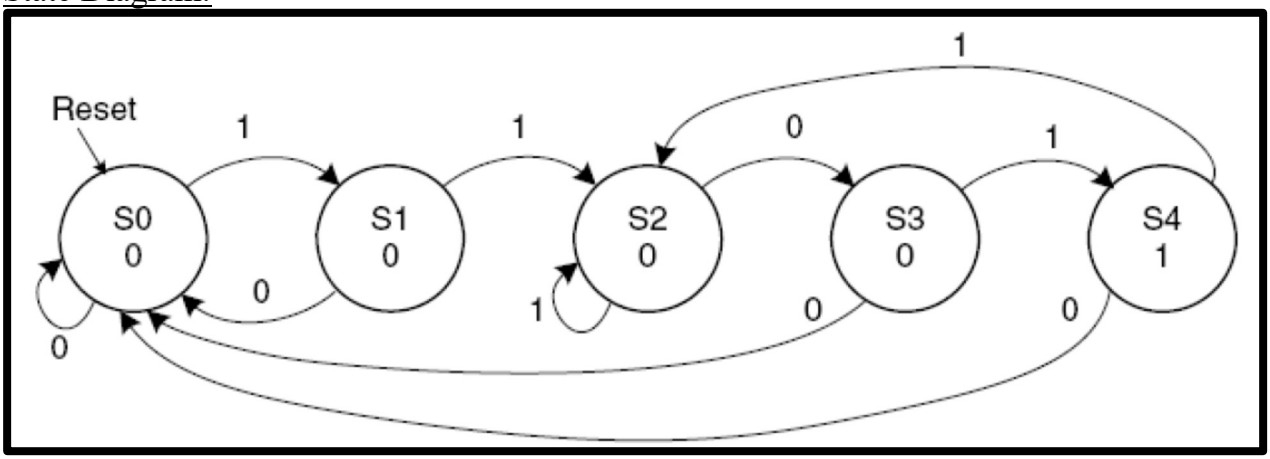
Switching Theory Lab – CS226

Name: M. Maheeth Reddy **Roll No.**: 1801CS31 **Date**: 09-May-2020

Lab No.: 14

Ans 1:

State Diagram:



Source Code:

```
// FSM to detect the pattern 1101
module p1(Y,A,clk);
    // A is input bit
    input A,clk;
    // clk is Clock
    output reg Y;   // Y is the output
    // Y is high when the last four input bits are 1101

    reg [2:0] state = 3'b000;   // State Register
    wire s2 = state[2];
    wire s1 = state[1];
    wire s0 = state[0];

    always @(posedge clk) begin
        // Combinational Logic for output
        assign Y = state[2];

        // Combinational Logic for Next State
        state = {(s1 & s0 & A),                                  // state[0]
                 (~s1 & s0 & A) + (s1 & ~s0) + (s2 & A),         // state[1]
                 (~s2 & ~s1 & ~s0 & A) + (s1 & ~s0 & ~A)};       // state[2]
    end
endmodule
```

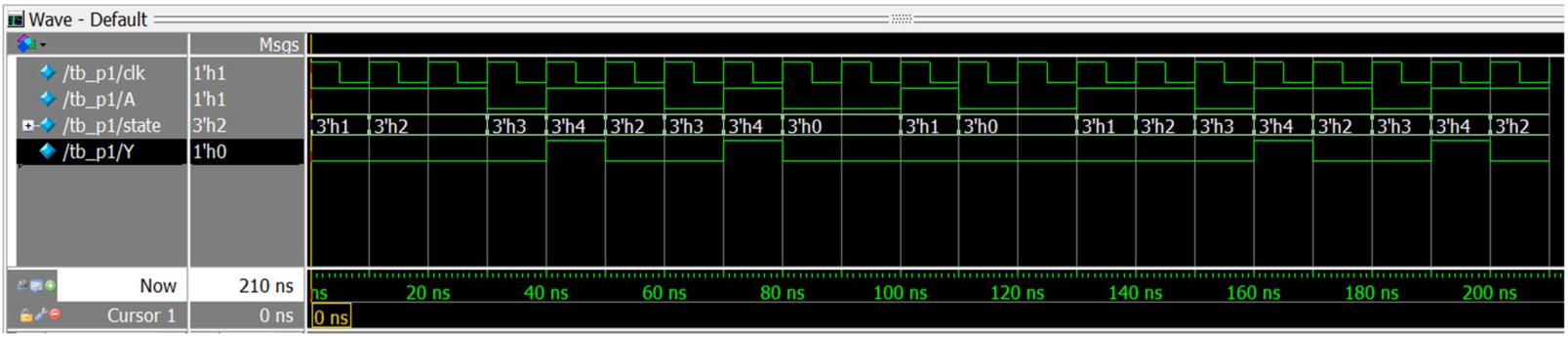
Testbench: (Test Sequence: **11101101001001101101**)

```
// Testbench for module p1 in p1.v
module tb_p1();
    // Mode Declaration
    reg A,clk;  // Inputs
    wire Y;     // Outputs

    p1 UUT (Y,A,clk);   // Instantiate p1

    wire [2:0] state = UUT.state;

    // Test Sequence: 11101101001001101101
    initial begin
        clk = 1;

        A = 1'b1;  #30;
        A = 1'b0;  #10;
        A = 1'b1;  #20;
        A = 1'b0;  #10;
        A = 1'b1;  #10;
        A = 1'b0;  #20;
        A = 1'b1;  #10;
        A = 1'b0;  #20;
        A = 1'b1;  #20;
        A = 1'b0;  #10;
        A = 1'b1;  #20;
        A = 1'b0;  #10;
        A = 1'b1;  #10;
    end

    // 1 Clock Cycle = 10ns
    always #5 clk = ~clk;

    // Display all parameters
    initial begin
        $monitor("time=%g\tclk=%b A=%b state=%b Y=%b",$time,clk,A,state,Y);
    end
endmodule
```

