11/28/2016

Project 3

Sequence Recognizer 1001

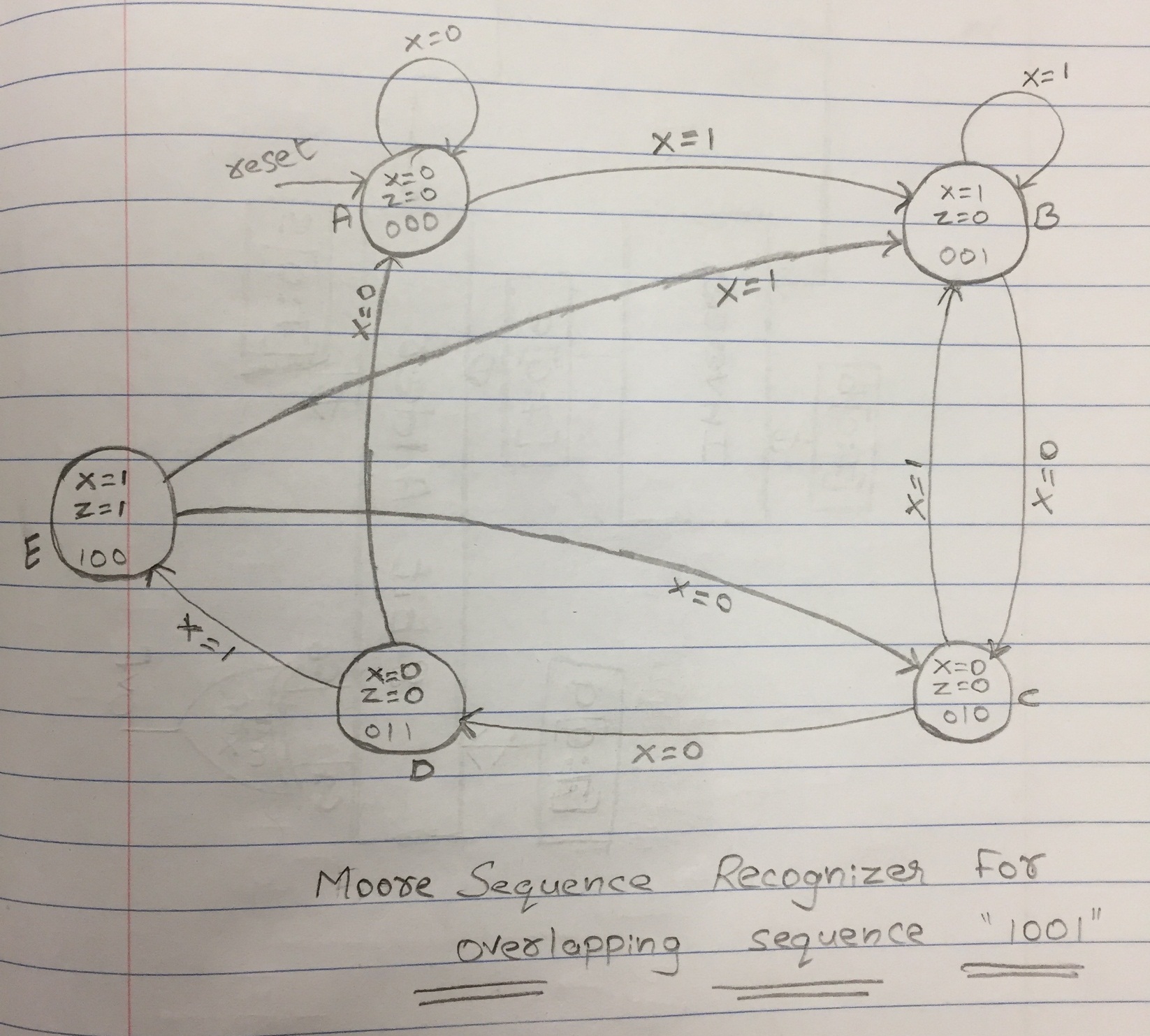
Bibodi, Jay Nikhil

Sec 2: 5:30-6:45

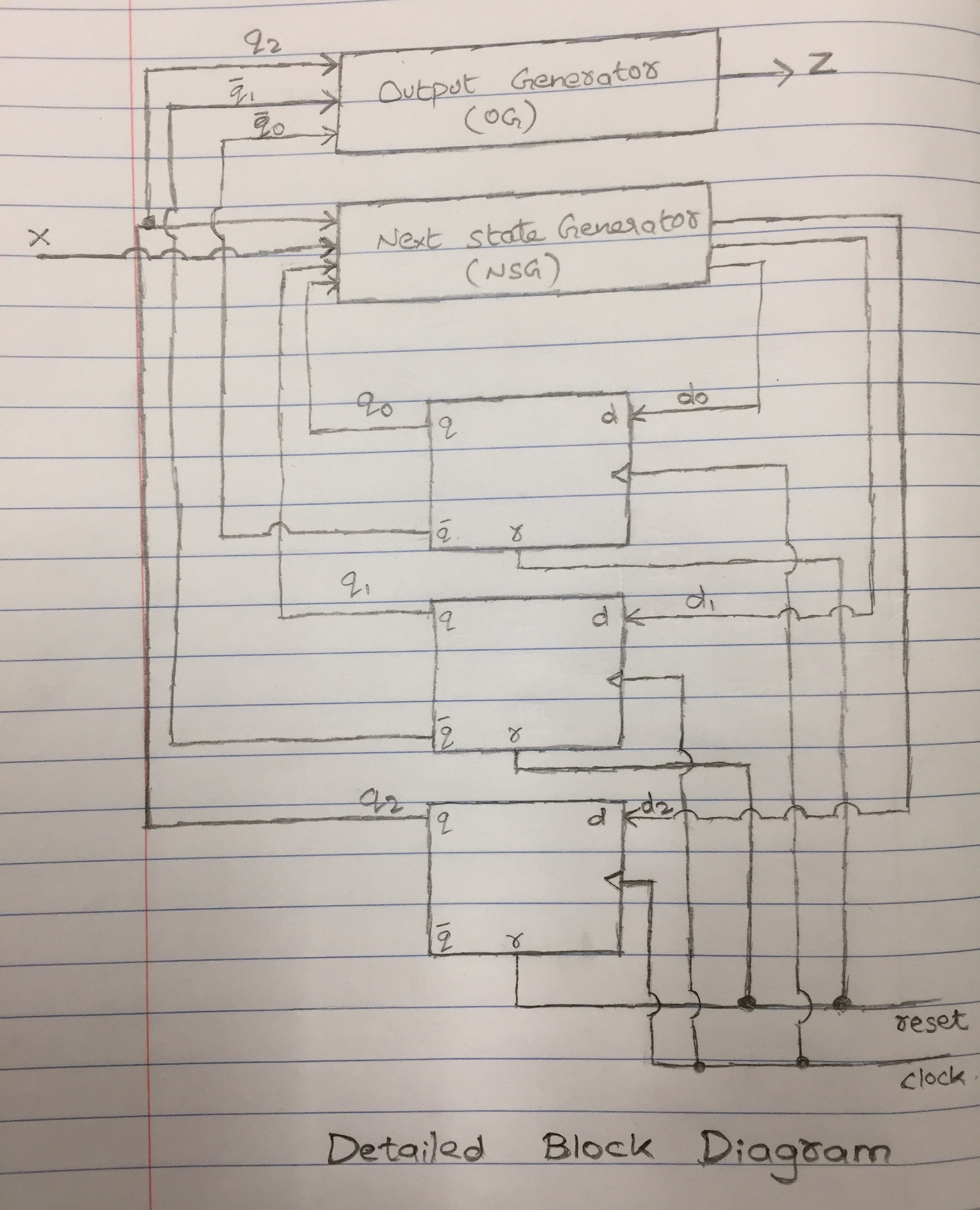
Fall 2016

**1. Consider problem 5.10**

1. **Moore’s