CPE/EE 422/522, Fall 2012

Exam II -- Take Home Portion

INSTRUCTIONS: **Work in a clear and detailed manner on separate paper all parts of the problem given on this exam. You may use any resources available including class notes, textbooks, internet and research articles but you are not to employ any outside help from any individual. Students who violate this rule will be considered to be guilty of academic misconduct as defined in the UAH Student Handbook.**

The elevator of a four story building can be directed to stop on any of the floors (1 through 4) by pressing the appropriate button on the control panel which is located inside the elevator. The four buttons on the control panel are connected to a special binary encoder which produces the two bit binary code that corresponds to the button being pressed (see Figure 1) and an enabling clock pulse that is a logic high for exactly one clock cycle. The binary encoder logic is designed so that it only accepts the first button that is pressed (i.e. it ignores all buttons pressed until it reaches the desired floor) and in the case where multiple buttons are pressed simultaneously, the highest floor button has precedence. The binary encoder connects to the elevator control logic in the manner shown in the figure and produces the three-bit two's complement differential output (defined in the actuator value table of Figure 1) that is needed by the elevator actuator electronics. The differential output is used to direct the elevator from the current floor to the selected floor by outputting the binary code which indicates in relative terms how many floors the elevator needs to travel up or down to get to the desired floor. When the elevator is in the process of traveling to a given floor the elevator actuator electronics logic disables the binary encoder using the *lockout* signal so that it does not respond to the pressing of any additional buttons until after it arrives at the specified floor at which point it is reenabled.

You are to design the differential elevator control logic for this elevator by modelling it as a Mealy sequential network.

a) To do this first, develop a complete SM Chart diagram. You may assume that all state registers have an enabling input separate from its clock. Assume that the elevator is initially at the first floor.

b) Verify the correctness of your design through the use of a *behavioral* model written in VHDL. Simulate this model using an input stimulus pattern that will take the elevator at least once to each of the floors. You should assume a rising edge clock.

c) Implement your design using D Flip Flops and basic combinational logic elements of your choosing (i.e. AND, OR, NOT, NOR, and NAND gates). You should assume a rising edge clock. Minimize your state diagram. Simplify the state assignment assuming a state assignment will be made with the goal to minimize the number of flip-flops. Is this step necessary for this design? Why or why not?

d) Enter your design using either a *data flow* or *structural* VHDL model assuming your specified state assignment from part c. Simulate your design using the same set of stimulus patterns used in part b. Explain any differences with part b.

e) Without working the problem in detail, estimate how many states would be required if this problem were modeled as a Moore sequential network. Justify your answer

.

Due Date: Midnight--Thursday April 19, 2012.

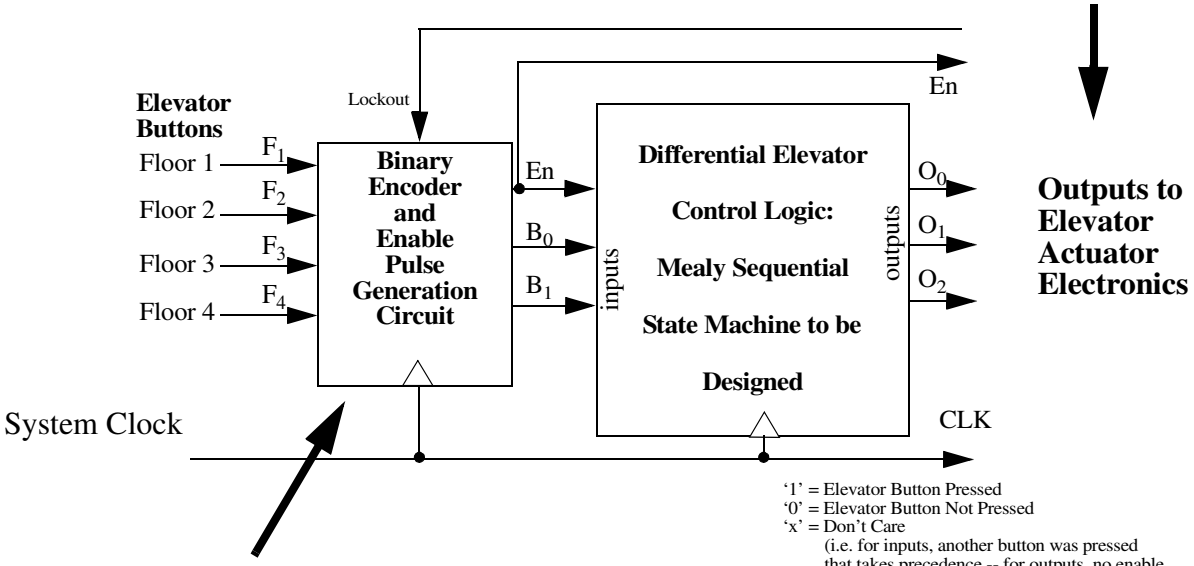
Submission to be made using your CPE/EE 422/522 Angel Dropbox.