Text Output:

```
VSIM 12> run
# time=0 clk=1 A=1 state=001 Y=0
# time=5 clk=0 A=1 state=001 Y=0
# time=10 clk=1 A=1 state=010 Y=0
# time=15 clk=0 A=1 state=010 Y=0
# time=20 clk=1 A=1 state=010 Y=0
# time=25 clk=0 A=1 state=010 Y=0
# time=30 clk=1 A=0 state=011 Y=0
# time=35 clk=0 A=0 state=011 Y=0
# time=40 clk=1 A=1 state=100 Y=1
# time=45 clk=0 A=1 state=100 Y=1
# time=50 clk=1 A=1 state=010 Y=0
# time=55 clk=0 A=1 state=010 Y=0
# time=60 clk=1 A=0 state=011 Y=0
# time=65 clk=0 A=0 state=011 Y=0
# time=70 clk=1 A=1 state=100 Y=1
# time=75 clk=0 A=1 state=100 Y=1
# time=80 clk=1 A=0 state=000 Y=0
# time=85 clk=0 A=0 state=000 Y=0
# time=90 clk=1 A=0 state=000 Y=0
# time=95 clk=0 A=0 state=000 Y=0
# time=100 clk=1 A=1 state=001 Y=0
# time=105 clk=0 A=1 state=001 Y=0
# time=110 clk=1 A=0 state=000 Y=0
# time=115 clk=0 A=0 state=000 Y=0
# time=120 clk=1 A=0 state=000 Y=0
# time=125 clk=0 A=0 state=000 Y=0
# time=130 clk=1 A=1 state=001 Y=0
# time=135 clk=0 A=1 state=001 Y=0
# time=140 clk=1 A=1 state=010 Y=0
# time=145 clk=0 A=1 state=010 Y=0
# time=150 clk=1 A=0 state=011 Y=0
# time=155 clk=0 A=0 state=011 Y=0
# time=160 clk=1 A=1 state=100 Y=1
# time=165 clk=0 A=1 state=100 Y=1
# time=170 clk=1 A=1 state=010 Y=0
# time=175 clk=0 A=1 state=010 Y=0
# time=180 clk=1 A=0 state=011 Y=0
# time=185 clk=0 A=0 state=011 Y=0
# time=190 clk=1 A=1 state=100 Y=1
# time=195 clk=0 A=1 state=100 Y=1
# time=200 clk=1 A=1 state=010 Y=0
# time=205 clk=0 A=1 state=010 Y=0
```

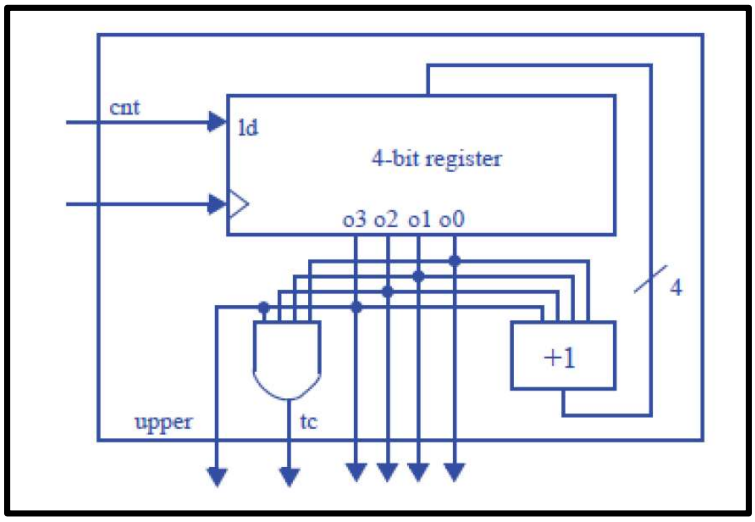
Simulation Waveform:



Simulation Time: **210ns** 1 Clock Cycle = **10ns**

Ans 2:

Architecture:



Source Code:

```
// 4-bit Up Counter with
// additional output 'upper'
module p2(count,upper,clk);
    // 4-bit register for counting
    output reg [3:0] count = 4'b0;
    output upper;       // output -> upper
    input clk;          // Clock

    always @(posedge clk)
        // At rising edge of clock
        // Increase count by 1
```

Text Output:

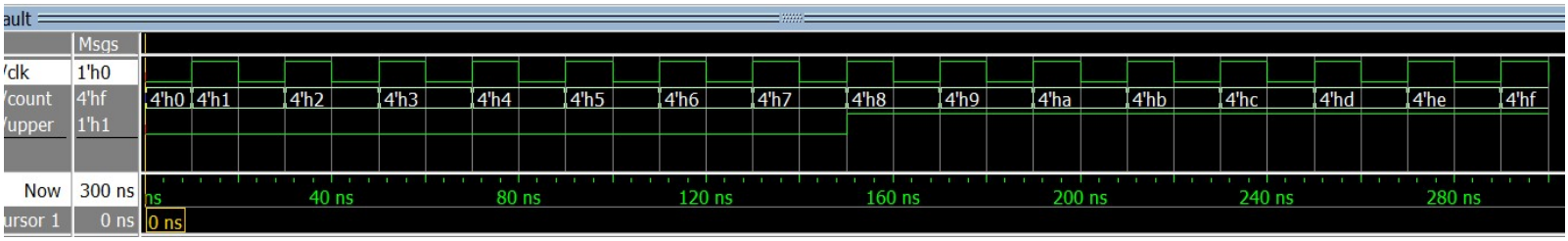
```
VSIM 20> run
# time=0 clk=0 count=0 upper=0
# time=10 clk=1 count=1 upper=0
# time=20 clk=0 count=1 upper=0
# time=30 clk=1 count=2 upper=0
# time=40 clk=0 count=2 upper=0
# time=50 clk=1 count=3 upper=0
# time=60 clk=0 count=3 upper=0
# time=70 clk=1 count=4 upper=0
# time=80 clk=0 count=4 upper=0
# time=90 clk=1 count=5 upper=0
# time=100 clk=0 count=5 upper=0
# time=110 clk=1 count=6 upper=0
# time=120 clk=0 count=6 upper=0
# time=130 clk=1 count=7 upper=0
# time=140 clk=0 count=7 upper=0
# time=150 clk=1 count=8 upper=1
# time=160 clk=0 count=8 upper=1
# time=170 clk=1 count=9 upper=1
# time=180 clk=0 count=9 upper=1
# time=190 clk=1 count=a upper=1
# time=200 clk=0 count=a upper=1
# time=210 clk=1 count=b upper=1
# time=220 clk=0 count=b upper=1
# time=230 clk=1 count=c upper=1
# time=240 clk=0 count=c upper=1
# time=250 clk=1 count=d upper=1
# time=260 clk=0 count=d upper=1
# time=270 clk=1 count=e upper=1
# time=280 clk=0 count=e upper=1
# time=290 clk=1 count=f upper=1
```

```
        count = count + 1;

        // Upper is High when count is
        // within upper half of counter range
        // i.e., 8-15
        assign upper = count[3];
endmodule
```

Testbench:

```
// Testbench for p2 in p2.v
module tb_p2();
    // Mode Declaration
    reg clk = 1'b0;     // Clock
    wire [3:0] count;   // Counter value
    wire upper;         // output 'upper'

    p2 UUT (count,upper,clk);   // Instantiate p2

    // 1 Clock Cycle = 20ns
    always #10 clk = ~clk;

    // Display all parameters
    initial begin
        $monitor("time=%g\tclk=%b count=%h upper=%b",$time,clk,count,upper);
    end
endmodule
```

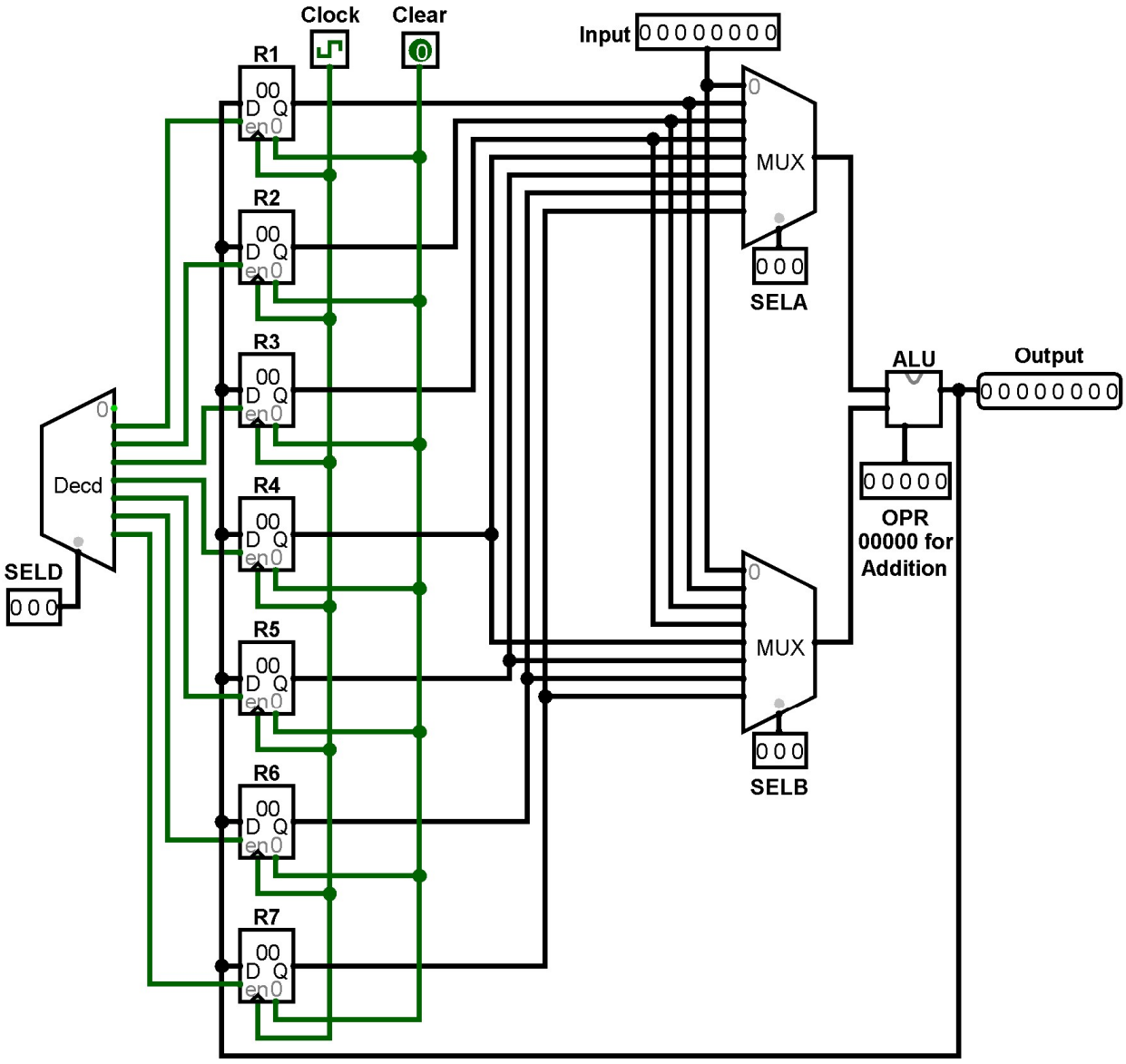
Simulation Waveform:



Simulation Time: **300ns** 1 Clock Cycle = **20ns**

Ans 3:

(a) Simulate the block given in logic-sim



Steps to Operate:

1. Give inputs for pins Input, SELA, SELB, SELD, OPR.
2. Toggle Clock. Output is seen at the Output pin.

Ans 3:
(b) Verilog model and a test bench to simulate the behavior block behavior

Source Code:

```
module p3_b (Output, Input, SELA, SELB, SELD, OPR, Clock, Clear);
    input [7:0] Input;              // Input Line
    input [2:0] SELA, SELB, SELD;   // Select Lines
    input [4:0] OPR;                // Operation Code
    input Clock;                    // Clock
    input Clear;                    // Clear data from registers
    output [7:0] Output;            // Output Line

    reg [7:0] R [7:1];              // Registers 1-7
    wire [7:0] A_bus, B_bus;        // A bus and B bus

    // Instantiate MUX and ALU submodules
    MUX8_1 muxA (A_bus, Input, R[1], R[2], R[3], R[4], R[5], R[6], R[7], SELA);
    MUX8_1 muxB (B_bus, Input, R[1], R[2], R[3], R[4], R[5], R[6], R[7], SELB);
    ALU     alu (Output, A_bus, B_bus, OPR);

    // Initialize all Register Data to Zero
    integer i,j;
    initial begin
        for (i=1; i<=7; i=i+1) begin
            R[i] = 8'b0;
        end
    end

    // Store Output in Registers
    always @(posedge Clock) begin
        // if Clear = 1 Erase Data from all Registers
        if (Clear) begin
            for (j=1; j<=7; j=j+1) begin
                R[j] = 8'b0;
            end
        end
        else R[SELD] = Output;
        // Store in Register selected by SELD
    end
endmodule
```

Testbench:

```
// Testbench for p3_b in p3.v
module tb_p3_b();
    // Mode Declaration
    reg [7:0] Input;
    reg [2:0] SELA, SELB, SELD;
```

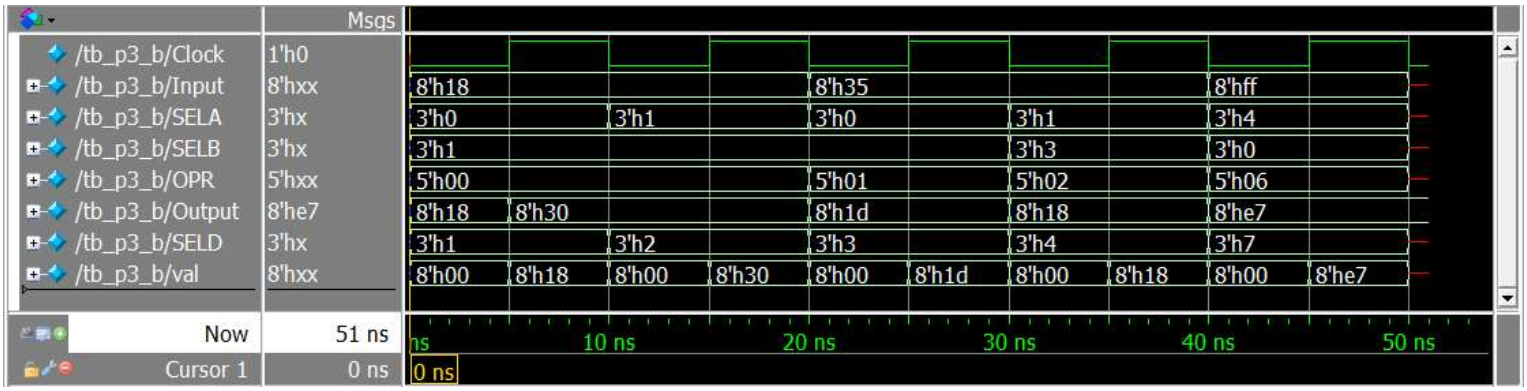
```
reg [4:0] OPR;
reg Clock = 0;
wire [7:0] Output;

// Instantiate p3_b
p3_b UUT (Output, Input, SELA, SELB, SELD, OPR, Clock);

wire [7:0] val = UUT.R[SELD];   // Value Stored in desired register

// 1 Clock Cycle = 10ns
always #5 Clock = ~Clock;

initial begin
    // Addition of Input=24 and R1=0, store result=24 in R1
    Input = 8'd24;
    SELA = 3'd0;
    SELB = 3'd1;
    SELD = 3'd1;
    OPR = 5'd0;
    #10;

    // Addition of R1=24 and R1=24, store result=48 in R2
    SELA = 3'd1;
    SELB = 3'd1;
    SELD = 3'd2;
    OPR = 5'd0;
    #10;

    // Subtraction of R1=24 and Input=53, store result=29 in R3
    Input = 8'd53;
    SELA = 3'd0;
    SELB = 3'd1;
    SELD = 3'd3;
    OPR = 5'd1;
    #10;

    // Bitwise AND of R1=24 and R3=29, store result=20 in R4
    SELA = 3'd1;
    SELB = 3'd3;
    SELD = 3'd4;
    OPR = 5'd2;
    #10;

    // Bitwise XOR of R4=20 and Input=255, store result=235 in R7
    Input = 8'd255;
    SELA = 3'd4;
    SELB = 3'd0;
    SELD = 5'd7;
```

```
        OPR = 5'd6;
        #10;

        // End Simulation
        Input = 8'bx;
        SELA = 3'bx;
        SELB = 3'bx;
        SELD = 5'bx;
        OPR = 5'bx;
    end

    // Display all parameters
    initial begin
        $monitor("time=%g\tClock=%b Input=%d SELA=%d A_Bus=%d SELB=%d B_bus=%d OPR=%d
 Output=%d SELD=%d Value_Stored=%d",$time,Clock,Input,SELA,UUT.R[SELA],SELB,UUT.R[SEL
B],OPR,Output,SELD,val);
    end
endmodule
```

