Sean Fuhrman Nathaniel Greenberg
Professor Koushanfar
ECE 111
16 December 2023

Lab Report ECE 111 Final Project

SHA-256

Explain briefly what SHA-256 is

SHA-256 is a cryptographic hash function used to create a unique 256-bit hash value for any input message. Its primary purpose is to enhance security by ensuring that even a small change in the input produces a significantly different hash output. This fixed-size output, commonly 32 bytes, adds consistency and predictability to the hashing process. SHA-256 is designed to resist collision attacks and preimage attacks, making it a secure choice for various cryptographic applications. In blockchain systems like Bitcoin, SHA-256 is integral to the proof-of-work algorithm, securing transactions and linking blocks together. Beyond blockchain, SHA-256 is widely applied in digital signatures and certificate authorities. Its role is to verify the authenticity and integrity of digital documents, software, and communication by generating a reliable hash value. This makes SHA-256 a fundamental tool for securing digital information in diverse online applications.

SHA256 Implementation Algorithm

We have 8 states: IDLE, BLOCK, COMPUTE, WRITE, READ, WAIT, INIT_WRITE.

IDLE:

- Wait for start signal, once it is received:
    - Initialize H0 - H7
    - Initialize A,B,C,D,E,F,G,H with H0-H7 values
    - Initialize all counting and tracking variables to 0
    - Go to BLOCK state

BLOCK:

- Check if all blocks have been processed
    - If yes, go to INIT_WRITE state
- Check if a block has been read into memory
    - If no, go to WAIT state to continue reading
    - If yes, go to COMPUTE state to compute current block

WAIT:

- Check if a full block has been read:
    - If no, go to READ
    - If yes, go to BLOCK

READ:

- Check if the current block/word needs to be padding:

    - If so, add appropriate padding word
    - If not, read the current read_addr from memory and increment the offset
- Go to WAIT state

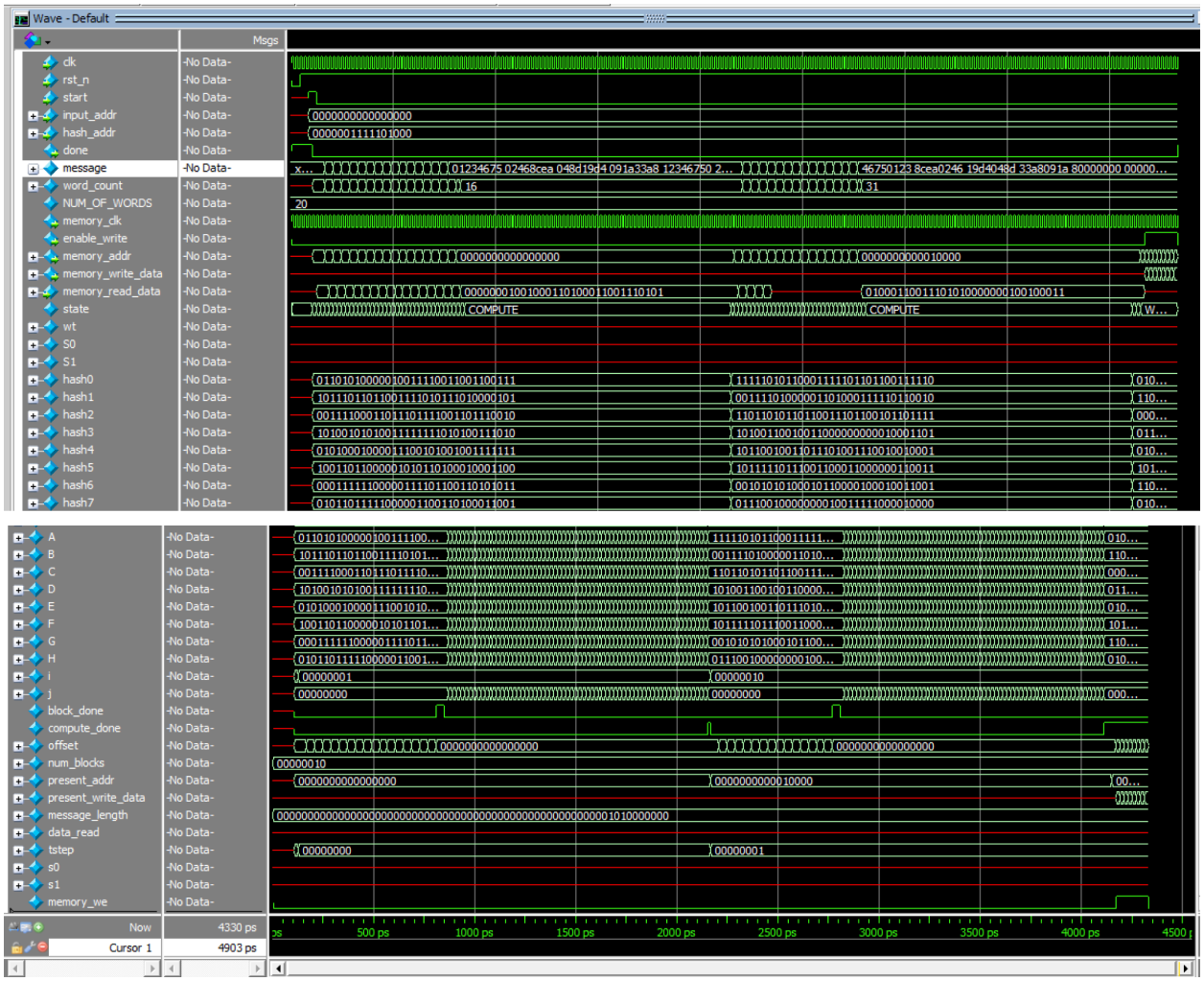
COMPUTE:
- For 64 cycles, compute next step of SHA256
- If 64 cycles is done, store output in H0-H7 then go to BLOCK state

INIT_WRITE:
- Initialize write address and offset, and assign H0-H7 to hash_out array
- Go to WRITE

