**SHA-256 Hardware Implementation**

**Project Overview:**

This document provides an in-depth explanation of the hardware implementation of the SHA-256 (Secure Hash Algorithm 256-bit) cryptographic hash function using structural modeling in Verilog HDL. The design adopts a modular approach, ensuring clarity in hierarchy and ease of debugging. The implementation processes input messages to generate secure 256-bit hash values, which can be used for cryptographic applications, data integrity verification, and digital signatures.

**Module Hierarchy:**

The SHA-256 implementation is structured into multiple functional modules, each performing a specific role in the hashing process:

1. **sha256\_top**
   1. input\_module
      1. Message buffer
      2. Padding unit
      3. Input controller
   2. message\_schedule\_module
      1. Message expansion logic
      2. Word generator
      3. Schedule buffer
   3. compression\_module
      1. State registers
      2. Logical operations unit
      3. Round processing unit
   4. output\_module
      1. Hash register
      2. Output controller
      3. Interface logic.

**Detailed Module Descriptions**

**1. Input Module**

The Input Module is responsible for message ingestion, padding, and preparing the message for further processing.

**Key Components:**

* **Message Buffer**
  + Implements 512-bit buffer for message blocks
  + Handles message segmentation
  + Provides flow control signals
* **Padding Unit**
  + Adds '1' followed by zeros
  + Appends message length as 64-bit value
  + Ensures message length is congruent to 448 modulo 512
* **Input Controller**
  + Manages input flow control
  + Coordinates with message schedule module
  + Generates control signals for buffer and padding operations
* **Interface Signals:**

///verilog

module sha256\_input (

input wire clk, rst\_n,

input wire start,

input wire [511:0] block\_in,

output reg input\_valid,

output reg [511:0] block\_out,

output reg ready,

input wire next\_stage\_ready );

**2. Message Schedule Module:**

Generates the expanded message schedule required for SHA-256 processing.

**Key Components:**

* **Message Expansion Logic**
  + Implements the message schedule array W[0..63]
  + Performs word expansion using σ0 and σ1 functions
  + Schedule calculation: W[t] = σ1(W[t-2]) + W[t-7] + σ0(W[t-15]) + W[t-16]
* **Word Generator**
  + Implements the σ0 and σ1 functions:
  + σ0(x) = ROTR^7(x) ⊕ ROTR^18(x) ⊕ SHR^3(x)
  + σ1(x) = ROTR^17(x) ⊕ ROTR^19(x) ⊕ SHR^10(x)
* **Schedule Buffer**
  + 64-word buffer for expanded message
  + Shift register implementation
  + Pipeline stage management
* **Interface Signals:**

///verilog

module sha256\_message\_scheduler (

input wire clk, rst\_n,

input wire input\_valid,

input wire [511:0] block\_in,

output reg scheduler\_valid,

output reg [255:0] w\_out,

output reg scheduler\_ready );

**3. Compression Module:**

Implements the core SHA-256 compression function and state updates.

**Key Components:**

* **State Registers**
  + Eight working variables (a, b, c, d, e, f, g, h)
  + Implementation of state update logic
  + Pipeline registers for intermediate values
* **Logical Operations Unit**
  + Implements Ch(x,y,z) = (x ∧ y) ⊕ (¬x ∧ z)
  + Implements Maj(x,y,z) = (x ∧ y) ⊕ (x ∧ z) ⊕ (y ∧ z)
  + Implements Σ0(x) = ROTR^2(x) ⊕ ROTR^13(x) ⊕ ROTR^22(x)
  + Implements Σ1(x) = ROTR^6(x) ⊕ ROTR^11(x) ⊕ ROTR^25(x)
* **Round Processing Unit**
  + 64 rounds of compression function
  + Round constant (K[t]) integration
  + State update calculation
* **Interface Signals:**

///verilog

module sha256\_compressor (

input wire clk,

input wire rst\_n,

input wire start,

input wire [511:0] message\_block,

input wire [255:0] initial\_hash,

output reg [255:0] hash\_out,

output reg done,

output reg ready\_for\_scheduler );

**4. Output Module:**

Manages final hash value generation and output interface.

