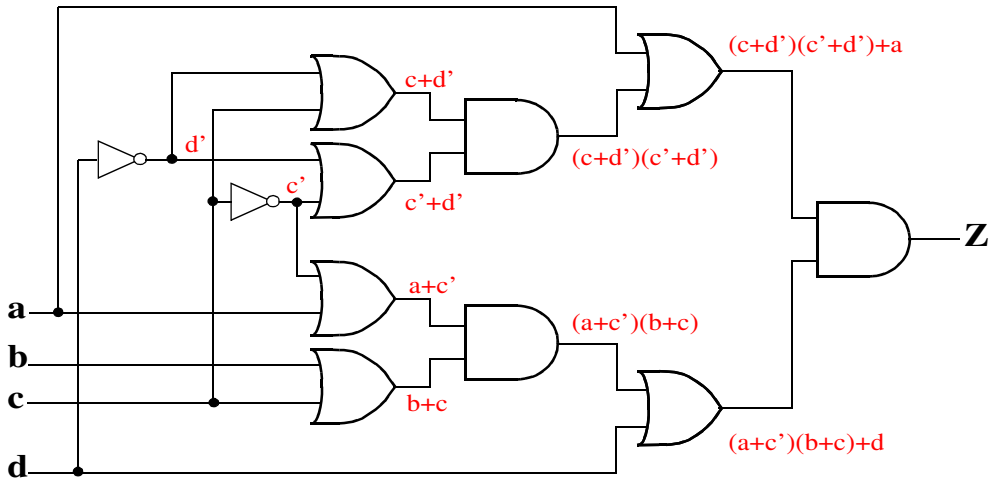
CPE 322 **EXAM I** **Name** KEY

Fall Semester 2014

Work should be performed systematically and neatly with the final answer being underlined. This exam is closed book, closed notes, closed neighbor. Allowable items on desk include: exam, pencils, and pens. All other items must be removed from student's desk. Students have approximately 90 minutes (1 1/2 hours) to complete this exam. Best wishes!

1. [12.5 points] For the network shown below, find all static 0-hazards. For each 0-hazard found, specify the conditions which will cause the hazard to appear at the output (i.e. specify the logic values of the variables which are constant and clearly specify the variable that is assumed to be changing). If there are no 0-hazards found, use a K'map to show why this is the case.



$Z = [(c+d')(c'+d')+a][(a+c')(b+c)+d]$

applying second distributive law
$W + XY = (W+X)(W+Y)$

$Z = [(a+c+d')(a+c'+d')][(a+c'+d)(b+c+d)]$

removing extra parentheses

$Z = (a+c+d')(a+c'+d')(a+c'+d)(b+c+d)$

0-Hazards

hazard #1
c changes 0→1 or 1→0 a=0, b=0, d=0

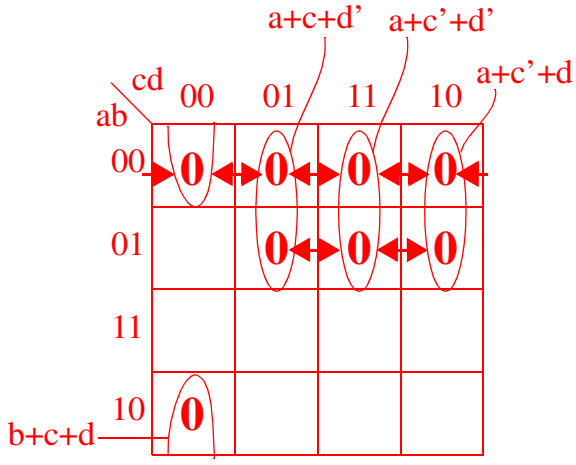
hazard #2
d changes 0→1 or 1→0 a=0, b=0, c=0

