Problem 1 (20 pts)

- **Digital Stopwatch**: The architecture of a digital stopwatch is provided. We require counters with an output 'z'. This output is asserted (for one clock cycle) when the counter reaches its maximum count.
  - Counter (0.01s). It counts up to \(10^6 - 1\), asserting 'z' when the count reaches \(10^6 - 1\). For an input clock frequency of 100 MHz, 'z' is asserted every 0.01 s.
  - BCD counter: It counts up to 9, asserting 'z' when the count reaches 9.
  - Modulo-6 counter: It counts up to 5, asserting 'z' when the count reaches 5.
- **NEXYS3 implementation details**: Only one 7-segment display can be used at a time → we serialize the four BCD outputs. In order for each digit to appear bright and continuously illuminated, we illuminate each digit for only 1ms every 4 ms. This is taken care of feeding the output 'z' of the counter to 0.001s to the enable input of the FSM.

Provide the Finite State Machine of the serializer in ASM (Algorithmic State Machine) form.
Provide the VHDL description of the entire circuit: FSM + Datapath circuit.

Tip: If you want to simulate your circuit, do not include the Counter to 0.01s and the Counter to 0.001s (instead, set the output signal 'z' to '1'). Otherwise, you might not be able to simulate your circuit.
** FSM (ASM chart):**

```
S1
resetn=0
s ← 00
E
0

S2
s ← 01
E
0

S3
s ← 10
E
0

S4
s ← 11
E
0
```

** VHDL code: 'dig_stopwatch.vhd'**

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.math_real.log2;
use ieee.math_real.ceil;

entity dig_stopwatch is
    port (resetn, clock, pause: in std_logic;
          segs: out std_logic_vector (6 downto 0);
          EN: out std_logic_vector (3 downto 0));
end dig_stopwatch;

architecture Behavioral of dig_stopwatch is

component my_genpulse
    generic (COUNT: INTEGER:= (10**8)/2); -- (10**8)/2 cycles of T = 10 ns --> 0.5 s
    port (clock, resetn, E: in std_logic;
          Q: out std_logic_vector ( integer(ceil(log2(real(COUNT)))) - 1 downto 0);
          z: out std_logic);
end component;

component sevenseg
    port (bcd: in std_logic_vector (3 downto 0);
          sevseg: out std_logic_vector (6 downto 0);
          EN: out std_logic_vector (3 downto 0));
end component;

signal npause, z, z_0, z_1, z_2, z_3, E_1, E_2, E_3: std_logic;

signal omux, Q_0, Q_1, Q_2, Q_3: std_logic_vector (3 downto 0);

signal s: std_logic_vector (1 downto 0);

signal E_fsm: std_logic;

type state is (S1, S2, S3, S4);

signal y: state;

begin

npause <= not (pause);
Q_3(3) <= '0';

-- Counter: 0.01s
gz: my_genpulse generic map (COUNT => 10**6)
   port map (clock => clock, resetn => resetn, E => npause, z => z);