Sequence Recognizer for overlapping sequence 1001**

****

1. **Detailed block diagram**



1. **Truth table for OG**

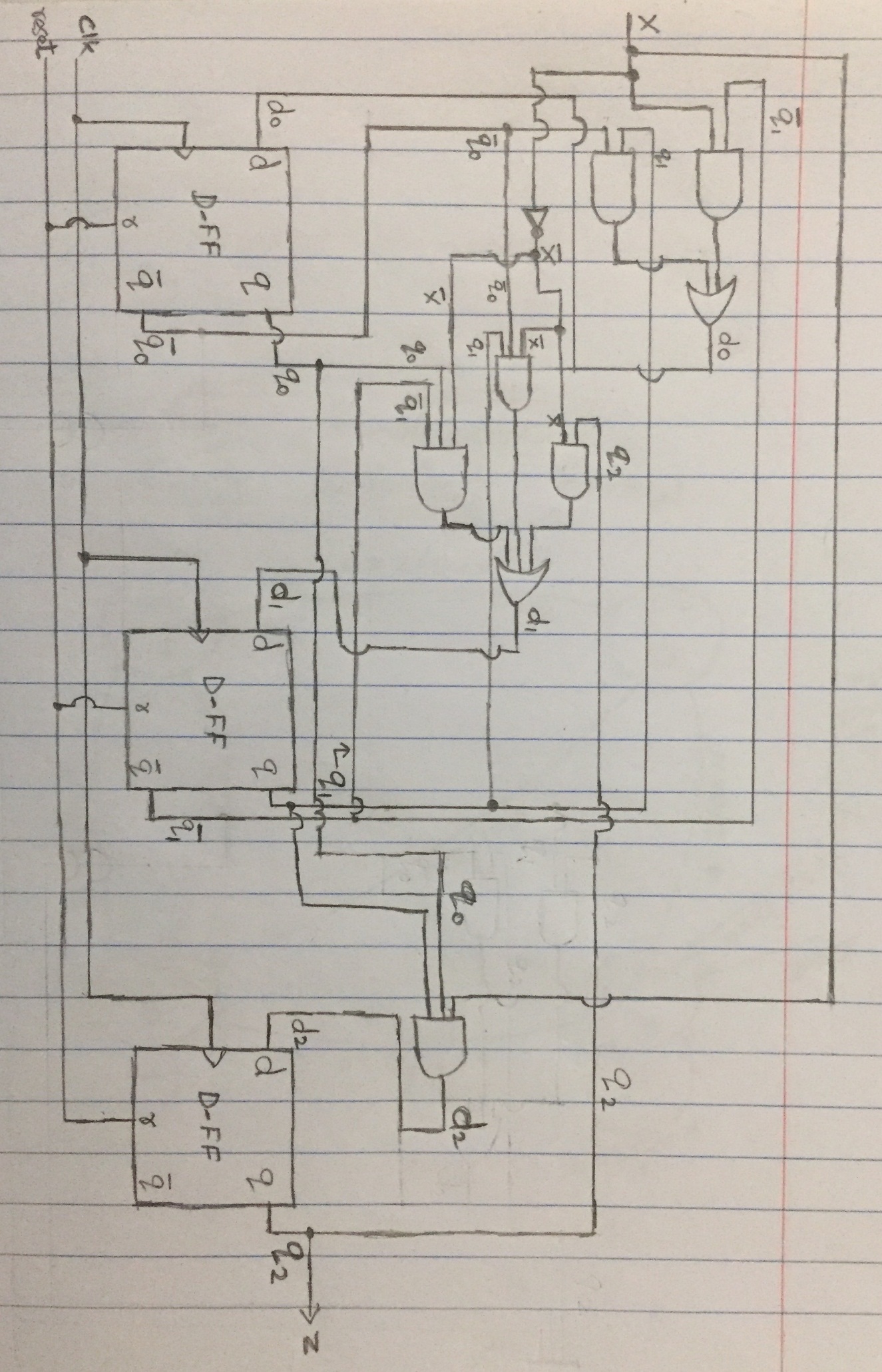
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **State Name** | **Current State** | | | **Output** |
|  | q2 | q1 | q0 | Z |
|  |  |  |  |  |
| A | 0 | 0 | 0 | 0 |
| B | 0 | 0 | 1 | 0 |
| C | 0 | 1 | 0 | 0 |
| D | 0 | 1 | 1 | 0 |
| E | 1 | 0 | 0 | 1 |
|  | d | d | d | d |

