ELEX 7660: Digital System Design

Assignment 2

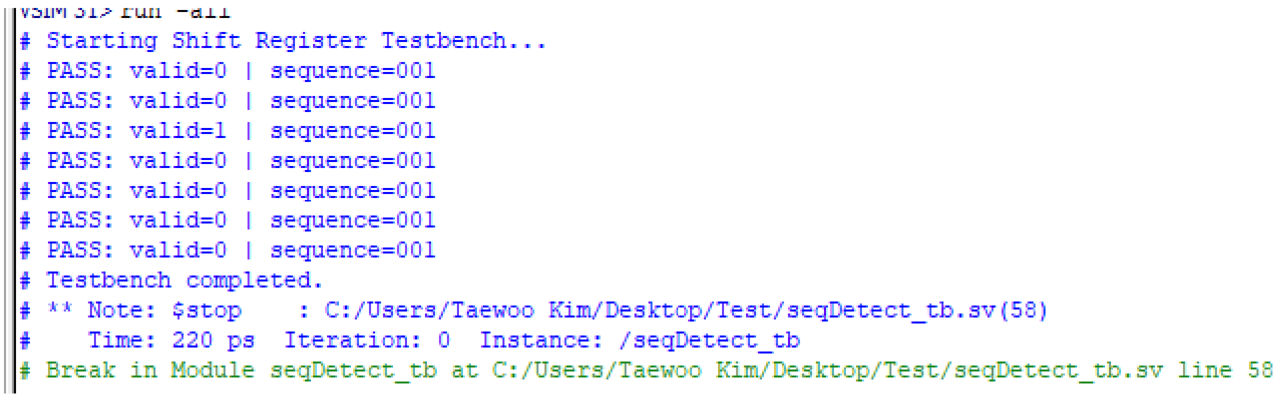
Student Name: Taewoo Kim	Student Number: A01284763	Set: T

Table of Contents

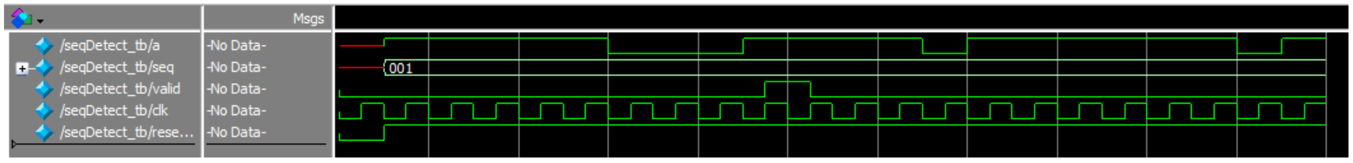
1 Screenshot of the simulations 3
1.1 problem 1 (seqDetect.sv) 3
1.2 problem 2 (vendingMachine.sv) 3
2 Source code of the module 3
2.1 seqDetect.sv 3
2.2 seqDetect_tb.sv 5
2.3 vendingMachine.sv 6
2.4 vendingMachine_tb.sv 7
3 Quartus compilation report 11
3.1 seqDetect compilation report 11
3.2 vendingMachine compilation report 12
4 RTL Netlist 12
4.1 seqDetect.sv 12
4.2 vendingMachine.sv 13

1 Screenshot of the simulations

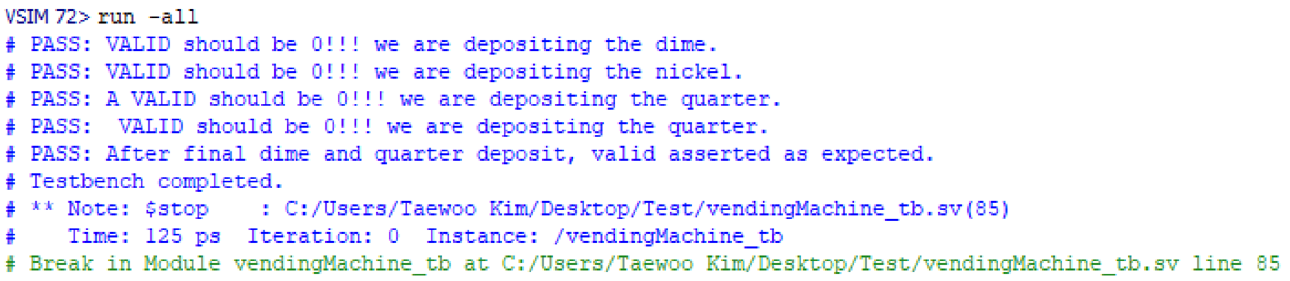
1.1 problem 1 (seqDetect.sv)



