1. (20 points) Write a VHDL entity specification for the 1 bit comparator component from problem 8 of homework 4, along with a behavioral specification of the logic. Simulate your component for all input combinations.

Write a VHDL entity specification for a four bit comparison circuit and a corresponding architecture. Your circuit should use four copies of the 1 bit comparator component. Use structural VHDL to combine these four components to create the comparison circuit. Simulate your design for the input values specified in problem 8 of homework 4.

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity vcompar1 is
port (x, z: in STD_LOGIC;
      Gi, Li, Qi: in STD_LOGIC;
      Go, Lo, Qo: out STD_LOGIC);
end vcompar1;

architecture vcompar1_arch of vcompar1 is
begin
    Qo <= Qi and (x xnor z);
    Go <= Gi or (Qi and x and (not z));
    Lo <= Li or (Qi and (not x) and z);
end vcompar1_arch;

library IEEE;
use IEEE.std_logic_1164.all;

entity vcompar4 is
port (x, z: in STD_LOGIC_VECTOR(3 downto 0);
      Gi, Li, Qi: in STD_LOGIC;
      Go, Lo, Qo: out STD_LOGIC);
end vcompar4;

architecture vcompar4_arch of vcompar4 is
component vcompar1
    port(x,z,Gi,Li,Qi: in STD_LOGIC; Go,Lo,Qo: out STD_LOGIC);
end component;
```