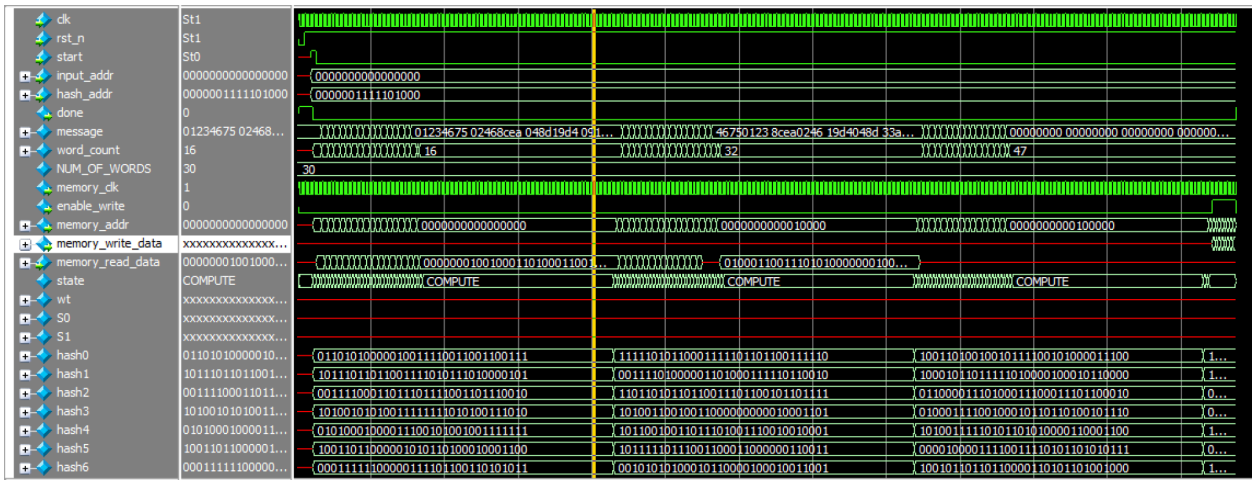
WRITE:
- Loops 8 times to write out the 8 blocks of hash_out
- Go to IDLE [algorithm done]

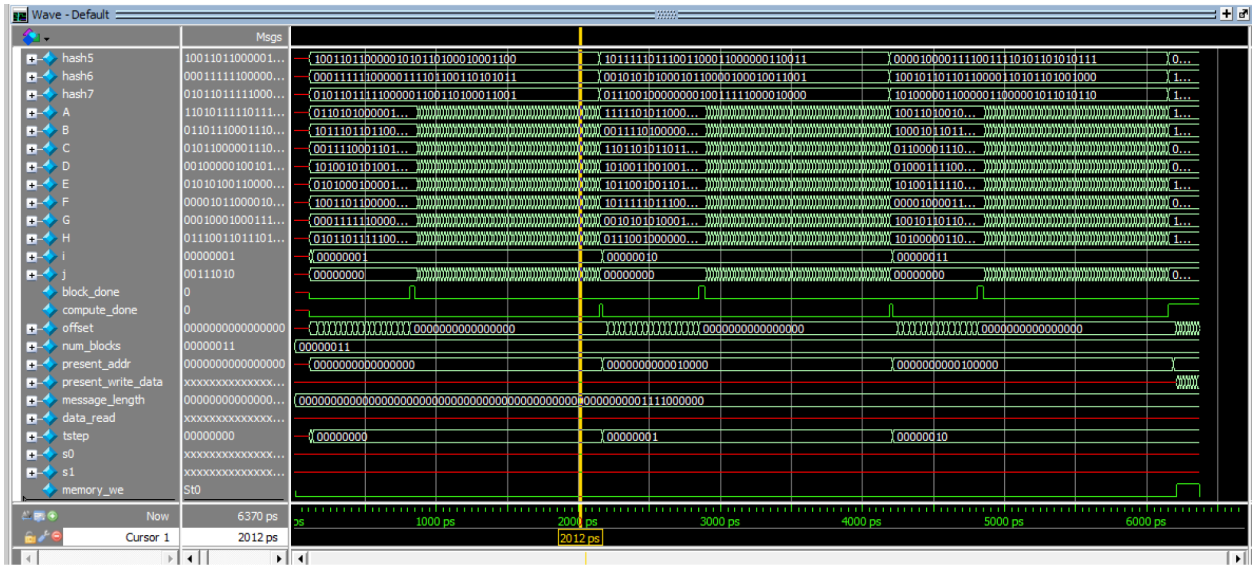
Simulation waveform snapshot for SHA-256

