Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

3C7 DIGITAL SYSTEMS DESIGN LABORATORY

Assignment 02 Report

Department of Electronic and Electrical Engineering

Professor: Dr. Shreejith Shanker

1. Abstract:

The 3C7 Digital Systems Design assignment involves bringing together our knowledge from all the implemented previous labs and design a Finite State Machine to detect a pattern and give the output of the counter on Basys-3 Board. For an understanding of the development and testing of Finite State Machines in digital systems, this study synthesizes knowledge from earlier laboratories especially labs F and G. The LFSR should run for a complete cycle, i.e., 2^N-1 where N is the bit length of the LFSR. A given n-bit pattern is to be detected in the intermediate transition states of LFSR. The assignment emphasizes the importance of understanding and applying digital design principles in a practical context to implement a suitable FPGA design.

Submitted By:
Shreshtha Kamboj
23364317
Submission Date: 14/04/2024

2. Introduction:

This assignment's objectives were to gather all the knowledge from previous labs to design a Verilog module for LFSR and to implement sequence detecting Finite State Machine. Additionally, we were required to design a counter as well , in order to count the number of times a certain pattern is detected in the stream of bits generated by the LFSR in a full cycle of that LFSR. We do both testing it on the Basys 3 Board and running its behavioural simulation to draw certain conclusions by observing its output. We also discuss previous lab session's implementation in the appendices section in the end.

3. Implementation:

We implement a Moore Model Finite State Machine and use the logic's build in previous lab sessions for implementation in this assignment.

4. Sources:

The Hierarchy of the design consists of Testbench_assign02 module which inherits the properties of other instantiations of modules shown below:

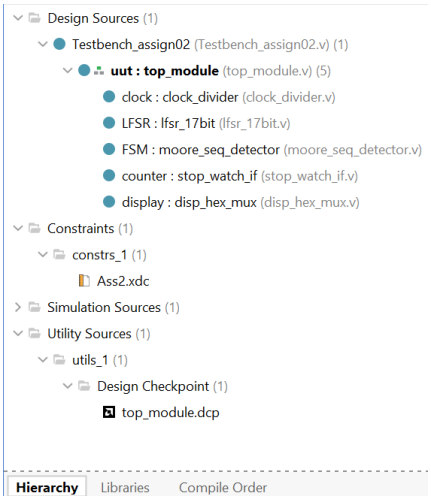


Fig 4.1 Design Sources

File directory is as follows:

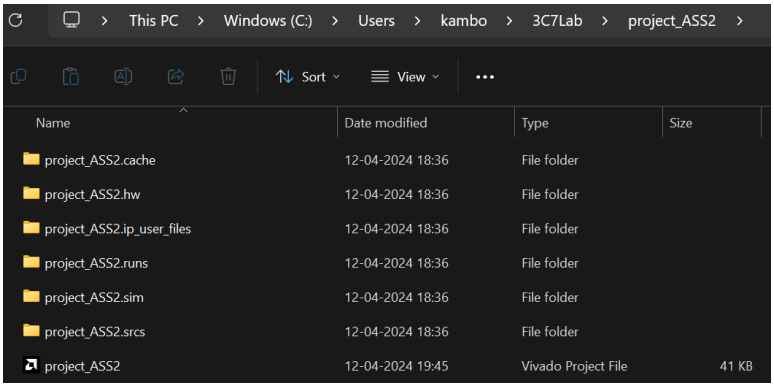


Fig 4.2 File Directory

5. Schematics Generated:

The below schematic serves as a Functional diagram clearly showing the hierarchy of our design and how all the modules are interconnected with each other.