-- Counter: 10
g0: my_genpulse generic map (COUNT => 10)
   port map (clock => clock, resetn => resetn, E => z, Q => Q_0, z => z_0);
```

Instructor: Daniel Llamocca
-- Counter: 10
g1: my_genpulse generic map (COUNT => 10)
  port map (clock => clock, resetn => resetn, E => E_1, Q => Q_1, z => z_1);
  E_1 <= z and z_0;

-- Counter: 10
g2: my_genpulse generic map (COUNT => 10)
  port map (clock => clock, resetn => resetn, E => E_2, Q => Q_2, z => z_2);
  E_2 <= E_1 and z_1;

-- Counter: 6
g3: my_genpulse generic map (COUNT => 6)
  port map (clock => clock, resetn => resetn, E => E_3, Q => Q_3 (2 downto 0), z => z_3);
  E_3 <= E_2 and z_2;

-- Multiplexor
with s select
  omux <= Q_0 when "00",
           Q_1 when "01",
           Q_2 when "10",
           Q_3 when others;

seg7: sevenseg port map (bcd => omux, sevseg => segs);

-- 2-to-4 decoder
with s select
  EN <= "1110" when "00",
       "1101" when "01",
       "1011" when "10",
       "0111" when "11",
       "1111" when others;

-- Counter: 0.001s
gfsm: my_genpulse generic map (COUNT => 10**5)
  port map (clock => clock, resetn => resetn, E => '1', z => E_fsm);

Transitions: process (resetn, clock, E_fsm)
begin
  if resetn = '0' then -- asynchronous signal
    y <= S1; -- if resetn asserted, go to initial state: S1
  elsif (clock'event and clock = '1') then
    if E_fsm = '1' then
      case y is
        when S1 => y <= S2;
        when S2 => y <= S3;
        when S3 => y <= S4;
        when S4 => y <= S1;
      end case;
    end if;
  end if;
end process;

Outputs: process (y)
begin
  case y is
    when S1 => s <= "00";
    when S2 => s <= "01";
    when S3 => s <= "10";
    when S4 => s <= "11";
  end case;
end process;
end Behavioral;
**PROBLEM 2 (2.5 pts)**

- Design of a configurable lights' pattern generator. \textit{sel}: selects the pattern. \textit{stop}: freezes the pattern when \textit{stop}=1. Two 7-segment displays are used. The datapath circuit is provided.
- Input 'x': Selects the rate at which the lights' pattern change (every 1.5, 1.0, 0.5, or 0.25 seconds)

- Provide both Finite State Machines in ASM form. ASM: Algorithmic State Machine.
- Provide the VHDL description of the entire circuit: FSMs + Datapath circuit.
- Main FSM (ASM chart):
FSM for 7-segment display control (ASM chart):

VHDL code: 'lights_pattern.vhd'

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.math_real.log2;
use ieee.math_real.ceil;

designation lights_pattern is
  port (resetn, clock, stop: in std_logic;
        x, sel: in std_logic_vector (1 downto 0);
        segs: out std_logic_vector (6 downto 0);
        EN: out std_logic_vector (3 downto 0));
end lights_pattern;

architecture Behavioral of lights_pattern is

  component my_genpulse
    generic (COUNT: INTEGER:= (10**8)/2); -- (10**8)/2 cycles of T = 10 ns --> 0.5 s
    port (clock, resetn, E: in std_logic;
          Q: out std_logic_vector ( integer(ceil(log2(real(COUNT)))) - 1 downto 0);
          z: out std_logic);
  end component;

  component my_rege
    generic (N: INTEGER:= 4);
    port ( clock, resetn: in std_logic;
           E, sclr: in std_logic; -- sclr: Synchronous clear
           D: in std_logic_vector (N-1 downto 0);
           Q: out std_logic_vector (N-1 downto 0));
  end component;

type state is (S1, S2a, S3a, S4a, S5a, S6a, S7a, S8a, S2b, S3b, S4b, S2c, S3c, S4c, S2d, S3d, S4d, S5d, S6d);
  signal y: state;

type stateo is (S1, S2);
  signal yo: stateo;

  signal E_fsm, E, Esg, za, zb, zc, zd, z, s: std_logic;
  signal dseg, seg: std_logic_vector (7 downto 0);
  signal psegs: std_logic_vector (6 downto 0);

begin

  -- Counter: 1.5 s
  ga: my_genpulse generic map (COUNT => 3*((10**8)/2))
      port map (clock => clock, resetn => resetn, E => '1', z => za);

  -- Counter: 1.0 s
  gb: my_genpulse generic map (COUNT => 10**8)
      port map (clock => clock, resetn => resetn, E => '1', z => zb);

  -- Counter: 0.5 s
  gc: my_genpulse generic map (COUNT => (10**8)/2)
      port map (clock => clock, resetn => resetn, E => '1', z => zc);

  -- Counter: 0.25 s
  gd: my_genpulse generic map (COUNT => (10**8)/4)
      port map (clock => clock, resetn => resetn, E => '1', z => zd);
```

Instructor: Daniel Llamocca
-- Multiplexor for the counter outputs:
with x select
z <= za when "00",
zb when "01",
zc when "10",
zd when others;

E <= z and not(stop);

-- 8-bit register
rP: my_rege generic map (N => 8)
port map (clock => clock, resetn => resetn, E => Esg, sclr => '0', D => dseg, Q => seg);

-- Multiplexor for the 7-segment displays:
with s select
psegs <= seg(6 downto 3)&"000" when '0',
    seg(7)&"000"&seg(2 downto 0)&'0' when others;
-- segs: |a|b|c|d|e|f|g|
segf <= not(psegs); -- Active-low outputs

-- Counter: 0.001 s
gfsm: my_genpulse generic map (COUNT => 10**5)
port map (clock => clock, resetn => resetn, E => '1', z => E_fsm);

