4. SHA-256 Hashing Accelerator (Group Project)

4.1 Design

```
module sha256_single(
    input wire clk,
    input wire rst,
    input wire start,
    input wire [511:0] message_block,
    output reg ready,
    output reg hash_valid,
    output wire [255:0] hash_out
);

    // State definitions
    parameter IDLE    = 2'b00;
    parameter PREPARE = 2'b01;
    parameter COMPUTE = 2'b10;
    parameter DONE    = 2'b11;

    // State and control registers
    reg [1:0] state;
    reg [5:0] round;  // 6 bits for rounds 0-63

    // SHA-256 working variables
    reg [31:0] a, b, c, d, e, f, g, h;
    reg [31:0] H0, H1, H2, H3, H4, H5, H6, H7;

    // W memory arrays - use wires for combinational generation
    wire [31:0] W [0:63];
    wire [31:0] w_data;

    // SHA-256 constants
    reg [31:0] K [0:63];

    // Temporary variables for computation
    reg [31:0] T1, T2;
```

```
    // Temporary variables for computation
    reg [31:0] T1, T2;
    reg [31:0] S0, S1;
    reg [31:0] ch, maj;

    // Pre-computed values for next cycle
    reg [31:0] a_next, b_next, c_next, d_next, e_next, f_next, g_next, h_next;

    // Initialize SHA-256 constants
    initial begin
        K[0] = 32'h428a2f98; K[1] = 32'h71374491; K[2] = 32'hb5c0fbcf; K[3] = 32'he9b5dba5;
        K[4] = 32'h3956c25b; K[5] = 32'h59f111f1; K[6] = 32'h923f82a4; K[7] = 32'hab1c5ed5;
        K[8] = 32'hd807aa98; K[9] = 32'h12835b01; K[10] = 32'h243185be; K[11] = 32'h550c7dc3;
        K[12] = 32'h72be5d74; K[13] = 32'h80deb1fe; K[14] = 32'h9bdc06a7; K[15] = 32'hc19bf174;
        K[16] = 32'he49b69c1; K[17] = 32'hefbe4786; K[18] = 32'h0fc19dc6; K[19] = 32'h240ca1cc;
        K[20] = 32'h2de92c6f; K[21] = 32'h4a7484aa; K[22] = 32'h5cb0a9dc; K[23] = 32'h76f988da;
        K[24] = 32'h983e5152; K[25] = 32'ha831c66d; K[26] = 32'hb00327c8; K[27] = 32'hbf597fc7;
        K[28] = 32'hc6e00bf3; K[29] = 32'hd5a79147; K[30] = 32'h06ca6351; K[31] = 32'h14292967;
        K[32] = 32'h27b70a85; K[33] = 32'h2e1b2138; K[34] = 32'h4d2c6dfc; K[35] = 32'h53380d13;
        K[36] = 32'h650a7354; K[37] = 32'h766a0abb; K[38] = 32'h81c2c92e; K[39] = 32'h92722c85;
        K[40] = 32'ha2bfe8a1; K[41] = 32'ha81a664b; K[42] = 32'hc24b8b70; K[43] = 32'hc76c51a3;
        K[44] = 32'hd192e819; K[45] = 32'hd6990624; K[46] = 32'hf40e3585; K[47] = 32'h106aa070;
        K[48] = 32'h19a4c116; K[49] = 32'h1e376c08; K[50] = 32'h2748774c; K[51] = 32'h34b0bcb5;
        K[52] = 32'h391c0cb3; K[53] = 32'h4ed8aa4a; K[54] = 32'h5b9cca4f; K[55] = 32'h682e6ff3;
        K[56] = 32'h748f82ee; K[57] = 32'h78a5636f; K[58] = 32'h84c87814; K[59] = 32'h8cc70208;
        K[60] = 32'h90befffa; K[61] = 32'ha4506ceb; K[62] = 32'hbef9a3f7; K[63] = 32'hc67178f2;
    end

    // SHA-256 functions
    function [31:0] rotr;
        input [31:0] x;
        input [4:0] n;
        rotr = (x >> n) | (x << (32 - n));
```

```
    input [4:0] n;
    rotr = (x >> n) | (x << (32 - n));
endfunction

function [31:0] ch_func;
    input [31:0] x, y, z;
    ch_func = (x & y) ^ (~x & z);
endfunction

function [31:0] maj_func;
    input [31:0] x, y, z;
    maj_func = (x & y) ^ (x & z) ^ (y & z);
endfunction

function [31:0] SIGMA0;
    input [31:0] x;
    SIGMA0 = rotr(x, 2) ^ rotr(x, 13) ^ rotr(x, 22);
endfunction

function [31:0] SIGMA1;
    input [31:0] x;
    SIGMA1 = rotr(x, 6) ^ rotr(x, 11) ^ rotr(x, 25);
endfunction

// Sigma functions for W expansion
function [31:0] sigma0;
    input [31:0] x;
    sigma0 = rotr(x, 7) ^ rotr(x, 18) ^ (x >> 3);
endfunction

function [31:0] sigma1;
    input [31:0] x;
    sigma1 = rotr(x, 17) ^ rotr(x, 19) ^ (x >> 10);
```

```
endfunction

// Combinational W array generation
// First 16 words from message block
assign W[0] = message_block[511:480];   assign W[1] = message_block[479:448];
assign W[2] = message_block[447:416];   assign W[3] = message_block[415:384];
assign W[4] = message_block[383:352];   assign W[5] = message_block[351:320];
assign W[6] = message_block[319:288];   assign W[7] = message_block[287:256];
assign W[8] = message_block[255:224];   assign W[9] = message_block[223:192];
assign W[10] = message_block[191:160];  assign W[11] = message_block[159:128];
assign W[12] = message_block[127:96];   assign W[13] = message_block[95:64];
assign W[14] = message_block[63:32];    assign W[15] = message_block[31:0];

// Generate W[16] to W[63] combinationally
genvar i;
generate
    for (i = 16; i < 64; i = i + 1) begin : gen_w
        assign W[i] = sigma1(W[i-2]) + W[i-7] + sigma0(W[i-15]) + W[i-16];
    end
endgenerate

// W data selection
assign w_data = W[round];

// Combinational logic for round computation
always @(*) begin
    // Compute SHA-256 round function
    ch = ch_func(e, f, g);
    maj = maj_func(a, b, c);
    S0 = SIGMA0(a);
    S1 = SIGMA1(e);
```

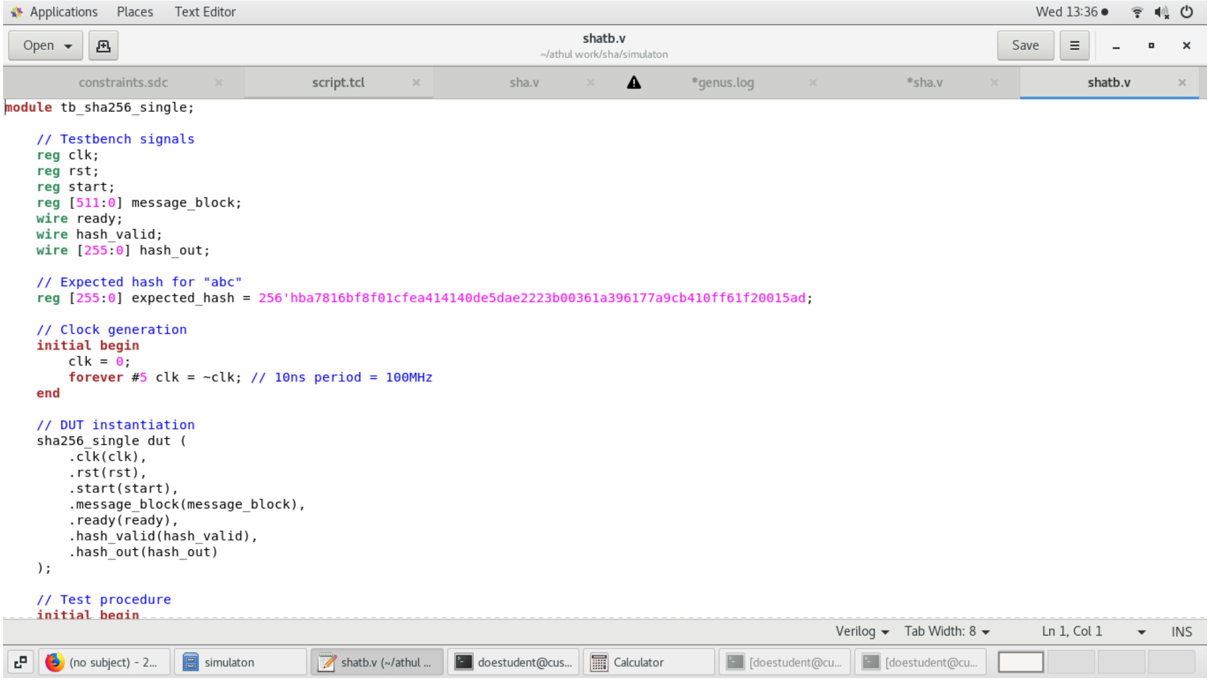
```
    // T1 and T2 calculations
    T1 = h + S1 + ch + K[round] + w_data;
    T2 = S0 + maj;

    // Compute next working variables
    h_next = g;
    g_next = f;
    f_next = e;
    e_next = d + T1;
    d_next = c;
    c_next = b;
    b_next = a;
    a_next = T1 + T2;
end

// State machine
always @(posedge clk or posedge rst) begin
    if (rst) begin
        state <= IDLE;
        round <= 6'd0;
        ready <= 1'b1;
        hash_valid <= 1'b0;

        // Initialize SHA-256 hash values
        H0 <= 32'h6a09e667;
        H1 <= 32'hbb67ae85;
        H2 <= 32'h3c6ef372;
        H3 <= 32'ha54ff53a;
        H4 <= 32'h510e527f;
        H5 <= 32'h9b05688c;
        H6 <= 32'h1f83d9ab;
        H7 <= 32'h5be0cd19;
```

```
        // Clear working variables
        a <= 32'h0; b <= 32'h0; c <= 32'h0; d <= 32'h0;
        e <= 32'h0; f <= 32'h0; g <= 32'h0; h <= 32'h0;
    end
    else begin
        case (state)
            IDLE: begin
                ready <= 1'b1;
                hash_valid <= 1'b0;

                if (start) begin
                    state <= PREPARE;
                    ready <= 1'b0;
                    $display("Time %0t: CORE: Starting hash computation", $time);
                end
            end

            PREPARE: begin
                // Initialize working variables with current hash values
                a <= H0; b <= H1; c <= H2; d <= H3;
                e <= H4; f <= H5; g <= H6; h <= H7;

                round <= 6'd0;
                state <= COMPUTE;

                $display("Time %0t: CORE: PREPARE complete, W array ready", $time);
                $display("Time %0t: W_MEM: W[0]=%h, W[1]=%h, W[14]=%h, W[15]=%h",
                         $time, W[0], W[1], W[14], W[15]);
                $display("Time %0t: W_MEM: W[16]=%h, W[17]=%h, W[18]=%h, W[19]=%h",
                         $time, W[16], W[17], W[18], W[19]);
            end
```

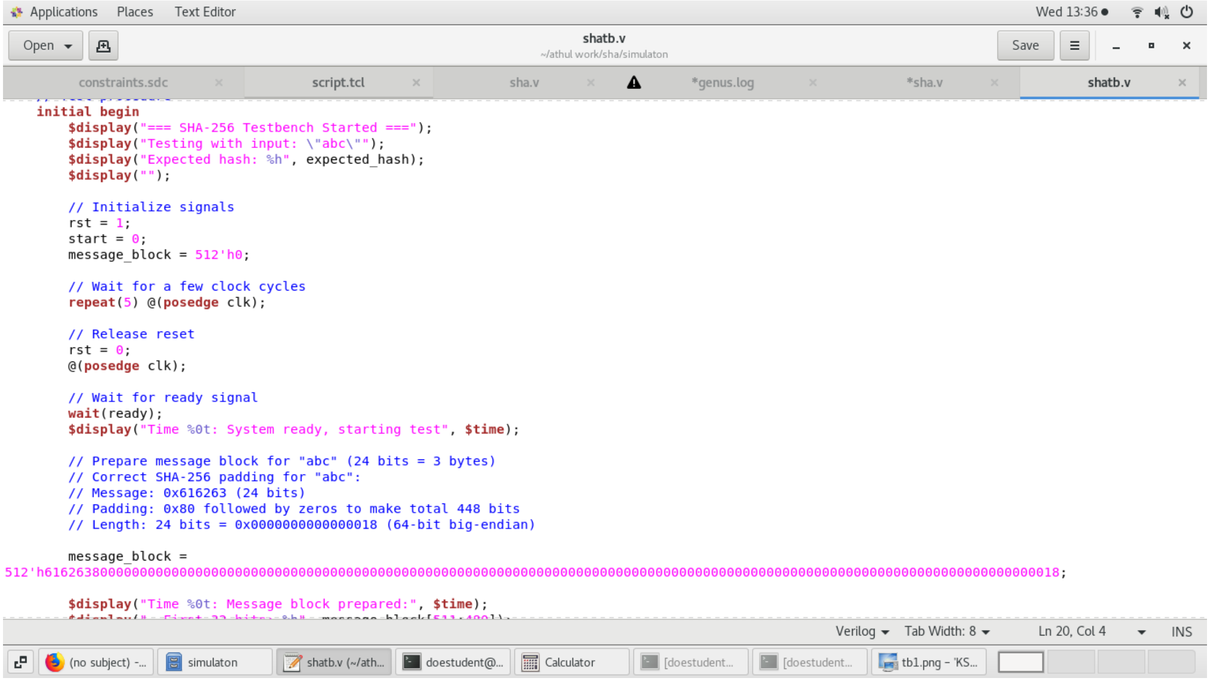
```
            COMPUTE: begin
                // Update working variables with pre-computed values
                a <= a_next;
                b <= b_next;
                c <= c_next;
                d <= d_next;
                e <= e_next;
                f <= f_next;
                g <= g_next;
                h <= h_next;

                if (round < 4 || round > 60) begin
                    $display("Time %0t: CORE: Round %0d - W=%h, a=%h, e=%h", $time, round, w_data, a_next, e_next);
                end

                // Check if all rounds completed
                if (round == 6'd63) begin
                    state <= DONE;
                    $display("Time %0t: CORE: All 64 rounds completed", $time);
                end
                else begin
                    round <= round + 1'b1;
                end
            end

            DONE: begin
                // Add compressed chunk to current hash value
                H0 <= H0 + a;
                H1 <= H1 + b;
                H2 <= H2 + c;
                H3 <= H3 + d;
                H4 <= H4 + e;
```

```
            DONE: begin
                // Add compressed chunk to current hash value
                H0 <= H0 + a;
                H1 <= H1 + b;
                H2 <= H2 + c;
                H3 <= H3 + d;
                H4 <= H4 + e;
                H5 <= H5 + f;
                H6 <= H6 + g;
                H7 <= H7 + h;

                hash_valid <= 1'b1;
                ready <= 1'b1;
                round <= 6'd0;
                state <= IDLE;

                $display("Time %0t: CORE: Hash computation complete", $time);
                $display("Time %0t: CORE: Hash = %h%h%h%h%h%h%h%h",
                        $time, H0+a, H1+b, H2+c, H3+d, H4+e, H5+f, H6+g, H7+h);
            end

            default: begin
                state <= IDLE;
                round <= 6'd0;
                ready <= 1'b1;
                hash_valid <= 1'b0;
            end
        endcase
    end
end
// Output hash
assign hash_out = {H0, H1, H2, H3, H4, H5, H6, H7};
endmodule
```

4.2 Testbench



```verilog
module tb_sha256_single;

    // Testbench signals
    reg clk;
    reg rst;
    reg start;
    reg [511:0] message_block;
    wire ready;
    wire hash_valid;
    wire [255:0] hash_out;

    // Expected hash for "abc"
    reg [255:0] expected_hash = 256'hba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad;

    // Clock generation
    initial begin
        clk = 0;
        forever #5 clk = ~clk; // 10ns period = 100MHz
    end

    // DUT instantiation
    sha256_single dut (
        .clk(clk),
        .rst(rst),
        .start(start),
        .message_block(message_block),
        .ready(ready),
        .hash_valid(hash_valid),
        .hash_out(hash_out)
    );

    // Test procedure
    initial begin
```



```verilog
    initial begin
        $display("=== SHA-256 Testbench Started ===");
        $display("Testing with input: \"abc\"");
        $display("Expected hash: %h", expected_hash);
        $display("");

        // Initialize signals
        rst = 1;
        start = 0;
        message_block = 512'h0;

        // Wait for a few clock cycles
        repeat(5) @(posedge clk);

        // Release reset
        rst = 0;
        @(posedge clk);

        // Wait for ready signal
        wait(ready);
        $display("Time %0t: System ready, starting test", $time);

        // Prepare message block for "abc" (24 bits = 3 bytes)
        // Correct SHA-256 padding for "abc":
        // Message: 0x616263 (24 bits)
        // Padding: 0x80 followed by zeros to make total 448 bits
        // Length: 24 bits = 0x0000000000000018 (64-bit big-endian)

        message_block =
512'h61626380000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000018;

        $display("Time %0t: Message block prepared:", $time);
```

Applications Places Text Editor Wed 13:36

Open shatb.v ~/athul work/sha/simulaton Save

constraints.sdc script.tcl sha.v *genus.log *sha.v shatb.v

```
        $display("Time %0t: Message block prepared:", $time);
        $display("  First 32 bits: %h", message_block[511:480]);
        $display("  Last 64 bits (length): %h", message_block[63:0]);
        $display("  Message length field: %d bits", message_block[63:0]);

        // Start hashing
        @(posedge clk);
        start = 1;
        @(posedge clk);
        start = 0;

        $display("Time %0t: Hash operation started", $time);

        // Wait for completion
        wait(hash_valid);

        $display("");
        $display("=== HASH COMPUTATION COMPLETE ===");
        $display("Time %0t: Hash Valid", $time);
        $display("Computed hash: %h", hash_out);
        $display("Expected hash: %h", expected_hash);

        // Compare results
        if (hash_out == expected_hash) begin
            $display("? SUCCESS: Hash matches expected value!");
        end else begin
            $display("? FAILURE: Hash does not match expected value!");
            $display("Difference:    %h", hash_out ^ expected_hash);
            $display("");
            $display("Hash breakdown (32-bit words):");
            $display("H0: %h (expected: %h)", hash_out[255:224], expected_hash[255:224]);
            $display("H1: %h (expected: %h)", hash_out[223:192], expected_hash[223:192]);
```

Verilog Tab Width: 8 Ln 20, Col 4 INS

(no subject) -... simulaton shatb.v (~/ath... doestudent@... Calculator [doestudent... [doestudent... tb1.png - 'KS...

Applications Places Text Editor Wed 13:37

Open shatb.v ~/athul work/sha/simulaton Save

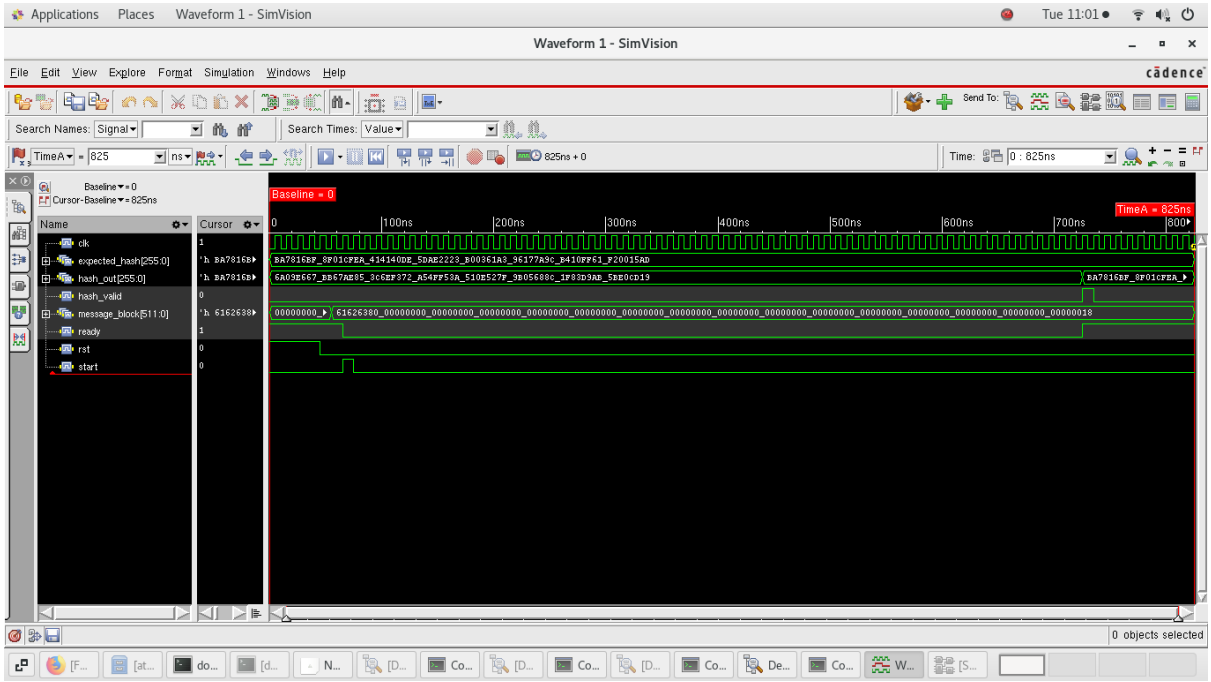
constraints.sdc script.tcl sha.v *genus.log *sha.v shatb.v

```
            $display("H7: %h (expected: %h)", hash_out[31:0],    expected_hash[31:0]);
        end

        // Wait a bit more to see final state
        repeat(10) @(posedge clk);

        if (ready) begin
            $display("Time %0t: System ready for next operation", $time);
        end else begin
            $display("Time %0t: System not ready after completion", $time);
        end

        $display("");
        $display("=== Test Complete ===");

        // End simulation
        $finish;
    end

    // Timeout protection
    initial begin
        #1000000; // 1ms timeout
        $display("ERROR: Simulation timeout!");
        $finish;
    end

    // Monitor key signals
    always @(posedge clk) begin
        $display("Time %0t: ready=%b, hash_valid=%b, start=%b, state=%b",
                 $time, ready, hash_valid, start, dut.state);
    end

endmodule
```

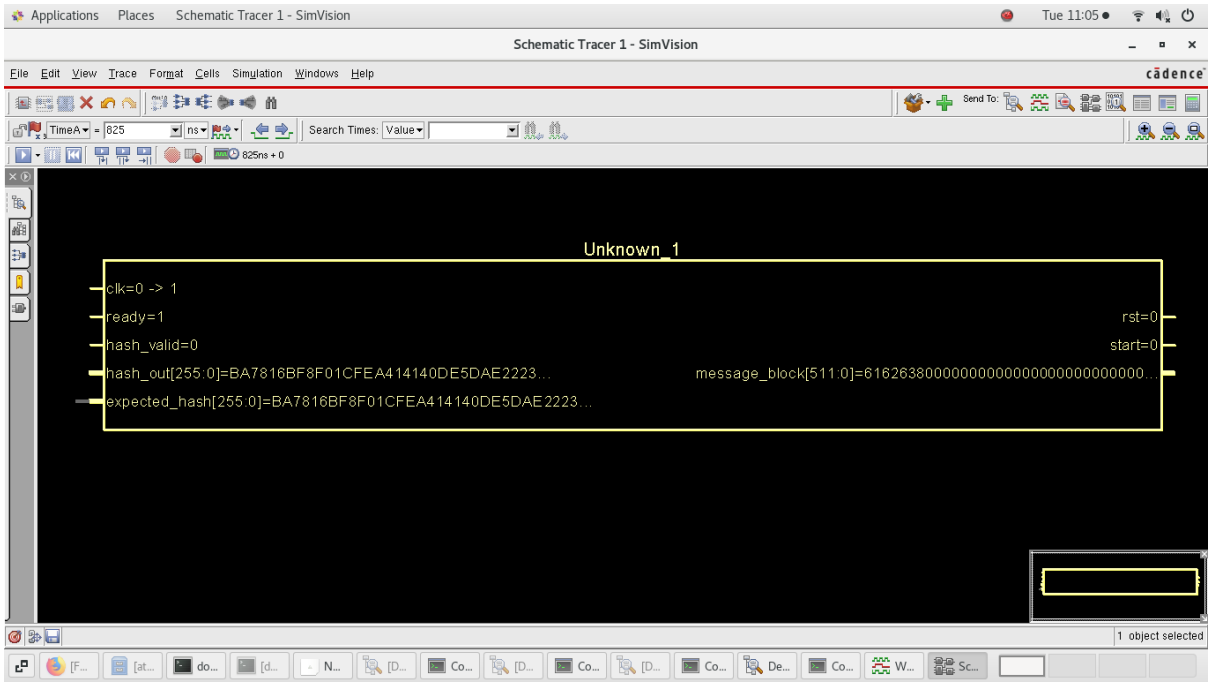
Verilog Tab Width: 8 Ln 20, Col 4 INS

(no subject) -... simulaton shatb.v (~/ath... doestudent@... Calculator [doestudent... [doestudent... tb1.png - 'KS...

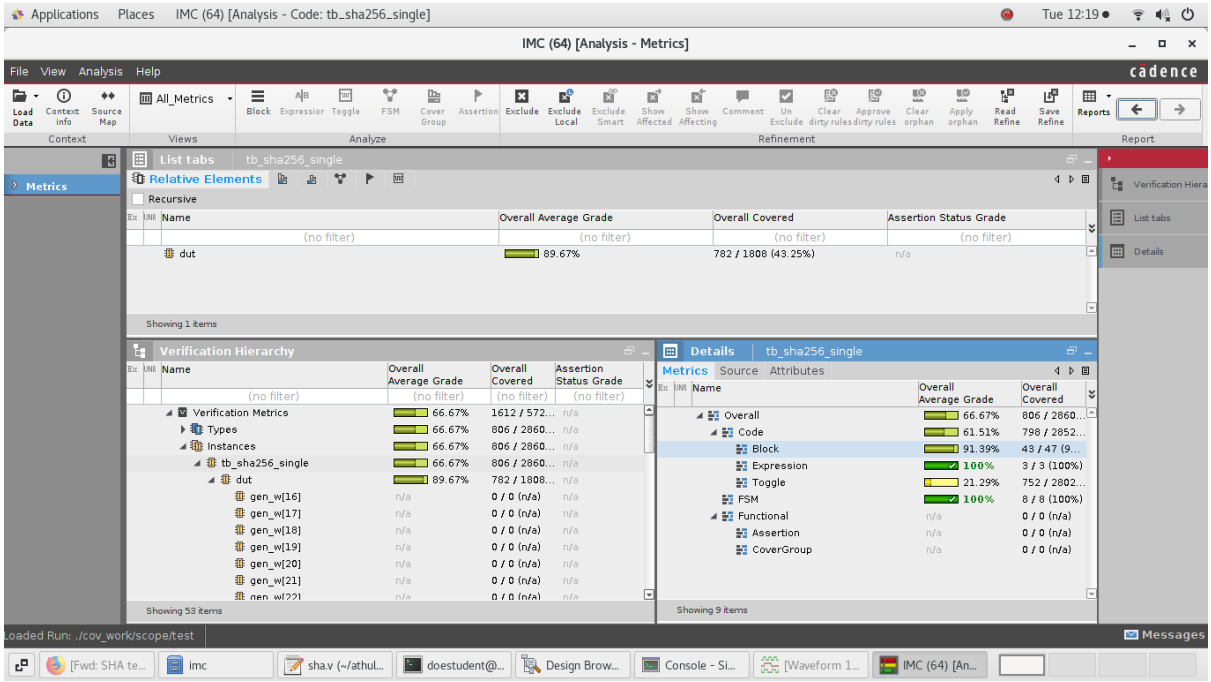
4.3 Waveform



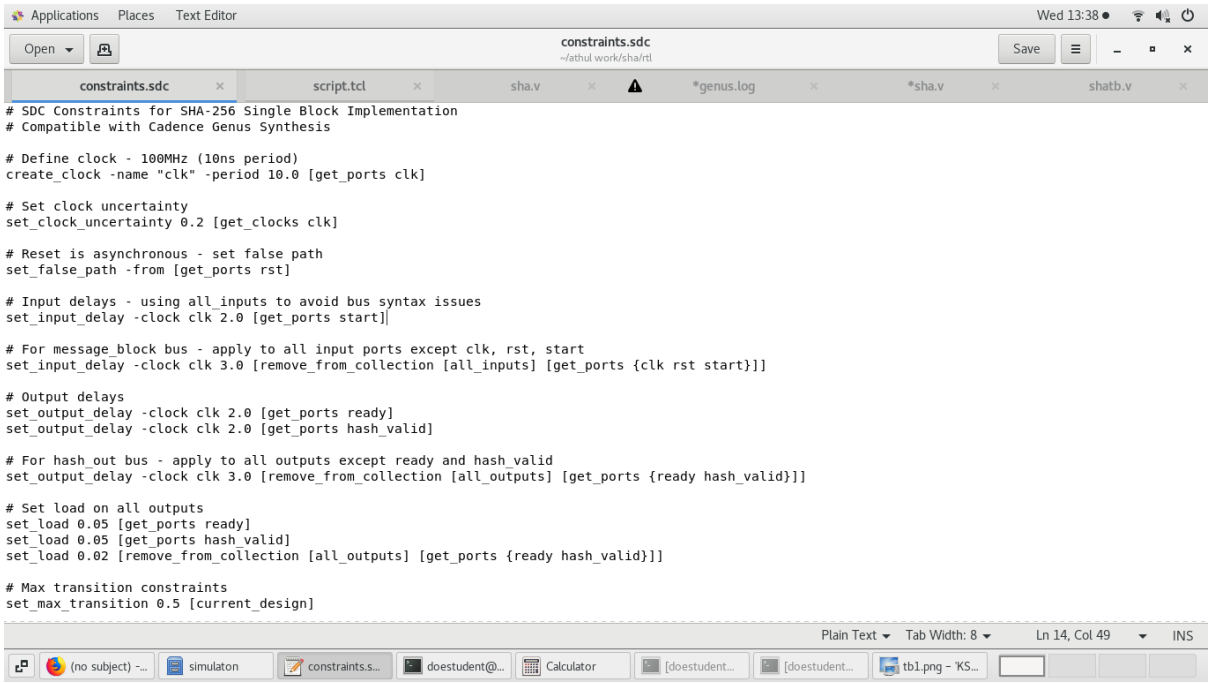
4.4 Schematic Diagram



4.5 Coverage Analysis



4.6 Constraints File



4.7 Script File

```
read_libs /home/installs/FOUNDRY/digital/90nm/dig/lib/slow.lib
read_hdl sha.v
elaborate
read_sdc constraints.sdc
syn_generic
syn_map
syn_opt
gui_show

write_hdl > sha_netlist.v
write_sdc > output_constraints.sdc
report_area > sha_area.rpt
report_power > sha_power.rpt
report_timing > sha_timing.rpt
report_gates > sha_gates.rpt
gui_show
```

4.8 Reports

```
##>Summary table of configs (Best config is CDN_DP_region_1_0_c5)
##>+-------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------+
##>                    0                1                2                3                4                5                6

##>+-------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------+
##>           Area     500028556385     487364100985     364644765240     487364100985     370877813470     370877813470     484529856900

##>            WNS        -64291.80        -64284.70        -64284.70        -64284.70        -64284.70        -64284.70        -64284.70

##>            TNS       2355119203       2330944259       2320361366       2330944259       2319933365       2319933365       2331069702

##>    Num Rewrite                0               12               96               12              193              193                2

##>     Num Factor                0                1                0                1                1                1                0

##>      Num Share                0                1                1                1                1                1                3

##>   Num CmultCse                0                0                0                0                0                0                0

##>   Num Downsize                0                0                0                0                0                0                0

##>  Num Speculate                0                0                0                0                0                0                0

##>     Runtime(s)                0               50               23               48               44               44               16
```

```
      "all_inputs"                - successful      1 , failed      0 (runtime  0.00)
      "all_outputs"               - successful      1 , failed      0 (runtime  0.00)
      "create_clock"              - successful      1 , failed      0 (runtime  0.00)
      "get_clocks"                - successful      3 , failed      0 (runtime  0.00)
      "get_ports"                 - successful      1 , failed      0 (runtime  0.00)
      "set_clock_transition"      - successful      2 , failed      0 (runtime  0.00)
      "set_clock_uncertainty"     - successful      1 , failed      0 (runtime  0.00)
      "set_input_delay"           - successful      1 , failed      0 (runtime  0.00)
      "set_output_delay"          - successful      1 , failed      0 (runtime  0.00)
    read_sdc completed in 00:00:00 (hh:mm:ss)
```

5. Conclusion

The SHA-256 hardware accelerator design was successfully elaborated using the Cadence synthesis environment. However, due to a system-level hang during the super-threading initialization phase, the synthesis tool was unable to proceed beyond elaboration. As a result, critical post-synthesis reports such as Power, Area, Timing, and Gate count could

not be generated. This issue is suspected to be caused by either resource exhaustion, license constraints, or tool misconfiguration. Despite this, the elaboration stage confirms the structural integrity of the design, validating its RTL-level correctness.