```vhdl
```
end component;

signal G,L,Q: STD_LOGIC_VECTOR(4 downto 0);

begin
  G(4) <= Gi; L(4) <= Li; Q(4) <= Qi;
  c3: vcompar1 port map (x(3),z(3),G(4),L(4),Q(4),G(3),L(3),Q(3));
  c2: vcompar1 port map (x(2),z(2),G(3),L(3),Q(3),G(2),L(2),Q(2));
  c1: vcompar1 port map (x(1),z(1),G(2),L(2),Q(2),G(1),L(1),Q(1));
  c0: vcompar1 port map (x(0),z(0),G(1),L(1),Q(1),G(0),L(0),Q(0));
  Go <= G(0); Lo <= L(0); Qo <= Q(0);
end vcompar4_arch;
2. (15 points) Enter a schematic of a $D$-latch and simulate it using the unit delay mode with a simulation precision of 1 ns. Demonstrate how the $D$-latch can be made to oscillate by changing the $D$ input just before the control input drops. Turn in your schematic and simulation output.
3. (12 points) Consider the sequential circuit shown below. Assume that the flip flops have a setup time of 2 ns, a hold time of 1 ns and a propagation delay between 1 and 3 ns. Also assume that the maximum clock skew is 1 ns and that all the gates have a propagation delay between .5 and 2 ns. What is the shortest clock period for which we can be certain that there are no violations of setup times, assuming no changes at the input X? Is the circuit subject to hold time violations? If so, what would you do to eliminate the hold time violations? If the clock goes high at time 0, during what time period must X be stable to ensure that there are no violations of setup and hold times? During what time period is it possible for the output to be changing. If the clock could change anytime between $t=-1$ ns and $t=+1$ ns, how do the last two answers change?

To avoid setup time violations, we must have

$$(\text{flip flop propagation delay}) + (\text{comb. circuit delay}) + (\text{clock skew}) + \text{setup time} < \text{clock period}$$

Using the maximum values for the propagation delays (and the longest delay path through the combinational circuits), we find that the period must be at least $3 + (3*2) + 1 + 2 = 12$ ns.

To avoid hold time violations, we must have

$$(\text{flip flop propagation delay}) + (\text{comb. circuit delay}) > (\text{clock skew}) + (\text{hold time})$$

Using the minimum values for the propagation delays, we find that the smallest possible value for the left side is 1.5 ns and the largest possible value for the right side is 2 ns. So, hold time violations can occur. We can eliminate hold-time violations by inserting a pair of inverters in the circuit just before the input of the lower flip flow. With this change, the smallest possible value for the left side of the above inequality is 2.5 ns. Now, when you add these inverters, you change the timing for the setup time analysis. Specifically, the addition of the inverters adds 2 ns to the minimum clock period, making it 16 ns.

If the clock changes at $t=0$, then X must be stable starting at $t=-6$ and continuing until $t=0$. If the clock can change anytime within 1 ns of $t=0$, then X must be stable starting at $t=-7$ and continuing until time $t=+1$. The output Z can change anytime between $t=+1.5$ ns and $t=+5$ ns, if the clock changes at $t=0$, but if the clock can change anytime in the interval between $-1$ and $+1$, ...
then the output can change anytime between +.5 and +6.

Repeat the above analysis for the circuit shown below. That is, (a) determine if there are any hold time violations, and if so, explain how they can be fixed, (b) determine the fastest clock frequency for which there are no setup time violations, (c) determine when, relative to a rising clock transition each of the inputs must be stable to ensure correct operation.

Note that for this circuit, the timing of an output change is dependent not just on the clock, but on the data input as well. How does this affect the construction of larger circuits that might use this component?

To avoid hold time violations, we must have

(flip flop propagation delay) + (comb. circuit delay) > (clock skew) + (hold time)

Using the minimum values for the propagation delays, we find that the smallest possible value for the left side is 1.5 ns and the largest possible value for the right side is 2 ns. So, hold time violations can occur. We can eliminate hold-time violations by inserting a pair of inverters in the circuit on the feedback paths from the outputs of the all three flip flop. With this change, the smallest possible value for the left side of the above inequality is 2.5 ns. However, there is also another way of eliminating hold time violations that has less impact on the clock period needed to avoid setup time violations. Specifically, if we put an inverter pair just before the top input of the exclusive-OR gate, another inverter pair on the feedback path from B third inverter pair just before the D input of the third flip flop, we can eliminate hold time violations without increasing the maximum path length from a flip flop output to a flip flop input.

To avoid setup time violations, we must have

(flip flop propagation delay) + (comb. circuit delay) + (clock skew) + setup time < clock period

Using the maximum values for the propagation delays (and the longest delay path through the combinational circuits), we find that the period must be at least 3+(4*2)+1+2=14 ns if the circuit is left as it is. This means we can have a clock frequency of up to 83 MHz. If we use the second method for eliminating hold time violations that was described above, we can also operate the circuit safely at 83 MHz. However, if we add the inverter pairs on the three feedback paths, then
