1. VHDL Analysis (timing diagrams): Given the following VHDL specification, complete the following timing diagram for outputs Z(0), Z(1), Z(2), Z(3).

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Prob1 IS
    PORT (D, CLOCK, CLR1, CLR2 : IN STD_LOGIC;
          Z         : OUT STD_LOGIC_VECTOR(0 TO 3));
END Prob1;

ARCHITECTURE P1Arch OF Prob1 IS
BEGIN
    PROCESS (CLOCK, CLR1)
    BEGIN
        IF CLR1 = '0' THEN Z(0) <= '0';
        ELSIF CLOCK EVENT AND CLOCK = '1' THEN
            IF CLR2 = '0' THEN Z(0) <= '0';
            ELSE Z(0) <= D;
            END IF;
        END IF;
    END PROCESS;

    PROCESS (D, CLOCK)
    BEGIN
        IF CLOCK = '1' THEN Z(1) <= D;
        END IF;
    END PROCESS;

    PROCESS (D, CLOCK)
    BEGIN
        IF CLOCK = '1' THEN Z(2) <= D;
        ELSE Z(2) <= '0';
        END IF;
    END PROCESS;

    PROCESS (CLOCK)
    BEGIN
        IF CLOCK EVENT AND CLOCK = '1' THEN Z(3) <= D;
        END IF;
    END PROCESS;

END P1Arch;
```

**Important Note:**
- Every flip-flop and latch starts off with an unknown value.
- Please show propagation delays.
2(a). Using the GENERIC feature of VHDL, complete the following code that will define a "generic" component named nBitAdd shown below. The generic component is a \( n \)-bit, ripple-carry adder containing \( n \) full adders.

\[
\begin{array}{c}
\text{nBitAdd} \\
\hline \\
A & S \\
B & \text{Cout} \\
\hline \\
\text{Cin} & n \\
\end{array}
\]

Equations for a full adder:
\[
S \leftarrow A \oplus B \oplus \text{Cin} \\
\text{Cout} \leftarrow (A \land B) \lor ((A \lor B) \land \text{Cin})
\]

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
ENTITY nBitAdd IS
  GENERIC (M : INTEGER := 4);
  PORT (A, B : IN STD_LOGIC_VECTOR(M-1 DOWNTO 0);
        Cin : IN STD_LOGIC;
        Sum : OUT STD_LOGIC_VECTOR(M-1 DOWNTO 0);
        Cout : OUT STD_LOGIC);
END nBitAdd;
ARCHITECTURE genericAdder OF nBitAdd IS
SIGNAL CT : STD_LOGIC_VECTOR(M DOWNTO 0); -- M+1 bits
BEGIN
  PROCESS (A, B, Cin) -- Hint: Use FOR loop;
  BEGIN
    CT(0) := Cin;
    FOR i IN 0 TO M-1 LOOP
      S(i) := A(i) XOR B(i) XOR CT(i);
      CT(i+1) := (A(i) AND B(i)) OR ((A(i) OR B(i)) AND CT(i));
    END LOOP;
    Cout := CT(M);
  END PROCESS;
END genericAdder;
2(b). Given the following ENTITY definition, give me the PORT MAP statement that will create a 32-bit adder using the above generic adder.

ENTITY TestGenAdd IS
  PORT( Cin : IN STD_LOGIC;
        x, y : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        Cout: OUT STD_LOGIC;
        sumOut: OUT STD_LOGIC_VECTOR(31 DOWNTO 0));
END TestGenAdd;

**Put the PORT MAP statement here:**

```vhdl
adda32 : mBitAdd GENERIC MAP(m => 32)
  PORT MAP (X, Y, Cin, sumOut, Cout);
```

---

**Problem 3**

Shown below is the desired timing of a 2-bit counter (countOut).

![Timing Diagram](image)

3. Complete the ARCHITECTURE section on the next page to implement the **2-bit counter** with an asynchronous Resetn.

**Important notes and hints:**

- You can determine the count sequence from the above timing diagram (countOut).
- The count sequence is also affected by input X (as seen from timing diagram).
- There are three more outputs (Z1, Z2, and Z3). They are determined by the countOut value and input X.
- Hint: Z3 is true for one clock cycle after countOut = "11".
- CASE statement will make your life a lot easier.

**ENTITY T1Prob3 IS**

```vhdl
ENTITY T1Prob3 IS
  PORT ( Clock, Resetn, X: IN STD_LOGIC;
         countOut : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
         Z1, Z2, Z3 : OUT STD_LOGIC );
END T1Prob3 ;
```

