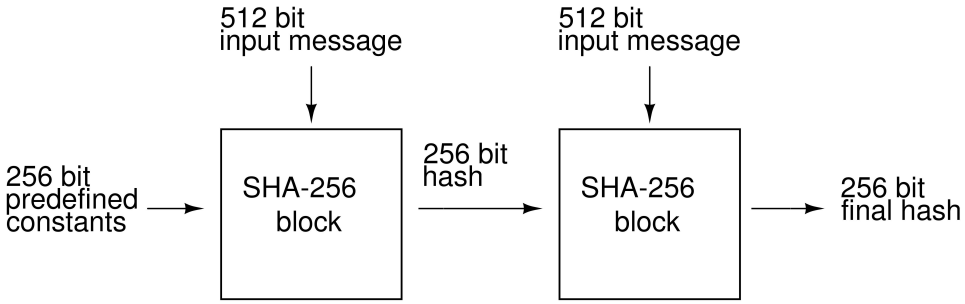

Figure 2: SHA-256 hashing process

The most important feature of SHA-256 is that even a small change in the message will (with overwhelming probability) result in a vastly different hash, due to the avalanche effect. Hence, one-way hash can be generated from any piece of data, but the data cannot be generated from the hash.

This is the reason it is used as the proof of work algorithm for adding a new block to the blockchain (and hence bitcoin mining).

Inner workings of the SHA-256 algorithm:

As explained before, the hashing algorithm takes 512 bits of input message at a time and produces a 256 bit hash. Out of these 512 bits of input, the first 448 bits consist of the message and the rest 64 bits are reserved to specify the length of the actual message.

If the actual message is greater than 448 bits, then the input to the hashing algorithm must be 1024 bits long, with the length of the actual message specified in the last 64 bits (and rest of the space must be padded with zeros). Same logic applies if the actual message length is greater than 960 bits too and so on.

The entire algorithm can be divided into 2 processes.

1. Message Expansion
2. Message Compression

2.3.1 Message Expansion

The process is expansion is straightforward. The 512 bit input data can be represented as 16 words of 32 bit each. As the name suggests, the process takes these 16 words and expands them into 64 words. It takes 64 iterations.

For the first 16 iterations, each of these 16 words are directly mapped as the first 16 words of the 64 word expanded block.
In the next 48 iterations, new words are generated as functions of previously generated words.

The process of expansion can be mathematically represented as follows.

for $i = 0$ to 15 : $w(i) = m(i)$
for $i = 16$ to 63 : $w(i) = \sigma_1(w(i-2)) + w(i-7) + \sigma_0(w(i-15)) + w(i-16)$
w is the expanded message and m is the input message

Note that since everything is being operated on 32 bit data, the addition will be mod-32 addition (i.e carry bits are always ignored).

The 2 small sigma functions are defined below.
$\sigma_0(x) = ROTR\ (x, 7)\ \oplus\ ROTR\ (x, 18)\ \oplus\ SHR\ (x, 3)$
$\sigma_1(x) = ROTR\ (x, 17)\ \oplus\ ROTR\ (x, 19)\ \oplus\ SHR\ (x, 10)$

Hence, the combinational blocks needed for message expansion are : mod-32 Adders and 32-bit Xor gates. Since rotate and shift operations are fixed, they do not need additional hardware.

2.3.2 Message Compression

The process of compression is more involved. The following points explain compression step by step.

- To start, 8 constants of 32 bits each are initialized. These constants (a, b, c, d, e, f, g, h) are obtained from the fractional parts of cube roots of prime numbers.

- These constant values are updated every cycle for 64 cycles using the following logic.

  $T_1 = h \ + \ \Sigma_1(e) \ + \ Ch(e, f, g) \ + \ K(i) \ + \ w(i)$
  $T_2 = \Sigma_0(a) \ + \ Maj(a, b, c)$
  $h = g \ ; \quad g = f$
  $f = e \ ; \quad e = d + T_1$
  $d = c \ ; \quad c = b$
  $b = a \ ; \quad a = T_1 + T_2$

  Here, K values are obtained from fractional parts of first 64 square roots of prime numbers and w values are the 64 words obtained from expansion.

  The big sigma functions, the majority function and the choice function are defined as follows.
  $\Sigma_0(x) = ROTR\ (x, 12) \ \oplus \ ROTR\ (x, 13) \ \oplus \ ROTR\ (x, 22)$
  $\Sigma_1(x) = ROTR\ (x, 6) \ \oplus \ ROTR\ (x, 11) \ \oplus \ ROTR\ (x, 25)$
  $Ch(x, y, z) = x \ ? \ z : y$
  $Maj(x, y, z) = (x \& y) \mid (x \& z) \mid (y \& z)$

- After 64 cycles of updating, the final values of a, b, c, d, e, f, g, h and added to their respective initial values (mod-32 addition).

- These added outputs are then concatenated to obtain the final hash. 8 words of 32 bits each concatenated gives the 256 bit hash function.

3 Components Used

Software :

- Intel Quartus Prime Lite - VHDL programming interface
- ModelSim-Altera - for simulation, testing and debugging
- TimeQuest Timing Analyzer - to perform static timing analysis
- QFlow - to generate layouts of some important components
- Magic - to visualize generated layouts
- XCircuit - to draw block diagrams
- Draw.io - to draw block diagrams
- LucidChart - to draw flowchart
- Overleaf - to prepare report
- Paint - to edit images and diagrams

4 SHA-256 Hardware

This section explains the step by step implementation of SHA-256 algorithm in hardware.

4.1 Primitive Implementation

The most basic and easy way to implement SHA-256 algorithm is to first perform message expansion on the input message independently and then perform message compression on the expanded result to obtain the hash.

The expansion block takes 64 iterations and the compression block also takes 64 iterations. So if they are performed independently, SHA-256 algorithm will take 128 cycles.

Critical paths (i.e paths with worst case delay)-

- Expansion block:

$$T_{exp} = T_{\sigma} + 2\ T_{adder}$$

- Compression block:

$$T_{comp} = \max(T_{\Sigma}, T_{Ch}) + 3\ T_{adder}$$

  Typically, T_{Σ} will be greater than T_{Ch}.

The worst case delay for the expansion block will be less than that of the compression block, but in order to run the circuit under the same clock, the compression block delay has to be considered.

Therefore, the time period of the clock must be greater than T_{comp}. This will give the maximum frequency of operation.

$$f_{clk}\big|_{max} = \frac{1}{T_{\Sigma} + 3\ T_{adder}}$$

Assuming the time period of the clock is $T_{clk} = T_{\Sigma} + 3T_{adder} + T_{overhead}$, the time taken to obtain 1 hash function will be 128 T_{clk}.

4.2 Accelerated SHA-256 Hardware

It is quite evident that the primitive implementation is not optimal. Most of the combinational blocks are idle for a lot of time. Hence, there is room for pipelining.

The proposed pipelined data-path consists of a 4-stage pipeline. [2]
The combinational logic delay of each stage is approximately T_{adder} (it will be slightly more, but the delay is dominated by the adder delay).