1. **Truth table for NSG**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **State Name** | **Current State** | | | **Input** | **Next State** | | | **Next State Name** |
|  | q2 | q1 | q0 | x | d2 | d1 | d0 |  |
|  |  |  |  |  |  |  |  |  |
| A | 0 | 0 | 0 | 0 | 0 | 0 | 0 | A |
| A | 0 | 0 | 0 | 1 | 0 | 0 | 1 | B |
|  |  |  |  |  |  |  |  |  |
| B | 0 | 0 | 1 | 0 | 0 | 1 | 0 | C |
| B | 0 | 0 | 1 | 1 | 0 | 0 | 1 | B |
|  |  |  |  |  |  |  |  |  |
| C | 0 | 1 | 0 | 0 | 0 | 1 | 1 | D |
| C | 0 | 1 | 0 | 1 | 0 | 0 | 1 | B |
|  |  |  |  |  |  |  |  |  |
| D | 0 | 1 | 1 | 0 | 0 | 0 | 0 | A |
| D | 0 | 1 | 1 | 1 | 1 | 0 | 0 | E |
|  |  |  |  |  |  |  |  |  |
| E | 1 | 0 | 0 | 0 | 0 | 1 | 0 | C |
| E | 1 | 0 | 0 | 1 | 0 | 0 | 1 | B |
|  |  |  |  |  |  |  |  |  |
| - | d | d | d | d | d | d | d | - |

d – don’t Cares in the truth table

1. **Logic Expression for output and next state variables.**
2. **Final Circuit for FSM**



1. **Behavioral Model for OG. (OG.v)**

module OG

(

input q2,

output z

);

assign z = q2;

endmodule

1. **Behavioral Model for NSG. (NSG.v)**

module NSG

(

input [2:0] q,

input x,

output [2:0] d

);

assign d[2] = q[1] & q[0] & x;

assign d[1] = (q[2] & ~x) | (q[1] & ~q[0] & ~x) | (~q[1] & q[0] & ~x);

assign d[0] = (~q[1] & x) | (q[1] & ~q[0]);

endmodule

1. **Behavioral Model for FF. (FlipFlop.v)**

module FlipFlop

(

input clock,reset,

input [2:0] nextState,

output reg [2:0] currentState

);

always@(posedge clock,posedge reset)

begin

if(reset == 1)

currentState <= 3'b000;

else

currentState <= nextState;

end

endmodule

1. **Combined Models to create FSM (Project.v)**

`include "OG.v";

`include "FlipFlop.v";

`include "NSG.v";

module Project

(

input clock,reset,x,

output out

);

wire [2:0] currentState, nextState;

NSG a(currentState,x,nextState);

FlipFlop b(clock,reset,nextState,currentState);

OG h(currentState[2],out);

endmodule

1. **TestBench (TestBench.v)**

`include "Project.v";

module TestBench();

reg clock,reset,x;

wire z;

Project ProjectOne(clock,reset,x,z);

initial begin

$monitor("%4d: z = %b",$time,z);

clock=0;

reset=1; //Resets the FlipFlop

x=0;

#10 reset=0; //end reset

end

always

begin

#5clock=~clock; //generates a clock signal with period 10

end

initial begin // One input per clock cycle

#10 x = 0; $display ("%4d: x = %b",$time,x);

#10 x = 0; $display ("%4d: x = %b",$time,x);

#10 x = 1; $display ("%4d: x = %b",$time,x);

#10 x = 0; $display ("%4d: x = %b",$time,x);

#10 x = 0; $display ("%4d: x = %b",$time,x);

#10 x = 1; $display ("%4d: x = %b",$time,x);

#10 x = 1; $display ("%4d: x = %b",$time,x);

#10 x = 0; $display ("%4d: x = %b",$time,x);

#10 x = 0; $display ("%4d: x = %b",$time,x);

#10 x = 1; $display ("%4d: x = %b",$time,x);

#10 x = 0; $display ("%4d: x = %b",$time,x);

#10 x = 0; $display ("%4d: x = %b",$time,x);