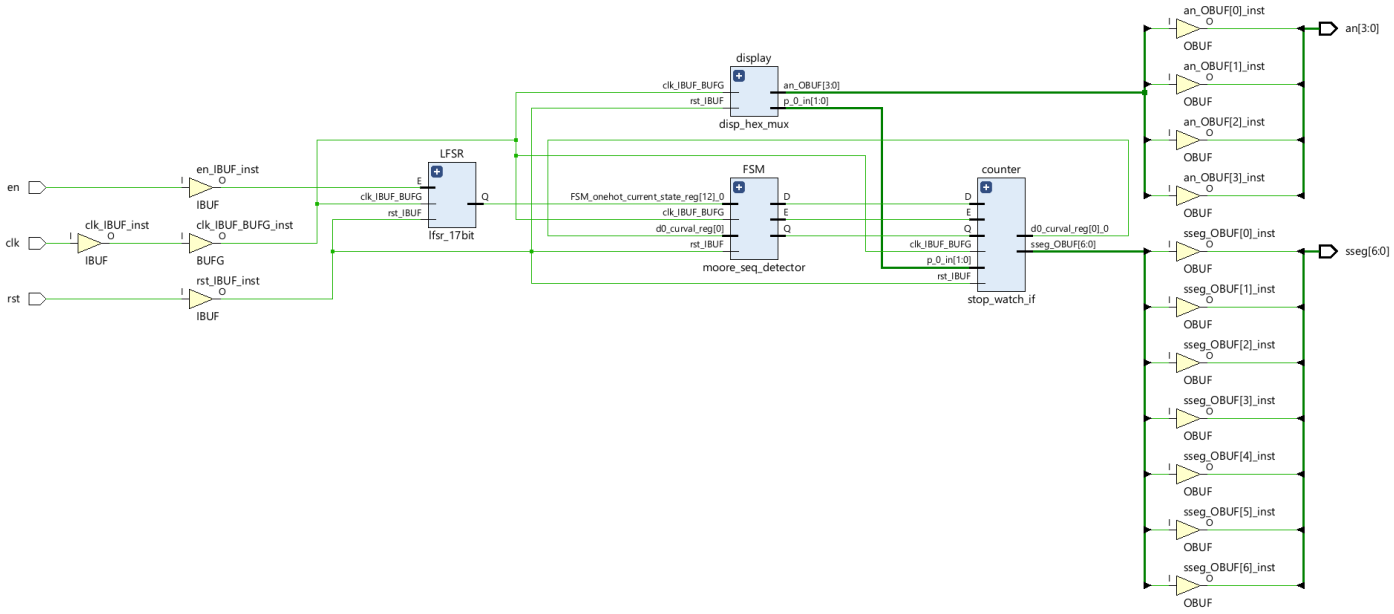


Fig 5.1 Elaborated Design

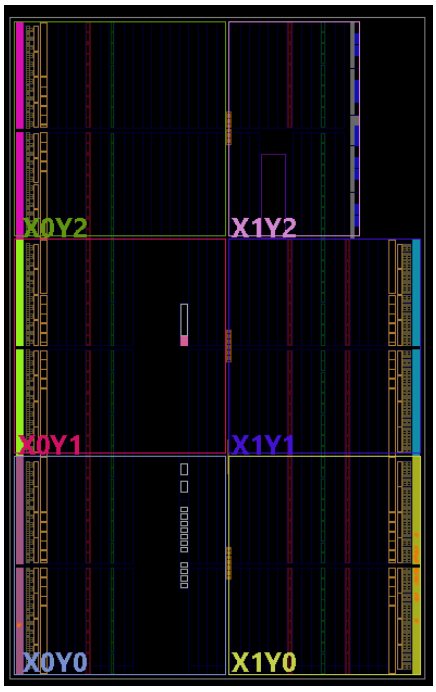


Fig 5.2 Implemented Device

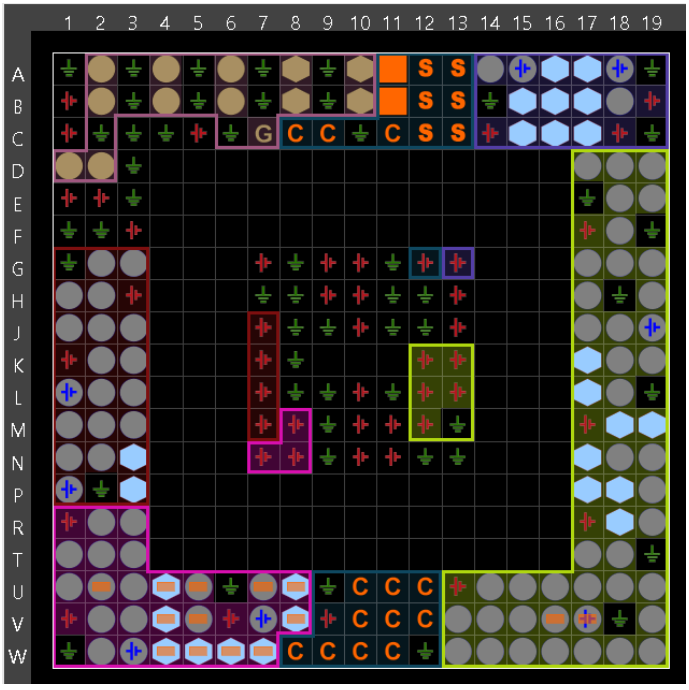


Fig 5.3 Implemented Package

6. Code Snippets:

We have included the following modules in this assignment to correctly implement the given task:-

- Testbench_assign02.v
- top_module.v
- lfsr_17bit.v
- stop_watch_if.v
- disp_hex_mux.v
- moore_seq_detector.v
- clock_divider.v
- Ass2.xdc

C:/Users/kambo/3C7Lab/project_ASS2/project_ASS2.srcs/sources_1/new/Testbench_assign02.v

```
module Testbench_assign02;
reg clk_tb;
reg rst_tb;
wire [3:0] an_tb;
wire [6:0] sseg_tb;
wire p;
wire [16:0] Q_d;
// Instantiate the top module
top_module uut(
    .clk(clk_tb),
    .rst(rst_tb),
    .an(an_tb),
    .pattern_detected(p),
    .sseg(sseg_tb),
    .Q_d(Q_d));
// Clock generation
always #10 clk_tb = ~clk_tb;  // Generate a clock with a period of 20ns
initial begin
    // Initialize signals
    clk_tb = 0;  // Initial state of the clock
    rst_tb = 1;  // Assert reset initially
    // Reset the system
    #20 rst_tb = 0;  // De-assert reset after 20ns
    #2621420;  // Run simulation for 2621420ns to observe multiple cycles
    $finish;  // End the simulation
end
endmodule
```

Fig 6.1 Testbench Module having test vectors

C:/Users/kambo/3C7Lab/project_ASS2/project_ASS2.srcs/sources_1/new/top_module.v

```
module top_module(
    input wire clk,
    input wire rst,
    input wire en,
    output wire [3:0] an,
    output wire [6:0] sseg);
//    output wire [16:0] Q_d);
wire lfsr_bit;
wire [3:0]  d3, d2, d1, d0;;
wire slow_clk;
wire pattern_detected;
// Instantiating the Clock Divider Module
clock_divider clock(
    .clk_in(clk),
    .rst_n(rst),
    .clk_out(slow_clk));
// Instantiating the LFSR Module
lfsr_17bit LFSR(
    .clk(clk),
    .rst_n(rst),
    .sh_en(en),
    .Q_out(lfsr_bit));
//    .Q_d(Q_d));
// Instantiating the Moore Model Finite State Machine
moore_seq_detector FSM(
    .clk(clk),
    .rst_n(rst),
    .check(lfsr_bit),
    .pattern_detected(pattern_detected));
// Instantiating the Counter Module
stop_watch_if counter(
    .clk(clk),
    .pattern_detected(pattern_detected),
    .clr(rst),
    .d3(d3),.d2(d2),.d1(d1),.d0(d0));
// Instantiating the Seven Segment Module
disp_hex_mux display(
    .clk(clk),
    .reset(rst),
    .hex3(d3), .hex2(d2), .hex1(d1), .hex0(d0),
    .dp_in(4'b1101),
    .an(an),
    .sseg(sseg)
);
endmodule
```

