

Final report

SHA-256 is and Bitcoin hashing explanation:

The final project part 1 is an implementation of the SHA-256 hashing algorithm in System Verilog. SHA-256 is a cryptographic hash function that takes an input message and produces a fixed-size (256-bit) hash value that is typically represented as a sequence of eight 32-bit words.

The final project part 2 is an implementation of the Bitcoin hashing algorithm in System Verilog, which is designed to perform Bitcoin hashing using chained SHA-256 algorithms. The goal is to hash an input message, which consists of 20 words (19 words and 1 nonce value) and return an output of eight 32-bit words.

SHA-256 and Bitcoin hashing algorithm:

description:SHA-256 algorithm description:

Module Definition and Parameters:

The module is named `simplified_sha256`. It takes various input signals like `clk`, `reset_n`, `start`, `message_addr`, and `output_addr`. It provides output signals such as `done`, `mem_clk`, `mem_we`, `mem_addr`, `mem_write_data`, and `mem_read_data`. `NUM_OF_WORDS` (default is 20) is a parameter that specifies the number of words in the input message.

Finite State Machine (FSM):

The module uses an FSM to control its operation. The FSM has several states:

IDLE: The initial state where the module is waiting for a start signal.

LOAD: Reads the input message from memory into the module.

READ: Reads data from memory.

BLOCK: Processes data in 512-bit blocks.

WAIT: Prepares data for the next block.

COMPUTE: Performs SHA-256 hash computations.

WRITE: Writes the final hash value back to memory.

Local Variables:

The module declares various local variables, including message storage, hash values (`h0` through `h7`), loop counters (`i` and `j`), offsets, and others.

Initial State (IDLE): When the start signal is asserted, the module initializes hash values `h0` through `h7`, sets up some variables, and transitions to the LOAD state.

LOAD State: In this state, the module reads data from memory into its message buffer. Once all the required data is loaded, it transitions to the BLOCK state.

BLOCK State: The BLOCK state processes the data in 512-bit blocks. It loads data from the

message buffer and prepares it for hashing.

WAIT State: In the WAIT state, the module prepares data for the next block and computes values like P.

COMPUTE State: The COMPUTE state performs the SHA-256 hash computations using various helper functions (sha256_op, Word_Expansion). It iterates through 64 rounds for each 512-bit block.

WRITE State: In the WRITE state, the final hash values are written back to memory. Once all hash values are written, the module transitions back to the IDLE state.

Helper Functions:

The module defines two helper functions, sha256_op, and Word_Expansion, which assist in performing the hash computations and expanding words, respectively.

Assignments and Expressions:

Various assignments and expressions are used to manipulate data and signals within the module.

Final Output (done):

The done signal indicates when the SHA-256 hash computation is complete, and the module is in the IDLE state.

The code implements the core functionality of the SHA-256 algorithm, including message loading, block processing, hash computation, and result writing. The actual SHA-256 operations are carried out in the COMPUTE state, with values stored in the h0 to h7 variables. The final hash values are written back to memory in the WRITE state.

Bitcoin hash algorithm description:

Module Parameters:

num_nonces and **num_words:** These parameters specify the number of nonces (16) and words in the message (21).

SHA-256 Constants:

The code defines an array **k[64]** to hold the SHA-256 constants used in the algorithm. These constants are necessary for the chained SHA-256 computations.

Finite State Machine (FSM):

The module uses an FSM with eight states (IDLE, LOAD, PHASE_1, PHASE_2, PHASE_3, WAIT_1, WAIT_2, WRITE) to control the Bitcoin hashing process.

Local Variables:

Various local variables, including **w[16][16]**, **message[32]**, **ha[8]**, **hb[16][8]**, **offset**, **cur_we**, **cur_addr**, **cur_write_data**, **start_a**, **start_b**, **in_h**, **finish_a**, and **finish_b[16]**, are used for intermediate computations and managing the hashing process.

Memory Interface:

The module interfaces with memory using signals **mem_clk**, **mem_we**, **mem_addr**, **mem_write_data**, and **mem_read_data** to read the input message and write the computed hash back to memory.

FSM Operation:

The FSM controls the flow of the Bitcoin hashing process. It begins in the IDLE state, initializes variables, and transitions to the LOAD state when **start** is asserted.

Load Phase (LOAD and PHASE_1):

In the LOAD state, the module reads the input message from memory in 8-byte chunks. When the message is fully loaded, it adds the necessary padding and transitions to PHASE_2. Phase 2 (PHASE_2):

PHASE_2 prepares the message for nonce variations. It initializes nonce-specific values in w. Phase 3 (PHASE_3):

PHASE_3 prepares the w values for the final SHA-256 hashing. Wait States (WAIT_1 and WAIT_2):

These states provide time for the chained SHA-256 algorithms to complete their computations.

Write State (WRITE):

In the WRITE state, the module writes the resulting hash values back to memory in 8-byte chunks.