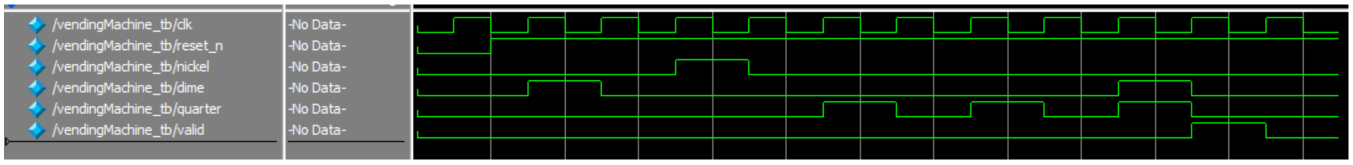
```
VSIM 31> run -all
# Starting Shift Register Testbench...
# PASS: valid=0 | sequence=001
# PASS: valid=0 | sequence=001
# PASS: valid=1 | sequence=001
# PASS: valid=0 | sequence=001
# PASS: valid=0 | sequence=001
# PASS: valid=0 | sequence=001
# PASS: valid=0 | sequence=001
# Testbench completed.
# ** Note: $stop    : C:/Users/Taewoo Kim/Desktop/Test/seqDetect_tb.sv(58)
#    Time: 220 ps  Iteration: 0  Instance: /seqDetect_tb
# Break in Module seqDetect_tb at C:/Users/Taewoo Kim/Desktop/Test/seqDetect_tb.sv line 58
```



1.2 problem 2 (vendingMachine.sv)



```
VSIM 72> run -all
# PASS: VALID should be 0!!! we are depositing the dime.
# PASS: VALID should be 0!!! we are depositing the nickel.
# PASS: A VALID should be 0!!! we are depositing the quarter.
# PASS:  VALID should be 0!!! we are depositing the quarter.
# PASS: After final dime and quarter deposit, valid asserted as expected.
# Testbench completed.
# ** Note: $stop    : C:/Users/Taewoo Kim/Desktop/Test/vendingMachine_tb.sv(85)
#    Time: 125 ps  Iteration: 0  Instance: /vendingMachine_tb
# Break in Module vendingMachine_tb at C:/Users/Taewoo Kim/Desktop/Test/vendingMachine_tb.sv line 85
```



2 Source code of the module

2.1 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
```

2.2 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
```

2.3 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
```

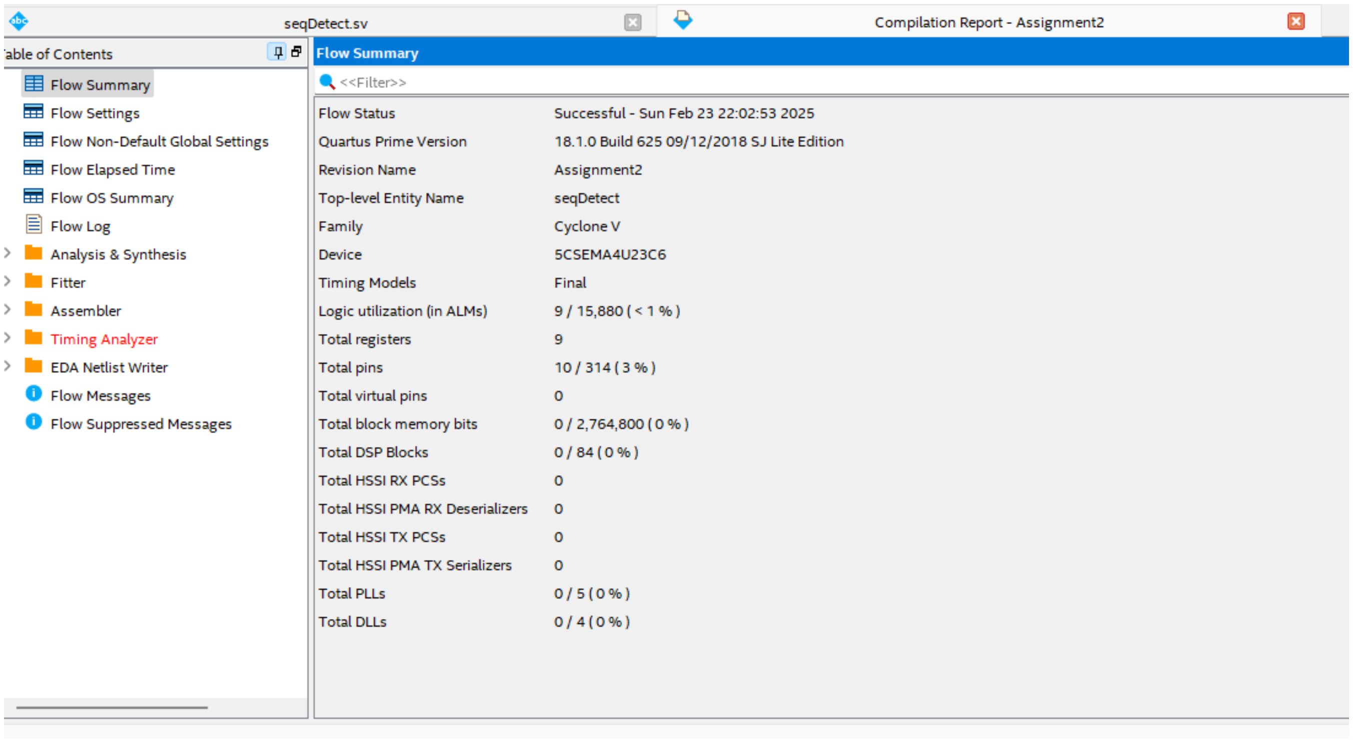
2.4 vendingMachine_tb.sv