Text Output:

```
VSIM 52> run
# time=0 Clock=0 Input= 24 SELA=0 A_Bus=  x SELB=1 B_bus=  0 OPR= x Output=  x SELD=x Value_Stored=  x
# time=5 Clock=1 Input= 24 SELA=0 A_Bus=  x SELB=1 B_bus=  0 OPR= 0 Output= 24 SELD=x Value_Stored=  x
# time=10 Clock=0 Input= 24 SELA=0 A_Bus=  x SELB=1 B_bus=  0 OPR= 0 Output= 24 SELD=x Value_Stored=  x
# time=15 Clock=1 Input= 24 SELA=0 A_Bus=  x SELB=1 B_bus= 24 OPR= 0 Output= 48 SELD=1 Value_Stored= 24
# time=20 Clock=0 Input= 24 SELA=0 A_Bus=  x SELB=1 B_bus= 24 OPR= 0 Output= 48 SELD=1 Value_Stored= 24
# time=25 Clock=1 Input= 53 SELA=1 A_Bus= 24 SELB=0 B_bus=  x OPR= 0 Output= 77 SELD=1 Value_Stored= 24
# time=30 Clock=0 Input= 53 SELA=1 A_Bus= 24 SELB=0 B_bus=  x OPR= 0 Output= 77 SELD=1 Value_Stored= 24
# time=35 Clock=1 Input= 53 SELA=1 A_Bus= 24 SELB=0 B_bus=  x OPR= 0 Output= 77 SELD=1 Value_Stored= 24
# time=40 Clock=0 Input= 53 SELA=1 A_Bus= 24 SELB=0 B_bus=  x OPR= 0 Output= 77 SELD=1 Value_Stored= 24
# time=45 Clock=1 Input= 53 SELA=1 A_Bus= 24 SELB=0 B_bus=  x OPR= 0 Output= 77 SELD=2 Value_Stored= 77
# time=50 Clock=0 Input= 53 SELA=1 A_Bus= 24 SELB=0 B_bus=  x OPR= 0 Output= 77 SELD=2 Value_Stored= 77
```

Simulation Waveform:

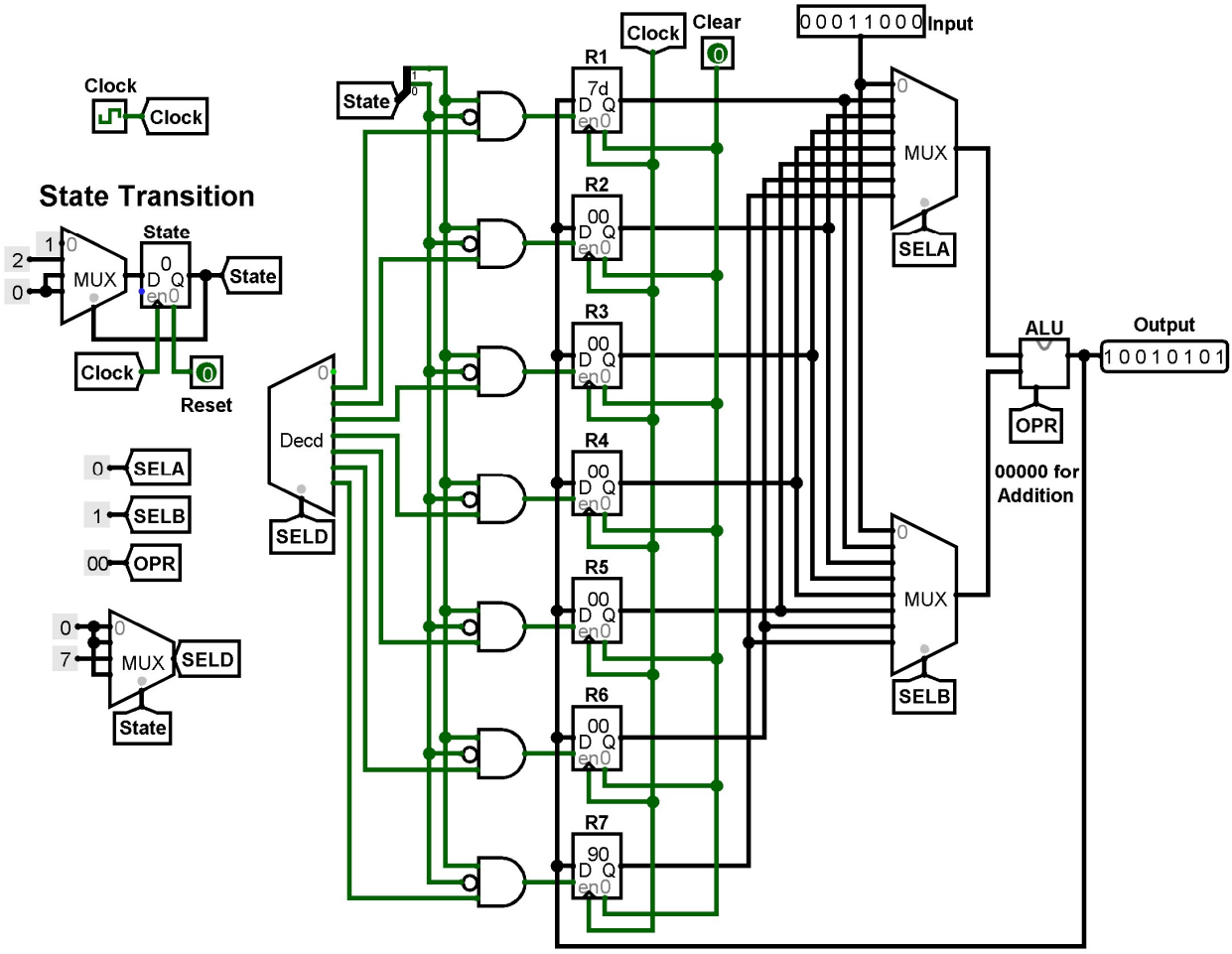


Simulation Waveform: **51ns** 1 Clock Cycle = **10ns**

Ans 3:

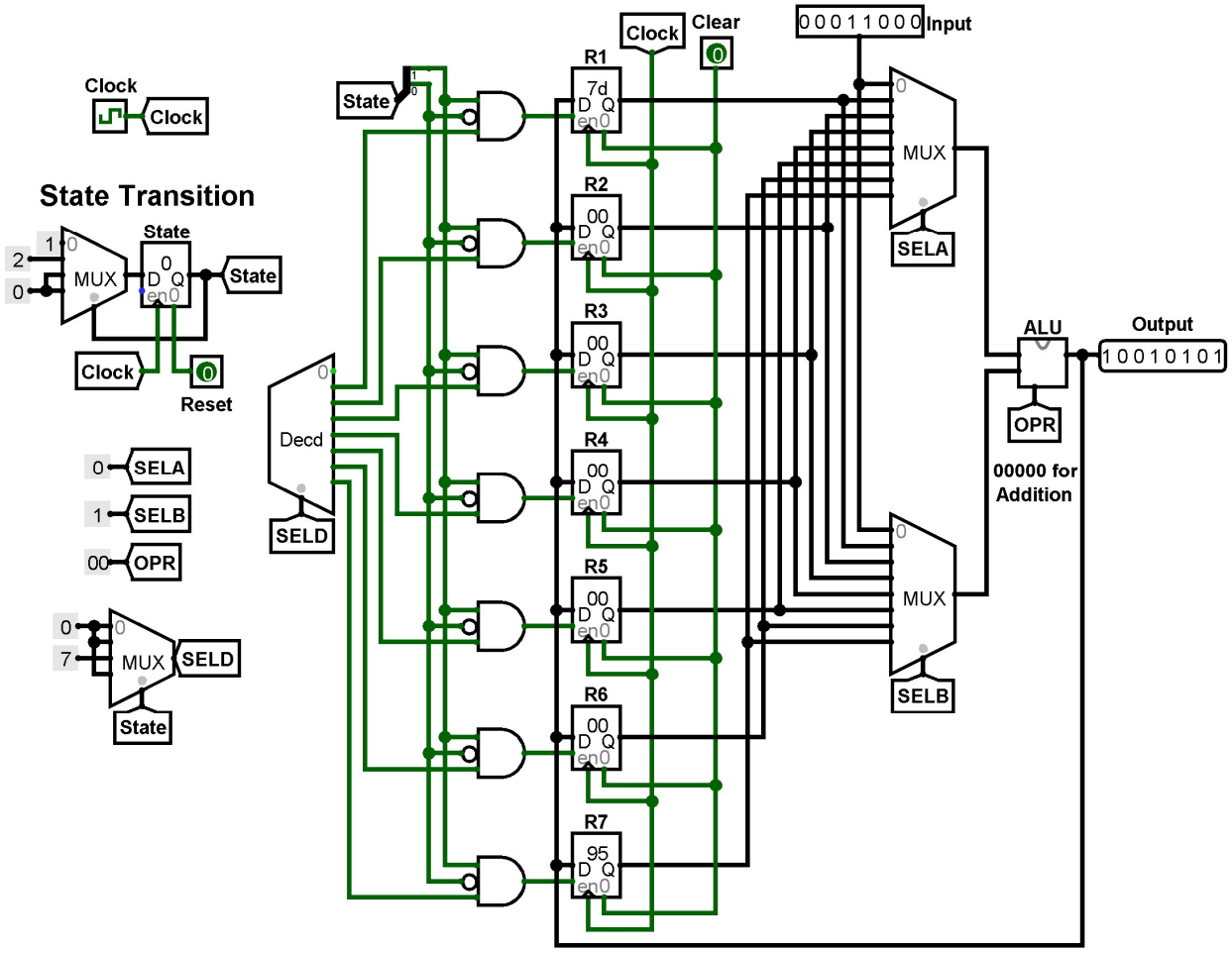
(c) Design a simple state machine to add two numbers that is stored in the registers(R1-R7) and store the result in another Register(R1-R7). Hint: generate control signals (SELA, SELB, OPR etc) . Integrate this FSM to Part (a) and Part (b)

Logisim Circuit:



Sample Test Run:

1. In the registers R1-7, R1 already has value DEC 125 or HEX 7d. R7 has DEC 144 or HEX 90.
2. Initially the FSM is at **State is 0 (READ state).** In this state, SELA = 0, so Input = DEC 24 or HEX 18 is input to the ALU. Also, since SELB = 1, R1 = DEC 125 or HEX 7d is another input to ALU.
3. Toggle Clock. Now, the State is 1 (CALC state). In this state, OPR = 0 implies the inputs to the ALU are added.