```
ARCHITECTURE T1P3Arch OF T1Prob3 IS
  SIGNAL  count : STD_LOGIC_VECTOR (1 DOWNTO 0);
BEGIN
  PROCESS (Resetn, Clock)
  BEGIN
    IF Resetn = '0' THEN
      count <= "00";
    ELSIF (Clock'EVENT AND Clock = '1') THEN
      CASE count IS
        WHEN "00" =>
          count <= "10";
        WHEN "01" =>
          count <= "00";
        WHEN "10" =>
          count <= "11";
        WHEN OTHERS => -- count = "11"
          IF X = '0' THEN
            count <= "10";
          ELSE
            count <= "01";
          END IF;
      END CASE;
      IF count = "11" -- Inside Clock'EVENT THEN
        z3 <= '1'; -- Create a flip-flop
      ELSE
        z3 <= '0'; -- and a clock cycle delay
      END IF;
    END IF;
  END PROCESS;
  PROCESS (Count, X) -- Combinatorial z1 and z2
  BEGIN
    IF ((count = "00") OR (count = "11" AND X = '1')) THEN
      z1 <= '1';
    ELSE
      z1 <= '0';
    END IF;
    IF (count = "01") THEN
      z2 <= '1';
    ELSE
      z2 <= '0';
    END IF;
  END PROCESS;
  countOut <= count;
END T1P3Arch;
4. Complete the following Architecture section to implement the above circuit.
   - Note that Z4_L and Z5_L are active low output signals. All the other signals active high.
   - The signal Clock is connected to the clock inputs of all the flip-flops.
   - All you code must be inside the PROCESS block.

```vhdl
ENTITY P4Circuit IS
  PORT ( Clock, X : IN STD_LOGIC;
         Z0_H, Z1_H, Z2_H, Z3_H, Z4_L, Z5_L : OUT STD_LOGIC);
END P4Circuit;

ARCHITECTURE Behavior OF P4Circuit IS
  SIGNAL temp20, temp21 : STD_LOGIC;
BEGIN
  -- Other than SIGNAL statements, your code must be inside the PROCESS block.
  PROCESS (Clock, X)
  BEGIN
    IF (Clock 'EVENT AND Clock = '1') THEN
      temp20 <= X;
      temp21 <= temp20;
      Z2_H <= temp21;
      Z3_H <= temp21;
      Z4_L <= NOT(temp20);
    END IF;
  END PROCESS;
END Behavior;
```

The code must be inside the CLOCK 'EVENT to create the flip-flops. Code can be anywhere outside of CLOCK-EVENT.
Problem 5

5. Complete the Architecture section (see next page) for the above BCDCounter8, which consists of two 4-bit counters (Digit1[3..0] and Digit0[3..0]). If EN = ‘1’, then BCDCount8 will count in hex as follows:

00, 01, ...08,09,10,11, ..., 17,18,18,18,19,20, ..., 91,92, ..., 99,00,01,...

EN = '0' here

- EN is an active high "enable" input. See example above.
- When Digit1,Digit0 reaches hex 99, TC will become ‘1’ during that clock cycle and the BCDCounter8 resets to 00 at the next active clock transition.

Be sure to follow these instructions:

- **PROCESS block 1**: Implement the Digit0 counter here. You must use a WAIT UNTIL statement.
- **PROCESS block 2**: Implement the Digit1 counter here. Don’t use a WAIT UNTIL statement.
- You must implement TC after PROCESS block 2 and must use a conditional assignment statement.

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY BCDCounter8 IS
PORT( EN, CLK : IN STD_LOGIC;
      Digit1, Digit0 : BUFFER STD_LOGIC_VECTOR(3 downto 0);
      TC : OUT STD_LOGIC );
END BCDCounter8;

-- Complete the Architecture section on the next page.
ARCHITECTURE design OF BCDCOUNTER8 IS
BEGIN

-- Use this PROCESS block to implement Digit0. You must use an WAIT UNTIL statement.
PROCESS
BEGIN
WAIT UNTIL (CLK'EVENT AND CLK='1');
  IF EN='1' THEN
    IF Digit0 = "1001" THEN
      Digit0 <= "0000";
    ELSE Digit0 <= Digit0+1;
  END IF;
END IF;
END PROCESS;

-- Use this PROCESS block implement Digit1. Don't use an WAIT UNTIL statement.
PROCESS (CLK)
BEGIN
  IF (CLK'EVENT AND CLK='1') THEN
    IF EN='1' THEN
      IF (Digit1 = "1001" AND Digit0 = "1001") THEN
        Digit1 <= "0000";
      ELSEIF Digit0 = "1001" THEN
        Digit1 <= Digit1+1;
      END IF;
    END IF;
  END IF;
END PROCESS;

-- Implement TC here; You must use a conditional assignment statement.
TC <= '1' WHEN (Digit1 = "1001" AND Digit0 = "1001")
  ELSE '0';
END design;
6. Miscellaneous:

LSA Question: The signals A, B, C, and CLK represent synchronous outputs from your UF-4712 board. The LSA is connected to your board in the following fashion:

<table>
<thead>
<tr>
<th>LSA channel</th>
<th>Signal name</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>A</td>
</tr>
<tr>
<td>17</td>
<td>B</td>
</tr>
<tr>
<td>18</td>
<td>C</td>
</tr>
<tr>
<td>31</td>
<td>CLK</td>
</tr>
</tbody>
</table>

Let's assume that “actual” signals behave as follows (vs. what is captured by an LSA).