hazard #3
c changes 0→1 or 1→0 a=0, b=0, d=1

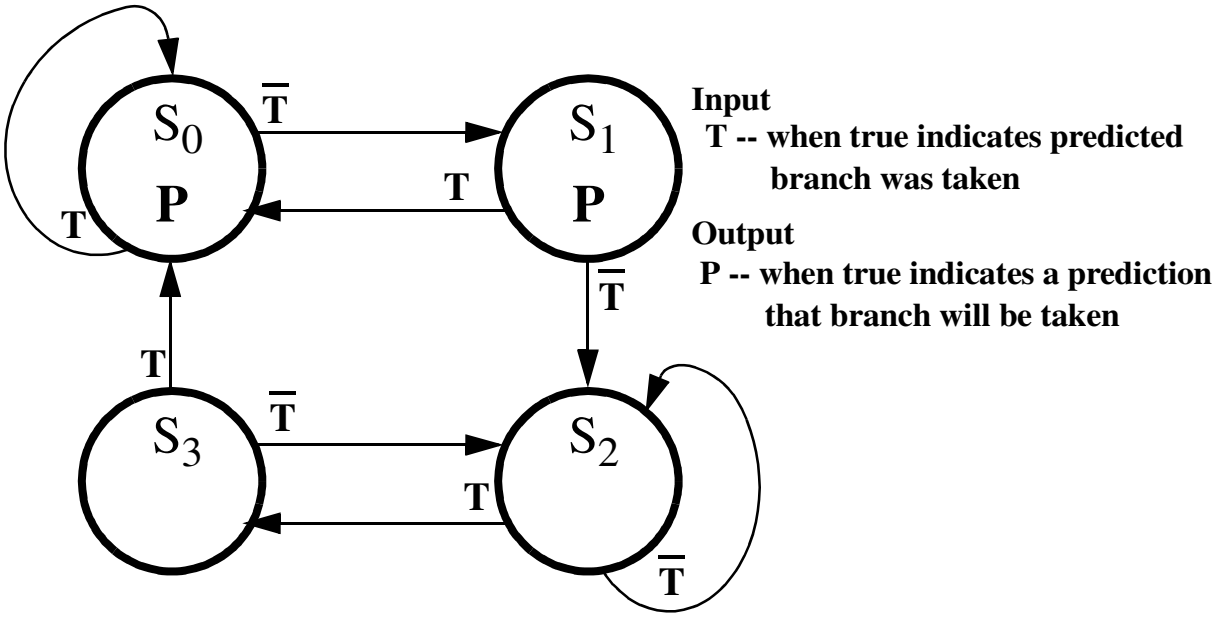
hazard #4
d changes 0→1 or 1→0 a=0, b=0, c=1

hazard #5
c changes 0→1 or 1→0 a=0, b=1, d=1

hazard #6
d changes 0→1 or 1→0 a=0, b=1, c=1



2. [12.5 points] Below is an extended state transition graph for a finite state machine for a simple branch prediction unit of a super scalar CPU implementation.



a) What type of synchronous finite state machine is this? Explain the reason(s) why this is the case.

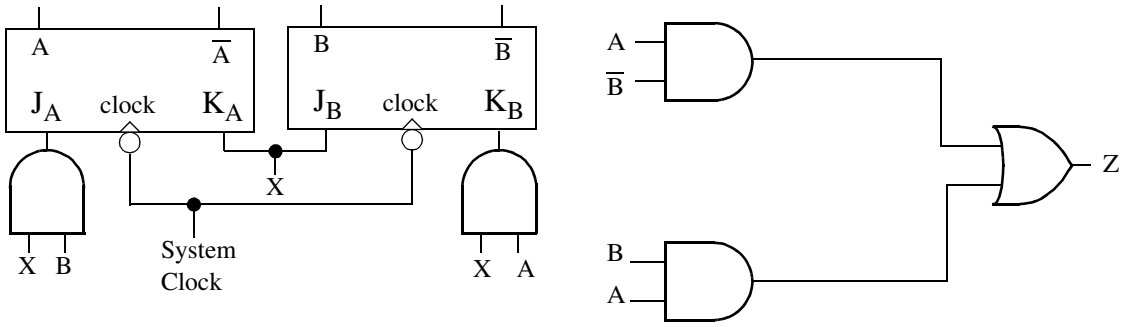
This is a Moore type Synchronous network because the output P is associated with the state itself (not the transition) which is a major characteristic of a Moore network.

