Final Project

Wednesday, September 15, 2021 9:52 PM

Baiqiang Zhao
A16083870
Lingpeng Meng
A16603947

Sha-256 and Bitcoin explanation

The Sha-256 I have implemented in this final project is a simplified secure hash algorithm.
It can compute eight 32-bits hash value for any message which is in 20 words.

Bitcoin is a cryptocurrency that can be traded without the need for a central authority to manage transactions. This means no bank, which in turn means no middleman, to make sure you get a real deal, not one "Fake." So how do you know if a transaction is fraudulent? You keep a public ledger of all transactions and check them. If this person has Bitcoin sent to you. If they do, the transaction is added to the ledger, you know The rich this ledger is called "blockchain". Divide the ledger into chunks and hashes. Each block, but in each block, you include the hash of the previous block. If you try to change any Trade, and then after each block, it will have to be recalculated. The hash of each new block must also be satisfied. Calculating difficult conditions, which makes it possible to change the public ledger and get others to agree to it. They Get a cut in the deal in the form of a fee. In addition, until 21,000,000 bitcoins existed, They are also rewarded. The reward was originally 50 bitcoins, but it halved for every 210,000. That's how Bitcoin is generated. Miners use the SHA256 algorithm to find potential hashing for blocks. The last item is the Bitcoin hash Processor," which uses the SHA256 algorithm, uses a few changes to the input to find many hashings using nobody.

SHA-256 and Bitcoin hashing algorithm description

SHA-256 algorithm description:

First I wrote helper function wtnew to calculate the next wt and rightrotates to rotate right the input value, and the sha256_op which does the hash operation.
In IDLE state, I initialize hash value for h0 to h7 and a to h and all other variables, and set cur_addr signal to the message_addr.

Next state goes to EMPTY and goes to READ.

In READ state, I put value from mem_read_data into message variable one word by one word. After all twenty words are inserted, it goes to BLOCK state.

In BLOCK state, block_1 = 0, so it does the first block operation. It takes 16 words from message and put into w[0] to w[15]. Then, it goes to COMPUTE state.

In COMPUTE state, it does sha256 operation for each w, after 64 times operation, variable block_1 becomes 1 and it update the h0 … h7 to its value plus the a...h respectively. Then it goes back to BLOCK state.

When we enter BLOCK state the second time, variable block_1 = 1, so it does the second block

operation. It insert the rest messages into w[0] to w[3] and set w[4] to 32'h8000000, and w[5] … w[14] to 0 and w[15] to 32'd640. Then, it updates a…h to h0 … h7. It goes to compute state again.

In COMPUTE state, it does the sha256 operation as before and updates values. But this time variable block_1 = 1, so it goes to WRITE state.

In WRITE state, it set cur_we to 1 and decrement output_addr by 1. Then we can write h0 … h7 to cur_write_data respectively. After all hash value are written, it goes to IDLE state and set done = 1. The job of SHA-256 has done.

Bitcoin hash algorithm description:

For bitcoin_hash, I did similar things as the sha_256. I have the same wtnew and rightrotate helper functions and sha256 hash round function.

In IDLE state, it initializes fh0 … fh7 and a … b given values and all other variables like Part 1 did.

It goes to EMPTY state and goes to READ state like Part 1 did.

In READ state, it is exactly same as the part 1. it takes 20 wrods from mem_read_data to message. Then it goes to BLOCK_1 state.

In BLOCK_1 state, it takes 16 words form message to w[0] …w[15] like part 1 did. Then it goes to COMPUTE state.

In COMPUTE state, it does right rotation and sha256 operation like Part 1 did. And variable first_block = 1(initialized in IDLE state), so after 64 times sha256 operation, it goes to BLOCK_2 state and updates fh0 …fh7 to its value plus a…h. It also updates variable first_block = 0 and second_block = 1.

In BLOCK__2 state, it takes rest of message to w[0]…w[2] and puts n (nonces value which at this time equals to 0) to w[3]. It sets w[4] to 32'h80000000 and w[5] … w[14] to 0 and w[15] to 32'd640. Then it stores fh0 … fh7 to h0[n] … h7[n], and updates a…h to fh0 … fh7. It goes to COMPUTE state again.

In COMPUTE state at the second time, it does sha256 operation as before, but this time after sha256 it updates h0[n] … h7[n] to its value plus a…h respectively. After that, it goes to PHASE_3 state.

In PHASE_3 state, n still equals 0 for now, the sha operation for second block has done at n = 0. It takes h0[n] …h7[n] values to w[0]…w[7] and sets w[8] to 32'h80000000 and w[9]…w[14] to 0 and w[15] to 32'd256. Then it initialize hash, sets h0[n] …h7[n] and a… h to given initial values. It goes to COMPUTE.