-- 1-to-2 decoder
with s select
EN <= "1011" when '0',
    "0111" when others;

-- Main FSM:
Transitions: process (resetn, clock, E, sel)
begin
    if resetn = '0' then -- asynchronous signal
        y <= S1; -- if resetn asserted, go to initial state: S1
    elsif (clock'event and clock = '1') then
        case y is
            when S1 =>
                if E = '1' then
                    case sel is
                        when "00" => y <= S2a;
                        when "01" => y <= S2b;
                        when "10" => y <= S2c;
                        when others => y <= S2d;
                    end case;
                else
                    y <= S1;
                end if;
            when S2a => if E = '1' then y <= S3a; else y <= S2a; end if;
            when S3a => if E = '1' then y <= S4a; else y <= S3a; end if;
            when S4a => if E = '1' then y <= S5a; else y <= S4a; end if;
            when S5a => if E = '1' then y <= S6a; else y <= S5a; end if;
            when S6a => if E = '1' then y <= S7a; else y <= S6a; end if;
            when S7a => if E = '1' then y <= S8a; else y <= S7a; end if;
            when S8a => if E = '1' then y <= S1; else y <= S8a; end if;
            when S2b => if E = '1' then y <= S3b; else y <= S2b; end if;
            when S3b => if E = '1' then y <= S4b; else y <= S3b; end if;
            when S4b => if E = '1' then y <= S1; else y <= S4b; end if;
            when S2c => if E = '1' then y <= S3c; else y <= S2c; end if;
            when S3c => if E = '1' then y <= S4c; else y <= S3c; end if;
            when S4c => if E = '1' then y <= S1; else y <= S4c; end if;
            when S2d => if E = '1' then y <= S3d; else y <= S2d; end if;
            when S3d => if E = '1' then y <= S4d; else y <= S3d; end if;
            when S4d => if E = '1' then y <= S5d; else y <= S4d; end if;
            when S5d => if E = '1' then y <= S6d; else y <= S5d; end if;
            when S6d => if E = '1' then y <= S1; else y <= S6d; end if;
        end case;
    end if;
end process;
Outputs: process (y, E, sel)
begin
   -- Initialization of FSM outputs:
   dseg <= (others => '0'); Esg <= '0';
   case y is
      when S1 =>
         if E = '1' then
            case sel is
               when "00" => dseg <= "00000111"; Esg <= '1';
               when "01" => dseg <= "00000101"; Esg <= '1';
               when "10" => dseg <= "00110001"; Esg <= '1';
               when others => dseg <= "00110011"; Esg <= '1';
            end case;
         end if;
      end case;
end if;
when S2a => if E = '1' then dseg <= "00010100"; Esg <= '1'; end if;
when S3a => if E = '1' then dseg <= "00011100"; Esg <= '1'; end if;
when S4a => if E = '1' then dseg <= "01110000"; Esg <= '1'; end if;
when S5a => if E = '1' then dseg <= "00010100"; Esg <= '1'; end if;
when S6a => if E = '1' then dseg <= "00011100"; Esg <= '1'; end if;
when S7a => if E = '1' then dseg <= "01110000"; Esg <= '1'; end if;
when S8a => if E = '1' then dseg <= "00010100"; Esg <= '1'; end if;
when S2b => if E = '1' then dseg <= "00010100"; Esg <= '1'; end if;
when S3b => if E = '1' then dseg <= "00011100"; Esg <= '1'; end if;
when S4b => if E = '1' then dseg <= "01110000"; Esg <= '1'; end if;
when S5b => if E = '1' then dseg <= "00010100"; Esg <= '1'; end if;
when S6b => if E = '1' then dseg <= "00011100"; Esg <= '1'; end if;
when S7b => if E = '1' then dseg <= "01110000"; Esg <= '1'; end if;
when S8b => if E = '1' then dseg <= "00010100"; Esg <= '1'; end if;
when S2c => if E = '1' then dseg <= "00010100"; Esg <= '1'; end if;
when S3c => if E = '1' then dseg <= "00011100"; Esg <= '1'; end if;
when S4c => if E = '1' then dseg <= "01110000"; Esg <= '1'; end if;
when S5c => if E = '1' then dseg <= "00010100"; Esg <= '1'; end if;
when S6c => if E = '1' then dseg <= "00011100"; Esg <= '1'; end if;
when S7c => if E = '1' then dseg <= "01110000"; Esg <= '1'; end if;
when S8c => if E = '1' then dseg <= "00010100"; Esg <= '1'; end if;
   end case;
end process;