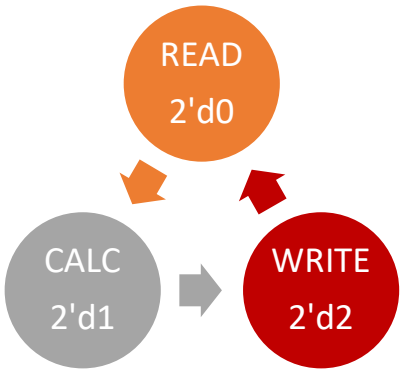
4. Toggle Clock. Now, state is 2, SELD is set to 7. Hence Output is stored in register R7, in next clock cycle.

Verilog Model:

```
 // FSM to add 2 numbers from Registers R1-R7 of given block
module p3_c (Output, Input, Clock, Clear, Reset);
    input [7:0] Input;                  // Input Line
    input Clock;                        // Clock
    input Clear;                        // Clear data from registers
    input Reset;                        // Back to Initial State
    output [7:0] Output;                // Output Lines

    // State Encoding
    parameter READ  = 2'd0;             // Read from registers
    parameter CALC  = 2'd1;             // Calculate Output
    parameter WRITE = 2'd2;             // Write to register

    reg [7:0] R [7:1];                  // Registers 1-7
    reg [4:0] OPR;                      // Operation Code
    reg [1:0] state = READ;             // State Register
```

```
reg [2:0] SELA, SELB, SELD;         // Select Lines
wire [7:0] A_bus, B_bus;            // A bus and B bus

// Instantiate MUX and ALU submodules
MUX8_1 muxA (A_bus, Input, R[1], R[2], R[3], R[4], R[5], R[6], R[7], SELA);
MUX8_1 muxB (B_bus, Input, R[1], R[2], R[3], R[4], R[5], R[6], R[7], SELB);
ALU    alu (Output, A_bus, B_bus, OPR);

integer i,j;
initial begin
    for (i=1; i<=7; i=i+1) begin
        R[i] = 8'b0;
    end
    // Initial Values in R1, R2, R7
    R[1] = 8'd125;
    R[7] = 8'd144;
end

always @(posedge Clock) begin
    // State Transition
    // If Reset = 1, go back to Read State
    if (Reset) state = READ;
    else state = (state+1)%3;
    //Else, move to next state

    // Clear Data in all Registers
    if (Clear) begin
        for (j=1; j<=7; j=j+1) begin
            R[j] = 8'b0;
        end
    end
end

always @(state) begin
    case(state)
    READ: begin
        // Input line and R1 will be added
        SELA = 3'd0;
        SELB = 3'd1;
    end
    CALC: begin
        // Add Input line and R1
        OPR = 5'd0;
    end
    WRITE: begin
        // Output is stored in Register 7
        SELD = 3'd7;
        R[SELD] = Output;
```

State Diagram:



State Description

READ:
Read from registers/input line corresponding to **SELA (= Input line)** and **SELB (= R1)** values.

CALC:
Calculate i.e., Add values read in READ state (**OPR = 0** for addition)

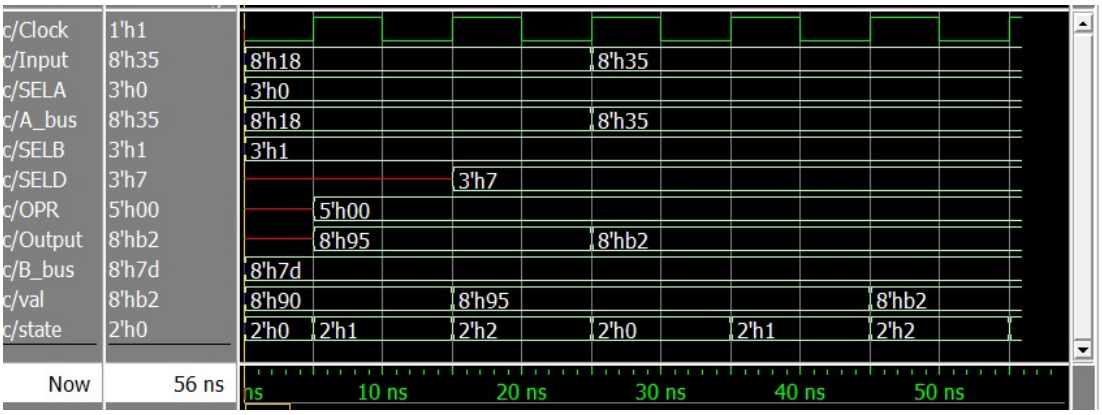
WRITE:
Store the output in CALC state to register corresponding to **SELD**

```
        end
        endcase
    end
endmodule
```