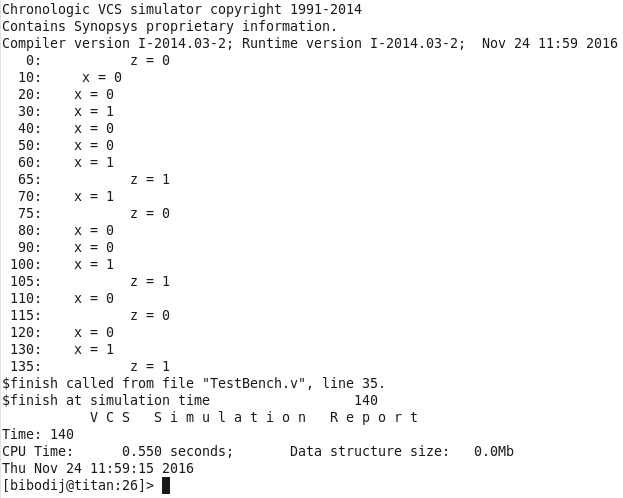
#10 x = 1; $display ("%4d: x = %b",$time,x);

#10 $finish;

end

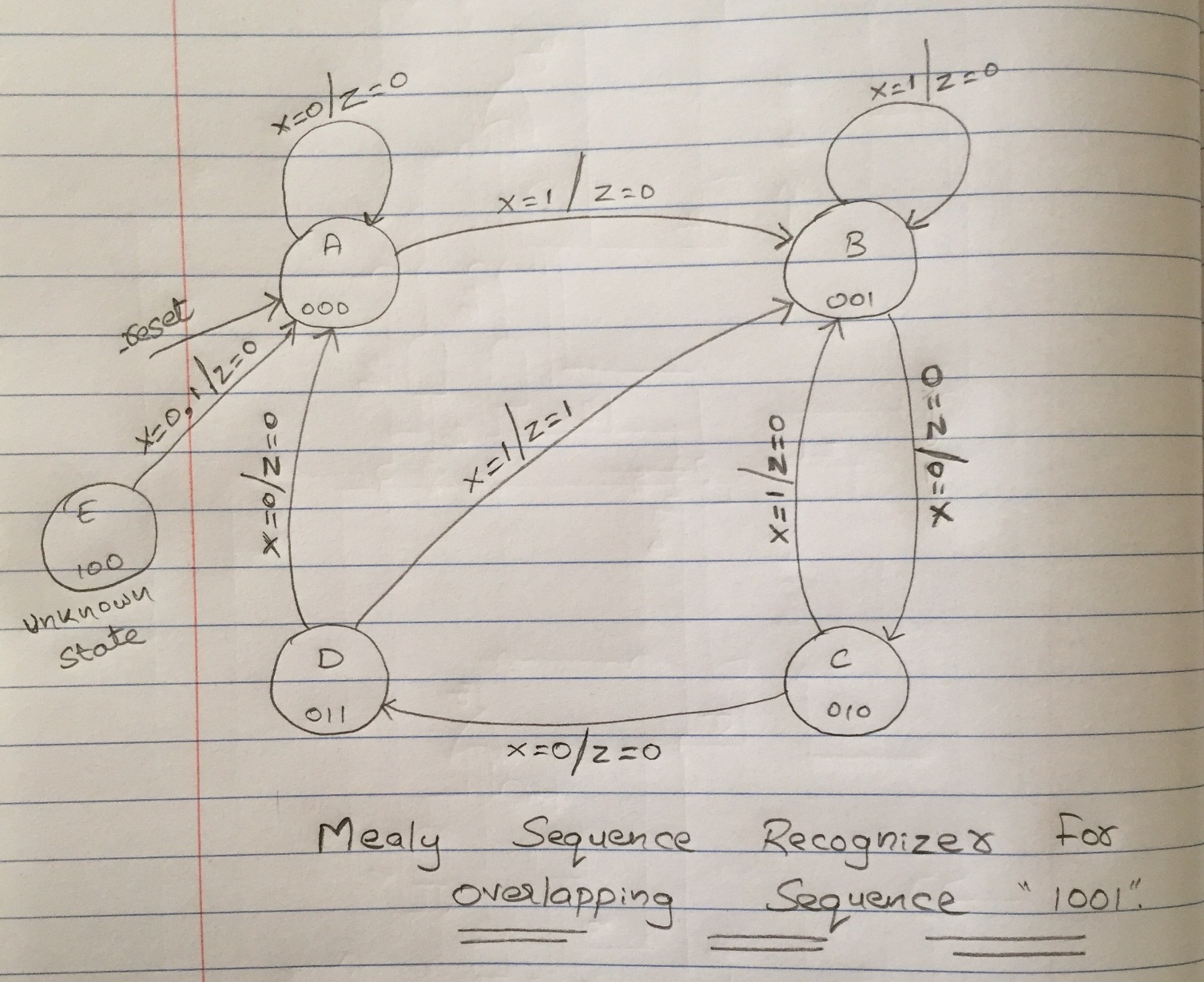
endmodule

**Output:**

****

**2. Considered Problem 5.11**

**a. Mealy’s Sequence Recognizer for overlapping sequence 1001**

****

module MealyProject ( input clock,reset,x, output reg z );

//assign binary encoded codes to the states A through D

parameter A = 3'b000, B = 3'b001, C = 3'b010, D = 3'b011;

reg[2:0] current\_state, next\_state;

//Section 1: Next State Generator (NSG) always@(\*) begin casex (current\_state) // ignore unknown and high impedence (Z) inputs A: if(x == 1) next\_state = B; else next\_state = A; B: if(x == 1) next\_state = B; else next\_state = C; C: if(x == 1) next\_state = B; else next\_state = D; D: if(x == 1) next\_state = B; else next\_state = A; default: next\_state = 3'bxxx; endcase

end

//Section 2: Output Generator (OG)

always@(\*) begin if((x == 1) && (current\_state == D) && (next\_state == B )) z = 1'b1; else z = 1'b0; end

//Section 3: FlipFlop

always@(posedge clock,posedge reset) begin if(reset ==1) current\_state <= A; else current\_state <= next\_state; end