In COMPUTE state at the third time, it does the same thing of sha256 operation and updating values. Then, it will goes back to BLOCK_2 state and increase n by 1 to do the rest operations. The code will cycle between BLOCK_2 -> COMPUTE -> PHASE_3 -> COMPUTE, each cycle n will increase by 1.

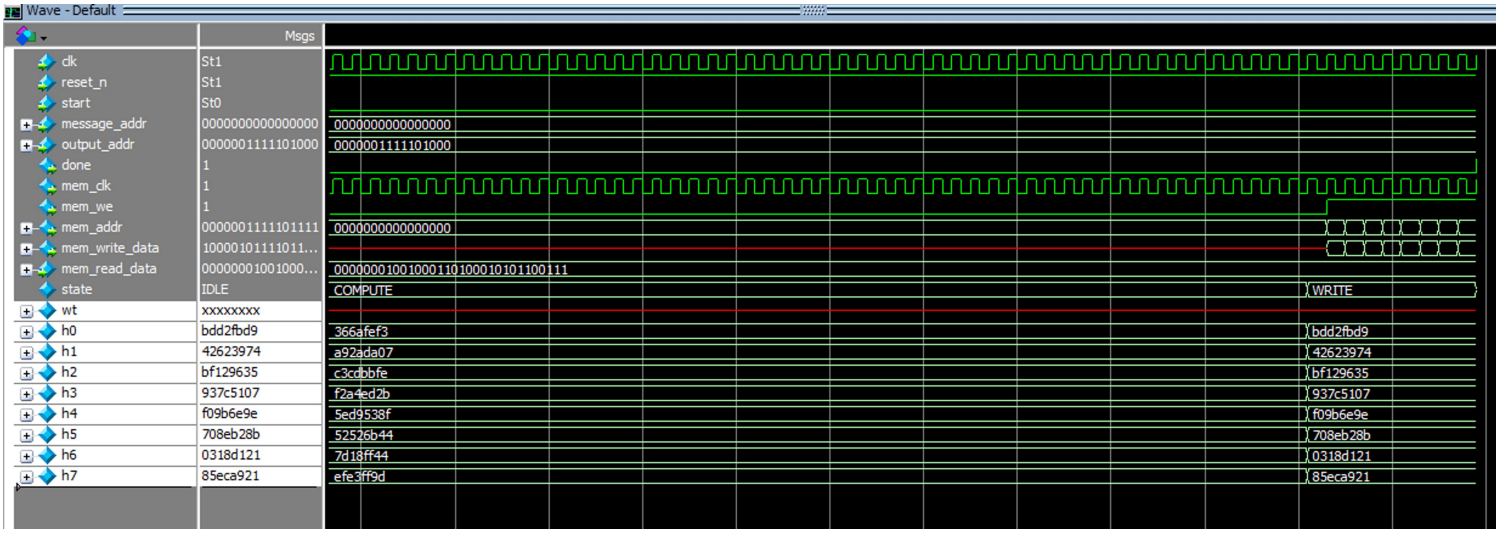
After n = num_nonces, it will goes to WRITE state.

In WRITE state, it sets cur_we to 1 and increase output_addr by 1.
And it stores h0[0] … h0[16] to cur_write_data one by one. After it has done, it goes to IDLE state and set done to 1. The job of bitcoin hash has done.

