PROJECT 2:

Project: Product Machine and State Enumeration

Nikolay Nikolov

1 Problem 1:

Design a sequential circuit S1 with the following requirements

a) The combination logic part for the next state function should contain at least 2 NAND gates, 1 XOR gate, and one NOR.

b) The number of the flip-flops should be at least 2.

c) The sequential circuit should have an output.

d) Make sure that your design be different from those used by other students.

If your design is identical to some design used by other students, further investigation will be conducted and you will be asked to revise your design and redo the work.

1.1 Draw your sequential circuit S

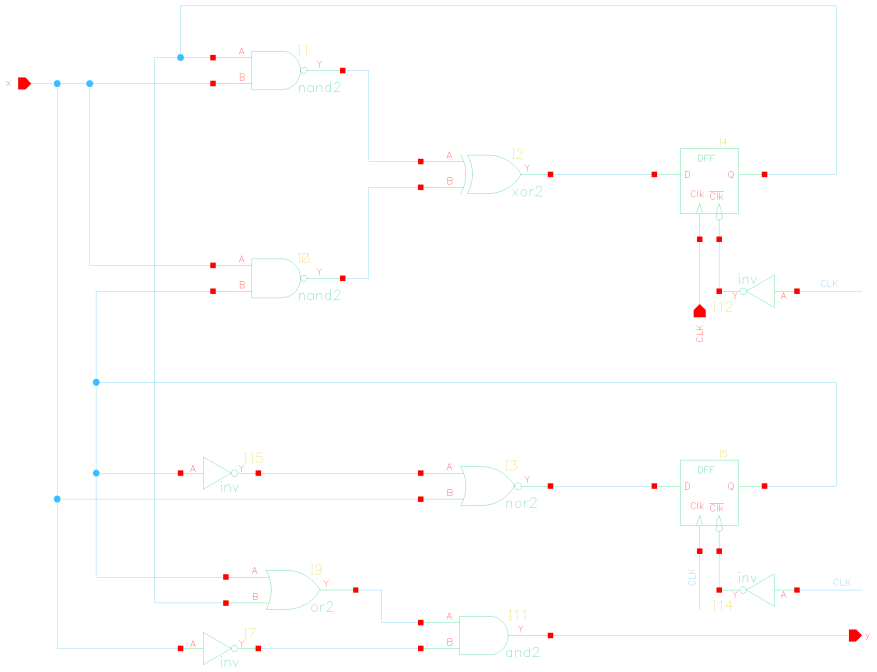

Fig. 1: Sequential Circuit Schematic for S

1.2 Create the next state function table for S1

Present State	Input	Next State	Output		
A	B	X	A	B	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	0	0
1	0	0	0	0	1
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	0	0	0