endmodule

**b. TestBench (MealyTestBench.v)**

`include "MealyProject.v";

module MealyTestBench(); reg clock,reset,x; wire z;

MealyProject projectTwo(clock,reset,x,z);

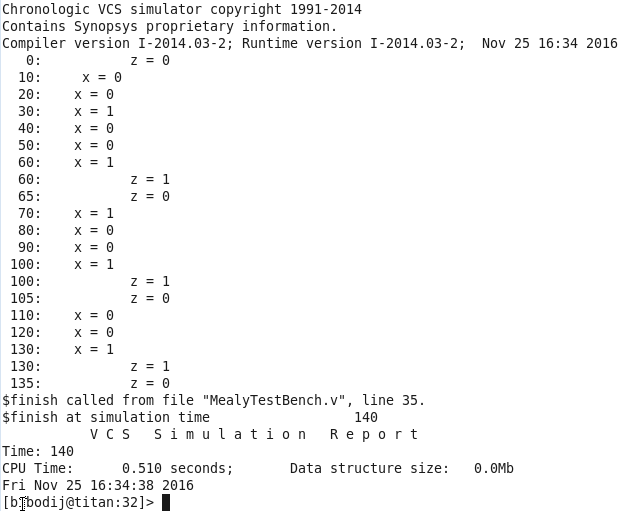
initial begin $monitor("%4d: z = %b",$time,z); clock=0; reset=1; //Resets the FlipFlop x=0; #10 reset=0; //end reset end

always begin #5clock=~clock; //generates a clock signal with period 10 end

initial begin // One input per clock cycle #10 x = 0; $display ("%4d: x = %b",$time,x); #10 x = 0; $display ("%4d: x = %b",$time,x); #10 x = 1; $display ("%4d: x = %b",$time,x); #10 x = 0; $display ("%4d: x = %b",$time,x); #10 x = 0; $display ("%4d: x = %b",$time,x); #10 x = 1; $display ("%4d: x = %b",$time,x); #10 x = 1; $display ("%4d: x = %b",$time,x); #10 x = 0; $display ("%4d: x = %b",$time,x); #10 x = 0; $display ("%4d: x = %b",$time,x); #10 x = 1; $display ("%4d: x = %b",$time,x); #10 x = 0; $display ("%4d: x = %b",$time,x); #10 x = 0; $display ("%4d: x = %b",$time,x); #10 x = 1; $display ("%4d: x = %b",$time,x); #10 $finish; end

endmodule

**Output**

****