A blue square with white text

Description automatically generated

ELEX 7660: Digital System Design

Assignment 1

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

Table of Contents

[1 Screenshot of the simulations 3](#_Toc189947875)

[1.1 problem 1 (encoder.sv) 3](#_Toc189947876)

[1.2 problem 2 (shitReg.sv) 4](#_Toc189947877)

[2 Source code of the module 4](#_Toc189947878)

[2.1 encoder.sv 4](#_Toc189947879)

[2.2 encoder\_tb.sv code 5](#_Toc189947880)

[2.3 shiftReg.sv 6](#_Toc189947881)

[2.4 shiftReg\_tb.sv 7](#_Toc189947882)

[3 Quartus compilation report 9](#_Toc189947883)

[3.1 shiftReg compilation report 9](#_Toc189947884)

[3.2 encoder compilation report 10](#_Toc189947885)

[4 RTL Netlist 10](#_Toc189947886)

[4.1 encoder.sv 10](#_Toc189947887)

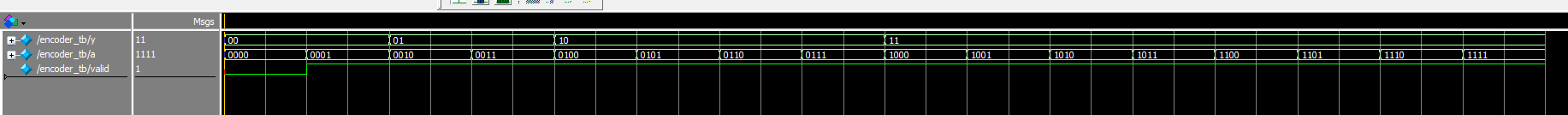
[4.2 shiftReg.sv 11](#_Toc189947888)

# Screenshot of the simulations

## problem 1 (encoder.sv)

A screenshot of a computer program

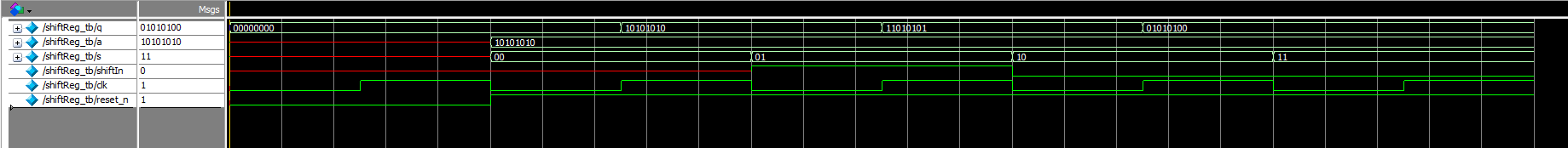
AI-generated content may be incorrect.



## problem 2 (shitReg.sv)

A computer code with numbers and letters

AI-generated content may be incorrect.



# Source code of the module

## 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 |

## 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 |

## 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 |

## 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 |

# Quartus compilation report

## shiftReg compilation report

A screenshot of a computer

AI-generated content may be incorrect.

## encoder compilation report

A screenshot of a computer

AI-generated content may be incorrect.

# RTL Netlist

## encoder.sv

A diagram of a computer program

AI-generated content may be incorrect.

## shiftReg.sv

A diagram of a computer

AI-generated content may be incorrect.