Instantiation of simplified_sha256 Modules:

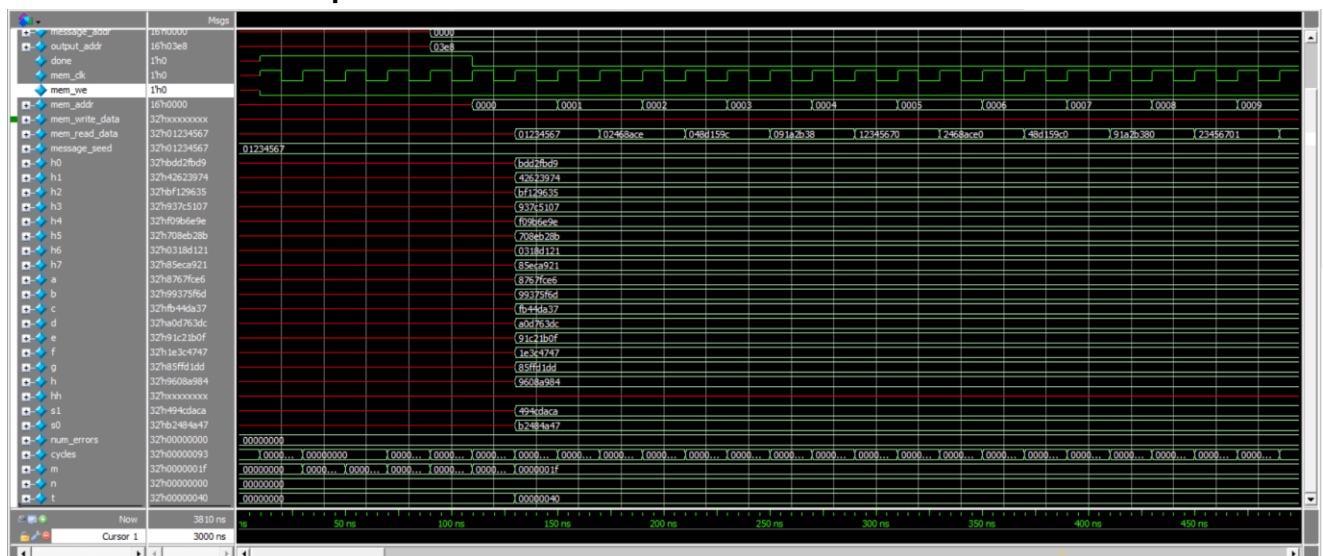
The code instantiates a main simplified_sha256 module called insha to perform the first round of SHA-256 hashing. It also generates 16 additional instances of simplified_sha256 modules (inside a generate block) to handle nonce-specific SHA-256 computations in parallel.