**Key Components:**

* **Hash Register**
  + 256-bit hash value storage
  + Hash value accumulation
  + Final state management
* **Output Controller**
  + Output timing control
  + Interface protocol management
  + Status signal generation
* **Interface Logic**
  + Hash value formatting
  + Output validation
  + Ready/valid handshaking
* Interface Signals:

///verilog

module sha256\_output (

input wire clk, rst\_n,

input wire compression\_valid,

input wire [255:0] state\_in,

output reg [255:0] hash\_out,

output reg done,

output reg ready\_for\_next );

**5. Top Module Integration:**

The top module (sha256\_top) integrates all sub-modules and provides the main interface to the external system.

module sha256\_top (

input wire clk,

input wire rst\_n,

input wire start,

input wire [511:0] data\_in,

output wire [255:0] hash\_out,

output wire done,

// Internal signals for module interconnection

output wire input\_valid,

output wire [511:0] input\_block,

output wire scheduler\_valid,

output wire [255:0] scheduler\_words,

output wire compression\_valid,

output wire [255:0] compression\_state,

output wire ready\_for\_next,

output wire input\_ready,

output wire scheduler\_ready,

output wire compression\_ready,

wire gated\_clk,

reg clock\_enable );

// Module instantiations

// Input stage

sha256\_input input\_module (

.clk(gated\_clk), .rst\_n(rst\_n\_sync),

.start(start), .block\_in(data\_in),

.input\_valid(input\_valid),

.block\_out(input\_block),

.ready(input\_ready),

.next\_stage\_ready(scheduler\_ready) );

// Message scheduler stage

sha256\_message\_scheduler scheduler (

.clk(gated\_clk), .rst\_n(rst\_n\_sync),

.input\_valid(input\_valid),

.block\_in(input\_block),

.scheduler\_valid(scheduler\_valid),

.w\_out(scheduler\_words),

.scheduler\_ready(scheduler\_ready) );

// Compression stage

sha256\_compressor compressor (

.clk(gated\_clk),

.rst\_n(rst\_n\_sync),

.start(scheduler\_valid),

.message\_block(input\_block),

.initial\_hash(initial\_hash),

.hash\_out(compression\_state),

.done(compression\_valid),

.ready\_for\_scheduler(compression\_ready) );

// Output stage

sha256\_output output\_module (

.clk(gated\_clk), .rst\_n(rst\_n\_sync),

.compression\_valid(compression\_valid),

.state\_in(compression\_state),

.hash\_out(hash\_out), .done(done),

.ready\_for\_next(ready\_for\_next) );

endmodule

**Timing and Control**

**5.1 Critical Timing Sequences:**

* Input Processing: Message segmentation, padding, and buffering.
* Message Scheduling: Word expansion, pipelining, and synchronization.
* Compression Processing: Round execution, state updates, and timing.
* Output Generation: Hash finalization, formatting, and validation.

**5.2 Control Signals:**

* Module-specific ready/valid handshaking.
* Pipeline stall/advance signals.
* Reset and initialization controls.
* Error handling mechanisms.

