# Final report SHA-256 is and Bitcoin hashing explanation:

**Ka Wing Yan A17177676**

The final project part 1 is an implementation of the SHA-256 hashing algorithm in System Verilog. SHA-256 is a cryptographic hash function that takes an input message and produces a fixed-size (256-bit) hash value that is typically represented as a sequence of eight 32-bit words.

The final project part 2 is an implementation of the Bitcoin hashing algorithm in System Verilog. which is designed to perform Bitcoin hashing using chained SHA-256 algorithms. The goal is to hash an input message, which consists of 20 words (19 words and 1 nonce value) and return an output of eight 32-bit words.

# SHA-256 and Bitcoin hashing algorithm: description:SHA-256 algorithm description:

Module Definition and Parameters:

The module is named simplified\_sha256. It takes various input signals like clk, reset\_n, start, message\_addr, and output\_addr.It provides output signals such as done, mem\_clk, mem\_we, mem\_addr, mem\_write\_data, and mem\_read\_data. NUM\_OF\_WORDS (default is 20) is a parameter that specifies the number of words in the input message.

Finite State Machine (FSM):

The module uses an FSM to control its operation. The FSM has several states:

IDLE: The initial state where the module is waiting for a start signal.

LOAD: Reads the input message from memory into the module.

READ: Reads data from memory.

BLOCK: Processes data in 512-bit blocks.

WAIT: Prepares data for the next block.

COMPUTE: Performs SHA-256 hash computations.

WRITE: Writes the final hash value back to memory.

Local Variables:

The module declares various local variables, including message storage, hash values (h0 through h7), loop counters (i and j), offsets, and others.

Initial State (IDLE): When the start signal is asserted, the module initializes hash values h0 through h7, sets up some variables, and transitions to the LOAD state.

LOAD State: In this state, the module reads data from memory into its message buffer. Once all the required data is loaded, it transitions to the BLOCK state.

BLOCK State: The BLOCK state processes the data in 512-bit blocks. It loads data from the message buffer and prepares it for hashing.

WAIT State: In the WAIT state, the module prepares data for the next block and computes values like P.

COMPUTE State: The COMPUTE state performs the SHA-256 hash computations using various helper functions (sha256\_op, Word\_Expansion). It iterates through 64 rounds for each 512-bit block.

WRITE State: In the WRITE state, the final hash values are written back to memory. Once all hash values are written, the module transitions back to the IDLE state.

Helper Functions:

The module defines two helper functions, sha256\_op, and Word\_Expansion, which assist in

performing the hash computations and expanding words, respectively.

Assignments and Expressions:

Various assignments and expressions are used to manipulate data and signals within the module.

Final Output (done):

The done signal indicates when the SHA-256 hash computation is complete, and the module is in the IDLE state.

The code implements the core functionality of the SHA-256 algorithm, including message loading, block processing, hash computation, and result writing. The actual SHA-256 operations are carried out in the COMPUTE state, with values stored in the h0 to h7 variables. The final hash values are written back to memory in the WRITE state.

# Bitcoin hash algorithm description:

Module Parameters:

num\_nonces and num\_words: These parameters specify the number of nonces (16) and words in the message (21).

SHA-256 Constants:

The code defines an array k[64] to hold the SHA-256 constants used in the algorithm. These constants are necessary for the chained SHA-256 computations.

Finite State Machine (FSM):

The module uses an FSM with eight states (IDLE, LOAD, PHASE\_1, PHASE\_2, PHASE\_3, WAIT\_1, WAIT\_2, WRITE) to control the Bitcoin hashing process.

Local Variables:

Various local variables, including w[16][16], message[32], ha[8], hb[16][8], offset, cur\_we, cur\_addr, cur\_write\_data, start\_a, start\_b, in\_h, finish\_a, and finish\_b[16], are used for intermediate computations and managing the hashing process.

Memory Interface:

The module interfaces with memory using signals mem\_clk, mem\_we, mem\_addr, mem\_write\_data, and mem\_read\_data to read the input message and write the computed hash back to memory.

FSM Operation:

The FSM controls the flow of the Bitcoin hashing process.

It begins in the IDLE state, initializes variables, and transitions to the LOAD state when start is asserted.

Load Phase (LOAD and PHASE\_1):

In the LOAD state, the module reads the input message from memory in 8-byte chunks.

When the message is fully loaded, it adds the necessary padding and transitions to PHASE\_2.

Phase 2 (PHASE\_2):

PHASE\_2 prepares the message for nonce variations. It initializes nonce-specific values in w.

Phase 3 (PHASE\_3):

PHASE\_3 prepares the w values for the final SHA-256 hashing.

Wait States (WAIT\_1 and WAIT\_2):

These states provide time for the chained SHA-256 algorithms to complete their computations.

Write State (WRITE):

In the WRITE state, the module writes the resulting hash values back to memory in 8-byte chunks.

Instantiation of simplified\_sha256 Modules:

The code instantiates a main simplified\_sha256 module called insha to perform the first round of SHA-256 hashing. It also generates 16 additional instances of simplified\_sha256 modules (inside a generate block) to handle nonce-specific SHA-256 computations in parallel.

done Signal:

The done signal is set when the FSM reaches the IDLE state, indicating that the Bitcoin hashing process is complete.

# SHA-256 and Bitcoin hashing simulation waveform snapshot SHA-256 waveform snapshot

# A screen shot of a computer Description automatically generatedA screen shot of a computer Description automatically generatedA screen shot of a graph Description automatically generatedA grid with green and white lines Description automatically generatedA screen shot of a computer Description automatically generatedA screen shot of a graph Description automatically generatedA screen shot of a black screen Description automatically generatedA screen shot of a computer Description automatically generatedA screen shot of a computer Description automatically generated

This is the waveform snapshot of the 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 at 3650 ns.

# Bitcoin hashing waveform snapshot

# A screen shot of a computer Description automatically generated

# A screen shot of a computer Description automatically generatedA screen shot of a computer Description automatically generatedA screen shot of a computer Description automatically generatedA screen shot of a graph Description automatically generatedA screen shot of a computer Description automatically generated

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 at around 5170ns.

# Modelsim transcript window output for SHA-256 and Bitcoin hashing

Simplified SHA

A screenshot of a computer

Description automatically generated

Bitcoin

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

# Synthesis resource usage and timing report for bitcoin\_hash only.

Synthesis resource usage

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Fitter report

A screenshot of a computer

Description automatically generated

Timing report

A screenshot of a computer

Description automatically generated