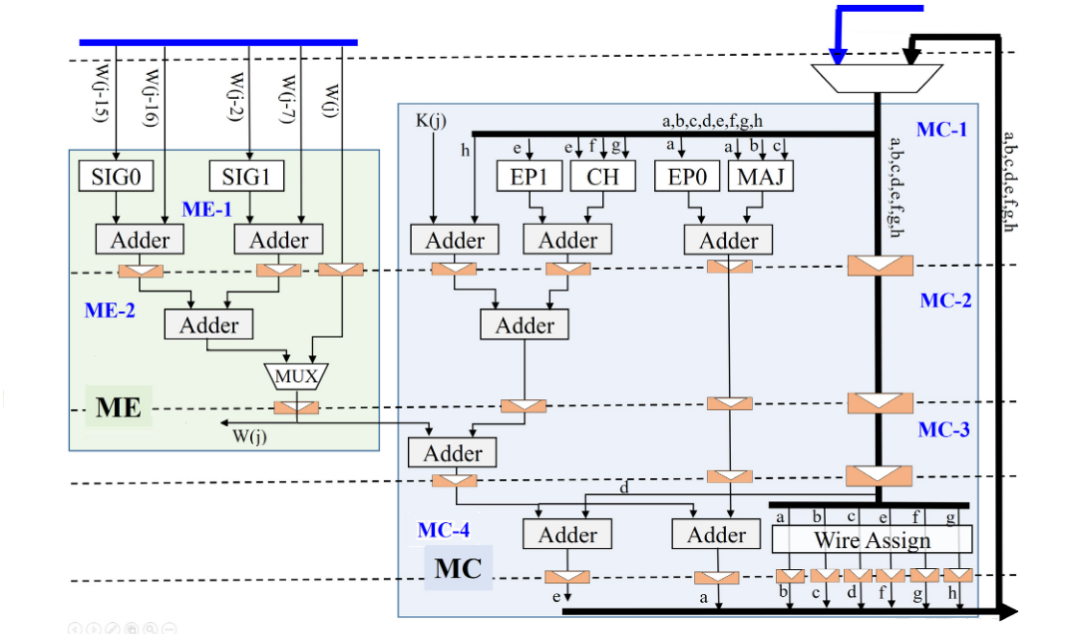

Figure 3: 4 stage pipelined data-path

Message expansion consists of 2 stages : ME-1 and ME-2.
Message compression consists of 4 stages : MC-1, MC-2, MC-3 and MC-4.

Note that ME-1 and MC-1 run in parallel, and so do ME-2 and MC-2. Hence effectively there are a total of 4 pipelined stages.

Because of the 4-stage pipeline, 4 hashes can be computed for 4 different message words using the same hardware. This vastly increases the throughput of the circuit.

Analysis of pipelined data-path-

- Number of clock cycles taken to compute a single hash function = 256.
- Number of clock cycles taken to compute 4 hash functions = 259.
- Maximum propagation delay of any pipeline stage = T_{adder}+max($T_\sigma, T_\Sigma, T_{Ch}, T_{Maj}$).

Comparison with primitive circuit to measure performance improvement.

- The maximum clock frequency has increased to almost thrice (because 1 adder delay instead of 3 adders delay).
- The number of clock cycles taken has increased by a bit more than 2 times (from 128 to 259).
- The number of hash calculated has increased by 4 times.

$\implies$ Performance improvement = $4 \times 3/2 = 6$.

To implement the hashing using the above technique, other blocks apart from just the ALU (data-path) are necessary.

The efficient SHA-256 hardware block consists of 4 parts -

1. Pipelined ALU
2. Scheduler
3. Extractor
4. Controller

Figure 4 shows the block diagram of the SHA-256 Accelerator unit consisting of all the 4 parts and the input messages.

The functions and working of each of these blocks is explained in detail in the following subsubsections.

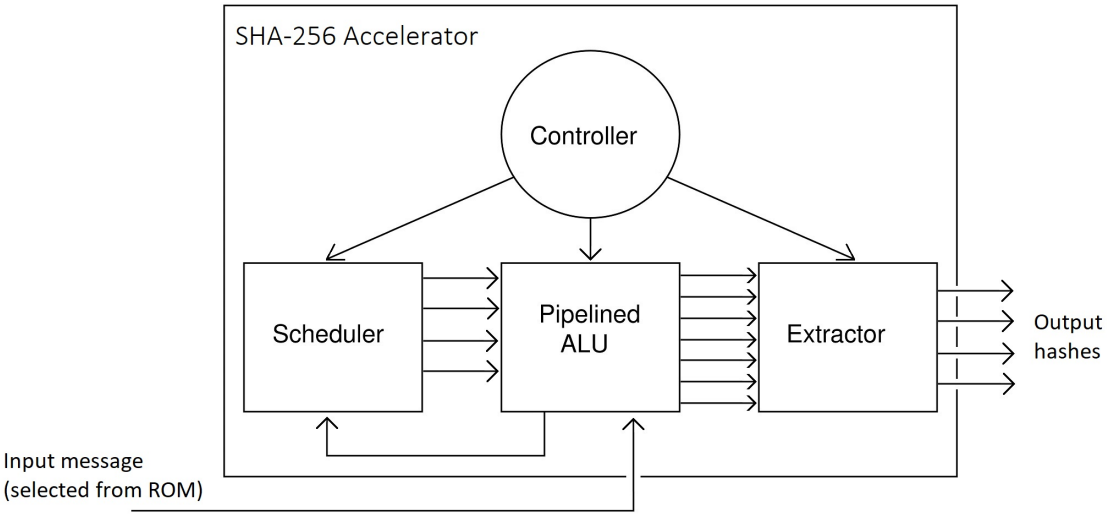

Figure 4: SHA-256 Accelerator Unit Block Diagram

4.2.1 Pipelined ALU

As explained earlier, the work of the pipelined ALU is to perform message expansion and message compression in parallel.

Figure 3 (shown earlier) illustrates the pipelined ALU. It takes message input directly as well as 4 inputs from the scheduler. The expansion block outputs the expanded message, which goes back to scheduler for storage while the compression block outputs the updated values of (a, b, c, d, e, f, g, h).

It has been established that the combinational delay is dominated by the 32-bit modulo-32 adder.
In order to optimize the design, the adder delay must be minimized. This is achieved using **32-bit Brent-Kung Adder**.

4.2.2 Scheduler

The Scheduler block is responsible for sending the required data into the pipelined ALU.

The expansion process converts 16 input message words to 64 expanded message words. Since there are 4 different inputs being dealt with, there are 4 register files consisting of 64 registers each (who are all 32 bits long) in order to store the expanded message words.
The scheduler takes the expansion message words from pipelined ALU and stores them in registers. It also sends in the required message words i.e $w(j-16)$, $w(j-15)$, $w(j-7)$ and $w(j-2)$ from the register files to the pipelined ALU.

