SHA 256 brief explain

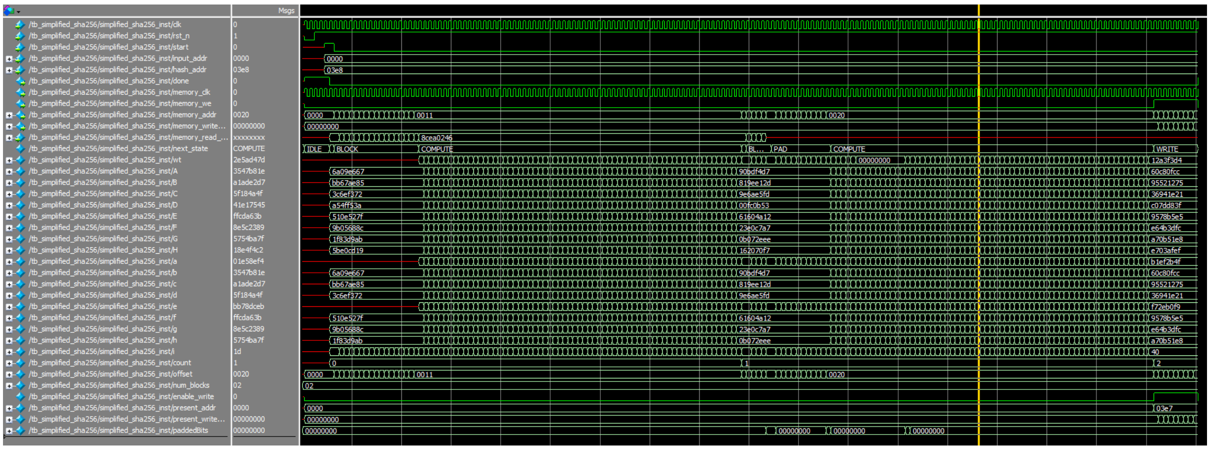
SHA 256 takes a message and creates a 256 bit hash. The process for creating this hash is one way so that it is impossible to find the input given the hash. It is also impossible to find another input that creates the same hash. Small changes in the message completely change the hash value. This can be used in many applications such as blockchain and verifying file integrity.

SHA 256 algorithm

The SHA 256 implemented uses different states to go through the algorithm. It first starts off in and IDLE state which waits for a signal to start the algorithm. Once it starts, it assigns initial values and starts to read in the message from memory. After reading in 1 block of 512 bits, it then goes through 64 rounds of processing that involve calculating w and other sha operations. This results in a first hash value. It then does this until the entire message has been computed and the final hash value is available. In addition, it will check if the message has been fully read and then will pad the message until it is Nx512 bits long. The pad consists of 1 1 bit and then however many 0 bits are needed. It ends the pad with the message size in the last 64 bits. The SHA 256 also uses a shift register to store the w that is calculated so less memory is needed.

SHA 256 waveform and transcript

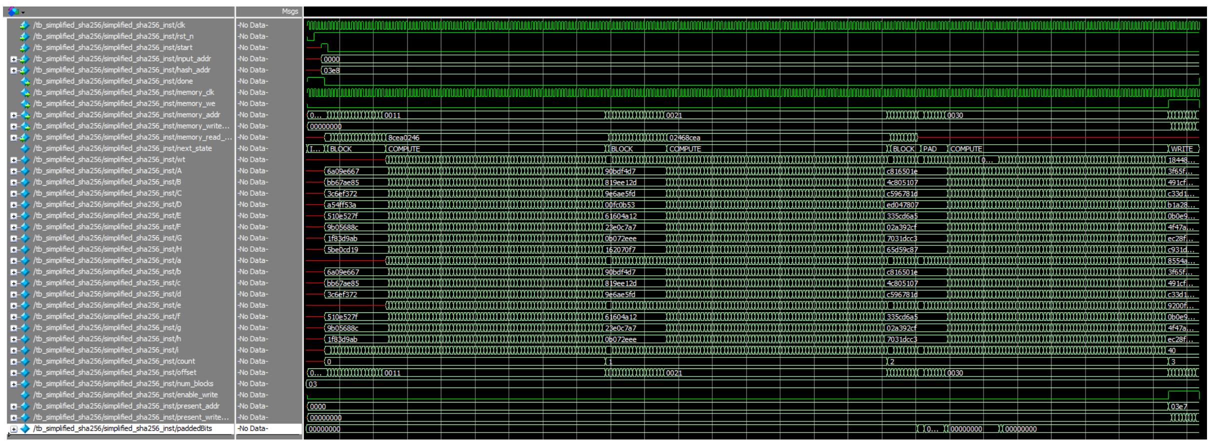
- 20 word



```
# --------
# MESSAGE:
# --------
# 01234675
# 02468cea
# 048d19d4
# 091a33a8
# 12346750
# 2468cea0
# 48d19d40
# 91a33a80
# 23467501
# 468cea02
# 8d19d404
# 1a33a809
# 34675012
# 68cea024
# d19d4048
# a33a8091
# 46750123
# 8cea0246
# 19d4048d
# 33a8091a
# ***************************
#
# ---------------------
# 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:         178
#
#
# ***************************
#
# ** Note: $stop    : C:/Users/shima/Documents/ECE 111/Fall2023_final_project_SHA256_and_Bitcoin/Final_Project/simplified_sha256/tb_simplified_sha256_20w.sv(231)
#    Time: 3610 ps  Iteration: 2  Instance: /tb_simplified_sha256
# Break in Module tb_simplified_sha256 at C:/Users/shima/Documents/ECE 111/Fall2023_final_project_SHA256_and_Bitcoin/Final_Project/simplified_sha256/tb_simplified_sha256_20w.sv line 231

VSIM 9>
```

