ELEX 7660: Digital System Design

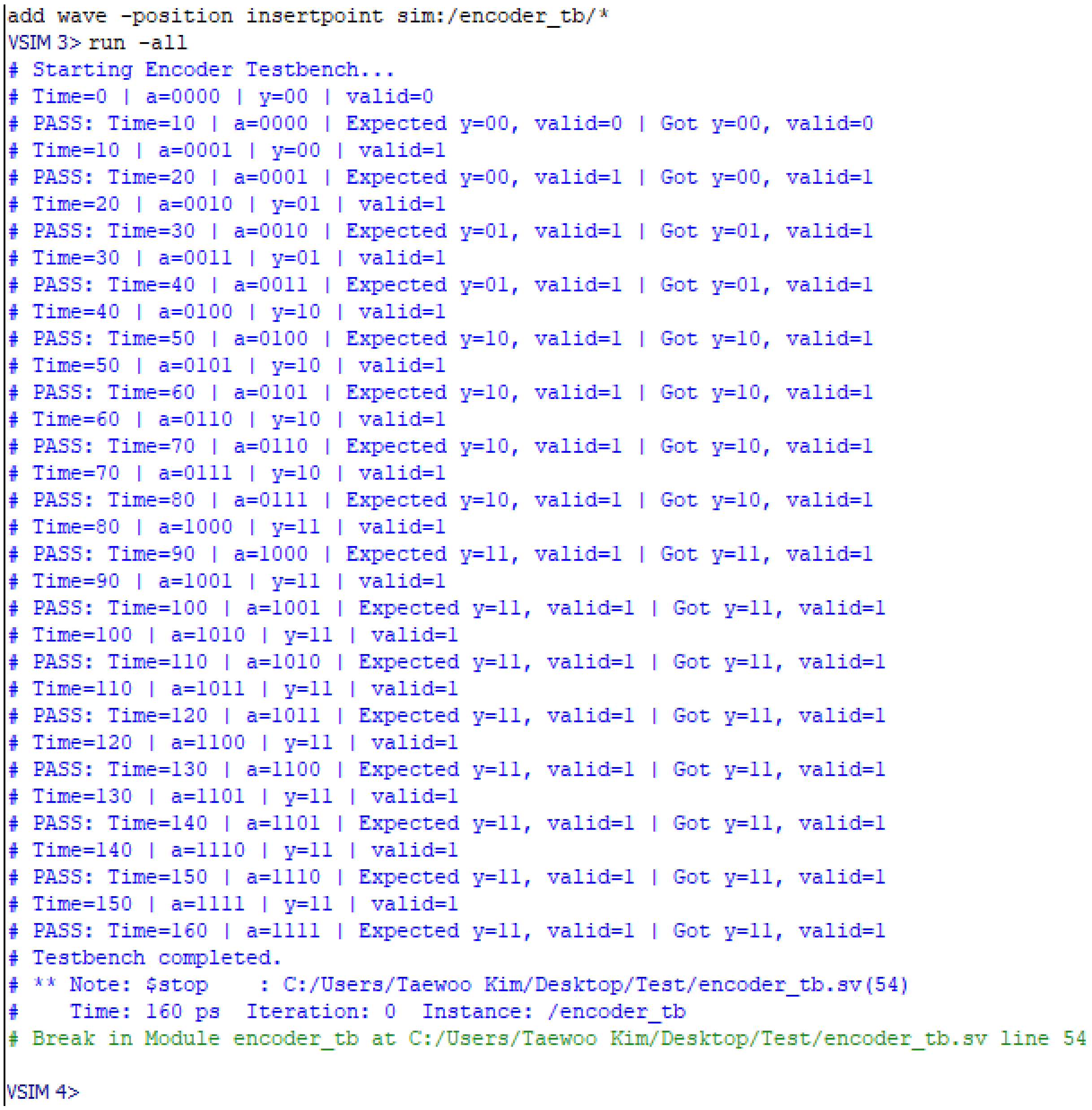
Assignment 1

Student Name: Taewoo Kim	Student Number: A01284763	Set: T

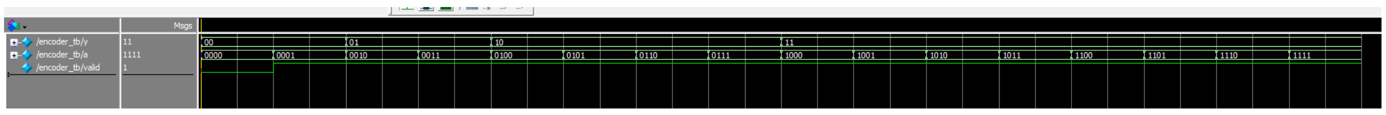
Table of Contents

1 Screenshot of the simulations

1.1 problem 1 (encoder.sv)

