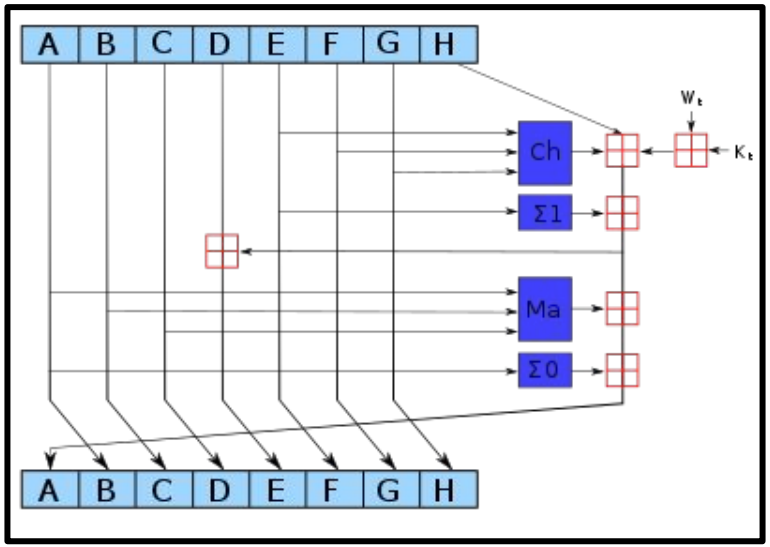
ECE 111 Final Project

By: Sean Fuhrman
and Nathaniel Greenberg

Project description (Sha-256)

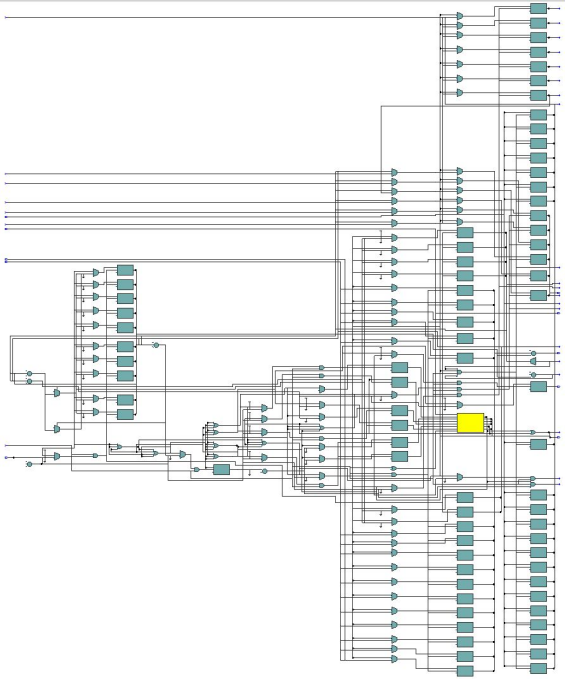


- SHA-256, is a cryptographic hash function
- Goal is to compute a unique hash value for any input “message”, where a “message” can be anything.
- It generates a fixed-size 256-bit (32-byte) hash output from input data, ensuring consistency and determinism.
- SHA-256 is designed to be resistant to collision attacks and pre-image attacks, making it a secure choice for cryptographic applications
- Widely used in blockchain systems like Bitcoin, SHA-256 forms the basis of the proof-of-work algorithm, securing transactions and linking blocks
- Employed in digital signatures and certificate authorities, SHA-256 verifies the authenticity and integrity of digital documents, software, and communication

Design objectives and constraints

- Objectives: Minimize the delay and area for efficient usage in the bitcoin hash circuit
- Constraints:
- No inferred megafunctions or latches
- The same input must always generate the same output.
- The hash should be of a fixed number of characters, regardless the size or type of input data
- There should be no way to reverse the hashing process to see the original data set.
- Any change in the input must produce an entirely different output
- Practically impossible to find two different inputs that produce the same output
- Creating the hash should be a fast process that doesn't make heavy use of computing power

Design Challenges, and how do you overcome those?