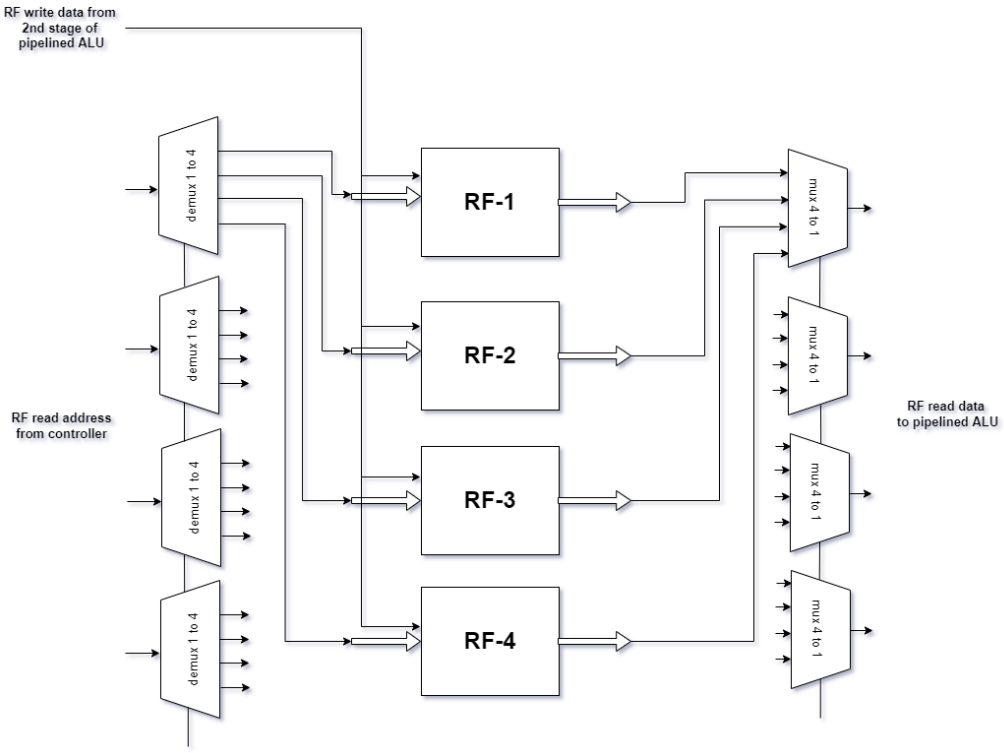

Figure 5: Scheduler block diagram

The block diagram in Figure 5 illustrates the structure of the scheduler.

4.2.3 Extractor

The Extractor block receives final outputs of the pipelined ALU and adds them one by one to the initial constant values.

The compression process outputs updated (a, b, c, d, e, f, g, h) values which need to be added to their initial values respectively and stored. Since there are 4 different inputs being dealt with, there are 4 sets of updated values of (a, b, c, d, e, f, g, h), which means 4 register files consisting of 8 registers each (who are all 32 bits long) are needed to store the final values.
The extractor takes in compressed output i.e updated values of (a, b, c, d, e, f, g, h), adds them to the initial values of (a, b, c, d, e, f, g, h) respectively and stores them in the required registers.
These 8 registered values of each of the 4 register files concatenated will give the 4 required hash functions.

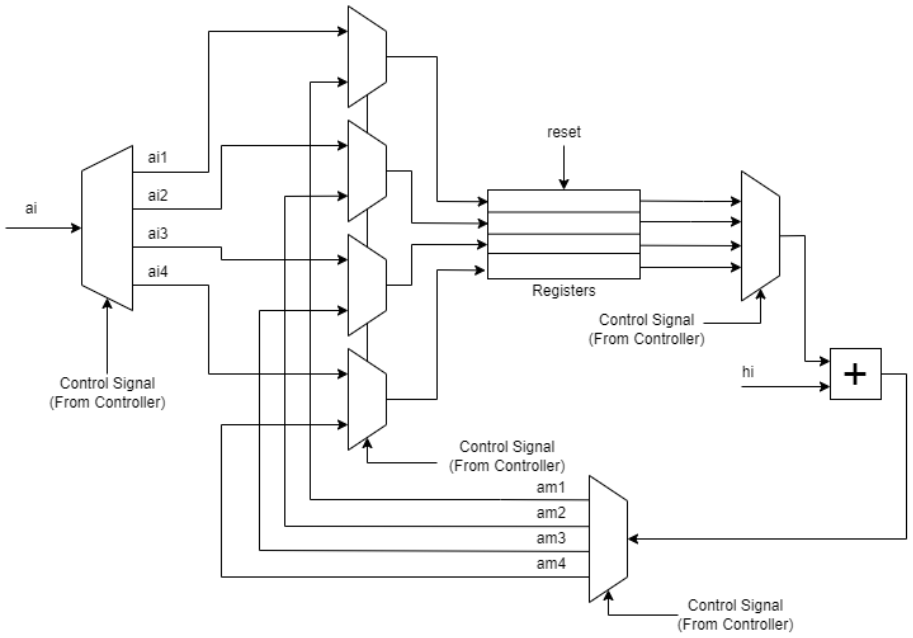

Figure 6: Extractor block diagram

The block diagram in Figure 6 illustrates the structure of 1 part of the extractor. There will be a total of 8 such structures in the extractor.

4.2.4 Controller

The controller provides control signals to all the other 3 blocks based on the count value generated by a global counter.
The global counter is a mod-260 up counter.

The control signals generated are listed-

- For scheduler:
  - $c1, c2, c3, c4$ - Register read addresses
  - md_sel - Select lines for the multiplexers and de-multiplexers
  - reg_wrt_vec - Specifies which of the 4 Register files to write into
  - wr_addr_common - Register write addresses
- For extractor:
  - $ext\_demux\_sel1, ext\_demux\_sel2$ - Select lines for de-multiplexers
  - $sel\_f1, sel\_f2, sel\_f3, sel\_f4$ - Select lines for multiplexers
  - $enable\_1, enable\_2, enable\_3, enable\_4$ - Enable signals to each of the 4 register files
- For pipelined ALU:
  - exp_mux_sel - Select between $w(j)$ [before 16 cycles] and expansion ALU output [after 16 cycles]
  - $init$ - Select between constants [only 1st cycle] and updated values [after 1st cycle] for compression
  - ext_on - Indicator for extractor to start working on the ALU data

Apart from these 4 blocks, a hardwired test-bench is necessary to fetch the relevant input message word from the ROM (consisting of all the 4 messages) and deliver to the pipelined ALU.
This test-bench action is also based on the global counter that is used for the controller.

The ROM is consists of 64 locations, each of 32 bits long. It can store 4 messages. It is illustrated in Figure 7.

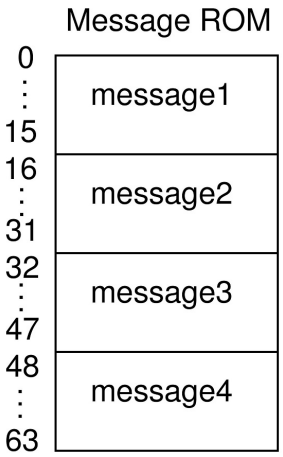

Figure 7: Message ROM

For the SHA-256 accelerator, the test-bench has to pick up 32 bit message words from the ROM in the given pattern : 0, 16, 32, 48, 1, 17, 33, 49, 2, ...

Figure 8 shows the RTL view of the realized SHA-256 Accelerator hardware.

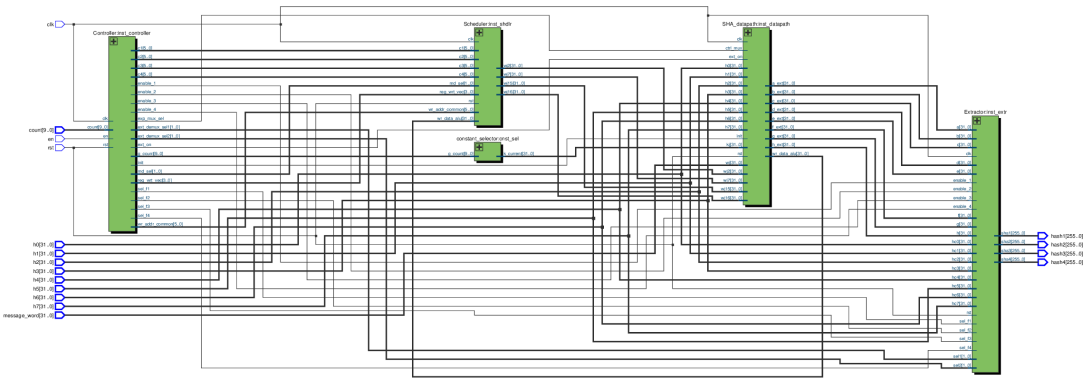

Figure 8: SHA-256 Accelerator Unit RTL View

5 Bitcoin Miner

This section goes into the intricate details of the bitcoin mining process and how to use the designed SHA-256 accelerator unit to mine bitcoins.

5.1 Process of bitcoin mining

It is known that a block header is 80 bytes long (640 bits). This is too long to fit into a single message input to the SHA-256 algorithm. Hence, 2 input messages each 512 bits long are necessary.

Figure 9 illustrates how the block header is sliced into 2 messages of 512 bits each in order to feed to the SHA-256 algorithm.

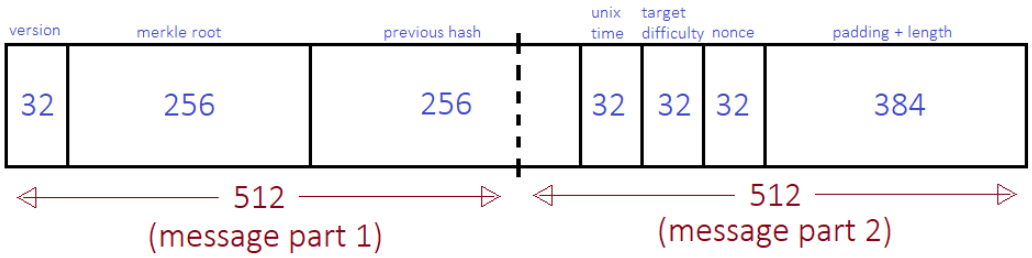

Figure 9: Block Header as Message

As explained earlier, when the message is longer than 512 bits, the algorithm has to run more than once. In this case, the algorithm has to run twice. For the second block, the hash of the first block (sliced accordingly) has to be fed instead of constant (a, b, c, d, e, f, g, h) values.

Bitcoin mining requires another SHA-256 block to hash the hash of the block header. $\implies final\ hash = SHA256\ (SHA256\ (block\ header))$

Since the hash of the block header is only 256 bits long, it can be hashed again using SHA-256 only once (as opposed to twice with the block header).

The process of bitcoin mining is illustrated in the flowchart (Figure 10).

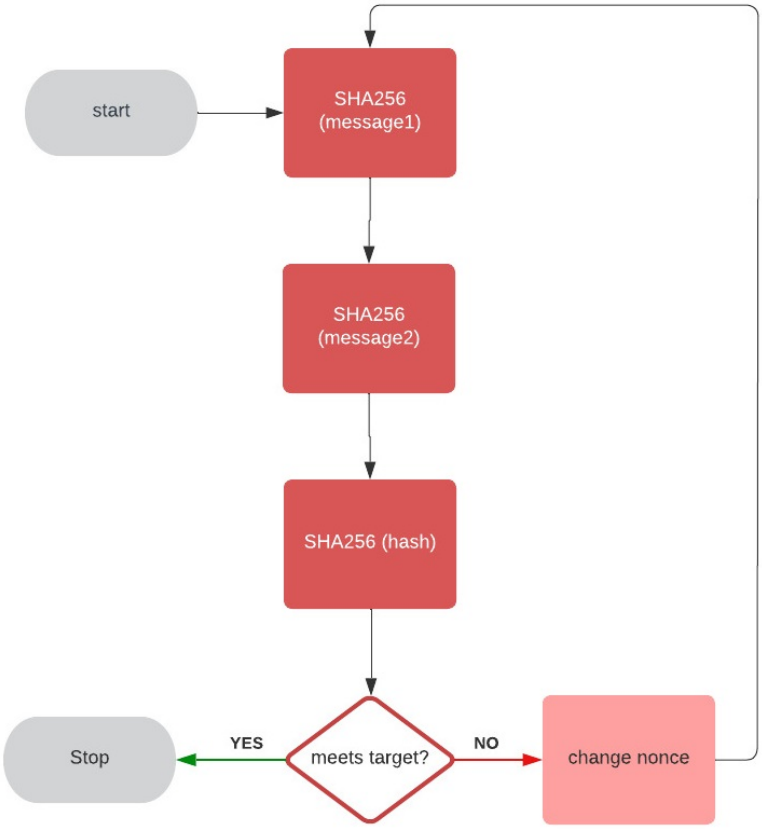

Figure 10: Bitcoin mining process Flowchart

5.2 Hardware for bitcoin mining

The process of bitcoin mining has been explained. This can be done using a single hardware unit or by using 3 cascaded hardware units.

In cascaded version, each unit needs input from previous unit to calculate a hash function. Hence, both these methods will take the same time to calculate one final hash. However, in the case of bitcoin mining, it is necessary to try to keep hashing different input messages till required hash is obtained. One time hashing is not enough (unless you get extremely lucky!!).

For this reason, cascaded version is used in a pipelined manner so that new output hashes can be obtained after every 260 clock cycles (ignoring latency), as opposed to 780 clock cycles if a single hardware was used. [1]

Figure 11 shows the block diagram of the bitcoin mining unit.