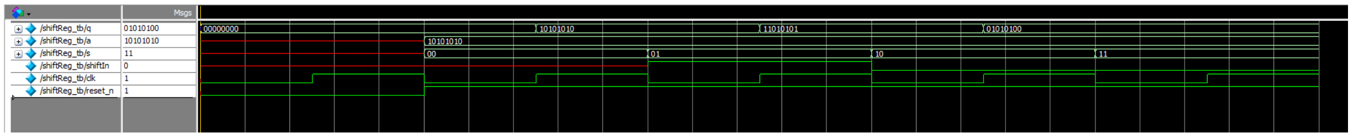


```
add wave -position insertpoint sim:/encoder_tb/*
VSIM 3> run -all
# Starting Encoder Testbench...
# Time=0 | a=0000 | y=00 | valid=0
# PASS: Time=10 | a=0000 | Expected y=00, valid=0 | Got y=00, valid=0
# Time=10 | a=0001 | y=00 | valid=1
# PASS: Time=20 | a=0001 | Expected y=00, valid=1 | Got y=00, valid=1
# Time=20 | a=0010 | y=01 | valid=1
# PASS: Time=30 | a=0010 | Expected y=01, valid=1 | Got y=01, valid=1
# Time=30 | a=0011 | y=01 | valid=1
# PASS: Time=40 | a=0011 | Expected y=01, valid=1 | Got y=01, valid=1
# Time=40 | a=0100 | y=10 | valid=1
# PASS: Time=50 | a=0100 | Expected y=10, valid=1 | Got y=10, valid=1
# Time=50 | a=0101 | y=10 | valid=1
# PASS: Time=60 | a=0101 | Expected y=10, valid=1 | Got y=10, valid=1
# Time=60 | a=0110 | y=10 | valid=1
# PASS: Time=70 | a=0110 | Expected y=10, valid=1 | Got y=10, valid=1
# Time=70 | a=0111 | y=10 | valid=1
# PASS: Time=80 | a=0111 | Expected y=10, valid=1 | Got y=10, valid=1
# Time=80 | a=1000 | y=11 | valid=1
# PASS: Time=90 | a=1000 | Expected y=11, valid=1 | Got y=11, valid=1
# Time=90 | a=1001 | y=11 | valid=1
# PASS: Time=100 | a=1001 | Expected y=11, valid=1 | Got y=11, valid=1
# Time=100 | a=1010 | y=11 | valid=1
# PASS: Time=110 | a=1010 | Expected y=11, valid=1 | Got y=11, valid=1
# Time=110 | a=1011 | y=11 | valid=1
# PASS: Time=120 | a=1011 | Expected y=11, valid=1 | Got y=11, valid=1
# Time=120 | a=1100 | y=11 | valid=1
# PASS: Time=130 | a=1100 | Expected y=11, valid=1 | Got y=11, valid=1
# Time=130 | a=1101 | y=11 | valid=1
# PASS: Time=140 | a=1101 | Expected y=11, valid=1 | Got y=11, valid=1
# Time=140 | a=1110 | y=11 | valid=1
# PASS: Time=150 | a=1110 | Expected y=11, valid=1 | Got y=11, valid=1
# Time=150 | a=1111 | y=11 | valid=1
# PASS: Time=160 | a=1111 | Expected y=11, valid=1 | Got y=11, valid=1
# Testbench completed.
# ** Note: $stop    : C:/Users/Taewoo Kim/Desktop/Test/encoder_tb.sv(54)
#    Time: 160 ps  Iteration: 0  Instance: /encoder_tb
# Break in Module encoder_tb at C:/Users/Taewoo Kim/Desktop/Test/encoder_tb.sv line 54

VSIM 4>
```



