1. VHDL specification.

Complete the VHDL specification (below and on the next page) for the following circuit:

![Diagram of the circuit]

Notes:
- REG is an 8-bit storage register with an asynchronous CLR and synchronous LD inputs (CLR has priority over LD).
- There are eight 2-to-1 MUX's.
- INCR functions as follows:
  - If inc = 0, F <= 0,
  - If inc = 1, F <= X + 1 (i.e., increment X).
- All signals are active high.

(a) Complete the following Entity declaration for the Prob1 Module: (2 pts)

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
ENTITY Prob1 IS
  -- Declare ZOut to be an OUT signal type.
  PORT( CLR, LD, CLOCK, SEL, EN : IN STD_LOGIC;
  InData : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
  ZOut : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END Prob1;
```

(b) On the next page, complete the architecture section to specify the behavior of the Prob1 Module in behavioral VHDL. (20 pts.)
ARCHITECTURE behaviorArch OF Prob1 IS
SIGNAL REGQ, FOUT : STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
  PROCESS
  BEGIN
    BEGIN
      -- You must use a WAIT UNTIL statement to specify REG
      WAIT UNTIL (CLOCK'EVENT AND CLOCK='1')
      IF CUR='1' THEN -- asynchronous clear
        REGQ <= "00000000";
      ELSIF LD='1' THEN -- synchronous load
        REGQ <= FOUT;
      ELSE
        REGQ <= REGQ; -- not necessary
      END IF;
    END PROCESS;
    PROCESS
    BEGIN
      IF EN='0' THEN
        FOUT <= "00000000";
      ELSIF SEL='0' THEN
        FOUT <= REGQ + 1;
      ELSE
        FOUT <= IN_DATA + 1;
      END IF;
    END PROCESS;
    ZOUT <= FOUT;
  END PROCESS;
END behaviorArch;
2. VHDL specification.

Complete the VHDL specification (on the next page) for the following circuit: (Note that a "bubble" indicates that the signal is active low.

- Shifter4 is a 4-bit shift register that can shift left when (DIR = 0) or shift right (when DIR = 1):
  - When shifting left, Q0 <= Lin; when shifting right, Q3 <= Rin.
  - When LDn = true (active low), then D3 is loaded into the Shifter4. LDn is synchronous and has priority over shifting.
- The 2-to-4 decoder (with an active low enable EN) must be specified using a WITH-SELECT assignment statement. (All decoder outputs are active low).
- The logic for the MUX must be specified using a conditional assignment statement.
Put solution for Problem 2 here:

ENTITY Test1P2 IS
  PORT (Lin, Rin, DIR, LDn, Clock, SEL : IN STD_LOGIC;
         D : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
         Y : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END Test1P2;

ARCHITECTURE P2Arch OF T1Prob2 IS
  SIGNAL Q : STD_LOGIC_VECTOR (3 DOWNTO 0);
  SIGNAL Z : STD_LOGIC;
BEGIN
  PROCESS (Clock)
  BEGIN
    IF (Clock'EVENT AND Clock = '1') THEN
      IF (LDn = '0') THEN -- regen. load
        Q <= D;
      ELSIF DIR = '0' THEN -- left shift
        Q(3) <= Q(2);
        Q(2) <= Q(1);
        Q(1) <= Q(0);
        Q(0) <= LDn;
      ELSE -- DIR = '1', right shift
        Q(3) <= Rin;
        Q(2) <= Q(3);
        Q(1) <= Q(2);
        Q(0) <= Q(1);
      END IF;
    END IF;
  END PROCESS;

  WITH EN & C1 & C0 SELECT -- decoder
  BEGIN
    Y <= "1110" WHEN "0000";
    "1101" WHEN "0001";
    "1011" WHEN "0010";
    "0111" WHEN "0011";
    "1111" WHEN OTHERS;
  END WITH;

  Z <= Q(1) WHEN SEL = '0' ELSE -- MUX
       Q(0);
END P2Arch;
3. **VHDL Analysis – Timing diagrams.** Given the following VHDL specification, complete the following timing diagram for the outputs, Q1, Q2, and Q3.

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY Test1P3 IS
    PORT ( D, Resetn, Clock : IN STD_LOGIC;
           Q1, Q2, Q3 : OUT STD_LOGIC);
END Test1P3;

ARCHITECTURE Behavior OF Test1P3 IS
BEGIN
    PROCESS ( D, Resetn, Clock )
    BEGIN
        Q3 <= '0';
        IF Resetn = '0' THEN
            Q1 <= '0';
        ELSIF Clock = '1' THEN
            Q1 <= D;
        END IF;
        IF Resetn = '0' THEN
            Q2 <= '0';
        ELSIF (Clock'event AND Clock = '1') THEN
            Q2 <= D;
        END IF;
        IF (Resetn = '1' AND Clock = '1') THEN
            Q3 <= D;
        END IF;
    END PROCESS;
END Behavior;
```

**Note:** Please show delays. The initial contents of all flipflops are unknown. Also, go as far as you can.

<table>
<thead>
<tr>
<th>Name</th>
<th>0 ps</th>
<th>80.0 ns</th>
<th>160.0 ns</th>
<th>240.0 ns</th>
<th>320.0 ns</th>
<th>400.0 ns</th>
<th>480.0 ns</th>
<th>56</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The following VHDL code is used for Problem 4 and Problem 5:

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

ENTITY Test1P4_5 IS
  PORT ( Clock, Resetn : IN STD_LOGIC;
         Z1, Z2, Z3 : OUT STD_LOGIC;
         Q, QQA, QQB, QQC : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END Test1P4_5;

ARCHITECTURE Behavior OF Test1P4_5 IS
  SIGNAL Count : STD_LOGIC_VECTOR (3 DOWNTO 0);

BEGIN
  QQA <= Count;

  PROCESS ( Clock, Resetn )
  BEGIN
    Q <= Count;
    IF Resetn = '0' THEN
      Count <= "0000";
      ELSIF (Clock'EVENT AND Clock = '1') THEN
        forloop: FOR i IN 0 TO 3 LOOP
          Count <= Count + 1;
          Z1 <= Count(0);
        END LOOP;
        IF (Count = "0000") THEN -- This IF statement is used for Problem 5(b)
          Z2 <= '1';
          ELSE
            Z2 <= '0';
          END IF;
        QQB <= Count;
        ELSE
          COUNT <= COUNT + 1;
          Z1 <= COUNT(0);
          END IF;
    END PROCESS;
    END Behavior;
4. VHDL analysis – Timing diagram.

(a) Based on the code shown on Page 7, complete the following timing diagram for the values for Count (in binary): (The initial contents of all flipflops are unknown. Also, go as far as you can.) (3 pts)

(b) Assume the given Count values, complete the following timing diagram based on the code shown on Page 7 (i.e., don't use the code to figure out the Count values, they are given to you for this part of the problem). Give all values in binary. (The initial contents of all flipflops are unknown. Also, go as far as you can.) (15 pts)
5. VHDL Analysis and re-code

(a) Given the IF statement at the bottom of the code shown on Page 7, convert it into a SELECT assignment statement that can be specified outside the PROCESS block. (3 pts)

**IMPORTANT:** If necessary, you need to tell me what else you need to add to the code (if anything) to maintain the behavior of the original code.

```
WITH Count SELECT
  Z3 <= '1' WHEN "0000";
  '0' WHEN OTHERS;
```

(b) Given the IF statement in the middle of the code shown on Page 7, convert it into a conditional assignment statement that can be specified outside the PROCESS block. (6 pts)

**IMPORTANT:** If necessary, you need to tell me what else you need to add to the code (if anything) to maintain the behavior of the original code.

```
SIGNAL temp2 : STD_LOGIC;

temp2 <= '1' WHEN count = "0000" ELSE
  '0'; -- This statement is adequate
       -- of the process statement
       -- to replace the logic

-- Inside the process block, we still need
-- the following to implement the flip-flop.
Z2 <= temp2;
```
6. Other topics.

(a) Let us assume you have the components from Lab 3: adder2lca (2-bit adder) and lca2gen (2-bit look-ahead carry generator). (3 pts.)

For a 64-bit adder, how many lca2gen components will you need? \[31\]

For a 64-bit adder, how many levels of lca2gen will you need? \[5\]

(b) Let us assume you have the components from Lab 3: adder2lca (2-bit adder) and lca2gen (2-bit look-ahead carry generator). (3 pts.)

For a 48-bit adder, how many lca2gen components will you need? \[23\]

For a 48-bit adder, how many levels of lca2gen will you need? \[5\]

(c) Assume you are designing an 8-bit adder to perform two's complement addition. Given me the logic table and equation for OV (two's complement overflow) as a function of the sign bits of the two operands and the sum. (3 pts)

Put logic table here: 

<table>
<thead>
<tr>
<th>A7</th>
<th>B7</th>
<th>S7</th>
<th>OV</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 0 1 0</td>
<td>0 0 1 0</td>
<td></td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 1 0 0</td>
<td>0 1 0 0</td>
<td></td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>0 1 1 0</td>
<td>0 1 1 0</td>
<td></td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>1 0 0 0</td>
<td>1 0 0 0</td>
<td></td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1 0 1 0</td>
<td>1 0 1 0</td>
<td></td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>1 1 0 0</td>
<td>1 1 0 0</td>
<td></td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1 1 1 0</td>
<td>1 1 1 0</td>
<td></td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>0 0 1 1</td>
<td>0 0 1 1</td>
<td></td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>0 1 0 0</td>
<td>0 1 0 0</td>
<td></td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>0 1 1 1</td>
<td>0 1 1 1</td>
<td></td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
<td></td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>0 0 1 0</td>
<td>0 0 1 0</td>
<td></td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>0 1 0 1</td>
<td>0 1 0 1</td>
<td></td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>0 1 1 0</td>
<td>0 1 1 0</td>
<td></td>
</tr>
</tbody>
</table>

Put equation for OV here:

\[OV = \overline{A7} \cdot B7 \cdot S7 + A7 \cdot B7 \cdot \overline{S7}\]

(d) Compare/contrast the function of a logic state analyzer (LSA) vs. the function of an oscilloscope. (2 pts)

LSA: capture the logical values of signals of a digital circuit and displays them over time.
Oscilloscope: capture and display the voltage value of signals over time.

(e) Compare/contrast the functions of an LSA, Quartus simulation, and Quartus SignalTap. (3 pts)

They all capture the logical values of signals and display them over time.
LSA: trace external signals of a physical chip.
SignalTap: trace signals within a physical FPGA.
Quartus simulation - using models of an FPGA, simulate the results before downloading onto the FPGA.