A blue square with white text

Description automatically generated

ELEX 7660: Digital System Design

Assignment 2

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

Table of Contents

[1 Screenshot of the simulations 3](#_Toc191246200)

[1.1 problem 1 (seqDetect.sv) 3](#_Toc191246201)

[1.2 problem 2 (vendingMachine.sv) 3](#_Toc191246202)

[2 Source code of the module 3](#_Toc191246203)

[2.1 seqDetect.sv 3](#_Toc191246204)

[2.2 seqDetect\_tb.sv 5](#_Toc191246205)

[2.3 vendingMachine.sv 6](#_Toc191246206)

[2.4 vendingMachine\_tb.sv 7](#_Toc191246207)

[3 Quartus compilation report 11](#_Toc191246208)

[3.1 seqDetect compilation report 11](#_Toc191246209)

[3.2 vendingMachine compilation report 12](#_Toc191246210)

[4 RTL Netlist 12](#_Toc191246211)

[4.1 seqDetect.sv 12](#_Toc191246212)

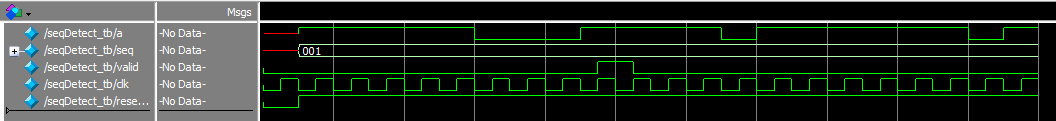
[4.2 vendingMachine.sv 13](#_Toc191246213)

# Screenshot of the simulations

## problem 1 (seqDetect.sv)

A computer screen shot of a computer code

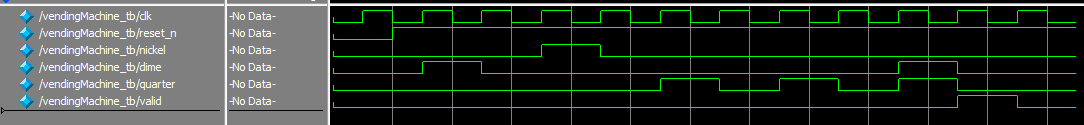
AI-generated content may be incorrect.



## problem 2 (vendingMachine.sv)

A computer screen shot of text

AI-generated content may be incorrect.



# Source code of the module

## seqDetect.sv

|  |
| --- |
| // File: seqDetect.sv  // Description: This is a simple sequence detector module  //              The output valid should be asserted for one clock cycle if the last N 'a' bits match the seq input.  // Author: Taewoo Kim  // Date: 2025-02-20  module seqDetect #(parameter N=6)(         // Default N --> 6      output logic valid,                    // Output signal that is asserted when sequence is detected      input logic a,                         // Serial input bit stream      input logic [N-1:0] seq,               // Sequence to be detected (N-bit pattern)      input logic clk, reset\_n               // Clock and active-low reset  );      // Declare a register to store the last N received bits      logic [N-1:0] valid\_buffer, next\_valid\_buffer;        // Counter to keep track of how many bits have been received      logic [$clog2(N+1)-1:0] bit\_count, next\_bit\_count;        // Temporary variable for next valid state      logic next\_valid;      //-------------------------      // 1) Combinational logic --> update one by one      //-------------------------      always\_comb begin          // Shift the new bit 'a' into the valid\_buffer (shifting left)          next\_valid\_buffer = {valid\_buffer[N-2:0], a};            // Increment bit count until it reaches N, then keep it at N          next\_bit\_count    = (bit\_count < N) ? (bit\_count + 1) : bit\_count;            // Check if we have received at least N bits and the last N bits match the expected sequence          next\_valid        = (next\_bit\_count >= N && next\_valid\_buffer == seq);      end      //-------------------------      // 2.) always\_ff logic --> synchronize the variables      //-------------------------      // Using always\_ff to synchronize valid\_buffer, bit\_count, and valid with the clock      always\_ff @(posedge clk, negedge reset\_n) begin          if(~reset\_n) begin  // Asynchronous active-low reset              valid\_buffer <= '0;  // Clear the buffer storing received bits              bit\_count    <= '0;  // Reset the bit counter              valid        <= 1'b0; // Deassert valid output          end          else if (next\_valid) begin // If sequence detected, reset the system              valid        <= 1'b1;  // Assert valid for one cycle              valid\_buffer <= '0;    // Reset the buffer              bit\_count    <= '0;    // Reset the counter          end          else begin              valid\_buffer <= next\_valid\_buffer; // Update valid\_buffer with the shifted sequence              bit\_count    <= next\_bit\_count;    // Update bit counter              valid        <= 1'b0;              // Deassert valid in other cycles          end      end  endmodule |