Fig. 2: Next State function table

1.3 From an initial state (you decide), explore the entire state space and draw the state transition graph.

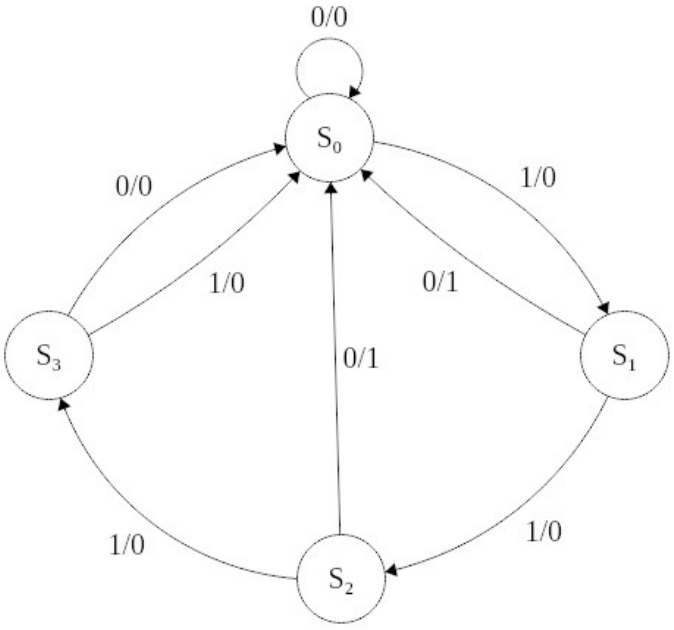

Fig. 3: State Diagram

1.4 Design in verilog as a Mealy FSM

```
`timescale 1 ns/10 ps

module S (output reg Y,output reg [1:0] state,
          input wire clk, reset, x);

   reg [1:0]          current_state, next_state;

   parameter state00 = 2'b00;
   parameter state01 = 2'b01;
   parameter state10 = 2'b10;
   parameter state11 = 2'b11;

   assign state = current_state;

   // ********************************
   // Memory block
   // ********************************
   always @ (posedge clk or negedge reset)
     begin: STATE_MEMORY
        if (!reset)
          current_state <= state00;
        else
          current_state <= next_state;
     end
   // ********************************
   // Next State
   // ********************************
   always @ (current_state or x)
     begin: NEXT_STATE_LOGIC
        case (current_state)
          state00:
            if (x == 1'b1)
            next_state = state01;
          else
            next_state = state00;
          state01:
            if (x == 1'b1)
            next_state = state10;
          else
            next_state = state00;
          state10:
            if (x == 1'b1)
            next_state = state11;
          else
```

```
            next_state = state00;
          state11:
             next_state = state00;
          default: next_state = state00;
        endcase // case (current_state)
    end
  // ******************************
  // Output Logic
  // ******************************
  always @ (current_state or x)
    begin : OUTPUT_LOGIC
       case (current_state)
         state00:
             Y = 1'b0;
         state01:
           if (x == 1'b0)
             Y = 1'b1;
           else
             Y = 1'b0;
         state10:
           if (x == 1'b0)
             Y = 1'b1;
           else
             Y = 1'b0;
         state11:
             Y= 1'b1;
         default: Y = 1'b0;
       endcase // case (current_state)
    end
endmodule // S



// TestBench
`timescale 1 ns/10 ps

module tb_S ();
        wire Y;
        reg  clk, reset, x;
        reg [1:0] state;

  S DUT (.Y(Y),.state(state),.clk(clk), .reset(reset), .x(x));

   initial
     begin
        reset = 1'b0;
```

```
        reset= 1'b1;
    end

  initial
    begin
        clk= 1'b0;
    end

  always
    begin
        #5 clk= ~clk;
    end

  initial
    begin
            x = 1'b0;
        #9 x = 1'b1;
        #9 x = 1'b1;
        #9 x = 1'b1;
        #9 x = 1'b1;
        #9 x = 1'b1;
        #9 x = 1'b1;
        #9 x = 1'b1;
        #9 x = 1'b1;
        #9 x = 1'b1;
        #9 x = 1'b1;
        #9 x = 1'b1;
        #9 x = 1'b1;
        #9 x = 1'b0;
        #9 x = 1'b0;
        #9 x = 1'b0;
        #9 x = 1'b1;
        #9 x = 1'b0;
        #9 x = 1'b1;

    end

initial
  begin
    $display(" state x Y");
    $monitor("%2b, %2b, %2b", state, x, Y);
  end
initial
  begin
$dumpfile("dump.vcd"); $dumpvars;
#500 $finish;
```

```
    end

endmodule // tb_S
```

1.5 Simulation as a Mealy FSM

```
[2020-07-20 01:41:59 EDT] iverilog '-Wall' '-g2012' design.sv testbench.sv
&& unbuffer vvp a.out
 state x Y
VCD info: dumpfile dump.vcd opened for output.
xx,  0,  0
00,  0,  0
00,  1,  0
01,  1,  0
10,  1,  0
11,  1,  1
00,  1,  0
01,  1,  0
10,  1,  0
11,  1,  1
00,  1,  0
01,  1,  0
10,  1,  0
11,  1,  1
11,  0,  1
00,  0,  1
00,  1,  0
```