20 words

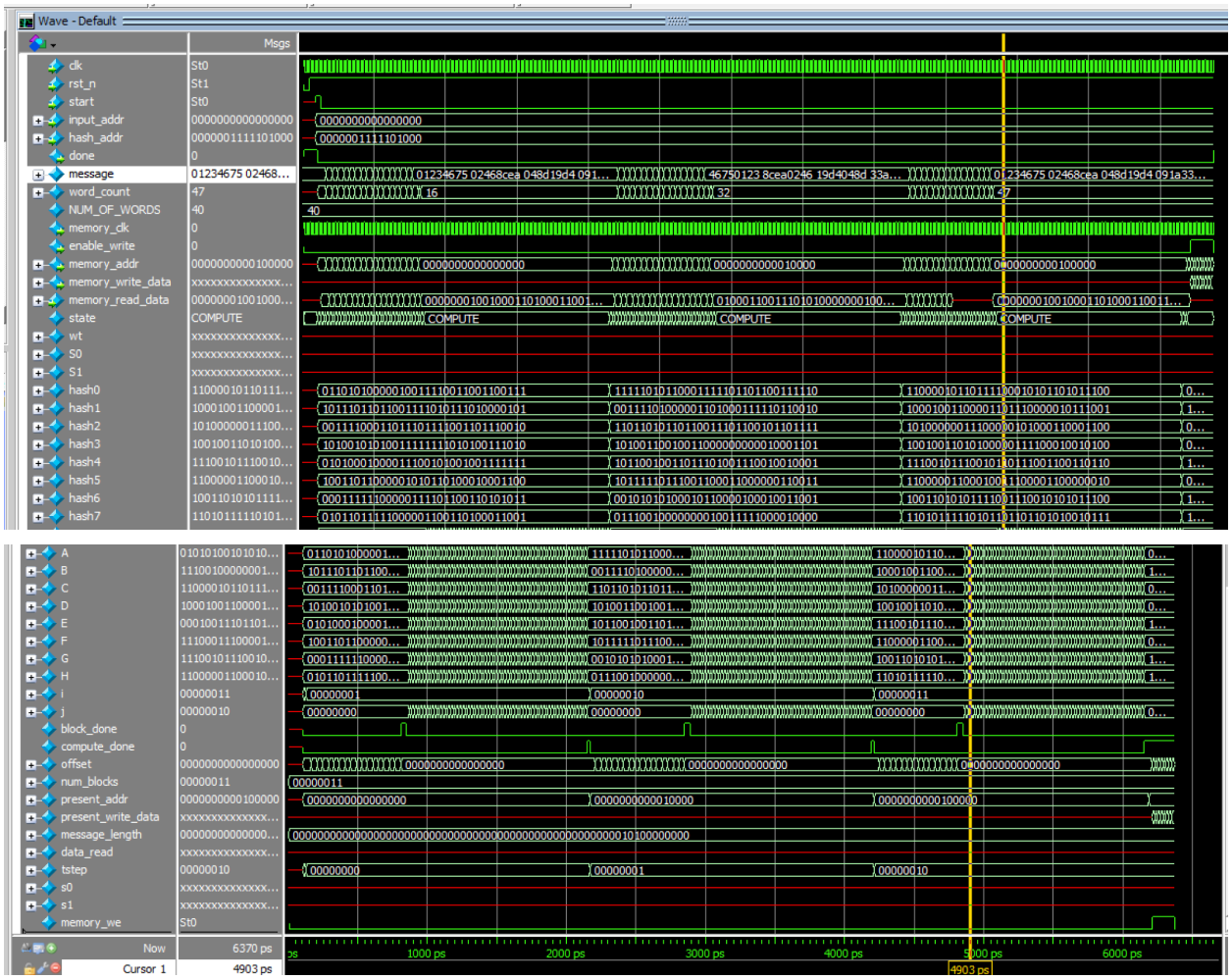

Wave - Default
Msgs
clk
rst_n
start
input_addr
hash_addr
done
message
word_count
NUM_OF_WORDS
memory_clk
enable_write
memory_addr
memory_write_data
memory_read_data
state
wt
S0
S1
hash0
hash1
hash2
hash3
hash4
hash5
hash6
hash7
COMPUTE
A
B
C
D
E
F
G
H
i
j
block_done
compute_done
offset
num_blocks
present_addr
present_write_data
message_length
data_read
tstep
s0
s1
memory_we
Now
4330 ps
Cursor 1
4903 ps
500 ps
1000 ps
1500 ps
2000 ps
2500 ps
3000 ps
3500 ps
4000 ps

30 words


clk
rst_n
start
input_addr
hash_addr
done
message
word_count
NUM_OF_WORDS
memory_clk
enable_write
memory_addr
memory_write_data
memory_read_data
state
wt
S0
S1
hash0
hash1
hash2
hash3
hash4
hash5
hash6


Wave - Default
Msgs
hash5
hash6
hash7
A
B
C
D
E
F
G
H
i
j
block_done
compute_done
offset
num_blocks
present_addr
present_write_data
message_length
data_read
tstep
s0
s1
memory_we
Now
6370 ps
Cursor 1
2012 ps
1000 ps
2000 ps
3000 ps
4000 ps
5000 ps
6000 ps

40 words



Resource utilization for SHA-256

20 words

	Resource	Usage
1	Estimated ALUTs Used	1773
1	-- Combinational ALUTs	1773
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	1890
3		
4	Estimated ALUTs Unavailable	237
1	-- Due to unpartnered combinational logic	237
2	-- -- Due to unpartnered combinational logic	0
5		
6	Total combinational functions	1773
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	407
3	-- 5 input functions	296
4	-- 4 input functions	9
5	-- <=3 input functions	1061
8		
9	Combinational ALUTs by mode	
1	-- normal mode	1226
2	-- extended LUT mode	0
3	-- arithmetic mode	419
4	-- shared arithmetic mode	128
10		
11	Estimated ALUT/register pairs used	2690
12		
13	Total registers	1890
1	-- Dedicated logic registers	1890

13	Total registers	1890
1	-- Dedicated logic registers	1890
2	-- I/O registers	0
3	-- LUT_REGs	0
14		
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	1891
22	Total fan-out	15260
23	Average fan-out	3.91

30 words

	Resource	Usage
1	Estimated ALUTs Used	1767
1	-- Combinational ALUTs	1767
2	-- Combinational ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	1890
3		
4	Estimated ALUTs Unavailable	34
1	-- Due to unpartnered combinational logic	34
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	1767
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	204
3	-- 5 input functions	277
4	-- 4 input functions	227
5	-- <=3 input functions	1059
8		
9	Combinational ALUTs by mode	
1	-- normal mode	1220
2	-- extended LUT mode	0
3	-- arithmetic mode	419
4	-- shared arithmetic mode	128
10		

10		
11	Estimated ALUT/register pairs used	2481
12		
13	Total registers	1890
1	-- Dedicated logic registers	1890
2	-- I/O registers	0
3	-- LUT_REGs	0
14		
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	1891
22	Total fan-out	14813
23	Average fan-out	3.81

40 words

	Resource	Usage
1	Estimated ALUTs Used	1774
1	-- Combinational ALUTs	1774
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	1890
3		
4	Estimated ALUTs Unavailable	254
1	-- Due to unpartnered combinational logic	254
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	1774
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	424
3	-- 5 input functions	280
4	-- 4 input functions	11
5	-- <=3 input functions	1059
8		
9	Combinational ALUTs by mode	
1	-- normal mode	1227
2	-- extended LUT mode	0
3	-- arithmetic mode	419
4	-- shared arithmetic mode	128
10		
11	Estimated ALUT/register pairs used	2708
12		
13	Total registers	1890

13	Total registers	1890
1	-- Dedicated logic registers	1890
2	-- Dedicated logic registers	0
3	-- LUT_REGs	0
14		
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	1891
22	Total fan-out	15283
23	Average fan-out	3.92

The resource utilization for 20 words, 30 words, and 40 words are all very similar. This makes sense because the state machine/ hardware doesn't really change for all the word settings. It would just have a longer delay because it requires a longer number of cycles to process more blocks.