**6. Implementation Considerations:**

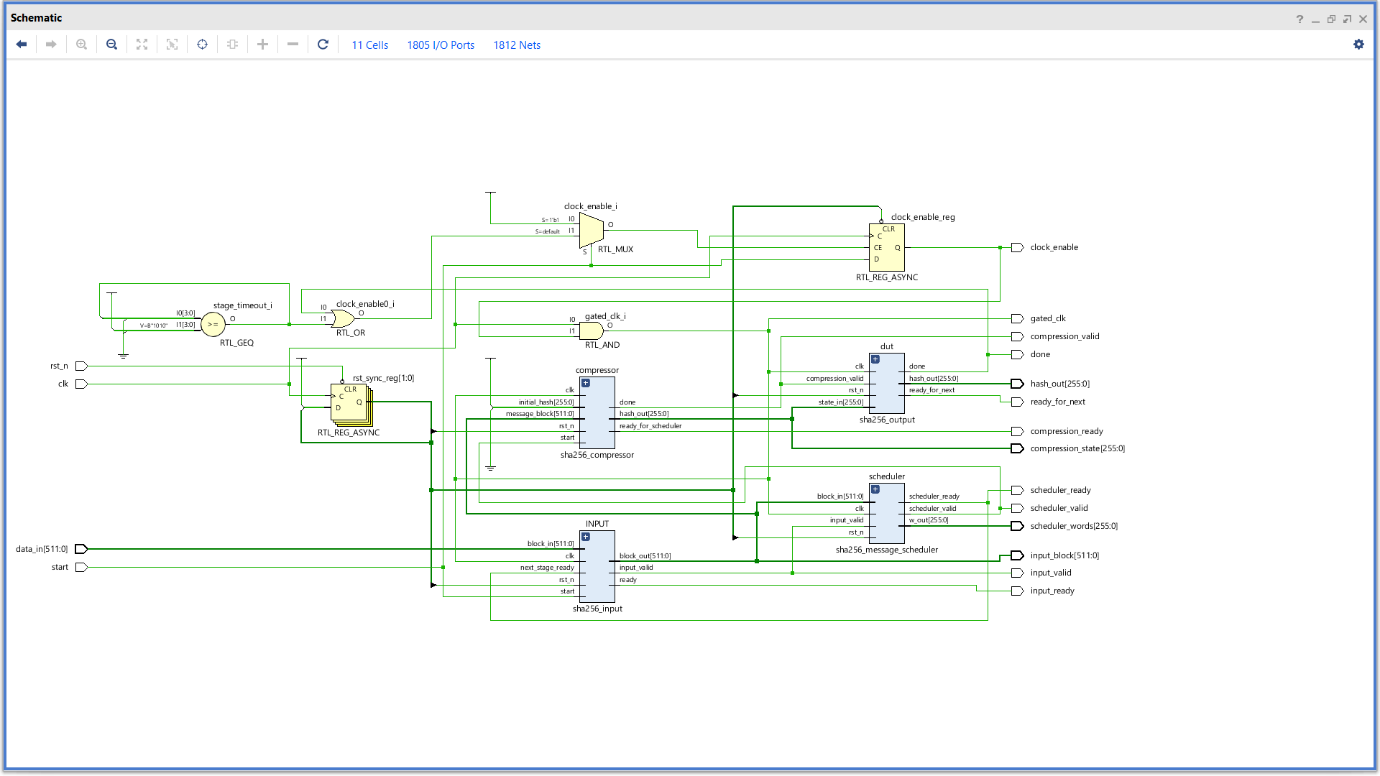
**6.1 Resource Optimization:**

* Pipelining Strategy: Reducing latency while optimizing resource usage.
* Memory Organization: Efficient buffer management.
* Timing Optimization: Handling setup/hold constraints.

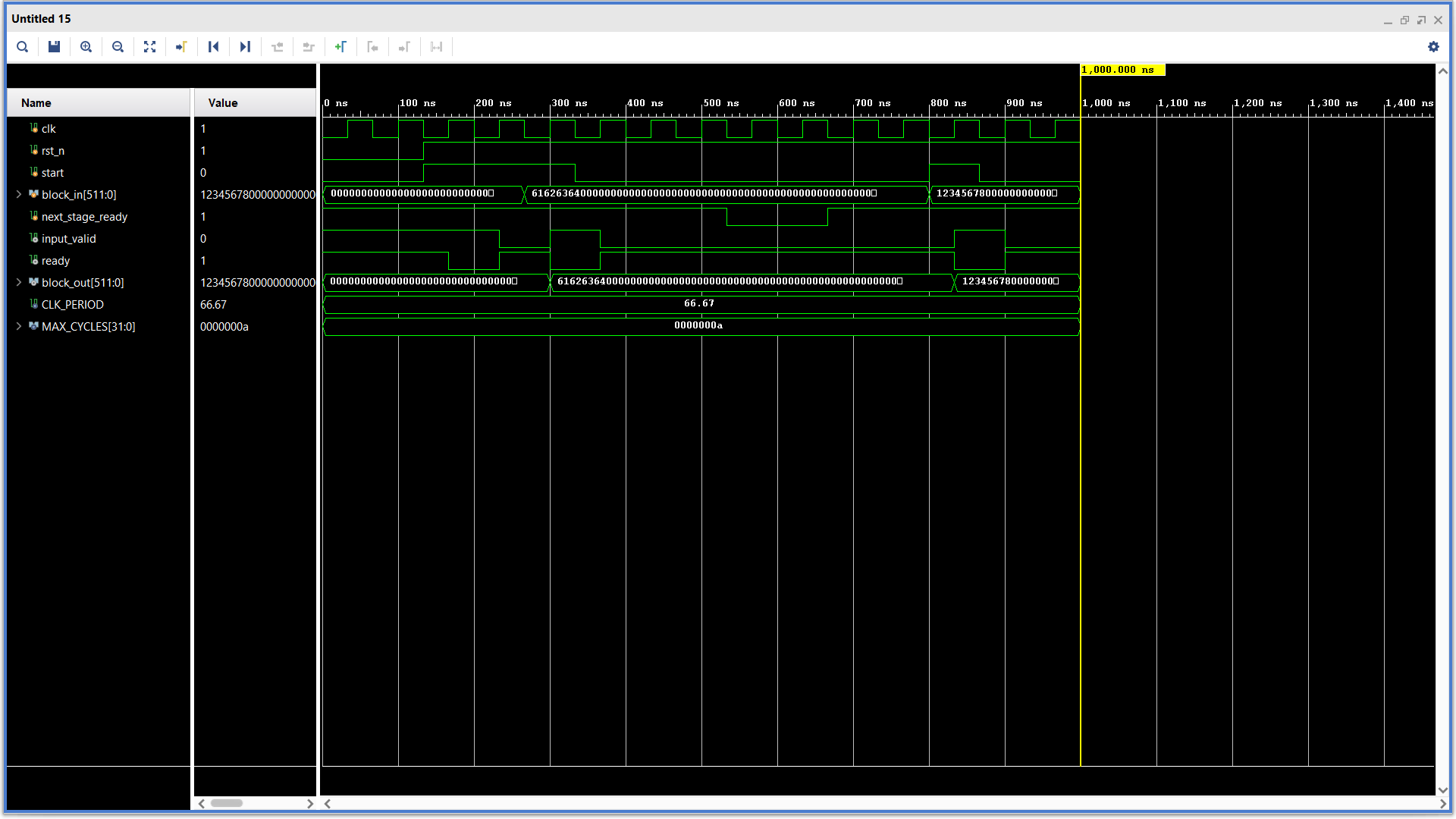
**6.2 Verification Requirements:**

* Functional Verification: Unit testing of modules.
* Performance Testing: Evaluating latency and throughput.
* Compliance Testing: Validating against FIPS 180-4 test vectors.