On the same submission attach multiple files: Two VHDL files one for part b and one for part d. Also attach a single MS word file or pdf file that contains all parts (parts a,b,c,d,e) of this exam.

.

Desired Output to Differential Elevator Actuator

Operation	O_2	O_1	O_0
Up Three floors	0	1	1
Up Two Floors	0	1	0
Up One Floor	0	0	1
Current Floor (not real useful but could be used to open the door)	0	0	0
Down One Floor	1	1	1
Down Two Floors	1	1	0
Down Three Floors	1	0	1



'1' = Elevator Button Pressed
'0' = Elevator Button Not Pressed
'x' = Don't Care
(i.e. for inputs, another button was pressed that takes precedence -- for outputs, no enable pulse generated because no button was pressed)

Truth Table for Binary Encoder

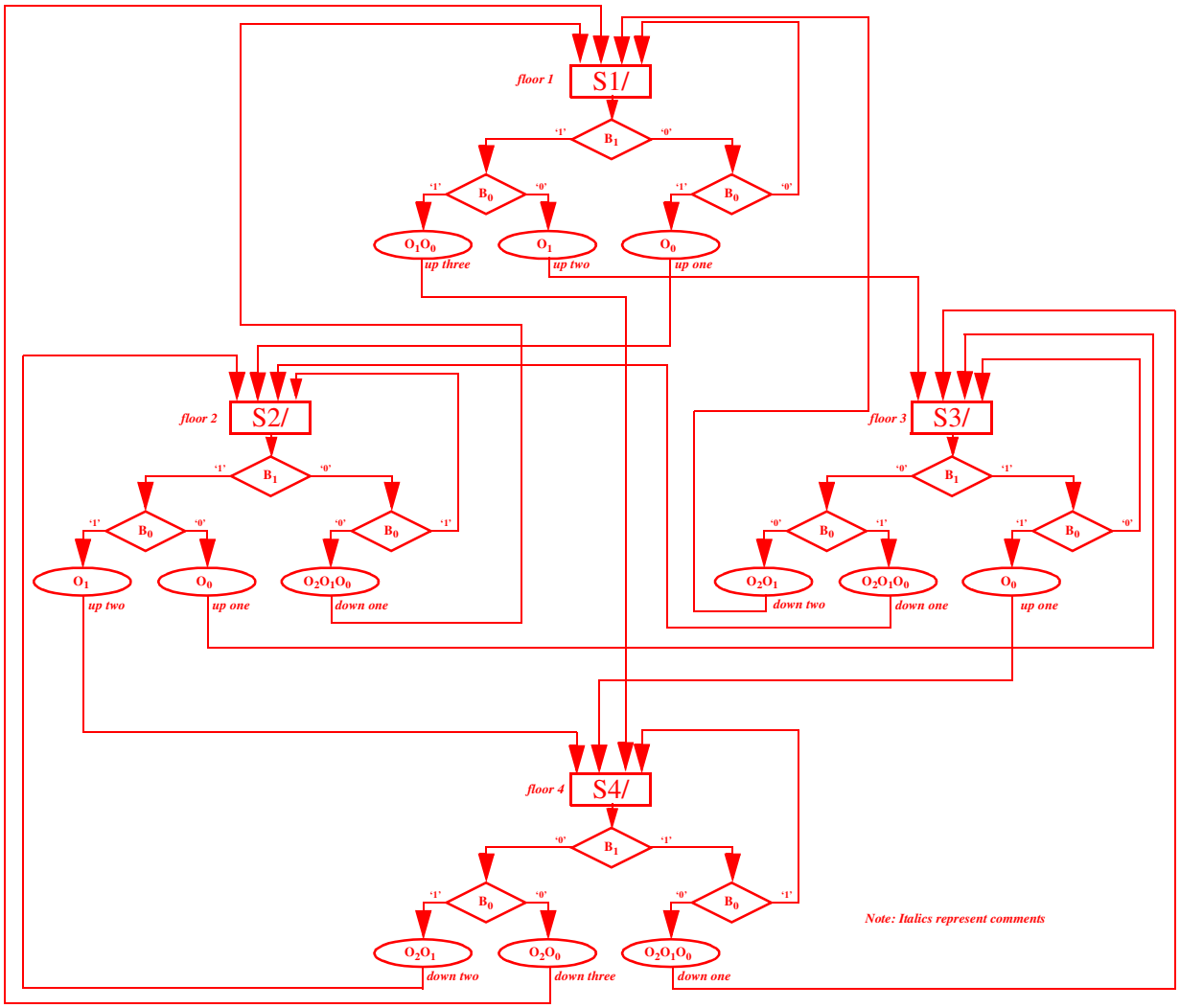
Lockout signal	Elevator Input Buttons				Binary Coded Output	
	F_4	F_3	F_2	F_1	B_1	B_0
1	x	x	x	x	x (ignores buttons)	x (ignores buttons)
0	1	x	x	x	1	1
0	0	1	x	x	1	0
0	0	0	1	x	0	1
0	0	0	0	1	0	0
0	0	0	0	0	x (no Enable pulse generated)	x (no Enable pulse generated)