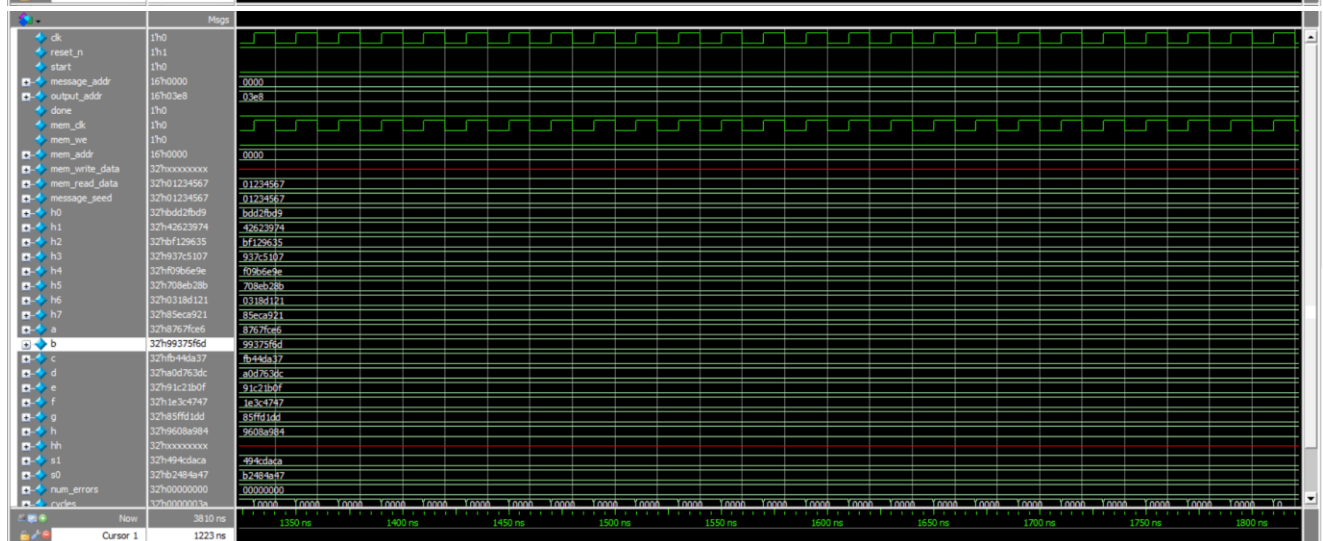
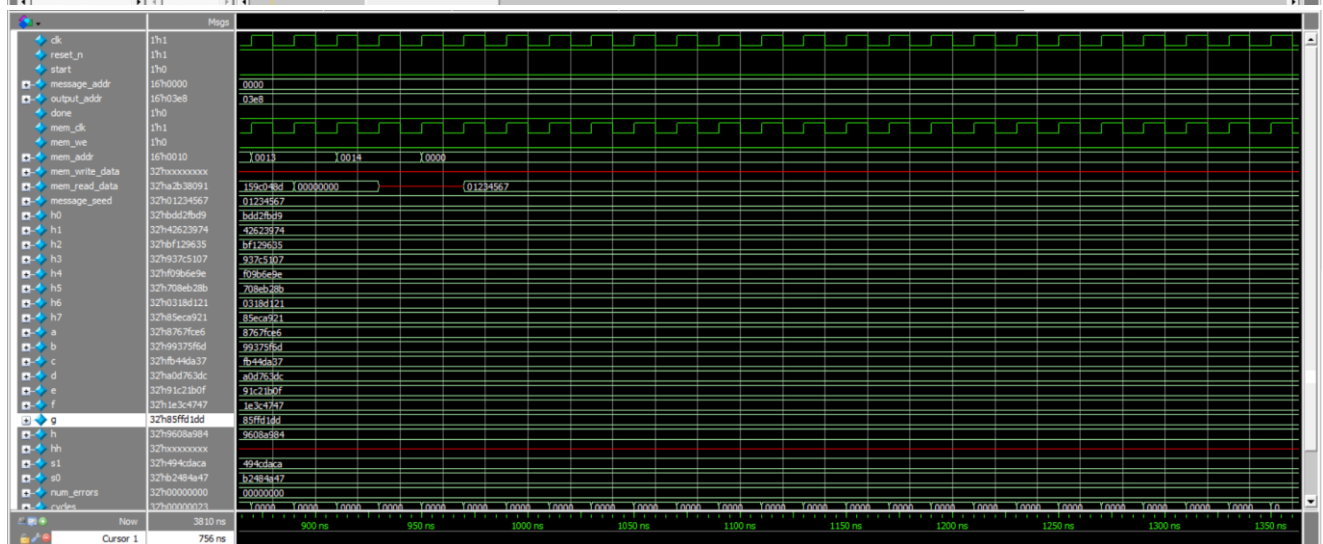
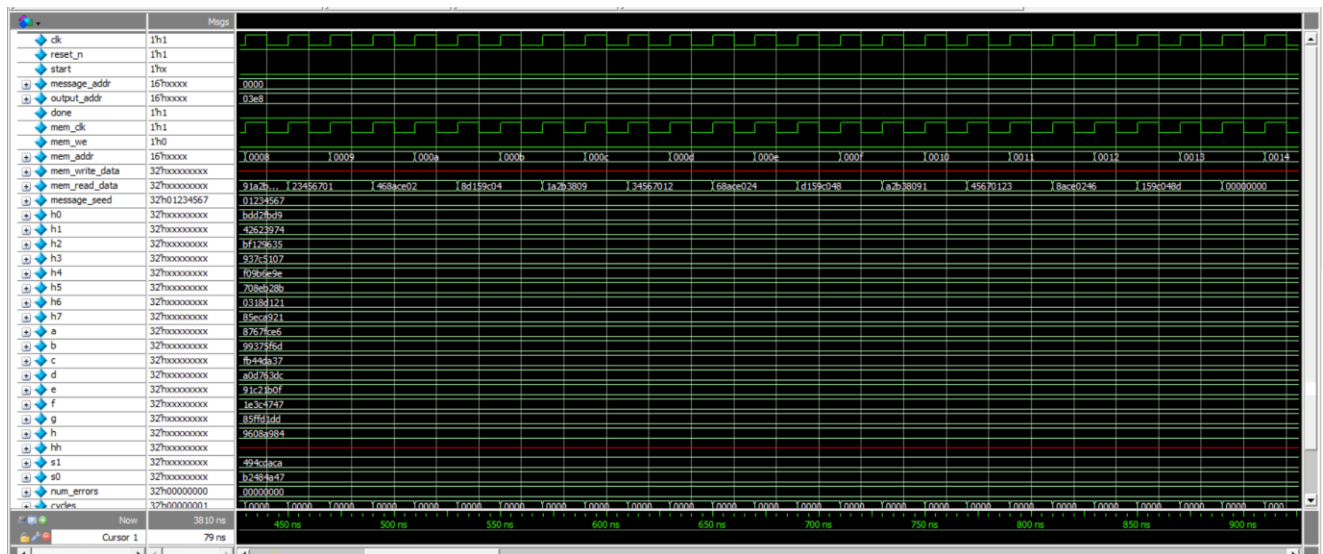
done Signal:

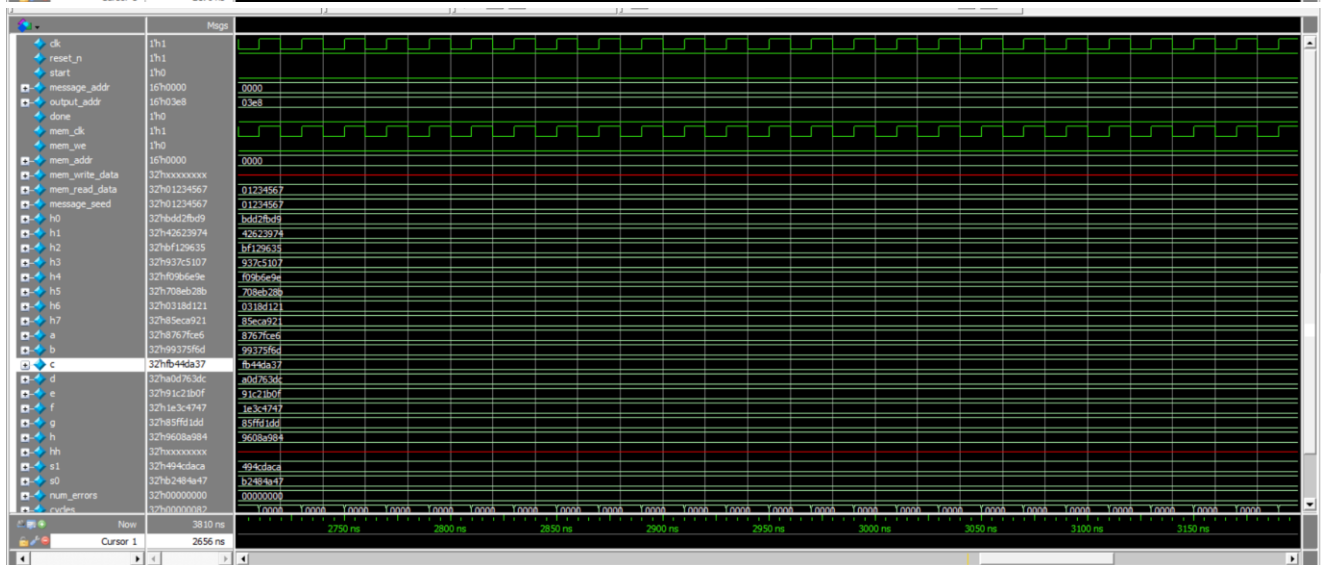
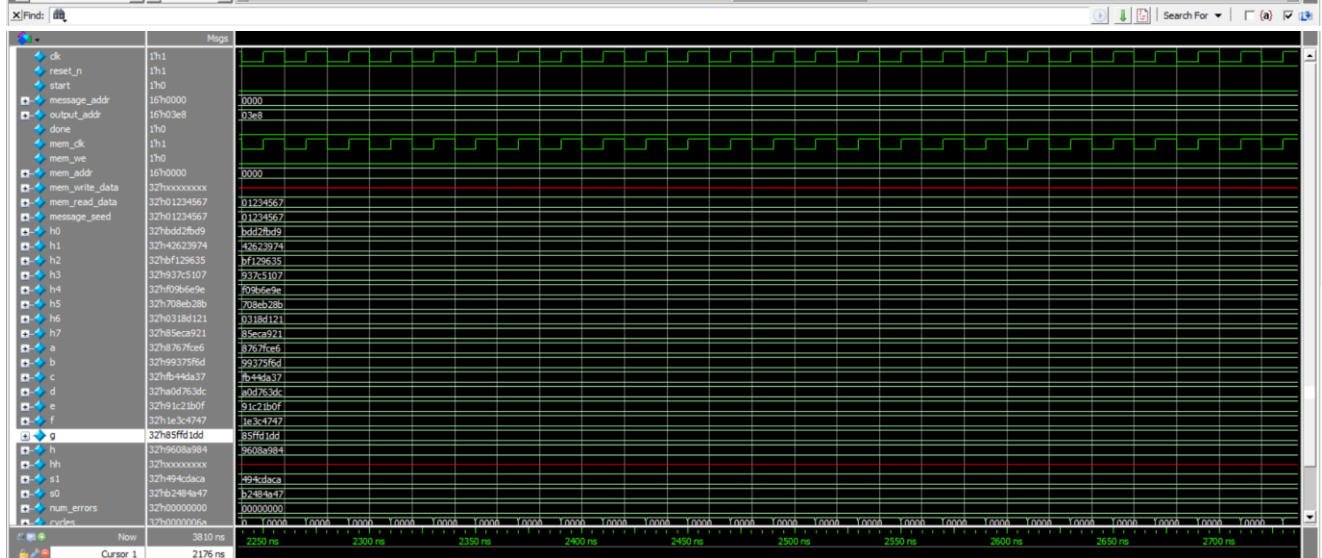
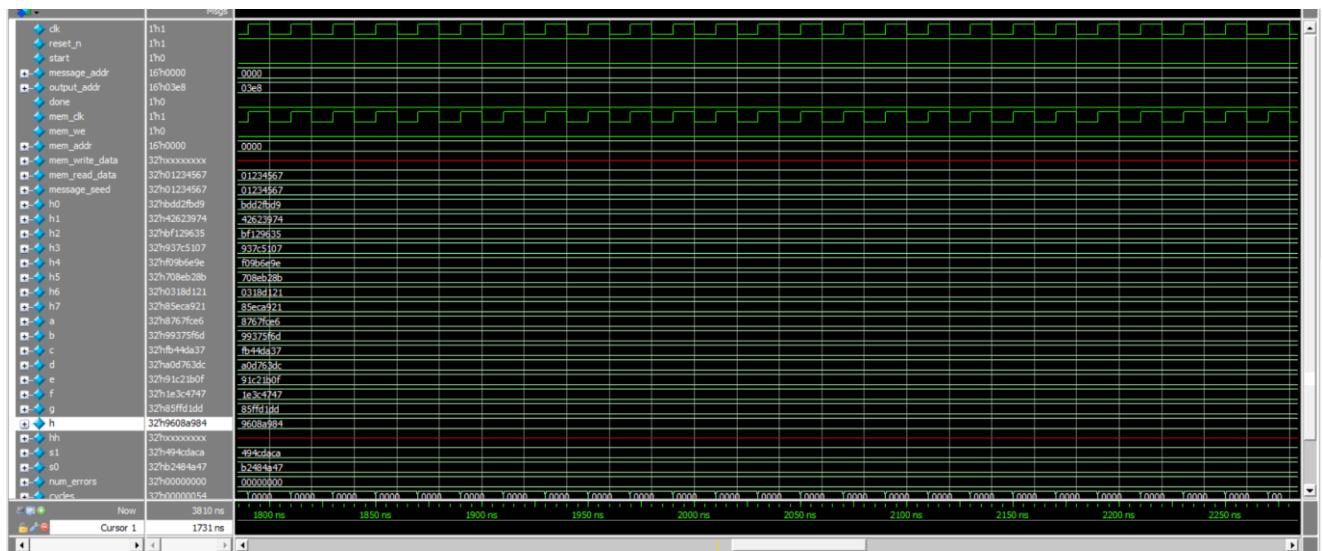
The done signal is set when the FSM reaches the IDLE state, indicating that the Bitcoin hashing process is complete.

