1. The logic diagram at left below shows a 5 bit ripple-carry decrement circuit. Draw a logic diagram for a 5 bit borrow-lookahead decrement circuit. You may use gates with more than 2 inputs. What is the worst-case propagation delay for your circuit, if all gates with 1 input have a delay of 1 ns, all gates with 2 inputs have a delay of 2 ns, all gates with 3 or 4 inputs have a delay of 4 ns and all gates with 5 to 8 inputs have a delay of 6 ns?

The max delay is from the DECR input through the 5 input OR gate to Q4. This delay is 1+6+2=9 ns.
2. An $n$ bit comparator takes two $n$ bit input values $x$ and $y$ and has two outputs $L$ and $G$. If the value of $x$ is numerically smaller than $y$, then $L$ is high and $G$ is low. If the value of $x$ is greater than $y$, then $G$ is high and $L$ is low. Otherwise, both $L$ and $G$ are low. An $n$ bit comparator can be implemented using $n$ 1 bit comparators. Show how to implement a 4 bit comparator in this way. (Note that the outcome of the comparison is determined entirely by the most significant bit position in which the value of $x$ differs from $y$.)

The two outputs $L$ and $G$ of a 1 bit comparator are defined by the logic equations $L = x \cdot y'$ and $G = xy'$. For a 2 bit comparator, the corresponding equations are $L = x_1y_1' + (x_1y_1)(x_0y_0)' + (x_1' y_1)(x_0y_0')$ and $G = x_1y_1' + (x_1' y_1)(x_0y_0')$. This leads to a circuit that looks like this.

The portion of the circuit highlighted by the dashed lines can be cascaded to produce an $n$ bit comparator. So, for example, using 4 copies of this circuit, we can get the 4 bit comparator shown below.
If every gate has a worst-case delay of 1 ns, what is the longest possible delay in your 4 bit circuit from the time the input values change to the time the output values finish changing? What is the worst-case delay for an $n$ bit version of your circuit? Show how to build a faster lookahead version of your circuit, using only simple gates.

The longest path from an input to an output is from input $x_{n-1}$ to output $L$, which must go through $n+2$ gates. So, with a 1 ns delay per gate, the longest possible delay is 6 ns and for an $n$ bit comparator, the worst-case delay is $n+2$ ns. We can build a lookahead version of the circuit we first need to derive the appropriate logic equations. Let $L_i = x_i \overline{y_i}$ and $G_i = x_i y_i$. Then, for the 4 bit comparator, we can write $L = L_3 + G_3 L_2 + G_3 G_2 L_1 + G_3 G_2 G_1 L_0$ and $G = G_3 + L_3 G_2 + L_3 L_2 G_1 + L_3 L_2 L_1 G_0$. A circuit implementing these equations appears below. This can be converted to a circuit using simple gates by replacing each of the $k$ input gates (for $k>2$) with a balanced tree of $k-1$ simple gates. This circuit has a worst-case delay of 7 ns, which is actually worse than the earlier circuit, but for larger $n$ it will be faster. The $n$ bit version of the circuit has a worst-case delay of $3 + 2 \log_2 n$ for any $n$ that is a power of 2.
There is an alternative design for a lookahead comparator that requires fewer gates than the $n$ bit version of the design shown above, which requires about $n^2$ simple gates. This one is based on a different way of writing the logic equations. Define $L(i,j)$ to be true if the number formed from bits $x_{i+j-1}...x_i$ is less than the number formed from bits $y_{i+j-1}...y_i$. Define $G(i,j)$ similarly. Then for an $n$ bit comparator, the final outputs $L = L(0,n)$ and $G = G(0,n)$. Now if $n$ is a power of 2, we can write

$L(0,n) = L(n/2,n/2) + G(n/2,n/2)$ and $G(0,n) = G(n/2,n/2) + L(n/2,n/2)$ $G(0,n/2)$. Similarly, $L(0,n/2) = L(n/4,n/4) + G(n/4,n/4)$ $L(0,n/4)$ and $L(0,n/2) = L(n/4,n/4) + G(n/4,n/4)$ $L(0,n/4)$ and in general $L(i,j) = L(i+j/2,j/2) + G(i+j/2,j/2)$ $L(i,j/2)$ and $G(i,j) = G(i+j/2,j/2) + L(i+j/2,j/2)$ $G(i,j/2)$. A 4 bit version of the circuit obtained using these equations is shown below. The $n$ bit version has a worst-case delay of $2 + 3 \log_2 n$ which is not quite as good as the first lookahead circuit, but this circuit has a gate count of $8n - 5$, which is a huge improvement.
3. Write a VHDL module (entity and architecture) that implements a 2→4 decoder with an enable input. Your decoder should have three inputs (including the enable) and four outputs. The outputs should all be low when the enable is low. This can be written using a single conditional signal assignment statement.