- 30 word

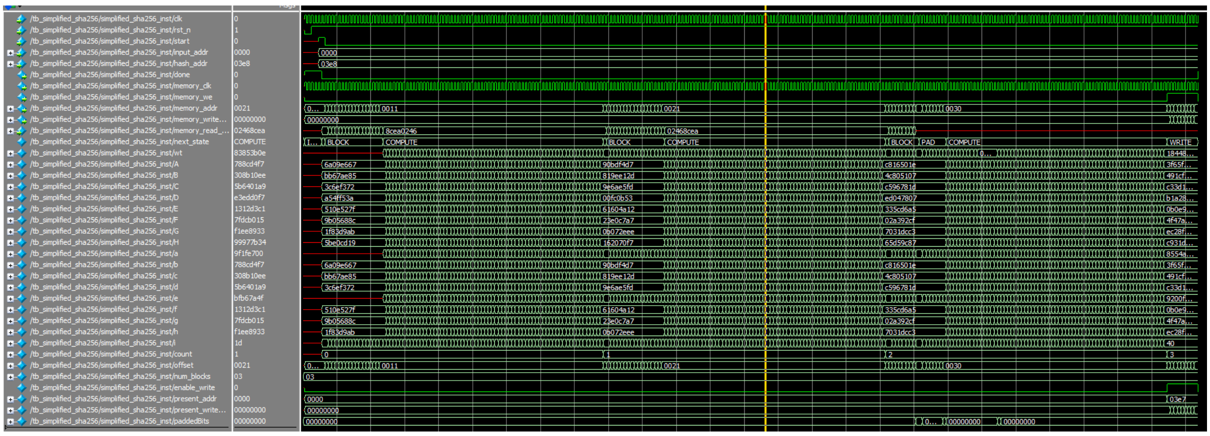


```
VSIM 12> run -all
# --------
# MESSAGE:
# --------
# 01234675
# 02468cea
# 048d19d4
# 091a33a8
# 12346750
# 2468cea0
# 48d19d40
# 91a33a80
# 23467501
# 468cea02
# 8d19d404
# 1a33a809
# 34675012
# 68cea024
# d19d4048
# a33a8091
# 46750123
# 8cea0246
# 19d4048d
# 33a8091a
# 67501234
# cea02468
# 9d4048d1
# 3a8091a3
# 75012346
# ea02468c
# d4048d19
# a8091a33
# 50123467
# a02468ce
# ***************************
#
# ---------------------
# 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:         261
#

# ***************************
#
# ** Note: $stop    : C:/Users/shima/Documents/ECE 111/Fall2023_final_project_SHA256_and_Bitcoin/Final_Project/simplified_sha256/tb_simplified_sha256_30w.sv(232)
#    Time: 5270 ps  Iteration: 2  Instance: /tb_simplified_sha256
# Break in Module tb_simplified_sha256 at C:/Users/shima/Documents/ECE 111/Fall2023_final_project_SHA256_and_Bitcoin/Final_Project/simplified_sha256/tb_simplified_sha256_30w.sv line 232

VSIM 13>
```