1.2 problem 2 (shitReg.sv)

```
VSIM 3> run -all
# Starting Shift Register Testbench...
# PASS: Time=20 | s=00 | shiftIn=x | a=10101010 | Expected q=10101010 | Got q=10101010
# PASS: Time=30 | s=01 | shiftIn=1 | a=10101010 | Expected q=11010101 | Got q=11010101
# PASS: Time=40 | s=10 | shiftIn=0 | a=10101010 | Expected q=01010100 | Got q=01010100
# PASS: Time=50 | s=11 | shiftIn=0 | a=10101010 | Expected q=01010100 | Got q=01010100
# Testbench completed.
# ** Note: $stop    : C:/Users/Taewoo Kim/Desktop/Test/shiftReg_tb.sv(60)
#    Time: 50 ps  Iteration: 0  Instance: /shiftReg_tb
# Break in Module shiftReg_tb at C:/Users/Taewoo Kim/Desktop/Test/shiftReg_tb.sv line 60
```



2 Source code of the module

2.1 encoder.sv

```
// File: encoder.sv
// Description: This is a simple priority encoder module
// Author: Taewoo Kim
// Date: 2025-02-02

module encoder (
    output logic [1:0] y,  // 2-bit output representing the encoded value
    output logic valid,     // Output signal indicating if the input is valid
    input logic [3:0] a     // 4-bit input representing the encoded signal
);

    // Always block using combinational logic to determine output
    always_comb begin
        // Case statement to encode the input based on priority
        casez (a)
            4'b0000: {y[1], y[0], valid} = 3'b000; // No active input, output is 00, valid=0
            4'b0001: {y[1], y[0], valid} = 3'b001; // Input 0001, encode as 00, valid=1
            4'b001?: {y[1], y[0], valid} = 3'b011; // Input 001x, encode as 01, valid=1
            4'b01??: {y[1], y[0], valid} = 3'b101; // Input 01xx, encode as 10, valid=1
            4'b1???: {y[1], y[0], valid} = 3'b111; // Input 1xxx, encode as 11, valid=1
            default: {y[1], y[0], valid} = 3'b000; // Default case, output is 00, valid=0
        endcase
    end

endmodule
     case (digit)
        2'b00: disp_digit = desired_freq[3:0]; // if digit is 0
        2'b01: disp_digit = desired_freq[7:4]; // if digit is 1
        2'b10: disp_digit = desired_freq[11:8]; // if digit is 2
        2'b11: disp_digit = desired_freq[15:12]; // if digit is 3
        default: disp_digit = 4'b0000;       // Default or error value
      endcase
  end


endmodule
```

2.2 encoder_tb.sv code

```
// File: encoder_tb.sv
// Description: This is a simple testbench to check priority encoder module
// Author: Taewoo Kim
// Date: 2025-02-02

module encoder_tb;

    // Declare testbench signals
    logic [1:0] y;    // Output of the encoder
    logic [3:0] a;    // Input to the encoder
    logic valid;      // Valid output flag

    // Instantiate the encoder module
    encoder dut (.y(y), .a(a), .valid(valid));

    // Task to check expected vs actual output
    task check_output(input [3:0] a_in, input [1:0] expected_y, input
expected_valid);
        a = a_in; // Apply input to the encoder
        #10; // Wait for output to settle

        // Compare expected vs actual results and display outcome
        if (y === expected_y && valid === expected_valid)
            $display("PASS: Time=%0t | a=%b | Expected y=%b, valid=%b | Got y=%b,
valid=%b",
                     $time, a, expected_y, expected_valid, y, valid);
        else
            $display("FAIL: Time=%0t | a=%b | Expected y=%b, valid=%b | Got y=%b,
valid=%b",
                     $time, a, expected_y, expected_valid, y, valid);
    endtask

    initial begin
        $display("Starting Encoder Testbench..."); // Start message
        $monitor("Time=%0t | a=%b | y=%b | valid=%b", $time, a, y, valid); // Monitor
values

        // Test cases with expected outputs
        check_output(4'b0000, 2'b00, 0); // No valid input
        check_output(4'b0001, 2'b00, 1); // Encoding input 0001
        check_output(4'b0010, 2'b01, 1); // Encoding input 0010
        check_output(4'b0011, 2'b01, 1); // Encoding input 0011
        check_output(4'b0100, 2'b10, 1); // Encoding input 0100
        check_output(4'b0101, 2'b10, 1); // Encoding input 0101
        check_output(4'b0110, 2'b10, 1); // Encoding input 0110
        check_output(4'b0111, 2'b10, 1); // Encoding input 0111
        check_output(4'b1000, 2'b11, 1); // Encoding input 1000
        check_output(4'b1001, 2'b11, 1); // Encoding input 1001
        check_output(4'b1010, 2'b11, 1); // Encoding input 1010
        check_output(4'b1011, 2'b11, 1); // Encoding input 1011
        check_output(4'b1100, 2'b11, 1); // Encoding input 1100
        check_output(4'b1101, 2'b11, 1); // Encoding input 1101
        check_output(4'b1110, 2'b11, 1); // Encoding input 1110
        check_output(4'b1111, 2'b11, 1); // Encoding input 1111

        // Finish simulation
        $display("Testbench completed.");
        $stop;
    end

endmodule
```