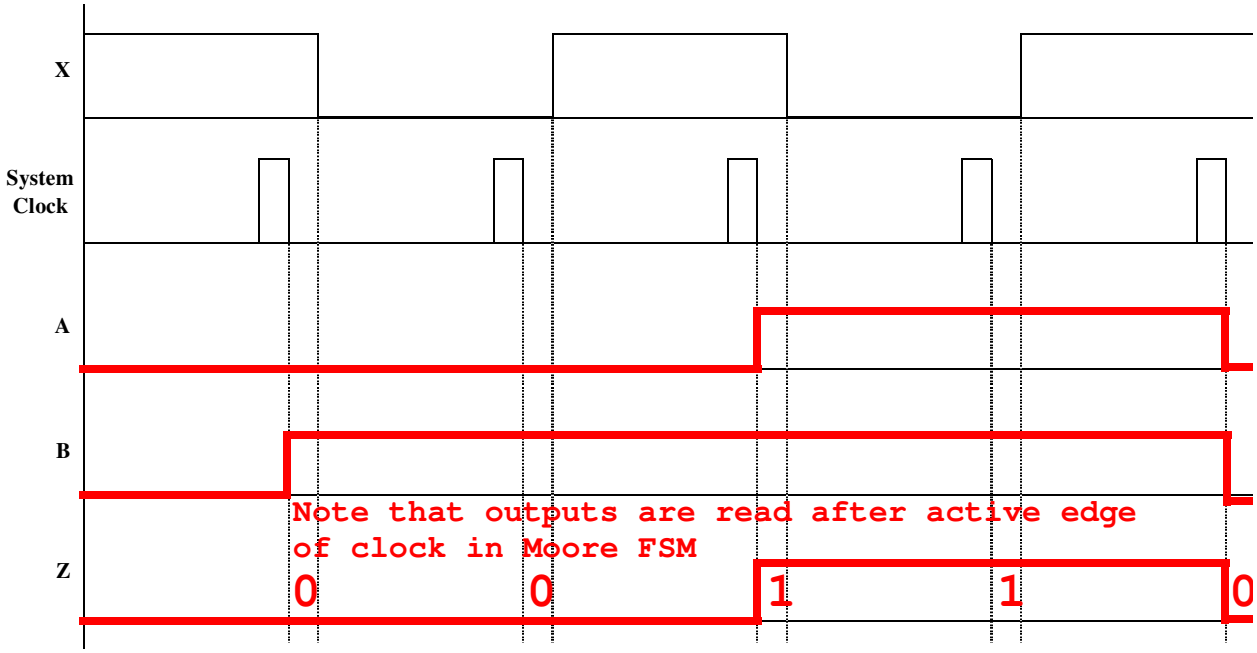
b) Write a behavioral Verilog HDL model that implements the state transition graph. In your model include a synchronous active high ***reset*** input signal that will always place the design in State S_0 on the active edge of the next clock pulse regardless of the current state of the network.Assume an active low clock in your Verilog Implementation.

```
module problem2(input clock, reset, T,
                output reg P);
   reg [1:0] state=0;
   reg [1:0] nextstate;


   // on negative edge of clock
   // go to next state
   always @ (negedge clock)
      state = nextstate;

   // for Moore Network Output is only
   // a function of the input
   always @ (state)
      if (state==0 || state == 1)
         P=1; // for states 0 & 1
      else
         P=0; // for states 2 & 3
```

```
   // Next State is a function of
   // current state and the input T
   // Set Next State to new value
   // whenever current state or an input
   // changes value
   always @ (T,state)
     case (state)
        0: if (T) nextstate=state;
           else nextstate=1;
        1: if (T) nextstate=0;
           else nextstate=2;
        2: if (T) nextstate=3;
           else nextstate=2;
        3: if (T) nextstate=0;
           else nextstate=2;
      endcase

endmodule
```