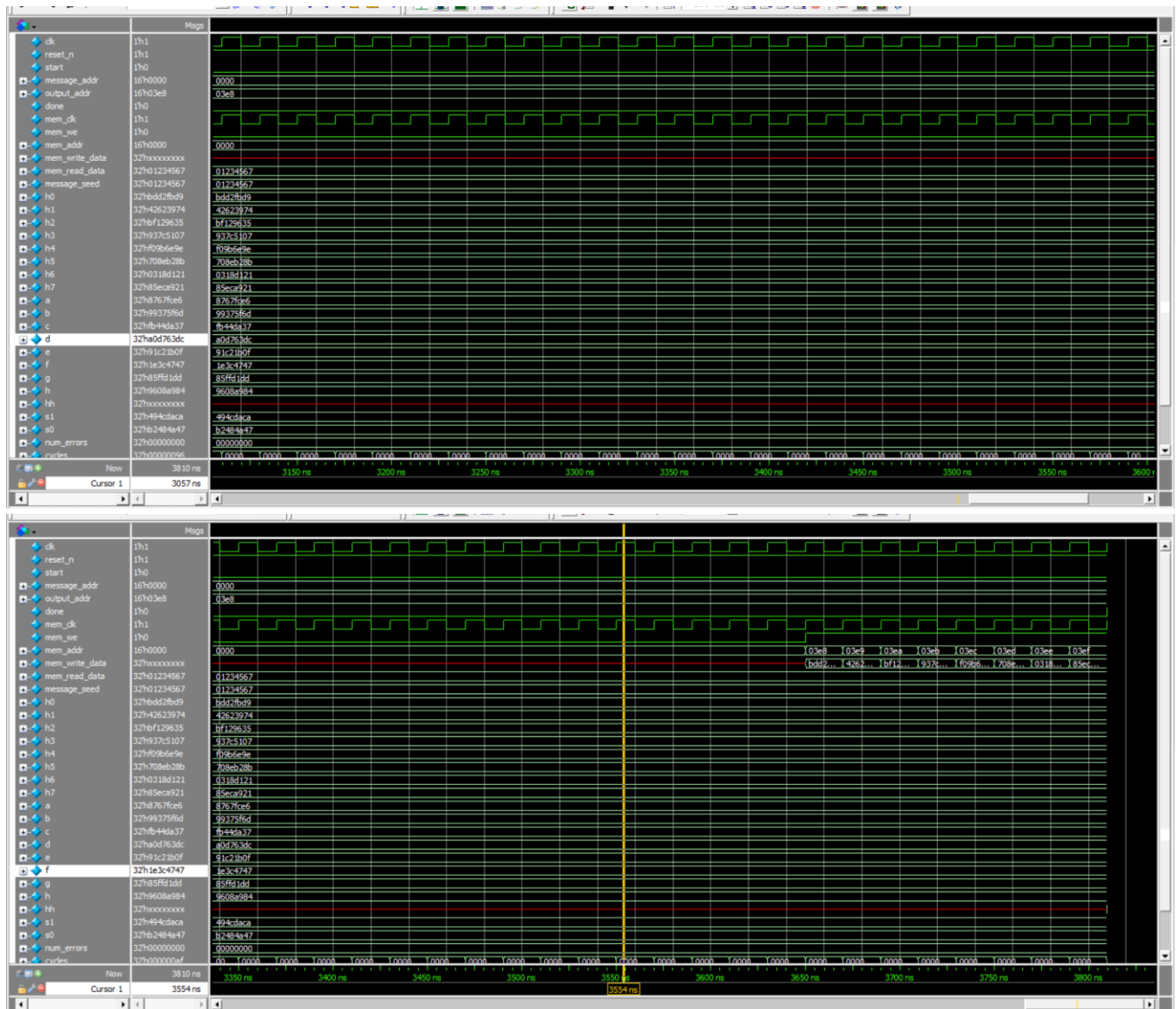
SHA-256 and Bitcoin hashing simulation waveform snapshot