Figure 1: Elevator Control Scenario

You are to design the differential elevator control logic for this elevator by modelling it as a Mealy sequential network.

a)To do this first, develop a complete SM Chart diagram. You may assume that all state registers have an enabling input separate from its clock. Assume that the elevator is initially at the first floor.

As stated in class, assuming that transitions only occur when enable is high and a valided active clock edge occurs -- for this reason enabling signal not explicitally shown in SM Chart.



b) Verify the correctness of your design through the use of a *behavioral* model written in VHDL. Simulate this model using an input stimulus pattern that will take the elevator at least once to each of the floors. You should assume a rising edge clock.

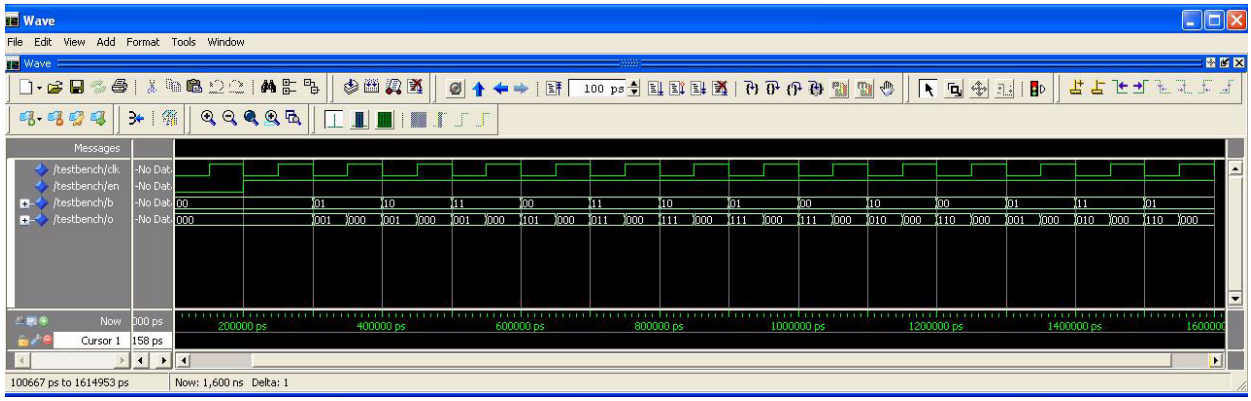
Date: April 09, 2012 testbench.vhd* Project:

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.all;
3
4   entity ELEVATOR is
5      port (CLK,EN : in STD_LOGIC;
6            B: in STD_LOGIC_VECTOR(1 downto 0);
7            O: out STD_LOGIC_VECTOR(2 downto 0));
8   end ELEVATOR;
9
10  architecture BEHAVIORAL of ELEVATOR is
11     signal STATE,NEXTSTATE: integer range 1 to 4 :=1;
12  begin
13     process(STATE,B)
14        begin
15        case STATE is
16
17           when 1 =>
18              if (B(1)='0') then
19                 if (B(0)='0') then
20                    NEXTSTATE <= 1;
21                    O <= "000";
22                 else
23                    NEXTSTATE <= 2;
24                    O <= "001";
25                 end if;
26              else
27                 if (B(0)='0') then
28                    NEXTSTATE <= 3;
29                    O <= "010";
30                 else
31                    NEXTSTATE <= 4;
32                    O <= "011";
33                 end if;
34              end if;
35
36        when 2 =>
37           if (B(1)='0') then
38              if (B(0)='0') then
39                 NEXTSTATE <= 1;
40                 O <= "111";
41              else
42                 NEXTSTATE <= 2;
43                 O <= "000";
44              end if;
45           else
46              if (B(0)='0') then
47                 NEXTSTATE <= 3;
48                 O <= "001";
49              else
50                 NEXTSTATE <= 4;
51                 O <= "010";
52              end if;
53           end if;
54
```

```
56          when 3 =>
57             if (B(1)='0') then
58                if (B(0)='0') then
59                   NEXTSTATE <= 1;
60                   O <= "110";
61                else
62                   NEXTSTATE <= 2;
63                   O <= "111";
64                end if;
65             else
66                if (B(0)='0') then
67                   NEXTSTATE <= 3;
68                   O <= "000";
69                else
70                   NEXTSTATE <= 4;
71                   O <= "001";
72                end if;
73             end if;
74
75             when 4 =>
76                if (B(1)='0') then
77                   if (B(0)='0') then
78                      NEXTSTATE <= 1;
79                      O <= "101";
80                   else
81                      NEXTSTATE <= 2;
82                      O <= "110";
83                   end if;
84                else
85                   if (B(0)='0') then
86                      NEXTSTATE <= 3;
87                      O <= "111";
88                   else
89                      NEXTSTATE <= 4;
90                      O <= "000";
91                   end if;
92                end if;
93
94             when others => null;
95          end case;
96       end process;
97
98       process(EN,CLK)
99          begin
100         if (EN='1') then
101            if (CLK='1' and CLK'event) then
102               STATE <= NEXTSTATE;
103            end if;
104         end if;
105      end process;
106
107   end BEHAVIORAL;
108
109
110
```