the combinational circuit delay increases by 4 ns, giving a clock period of 18 ns and a maximum clock frequency of 56 MHz.

First, consider the circuit as shown above. If the clock can change anytime within 1 ns of \( t=0 \), then \( X \) must be stable starting a \( t= -9 \) and continuing until \( t=+1 \). This input timing also holds if we fix the hold time violations by inserting inverter pairs on the feedback paths. However, if we fix the hold time violations in the second way described above, then the input timing changes. In this case, the input \( X \) must be stable from \( t= -9 \) until \( t=0 \). Because the timing of \( Z \) is dependent on changes in the timing of \( X \), we cannot say by looking at this circuit, when \( Z \) will be stable, in relation to the clock signal. This makes it more difficult for the designer of a circuit that uses \( Z \) to do so safely, since the designer must consider the timing of changes in \( X \).

4. (12 points) For each of the sequential circuits in problem 3, construct a state table and a state transition diagram. Note that one of these is a Mealy model circuit and one is a Moore model circuit.

\[
\begin{array}{c|cccc}
ABX & D_A & D_B & Z \\
\hline
000 & 0 & 0 & 0 & 0 \\
001 & 1 & 0 & 0 & 0 \\
010 & 0 & 0 & 0 & 0 \\
011 & 1 & 0 & 0 & 0 \\
100 & 0 & 1 & 1 & 0 \\
101 & 0 & 1 & 0 & 0 \\
110 & 0 & 1 & 0 & 0 \\
111 & 0 & 1 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c|cccc}
ABCX & D_A & D_B & D_C & Z \\
\hline
0000 & 0 & 0 & 0 & 0 \\
0001 & 0 & 0 & 0 & 0 \\
0010 & 0 & 0 & 0 & 0 \\
0011 & 0 & 0 & 0 & 0 \\
0100 & 0 & 0 & 0 & 0 \\
0101 & 0 & 0 & 0 & 0 \\
0110 & 0 & 0 & 0 & 0 \\
0111 & 0 & 0 & 0 & 0 \\
1000 & 0 & 0 & 0 & 0 \\
1001 & 0 & 0 & 0 & 0 \\
1010 & 0 & 0 & 0 & 0 \\
1011 & 0 & 0 & 0 & 0 \\
1100 & 0 & 0 & 0 & 0 \\
1101 & 0 & 0 & 0 & 0 \\
1110 & 0 & 0 & 0 & 0 \\
1111 & 0 & 0 & 0 & 0 \\
\end{array}
\]
5. (12 points) Design sequential circuits that implement the state transition diagrams shown below. Note that one of these is for a Mealy model circuit and one is for a Moore model circuit.
\[ D_A = A(F' + G') + B'C'F'G + BCFG' \]

\[ D_B = B'CFG' + BC'(F+G') + BC(F'+G) \]

\[ D_C = A'C'FG' + BC'(F\oplus G) + C(F\oplus G)' \]
6. (20 points) Design a serial multiply-by-5 circuit. Your circuit will have a single data input \( D \), a synchronous reset input \( R \) and a single output \( Q \). Whenever \( R \) is high, your circuit should output 0, but when \( R \) drops low, your circuit should treat input \( D \) as a binary value, received with the least significant bit first. Your circuit should output a value which is equal to the input value, multiplied by 5. So if you receive the values 0001101 (where the rightmost bit is received on the first clock tick following reset, then the second bit from the right, etc.) your circuit will output 1000001.

Design your circuit as a Moore model circuit. Carry out each of the following steps

(a) create a state transition diagram for the circuit,
(b) determine the number of flip flops needed and select a state assignment,
(c) write logic equations for each flip flop input,
(d) write logic equations for the output \( Q \),
(e) draw a schematic for your circuit and
(f) do a timing analysis of your circuit; specifically, determine if there are any hold time violations and if so, explain how to fix them, determine the smallest clock period that it can handle without risk of setup time violations, determine when the input must be stable relative to the rising clock edge and determine the time period following a clock transition when the output can be changing (use the timing parameters from problem 3).

To do a multiply by five serially, we need to add the “current” input bit, with the input bit from two clock ticks in the past. This gives us the next bit of the product. To do this, we need to remember the last two bits and the carry from the addition used to produce the output. We can implement this as a Mealy model circuit, based on the state diagram shown below. In this design, the state is ABC, where A is the previous input bit, B is the bit that was received before that and C is the carry from the last addition.
We can convert any Mealy model circuit to a Moore model circuit by adding flip flops at the inputs or outputs. In the circuit shown below, an extra flip flop has been added at the input. The circuit diagram appears below, along with the next state and output equations. The reset is added as a gating signal that clears all the flip flops when it is asserted, putting the circuit into the initial state 0000.

\[ Y = X \quad B = Y \quad B = A \quad D = YA + AB + YB \quad Z = Y \oplus B \oplus C \]

To avoid hold time violations, we must have

\[(\text{flip flop propagation delay}) + (\text{comb. circuit delay}) > (\text{clock skew}) + (\text{hold time})\]

Using the minimum values for the propagation circuit delays, we find that the smallest possible value for the left side is 1.5 ns and the largest possible value for the right side is 2 ns. So, hold time violations can occur. We can eliminate hold-time violations by inserting inverter pairs in front of the D inputs of flip flops B and C. With this change, the smallest possible value for the left side of the above inequality is 2.5 ns.