Provide modelsim transcript window output indicating passing test results generated from self-checker in testbench for SHA-256

20 words

```
# ****************************
#
# --------------------
# COMPARE HASH RESULTS:
# --------------------
# Correct H[0] = 5b8feb0a Your H[0] = 5b8feb0a
# Correct H[1] = d258a227 Your H[1] = d258a227
# Correct H[2] = 116df790 Your H[2] = 116df790
# Correct H[3] = 66c9d8cc Your H[3] = 66c9d8cc
# Correct H[4] = 47e75276 Your H[4] = 47e75276
# Correct H[5] = a5316e2f Your H[5] = a5316e2f
# Correct H[6] = d1965a81 Your H[6] = d1965a81
# Correct H[7] = 5904edff Your H[7] = 5904edff
# ****************************
#
# CONGRATULATIONS! All your hash results are correct!
#
# Total number of cycles:         214
#
#
# ****************************
#
# ** Note: $stop    : H:/Fall2023_final_project_SHA256_and_Bitcoin/Final_Project/Final_Project/simplified_sha256/tb_simplified_sha256_20w.sv(231)
#    Time: 4330 ps  Iteration: 2  Instance: /tb_simplified_sha256
```

30 words

```
# ---------------------
# COMPARE HASH RESULTS:
# ---------------------
# Correct H[0] = abde57db Your H[0] = abde57db
# Correct H[1] = fb3f1bda Your H[1] = fb3f1bda
# Correct H[2] = 6c6f969c Your H[2] = 6c6f969c
# Correct H[3] = 6bdea244 Your H[3] = 6bdea244
# Correct H[4] = 8d5ff7cf Your H[4] = 8d5ff7cf
# Correct H[5] = 6d41389a Your H[5] = 6d41389a
# Correct H[6] = f85598ec Your H[6] = f85598ec
# Correct H[7] = 9b97ccc1 Your H[7] = 9b97ccc1
# ***************************
#
# CONGRATULATIONS! All your hash results are correct!
#
# Total number of cycles:         316
#
#
# ***************************
#
# ** Note: $stop    : H:/Fall2023_final_project_SHA256_and_Bitcoin/Final_Project/Final_Project/simplified_sha256/tb_simplified_sha256_30w.sv(233)
#    Time: 6370 ps  Iteration: 2  Instance: /tb_simplified_sha256
```

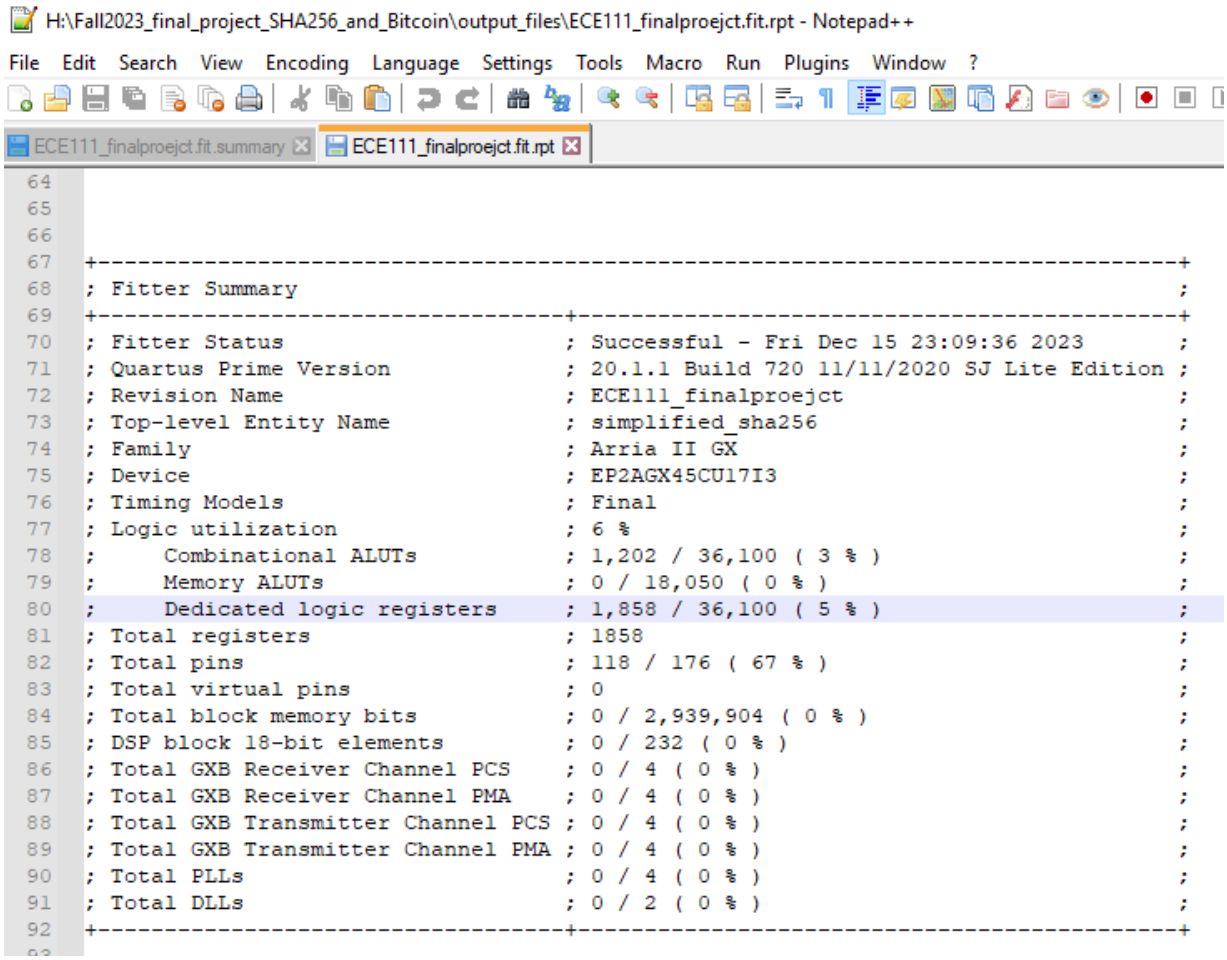
40 words

```
# ---------------------
# COMPARE HASH RESULTS:
# ---------------------
# Correct H[0] = 0244238c Your H[0] = 0244238c
# Correct H[1] = d2a3d7be Your H[1] = d2a3d7be
# Correct H[2] = 63ad61e7 Your H[2] = 63ad61e7
# Correct H[3] = 44f2fad8 Your H[3] = 44f2fad8
# Correct H[4] = f0da0c1b Your H[4] = f0da0c1b
# Correct H[5] = 10d16d52 Your H[5] = 10d16d52
# Correct H[6] = 86e5e36d Your H[6] = 86e5e36d
# Correct H[7] = a108b1c4 Your H[7] = a108b1c4
# ***************************
#
# CONGRATULATIONS! All your hash results are correct!
#
# Total number of cycles:         316
#
#
# ***************************
#
# ** Note: $stop    : H:/Fall2023_final_project_SHA256_and_Bitcoin/Final_Project/Final_Project/simplified_sha256/tb_simplified_sha256_40w.sv(232)
#    Time: 6370 ps  Iteration: 2  Instance: /tb_simplified_sha256
```