- 40 word



```
VSIM 16> run -all
# --------
# MESSAGE:
# --------
# 01234675
# 02468cea
# 048d19d4
# 091a33a8
# 12346750
# 2468cea0
# 48d19d40
# 91a33a80
# 23467501
# 468cea02
# 8d19d404
# 1a33a809
# 34675012
# 68cea024
# d19d4048
# a33a8091
# 46750123
# 8cea0246
# 19d4048d
# 33a8091a
# 67501234
# cea02468
# 9d4048d1
# 3a8091a3
# 75012346
# ea02468c
# d4048d19
# a8091a33
# 50123467
# a02468ce
# 4048d19d
# 8091a33a
# 01234675
# 02468cea
# 048d19d4
# 091a33a8
# 12346750
# 2468cea0
# 48d19d40
# 91a33a80
# **************************
#
# --------------------
# 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:         261
#
#
# **************************
#
# ** Note: $stop    : C:/Users/shima/Documents/ECE 111/Fall2023_final_project_SHA256_and_Bitcoin/Final_Project/simplified_sha256/tb_simplified_sha256_40w.sv(231)
#    Time: 5270 ps  Iteration: 2  Instance: /tb_simplified_sha256
# Break in Module tb_simplified_sha256 at C:/Users/shima/Documents/ECE 111/Fall2023_final_project_SHA256_and_Bitcoin/Final_Project/simplified_sha256/tb_simplified_sha256_40w.sv line 231
```

Bitcoin Hash brief explain

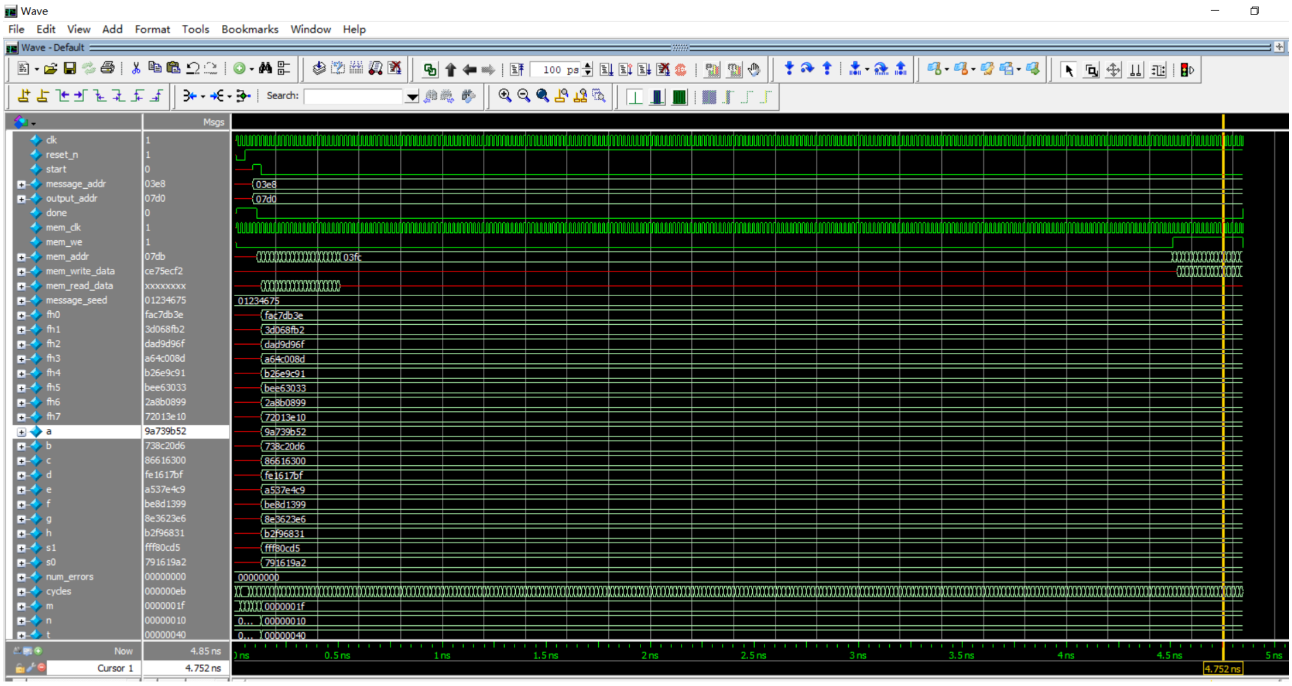
Bitcoin is a type of blockchain which is a chain of digital blocks that can store information. The blocks are chained together through the use of cryptographic hashing algorithms. Bitcoin uses SHA256 hashing. When blocks are chained, their data can no longer be changed and everyone can see the entire blockchain. Bitcoin is the oldest blockchain and holds data from all bitcoin transactions. When a transaction is made, a signature is made which is included in the data of the blocks which links them. In addition, the signature must start with a certain sequence. Part of the data called the nonce is changed randomly to find a matching signature. Furthermore, the transaction data can’t be changed since the signatures will no longer match. A corrupt miner who changes a block of transactions would have to calculate new signatures for all subsequent blocks for the change to be accepted. This would be impossible since the rest of the network will have much more computational power and the corrupt miner would not be able to catch up.