Fig 6.2 Top Module having instantiations of other modules

C:/Users/kambo/3C7Lab/project_ASS2/project_ASS2.srcs/sources_1/new/lfsr_17bit.v

```
module lfsr_17bit
    #(parameter seed = 17'b01000001011110001) // Seed parameter for LFSR initialization
    (input clk,          // Clock input signal
     input rst_n,        // Active low asynchronous reset signal
     input sh_en,        // Shift enable signal to control LFSR operation
     output wire Q_out  // MSB of LFSR
//     output wire [16:0] Q_d // Current State of LFSR bit by bit
    );
    reg [16:0] Q_state; // Register holding current state of the LFSR
    wire Q_fb;          // Wire holding feedback bit calculated
    wire [16:0] Q_ns;   // Wire holding next state of the LFSR
    // Sequential logic block to update the state of the LFSR on the rising edge of the clock
    always @(posedge clk or posedge rst_n) begin
        if(rst_n) // If reset is active (low), reset the LFSR state to the seed value
            Q_state <= seed;
        else if (sh_en) // If shift enable is active (high), update the LFSR state to the next state
            Q_state <= Q_ns;
    end
    assign Q_fb = ~(Q_state[16] ^ Q_state[13]); // Calculate feedback using XNOR where taps are at 17 and 14
    assign Q_ns = {Q_state[15:0], Q_fb}; // Form the next state by shifting left and inserting feedback bit
//    assign Q_d = Q_ns; // Output the next state of the LFSR
    assign Q_out = Q_ns[16]; // Output the most significant bit of the LFSR
endmodule
```

Fig 6.3 Module having logic for 17-bit LFSR

C:/Users/kambo/3C7Lab/project_ASS2/project_ASS2.srcs/sources_1/new/moore_seq_detector.v

```
module moore_seq_detector(
    input clk, // Clock input signal
    input rst_n, // Active low asynchronous reset signal
    input check, // Input signal to start checking the pattern
    output reg pattern_detected); // Output signal indicating when the pattern is detected
// State declaration with 4-bit encoding for a 12-bit sequence detection
localparam [3:0] A = 4'b0001, // Initial state
                 B = 4'b0010, // State for each bit in the sequence
                 C = 4'b0011,
                 D = 4'b0100,
                 E = 4'b0101,
                 F = 4'b0110,
                 G = 4'b0111,
                 H = 4'b1000,
                 I = 4'b1001,
                 J = 4'b1010,
                 K = 4'b1011,
                 L = 4'b1100,
                 M = 4'b1101; // Final state indicating the end of the sequence
reg [3:0] current_state, next_state; // Registers holding current and next state
// State transition logic, updates the current state on the rising edge of the clock
always @(posedge clk or posedge rst_n) begin
    if (rst_n) // If reset is active (low), reset to initial state A
        current_state <= A;
    else
        current_state <= next_state; // Otherwise, transition to the next state
end
// Next state logic based on the current state and input bit
always @(*) begin
    pattern_detected = 1'b0; // Default to not detected
    next_state = current_state; // Default to stay in the current state
    case (current_state)
        A: if (check == 0) next_state = B; // Transition logic for each state
            else next_state = A;
        B: if (check == 1) next_state = C;
            else next_state = B;
        C: if (check == 1) next_state = D;
            else next_state = B;
        D: if (check == 1) next_state = E;
            else next_state = B;
        E: if (check == 0) next_state = F;
            else next_state = A;
```

```
        F: if (check == 0) next_state = G;
            else next_state = A;
        G: if (check == 0) next_state = H;
            else next_state = A;
        H: if (check == 1) next_state = I;
            else next_state = B;
        I: if (check == 1) next_state = J;
            else next_state = B;
        J: if (check == 0) next_state = K;
            else next_state = A;
        K: if (check == 1) next_state = L;
            else next_state = B;
        L: if (check == 1) next_state = M;
            else next_state = B;
        M: begin
            next_state = A; // Return to the initial state after detecting the pattern
            pattern_detected = 1'b1; // Indicate that the pattern has been detected
            end
        default: next_state = A; // Default case to handle any undefined states
    endcase
end
endmodule
```