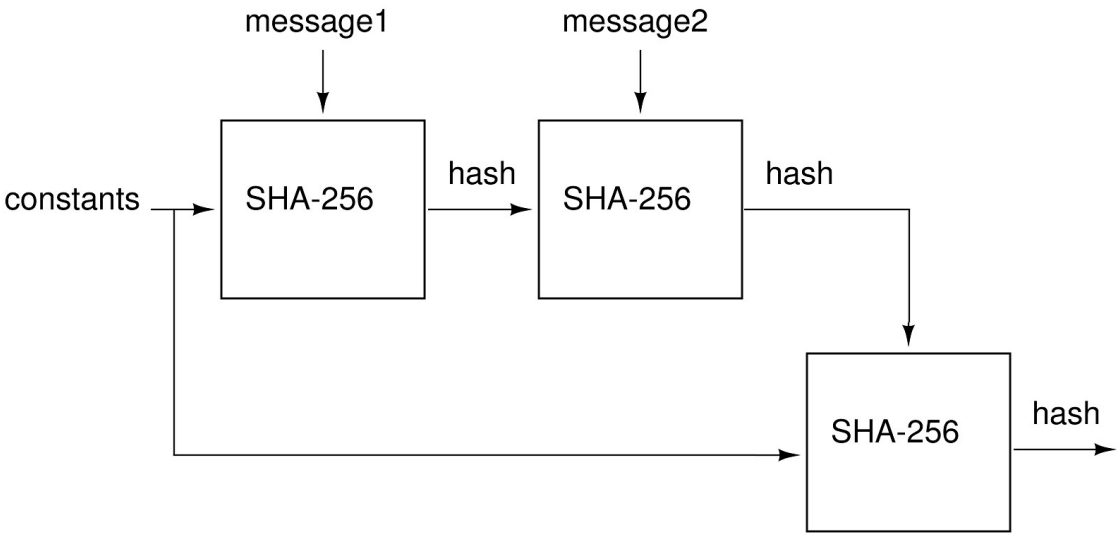

Figure 11: Bitcoin Mining Unit Block Diagram

Figure 12 shows the RTL view of the bitcoin mining unit.

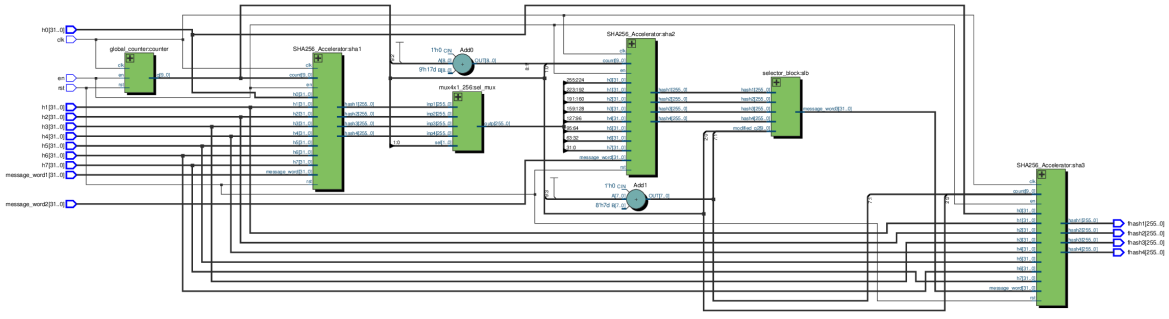

Figure 12: Bitcoin Mining Unit RTL View

6 Results

This section shows the simulations performed and the results obtained.

6.1 SHA-256 Accelerator

Simulations for the SHA-256 accelerator block is for logic verification only, since timing analysis is anyway done for the overall bitcoin miner hardware.

6.1.1 RTL Simulation

4 sample message bits (each of 512 bits length) have been given as input to a single SHA-256 accelerator unit. The below figure (Figure 13) shows the respective RTL simulation.

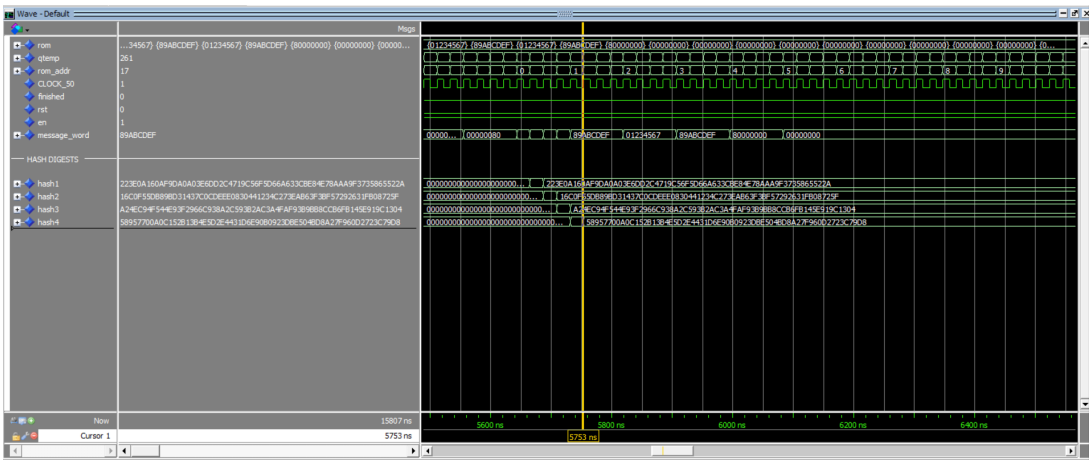

Figure 13: Accelerator : RTL simulation

The accelerator unit is running at a frequency of around 46 MHz, which is derived after STA of the entire miner (illustrated in the coming subsection). Since it takes 260 clock cycles, the speed at which the accelerator generates 4 hash functions is given by,

$$T = \text{no. of cycles} \times \frac{1}{f_{clk}} = 260 \times \frac{1}{46 \times 10^6} \approx 5.65\ us$$

The following figures (Figure 14 and Figure 15) show the execution of the same 4 hashes on CPU and on GPU using optimized Python function.

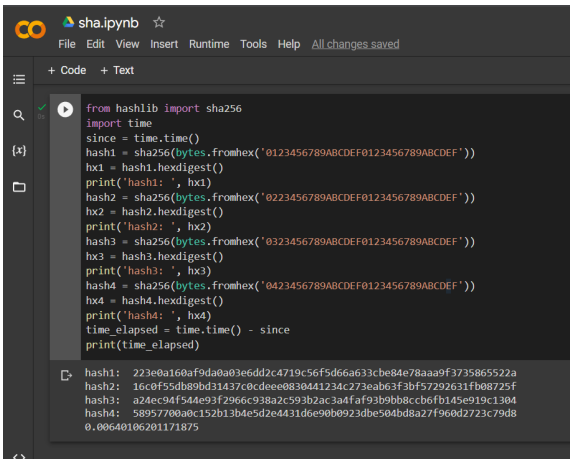

Figure 14: Python code - CPU

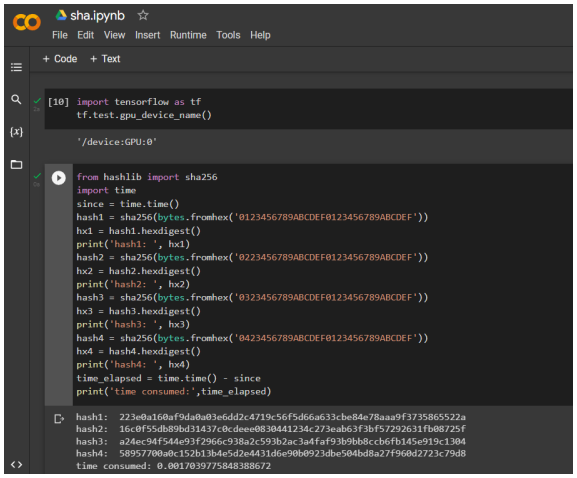

Figure 15: Python code - GPU

Hence, the designed hardware block outperforms both CPU and GPU. The accelerator is over a 1000 times faster than CPU and is around 300 times faster than GPU.

