AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING

Computer and Systems Engineering Department
2nd Year, Electrical Engineering

1st Semester, 2018/2019 **Course Code: *CSE 241***

Logic Design

Assignment 4: (4 Marks)

In this assignment you are required to create a non-overlapping sequence detector for the last 3 digits in your bench number encoded as BCD using d-type flipflop.

Example: if your BN was 23871 then you should detect the sequence 871 encoded as BCD "100001110001". If your BN was 23001 you need to detect "000000000001" not "0001".

Methodology:

We will use a certain syntax for the deliverables. The syntax is called Verilog. Don't waste your time learning this syntax. We will only need what is written in this document to finish the project.

You can use the following link to verify that you have no syntax error.

https://www.tutorialspoint.com/compile_verilog_online.php

If you want to simulate the circuit and check its waveform:

- Use https://www.edaplayground.com
- Select "`Icarus Verilog 0.9.7`" from under tools and simulators.
- Select "Open **EPWave** after run"
- Add signals from "Get signals" button and use seq_tb for scope.

Structure:

The following is the general structure you are going to deliver

```
module seq_{BN}(output out, input seq, input clk, input clear);

/*
This is a block comment. Don't forget to replace {BN} with your BN
*/

gatename(output_wire_name, input_wire_name1, input_wire_name2, …) ;

gatename(output_wire_name, input_wire_name1, input_wire_name2, …) ;

…

endmodule
```

You will only need to change the line between module and endmodule.

Warning: Don't change module signature or you will not be graded.

AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING

Computer and Systems Engineering Department
2nd Year, Electrical Engineering

1st Semester, 2018/2019 **Course Code: *CSE 241***

Logic Design

Available Gates:

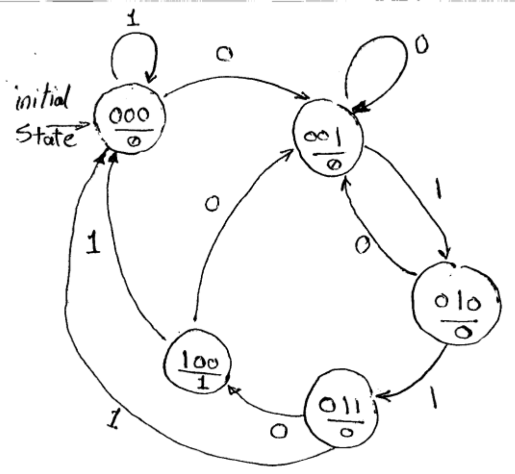
- You are going to use the following gates:
    - "and" it will have one output and can have as much input as needed.
      Ex: and(y3, y1, y2) is 2 input and gate (y1, y2) with one output (y3)
      Ex: and(y3, y1, y2, y0) is 3 input and gate (y1, y2, y0) with one output (y3)
    - "or" same as "and"
    - "xor" same as "and"
    - "not" and used like not(y1, y2) that is an inverter gate with input (y2) and output (y1)
    - "dff" this module is included in the appendix.
      Ex: dff dff1 (q1, d1, clk, reset)
      That is a dff named dff1 with output q1, input d1, clock clk and an async reset signal.

Steps:

1. You will draw the state diagram listing all transitions and states as you would normally do.
2. Fill the truth table. Don't forget the state reduction step!
3. Minimize the Boolean expressions for the states and output. You can use logic Friday to minimize the expression.
4. Draw the gates on a paper and give each wire a name like y{wire_number} like y1,y2,…
   Except for the ones already defined. The input will be called seq, the output seq.
   The other two inputs are for the dff module. (clk for clock and clear for async reset)
5. Write the code in form of gates and dff.

Example:

To make the example easy, we will consider making a sequence detector for the pattern 0110.