Fig 6.4 Module having code for Moore Model Finite State Machine

C:/Users/kambo/3C7Lab/project_ASS2/project_ASS2.srcs/sources_1/new/stop_watch_if.v

```
module stop_watch_if(
    input wire clk, // Clock input signal
    input wire pattern_detected, // Signal indicating a pattern has been detected
    input wire clr, // Clear signal to reset the counter
    output wire [3:0] d3, d2, d1, d0); // 4-digit BCD outputs
   // Declaration of registers to hold the current value of each BCD digit
   reg [3:0] d3_curval, d2_curval, d1_curval, d0_curval;
   // Declaration of registers to hold the next value of each BCD digit
   reg [3:0] d3_nextval, d2_nextval, d1_nextval, d0_nextval;
   // Register block to update the current value of each BCD digit on the rising edge of the clock
   always @(posedge clk)
   begin
      d3_curval <= d3_nextval;
      d2_curval <= d2_nextval;
      d1_curval <= d1_nextval;
      d0_curval <= d0_nextval;
   end
   // Combinational logic block to determine the next value of each BCD digit
   always @*
   begin
      // Default behavior: keep the previous value
      d0_nextval = d0_curval;
      d1_nextval = d1_curval;
       d3_nextval = d3_curval;
       // If the clear signal is asserted, reset all digits to 0
       if (clr)
          begin
             d0_nextval = 4'b0;
             d1_nextval = 4'b0;
             d2_nextval = 4'b0;
             d3_nextval = 4'b0;
          end
       // If a pattern is detected, increment the counter
       else if (pattern_detected)
          begin
               // Increment the least significant digit (d0) if it's not already 9
               if (d0_curval != 9)
                  d0_nextval <= d0_curval + 1;
               else
                  begin
                     // If d0 is 9, reset it to 0 and increment the next digit (d1)
                     d0_nextval = 0;
                     if (d1_curval != 9)
                        d1_nextval <= d1_curval + 1;
                     else
                        begin
```

```
                        // Applying the same logic for d1, d2, and d3
                        d1_nextval = 0;
                        if (d2_curval != 9)
                            d2_nextval <= d2_curval + 1;
                        else
                            begin
                                d2_nextval = 0;
                                if (d3_curval != 9)
                                    d3_nextval <= d3_curval + 1;
                                else
                                    // If all digits are 9, reset the counter to 0
                                    d3_nextval <= 0;
                            end
                    end
                end
            end
        end
        // Output logic: directly assign the current register values to the output
        assign d0 = d0_curval;
        assign d1 = d1_curval;
        assign d2 = d2_curval;
        assign d3 = d3_curval;
endmodule
```

Fig 6.5 Module having code for counter

C:/Users/kambo/3C7Lab/project_ASS2/project_ASS2.srcs/sources_1/new/disp_hex_mux.v

```
module disp_hex_mux(
    input wire clk, reset,
    input wire [3:0] hex3, hex2, hex1, hex0,  // Hexadecimal digits inputs
    input wire [3:0] dp_in,                   // Decimal points for each digit, active low
    output reg [3:0] an,                      // Active-low enable signals for each of the 4 digits
    output reg [7:0] sseg                     // Seven-segment display output, including decimal point
);
    localparam N = 18;  // Constant for counter size to achieve ~800 Hz refresh rate with a 50 MHz clock
    reg [N-1:0] q_reg = 0;  // N-bit counter register, initialized to 0
    wire [N-1:0] q_next;    // Next state of the counter
    reg [3:0] hex_in = 0;   // Current hex digit to display, initialized to 0
    reg dp = 1;             // Current decimal point state, initialized to 1 (off)
    // N-bit counter for multiplexing control
    always @(posedge clk or posedge reset)
        if (reset)
            q_reg <= 0;
        else
            q_reg <= q_next;
    assign q_next = q_reg + 1;
    // Multiplexing control based on the 2 MSBs of the counter
    always @*
        case (q_reg[N-1:N-2])
            2'b00: {an, hex_in, dp} = {4'b1110, hex0, dp_in[0]};
            2'b01: {an, hex_in, dp} = {4'b1101, hex1, dp_in[1]};
            2'b10: {an, hex_in, dp} = {4'b1011, hex2, dp_in[2]};
            default: {an, hex_in, dp} = {4'b0111, hex3, dp_in[3]};
        endcase
    // hex to seven-segment led display
    always @*
    begin
        case(hex_in)
            4'h0: sseg[6:0] = 7'b1000000;
            4'h1: sseg[6:0] = 7'b1111001;
            4'h2: sseg[6:0] = 7'b0100100;
            4'h3: sseg[6:0] = 7'b0110000;
            4'h4: sseg[6:0] = 7'b0011001;
            4'h5: sseg[6:0] = 7'b0010010;
            4'h6: sseg[6:0] = 7'b0000010;
            4'h7: sseg[6:0] = 7'b1111000;
            4'h8: sseg[6:0] = 7'b0000000;
            4'h9: sseg[6:0] = 7'b0010000;
            4'ha: sseg[6:0] = 7'b0001000;
            4'hb: sseg[6:0] = 7'b0000011;
            4'hc: sseg[6:0] = 7'b1000110;
            4'hd: sseg[6:0] = 7'b0100001;
            4'he: sseg[6:0] = 7'b0000110;
            4'hf: sseg[6:0] = 7'b0001110;
            default: sseg[6:0] = 7'b1111111;
        endcase
        sseg[7] = dp; // Corrected to properly handle active-low logic for decimal point
    end
endmodule
```

Fig 6.6 Module having code for display logic on the 7-Segment Display

C:/Users/kambo/3C7Lab/Lab F/project_LabFpartD/project_LabFpartD.srcs/sources_1/new/clock_divider.v

```
module clock_divider(
    input clk_in,  // Input clock, assumed to be 50MHz based on the context
    input rst_n,   // Asynchronous reset, active low
    output reg clk_out = 0);  // Output clock, initialized to 0, target is 1Hz after division
// Parameter for defining the division factor. Set to 25,000,000 for a 50MHz clock to achieve 1Hz.
parameter DIVIDE_BY = 25000000; // Division factor to toggle the output clock
// Define a 25-bit counter to count up to 25,000,000
reg [24:0] counter = 0; // Counter variable to store intermediate counts
// Always block triggered on the rising edge of clk_in or the falling edge of rst_n
always @(posedge clk_in or posedge rst_n) begin
    if (rst_n) begin // If reset is active (low)
        counter <= 0;      // Reset the counter to 0
        clk_out <= 0;      // Reset the output clock to 0
    end else begin
        if (counter == DIVIDE_BY-1) begin // If counter reaches the division value minus 1
            counter <= 0;                 // Reset the counter
            clk_out <= ~clk_out;          // Toggle the output clock
        end else begin
            counter <= counter + 1;       // Increment the counter
        end
    end
end
endmodule
```

Fig 6.7 Module having code for Clock Divider

The following image shows the constraints for Basys 3 board implementation. The implementation will be explained in the demonstration section.