Provide Timing Fmax report snapshots

	Fmax	Restricted Fmax	Clock Name	Note
1	181.69 MHz	181.69 MHz	clk	

simplified_sha256.fit



simplified_sha256.sta

Bitcoin Hashing

Explain briefly what Bitcoin Hashing is

In the Bitcoin mining project, the computer reads a 19-word block header and calculates the double SHA-256 hash for 16 different nonce values. These resulting hash values are stored in memory at a specific location. The computer keeps doing this until it finishes, and a "done" flag shows that the computation is complete. This project mimics the essential steps of Bitcoin mining, where miners try different nonces to find a hash value that meets specific requirements, allowing them to add a new block to the blockchain.

Describe algorithm for Bitcoin hashing

We have 7 states: IDLE,READ, WAIT, COMPUTE_FIRST, COMPUTE_SECOND, PREP_THIRD, COMPUTE_THIRD, WRITE

IDLE:

- Wait for start signal, once it is received:
    - Initialize H0 - H7
    - Initialize A,B,C,D,E,F,G,H with H0-H7 values
    - Initialize all counting and tracking variables to 0
    - Go to WAIT state

WAIT:

- Check if reading is done
    - If yes:
        - store the first 16 words from the message into w.
        - Go to COMPUTE_FIRST state
    - If no
        - Go to READ state

READ:

- Check if full 19-word header has been read in
    - If yes:
        - Add nonce, and 12 words of padding to header
    - If no:
        - Read in a word, increment address
- Go to WAIT state

COMPUTE_FIRST:

- Go through 64 cycles of computing the first hash block (same for all nonces)
- Once that cycle is done, save the results, and initialize w for the following nonce calculations.
    - A2,B2,C2,D2,E2,F2,G2,H2 are num_nonce long arrays for the sha computation cycles. They are initialized to the results of the first block.
- Go to Compute Second state

COMPUTE_SECOND

- Go through 64 cycles of computing the second hash block, all in parallel with different nonce values.
- After this cycle store the results and add padding for the second round of SHA algorithm
- Go to PREP_THIRD state

PREP_THIRD

- Store the results of COMPUTE_SECOND into W for the SHA algorithm. This is all still parallel for different nonces
- Go to COMPUTE_THIRD state