## seqDetect\_tb.sv

|  |
| --- |
| // File: seqDetect\_tb.sv  // Description: This is a simple testbench to check the sequence detector module  // Author: Taewoo Kim  // Date: 2025-02-10  module seqDetect\_tb;      // Declare testbench signals      parameter N = 3;          // Parameter to define the sequence width      logic a;                  // Serial input to the sequence detector      logic [N-1:0] seq;        // Register to store the detected sequence      logic valid;              // Output signal indicating sequence detection      logic clk, reset\_n;       // Clock and active-low reset signals      // Test patterns to be applied to the sequence detector      logic [N-1:0] test\_patterns [6:0] = '{3'b101, 3'b111, 3'b011, 3'b111, 3'b001, 3'b110, 3'b111};      // Instantiate the sequence detector module      seqDetect #(3) dut (.valid(valid), .a(a), .seq(seq), .clk(clk), .reset\_n(reset\_n));        // Generate a clock signal with a period of 10 time units      always #5 clk = ~clk;        // Task to check if the actual output matches the expected output      task check\_output(input expected\_valid);          if (valid === expected\_valid)              $display("PASS: valid=%b | sequence=%b", valid, seq);          else              $display("FAIL: valid=%b | sequence=%b", valid, seq);      endtask        // Task to send a sequence of bits serially to the sequence detector      task send\_serial\_input(input [N-1:0] data);          for (int i = N-1; i >= 0; i--) begin              a = data[i];  // Send bits one at a time              #10;          // Wait for one clock cycle          end      endtask      // Initial block to start the test sequence      initial begin          $display("Starting Shift Register Testbench...");          clk = 0;           // Initialize clock to 0          reset\_n = 0;       // Apply reset (active-low)          #10 reset\_n = 1;   // Release reset after 10 time units            seq = 3'b001;      // Initialize sequence to a known value            // Loop through all test patterns and apply them sequentially          for (int i = 0; i < 7; i++) begin              send\_serial\_input(test\_patterns[i]); // Send the test pattern serially              if(seq == test\_patterns[i])                  check\_output(1);  // Expected output is 1 if sequence matches              else                  check\_output(0);  // Otherwise, expected output is 0          end          // End of simulation          $display("Testbench completed.");          $stop; // Stop simulation      end    endmodule |

## vendingMachine.sv

|  |
| --- |
| // File: vendingMachine.sv  // Description: Vending Machine module that accumulates coin inputs (nickel, dime, quarter)  //              and sets 'valid' high when the total reaches 100 cents.  // Author: Taewoo Kim  // Date: 2025-02-20  module vendingMachine (output logic valid, // Output signal indicating if total amount has reached 100 cents                         input logic nickel, dime, quarter, // Coin inputs: 5 cents, 10 cents, 25 cents                         input logic clk, reset\_n); // Clock and active-low reset signals      // Define storage for total amount (in cents)      // Since max value needed is 100, clog2(100) bits are enough      logic [$clog2(100)-1:0] total\_amount;      logic [$clog2(100)-1:0] next\_total\_amount; // Next state value for total\_amount      logic next\_valid; // Next state value for valid signal      // Combinational logic: Compute next total amount based on coin inputs      always\_comb begin          next\_total\_amount = total\_amount                              + (nickel ? 5 : 0)   // Add 5 cents if nickel is inserted                              + (dime ? 10 : 0)    // Add 10 cents if dime is inserted                              + (quarter ? 25 : 0); // Add 25 cents if quarter is inserted            // Check if total amount reaches or exceeds 100 cents          next\_valid = (next\_total\_amount >= 100) ? 1'b1 : 1'b0;      end        // Sequential logic: Update total\_amount and valid on clock edge      always\_ff @(posedge clk, negedge reset\_n) begin          if (~reset\_n) begin              // Reset condition: Set total\_amount and valid to 0              total\_amount <= 0;              valid <= 0;          end else if (next\_valid) begin              // If total amount reaches 100 cents, reset total\_amount and set valid high              total\_amount <= 0;              valid <= 1'b1;          end else begin              // Otherwise, update total\_amount with new value and keep valid low              total\_amount <= next\_total\_amount;              valid <= 1'b0;          end      end  endmodule |