```
111
112
113   library IEEE;
114   use IEEE.STD_LOGIC_1164.all;
115
116   entity TESTBENCH is
117   end TESTBENCH;
118
119   architecture DATAFLOW of TESTBENCH is
120         signal CLK,EN : STD_LOGIC := '0';
121         signal B : STD_LOGIC_VECTOR(1 downto 0);
122         signal O : STD_LOGIC_VECTOR(2 downto 0);
123
124         component ELEVATOR is
125            port (CLK,EN : in STD_LOGIC;
126                  B: in STD_LOGIC_VECTOR(1 downto 0);
127                  O: out STD_LOGIC_VECTOR(2 downto 0));
128         end component;
129
130
131      begin
132
133         CLK <= not CLK after 50 ns;
134         EN <= '0' after 0 ns, '1' after 200 ns;
135         B <= "00","01" after 300 ns, "10" after 400 ns,
136                   "11" after 500 ns, "00" after 600 ns,
137                   "11" after 700 ns, "10" after 800 ns,
138                   "01" after 900 ns, "00" after 1000 ns,
139                   "10" after 1100 ns,"00" after 1200 ns,
140                   "01" after 1300 ns,"11" after 1400 ns,
141                   "01" after 1500 ns;
142
143         U1:ELEVATOR port map(CLK,EN,B,O);
144
145   end DATAFLOW;
```

ModelSim Simulation Behavioral Simulation



Note: the following waveform was used, inputs applied on falling edge, results valid right at rising edge of clock. EN signal driven to 1 at 200 ns and held there to compress simulation. This waveform should exercise all possible transitions and states which is more than what was requred of you. Note the false outputs that occur at the falling edge of the clock.

c) Implement your design using D Flip Flops and basic combinational logic elements of your choosing (i.e. AND, OR, NOT, NOR, and NAND gates). You should assume a rising edge clock. Minimize your state diagram. Simplify the state assignment assuming a state assignment will be made with the goal to minimize the number of flip-flops. Is this step necessary for this design? Why or why not?

State Table

Current State	Next State				Outputs, $O_2O_1O_0$			
	B_1B_0=00	B_1B_0=01	B_1B_0=10	B_1B_0=11	B_1B_0=00	B_1B_0=01	B_1B_0=10	B_1B_0=11
S_1	S_1	S_2	S_3	S_4	000	001	010	011
S_2	S_1	S_2	S_3	S_4	111	000	001	010
S_3	S_1	S_2	S_3	S_4	110	111	000	001
S_4	S_1	S_2	S_3	S_4	101	110	111	000