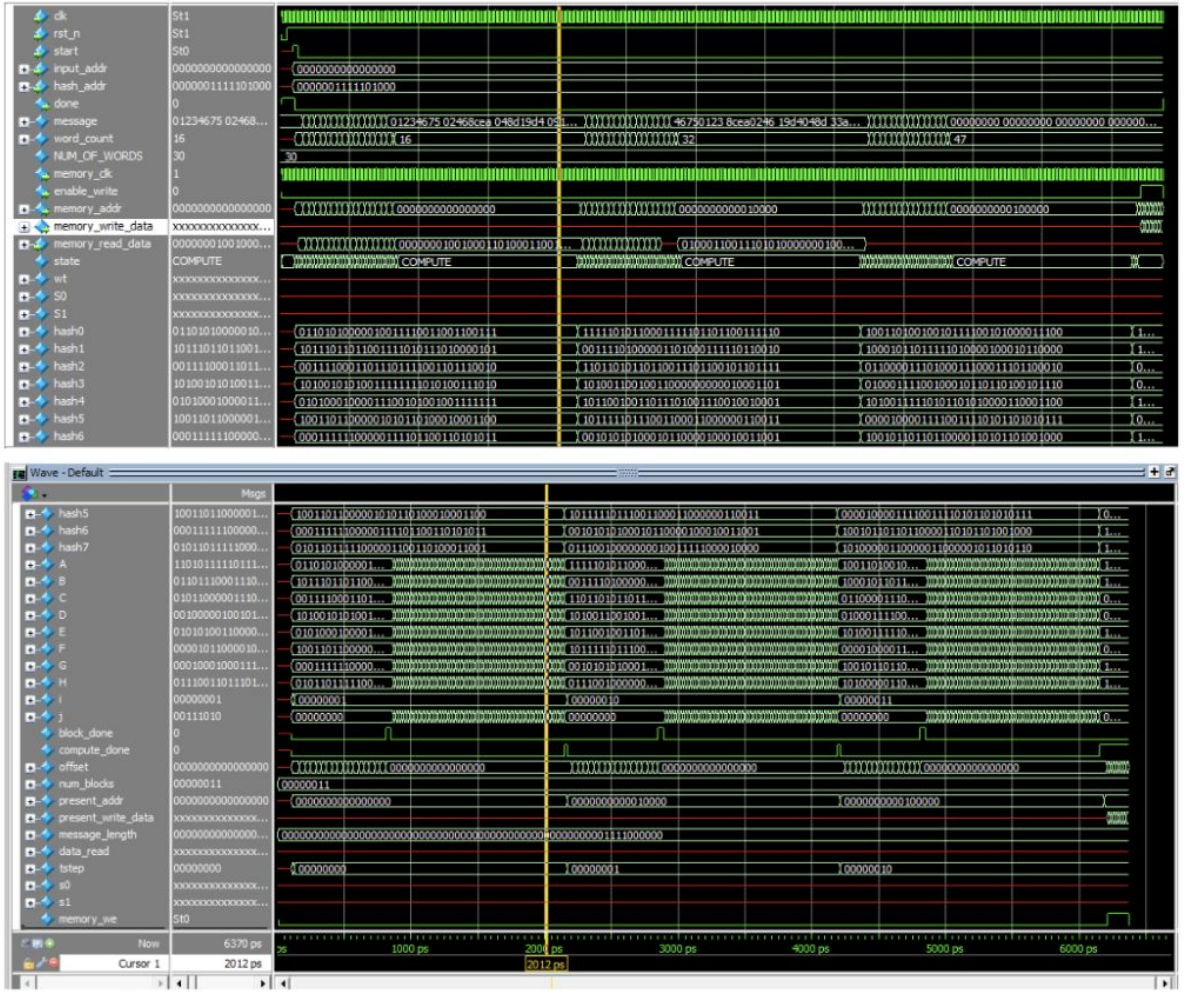
- Accurately adding padding
  - We ran into the problem of adding padding correctly, specifically for 30 words
  - To overcome this we had to redesign our padding implementation to account for when the padding must be added to create a new block
- Blocking vs. Non-Block
  - We decided to use an always_ff block with non-blocking statements, this led to a lot of inferred latches in our initial implementation because of bad syntax
  - We eventually found the errors and fix this

Optimizations

- W optimization
  - We used a w of size 16 with shifting during execution. This greatly reduced our ALUT number
- States
  - We found that using a few extra states help simplify our code, and while this may not be most efficient it proved very beneficial for implementation.
  - Simple code may lead to more optimized code anyway

Results

Example wave form for 30 words:



Project description (Bitcoin Hashing)

- The project involves Bitcoin mining, where the system reads a 19-word block header and computes the double SHA-256 hash for 16 different nonce values.
- The resulting hash values are then stored in memory, starting from a specified address.
- The process is repeated until completion, and a "done" flag is set to indicate the end of computation.
- This project mirrors the fundamental process of Bitcoin mining, where miners search for a nonce that results in a hash value meeting certain criteria to add a new block to the blockchain.

Design objectives and constraints

- Objective:
  - Minimize the delay and area
- Constraints:
  - No inferred latches
  - No inferred megafunctions
  - Operate on 20 word bitcoin header block.
  - 16 nounces to cycle through
  - Use Sha256 to create output hash for input block
  - Data block is 620 bits so it requires 2 bitcoin blocks for 1 input message.

Design Challenges, and how do you overcome those?

- How do we efficiently generate 16 hashes for each block
  - Parallelize the generation of all 16 hashes, however this could be space inefficient
- What states should we use?
  - We ended up using 7 states and specifically had states to handle each (parallel) block that needed to be processed. This proved efficient as the set header size means always 3 computation stages are necessary, and making them separate states allowed us to simplify our code.

Optimizations / parallelization

- Parallelize the nounces calculations to speed up clock cycles at sacrifice of area
  - We used many variables with arrays of [num_nonces] size to implement parralelization
- Use W of size 12 to optimize all the word blocks needed

Results

Same waveform result of one header: