Sequential Logic Design

1.

```
module ALSU
#(
    parameter INPUT_PRIORITY = "A",
                // Determines priority in conflicts ("A" or "B")
    parameter FULL_ADDER = "ON"
                // Enables carry-in for addition ("ON" or "OFF")
)(
    input wire clk,
    input wire rst,
    input wire [2:0] A, B,
    input wire [2:0] opcode,
    input wire cin,
    input wire serial_in,
    input wire red_op_A, red_op_B,
    input wire bypass_A, bypass_B,
    input wire direction,
    output reg [5:0] out,
    output reg [15:0] leds
);

always @(posedge clk or posedge rst) begin
    if (rst) begin
        out <= 6'b0;
        leds <= 16'b0;
    end else begin
        // Handle bypassing
        if (bypass_A || bypass_B) begin
            case ({bypass_A, bypass_B})
                2'b01: out <= {3'b000, B}; // Extend B to 6 bits
                2'b10: out <= {3'b000, A}; // Extend A to 6 bits
                2'b11: out <= (INPUT_PRIORITY == "A") ? {3'b000, A} : {3'b000, B};
            endcase
            leds <= 16'b0;
        end else begin
            case (opcode)
                3'b000: begin
                    case ({red_op_A, red_op_B})
                        2'b00: out <= A & B;  // AND
                        2'b01: out <= &B;     // Applying reduction operation on B
                        2'b10: out <= &A;     // Applying reduction operation on A
                        2'b11: out <= (INPUT_PRIORITY == "A") ? &A : &B;
                    endcase
                    leds <= 16'b0;
                end
```

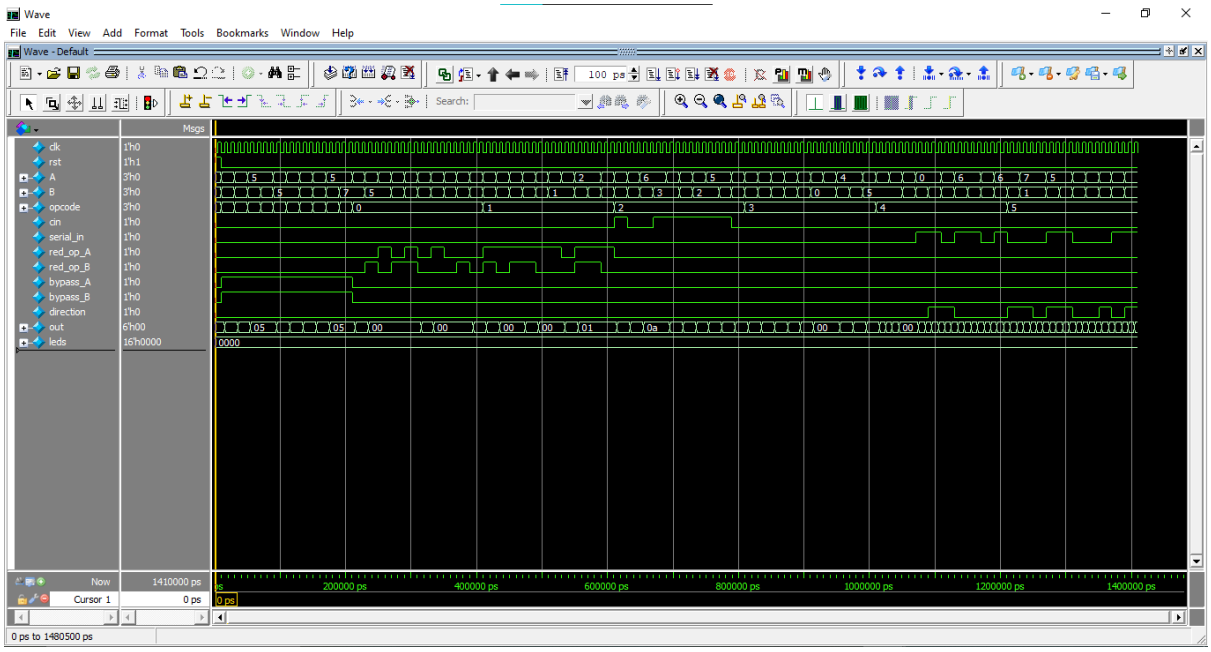
```
                3'b001: begin
                    case ({red_op_A, red_op_B})
                        2'b00: out <= A ^ B;  // XOR
                        2'b01: out <= ^B;     // Applying reduction operation on B
                        2'b10: out <= ^A;     // Applying reduction operation on A
                        2'b11: out <= (INPUT_PRIORITY == "A") ? ^A : ^B;
                    endcase
                    leds <= 16'b0;
                end
                3'b010: begin
                    out <= FULL_ADDER == "ON" ? A + B + cin : A + B; // Addition
                    leds <= 16'b0;
                end
                3'b011: begin
                    out <= A * B;  // Multiplication
                    leds <= 16'b0;
                end
                3'b100: begin
                    out <= direction ? {out[4:0], serial_in}
                                     : {serial_in, out[5:1]}; // Shift
                    leds <= 16'b0;
                end
                3'b101: begin
                    out <= direction ? {out[4:0], out[5]}
                                     : {out[0], out[5:1]}; // Rotate
                    leds <= 16'b0;
                end
                default: begin
                    leds <= ~leds; // Blink LEDs on invalid opcode
                    out <= 6'b0;
                end
            endcase
        end
    end
end

endmodule
```

```
module ALSU_tb;

  // Signals
  reg clk;
  reg rst;
  reg [2:0] A, B;
  reg [2:0] opcode;
  reg cin;
  reg serial_in;
  reg red_op_A, red_op_B;
  reg bypass_A, bypass_B;
  reg direction;
  wire [5:0] out;
  wire [15:0] leds;

  // DUT Instantiation
  ALSU DUT (
    .clk(clk),
    .rst(rst),
    .A(A),
    .B(B),
    .opcode(opcode),
    .cin(cin),
    .serial_in(serial_in),
    .red_op_A(red_op_A),
    .red_op_B(red_op_B),
    .bypass_A(bypass_A),
    .bypass_B(bypass_B),
    .direction(direction),
    .out(out),
    .leds(leds)
  );

  // Clock generation
  always #5 clk = ~clk;

  // Testbench logic
  initial begin
    // Initialize signals
    clk = 0;
    rst = 1;
    A = 0;
    B = 0;
    opcode = 0;
    cin = 0;
    serial_in = 0;
    red_op_A = 0;
    red_op_B = 0;
```

```
    bypass_A = 0;
    bypass_B = 0;
    direction = 0;

    // 2.1 Verify Asynchronous Reset Functionality
    @(negedge clk);
    if (out != 0 || leds != 0) $display("Reset failed!");

    // 2.2 Verify Bypass Functionality
    bypass_A = 1;
    bypass_B = 1;
    rst = 0;
    repeat(10) begin
        A = $random;
        B = $random;
        opcode = $urandom_range(0, 5);
        repeat(2) @(negedge clk);
        if (out != A || leds != 0)
            $display("Bypass Test Failed! A=%d, B=%d, Out=%d", A, B, out);
    end

    // 2.3 Verify opcode 0 Functionality
    bypass_A = 0;
    bypass_B = 0;
    opcode = 0;
    repeat(10) begin
        A = $random;
        B = $random;
        red_op_A = $random;
        red_op_B = $random;
        repeat(2) @(negedge clk);
        casex ({red_op_A, red_op_B})
            2'b00: if (out != (A & B)) $display("Opcode 0 Test Failed! A=%d,
B=%d, Out=%d", A, B, out);
            2'b01: if (out != &B) $display("Opcode 0 Test Failed! A=%d, B=%d,
Out=%d", A, B, out);
            2'b1x: if (out != &A) $display("Opcode 0 Test Failed! A=%d, B=%d,
Out=%d", A, B, out);
        endcase
    end

    // 2.4 Verify opcode 1 Functionality
    opcode = 1;
    repeat(10) begin
        A = $random;
        B = $random;
        red_op_A = $random;
        red_op_B = $random;
```

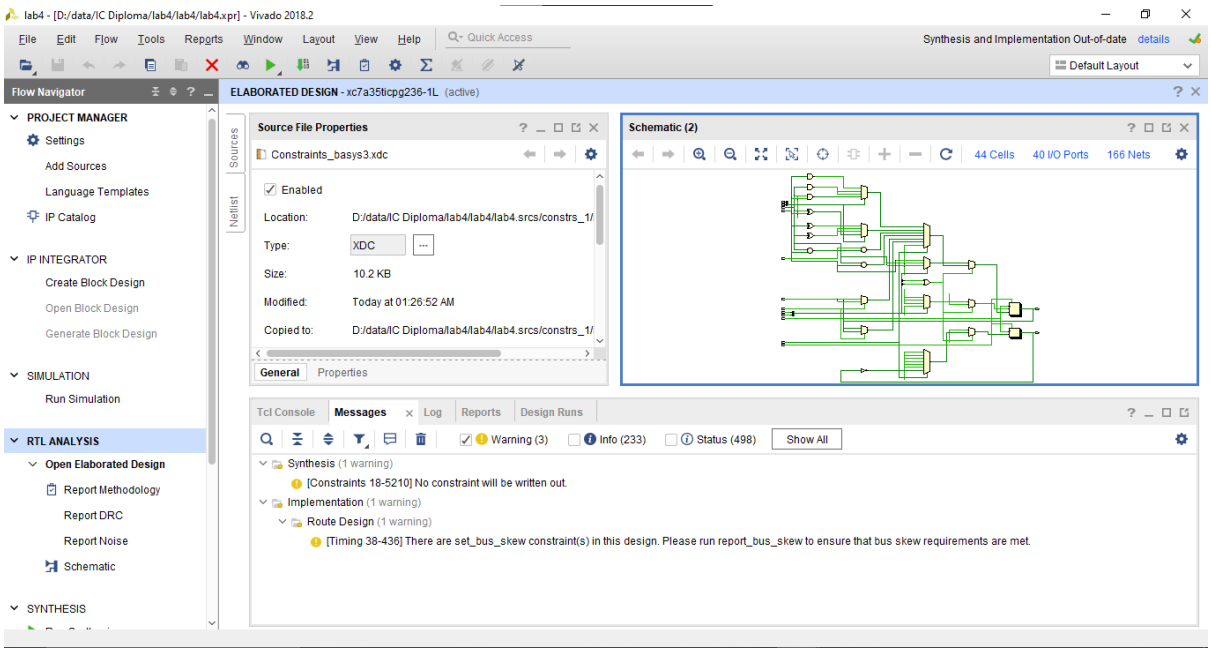
```
        repeat(2) @(negedge clk);
        casex ({red_op_A, red_op_B})
            2'b00: if (out != (A ^ B)) $display("Opcode 0 Test Failed! A=%d,
B=%d, Out=%d", A, B, out);
            2'b01: if (out != ^B) $display("Opcode 0 Test Failed! A=%d, B=%d,
Out=%d", A, B, out);
            2'b1x: if (out != ^A) $display("Opcode 0 Test Failed! A=%d, B=%d,
Out=%d", A, B, out);
        endcase
    end

    // 2.5 Verify opcode 2 Functionality
    red_op_A = 0;
    red_op_B = 0;
    opcode = 2;
    repeat(10) begin
        A = $random;
        B = $random;
        cin = $random;
        repeat(2) @(negedge clk);
        if (out !== (A + B + cin))
            $display("Opcode 2 Test Failed! A=%d, B=%d, Cin=%d, Out=%d", A, B,
cin, out);
    end

    // 2.6 Verify opcode 3 Functionality
    opcode = 3;
    repeat(10) begin
        A = $random;
        B = $random;
        repeat(2) @(negedge clk);
        if (out !== (A * B))
            $display("Opcode 3 Test Failed! A=%d, B=%d, Out=%d", A, B, out);
    end

    // 2.7 Verify opcode 3 Functionality
    opcode = 4;
    repeat(10) begin
        A = $random;
        B = $random;
        serial_in = $random;
        direction = $random;
        repeat(2) @(negedge clk);
    end

    // 2.7 Verify opcode 3 Functionality
    opcode = 5;
    repeat(10) begin
```

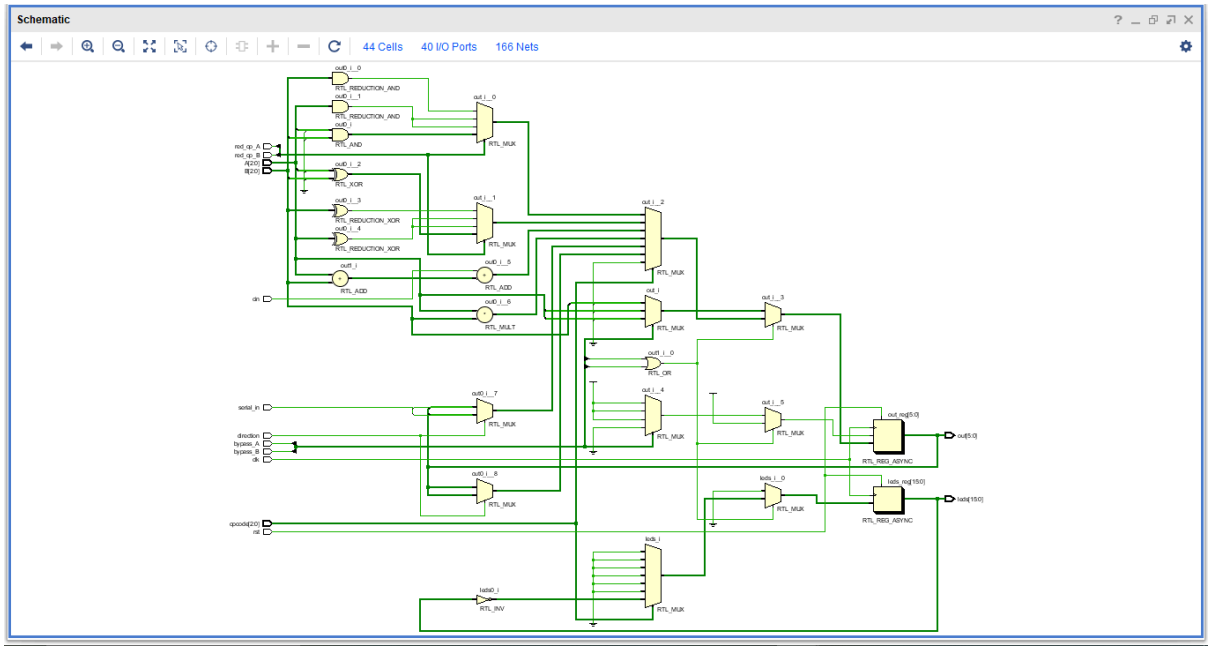
```
        A = $random;
        B = $random;
        serial_in = $random;
        direction = $random;
        repeat(2) @(negedge clk);
    end

    // Finish Simulation
    $display("All tests completed successfully!");
    $finish;
  end

endmodule
```



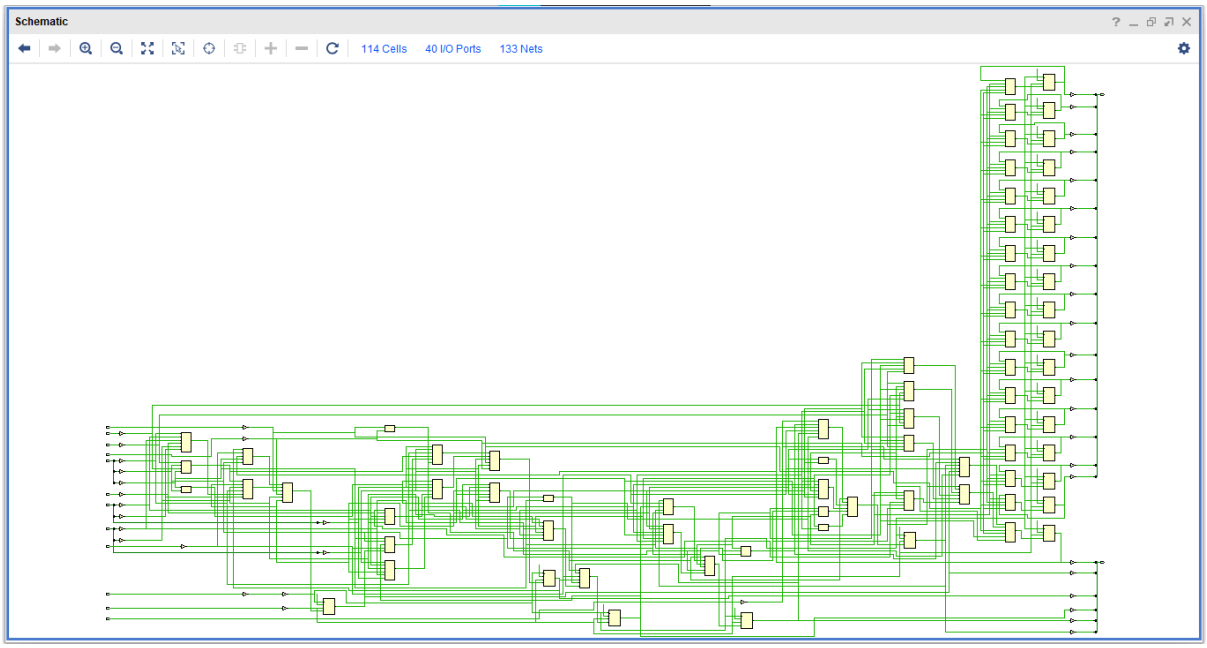

```
> run -all
# All tests completed successfully!

// run.do
vlib wrok

# Compile Design and Testbenches
vlog ALSU.v
vlog ALSU_tb.v

# Run Testbench 1 (DFF mode)
vsim -voptargs=+acc ALSU_tb
add wave *
run -all
quit
```

Elaborated Design:





Synthesis:

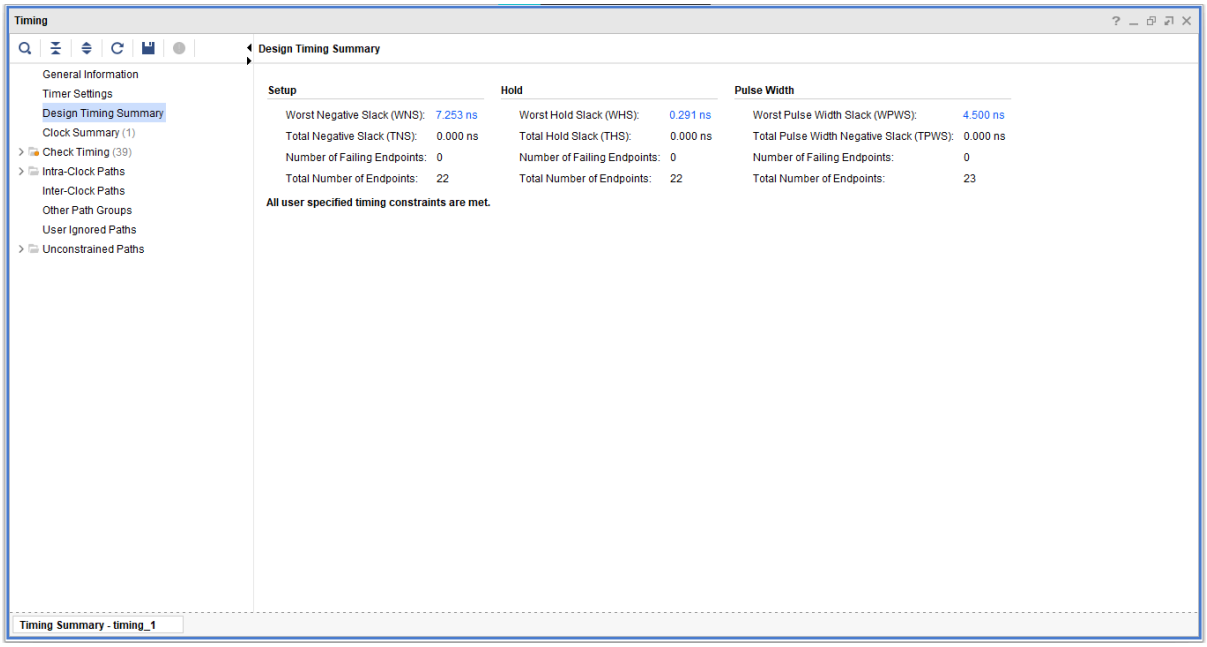


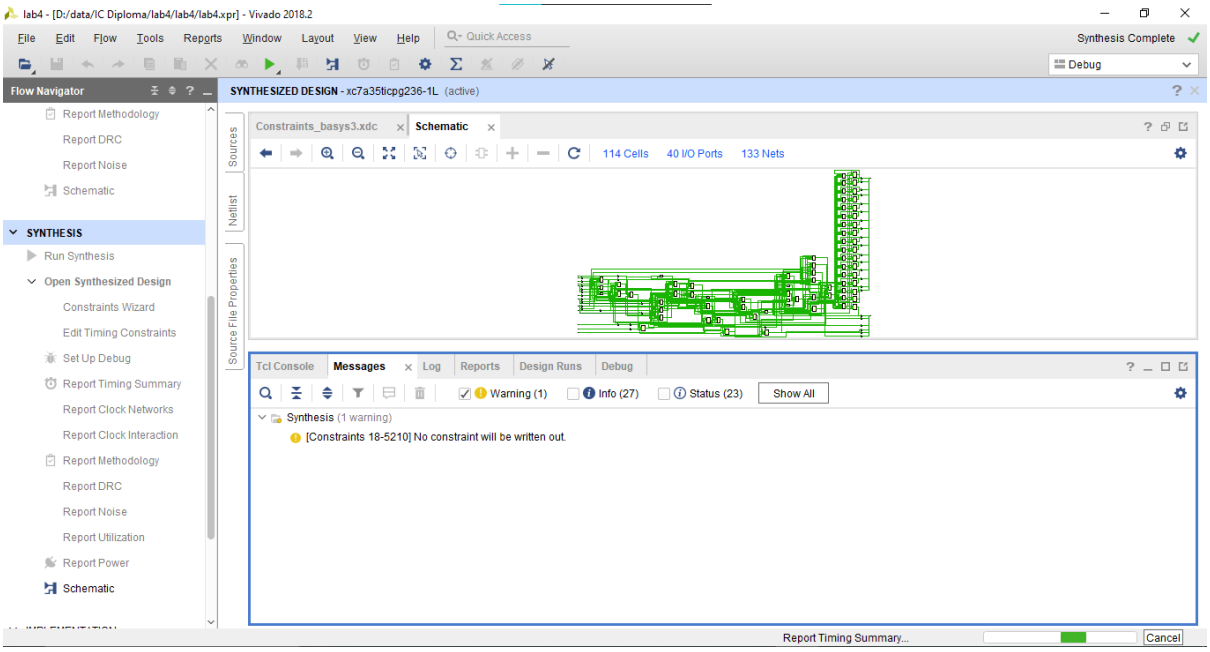
synth_1_synth_report_utilization_0 - synth_1

D:/data/IC Diploma/lab4/lab4/lab4.runs/synth_1/ALSU_utilization_synth.rpt

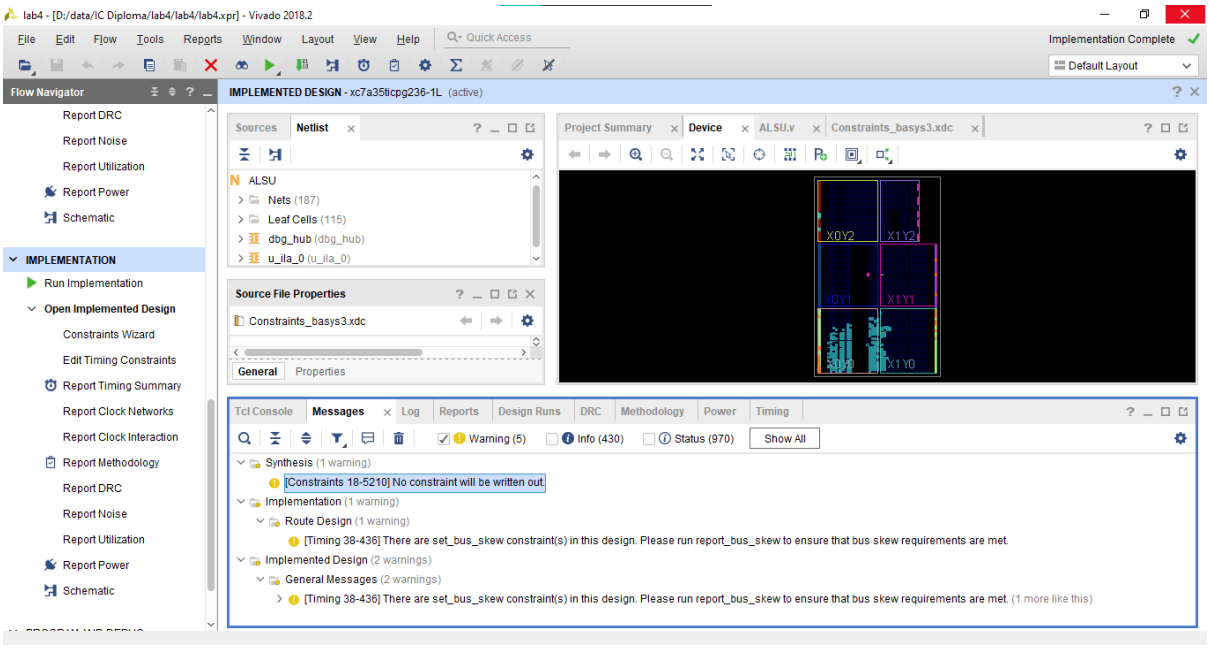
Read-only

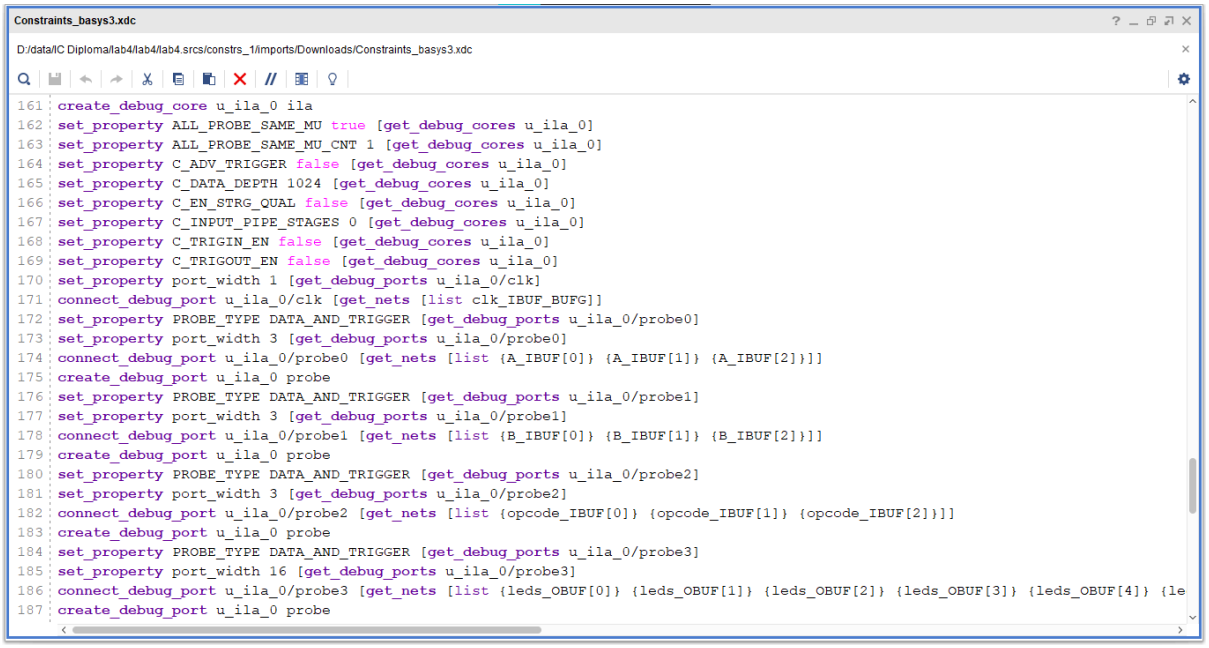
```
1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |   45 |     0 |     20800 |  0.22 |
|   LUT as Logic          |   45 |     0 |     20800 |  0.22 |
|   LUT as Memory         |    0 |     0 |      9600 |  0.00 |
| Slice Registers         |   22 |     0 |     41600 |  0.05 |
|   Register as Flip Flop |   22 |     0 |     41600 |  0.05 |
|   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 s


1.1 Summary of Registers by Type
--------------------------------

+-------+--------------+-------------+--------------+
| Total | Clock Enable | Synchronous | Asynchronous |
+-------+--------------+-------------+--------------+
| 0     |            _ |           - |            - |
| 0     |            _ |           - |          Set |
| 0     |            _ |           - |        Reset |
```