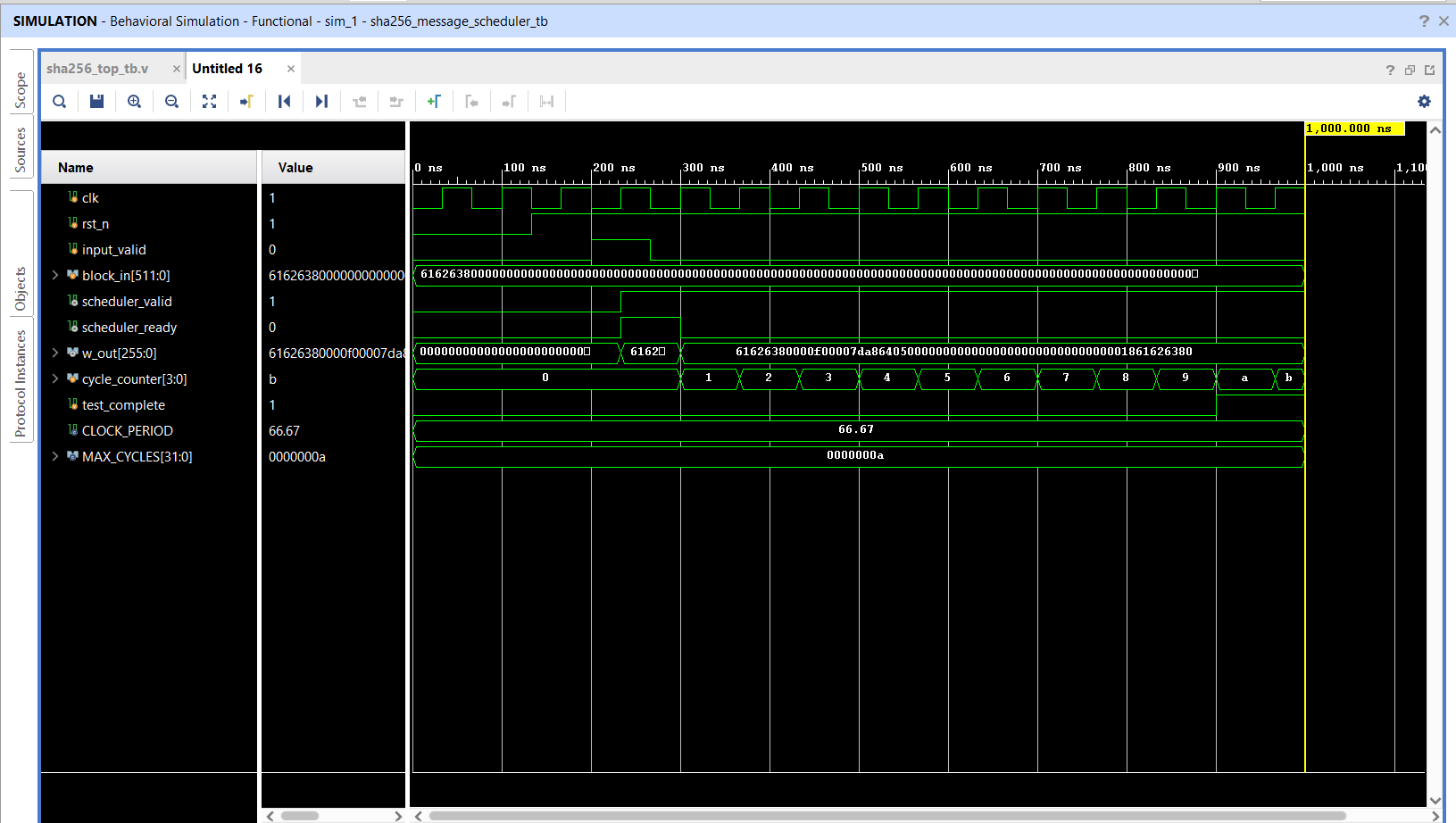
**7.** **Schematic Diagram:**

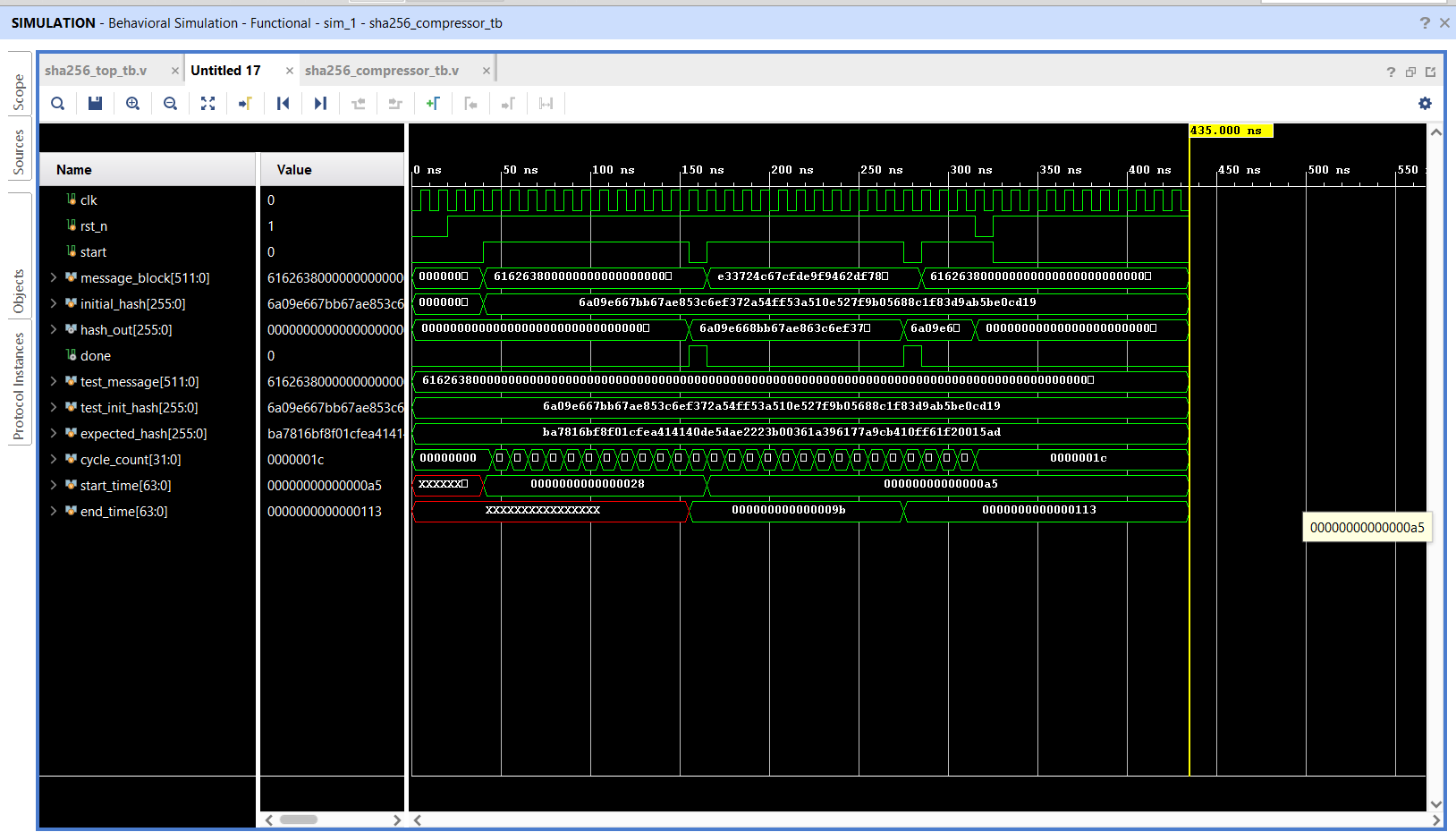
**8. Waveforms:**

**8.1 Input module:**