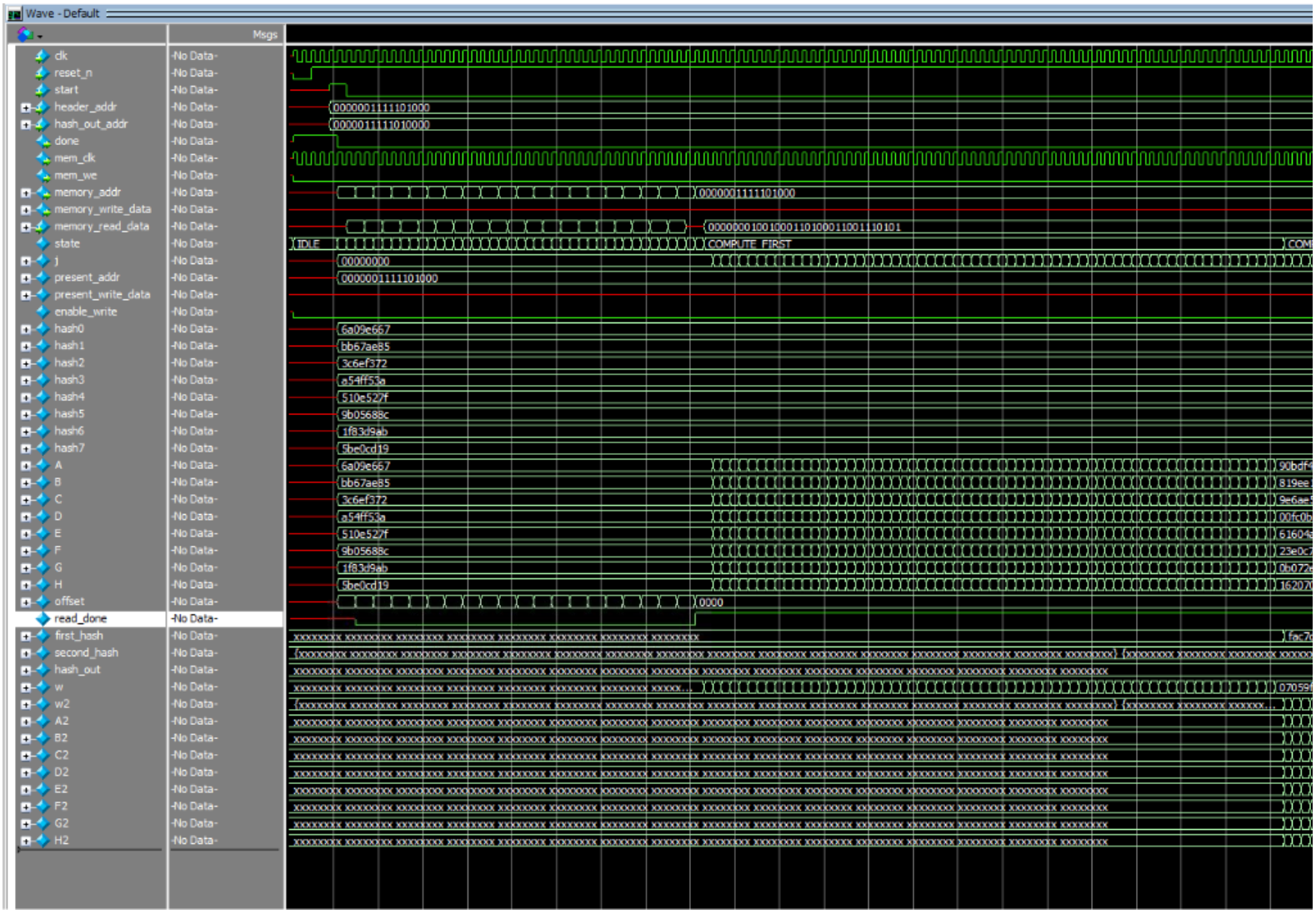
COMPUTE_THIRD

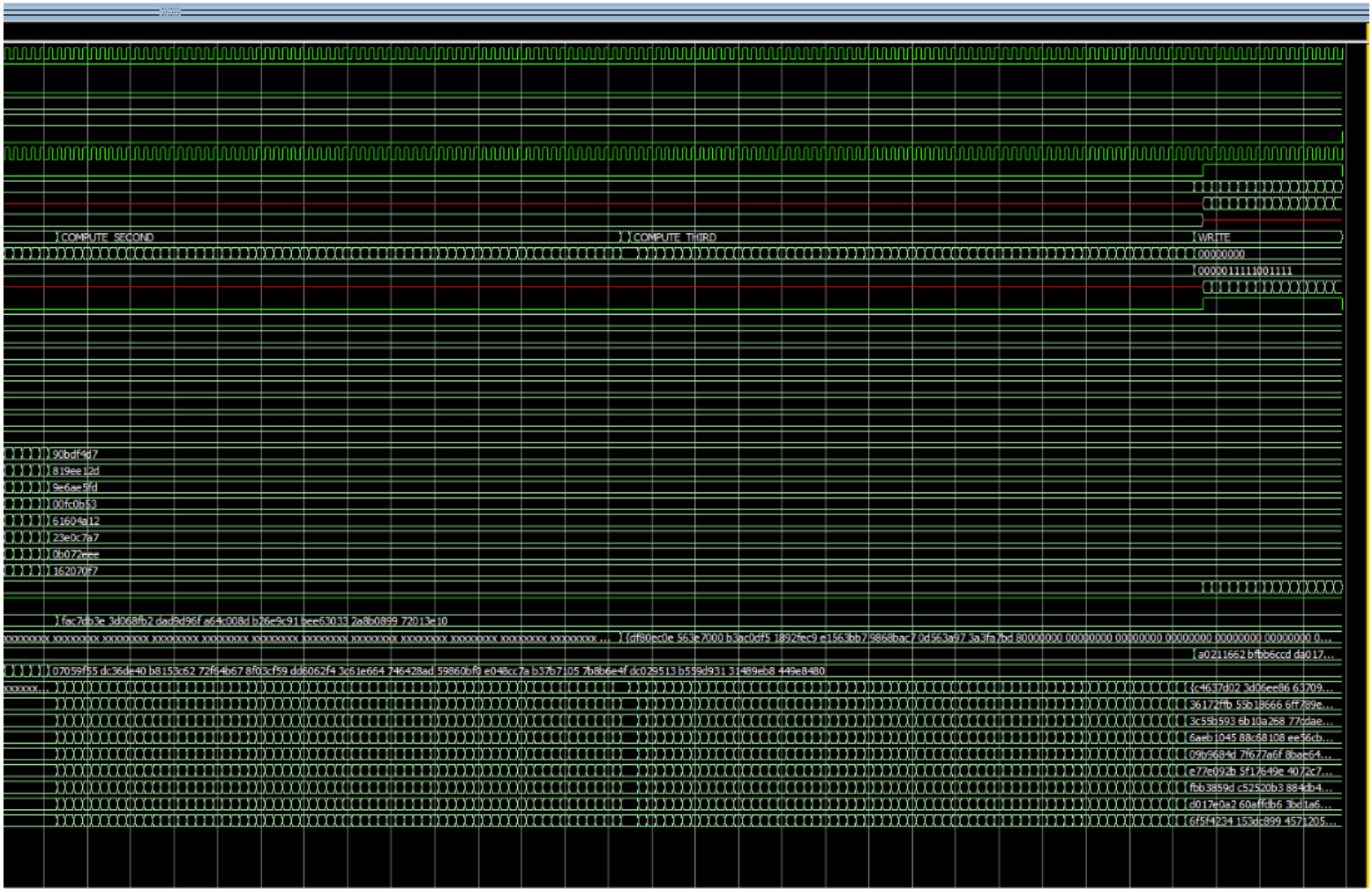
- Go through 64 cycles of computing the last SHA block, still all in parallel for different nonces
- After this, only store the first hash_out block for writing
- Go to WRITE state and initialize the write address.