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

entity dec24 is
    port (
        A: in std_logic_vector(1 downto 0);
        en: in std_logic;
        X: out std_logic_vector(3 downto 0)
    );
end dec24;
architecture dec24arch of dec24 is
begin
    X <= "0001" when en = '1' and A = "00" else
         "0010" when en = '1' and A = "01" else
         "0100" when en = '1' and A = "10" else
         "1000" when en = '1' and A = "11" else
         "0000";
end dec24arch;
```

Now, write a VHDL module that implements a 3→8 decoder using two copies of the 2→4 decoder, that are “wired” together using component statements with port maps.

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

entity dec38 is
    port (
        A: in std_logic_vector(2 downto 0);
        X: out std_logic_vector(7 downto 0)
    );
end dec38;
architecture dec38arch of dec38 is
signal Z: std_logic;
component dec24
    port (
        A: in std_logic_vector(1 downto 0);
        en: in std_logic;
        X: out std_logic_vector(3 downto 0)
    );
end component;
begin
    Z <= not A(2);
    lo: dec24 port map(A(1 downto 0),Z,X(3 downto 0));
    hi: dec24 port map(A(1 downto 0),A(2),X(7 downto 4));
end dec38arch;
```
4. Draw logic diagrams for combinational circuits that are equivalent to each of the VHDL modules shown below. Use simple gates only in your circuits.

(a) library IEEE;
    use IEEE.std_logic_1164.all;

    entity partA is
        port (A, B, C, D: in std_logic;
              X, Y: out std_logic);
    end partA;

    architecture partAarch of partA is
    signal Z: std_logic;
    begin
        Z <= (A and B) or C;
        X <= (not D) xor Z;
        Y <= (A or (not B)) and Z;
    end partAarch;
(b) library IEEE;
    use IEEE.std_logic_1164.all;

    entity partB is
        port (
            A, B, C, D: in std_logic;
            X, Y: out std_logic
        );
    end partB;
    architecture partBarch of partB is
    begin
        X <= (not D) xor (A or C);
        Y <= '0' when (A = '1' and B = '0')
            or (C = '1' and D = '1')
            or (A = '0' and B = '1')
            else '1';
    end partBarch;
5. Write two different VHDL modules that implement the logic unit in problem 1 of design problem 2. In the first version, write logic equations for each output signal that correspond directly to the schematic in the solution set provided on the web site. Pattern the second version, after the arithmetic unit design on page 3-36 of the lecture notes.

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

dentity version1 is
  port (
    A, B: in std_logic_vector(3 downto 0);
    S: in std_logic_vector(1 downto 0);
    R: out std_logic_vector(3 downto 0)
  );
end version1;
architecture v1arch of version1 is
begin
  R(0) <=(S(1) and not S(0) and not A(0)) or
         (not S(1) and A(0) and B(0)) or
         (S(0) and A(0) xor B(0));
  R(1) <=(S(1) and not S(0) and not A(1)) or
         (not S(1) and A(1) and B(1)) or
         (S(0) and A(1) xor B(1));
  R(2) <=(S(1) and not S(0) and not A(2)) or
         (not S(1) and A(2) and B(2)) or
         (S(0) and A(2) xor B(2));
  R(3) <=(S(1) and not S(0) and not A(3)) or
         (not S(1) and A(3) and B(3)) or
         (S(0) and A(3) xor B(3));
end v1arch;

library IEEE;
use IEEE.std_logic_1164.all;

dentity version2 is
  port (
    A, B: in std_logic_vector(3 downto 0);
    S: in std_logic_vector(1 downto 0);
    R: out std_logic_vector(3 downto 0)
  );
end version2;
architecture v2arch of version2 is
begin
  R <= A and B when S = "00" else
       A or B when S = "01" else
       not A when S = "10" else
       A xor B when S = "11";
end v2arch;
```
6. Draw a logic diagram using simple gates and a 4:1 multiplexor that is equivalent to the VHDL code shown below. Be sure to label all the inputs of your multiplexor correctly.

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

entity foo is
    port (curve, slider, wild: in std_logic;
          sign: in std_logic_vector (1 downto 0);
          strike, run: out std_logic
    );
end foo;
architecture bar of foo is
begin
    strike <= (curve or slider) and not wild;
    run <= (curve xor slider) when sign = "00" else
           wild when sign = "01" else
           slider and wild;
end bar;
```

[Logic diagram with labeled inputs and outputs]
7. Page 4-7 and 4-8 of the lecture notes discuss a serial parity generator. During what time period must the D input be stable if the gate delays can vary from .8 to 2.5 ns. During what period must the EN input be stable in this case? Draw a diagram like the one on page 4-8, showing the time periods during which the inputs must be stable.

*Input D must be stable from 7 ns before the clock changes to .6 ns before the clock changes. Input EN must be stable from 4.5 ns before the clock changes to .2 ns after the clock changes. The timing diagram appears below.*