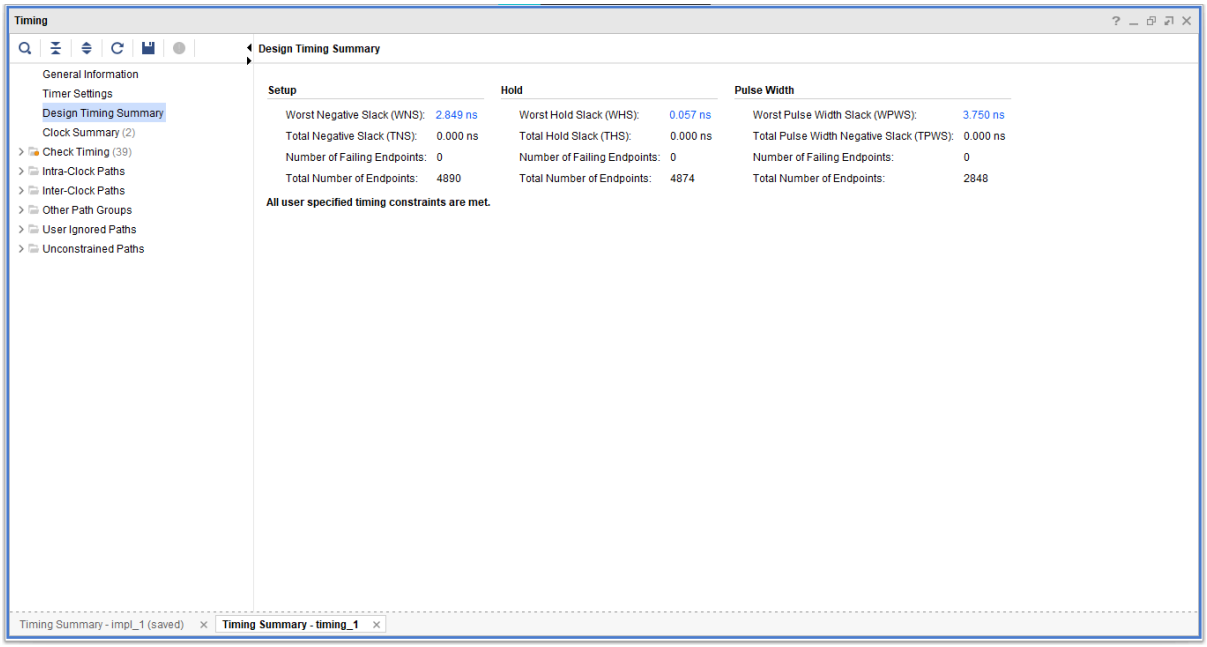

Timing
General Information
Timer Settings
Design Timing Summary
Clock Summary (1)
Check Timing (39)
Intra-Clock Paths
Inter-Clock Paths
Other Path Groups
User Ignored Paths
Unconstrained Paths
Design Timing Summary
Setup
Worst Negative Slack (WNS): 7.253 ns
Total Negative Slack (TNS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 22
Hold
Worst Hold Slack (WHS): 0.291 ns
Total Hold Slack (THS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 22
Pulse Width
Worst Pulse Width Slack (WPWS): 4.500 ns
Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 23
All user specified timing constraints are met.
Timing Summary - timing_1