## vendingMachine\_tb.sv

|  |
| --- |
| // File: vendingMachine\_tb.sv  // Description: Testbench for the vendingMachine module to verify its functionality.  // Author: Taewoo Kim  // Date: 2025-02-20  module vendingMachine\_tb();    // Declare test signals    logic clk, reset\_n, nickel, dime, quarter, valid;      // Instantiate the vendingMachine module (Device Under Test - DUT)    vendingMachine dut (        .valid(valid),        .nickel(nickel),        .dime(dime),        .quarter(quarter),        .clk(clk),        .reset\_n(reset\_n)    );      // Clock generation: toggles every 5 time units to create a 10-unit clock period    always #5 clk = ~clk;    // Test sequence    initial begin      // Initialize signals      reset\_n = 0;      nickel = 0;      dime = 0;      quarter = 0;        clk = 0;            // Initialize clock      reset\_n = 0;        // Hold reset low      #10 reset\_n = 1;    // Release reset after 10 time units      // Deposit 10 cents (dime)      @(posedge clk);      dime = 1;      @(posedge clk);      dime = 0;      // Check valid output, should still be 0 since total is only 10 cents      if (valid == 0)        $display("PASS: VALID should be 0!!! We are depositing the dime.");      else        $display("FAIL: valid should be 0 but got %b !!!", valid);        // Deposit 5 cents (nickel)      // Total should be 15 cents      @(posedge clk);      nickel = 1;      @(posedge clk);      nickel = 0;      // Check valid output, should still be 0 since total is only 15 cents      if (valid == 0)        $display("PASS: VALID should be 0!!! We are depositing the nickel.");      else        $display("FAIL: valid should be 0 but got %b", valid);      // Deposit 25 cents (quarter)      // Total should be 40 cents      @(posedge clk);      quarter = 1;      @(posedge clk);      quarter = 0;      // Check valid output, should still be 0 since total is only 40 cents      if (valid == 0)        $display("PASS: VALID should be 0!!! We are depositing the quarter.");      else        $display("FAIL: valid expected 0 but got %b", valid);        // Deposit another 25 cents (quarter)      // Total should be 65 cents      @(posedge clk);      quarter = 1;      @(posedge clk);      quarter = 0;      // Check valid output, should still be 0 since total is only 65 cents      if (valid == 0)        $display("PASS: VALID should be 0!!! We are depositing the quarter.");      else        $display("FAIL: valid expected 0 but got %b", valid);      // Deposit 25 cents (quarter) and 10 cents (dime) together      // Total should be 100 cents, so valid should now be 1      @(posedge clk);      quarter = 1;      dime = 1;      @(posedge clk);      quarter = 0;      dime = 0;      @(posedge clk);      // Check valid output, should be 1 now since total reached 100 cents      if (valid == 1)        $display("PASS: After final dime and quarter deposit, valid asserted as expected.");      else        $display("FAIL: After final dime and quarter deposit, valid expected 1 but got %b", valid);      #10;      // End of simulation      $display("Testbench completed.");      $stop;    end  endmodule |

# Quartus compilation report

## seqDetect compilation report

A screenshot of a computer

AI-generated content may be incorrect.

## vendingMachine compilation report

A screenshot of a computer

AI-generated content may be incorrect.

# RTL Netlist

## seqDetect.sv

A diagram of a circuit

AI-generated content may be incorrect.

## vendingMachine.sv

A diagram of a circuit

AI-generated content may be incorrect.