To avoid setup time violations, we must have

\[(\text{flip flop propagation delay}) + (\text{comb. circuit delay}) + (\text{clock skew}) + \text{setup time} < \text{clock period}\]

Using the maximum values for the propagation delays (and the longest delay path through the combinational circuits), we find that the period must be at least \(3 + (3^2) + 1 + 2 = 12\) ns. This is true, even after adding the inverter pairs needed to avoid hold time violations. This means we can have a clock frequency of up to 83 MHz.

If the clock can change anytime within 1 ns of \(t=0\), then \(X\) and \(\text{reset}\) must be stable starting at \(t=-4\) and continuing until time \(t=+1.5\). \(Z\) can change anytime between time \(+0.5\) and \(+6\)ns.
7. (25 points) Write two different VHDL specifications for the state machine shown below. In the first version, define a state type (as on page 4-27 of the course notes) using the symbolic state names shown on the diagram. Use the symbolic state names in the code, as on page 4-27. Write the second version in the style shown on page 4-26. That is, define explicit signals for the state variables and code the next-state logic explicitly. Use the following state assignment: peach=000, pit=001, plum =010, prune=011, nut=111.

Simulate both of your designs. For your simulations, design a test sequence that exercises all the transitions in the state diagram. To do this, you must first force the circuit into a known state. What initial sequence of input values will guarantee that the circuit starts in a known state?

Turn in listings of your VHDL source code and the simulation output from both runs. On the simulation output, label every clock tick to show the state transition that it exercises. Do this by writing a label consisting of the current state name and the input values that cause the transition to the next state.
library IEEE;
use IEEE.std_logic_1164.all;

entity fruity is
port (
    a,b,reset,clk: in STD_LOGIC;
    x,y: out STD_LOGIC
);
end fruity;

architecture tutti of fruity is
begin
    process begin
        wait until clk = '1';
        if reset = '1' then
            state <= peach;
        elsif state = peach then
            if a = '1' and b = '0' then
                state <= pit;
            elsif a = '1' and b = '1' then
                state <= nut;
            end if;
        elsif state = pit then
            if a = '0' and b = '1' then
                state <= peach;
            elsif a = '1' and b = '1' then
                state <= plum;
            end if;
        elsif state = plum then
            if a = '1' and b = '0' then
                state <= prune;
            elsif b = '1' then
                state <= peach;
            end if;
        elsif state = prune then
            if a = '0' and b = '0' then
                state <= pit;
            elsif a = '1' and b = '1' then
                state <= nut;
            end if;
        else -- state = nut
            if a = '0' and b = '1' then
                state <= prune;
            elsif a = '1' and b = '0' then
                state <= peach;
            end if;
        end if;
    end process;
end architecture;
x <= '1' when (state = peach) or (state = prune) else '0';
y <= '1' when (state = peach) or (state = prune)
or (state = nut) else '0';
end tutti;
library IEEE;
use IEEE.std_logic_1164.all;

entity fruity is
port(
a,b,reset,clk: in STD_LOGIC;
x,y: out STD_LOGIC
);
end fruity;

architecture tutti of fruity is
begin
    process begin
        wait until clk = '1';
        if reset = '1' then
            s2 <= '0'; s1 <= '0'; s0 <= '0';
        elsif s2 = '0' and s1 = '0' and s0 = '0' then
            if a = '1' and b = '0' then
                s2 <= '0'; s1 <= '0'; s0 <= '1';
            elsif a = '1' and b = '1' then
                s2 <= '1'; s1 <= '1'; s0 <= '1';
            end if;
        elsif s2 = '0' and s1 = '0' and s0 = '1' then
            if a = '0' and b = '1' then
                s2 <= '0'; s1 <= '0'; s0 <= '0';
            elsif a = '1' and b = '1' then
                s2 <= '0'; s1 <= '1'; s0 <= '0';
            end if;
        elsif s2 = '0' and s1 = '1' and s0 = '0' then
            if a = '1' and b = '0' then
                s2 <= '0'; s1 <= '1'; s0 <= '1';
            elsif b = '1' then
                s2 <= '0'; s1 <= '0'; s0 <= '0';
            end if;
        elsif s2 = '0' and s1 = '1' and s0 = '1' then
            if a = '0' and b = '0' then
                s2 <= '0'; s1 <= '0'; s0 <= '1';
            elsif a = '1' and b = '1' then
                s2 <= '1'; s1 <= '1'; s0 <= '1';
            end if;
        else -- state = nut
            if a = '0' and b = '1' then
                s2 <= '0'; s1 <= '1'; s0 <= '1';
            elsif a = '1' and b = '0' then
                s2 <= '0'; s1 <= '0'; s0 <= '0';
            end if;
        end if;
    end if;
end process;
end tutti;
end process;
x <= '1' when (s2 = '0' and s1 = '0' and s0 = '0') or (s2 = '0' and s1 = '1' and s0 = '1') else '0';
y <= '1' when (s2 = '0' and s1 = '0' and s0 = '0') or (s2 = '0' and s1 = '1' and s0 = '1') or (s2 = '1' and s1 = '1' and s0 = '1') else '0';
end tutti;