WRITE:

- In 8 cycles, write the hash_out into memory
- After Go to IDLE [algorithm] done

Simulation waveform snapshot for Bitcoin Hashing




COMPUTE_SECOND
COMPUTE_THIRD
WRITE
00000000
0000011111001111
90bdf9d7
819ee12d
9e6ae9fd
00fc0b53
61604a12
23e0c7a7
0b072eee
162070f7
fac7db3e 3d068fb2 dad9d96f a64c008d b26e9c91 bee63033 2a8b0899 72013e10
df80ec0e 563e7000 b3ac0df5 1892fec9 e1563bb7 9868bac2 0d563a97 3a3fa7bd 80000000 00000000 00000000 00000000 00000000 00000000 0...
a0211662 bf0b6ccd da017...
07059f55 dc36de40 b8153c62 72f64b67 8f03cf59 dd6062f4 3c61e664 746428ad 59860bf0 e048cc7a b37b7105 7b8b6e4f dc029513 b559d931 31489eb8 449e8480
c4637d02 3d06ee86 63709...
36172ffb 55b18666 6ff289e...
3c55b593 6b10a268 77ddae...
6aeb1045 88c68108 ee56cb...
09b9684d 7f677a6f 8bae64...
e77e092b 5f17649e 4072c7...
fbb3859d c52520b3 884db4...
d017e0a2 60affdb6 3bd1a6...
6f5f4234 153dc899 4571205...

Provide modelsim transcript window output indicating passing test results generated from self-checker in testbench for Bitcoin Hashing

```
COMPARE HASH RESULTS:
---------------------
Correct H0[ 0] = a0211662 Your H0[ 0] = a0211662
Correct H0[ 1] = bfbb6ccd Your H0[ 1] = bfbb6ccd
Correct H0[ 2] = da017047 Your H0[ 2] = da017047
Correct H0[ 3] = 1c34e2aa Your H0[ 3] = 1c34e2aa
Correct H0[ 4] = 58993aea Your H0[ 4] = 58993aea
Correct H0[ 5] = b41b7a67 Your H0[ 5] = b41b7a67
Correct H0[ 6] = 04cf2ceb Your H0[ 6] = 04cf2ceb
Correct H0[ 7] = 85ab3945 Your H0[ 7] = 85ab3945
Correct H0[ 8] = f4539616 Your H0[ 8] = f4539616
Correct H0[ 9] = 0e4614d7 Your H0[ 9] = 0e4614d7
Correct H0[10] = 6bec8208 Your H0[10] = 6bec8208
Correct H0[11] = ce75ecf2 Your H0[11] = ce75ecf2
Correct H0[12] = 672cb1a0 Your H0[12] = 672cb1a0
Correct H0[13] = 4d48232a Your H0[13] = 4d48232a
Correct H0[14] = cfe99db3 Your H0[14] = cfe99db3
Correct H0[15] = 047d81b9 Your H0[15] = 047d81b9
**************************

CONGRATULATIONS! All your hash results are correct!

Total number of cycles:         257


**************************

** Note: $stop    : H:/Fall2023_final_project_SHA256_and_Bitcoin/Final_Project/Final_Project/bitcoin_hash/tb_bitcoin_hash.sv(338)
   Time: 5190 ps  Iteration: 2  Instance: /tb_bitcoin_hash
Break in Module tb_bitcoin_hash at H:/Fall2023_final_project_SHA256_and_Bitcoin/Final_Project/Final_Project/bitcoin_hash/tb_bitcoin_hash.sv line 338
```

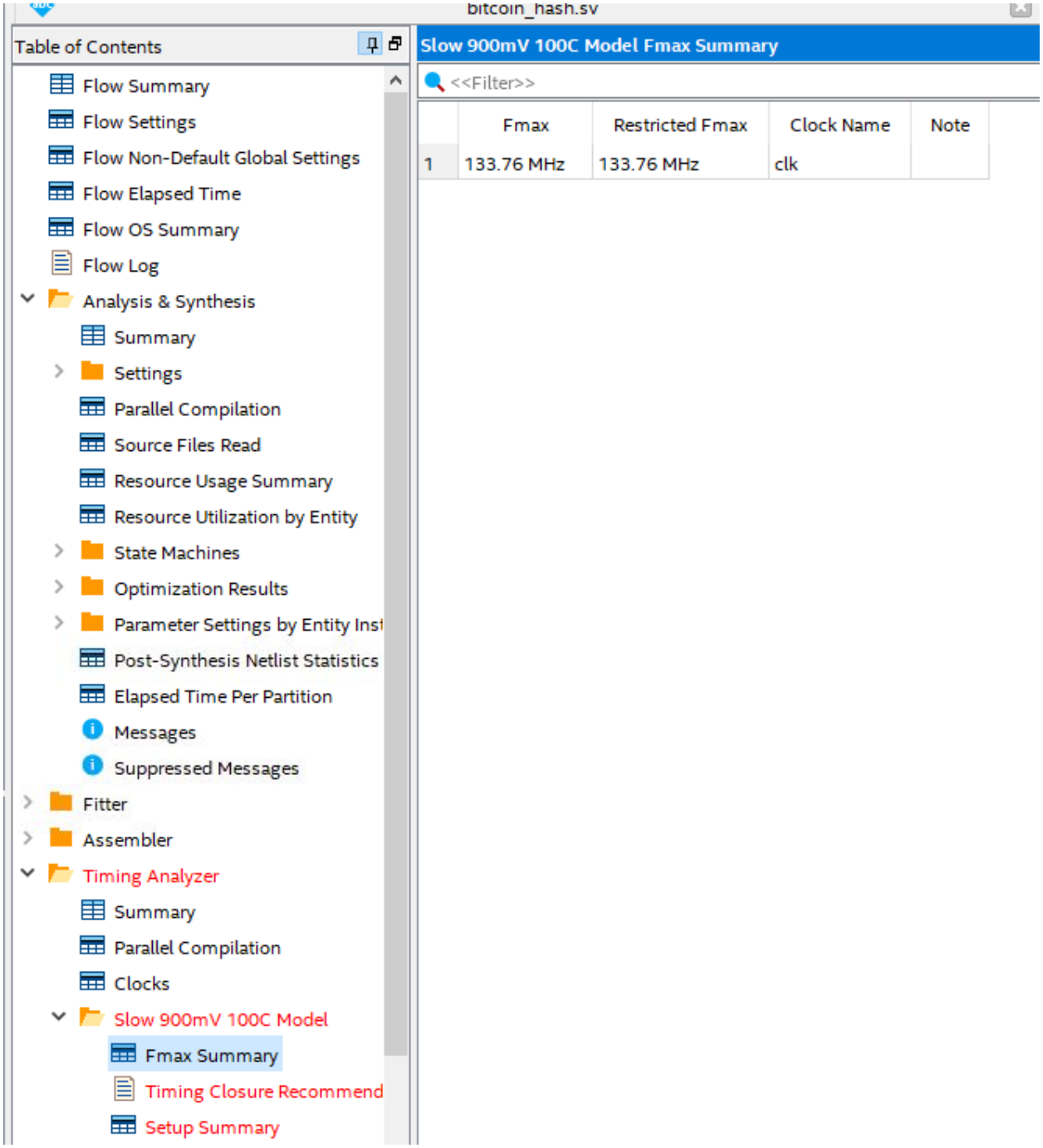
synthesis resource usage for bitcoin_hash