SHA-256 and Bitcoin hashing simulation waveform snapshot

Wednesday, September 15, 2021 11:06 PM

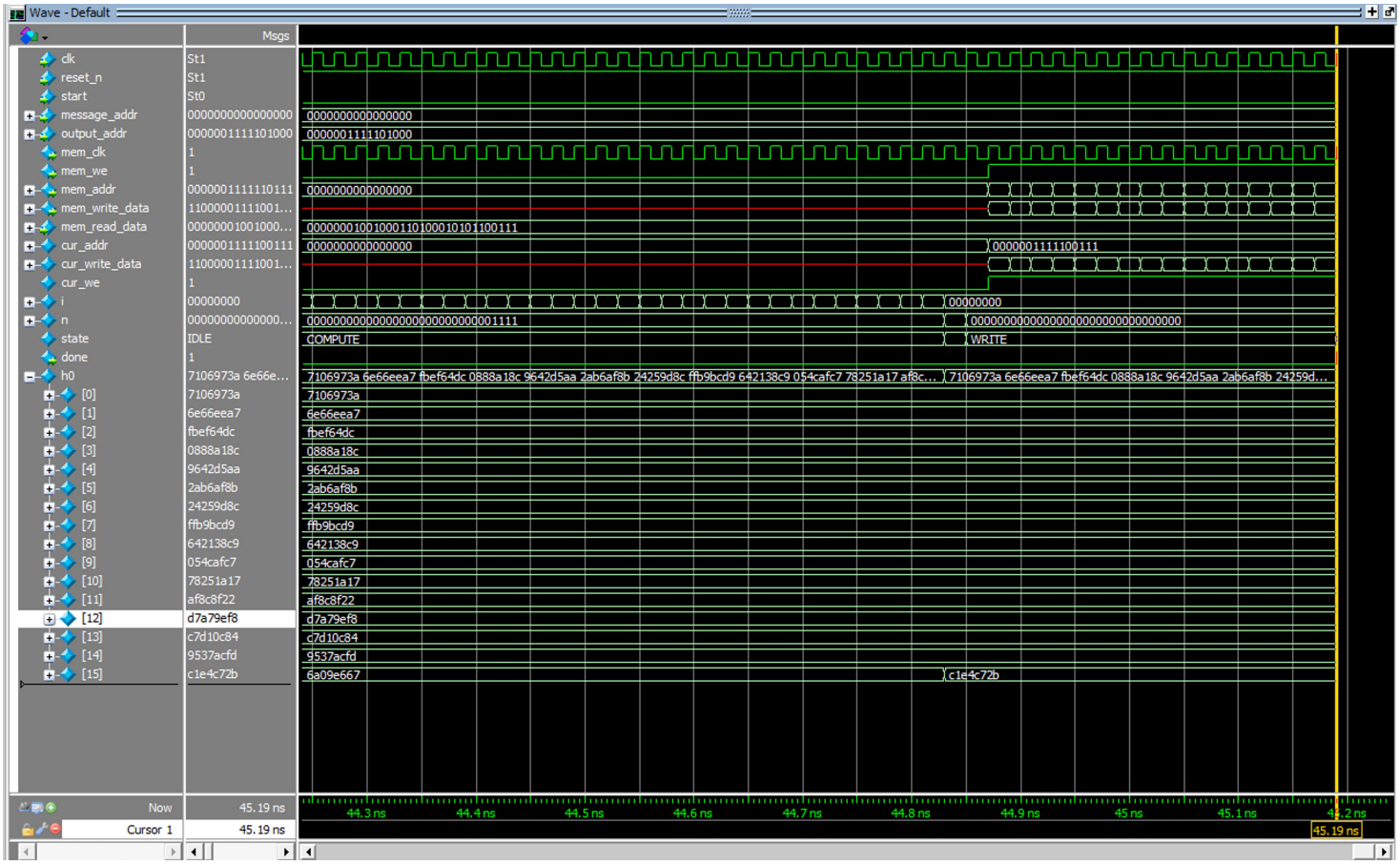
SHA-256 waveform snapshot



This is the waveform snapshot of 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.

Bitcoin hashing waveform snapshot

Thursday, September 16, 2021 12:06 AM



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.

Modelsim transcript window output for SHA-256 and Bitcoin hashing

Wednesday, September 15, 2021 11:11 PM

Simplified SHA

```
VSIM 6> 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:         202
#
#
# ****************************
#
# ** Note: $stop    : C:/Users/Sean/Desktop/ecelll/Final proje
#    Time: 4090 ps  Iteration: 2  Instance: /tb_simplified_sha
# Break in Module tb_simplified_sha256 at C:/Users/Sean/Deskto
```

Bitcoin

Thursday, September 16, 2021 12:06 AM

```
VSIM 7> 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] = c1e4c72b Your H0[15] = c1e4c72b
# ****************************
#
# CONGRATULATIONS! All your hash results are correct!
#
# Total number of cycles:        2257
#
```