Draw the data that would be captured by the LSA, as it would appear on the screen, for each of the following scenarios:

6(a). The LSA is sampling using “Channel 31” as the clock source. Assume the first sample is taken after 30 ns. Draw your answer for part a:
6(b). The LSA is sampling using an internal 50 MHz clock source. Assume the first sample is taken right before 20 ns. Draw your answer for part c:

<table>
<thead>
<tr>
<th>Name</th>
<th>0ns</th>
<th>20ns</th>
<th>40ns</th>
<th>60ns</th>
<th>80ns</th>
<th>100ns</th>
<th>120ns</th>
<th>140ns</th>
<th>160ns</th>
<th>180ns</th>
<th>200ns</th>
<th>220ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH-16</td>
<td>XXXX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CH-17</td>
<td>XXXX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CH-18</td>
<td>XXXX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CH-31</td>
<td>XXXX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6(c). For an n-bit adder, the inputs are A(n-1)..A(0) and B(n-1)..B(0) and carry-in C(0). The outputs are SUM(n-1)..SUM(0) and carry-out C(n). (4 pts.)

For an n-bit look-ahead carry generator, the equation for carry-out of stage "i" is:

\[ C(i+1) = G(i) \text{ OR } P(i) \text{ AND } C(i) \]

What is the equation for \( C(2) \) as a function of \( P(i)'s, \ G(i)'s, \text{ and } C(0) \) in a sum-of-product form?

\[
C(2) = G1 + P1 \cdot C1 = G1 + P1 \cdot (G0 + P0 \cdot C0)
\]

\[
= G1 + P1 \cdot G0 + P1 \cdot P0 \cdot C0
\]

What is the equation for \( C(n) \) as a function of \( P(i)'s, \ G(i)'s \text{ and } C(0) \)? You can use the ...

\[
C(n) = G_{m-1} + P_{m-1} G_{m-2} + P_{m-1} P_{m-2} G_{m-3} + \ldots
\]

\[
+ P_{m-1} P_{m-2} P_{m-3} \ldots P0 \cdot C0
\]
ENTITY _entity_name IS
  GENERIC( _variable_name : INTEGER := _constant_value );
  PORT( _input_name, _input_name : IN STD_LOGIC;
        _input_vector_name : IN STD_LOGIC_VECTOR( _high downto _low );
        _bidir_name, _bidir_name : INOUT STD_LOGIC;
        _output_name, _output_name : OUT STD_LOGIC );
END _entity_name ;

ARCHITECTURE a OF _entity_name IS
  SIGNAL _signal_name : STD_LOGIC;
  SIGNAL _signal_name : STD_LOGIC;
BEGIN
  -- Process Statement
  -- Concurrent Signal Assignment
  -- Conditional Signal Assignment
  -- Selected Signal Assignment
  -- Component Instantiation Statement
END a;

__instance_name:_ component_name
  GENERIC MAP ( _variable_name := _constant_value )
  PORT MAP ( _component_port => _connect_port, component_port => _connect_port );

WITH __expression SELECT
  _signal <= __expression WHEN _constant_value,
   __expression WHEN _constant_value,
   __expression WHEN _constant_value,
   __expression WHEN _constant_value;

  _signal <= __expression WHEN _boolean_expression ELSE
   __expression WHEN _boolean_expression ELSE
   __expression;

IF __expression THEN
  _statement;
  _statement;
ELSIF __expression THEN
  _statement;
  _statement;
ELSE
  _statement;
  _statement;
END IF;

CASE __expression IS
  WHEN _constant_value =>
    _statement;
    _statement;
  WHEN _constant_value =>
    _statement;
    _statement;
  WHEN OTHERS =>
    _statement;
    _statement;
END CASE;

WAIT UNTIL __expression;

<optional_label>: for <loop_id> in <range> loop
  -- Sequential Statement(s)
end loop;

<generate_label>: FOR <loop_id> IN <range> GENERATE
  -- Concurrent Statement(s)
END GENERATE;
**IMPORTANT:** Please be neat and write (or draw) carefully. If we cannot read it with a reasonable effort, it is assumed wrong.

**COVER SHEET:**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(16 pts)</td>
</tr>
<tr>
<td>2</td>
<td>(15 pts)</td>
</tr>
<tr>
<td>3</td>
<td>(22 pts)</td>
</tr>
<tr>
<td>4</td>
<td>(15 pts)</td>
</tr>
<tr>
<td>5</td>
<td>(20 pts)</td>
</tr>
<tr>
<td>6</td>
<td>(12 pts)</td>
</tr>
</tbody>
</table>

Total

**Re-Grade Information:**

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________