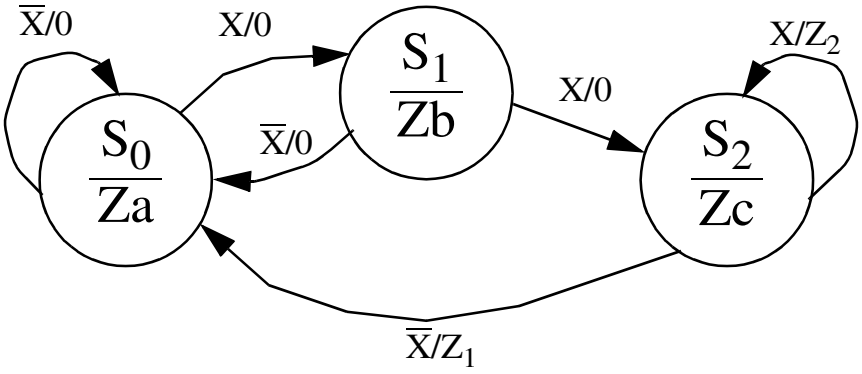
3. [12.5 points] Complete the following timing diagram for signals A, B, and Z for the network shown below assuming that all setup and hold times for the flip-flops have been met and all propagation delays through the gates and flip-flops are negligible (i.e. zero). Also assume that the two flip-flops are clocked on the falling edge of the system clock and are in the reset state (i.e. A='0' and B='0') at the beginning of the timing diagram.





What type of sequential synchronous network is this?
Moore -- output depends only on the current state which are represented by flip/flop A and B.
Write down the output sequence for Z.
Z= {0,0,1,1,0}
Are there any 'false' outputs on the timing diagram? If so clearly identify them.
No 'false' outputs possible in a Moore FSM network

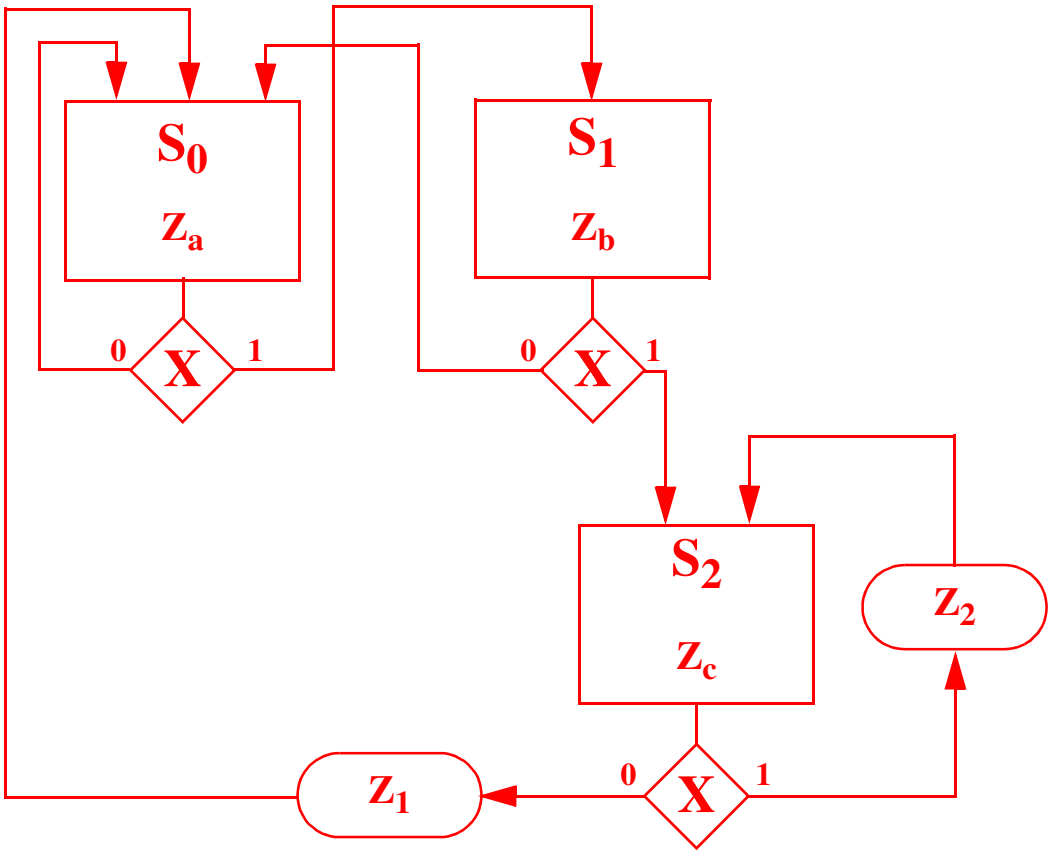
4.> [10 points} For the following Extended State Transition Graph shown below.