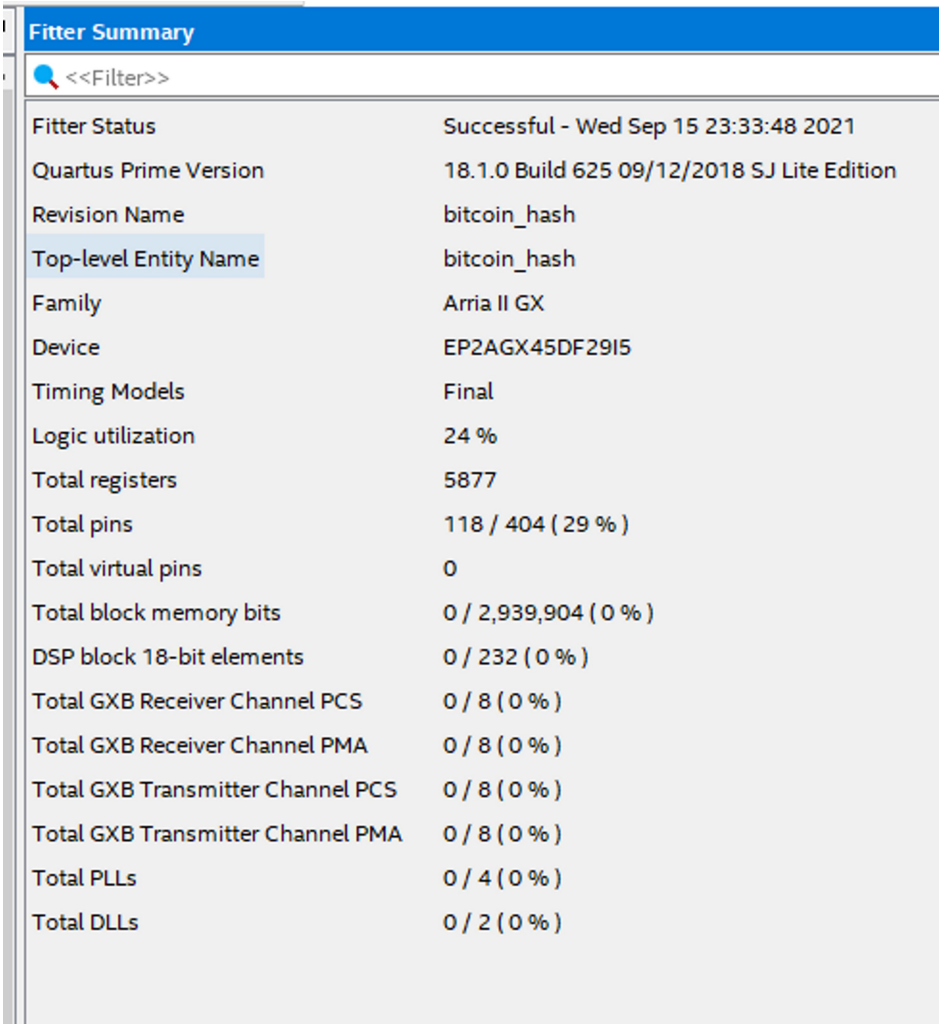
Synthesis resource usage and timing report for bitcoin_hash only.

Wednesday, September 15, 2021 11:25 PM

Synthesis resource usage

	Resource	Usage
1	Estimated ALUTs Used	3612
1	-- Combinational ALUTs	3612
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	5877
3		
4	Estimated ALUTs Unavailable	257
1	-- Due to unpartnered combinational logic	257
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	3612
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	32
2	-- 6 input functions	1852
3	-- 5 input functions	130
4	-- 4 input functions	47
5	-- <=3 input functions	1551
8		
9	Combinational ALUTs by mode	
1	-- normal mode	2788
2	-- extended LUT mode	32
3	-- arithmetic mode	664
4	-- shared arithmetic mode	128
10		
11	Estimated ALUT/register pairs used	8091
12		
13	Total registers	5877
1	-- Dedicated logic registers	5877
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	5878
22	Total fan-out	40284
23	Average fan-out	4.14

Fitter report

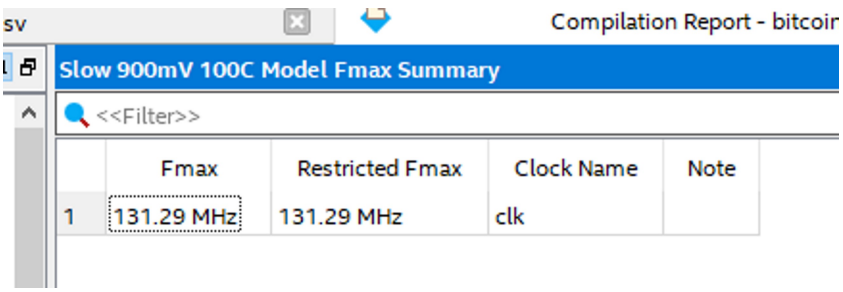


Fitter Summary

<<Filter>>

Fitter Status	Successful - Wed Sep 15 23:33:48 2021
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	bitcoin_hash
Top-level Entity Name	bitcoin_hash
Family	Arria II GX
Device	EP2AGX45DF29I5
Timing Models	Final
Logic utilization	24 %
Total registers	5877
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



sv

Compilation Report - bitcoir

Slow 900mV 100C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	131.29 MHz	131.29 MHz	clk	