6.1.2 Layouts

Layouts for some of the important components used in SHA-256 accelerator have been generated using QFlow and visualized using Magic.

All the combinational components of the accelerator data-path are listed below, along with their layouts.

32-bit Adder:

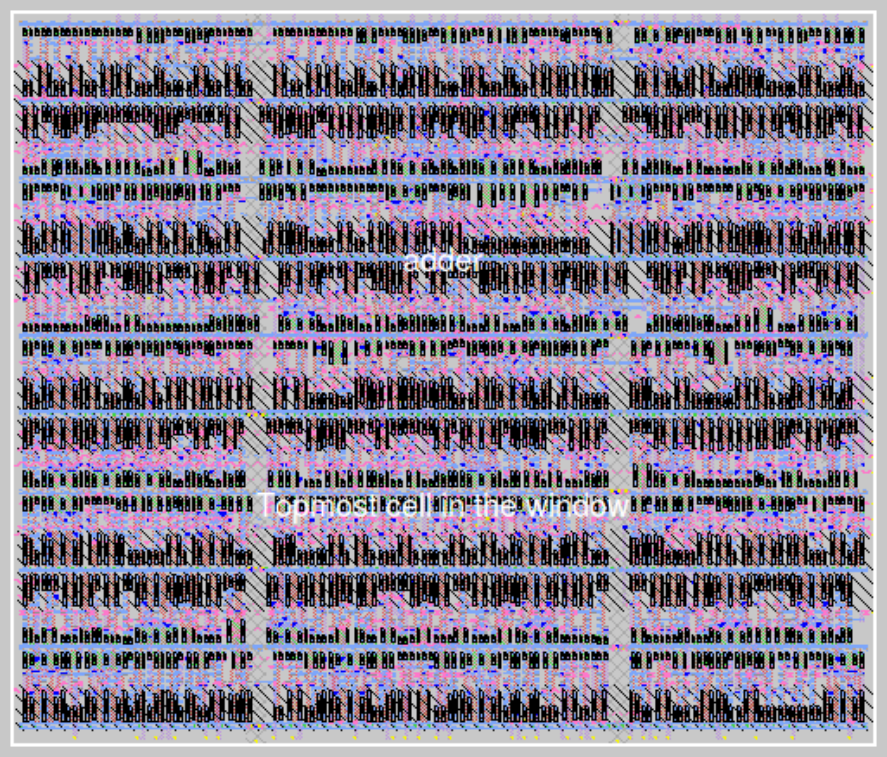

Figure 16: 32-bit Adder layout

(The accelerator data-path uses 10 such adders)

σ_0 **function**:

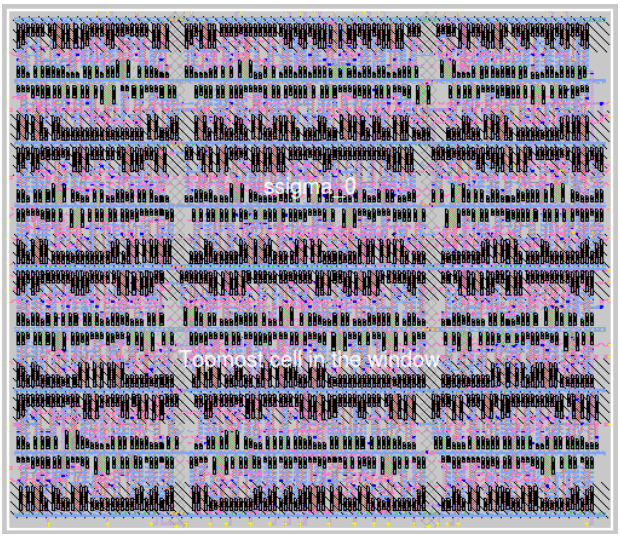

Figure 17: σ_0 function layout

σ_1 **function**:

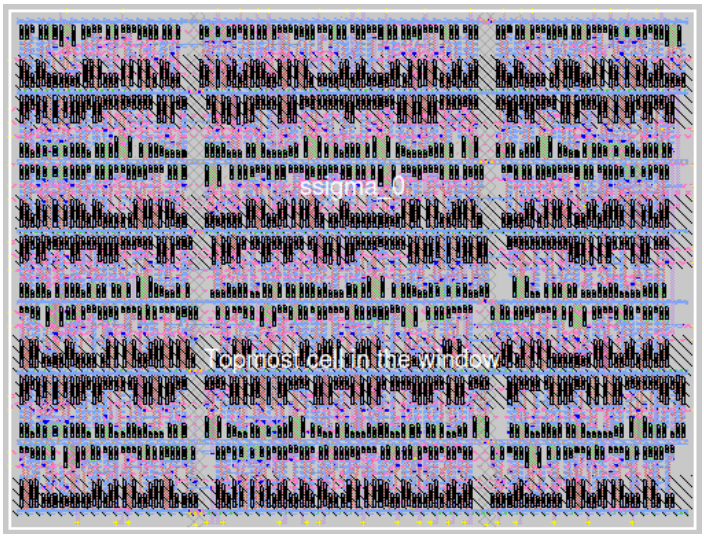

Figure 18: σ_1 function layout

Σ_0 **function**:

Figure 19: Σ_0 function layout

Σ_1 **function**:

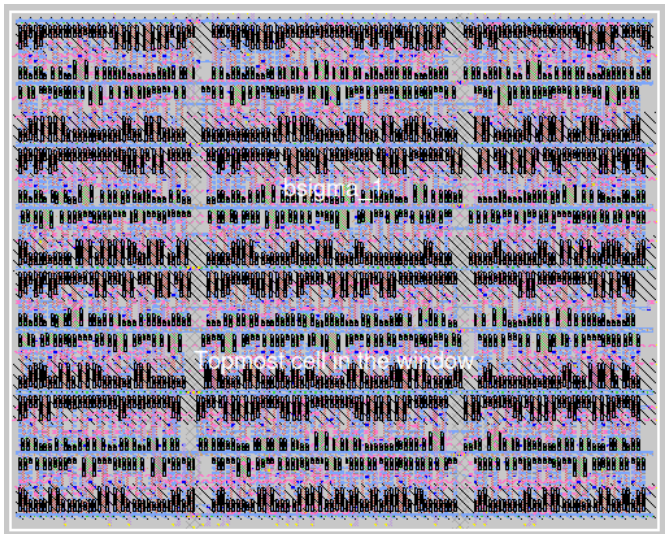

Figure 20: Σ_1 function layout

Choice function:

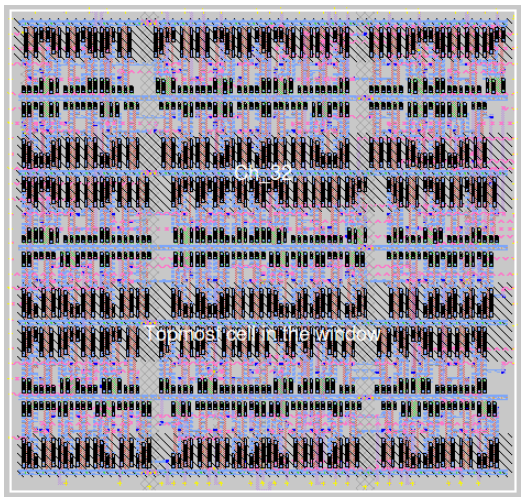

Figure 21: Choice function layout

Majority function:

Figure 22: Majority function layout

The controller is generated using the mod-260 up counter, hence the layout of the counter has been generated below.