C:/Users/kambo/3C7Lab/project_ASS2/project_ASS2.srcs/constrs_1/new/Ass2.xdc

```
### This file is a general .xdc for the Basys3 rev B board
### To use it in a project:
### - uncomment the lines corresponding to used pins
### - rename the used ports (in each line, after get_ports) according to the top

### Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

# Switches
set_property PACKAGE_PIN V17 [get_ports {rst}]
set_property IOSTANDARD LVCMOS33 [get_ports {rst}]
set_property PACKAGE_PIN V16 [get_ports {en}]
set_property IOSTANDARD LVCMOS33 [get_ports {en}]

#7 segment display
set_property PACKAGE_PIN W7 [get_ports {sseg[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sseg[0]}]
set_property PACKAGE_PIN W6 [get_ports {sseg[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sseg[1]}]
set_property PACKAGE_PIN U8 [get_ports {sseg[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sseg[2]}]
set_property PACKAGE_PIN V8 [get_ports {sseg[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sseg[3]}]
set_property PACKAGE_PIN U5 [get_ports {sseg[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sseg[4]}]
set_property PACKAGE_PIN V5 [get_ports {sseg[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sseg[5]}]
set_property PACKAGE_PIN U7 [get_ports {sseg[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sseg[6]}]

set_property PACKAGE_PIN V7 [get_ports {dp}]
set_property IOSTANDARD LVCMOS33 [get_ports {dp}]

set_property PACKAGE_PIN U2 [get_ports {an[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]
```

Fig 6.8 Constraints File for Basys 3 Board

7. Demonstration

Configuration Device: BASYS 3 (xc7a35tcpg236-1)
Target Language: Verilog

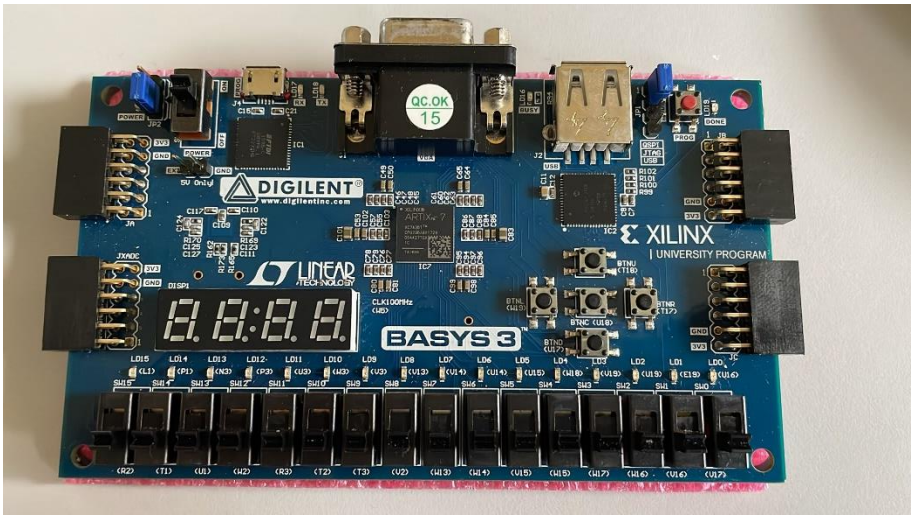


Fig 7.1 Basys 3 Board

A Finite State Machine is as the name suggests a system using a finite number of states and the associated transitions between them. In this case we have a rising edge sensitive system which may transition to the next state at the next rising edge of the clock. The transition depends on the input at that rising edge and it stays in that state until the next rising edge is detected. They are mostly used as a pattern/sequence detector. In the construction of an FSM, we need a register to store the current state and a combinational logic to determine the next state and output.

Mealy Model Finite State Machine and Moore Model Finite State Machine are two models of FSMs commonly used. This assignment's requirement was to implement a Moore Model Finite State Machine. The difference between the two lies in the state transition logic. Mealy Model takes both the current input and the current state in consideration when transitioning on to the next state whereas the Moore Model solely takes into consideration the current state when transitioning onto the next state.

We have also used the State Encoding logic to represent the states as binary values. In this case we had N=13 States so we will need at least $\log_2 N$ bits to encode them.

We have the inputs coming from the 17-bit LFSR in which the bits are shifted to left and at the least significant bit position the feedback of LFSR keeps on adding at each transition and the most significant bit in this case which is the 16^{th} bit is given as an input to the finite state machine for detecting the sequence itself.