2.3 shiftReg.sv

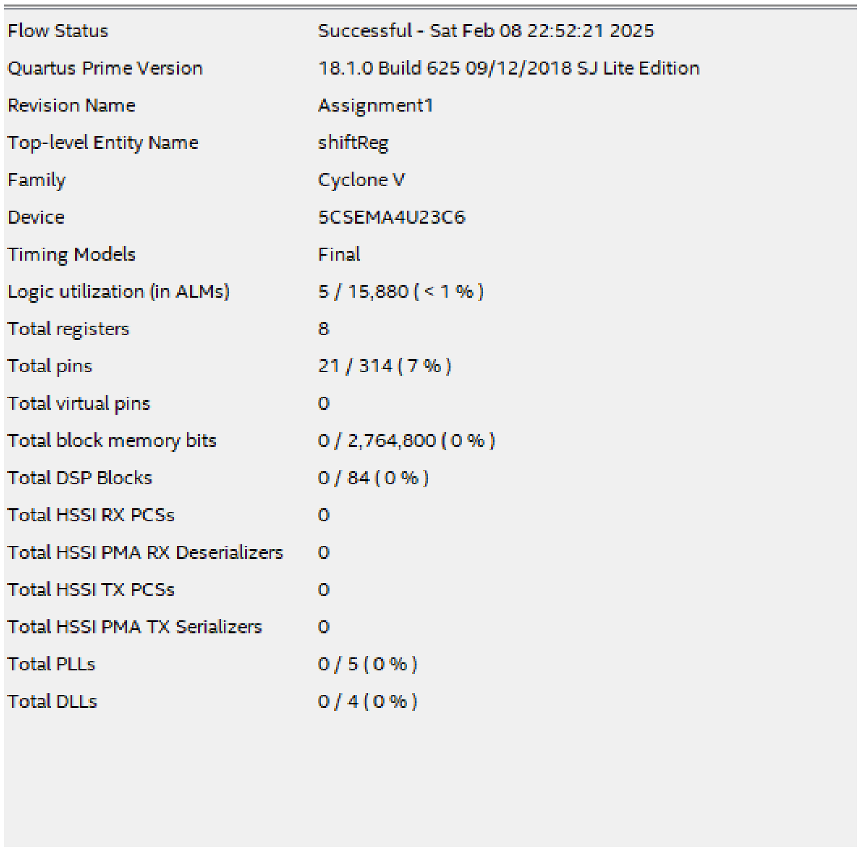
```
// File: shiftReg.sv
// Description: This is a simple module to create an 8-bit shift register
// Author: Taewoo Kim
// Date: 2025-02-02

module shiftReg (
    output logic [7:0] q,  // 8-bit output register
    input logic [7:0] a,   // 8-bit parallel load input
    input logic [1:0] s,   // 2-bit control signal to determine operation
    input logic shiftIn,   // Single-bit shift input for shifting operations
    input logic clk,       // Clock signal
    input logic reset_n    // Active-low asynchronous reset
);

    // Always block triggered on the rising edge of the clock or falling edge of
reset
    always_ff @(posedge clk, negedge reset_n) begin
        if (~reset_n) begin // If reset is active (low)
            q <= 8'b0; // Clear the shift register
        end else begin // Otherwise, check operation based on control signal 's'
            case (s) // Case statement to determine shift operation
                2'b00 : q <= a; // Parallel load from input 'a'
                2'b01 : q <= {shiftIn, q[7:1]}; // Shift right: insert shiftIn at
MSB, shift right
                2'b10 : q <= {q[6:0], shiftIn}; // Shift left: insert shiftIn at LSB,
shift left
                2'b11 : q <= q; // Hold the current value
            endcase
        end
    end
endmodule
```