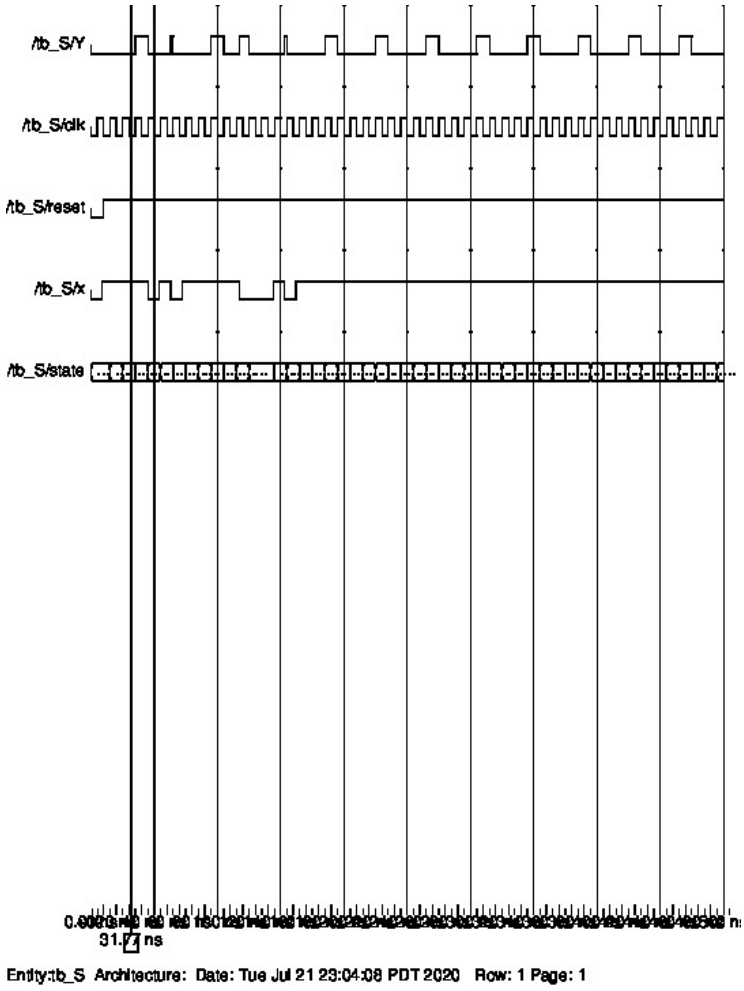

Fig. 4: Timing Diagram

2 Problem 2:

Get a copy of S1 and name it as S2.

a) Except the inputs, please add the index 1 to all the wire names of S1, while adding the index 2 to all the wire names of S2.

b) To use the shared input product machine, please make sure that the corresponding inputs have the same variable names.

c) Build the product machine with the shared same input P=S1S2 (as defined in class)

2.1 Source Code for S2

```
timescale 1 ns/10 ps

module S2 (output reg Y2, output reg [1:0] state2,
           input wire clk, reset, x2);

   reg [1:0]          current_state, next_state;

   parameter state00 = 2'b00;
   parameter state01 = 2'b01;
   parameter state10 = 2'b10;
   parameter state11 = 2'b11;
   // ********************************
   // Memory block
   // ********************************
   always @ (posedge clk or negedge reset)
     begin: STATE_MEMORY
        if (!reset)
          current_state <= state00;
        else
          current_state <= next_state;
     end
   // ********************************
   // Next State
   // ********************************
   always @ (current_state or x2)
     begin: NEXT_STATE_LOGIC
        case (current_state)
          state00:
            if (x2 == 1'b1)
              next_state = state01;
            else
              next_state = state00;
          state01:
            if (x2 == 1'b1)
              next_state = state10;
            else
              next_state = state00;
          state10:
            if (x2 == 1'b1)
              next_state = state11;
            else
              next_state = state00;
          state11:
            next_state = state00;
```

```
          default: next_state = state00;
        endcase // case (current_state)
    end
  // ******************************
  // Output Logic
  // ******************************
  always @ (current_state or x2)
    begin : OUTPUT_LOGIC
       case (current_state)
         state00:
           Y2 = 1'b0;
         state01:
           if (x2 == 1'b0)
             Y2 = 1'b1;
           else
             Y2 = 1'b0;
         state10:
           if (x2 == 1'b0)
             Y2 = 1'b1;
           else
             Y2 = 1'b0;
         state11:
           Y2= 1'b1;
         default: Y2 = 1'b0;
       endcase // case (current_state)
    end

  always @ (current_state or x2)
    begin
       state2 = current_state;
    end

endmodule // S2
```

Fig. 5: S2 source code

2.2 Test-bench for S2

```
`timescale 1 ns/10 ps

module tb_S2 ();
  wire Y2;
  reg  clk, reset, x2;
  wire [1:0] state2;
```

```
S2 DUT (.Y2(Y2),.state2(state2),.clk(clk), .reset(reset), .x2(x2));

initial
  begin
     reset = 1'b0;
     #10 reset= 1'b1;
  end

initial
  begin
     clk= 1'b0;
  end

always
  begin
     #5 clk= ~clk;
  end

initial
  begin
          x2 = 1'b0;
     #9 x2 = 1'b1;
     #9 x2 = 1'b1;
     #9 x2 = 1'b1;
     #9 x2 = 1'b1;
     #9 x2 = 1'b0;
     #9 x2 = 1'b1;
     #9 x2 = 1'b0;
     #9 x2 = 1'b1;
     #9 x2 = 1'b1;
     #9 x2 = 1'b1;
     #9 x2 = 1'b1;
     #9 x2 = 1'b1;
     #9 x2 = 1'b0;
     #9 x2 = 1'b0;
     #9 x2 = 1'b0;
     #9 x2 = 1'b1;
     #9 x2 = 1'b0;
     #9 x2 = 1'b1;

  end

initial
  begin
    $display(" state2 x2 Y2");
```

```
      $monitor("%2b, %2b %2b", state2, x2, Y2);
    end
  initial
    begin

  #500 $finish;
    end
```