a) Identify the type of synchronous sequential network it represents.

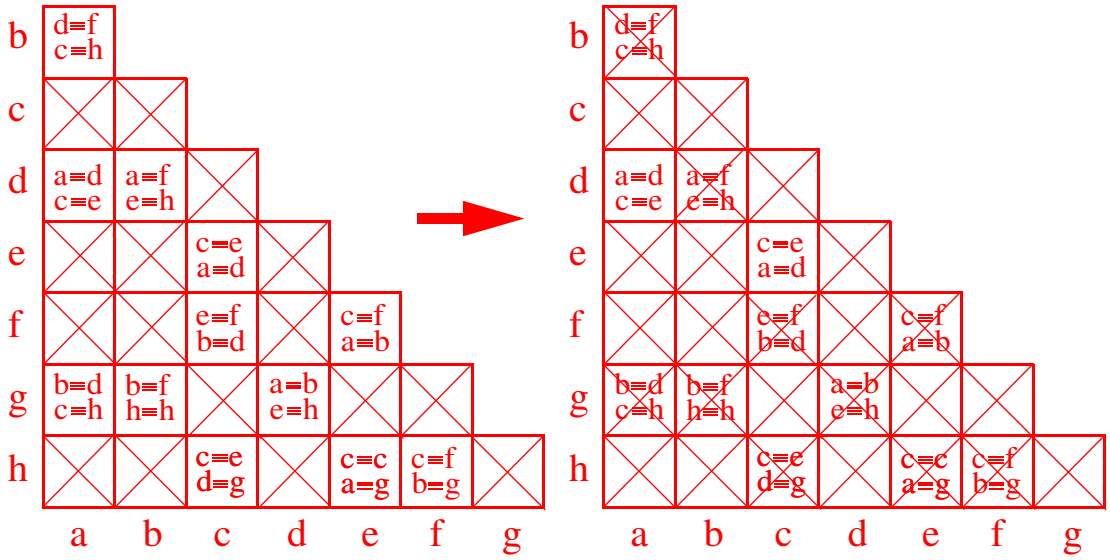
This is a hybrid Moore and Mealy network. The Outputs Za, Zb, and Zc are associated with the state which is a characteristic of a Moore FSM. The outputs Z_1 and Z_2 are associated with the state transitions which is a characteristic of a Mealy network.

b) Create an equivalent Algorithmic State Chart Representation.



5. [10 points] Reduce the following state table to a minimum number of states clearing identifying the states that are equivalent with one another. Show the final reduced state table.

present state	next state X=0	1	present output
a	d	c	0
b	f	h	0
c	e	d	1
d	a	e	0
e	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1



Equivalent States:

$a \equiv d$

$c \equiv e$

Reduced State Table

present state	next state X=0	1	present output
a	a	c	0
b	f	h	0
c	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1

6. [5 points] True/False. Circle the Correct Answer.

a) Verilog is case sensitive.
(**True** or **False**)

b) Verilog' synthesizers treat the white space ' ' and carriage returns differently.
(**True** or **False**)

All white space including carriage returns are ignored in Verilog

c) The use of a tick timescale ('timescale) enables code to be synthesized with the specified delays.
(**True** or **False**)

Synthesizers cannot create circuits that have specified delays 'timescale useful in simulation not synthesis

d) There can be only one *always* block in a module.
(**True** or **False**)

Many cases it is desirable to have multiple always blocks!

e) In Verilog, Behavioral implementations generally give the designer the most control on how the module will actually be implemented during synthesis process.
(**True** or **False**)

Designer works at a higher level of abstaction allowing the greatest flexibility to the synthesis process to determine exactly how the circuitry is to be implemented

7 [5 Points] Short Answer: What is Logic Synthesis? What is Digital Simulation? How do they differ from one another?