SHA-256 waveform snapshot



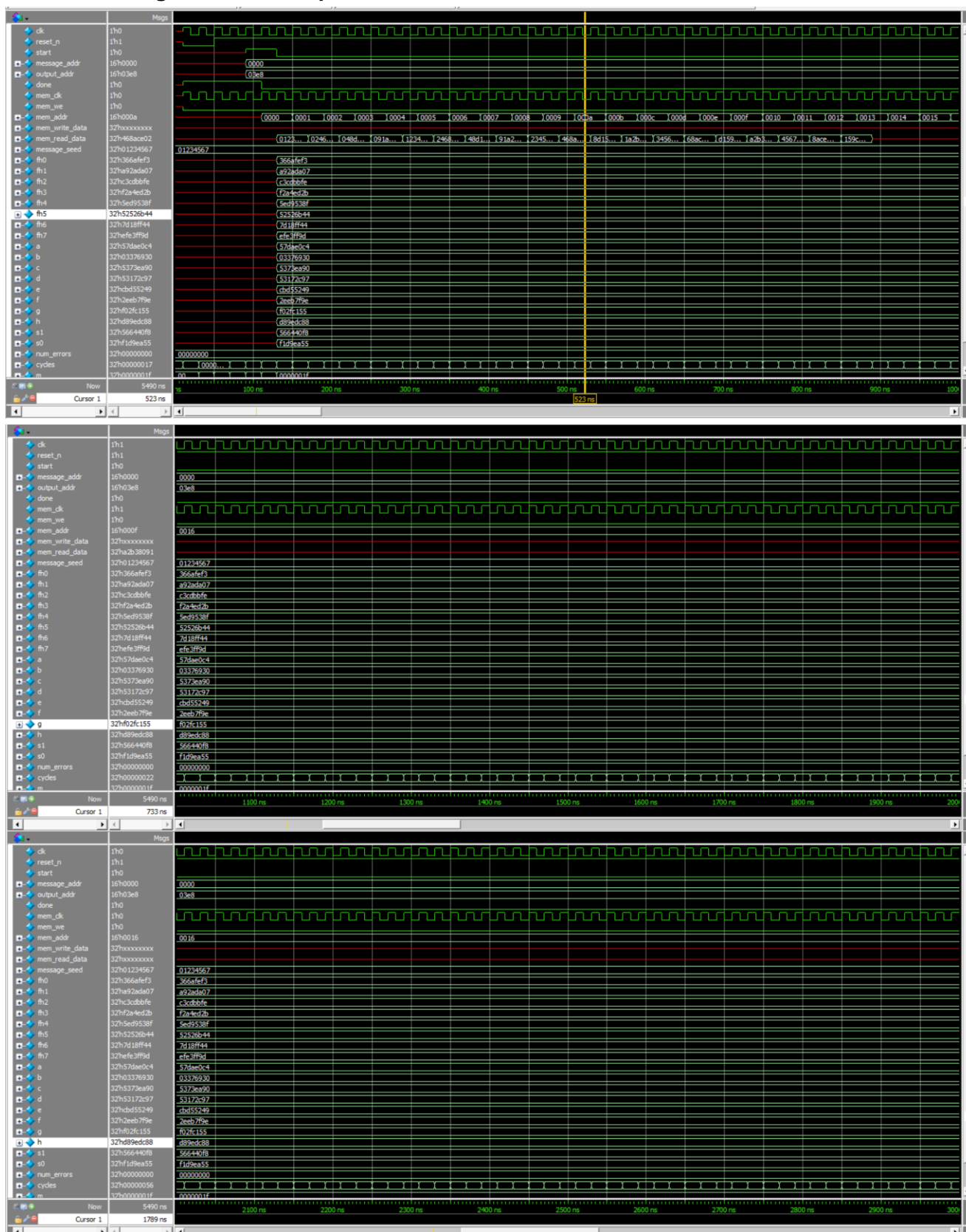


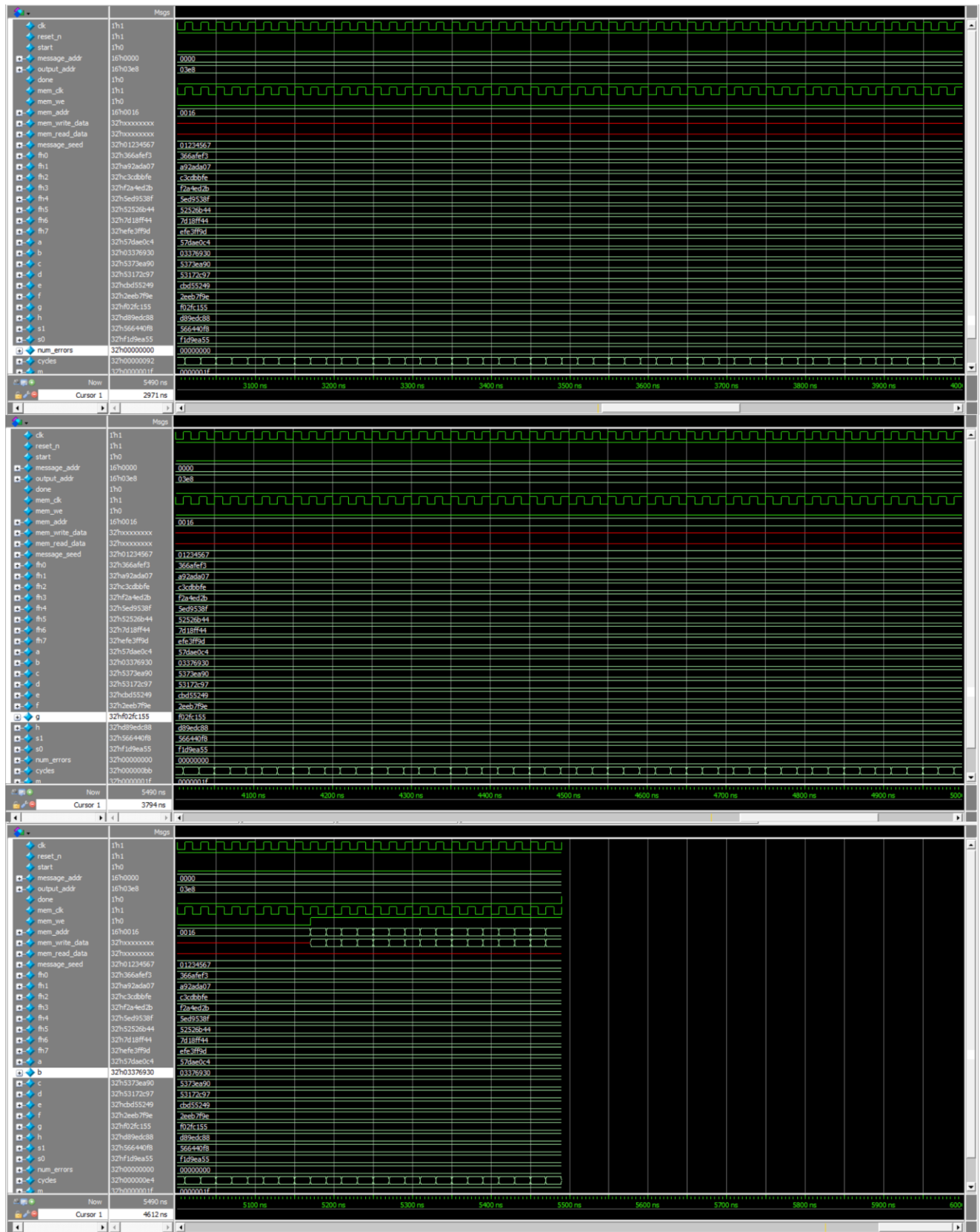




This is the waveform snapshot of the final done signals and showing final output hash hexadecimal values for the SHA-256 project. We see that clearly correct hash values are written into the memory when the done signal is set to 1 at 3650 ns.

Bitcoin hashing waveform snapshot





This is the waveform snapshot of final done signals and showing final output hash hexadecimal values for the Bitcoin project. We see that clearly correct hash values are written into the memory when the done signal is set to 1 at around 5170ns.

Modelsim transcript window output for SHA-256 and Bitcoin hashing

Simplified SHA

```
# Loading work.simplified_sha256.tcl
VSIM 33> run -all
# -----
# MESSAGE:
# -----
# 01234567
# 02468ace
# 048d159c
# 091a2b38
# 12345670
# 2468ace0
# 48d159c0
# 91a2b380
# 23456701
# 468ace02
# 8d159c04
# 1a2b3809
# 34567012
# 68ace024
# d159c048
# a2b38091
# 45670123
# 8ace0246
# 159c048d
# 00000000
# *****
#
# -----
# COMPARE HASH RESULTS:
# -----
# Correct H[0] = bdd2fbd9 Your H[0] = bdd2fbd9
# Correct H[1] = 42623974 Your H[1] = 42623974
# Correct H[2] = bf129635 Your H[2] = bf129635
# Correct H[3] = 937c5107 Your H[3] = 937c5107
# Correct H[4] = f09b6e9e Your H[4] = f09b6e9e
# Correct H[5] = 708eb28b Your H[5] = 708eb28b
# Correct H[6] = 0318d121 Your H[6] = 0318d121
# Correct H[7] = 85eca921 Your H[7] = 85eca921
# *****
#
# CONGRATULATIONS! All your hash results are correct!
#
# Total number of cycles:          188
#
```