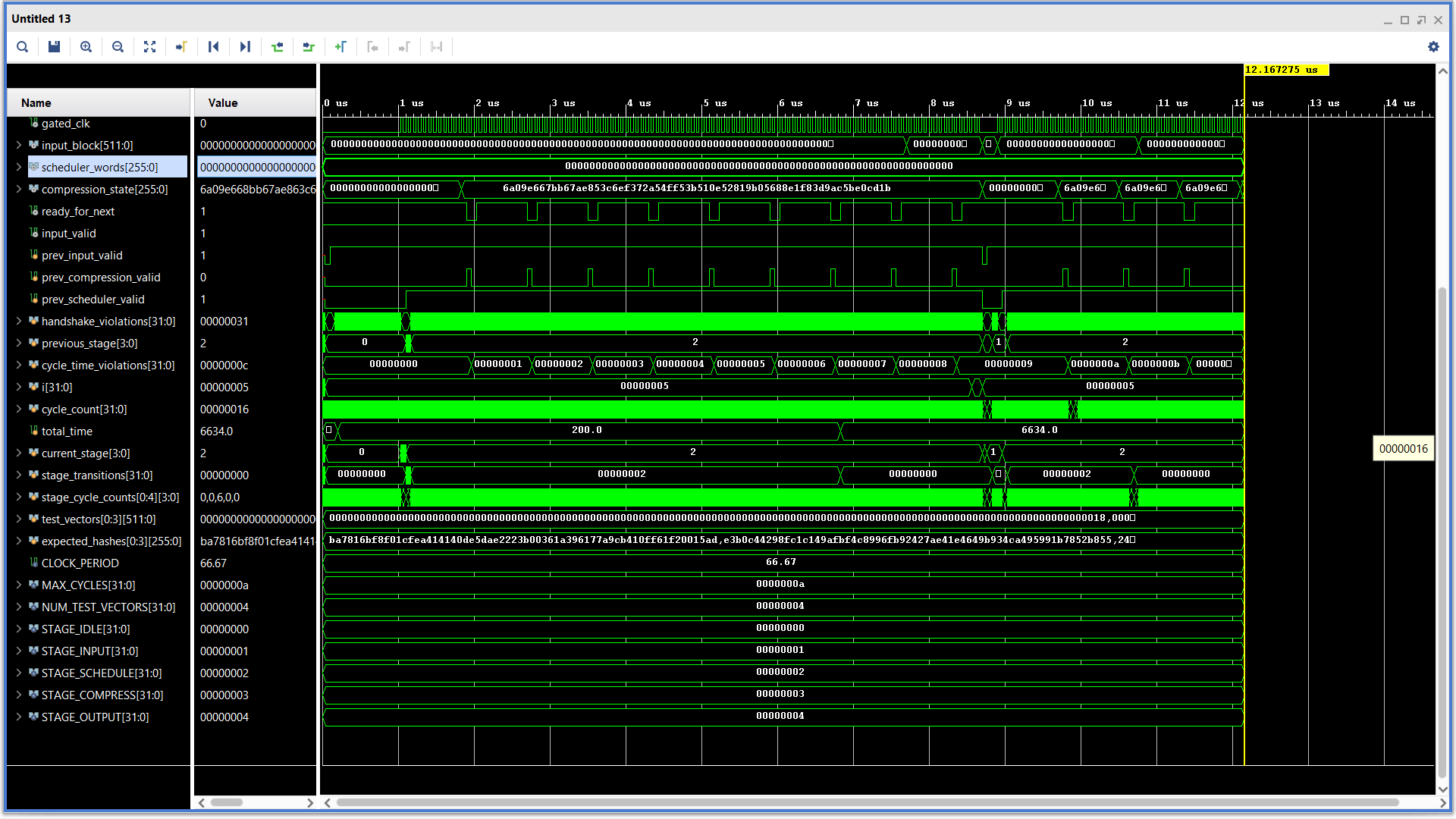
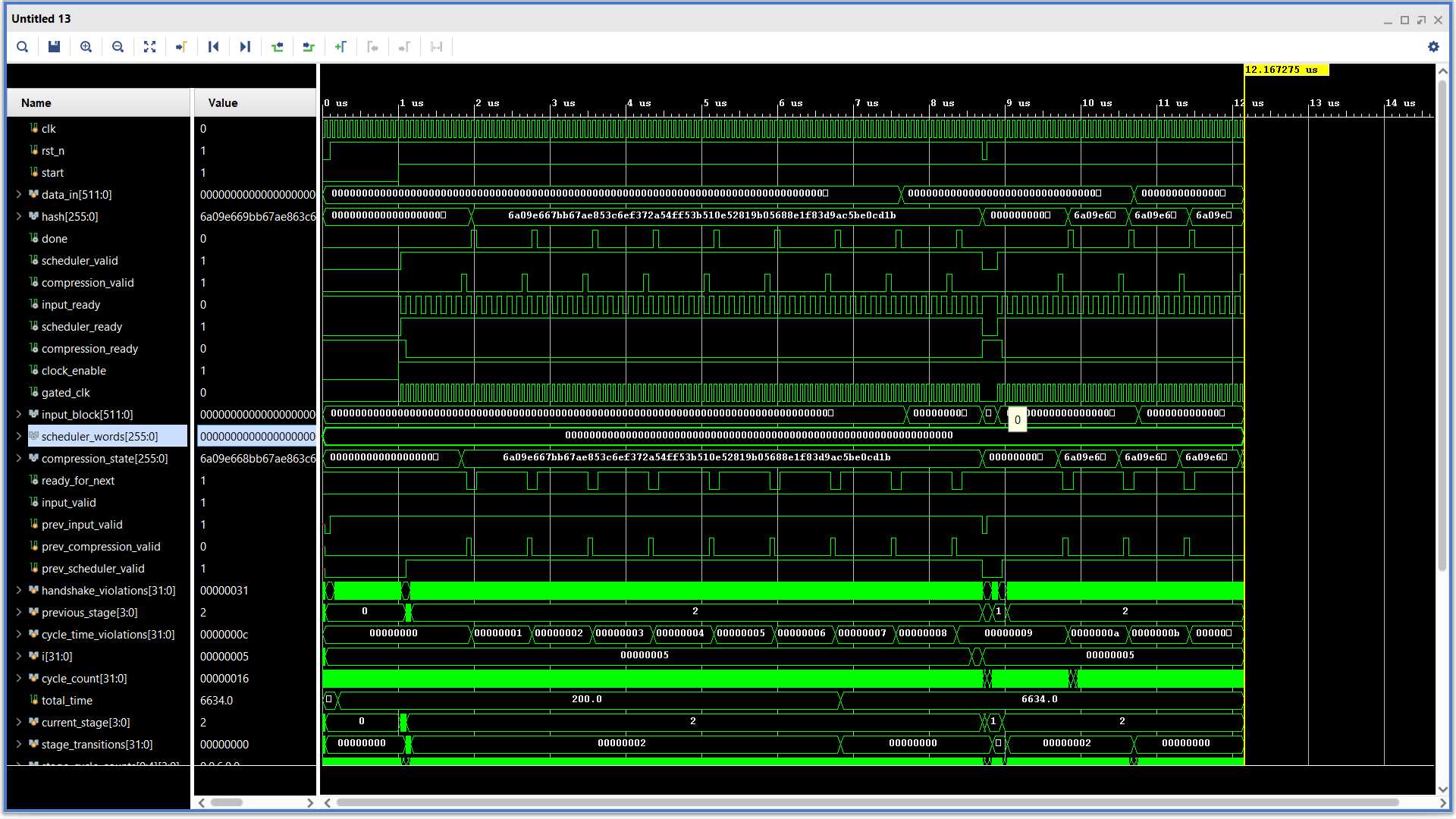
****