Y_2	Y_1	Y_0	Seq	Y_2^*	Y_1^*	Y_0^*	Out
0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	
0	0	1	0	0	0	1	0
0	0	1	1	0	1	0	
0	1	0	0	0	0	1	0
0	1	0	1	0	1	1	
0	1	1	0	1	0	0	0
0	1	1	1	0	0	0	
1	0	0	0	0	0	1	1
1	0	0	1	0	0	0	
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	

Out

Y_0 \ Y_2Y_1			11	10
	0	0	X	1
	0	0	X	X

$$Out = Y_2$$

Y_2^*

$Y_0\,Seq$ \ Y_2Y_1		01	11	
	0	0	X	0
	0	0	X	0
	0	0	X	X
10	0	1	X	X

$$Y_2^* = Y_1 Y_0 \overline{Seq}$$

Y_1^*

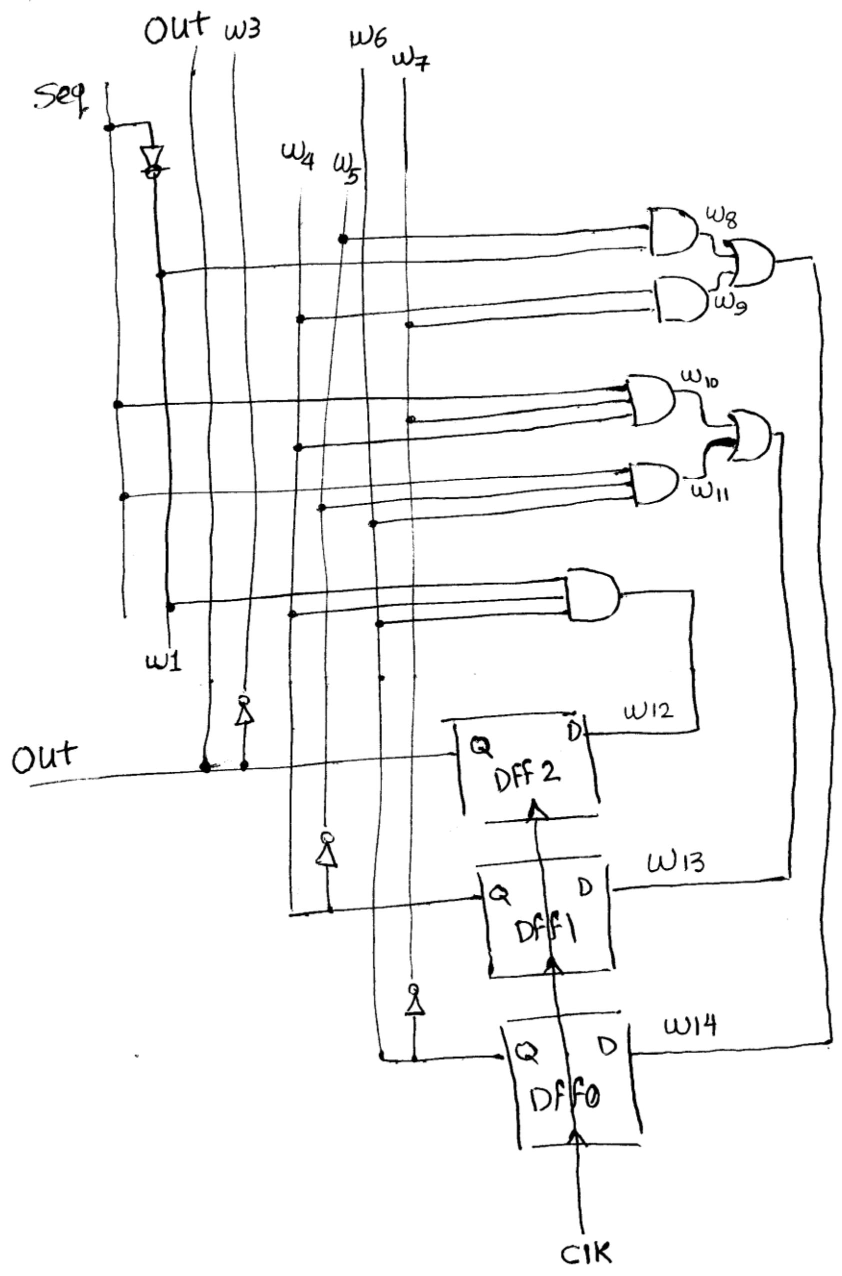
$Y_0\,Seq$ \ Y_2Y_1	00	01	11	10
	0	0	X	0
01	0	1	X	0
11	1	0	X	X
	0	0	X	X