Bitcoin

```
add wave -position insertpoint sim:/tb_bitcoin_hash/*
VSIM 43> run -all
# -----
# 19 WORD HEADER:
# -----
# 01234567
# 02468ace
# 048d159c
# 091a2b38
# 12345670
# 2468ace0
# 48d159c0
# 91a2b380
# 23456701
# 468ace02
# 8d159c04
# 1a2b3809
# 34567012
# 68ace024
# d159c048
# a2b38091
# 45670123
# 8ace0246
# 159c048d
# *****
#
# -----
# COMPARE HASH RESULTS:
# -----
# Correct H0[ 0] = 7106973a Your H0[ 0] = 7106973a
# Correct H0[ 1] = 6e66eea7 Your H0[ 1] = 6e66eea7
# Correct H0[ 2] = fbef64dc Your H0[ 2] = fbef64dc
# Correct H0[ 3] = 0888a18c Your H0[ 3] = 0888a18c
# Correct H0[ 4] = 9642d5aa Your H0[ 4] = 9642d5aa
# Correct H0[ 5] = 2ab6af8b Your H0[ 5] = 2ab6af8b
# Correct H0[ 6] = 24259d8c Your H0[ 6] = 24259d8c
# Correct H0[ 7] = ffb9bcd9 Your H0[ 7] = ffb9bcd9
# Correct H0[ 8] = 642138c9 Your H0[ 8] = 642138c9
# Correct H0[ 9] = 054cafc7 Your H0[ 9] = 054cafc7
# Correct H0[10] = 78251a17 Your H0[10] = 78251a17
# Correct H0[11] = af8c8f22 Your H0[11] = af8c8f22
# Correct H0[12] = d7a79ef8 Your H0[12] = d7a79ef8
# Correct H0[13] = c7d10c84 Your H0[13] = c7d10c84
# Correct H0[14] = 9537acfd Your H0[14] = 9537acfd
# Correct H0[15] = cle4c72b Your H0[15] = cle4c72b
# *****
#
```

```
# -----
# COMPARE HASH RESULTS:
# -----
# Correct H0[ 0] = 7106973a Your H0[ 0] = 7106973a
# Correct H0[ 1] = 6e66eea7 Your H0[ 1] = 6e66eea7
# Correct H0[ 2] = fbef64dc Your H0[ 2] = fbef64dc
# Correct H0[ 3] = 0888a18c Your H0[ 3] = 0888a18c
# Correct H0[ 4] = 9642d5aa Your H0[ 4] = 9642d5aa
# Correct H0[ 5] = 2ab6af8b Your H0[ 5] = 2ab6af8b
# Correct H0[ 6] = 24259d8c Your H0[ 6] = 24259d8c
# Correct H0[ 7] = ffb9bcd9 Your H0[ 7] = ffb9bcd9
# Correct H0[ 8] = 642138c9 Your H0[ 8] = 642138c9
# Correct H0[ 9] = 054cafc7 Your H0[ 9] = 054cafc7
# Correct H0[10] = 78251a17 Your H0[10] = 78251a17
# Correct H0[11] = af8c8f22 Your H0[11] = af8c8f22
# Correct H0[12] = d7a79ef8 Your H0[12] = d7a79ef8
# Correct H0[13] = c7d10c84 Your H0[13] = c7d10c84
# Correct H0[14] = 9537acfd Your H0[14] = 9537acfd
# Correct H0[15] = cle4c72b Your H0[15] = cle4c72b
# *****
#
# CONGRATULATIONS! All your hash results are correct!
#
# Total number of cycles:          272
#
# *****
#
# ** Note: $stop      : C:/Users/16262/Downloads/Final_Project (6)/bitcoin_hash/tb_bitcoin_hash.sv(334)
# Time: 5490 ns  Iteration: 2  Instance: /tb_bitcoin_hash
# -----
```

Synthesis resource usage and timing report for bitcoin_hash only.

Synthesis resource usage

1	▼ Estimated ALUTs Used	13623
1	-- Combinational ALUTs	13623
2	-- Memory ALUTs	0
3	-- LUT_REGS	0
2	Dedicated logic registers	27998
3		
4	▼ Estimated ALUTs Unavailable	0
1	-- Due to unpartnered combinational logic	0
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	13623
7	▼ Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	265
3	-- 5 input functions	752
4	-- 4 input functions	14
5	-- <=3 input functions	12592
8		
9	▼ Combinational ALUTs by mode	
1	-- normal mode	4930
2	-- extended LUT mode	0
3	-- arithmetic mode	6517
4	-- shared arithmetic mode	2176
10		
11	Estimated ALUT/register pairs used	30402
12		
13	▼ Total registers	27998
1	-- Dedicated logic registers	27998
2	-- I/O registers	0
3	-- LUT_REGS	0
14		
15		
16	I/O pins	118


15		
16	I/O pins	118
17		
18	DSP block 18-bit elements	0
19		
20	Maximum fan-out node	clk~input
21	Maximum fan-out	27999
22	Total fan-out	171634
23	Average fan-out	4.10

Fitter report



Fitter Status	Successful - Thu Sep 14 13:57:34 2023
Quartus Prime Version	21.1.1 Build 850 06/23/2022 SJ Lite Edition
Revision Name	bitcoin_hash
Top-level Entity Name	bitcoin_hash
Family	Arria II GX
Device	EP2AGX45DF29I5
Timing Models	Final
Logic utilization	96 %
Total registers	27998
Total pins	118 / 404 (29 %)
Total virtual pins	0
Total block memory bits	0 / 2,939,904 (0 %)
DSP block 18-bit elements	0 / 232 (0 %)
Total GXB Receiver Channel PCS	0 / 8 (0 %)
Total GXB Receiver Channel PMA	0 / 8 (0 %)
Total GXB Transmitter Channel PCS	0 / 8 (0 %)
Total GXB Transmitter Channel PMA	0 / 8 (0 %)
Total PLLs	0 / 4 (0 %)
Total DLLs	0 / 2 (0 %)

Timing report

Slow 900mV 100C Model Fmax Summary					
 <<Filter>>					
	Fmax	Restricted Fmax	Clock Name	Note	
1	164.39 MHz	164.39 MHz	clk		