**8.2 Message Schedule module:**

** 8.3 Compression module:**

****

**8.4 Output module:**

**** **8.5 Top module:**

**9. Utilization Report:**

Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

---------------------------------------------------------------------------------------------------------------

| Tool Version : Vivado v.2019.1 (win64) Build 2552052 Fri May 24 14:49:42 MDT 2019

| Date : Tue Feb 18 20:28:23 2025

| Host : LAPTOP-0AHIQS8P running 64-bit major release (build 9200)

| Command : report\_utilization -file sha256\_top\_utilization\_synth.rpt -pb sha256\_top\_utilization\_synth.pb

| Design : sha256\_top

| Device : 7a35tftg256-1

| Design State : Synthesized

---------------------------------------------------------------------------------------------------------------

Utilization Design Information

Table of Contents

-----------------

1. Slice Logic

1.1 Summary of Registers by Type

2. Memory

3. DSP

4. IO and GT Specific

5. Clocking

6. Specific Feature

7. Primitives

8. Black Boxes

9. Instantiated Netlists

1. Slice Logic

--------------

+-------------------------+------+-------+-----------+-------+

| Site Type | Used | Fixed | Available | Util% |

+-------------------------+------+-------+-----------+-------+

| Slice LUTs\* | 5644 | 0 | 20800 | 27.13 |

| LUT as Logic | 5644 | 0 | 20800 | 27.13 |

| LUT as Memory | 0 | 0 | 9600 | 0.00 |

| Slice Registers | 3933 | 0 | 41600 | 9.45 |

| Register as Flip Flop | 3933 | 0 | 41600 | 9.45 |

| Register as Latch | 0 | 0 | 41600 | 0.00 |

| F7 Muxes | 0 | 0 | 16300 | 0.00 |

| F8 Muxes | 0 | 0 | 8150 | 0.00 |

+-------------------------+------+-------+-----------+-------+

\* Warning! The Final LUT count, after physical optimizations and full implementation, is typically lower. Run opt\_design after synthesis, if not already completed, for a more realistic count.

1.1 Summary of Registers by Type

--------------------------------

+-------+--------------+-------------+--------------+

| Total | Clock Enable | Synchronous | Asynchronous |

+-------+--------------+-------------+--------------+

| 0 | \_ | - | - |

| 0 | \_ | - | Set |

| 0 | \_ | - | Reset |

| 0 | \_ | Set | - |

| 0 | \_ | Reset | - |

| 0 | Yes | - | - |

| 6 | Yes | - | Set |

| 3790 | Yes | - | Reset |

| 70 | Yes | Set | - |

| 67 | Yes | Reset | - |

+-------+--------------+-------------+--------------+

2. Memory

---------

+----------------+------+-------+-----------+-------+

| Site Type | Used | Fixed | Available | Util% |

+----------------+------+-------+-----------+-------+

| Block RAM Tile | 0 | 0 | 50 | 0.00 |

| RAMB36/FIFO\* | 0 | 0 | 50 | 0.00 |

| RAMB18 | 0 | 0 | 100 | 0.00 |

+----------------+------+-------+-----------+-------+

\* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a Block RAM Tile, that tile can still accommodate a RAMB18E1