Bitcoin Hash algorithm

For the bitcoin hash, we created a helper module(sha_256) to hash. We use 1 module(block 1) to hash the first 16 words, and then we create 16 modules(block 2) in parallel to compute the remaining hashes with 16 nounces together. In the IDLE state, when start is signaled, we initialize the values we need, like offset, control signals for the hashing modules. Then we go to READ. In the READ state, we read the 19 words of the input message and turn on the start signal for block 1 module. This will compute the hash for the first 16 words. Then, we go to WAIT1. In the WAIT1 start, we just turn off the start signal for block 1, and we go to FIRST. In the FIRST state, we wait until block 1 module to finish. Once it is done, this means that we have the first 16 words hash. Then, we turn on the start signal for the 16 modules. Then, we go to WAIT2. In the WAIT2 state, we turn off the start signal for the 16 modules, go to SECOND. In the SECOND state, we just wait until the 16 modules are done. Once they are done, we will go to the WRITE state, and write the results to memory.

The helper module sha_256 takes in 19 words message, the nounce for second hash, pre_hash(previously calculated hash), and a logic that indicates this is first hash or second hash. When this is the first hash, load w with the first 16 words in message and compute. If the second hash, load w with the last 3 words, nounce, etc. Then, compute the hash for phase 2, and compute phase 3 directly after phase 2. Then, the result will be sent out, and done is signaled to 1.

Bitcoin Hash waveform



Bitcoin Hash transcript

```
VSIM 5> run -all
# ---------------
# 19 WORD HEADER:
# ---------------
# 01234675
# 02468cea
# 048d19d4
# 091a33a8
# 12346750
# 2468cea0
# 48d19d40
# 91a33a80
# 23467501
# 468cea02
# 8d19d404
# 1a33a809
# 34675012
# 68cea024
# d19d4048
# a33a8091
# 46750123
# 8cea0246
# 19d4048d
# ***************************
#
# ---------------------
```

```
# ***************************
#
# ---------------------
# 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:         240
#
#
# ***************************
#
# ** Note: $stop    : C:/Users/POOTIS/Desktop/ece 111/Final_Project/bitcoin_hash/tb_bitcoin_hash.sv(334)
#    Time: 4850 ps  Iteration: 2  Instance: /tb_bitcoin_hash
# Break in Module tb_bitcoin_hash at C:/Users/POOTIS/Desktop/ece 111/Final_Project/bitcoin_hash/tb_bitcoin_hash.sv line 334
```

Bitcoin Hash Resource Usage

	Resource	Usage
1	Estimated ALUTs Used	22999
1	-- Combinational ALUTs	22999
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	14523
3		
4	Estimated ALUTs Unavailable	130
1	-- Due to unpartnered combinational logic	130
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	22999
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	1
2	-- 6 input functions	2974
3	-- 5 input functions	1032
4	-- 4 input functions	7
5	-- <=3 input functions	18985
8		
9	Combinational ALUTs by mode	
1	-- normal mode	12079
2	-- extended LUT mode	1
3	-- arithmetic mode	9831
4	-- shared arithmetic mode	1088
10		
11	Estimated ALUT/register pairs used	26961
12		
13	Total registers	14523
1	-- Dedicated logic registers	14523
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	14524
22	Total fan-out	142636
23	Average fan-out	3.78

Bitcoin Hash Fitter Report

Fitter Summary

<<Filter>>

Fitter Status	Successful - Fri Dec 15 23:52:54 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	ece111_hw1
Top-level Entity Name	bitcoin_hash
Family	Arria II GX
Device	EP2AGX45DF29I5
Timing Models	Final
Logic utilization	87 %
Total registers	14523
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 %)

Bitcoin Hash F_{max}

Slow 900mV 100C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	112.44 MHz	112.44 MHz	clk	