```
// 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
```

3 Quartus compilation report

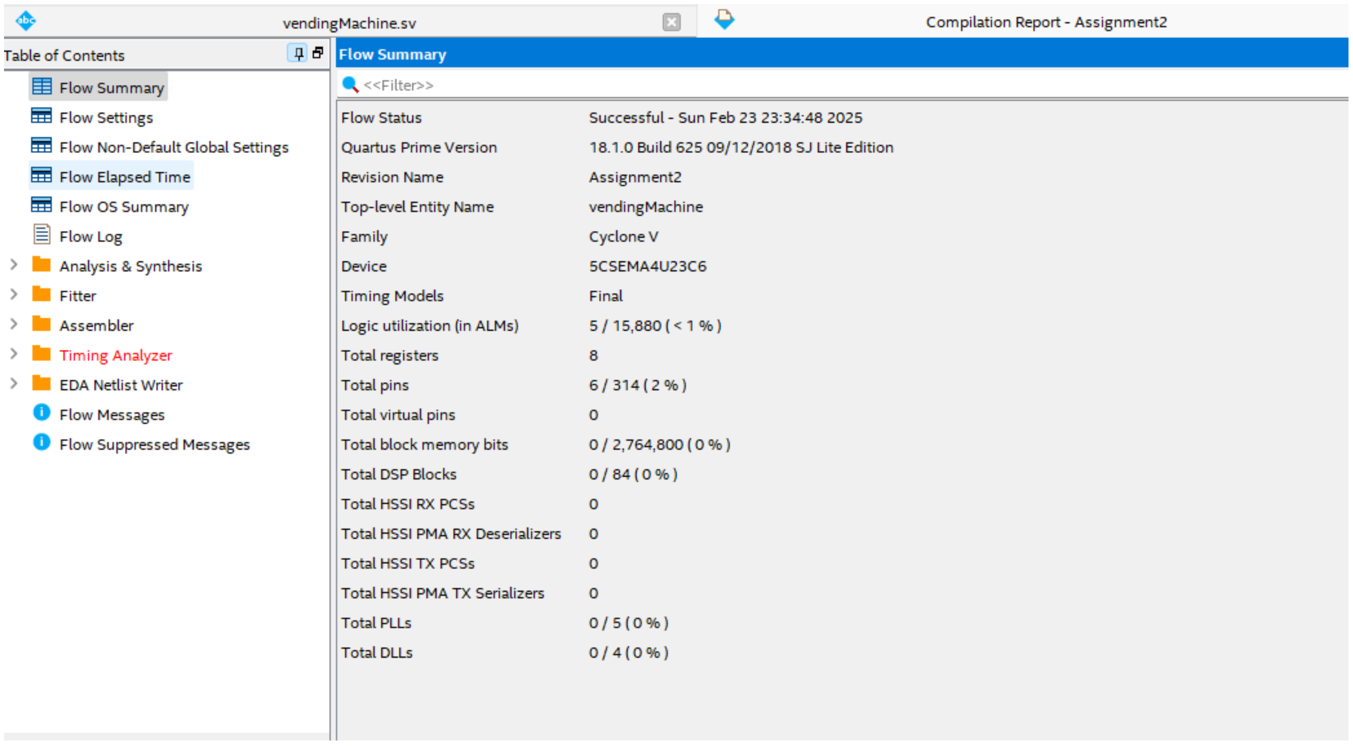
3.1 seqDetect compilation report



Flow Summary

Flow Status	Successful - Sun Feb 23 22:02:53 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Assignment2
Top-level Entity Name	seqDetect
Family	Cyclone V
Device	5CSEMA4U23C6
Timing Models	Final
Logic utilization (in ALMs)	9 / 15,880 (< 1 %)
Total registers	9
Total pins	10 / 314 (3 %)
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 vendingMachine compilation report

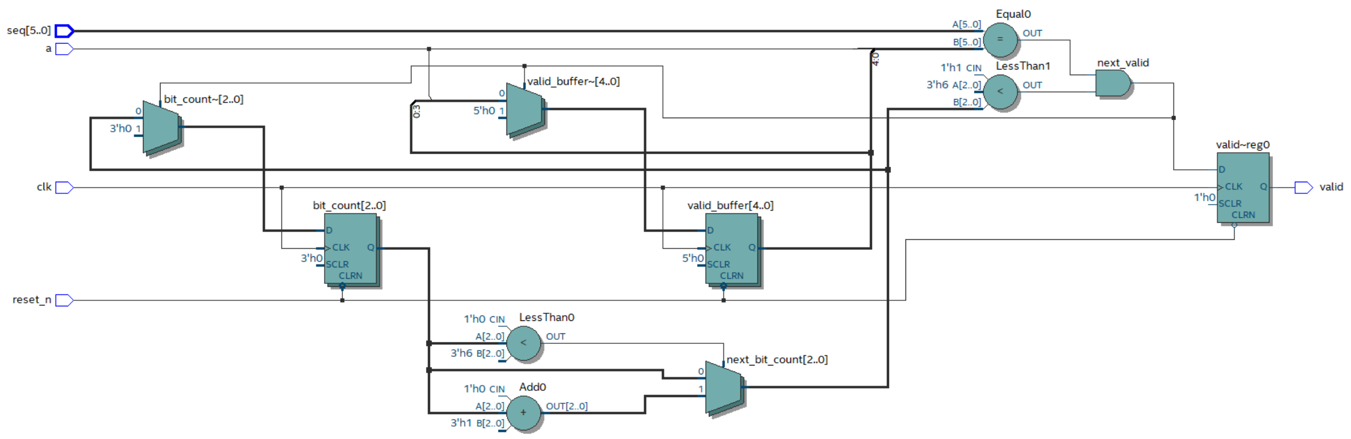


Flow Summary