Fig. 6: S2 Test-bech

2.3 Timing Diagram for S2

Fig. 7: S2 Timing Diagram

2.4 Trace for S2

```
sim 582P2.tb_S2
# vsim 582P2.tb_S2
# Start time: 21:37:48 on Jul 22,2020
# Loading 582P2.tb_S2
# Loading 582P2.S2
add wave sim:/tb_S2/*
run -all
#  state2 x2 Y2
# 00, 00 00
# 00, 01 00
# 01, 01 00
# 10, 01 00
# 11, 01 01
# 00, 00 00
# 00, 01 00
# 01, 01 00
# 01, 00 01
# 00, 00 00
# 00, 01 00
# 01, 01 00
# 10, 01 00
# 11, 01 01
# 00, 01 00
# 01, 01 00
# 01, 00 01
# 00, 00 00
# 00, 01 00
```

```
# 01, 01 00
# 01, 00 01
# 00, 00 00
# 00, 01 00
# 01, 01 00
# 10, 01 00
# 11, 01 01
# 00, 01 00
# 01, 01 00
# 10, 01 00
# 11, 01 01
# 00, 01 00
# 01, 01 00
# 10, 01 00
# 11, 01 01
# 00, 01 00
# 01, 01 00
# 10, 01 00
# 11, 01 01
# 00, 01 00
# 01, 01 00
# 10, 01 00
# 11, 01 01
# 00, 01 00
# 01, 01 00
# 10, 01 00
# 11, 01 01
# 00, 01 00
# 01, 01 00
# 10, 01 00
# ** Note: $finish    : source/tb_S2.v(58)
#    Time: 500 ns  Iteration: 0  Instance: /tb_S2
# 1
# Break in Module tb_S2 at source/tb_S2.v line 58
# Can't move the Now cursor.
```

2.5 Product State Machine

```
timescale 1 ns/10 ps

module top (output wire Y1,
            output wire       Y2,
            output wire [1:0] state1,
            output wire [1:0] state2,
            input wire       clk, reset, x);

   S1 U1 (.Y1(Y1),.state1(state1),.clk(clk), .reset(reset), .x1(x));
   S2 U2 (.Y2(Y2),.state2(state2),.clk(clk), .reset(reset), .x2(x));

endmodule
```

2.6 Transcript from simulation

```
vsim 582P2.tb_top
# vsim 582P2.tb_top
# Start time: 22:45:27 on Jul 22,2020
# Loading 582P2.tb_top
# Loading 582P2.top
# Loading 582P2.S1
# Loading 582P2.S2
add wave sim:/tb_top/*
run -all
#  x     state1     Y1      state2    Y2
# 00  00  00   00  00
# 01  00  00   00  00
# 01  01  00   01  00
# 01  10  00   10  00
# 01  11  01   11  01
# 00  00  00   00  00
# 01  00  00   00  00
# 01  01  00   01  00
# 00  01  01   01  01
# 00  00  00   00  00
# 01  00  00   00  00
# 01  01  00   01  00
# 01  10  00   10  00
# 01  11  01   11  01
# 01  00  00   00  00
# 01  01  00   01  00
# 00  01  01   01  01
# 00  00  00   00  00
# 01  00  00   00  00
# 01  01  00   01  00
# 00  01  01   01  01
```

```
# 00 00 00  00 00
# 01 00 00  00 00
# 01 01 00  01 00
# 01 10 00  10 00
# 01 11 01  11 01
# 01 00 00  00 00
# 01 01 00  01 00
# 01 10 00  10 00
# 01 11 01  11 01
# 01 00 00  00 00
# 01 01 00  01 00
# 01 10 00  10 00
# 01 11 01  11 01
# 01 00 00  00 00
# 01 01 00  01 00
# 01 10 00  10 00
# 01 11 01  11 01
# 01 00 00  00 00
# 01 01 00  01 00
# 01 10 00  10 00
# 01 11 01  11 01
# 01 00 00  00 00
# 01 01 00  01 00
# 01 10 00  10 00
# 01 11 01  11 01
# 01 00 00  00 00
# 01 01 00  01 00
# 01 10 00  10 00
# 01 11 01  11 01
# 01 00 00  00 00
# 01 01 00  01 00
# 01 10 00  10 00
# 01 11 01  11 01
# 01 00 00  00 00
# 01 01 00  01 00
# 01 10 00  10 00
# ** Note: $finish    : source/tb_top.v(57)
```