$$Y_1^* = Y_1 \overline{Y_0}\, Seq + \overline{Y_1}\, Y_0\, Seq$$

Y_0^*

$Y_0\,Seq$ \ Y_2Y_1		01	11	10
00	1	1	X	1
01	0	1	X	0
11	0	0	X	X
10	1	0	X	X

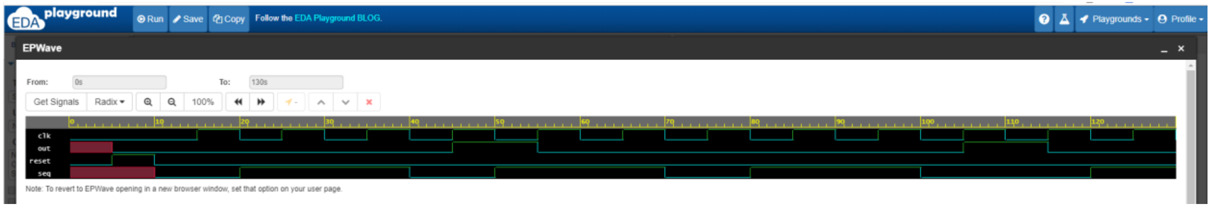
$$Y_0^* = \overline{Y_1}\,\overline{Seq} + Y_1 \overline{Y_0}$$


Out
w3
w6
w7
Seq
w4
w5
w8
w9
w10
w11
w1
w12
Out
Q
D
DFF 2
w13
Q
D
DFF1
w14
Q
D
DFF0
Clk

And the resulting code:

```
module seq_23871(output out, input seq, input clk, input clear);


dff dff2(out,w12,clk,clear);

dff dff1(w4,w13,clk,clear);

dff dff0(w6,w14,clk,clear);


not(w3,out);

not(w5,w4);

not(w7,w6);

not(w1,seq);


or(w14,w8,w9);

or(w13,w10,w11);


and(w12,w1,w4,w6);

and(w11,seq,w5,w6);

and(w10,seq,w4,w7);

and(w9,w4,w7);

and(w8,w1,w5);


endmodule
```

And a screenshot of the simulated result when using the pattern 011011011001



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

Notice:

- We must have the output wire named as “out” and input wire as “seq”
- For a pattern as 01101 we should go in the following state path S0->S1->S2->S3->S4->S0 (non-overlapping)
- For a pattern as 01100 we should go in the following state path S0->S1->S2->S3->S4->S1 (Doesn’t have to go to state 0 after the pattern or while in pattern)

Restrictions:

- You will encode the states either using Binary or Gray Code. Don’t use 1 Hot or 1 Cold.
- Your initial state MUST be encoded as 0000.
- You will follow some certain naming convention, or you will not be graded.
    - You will submit 3 files on https://goo.gl/forms/PYzYbKBjYVbL7MeX2
        - seq_{BN}.v containing the code. Ex: seq_23871.v
        - state_{BN}.png containing the states and the transition. (Hint: use draw.io website)
        - design_{BN}.png containing the designed gates with the dff showing.
- You will design the state machine as Moore finite state machine.

Logic Design

Appendix:

- Attached the D-type flip flop code in case you wanted to simulate the result. **Don't** include this in your submission.

```
module dff (output reg q, input d, input clk, input reset);
always @ (posedge clk or posedge reset)
begin
  if (reset == 1)
     q <= 0;
 else
     q <= d;
end
endmodule
```

AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING

Computer and Systems Engineering Department
2nd Year, Electrical Engineering

1st Semester, 2018/2019 **Course Code: *CSE 241***

Logic Design

- Attached a simple testing module (test bench) for simulation. You will replace the binary code with your BCD sequence and run the simulation. **Don't** include this in your submission.

```
module seq_tb;
   reg reset, clk, seq;
   wire out;
   integer index;
// Replace 100001110001 with your pattern
   reg[0:11] pattern = 12'b100001110001;
// Replace 23871 with your BN
   seq_23871 seq_det(out, seq, clk, reset);
   initial begin
      $monitor("%d,\t reset=%b,\t clk= %b, \t seq=%b, \t out=%b ",$time, reset,clk, seq, out);
      $dumpfile("dump.vcd"); $dumpvars;
      clk<=0; reset<=0;
      #5 reset<=1;
      #5 reset<=0;
      for(index = 0; index < 12; index = index + 1) begin
        seq <= pattern[index];
        #5 clk = ~clk;
        #5 clk = ~clk;
      end
      $finish();
   end
endmodule
```