Logic Synthesis is the process by which an abstract form of a design is turned in to an actual low-level hardware implementation. Simulation is the process by which the functionality of the design can be demonstrated on a computer without actually implementing it in hardware. Synthesis creates the actual hardware whereas simulation is used to observe its operation often before the hardware is created allowing the user to test various functional and timing scenarios and identify destructive conditions before hardware if fabricated.

8 [7.5 points] Short Answer: What is the difference between a *blocking* and *non-blocking procedural assignment statement* in Verilog. How can you tell them apart? In what subsection of a Verilog model would these constructs be used?

If a current statement contains a blocking procedural assignment then the next statement will be executed after the execution of the current statement. If a current statement contains a non-blocking procedural assignment then the next statement will be executed at the same time.
Blocking procedural assignment statements use the '=' operator. Nonblocking procedural assignment statements use the '<=' operator.
Procedural assignment statements can be placed in any procedural section including initial, always, tasks, and functions.

9 [7.5 points] Short Answer: Does the order of a *continuous assignment statements* make a difference in Verilog? Why or why not?

The order that the programmer places continuous assignment statements in a verilog file does not matter. This is because continuous assignment statements are used to model circuit elements that operate in parallel. Just because one is placed before another does

How do *continuous assignment statements* differ from *procedural assignment statements?*

Continuous assignment can only drive signals that are of a net types (i.e. wire or tri. They must appear outside of the procedural regions (outside always, initial, functions, tasks, etc.) Continuous assignments executes each time signals on the right hand side change value. A maximum of one continuous assignment should exist for each wire should be assigned. Continuous assignments are used to model combinational logic. Procedural assignment statements allow for the modeling of more complex operations at a higher level of abstraction. Procedural Assignment can drive reg, integer, real and time data type. They must appear in the procedural regions (such as initial and always) of the code. They easily model both combinational and sequential logic. Multiple assignments are possible to the same reg variable with procedural logic.

10 [7.5 points} Short Answer: What is the difference between *inertial* and *transport* delays? What delay aspects of real digital implementations are best modeled by each of these.

In inertial delays a signal will assume its new value after the specified propagation delay time if the signal is in the same state at least as long as this delay. If not the change of value for this signal does not occur.

In transport delays a signal will assume its new value after the specified delay no matter how the signal will be in this state.

Inertial delays are used to model gate delays where the capacitive effects tend to filter out short glitches and transport delays model delays in long wires due to routing effects which tend to pass the signal through without low-pass filtering effects.

Model in Verilog HDL a three input AND gate with a 5 ns rise time and a 3 ns fall time using inertial delays.

```
`timescale 1 ns / 100 ps

module and_gate(output O, input A,B,C);
   assign #(5,3) O = A & B & C;
endmodule
```

Using a Continuous Assignment Statement

```
`timescale 1 ns / 100 ps

module and_gate(output O, input A,B,C);
   and #(5,3) G1 (O,A,B,C);
endmodule
```

Using a Structurally Instatiated Primitive

11 [5 points] Develop a valid Verilog Model for a 4-to-1 Multiplexer where the general inputs represent a 4 bit bus and the output is a single signal. Label the inputs **I,S** and the output **O.**

```
module mux_4_1(input [1:0] S, input [3:0] I, output reg O);
   always @(S,I)
      O = I[S];
endmodule
```

12 [5 points] Develop a valid Verilog Model for a simple rising edge D Flip/Flop. Label the inputs, **CLK, D** and the output **Q.**

```
module d_ff(input CLK, D, output reg Q);
   always @(posedge CLK)
      Q <= D;
endmodule
```