State Assignment Rules

1. States that have the same next state for a given input should be given adjacent assignments.
   (S_1,S_2,S_3,S_4) should all be adjacent -- which is impossible.
2. States which are next states to a given state should be given adjacent assignments.
   (S_1,S_2,S_3,S_4) again should all be adjacent -- which is impossible.
3. States that have the same output for a given input should be given adjacent assignments.
   no state fall into this category -- all different outputs for same input

For this reason state assignment cannot be simplified using the techniques discussed in the text.
All assignments produce equivalently complicated combinational logic.
Assuming the following state assignment encoding for minimum number of two flip-flops is Q_A,Q_B: S_1=00, S_2=01, S_3=10, and S_4 = 11. Any other encoding is equally efficient.

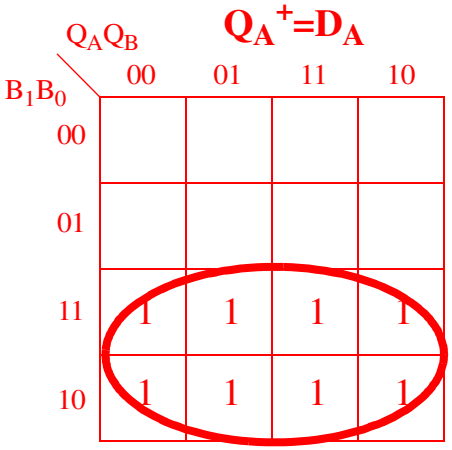
Transition Table

Current State Q_AQ_B	Next State $Q_A^+Q_B^+$				Outputs, $O_2O_1O_0$			
	B_1B_0=00	B_1B_0=01	B_1B_0=10	B_1B_0=11	B_1B_0=00	B_1B_0=01	B_1B_0=10	B_1B_0=11
00	00	01	10	11	000	001	010	011
01	00	01	10	11	111	000	001	010
10	00	01	10	11	110	111	000	001
11	00	01	10	11	101	110	111	000

Truth Table for Combinational Logic

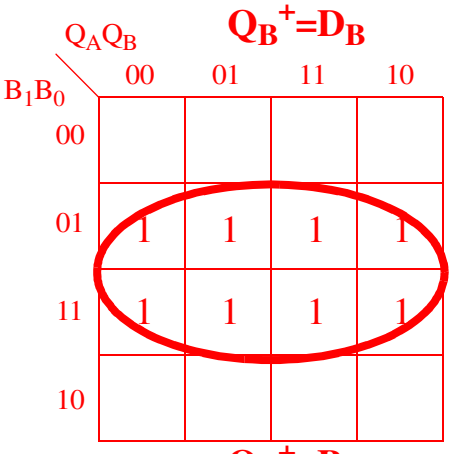
Inputs				Outputs				
Q_A	Q_B	B_1	B_0	$Q_A^+=D_A$	$Q_B^+=D_B$	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	1
0	0	1	0	1	0	0	1	0
0	0	1	1	1	1	0	1	1
0	1	0	0	0	0	1	1	1
0	1	0	1	0	1	0	0	0
0	1	1	0	1	0	0	0	1
0	1	1	1	1	1	0	1	0
1	0	0	0	0	0	1	1	0
1	0	0	1	0	1	1	1	1
1	0	1	0	1	0	0	0	0
1	0	1	1	1	1	0	0	1
1	1	0	0	0	0	1	0	1
1	1	0	1	0	1	1	1	0
1	1	1	0	1	0	1	1	1
1	1	1	1	1	1	0	0	0

$Q_A^+ = D_A$



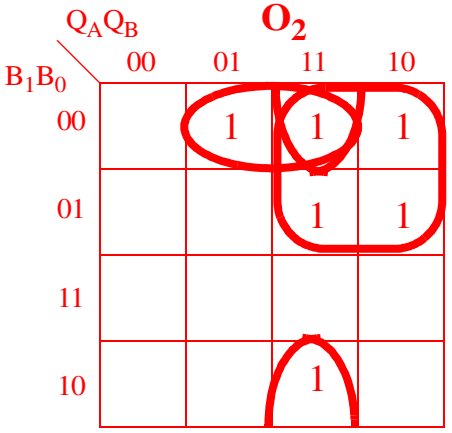
$Q_A^+ = B_1$

$Q_B^+ = D_B$



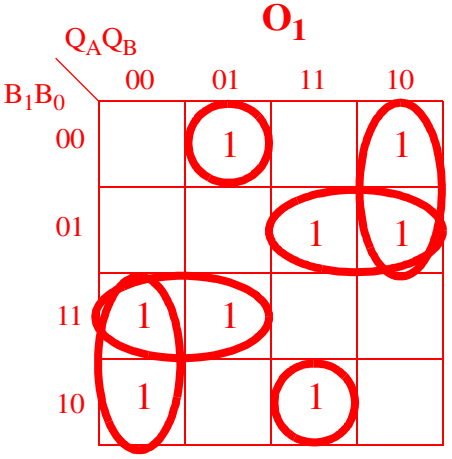
$Q_B^+ = B_0$

O_2



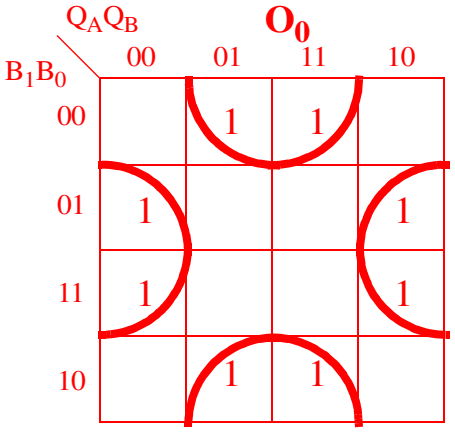
$O_2 = Q_A\overline{B}_1 + Q_AQ_B\overline{B}_0 + Q_B\overline{B}_1\overline{B}_0$

O_1



$O_1 = Q_A\overline{Q}_B\overline{B}_1 + Q_A\overline{B}_1B_0 + \overline{Q}_AQ_B\overline{B}_1\overline{B}_0 + \overline{Q}_A\overline{Q}_BB_1 + \overline{Q}_AB_1B_0 + Q_AQ_BB_1\overline{B}_0$

O_0



$O_0 = \overline{Q}_BB_0 + Q_B\overline{B}_0$

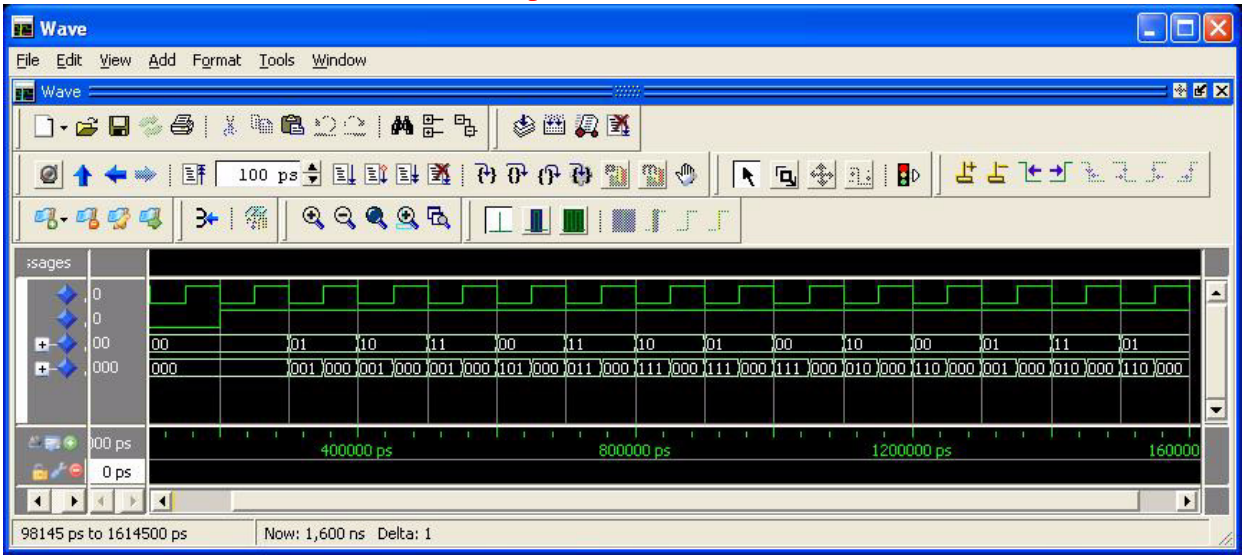
