1. (10 points) Draw a logic diagram that implements the expression \( A(B+C)(C'+D)(B+D') \) directly (do not simplify first), using only simple gates.

![Logic Diagram](image)

Simplify the expression as much as you can.

\[
A(B+C)(C'+D)(B+D') = A(BC' + BD + CD)(B+D') = A(BC'+BD) = AB(C'+D)
\]

How many simple gates of each type are required to implement the simplified expression? The circuit corresponding to this expression requires 2 gates, 1 or gate and 1 inverter.
2. (15 points) Draw a circuit that implements the VHDL code fragment shown below. Assume that \( x \) and \( z \) are two bit signals. All others are of type `std_logic`. You may use simple gates, multiplexors and flip flops in your circuit diagram.

```vhdl
with z select
  x <= a & b when "00" | "01",
      "01" when "10",
      "11" when others;
process (clk) begin
  if rising_edge(clk) then
    if a > b then
      y <= a and c;
    elsif a /= c then
      y <= x(0);
    end if;
  end if;
end process;
```

![Circuit Diagram]

-2-
3. (10 points) Consider the circuit shown below, which includes four copies of the same basic building block.

![Circuit Diagram]

Write a VHDL process containing a loop that specifies this circuit.

```vhdl
process(a, b, z) begin
    for i in 0 to 3 loop
        if b(i) /= z(i) then
            x(i) <= a(i) or z(i);
        else
            x(i) <= b(i);
        end if;
        z(i+1) <= a(i) and z(i);
    end loop;
end process;
```
4. (10 points) Draw a diagram of an 8-to-1 multiplexor with data inputs $D_0$ to $D_7$ and a 3 bit control input $C$, using smaller multiplexors as building blocks. Make sure that all signals and mux inputs are labeled appropriately. Pay special attention to the control inputs of the mux components in your circuit.

![Diagram of 8-to-1 multiplexor](image)

How many LUT4s does it take to implement this circuit?

*It takes 3 LUTs to implement each 4:1 mux, plus one more to implement the 2:1 mux, so 7 in all.*
5. (15 points) The VHDL module shown below counts the number of odd length pulses that have been observed on the dIn input since the last reset. What is the smallest number of flip flops needed to implement this VHDL spec?

We need at least three for the state register and eight for the oddCount register. So 11.

```
entity oddPulseCounter is port(
  clk, reset, dIn: in std_logic;
  oddCount: out std_logic_vector(7 downto 0));
end oddPulseCounter;
architecture a1 of oddPulseCounter is
begin
  type stateType is (resetState, start1, prev0, prevlodd, prevleven);
  signal state: stateType;
  process(clk) begin
    if rising_edge(clk) then
      if reset = '1' then state <= resetState;
      else case state is
        when resetState =>
          oddCount <= (others => '0');
        when start1 =>
          if dIn = '0' then state <= prev0; end if;
        when prev0 =>
          if dIn = '1' then state <= prevlodd; end if;
        when prevlodd =>
          if dIn = '0' then
            state <= prev0; oddCount <= oddCount + 1;
          else state <= prevleven; end if;
        end case;
      end if;
    end if;
  end process;
end a1;
```

Complete the state diagram for this VHDL module. Show updates to stored values.
6. (10 points) Consider the state diagram shown at left below. Fill in the entries in the state table at right. You may abbreviate the state names as R, G and B.

Consider the state table shown below. Draw a state diagram corresponding to this state table. Is this state machine a Mealy-mode machine or a Moore-mode machine?

*It’s a Mealy-mode state machine.*
7. (15 points) The VHDL module shown below defines a sequential circuit that looks for the minimum value present on the input A and counts the number of clock periods when this minimum value is present. It has two outputs, `minVal` and `minCount`. So for example, if the input sequence on A is 57, 85, 23, 34, 36, 23, 46, 23 then the sequences of values on the two outputs will be 57, 57, 23, 23, 23, 23, 23, 23 and 1,1,1,1,2,2,3.

```vhdl
entity minValCount is port (
    clk, reset: in std_logic;
    A : in std_logic_vector(7 downto 0);
    minVal, minCount : out std_logic_vector(7 downto 0));
end minValCount;
architecture a1 of minValCount is
    signal val, count: std_logic_vector(7 downto 0);
begin
    process (clk) begin
        if rising_edge(clk) then
            if reset = '1' then
                val <= x"FF"; count <= x"00";
            else
                if A < val then val <= A; count <= x"01";
                elsif A = val then count <= count + 1;
                end if;
            end if;
        end if;
    end process;
    minVal <= val; minCount <= count;
end a1;
```

Complete the circuit shown below, so that it implements the VHDL module above. Use only simple gates and 2:1 multiplexors.