LFSR Seed Value (17-bit): 01000001011110001

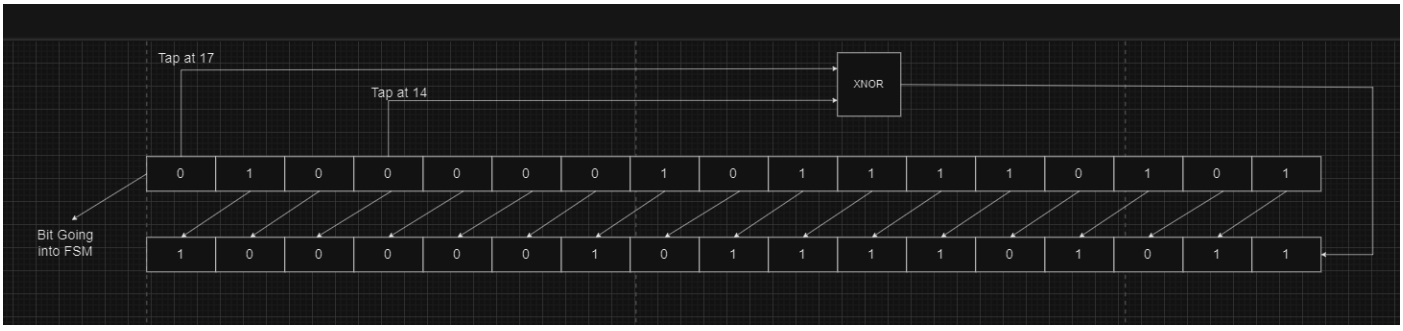


Fig 7.2 17-bit LFSR

Given below is a Moore Model Finite State Machine designed to detect a 12-bit pattern/sequence. Following the FSM design we have also designed to represent the transition logic in a tabular form.

Pattern: 110110001110 (We detect the pattern starting from the right most bit which is the LSB to the left most bit which is the MSB.)

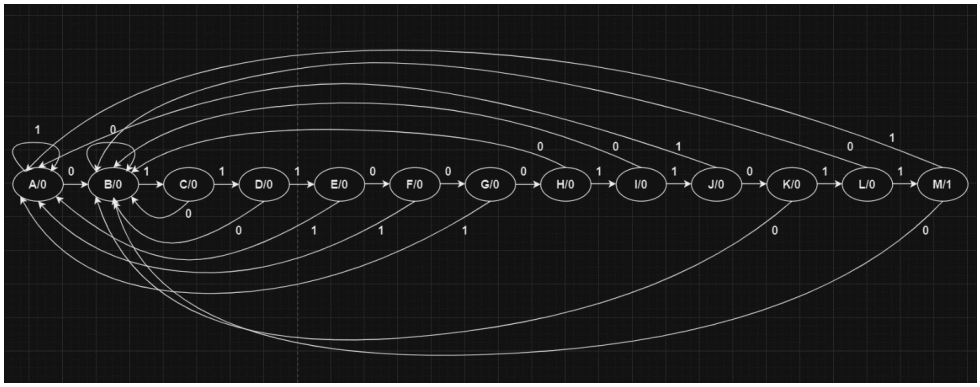


Fig 7.3 Moore Model Finite State Machine

In the below table the minimisation strategy cannot be applied as we do not have any identical states.

Current State	Input (Pattern)	Input (Otherwise)	Next State (Desired)	Next State (Otherwise)	Output
A	0	1	B	A	0
B	1	0	C	B	0
C	1	0	D	B	0
D	1	0	E	B	0
E	0	1	F	A	0
F	0	1	G	A	0
G	0	1	H	A	0
H	1	0	I	B	0
I	1	0	J	B	0
J	0	1	K	A	0
K	1	0	L	B	0
L	1	0	M	B	1
M	1	0	A	B	0

Table 1 State Transition Table

We also designed a testbench module to clearly see the output of LFSR and the output of FSM.

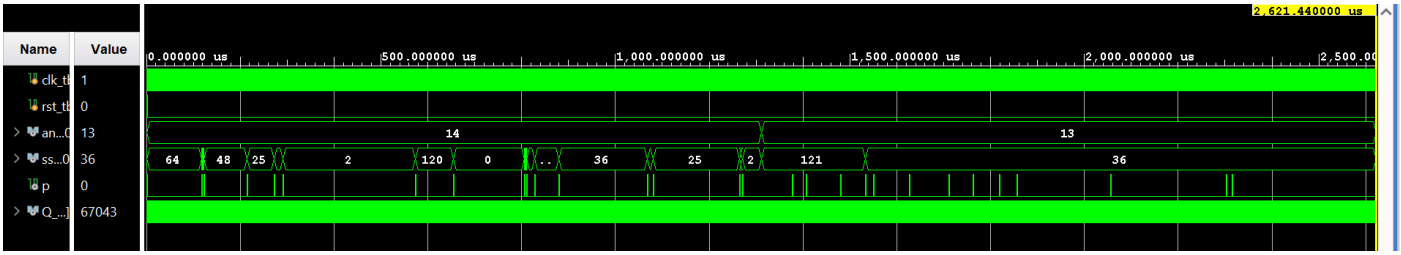


Fig 7.4 Behavioural simulation output on Vivado

The above waveform shows the output of FSM. We have the clock signal (clk), reset signal (rst), and seven segment signals (sseg). Whenever the reset is '0' and the shift enable is set to '1', the LFSR keeps on transitioning onto the next state. The green line in the Q_d output is showing how the output keeps on changing when we move the cursor on that line. The most important output signal to observe here is the pattern detecting one (p), whenever the pattern is detected, we see a green spike on the waveform.

We have used the formula $2^N - 1$ to calculate the total number of possible states in a complete cycle which in this case is $2^{17} - 1 = 131071$. The clock period is taken as 20ns. We have run the above behavioural simulation for 2621420ns (Total states x clock period) so that we can observe the number of times pattern is being detected in one complete LFSR cycle. If we count then we can see that a total of 28 times the pattern is being detected in the above LFSR in 1 complete cycle.

We also tried to target the above design on the BASYS 3 Board by instantiating the modules in a top module. The counter logic in the stop_watch_if.v is written such that whenever the pattern is detected then it increments the counter by 1 which is to be reflected on the 7-segment display. The logic is that whenever the count at a place is reached to maximum value, i.e., 9 then we shift on to the next place on the 7-segment. Despite all the efforts and time devoted we were not able to get it working as desired.