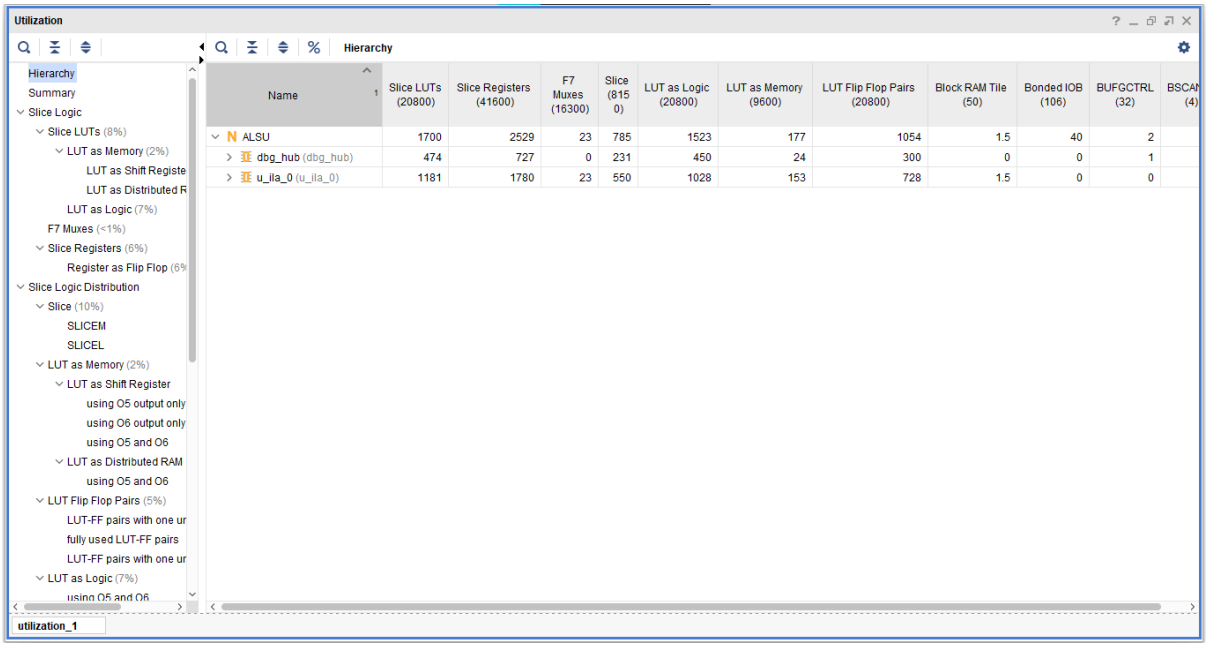

lab4 - [D:/data/IC Diploma/lab4/lab4/lab4.xpr] - Vivado 2018.2
File Edit Flow Tools Reports Window Layout View Help
Quick Access
Synthesis Complete
Debug
Flow Navigator
Report Methodology
Report DRC
Report Noise
Schematic
SYNTHESIS
Run Synthesis
Open Synthesized Design
Constraints Wizard
Edit Timing Constraints
Set Up Debug
Report Timing Summary
Report Clock Networks
Report Clock Interaction
Report Methodology
Report DRC
Report Noise
Report Utilization
Report Power
Schematic
SYNTHESIZED DESIGN - xc7a35ticpg236-1L (active)
Constraints_basys3.xdc
Schematic
114 Cells 40 I/O Ports 133 Nets
Sources
Netlist
Source File Properties
Tcl Console Messages Log Reports Design Runs Debug
Warning (1) Info (27) Status (23) Show All
Synthesis (1 warning)
[Constraints 18-5210] No constraint will be written out.
Report Timing Summary...
Cancel

Implementation:






Timing
General Information
Timer Settings
Design Timing Summary
Clock Summary (2)
Check Timing (39)
Intra-Clock Paths
Inter-Clock Paths
Other Path Groups
User Ignored Paths
Unconstrained Paths
Design Timing Summary
Setup
Worst Negative Slack (WNS): 2.849 ns
Total Negative Slack (TNS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 4890
Hold
Worst Hold Slack (WHS): 0.057 ns
Total Hold Slack (THS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 4874
Pulse Width
Worst Pulse Width Slack (WPWS): 3.750 ns
Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 2848
All user specified timing constraints are met.
Timing Summary - impl_1 (saved)
Timing Summary - timing_1

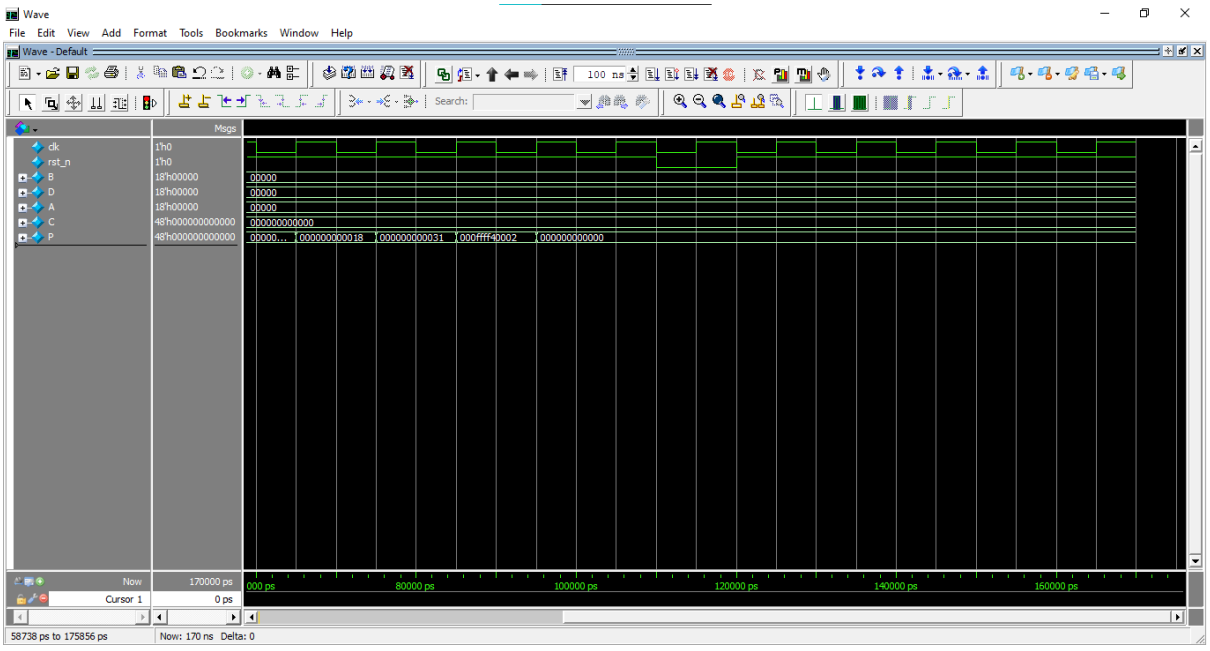

Utilization
Hierarchy
Summary
Slice Logic
Slice LUTs (8%)
LUT as Memory (2%)
LUT as Shift Registe
LUT as Distributed R
LUT as Logic (7%)
F7 Muxes (<1%)
Slice Registers (6%)
Register as Flip Flop (6%
Slice Logic Distribution
Slice (10%)
SLICEM
SLICEL
LUT as Memory (2%)
LUT as Shift Register
using O5 output only
using O6 output only
using O5 and O6
LUT as Distributed RAM
using O5 and O6
LUT Flip Flop Pairs (5%)
LUT-FF pairs with one ur
fully used LUT-FF pairs
LUT-FF pairs with one ur
LUT as Logic (7%)
using O5 and O6
Hierarchy
Name
Slice LUTs (20800)
Slice Registers (41600)
F7 Muxes (16300)
Slice (8150)
LUT as Logic (20800)
LUT as Memory (9600)
LUT Flip Flop Pairs (20800)
Block RAM Tile (50)
Bonded IOB (106)
BUFGCTRL (32)
ALSU 1700 2529 23 785 1523 177 1054 1.5 40 2
dbg_hub (dbg_hub) 474 727 0 231 450 24 300 0 0 1
u_ila_0 (u_ila_0) 1181 1780 23 550 1028 153 728 1.5 0 0
utilization_1