Mod-260 up counter

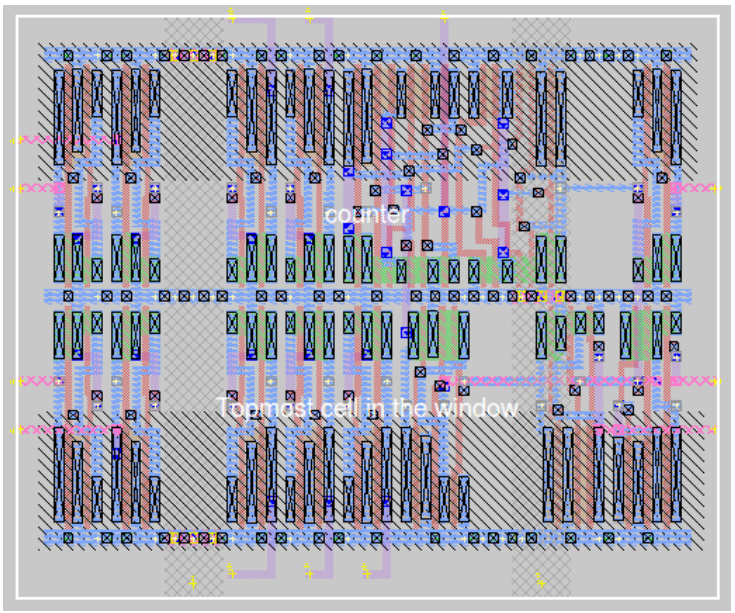

Figure 23: Mod-260 Counter layout

6.2 Bitcoin Miner

This subsection consists of logic verification, timing verification and post timing analysis.
The bitcoin miner hardware is compiled using Cyclone IV GX : EP4CGX110DF31C7.
The Chip Planner view and its details are shown in Figure 24.

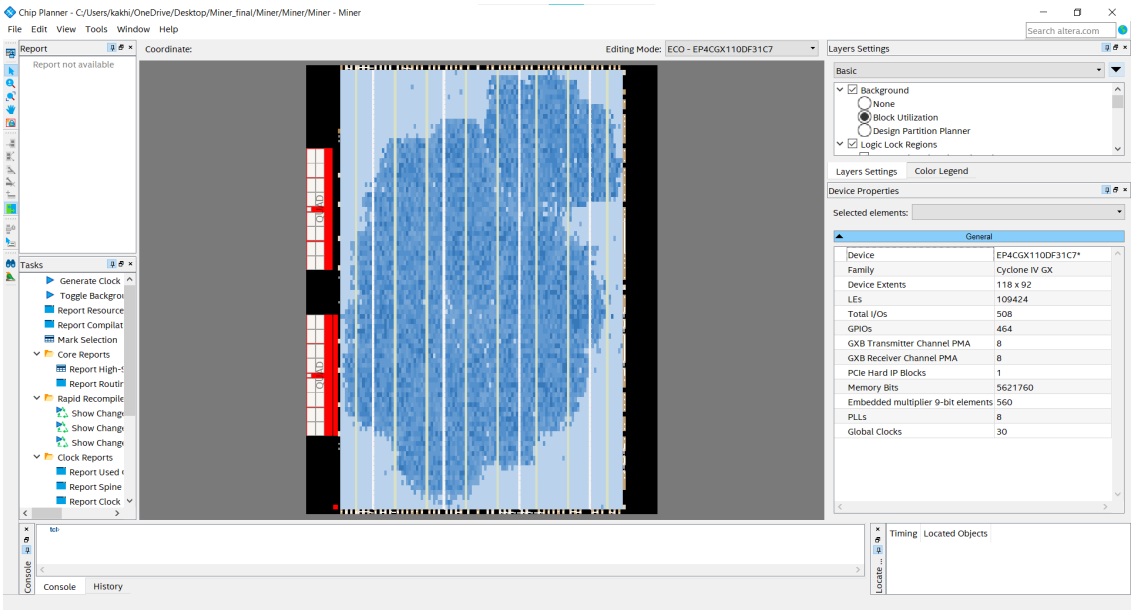

Figure 24: FPGA Chip planner

6.2.1 RTL Simulation

For the purpose of illustration, an already mined block header has been used.

```
02000000 ............................ Block version: 2

b6ff0b1b1680a2862a30ca44d346d9e8
910d334beb48ca0c0000000000000000 ... Hash of previous block's header
9d10aa52ee949386ca9385695f04ede2
70dda20810decd12bc9b048aaab31471 ... Merkle root

24d95a54 ............................ Unix time: 1415239972
30c31b18 ............................ nBits
fe9f0864 ............................ Nonce
```

Figure 25: Sample block header

The same block header is fed to the test bench, but with lower nonce values. The miner will find the hashes for the initial nonce values, then increment them and find the hashes again till the required nonce (and hence the required hash) is found.

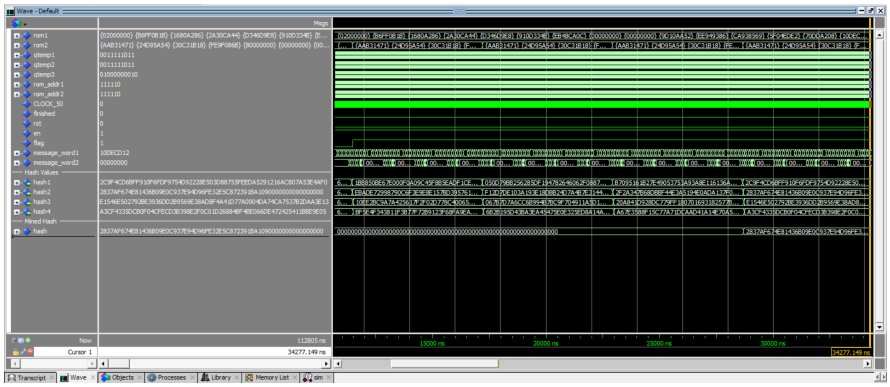

Figure 26: Miner : RTL simulation

The miner outputs only the hash that satisfies the required target. Note that the mined hash has 16 leading zeros.

6.2.2 Static Timing Analysis

STA has been performed to ensure that there are no negative slacks. The clock period has been fixed at 21.7 ns.