2.4 shiftReg_tb.sv

```
// File: shiftReg_tb.sv
// Description: This is a simple testbench to check shift register module
// Author: Taewoo Kim
// Date: 2025-02-02

module shiftReg_tb;

    // Declare testbench signals
    logic [7:0] q;          // Output of the shift register
    logic [7:0] a;          // Parallel load input
    logic [1:0] s;          // Control signal for operation
    logic shiftIn;          // Input for shifting
    logic clk;              // Clock signal
    logic reset_n;          // Asynchronous active-low reset

    // Instantiate the shift register module
    shiftReg dut
(.q(q), .a(a), .s(s), .shiftIn(shiftIn), .clk(clk), .reset_n(reset_n));

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

    // Task to check expected vs actual output
    task check_output(input [7:0] expected_q);
        #10; // Wait for one clock cycle
        if (q === expected_q)
            $display("PASS: Time=%0t | s=%b | shiftIn=%b | a=%b | Expected q=%b | Got
q=%b",
                     $time, s, shiftIn, a, expected_q, q);
        else
            $display("FAIL: Time=%0t | s=%b | shiftIn=%b | a=%b | Expected q=%b | Got
q=%b",
                     $time, s, shiftIn, a, expected_q, q);
    endtask
    initial begin
        $display("Starting Shift Register Testbench...");
        clk = 0;
        reset_n = 0; // Apply reset
        #10 reset_n = 1; // Release reset

        // Test case 1: Parallel load
        a = 8'b10101010;
        s = 2'b00;
        check_output(8'b10101010);

        // Test case 2: Shift right with shiftIn = 1
        shiftIn = 1;
        s = 2'b01;
        check_output(8'b11010101);

        // Test case 3: Shift left with shiftIn = 0
        shiftIn = 0;
        s = 2'b10;
        check_output(8'b01010100); // Correct expected output for left shift

        // Test case 4: Hold current state
        s = 2'b11;
        check_output(8'b01010100); // Hold should retain the previous value

        // Finish simulation
        $display("Testbench completed.");
        $stop;
    end

endmodule
```