Testbench:

```
// Testbench for p3_c (FSM) in p3.v
module tb_p3_c();
    // Mode Declaration
    reg [7:0] Input;
    reg Clear = 0, Clock = 0, Reset = 0;
    wire [7:0] Output;

    // Instantiate p3_c
    p3_c UUT (Output, Input, Clock, Clear, Reset);

    // Parameters of p3_c for simulation waveform
    wire [2:0] SELA = UUT.SELA;
    wire [2:0] SELB = UUT.SELB;
    wire [2:0] SELD = UUT.SELD;
    wire [4:0] OPR = UUT.OPR;
    wire [7:0] A_bus = UUT.A_bus;
    wire [7:0] B_bus = UUT.B_bus;
    wire [7:0] val = UUT.R[3'd7];   // Value stored in desired register
    wire [1:0] state = UUT.state;   // State of FSM

    // 1 Clock Cycle = 10ns
    always #5 Clock = ~Clock;

    initial begin
        // Add Input=24 and R1=125, store result=149 in R7
        Input = 8'd24;
        #25;
        // Add Input=53 and R1=125, store result=178 in R7
        Input = 8'd53;
    end

    // Display all parameters
    initial begin
        $monitor("time=%g\tClock=%b State=%d Input=%d SELA=%d A_Bus=%d SELB=%d B_bus=
%d OPR=%d Output=%d SELD=%d Value_Stored=%d",$time,Clock,state,Input,UUT.SELA,UUT.A_b
us,UUT.SELB,B_bus,UUT.OPR,Output,UUT.SELD,val);
    end
endmodule
```

Simulation Waveform:



Note:
val represents value stored in register corresponding to SELD.

Text Output:

```
# time=0 Clock=0 State=0 Input= 24 SELA=0 A_Bus= 24 SELB=1 B_bus=125 OPR= x Output=  x SELD=x Value_Stored=144
# time=5 Clock=1 State=1 Input= 24 SELA=0 A_Bus= 24 SELB=1 B_bus=125 OPR= 0 Output=149 SELD=x Value_Stored=144
# time=10 Clock=0 State=1 Input= 24 SELA=0 A_Bus= 24 SELB=1 B_bus=125 OPR= 0 Output=149 SELD=x Value_Stored=144
# time=15 Clock=1 State=2 Input= 24 SELA=0 A_Bus= 24 SELB=1 B_bus=125 OPR= 0 Output=149 SELD=7 Value_Stored=149
# time=20 Clock=0 State=2 Input= 24 SELA=0 A_Bus= 24 SELB=1 B_bus=125 OPR= 0 Output=149 SELD=7 Value_Stored=149
# time=25 Clock=1 State=0 Input= 53 SELA=0 A_Bus= 53 SELB=1 B_bus=125 OPR= 0 Output=178 SELD=7 Value_Stored=149
# time=30 Clock=0 State=0 Input= 53 SELA=0 A_Bus= 53 SELB=1 B_bus=125 OPR= 0 Output=178 SELD=7 Value_Stored=149
# time=35 Clock=1 State=1 Input= 53 SELA=0 A_Bus= 53 SELB=1 B_bus=125 OPR= 0 Output=178 SELD=7 Value_Stored=149
# time=40 Clock=0 State=1 Input= 53 SELA=0 A_Bus= 53 SELB=1 B_bus=125 OPR= 0 Output=178 SELD=7 Value_Stored=149
# time=45 Clock=1 State=2 Input= 53 SELA=0 A_Bus= 53 SELB=1 B_bus=125 OPR= 0 Output=178 SELD=7 Value_Stored=178
# time=50 Clock=0 State=2 Input= 53 SELA=0 A_Bus= 53 SELB=1 B_bus=125 OPR= 0 Output=178 SELD=7 Value_Stored=178
# time=55 Clock=1 State=0 Input= 53 SELA=0 A_Bus= 53 SELB=1 B_bus=125 OPR= 0 Output=178 SELD=7 Value_Stored=178
```

Simulation Time: **56ns** 1 Clock Cycle = **10ns**

ALU Module used for Question 3

```
// ALU with 32 Operations
module ALU (out,A,B,OPR);
    input [7:0] A,B;        // Input numbers
    input [4:0] OPR;        // Operation Code
    output reg [7:0] out;   // ALU Output

    // Operation Code for Addition is 5'b0_0000
    always @(OPR or A or B) begin
        case(OPR)
            5'd0: out = A+B;
            5'd1: out = A-B;
            5'd2: out = A&B;
            5'd3: out = A|B;
```

```
            5'd4: out = ~(A&B);
            5'd5: out = ~(A|B);
            5'd6: out = A^B;
            5'd7: out = ~(A^B);
            5'd8: out = A>B;
            5'd9: out = A==B;
            5'd10: out = A<B;
            5'd11: out = A*B;
            5'd12: out = A/B;
            5'd13: out = -A;
            5'd14: out = -B;
            5'd15: out = A<<2;
            5'd16: out = A>>2;
            5'd17: out = B<<2;
            5'd18: out = B>>2;
            5'd19: out = ~A;
            5'd20: out = ~B;
            5'd21: out = ~A & B;
            5'd22: out = A & ~B;
            5'd23: out = ~A | B;
            5'd24: out = A | ~B;
            5'd25: out = A;
            5'd26: out = B;
            5'd27: out = A^(8'hff);
            5'd28: out = B^(8'hff);
            5'd29: out = A^(8'hff)+1;
            5'd30: out = B^(8'hff)+1;
            5'd31: out = {A[7],A[6],A[5],A[4],A[3],A[2],A[1],A[0]};
        endcase
    end
endmodule
```

Multiplexer module used for Question 3:

```
module MUX8_1 (out, ip0, ip1, ip2, ip3, ip4, ip5, ip6, ip7, sel);
    // MUX Inputs
    input [7:0] ip0,ip1,ip2,ip3,ip4,ip5,ip6,ip7;
    input [2:0] sel;        // MUX Select
    output [7:0] out;       // MUX Output

    assign out = (sel == 3'd0) ? ip0 :
                 (sel == 3'd1) ? ip1 :
                 (sel == 3'd2) ? ip2 :
                 (sel == 3'd3) ? ip3 :
                 (sel == 3'd4) ? ip4 :
                 (sel == 3'd5) ? ip5 :
                 (sel == 3'd6) ? ip6 : ip7;
endmodule
```