	Resource	Usage
1	Estimated ALUTs Used	31687
1	-- Combinational ALUTs	31687
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	18994
3		
4	Estimated ALUTs Unavailable	154
1	-- Due to unpartnered combinational logic	154
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	31687
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	875
3	-- 5 input functions	11659
4	-- 4 input functions	12
5	-- <=3 input functions	19141
8		
9	Combinational ALUTs by mode	
1	-- normal mode	23507
2	-- extended LUT mode	0
3	-- arithmetic mode	6004
4	-- shared arithmetic mode	2176
10		
11	Estimated ALUT/register pairs used	33163
12		
13	Total registers	18994
1	-- Dedicated logic registers	18994

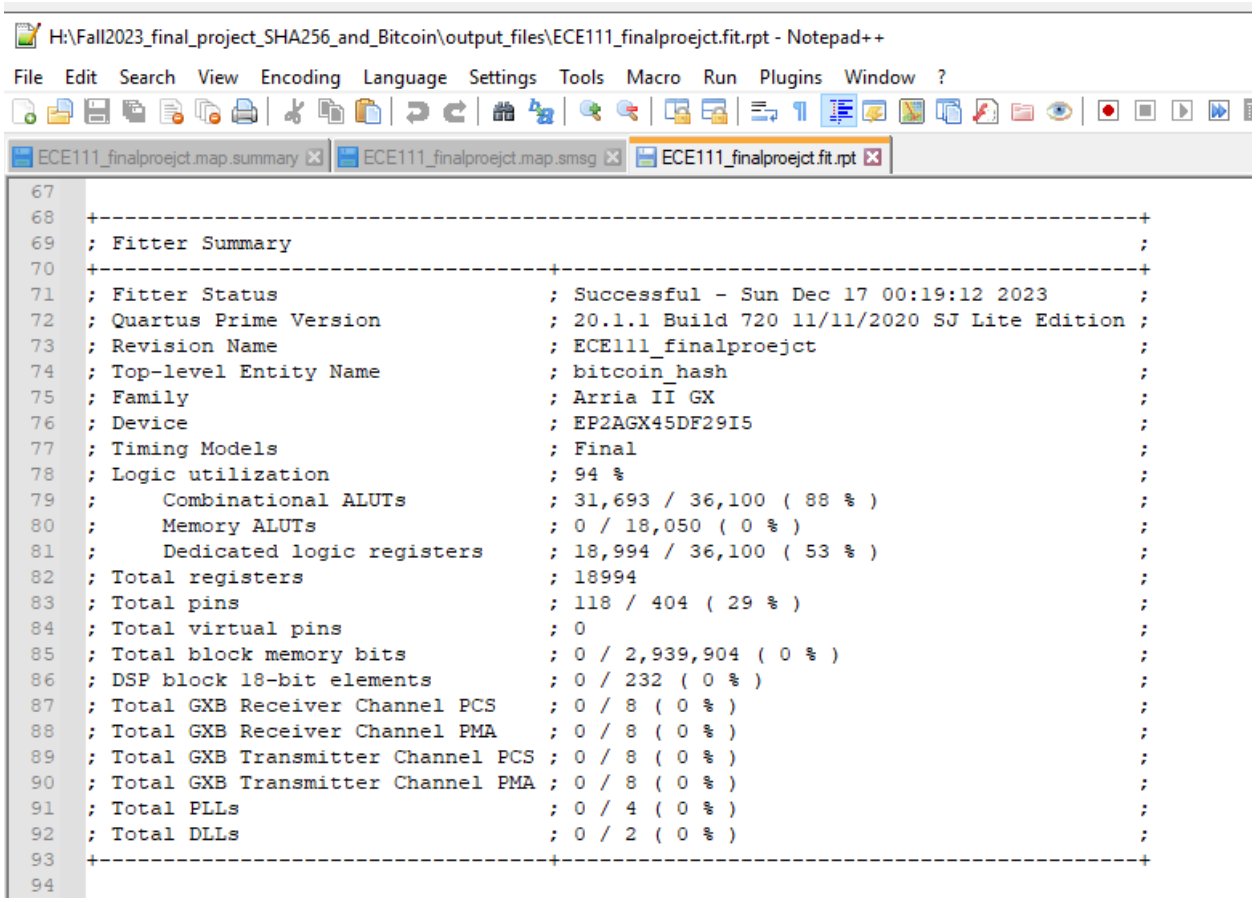
12		
13	Total registers	18994
1	-- Dedicated logic registers	18994
2	-- I/O registers	0
3	-- LUT_REGs	0
14		
15		
16	I/O pins	118
17		
18	DSP block 18-bit elements	0
19		
20	Maximum fan-out node	clk~input
21	Maximum fan-out	18995
22	Total fan-out	179811
23	Average fan-out	3.53

-- LUT_REGs

Timing Report for bitcoin hash



Fitter Report Snapshot for bitcoin hash



STA fmax for bitcoin hash

```
+-------------------------------------------------+
; Slow 900mV 100C Model Fmax Summary               ;
+------------+-----------------+------------+------+
; Fmax       ; Restricted Fmax ; Clock Name ; Note ;
+------------+-----------------+------------+------+
; 133.76 MHz ; 133.76 MHz      ; clk        ;      ;
+------------+-----------------+------------+------+
This panel reports FMAX for every clock in the design, regardless of the user-specified clock periods.
```