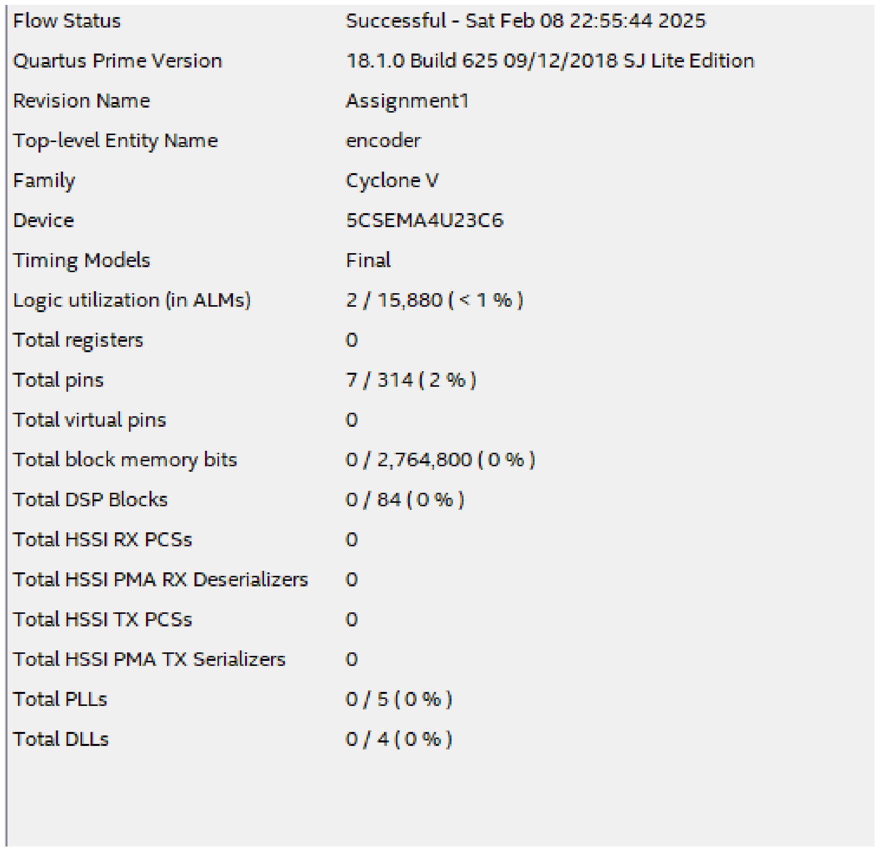
3 Quartus compilation report

3.1 shiftReg compilation report



Flow Status	Successful - Sat Feb 08 22:52:21 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Assignment1
Top-level Entity Name	shiftReg
Family	Cyclone V
Device	5CSEMA4U23C6
Timing Models	Final
Logic utilization (in ALMs)	5 / 15,880 (< 1 %)
Total registers	8
Total pins	21 / 314 (7 %)
Total virtual pins	0
Total block memory bits	0 / 2,764,800 (0 %)
Total DSP Blocks	0 / 84 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 5 (0 %)
Total DLLs	0 / 4 (0 %)