2.

```
module DSP
#(
    parameter OPERATION = "ADD" // OPERATION: take 2 values either "ADD" or
"SUBTRACT", Default value "ADD"
)(
    input wire clk,
    input wire rst_n,
    input wire [17:0] A,
    input wire [17:0] B,
    input wire [47:0] C,
    input wire [17:0] D,
    output reg [47:0] P
);

    localparam ADD = 1'b0, SUBTRACT = 1'b1;
    localparam OP_MODE = (OPERATION == "SUBTRACT") ? SUBTRACT : ADD;

    reg [17:0] first_add_sub_result;
    reg [35:0] multiplier_out;
    reg [47:0] second_add_sub_result;
    reg [17:0] A_r, A_rr;
    reg [17:0] B_r;
    reg [17:0] D_r;
    reg [47:0] C_r;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            A_r <= 18'b0;
            A_rr <= 18'b0;
            B_r <= 18'b0;
            D_r <= 18'b0;
            C_r <= 48'b0;

            first_add_sub_result <= 18'd0;
            multiplier_out <= 36'd0;
            second_add_sub_result <= 48'b0;
            P <= 48'b0;

        end else begin

            // register input ports
            A_r <= A;
            A_rr <= A_r;
            B_r <= B;
            D_r <= D;
            C_r <= C;
```

```
            // Addition or Subtraction operation for first stage
            case (OP_MODE)
                ADD: first_add_sub_result <= D_r + B_r;
                SUBTRACT: first_add_sub_result <= D_r - B_r;
            endcase

            // Multiplication operation
            multiplier_out <= A_rr * first_add_sub_result;

            // Addition or Subtraction operation for second stage
            case (OP_MODE)
                ADD: second_add_sub_result <= multiplier_out + C_r;
                SUBTRACT: second_add_sub_result <= multiplier_out - C_r;
            endcase

            // Pipeline stage
            P <= second_add_sub_result;
        end
    end

endmodule

module DSP_tb;

    reg clk;
    reg rst_n;
    reg [17:0] A;
    reg [17:0] B;
    reg [47:0] C;
    reg [17:0] D;
    wire [47:0] P;

    // Instantiate the DSP48A1_simplified module
    DSP uut (
        .clk(clk),
        .rst_n(rst_n),
        .A(A),
        .B(B),
        .C(C),
        .D(D),
        .P(P)
    );

    // Clock Generation
    always #5 clk = ~clk;

    initial begin
```

```
        // Initialize inputs
        clk = 0;
        rst_n = 0;
        A = 18'd0; B = 18'd0; C = 48'd0; D = 18'd0;

        // Apply reset
        #10 rst_n = 1;

        // Test Case 1: Simple Addition
        #10 A = 18'd5; B = 18'd3; C = 48'd10; D = 18'd2;

        // Test Case 2: Different Values
        #10 A = 18'd7; B = 18'd1; C = 48'd20; D = 18'd6;

        // Test Case 3: Edge Case - Maximum values
        #10 A = 18'h3FFFF; B = 18'h3FFFF; C = 48'hFFFFFFFFFFFF; D = 18'h3FFFF;

        // Apply Delay
        #10 A = 18'd0;  B = 18'd0; C = 48'd0; D = 18'd0;
        #50;

        // Test Case 4: Reset Condition
        #10 rst_n = 0;
        #10 rst_n = 1;

        // End simulation
        #50 $stop;
    end
endmodule
```



3.

```
module TDM_MUX (
    input wire clk,
    input wire rst,
    input wire [1:0] in0,
    input wire [1:0] in1,
    input wire [1:0] in2,
    input wire [1:0] in3,
    output reg [1:0] out
);

    reg [1:0] counter;

    always @(posedge clk or posedge rst) begin
        if (rst)
            counter <= 2'b00; // Reset to 0
        else
            counter <= counter + 2'b01; // Increment counter sequentially
    end

    always @(*) begin
        case (counter)
            2'b00: out = in0;
            2'b01: out = in1;
            2'b10: out = in2;
            2'b11: out = in3;
        endcase
    end
endmodule

module TDM_MUX_tb;
    reg clk;
    reg rst;
    reg [1:0] in0, in1, in2, in3;
    wire [1:0] out;

    // Instantiate the TDM_MUX module
    TDM_MUX uut (
        .clk(clk),
        .rst(rst),
        .in0(in0),
        .in1(in1),
        .in2(in2),
        .in3(in3),
        .out(out)
    );
```

```
    // Clock generation
    always #5 clk = ~clk;

    initial begin
        // Initialize inputs
        clk = 0;
        rst = 1;
        in0 = 2'b00;
        in1 = 2'b00;
        in2 = 2'b00;
        in3 = 2'b00;

        // Apply reset
        #10 rst = 0;

        repeat (1000) begin
            in0 = $random;
            in1 = $random;
            in2 = $random;
            in3 = $random;

            @(negedge clk);

            case (uut.counter)
                2'b00: if (out !== in0) begin
                            $display("Error at time %0t: Expected %b, got %b",
$time, in0, out);
                            $stop;
                        end
                2'b01: if (out !== in1) begin
                            $display("Error at time %0t: Expected %b, got %b",
$time, in1, out);
                            $stop;
                        end
                2'b10: if (out !== in2) begin
                            $display("Error at time %0t: Expected %b, got %b",
$time, in2, out);
                            $stop;
                        end
                2'b11: if (out !== in3) begin
                            $display("Error at time %0t: Expected %b, got %b",
$time, in3, out);
                            $stop;
                        end
            endcase

            $display("Test passed successfully!");
```

```
        end
        $stop;
    end

endmodule


# Test passed successfully!
# Test passed successfully!
# Test passed successfully!
# Test passed successfully!
# Test passed successfully!
# Test passed successfully!
# Test passed successfully!
# Test passed successfully!
# Test passed successfully!
# Test passed successfully!
# Test passed successfully!
# Test passed successfully!
# Test passed successfully!
```