d) Enter your design using either a *data flow* or *structural* VHDL model assuming your specified state assignment from part c. Simulate your design using the same set of stimulus patterns used in part b. Explain any differences with part b.

Date: April 12, 2012 testbench.vhd* Project:

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.all;
3
4   entity ELEVATOR is
5      port (CLK,EN : in STD_LOGIC;
6            B: in STD_LOGIC_VECTOR(1 downto 0);
7            O: out STD_LOGIC_VECTOR(2 downto 0));
8   end ELEVATOR;
9
10  architecture DATAFLOW of ELEVATOR is
11     signal QA,QB : STD_LOGIC := '0';
12  begin
13
14    O(0) <= (not QB and B(0)) or (QB and not B(0));
15    O(1) <= (QA and not QB and not B(1)) or (QA and not B(1) and B(0))
       or
16            (not QA and QB and not B(1) and not B(0)) or
17            (not QA and not QB and B(1)) or (not QA and B(1) and B(0))
       or
18            (QA and QB and B(1) and not B(0));
19    O(2) <= (QA and not B(1)) or (QA and QB and not B(0)) or
20            (QB and not B(1) and not B(0));
21
22     process(EN,CLK)
23        begin
24        if (EN='1') then
25           if (CLK='1' and CLK'event) then
26              QA <= B(1);
27              QB <= B(0);
28           end if;
29        end if;
30     end process;
31
32  end DATAFLOW;
```

Note: same stimulus testbench used as in behavioral simulation

ModelSim Simulation of Data Flow Implementation -- results identical to behavioral sim.



e) Without working the problem in detail, estimate how many states would be required if this problem were modeled as a Moore sequential network. Justify your answer.

In a Moore network the outputs are a function of only the state. Thus the states would have to represent both the current floor and the output that had to be supplied to the elevator actuattors to get to that floor. Since there are 3 other floors for each floor represented from which the elevator could have come from this means that the Moore machine would have required a total of 3 x 4 = 12 states (not counting the don't care situation when you press the same floor that you are on). The output of the Moore network would be taken after the clock pulse not before as in the Mealy case.