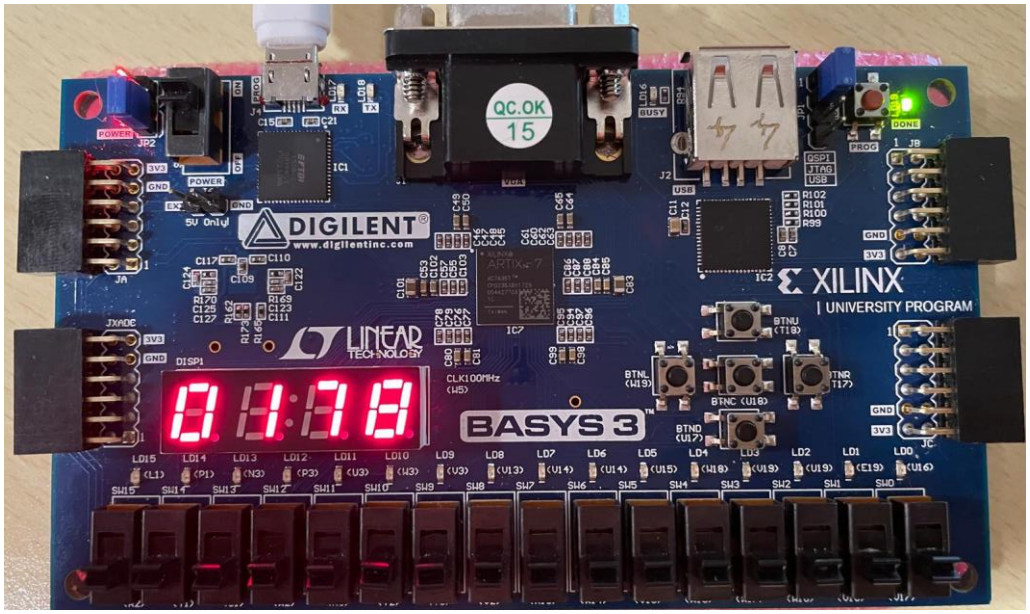


Fig 7.5 Output on Basys 3 Board

The above image shows the successful implementation of the assignment on BASYS 3 Board. When we set the switch (V16) to '1', then we enable the shift so that the LFSR can transition onto the next state and when switch is set to '0', then we stop at a transition state of the LFSR and see the current count of pattern detections as seen above. When the switch (V17) is set to '1', then the LFSR is reset to its seed value.

8. Observations:

Utilization Report, which is produced following the synthesis and implementation phases, provides information on how well our design is utilizing the FPGA's resources. Here we observe that IO resource is utilizing the most percentage of resources, i.e., 13% .

Resource	Utilization	Available	Utilization %
LUT	42	20800	0.20
FF	64	41600	0.15
IO	14	106	13.21

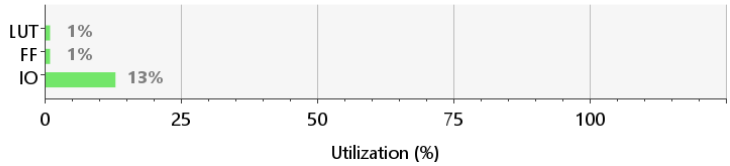


Fig 8.1 Summary of Utilization Report

Power Consumption Report provides information about the target FPGA device's power consumption profile throughout the execution of our design. Here we observe that the total on-chip power is 0.080W and the Junction Temperature is 25.4℃ which tells the designer reliability of their FPGA designs and making thermal management strategies to ensure safe operation of device. The figure on right tells us that which parameter utilizes how much of the chip power like in this case I/O utilizes the most power amongst Signals, Logic, and I/O.

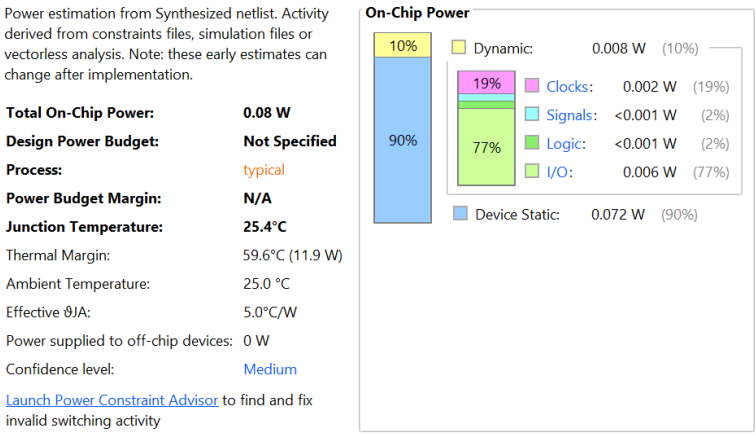


Fig 8.2 Summary of Power Report

Design Timing Report provides insights into our design's timing behaviour, including clock limitations, timing violations, and key routes. For a designer, this is the most crucial report because it is only via analysis of this report that he can determine whether he is fulfilling his timing goals. Here we see that all the specified timing constraints are met. We specified the clock period as 20ns in the XDC file.

