ECE 111 Final Project

SHA256 and bitcoin hashing

Names: Chengtao Wu, Max Shi

SHA256 - Description

- Takes input (<=2^{64} bits) and creates a unique 256 bit output called the hash value/message digest

Steps:

1. Append padding bits of 1, 0, and size of message so that the message is a multiple of 512
2. Initialize message digest buffers
3. Process message in blocks of 512 bits
4. Each block is processed in 64 rounds
5. After all blocks are processed, 256 bit hash value is outputted

SHA256 - Design objectives and constraints

- Objective: best F_{max} with area
  - Want to use least number of ALUTs and registers while completing the task as quickly as possible
- Must be able to work with messages of different number of words
- Can only read/write 32 bits from memory in one cycle
- No inferred megafunctions or latches

SHA256 - Design Challenges

Challenge	How it was solved
● Getting message from memory and padding memory bits	● Add delay before first BLOCK state ● Use state for padding
● Various bugs	● Use $display and review waveform to find the issue
● Large number of ALUTs and registers used initially	● This was due to computing all 64 rounds of processing at once. Changed to processing 1 round at a time

Can compare message to correct message to see problems

SHA256 - Optimizations

- Store w in 16 32 bit array instead of 64 32 bit array
- Possible further optimizations:
  - Parallel processing for the 64 rounds
    - 64 at a time is too much area
    - Could do two at a time

SHA256 - Results

20 word:

ALUTs: 1496

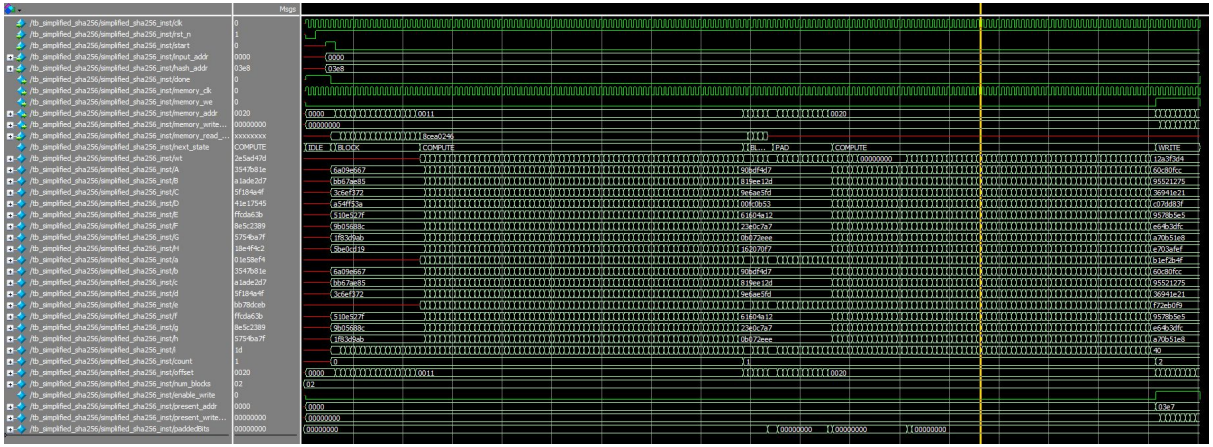
Registers: 1618

F_{max}: 114.1 MHz

Cycles: 178

Delay: 1.56 microseconds

Area*Delay: 4.86



```
# --------
# 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>
```

Bitcoin - Description

- Blocks are linked to each other
- Hash created from block data to link blocks
- Hash must start with certain sequence
- Nonce is changed until a hash that meets the requirement is created

Bitcoin - Design objectives and constraints

- Objective: best F_{max} with area
- Create hash for 20 word input
- 2 blocks so 2 hashes are needed
- Nonce values will only be from 0 to 15, must compute all of them
- Write final H0 of the 16 hashes created
- No inferred megafunctions or latches

Bitcoin - Design Challenges

Challenge	How it was solved
● Complexity is too high when implementing everything in the same file.	● Have a module call sha_256 that computes hashing.
● Various bugs	● Use $display and review waveform to find the issue

Bitcoin - Optimizations/parallelization

Possible optimizations

- Compute hashes in parallel
- Pipeline hash of first bitcoin block and hash of second bitcoin block

Bitcoin - Results

ALUTs: 22999

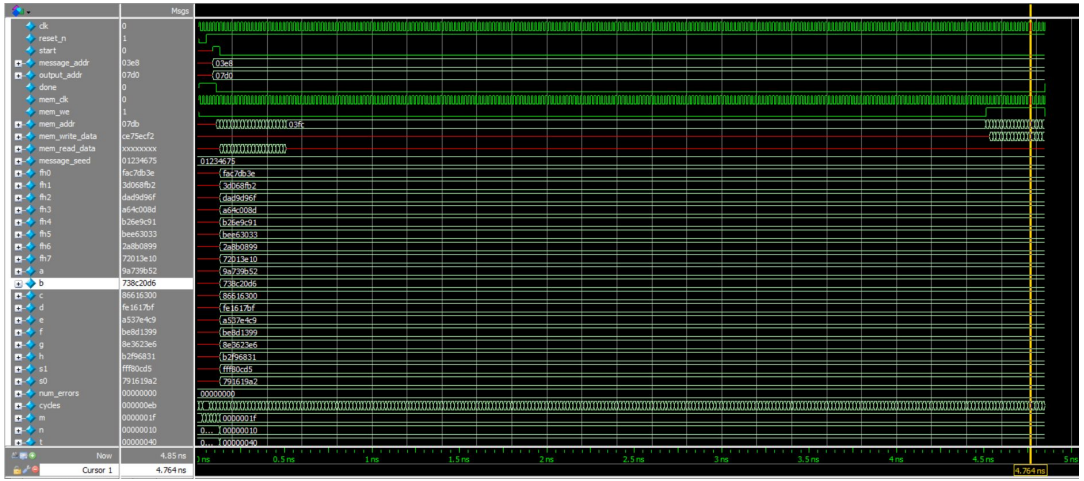
Registers: 14523

F_{max}: 112.44 MHz

Cycles: 240

Delay: 2.134 microseconds

Area*Delay: 80.09




```
# ---------------------
# 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
```

Conclusion

Both of these parts of the project provided valuable insight into designing hardware in System Verilog. They provided challenges in implementation and we encountered many problems that we needed to overcome. We were also able to attempt to optimize the bitcoin and sha 256 projects which taught us a lot about the difficulties and tradeoffs one must consider when designing hardware. We were able to successfully optimize the designs but there are more optimizations that could be done as well. This project was very helpful in widening our understanding of digital design.