1. Consider the circuit shown below. Assume that all gates have a minimum propagation delay (for both rising and falling edges) of 1 ns, and a maximum propagation delay of 2 ns. Construct two timing diagrams showing how all the labeled signals change when input $A$ changes from 1 to 0, while inputs $B$ and $C$ are held at 1 and 0, respectively. In the first timing diagram, assume the minimum propagation time for all gates and in the other, assume the maximum propagation time for all gates. Draw a third timing diagram which combines the first two, using shading to show the time periods where signal values could be either 0 or 1.

2. Analyze the two circuits shown below. Specifically, create a truth table showing how the outputs depend on the inputs, and give simplified Boolean expressions for each of the outputs.

3. Show how to implement the function $F = AB + AB'C$ using a 3→8 decoder and two additional gates (use only 2 input gates).
4. Design circuits that implement the logic in the truth table shown below. Design two circuits, one which has the smallest possible worst-case propagation delay, and a second which minimizes the gate count. Use only inverters and 2-input AND gates and 2-input OR gates. Assume each inverter has a propagation delay of 1 ns and each AND and OR gate has a 2 ns propagation delay. When counting gates, treat an inverter as a “half-gate.” What are the gate counts and worst-case propagation delays for your circuits?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

5. Show how to implement a circuit for each of the following expressions using a multiplexor (as on page 3-15 of the notes).
   (a) \((A + B)C' + B'D\)
   (b) \(A'C + (B'D + C)(A' + D)\)

6. The equations below are the basis for the ripple-carry adder circuit (see page 3-18 of the notes).
   \[ S = X \oplus Y \oplus C_{in} \quad C_{out} = XY + XC_{in} + YC_{in} = XY + (X + Y)C_{in} \]
   There is a similar set of equations that can be used to create a ripple-carry subtractor circuit.
   \[ D = X \oplus Y \oplus B_{in} \quad B_{out} = X'Y + X'B_{in} + YB_{in} = X'Y + (X' + Y)B_{in} \]
   Draw a logic diagram for a 4 bit subtractor, based on these equations.

7. An \(n\) bit barrel shifter takes an \(n\) bit data value, \(x\), and a \(\log n\) bit control value \(i\). The \(n\) bit output, \(y\), contains the same bits as \(x\), but rotated to the right by \(i\) bit positions. If \(x = x_{n-1} \ldots x_1 x_0\) then \(y = x_{i-1} \ldots x_0 x_{n-1} \ldots x_{i+1} x_i\). Design a circuit that implements a barrel shifter using \(n\) multiplexors with \(n\) inputs each and draw a 4 bit version of this circuit. How many simple gates does this design require? How many simple gates does the \(n\) bit version require? Design an alternative barrel shifter circuit that uses multiple 2-input multiplexors. Draw the 4 bit version of this circuit. How many simple gates does this design require? How many simple gates does the \(n\) bit version of this circuit require?
8. 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?

![Logic Diagram](image)

9. 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 $y$.)

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.

10. 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.

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.

Run a behavioral simulation of the a 3→8 decoder, showing all eight input combinations. The output should match that shown on page 3-12 of the lecture notes.

11. 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;
12. Pages 2.2, 2.3 and 3.2 of the lecture notes describe a 4 input arithmetic unit, that performs several different arithmetic operations. The schematic below shows a similar 4 input logic unit, that performs several different logic operations. In particular, when the control input, \( S = 0 \), the output is the bitwise AND of the \( A \) and \( B \) inputs (that is, each bit of \( A \) is AND-ed with the corresponding bit of \( B \)). When \( S = 1 \), the output is the bitwise OR of \( A \) and \( B \), when \( S = 3 \), the output is the bitwise EXOR of \( A \) and \( B \), and when \( S = 2 \), the output is the bitwise complement of \( A \).

Write two different VHDL modules that implement the logic. In the first version, write logic equations for each output signal that correspond directly to the schematic. Pattern the second version, after the arithmetic unit design on page 3-36 of the lecture notes.

13. 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;