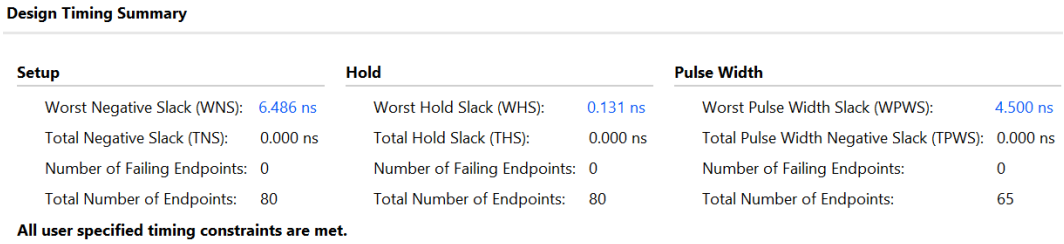


Fig 8.3 Summary of Design Timing

Register as Flip Flop section in the utilization summary tells us how many flip flops were utilized. Here we can see how many flip flops are used by each module.

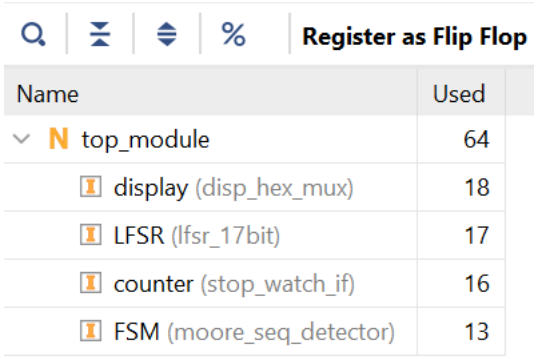
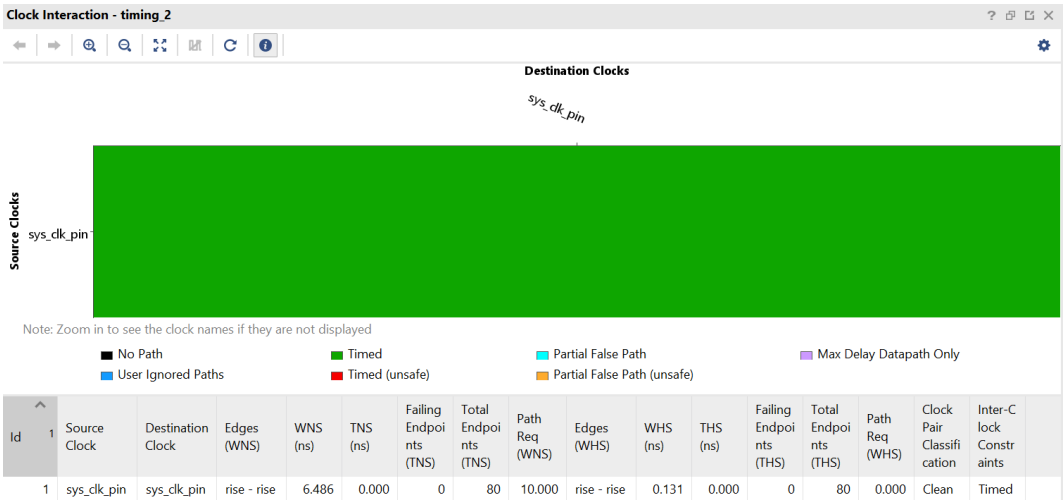


Fig 8.4 Summary of Register as Flip Flop

Clock interaction Report is a useful tool for comprehending the complexity of clock distribution and synchronization within our FPGA design, spotting possible timing problems associated with clock interactions, and putting strategies into place to guarantee correct synchronization and timing closure across various clock domains.



Id	Source Clock	Destination Clock	Edges (WNS)	WNS (ns)	TNS (ns)	Failing Endpoints (TNS)	Total Endpoints (TNS)	Path Req (WNS)	Edges (WHS)	WHS (ns)	THS (ns)	Failing Endpoints (THS)	Total Endpoints (THS)	Path Req (WHS)	Clock Pair Classification	Inter-Clock Constraints
1	sys_clk_pin	sys_clk_pin	rise - rise	6.486	0.000	0	80	10.000	rise - rise	0.131	0.000	0	80	0.000	Clean	Timed

Fig 8.5 Summary of Clock Interaction

9. Appendix:

9.1 LAB F:

In this lab session we familiarized with the concept of LFSR and designed a 13-bit LFSR. We targeted the design on the BASYS 3 Board so that we could observe the state transitions using the LEDS which gave us the idea of how the LFSR was working. We used the knowledge we gained from it and implemented the 17-bit LFSR in this assignment. We also used the clock_divider.v module to as required in this assignment.

9.2 LAB G:

In this lab session we familiarized with seven segment display. We learnt how to work with the seven segments by writing a logic. We were given two files stop_watch_if.v and disp_hex_mux.v which has logic for the counter and the 7-segment display respectively. We designed a logic for implementing a stop watch such that after 59 the next number should be 100 that indicates how the clock works. We designed two logic's, one for incrementing the counter when the up button is pressed and decrementing the counter when the down button is pressed. We used the knowledge we gained from this lab in the current assignment and changed the logic accordingly to fulfil the requirements of this lab.

10. Conclusion:

The 3C7 Digital Systems Design assignment successfully demonstrated the design, testing, and implementation of a Moore Model Finite State Machine on the BASYS 3 Board and were also able to clearly derive results from the behavioural simulation, synthesizing knowledge from previous lab experiences. The generated reports provided insights into the performance of the design, identifying opportunities for future improvements and optimization. These reports included the usage, power consumption, design timing, clock interaction and register as flip-flop reports. This assignment also marks the end of the practical applications for this module.

11. References:

- Lecture Slides
- Lab materials provided on blackboard
- Basic knowledge of electronics from previous modules
- Doubt clearing from the demonstrators
- Xilinx notes to work with LFSR
- Documentation for 7-Segment Display