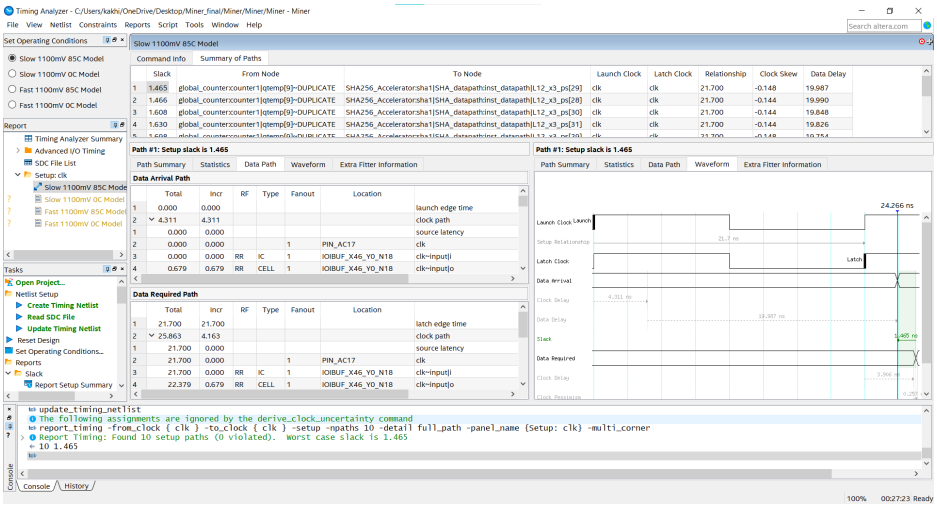

Figure 27: Timing Analysis Results

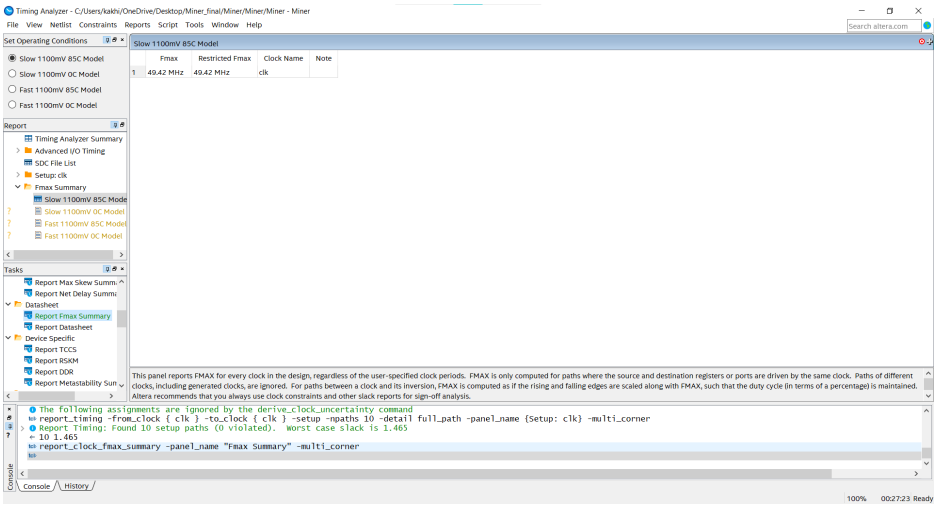

Figure 28: Maximum Clock Frequency

Therefore, the bitcoin miner can run at a clock frequency of about 49 MHz.

6.2.3 Gate Level Simulation

Gate level simulation for the same block header has been performed, after performing STA.

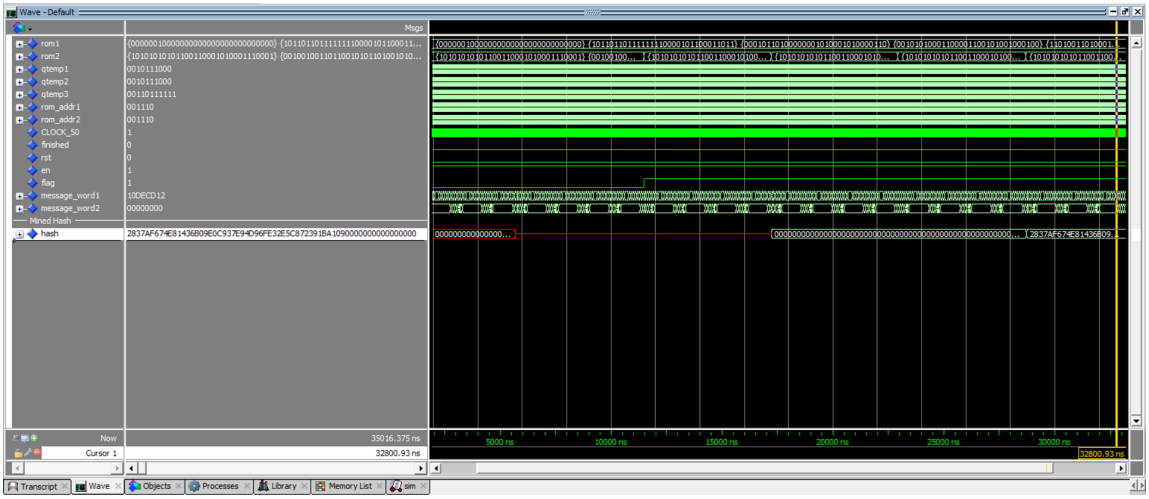

Figure 29: Miner : Gate level simulation

6.3 Conclusion

As proposed, the computation of SHA-256 has been accelerated when compared to that of a CPU or GPU; and the accelerator has been used to build a bitcoin miner.

The total number of possible nonce values is $2^{32} \approx 4.29 \times 10^9$. So in the worst case scenario, the miner will take $4.29 \times 10^9 \times 5.65 \times 10^{-6}/4 = 6067$ seconds (i.e less than 2 hours) to get the required hash.
Since the bitcoin mining time limit is predefined as 10 minutes (i.e 600 seconds), replicating the proposed hardware is necessary. 10 such hardware units running in parallel can pretty much always guarantee that the required hash function is found with and mine 6.25 bitcoins (this number is subjected to change with time).
If even more such units are running in parallel, the process becomes faster.

Improvements that can be made:

- The controller has been designed using behavioral style. It might be possible to optimize the speed by customizing the controller design using structural style.
- Power consumption has not been optimized in this design. Some power saving techniques such as clock gating can be implemented in order to save power, since mining hardware typically consume a lot of power.

7 Contributions

Siddhant Singh Tomar :

- Design of pipelined data-path
- Design of scheduler block
- Design of hardwired test bench
- Python code for verification

K Akhilesh Rao :

- Design of compression process
- Design of controller block
- Design of global counter
- Static timing analysis

Dhruva S Hegde :

- Design of expansion process
- Design of extractor block
- Design of pipelined miner unit
- Layouts of important components

Testing, debugging, drawing block diagrams, writing report and demo videos were done by all 3 together.

Special thanks to T.A, Harsh Paryani for guidance.

References

[1] Le Vu Trung Duong, Nguyen Thi Thanh Thuy, and Lam Duc Khai. A fast approach for bitcoin blockchain cryptocurrency mining system. *Integration*, 74:107–114, 2020.

[2] Thi Hong Tran, Hoai Luan Pham, and Yasuhiko Nakashima. A high-performance multimem sha-256 accelerator for society 5.0. *IEEE Access*, 9:39182–39192, 2021.

Websites referred for background information and understanding:

- `https://learnmeabitcoin.com/`
- `https://en.bitcoin.it/wiki/Main_Page`

Content covered and resources provided in the courses EE671, EE721, EE705 and EE739 were vital for the design and implementation of the project.