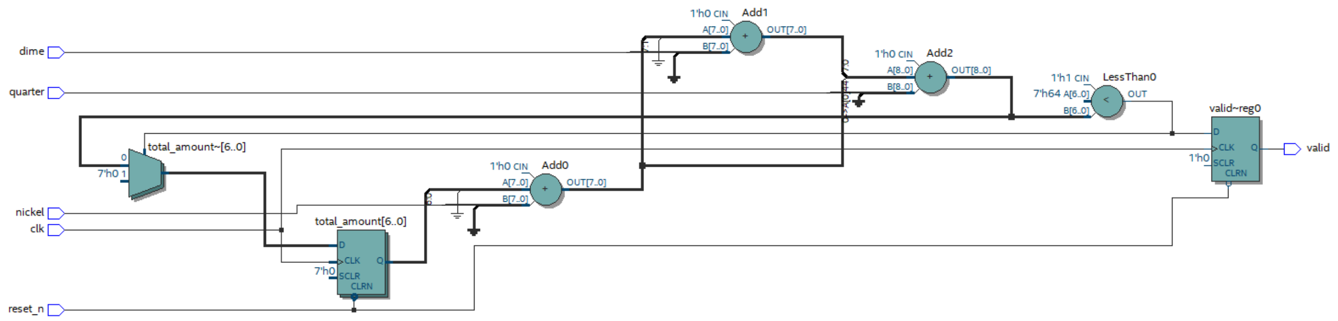
Flow Status	Successful - Sun Feb 23 23:34:48 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Assignment2
Top-level Entity Name	vendingMachine
Family	Cyclone V
Device	5CSEMA4U23C6
Timing Models	Final
Logic utilization (in ALMs)	5 / 15,880 (< 1 %)
Total registers	8
Total pins	6 / 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 seqDetect.sv



4.2 vendingMachine.sv


dime
quarter
nickel
clk
reset_n
valid
Add0
Add1
Add2
LessThan0
total_amount~[6..0]
total_amount[6..0]
valid~reg0
1'h0 CIN
1'h1 CIN
7'h64
7'h0
1'h0
A[7..0]
B[7..0]
OUT[7..0]
A[8..0]
B[8..0]
OUT[8..0]
A[6..0]
B[6..0]
OUT
D
CLK
SCLR
CLRN
Q