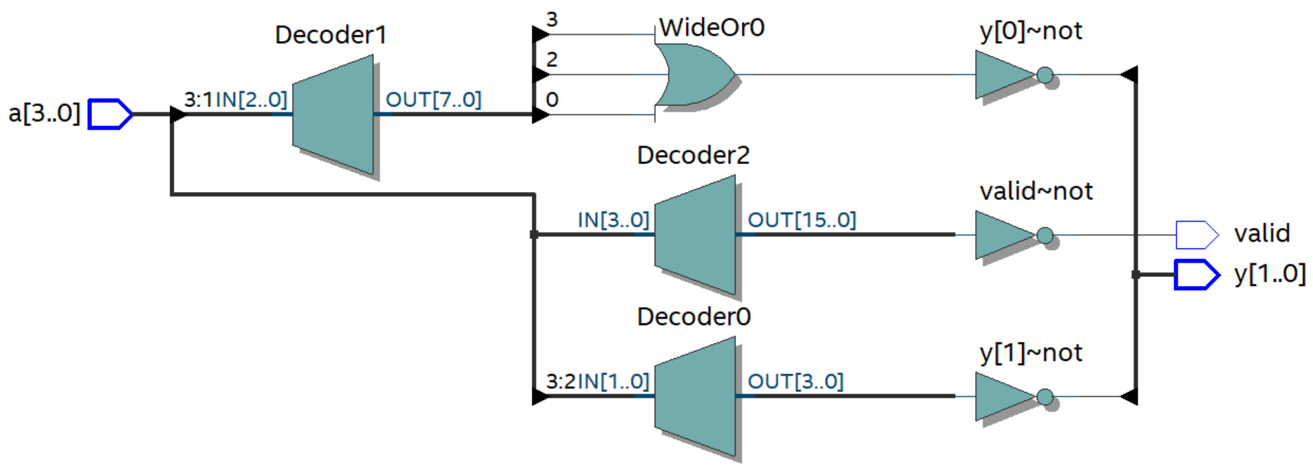
3.2 encoder compilation report



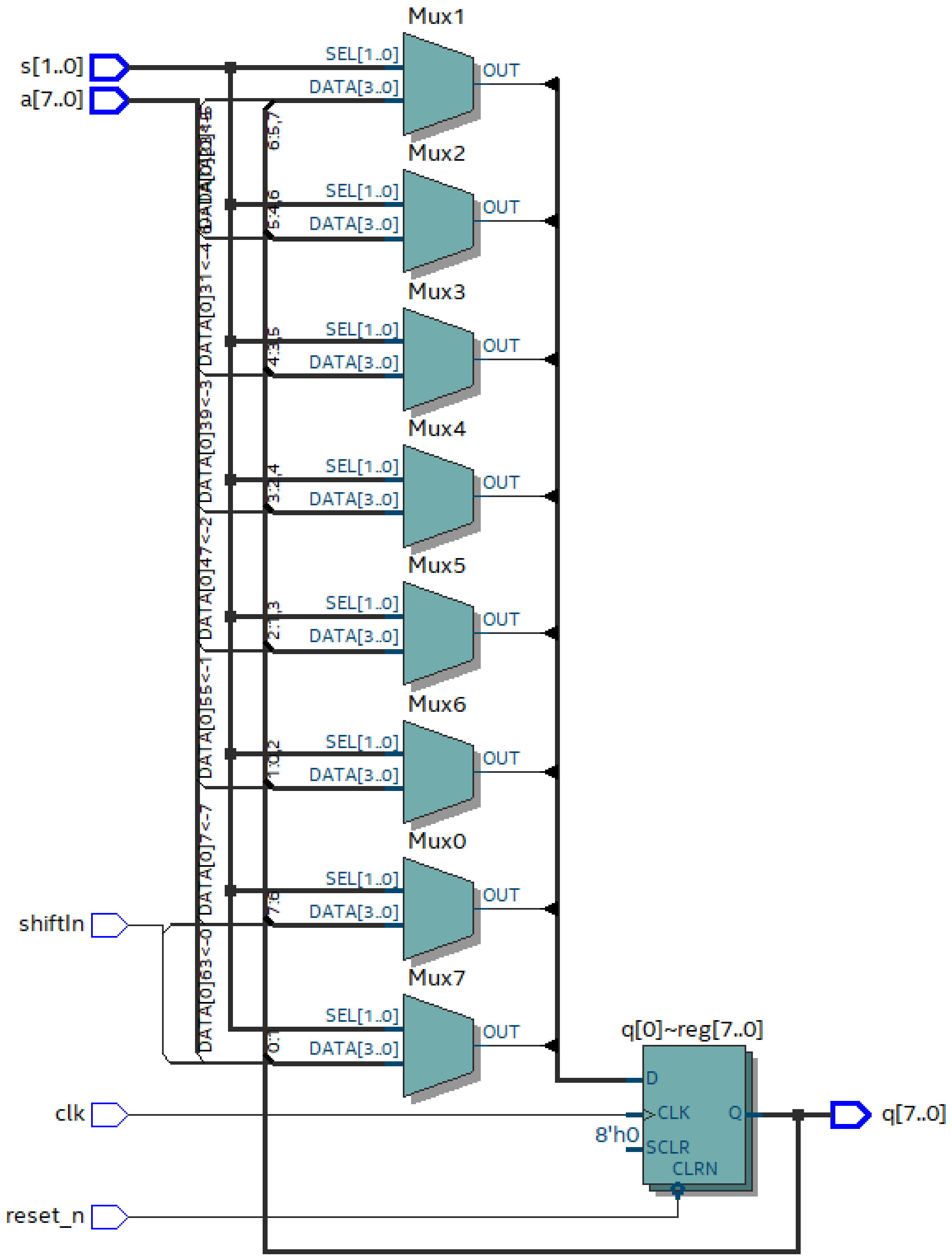
Flow Status	Successful - Sat Feb 08 22:55:44 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Assignment1
Top-level Entity Name	encoder
Family	Cyclone V
Device	5CSEMA4U23C6
Timing Models	Final
Logic utilization (in ALMs)	2 / 15,880 (< 1 %)
Total registers	0
Total pins	7 / 314 (2 %)
Total virtual pins	0
Total block memory bits	0 / 2,764,800 (0 %)
Total DSP Blocks	0 / 84 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 5 (0 %)
Total DLLs	0 / 4 (0 %)

4 RTL Netlist

4.1 encoder.sv



4.2 shiftReg.sv


Mux1
Mux2
Mux3
Mux4
Mux5
Mux6
Mux0
Mux7
SEL[1..0]
DATA[3..0]
OUT
s[1..0]
a[7..0]
shiftIn
clk
reset_n
q[0]~reg[7..0]
D
CLK
Q
SCLR
CLRN
8'h0
q[7..0]