3. DSP

------

+-----------+------+-------+-----------+-------+

| Site Type | Used | Fixed | Available | Util% |

+-----------+------+-------+-----------+-------+

| DSPs | 0 | 0 | 90 | 0.00 |

+-----------+------+-------+-----------+-------+

4. IO and GT Specific

---------------------

+-----------------------------+------+-------+-----------+---------+

| Site Type | Used | Fixed | Available | Util% |

+-----------------------------+------+-------+-----------+---------+

| Bonded IOB | 1805 | 0 | 170 | 1061.76 |

| Bonded IPADs | 0 | 0 | 2 | 0.00 |

| PHY\_CONTROL | 0 | 0 | 5 | 0.00 |

| PHASER\_REF | 0 | 0 | 5 | 0.00 |

| OUT\_FIFO | 0 | 0 | 20 | 0.00 |

| IN\_FIFO | 0 | 0 | 20 | 0.00 |

| IDELAYCTRL | 0 | 0 | 5 | 0.00 |

| IBUFDS | 0 | 0 | 163 | 0.00 |

| PHASER\_OUT/PHASER\_OUT\_PHY | 0 | 0 | 20 | 0.00 |

| PHASER\_IN/PHASER\_IN\_PHY | 0 | 0 | 20 | 0.00 |

| IDELAYE2/IDELAYE2\_FINEDELAY | 0 | 0 | 250 | 0.00 |

| ILOGIC | 0 | 0 | 170 | 0.00 |

| OLOGIC | 0 | 0 | 170 | 0.00 |

+-----------------------------+------+-------+-----------+---------+

5. Clocking

-----------

+------------+------+-------+-----------+-------+

| Site Type | Used | Fixed | Available | Util% |

+------------+------+-------+-----------+-------+

| BUFGCTRL | 2 | 0 | 32 | 6.25 |

| BUFIO | 0 | 0 | 20 | 0.00 |

| MMCME2\_ADV | 0 | 0 | 5 | 0.00 |

| PLLE2\_ADV | 0 | 0 | 5 | 0.00 |

| BUFMRCE | 0 | 0 | 10 | 0.00 |

| BUFHCE | 0 | 0 | 72 | 0.00 |

| BUFR | 0 | 0 | 20 | 0.00 |

+------------+------+-------+-----------+-------+

6. Specific Feature

-------------------

+-------------+------+-------+-----------+-------+

| Site Type | Used | Fixed | Available | Util% |

+-------------+------+-------+-----------+-------+

| BSCANE2 | 0 | 0 | 4 | 0.00 |

| CAPTUREE2 | 0 | 0 | 1 | 0.00 |

| DNA\_PORT | 0 | 0 | 1 | 0.00 |

| EFUSE\_USR | 0 | 0 | 1 | 0.00 |

| FRAME\_ECCE2 | 0 | 0 | 1 | 0.00 |

| ICAPE2 | 0 | 0 | 2 | 0.00 |

| PCIE\_2\_1 | 0 | 0 | 1 | 0.00 |

| STARTUPE2 | 0 | 0 | 1 | 0.00 |

| XADC | 0 | 0 | 1 | 0.00 |

+-------------+------+-------+-----------+-------+

7. Primitives

-------------

+----------+------+---------------------+

| Ref Name | Used | Functional Category |

+----------+------+---------------------+

| FDCE | 3790 | Flop & Latch |

| LUT4 | 2012 | LUT |

| LUT3 | 1849 | LUT |

| LUT5 | 1620 | LUT |

| LUT2 | 1479 | LUT |

| OBUF | 1290 | IO |

| CARRY4 | 796 | CarryLogic |

| IBUF | 515 | IO |

| LUT6 | 356 | LUT |

| LUT1 | 113 | LUT |

| FDSE | 70 | Flop & Latch |

| FDRE | 67 | Flop & Latch |

| FDPE | 6 | Flop & Latch |

| BUFG | 2 | Clock |

+----------+------+---------------------+

8. Black Boxes

--------------

+----------+------+

| Ref Name | Used |

+----------+------+

9. Instantiated Netlists

------------------------

+----------+------+

| Ref Name | Used |

+----------+------+

**10. Conclusion:**

This document provides a comprehensive overview of the SHA-256 hardware implementation, detailing each module's functionality, structure, and interconnections. The design prioritizes modularity, efficiency, and resource optimization while ensuring compliance with cryptographic standards. Future enhancements may include optimizations for lower power consumption, improved timing performance, and FPGA-based real-time implementations for cryptographic applications. By following a structured and hierarchical approach, this implementation facilitates secure and efficient cryptographic processing for various applications, including blockchain, secure communications, and data integrity verification.