-- FSM: serializer for the 7-segment display
Trans: process (resetn, clock, E_fsm)
begin
   if resetn = '0' then -- asynchronous signal
      yo <= S1; -- if resetn asserted, go to initial state: S1
   elsif (clock'event and clock = '1') then
      if E_fsm = '1' then
         case yo is
            when S1 => yo <= S2;
            when S2 => yo <= S1;
         end case;
      end if;
   end if;
end process;

Outps: process (yo)
begin
   case yo is
      when S1 => s <= '0';
      when S2 => s <= '1';
   end case;
end process;
end Behavioral;
PROBLEM 3 (20 pts)

- The figure depicts the architecture of a simple processor.

- Register G: Parallel Access left-shift register with enable. \( s_l = 1 \) → Load, \( s_l = 0 \) → Left-shift

- The figure below depicts the datapath and the FSM of the Control Circuit:

\[
\text{funq} = |f_2|f_1|f_0|Ry_1|Ry_0|Rx_1|Rx_0|
\]
The following table specifies the behavior of the Arithmetic-Logic Unit (ALU). This ALU is purely combinatorial.

<table>
<thead>
<tr>
<th>op</th>
<th>Operation</th>
<th>Function</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>y &lt;= A</td>
<td>Transfer 'A'</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>y &lt;= A + 1</td>
<td>Increment 'A'</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>y &lt;= A - 1</td>
<td>Decrement 'A'</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>y &lt;= B</td>
<td>Transfer 'B'</td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>y &lt;= B + 1</td>
<td>Increment 'B'</td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>y &lt;= B - 1</td>
<td>Decrement 'B'</td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>y &lt;= A + B</td>
<td>Add 'A' and 'B'</td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td>y &lt;= A - B</td>
<td>Subtract 'B' from 'A'</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>y &lt;= not A</td>
<td>Complement 'A'</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>y &lt;= not B</td>
<td>Complement 'B'</td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>y &lt;= A AND B</td>
<td>AND</td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>y &lt;= A OR B</td>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>y &lt;= A NAND B</td>
<td>NAND</td>
<td></td>
</tr>
<tr>
<td>1101</td>
<td>y &lt;= A NOR B</td>
<td>NOR</td>
<td></td>
</tr>
<tr>
<td>1110</td>
<td>y &lt;= A XOR B</td>
<td>XOR</td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>y &lt;= A XNOR B</td>
<td>XNOR</td>
<td></td>
</tr>
</tbody>
</table>

Operation: Every time w = '1', we store the input fun, then proceed to execute it.

The 7 bits of the stored input, called funq, are arranged as: funq = [f2|f1|f0|Ry1|Ry0|Rx1|Rx0]. The first 3 bits specify the operation, while the other 4 bits specify the registers over which the operations are applied.

<table>
<thead>
<tr>
<th>f</th>
<th>Operation</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Load Rx, Data</td>
<td>Rx ← Data</td>
</tr>
<tr>
<td>001</td>
<td>Add Rx, Data</td>
<td>Rx ← Rx + Data</td>
</tr>
<tr>
<td>010</td>
<td>Not Rx</td>
<td>Rx ← NOT (Rx)</td>
</tr>
<tr>
<td>011</td>
<td>Sub Rx, Ry</td>
<td>Rx ← Rx - Ry</td>
</tr>
<tr>
<td>100</td>
<td>Add Rx, Ry</td>
<td>Rx ← Rx + Ry</td>
</tr>
<tr>
<td>101</td>
<td>Move Rx, Ry</td>
<td>Rx ← Ry</td>
</tr>
<tr>
<td>110</td>
<td>sla Rx</td>
<td>Rx ← left-shift Rx</td>
</tr>
<tr>
<td>111</td>
<td>Addi Rx, 2</td>
<td>Rx ← Rx + 2</td>
</tr>
</tbody>
</table>

Design the Finite State Machine (provide it in ASM form) that can issue the correct set of signals for all the listed operations.
- **FSM (ASM chart):**

![ FSM Diagram ]

- **Problem 4 (10 pts)**
  - For the following error-detection system, provide the truth table and sketch the circuits of i) the Odd Parity Generator, and ii) Odd parity checker.

  ![ Error-detection System Diagram ]

  - **Odd parity generator:** TPB is such that $\{A|B|C|TPB\}$ has an odd number of bits.
  - **Odd parity checker:** Checks whether $\{A|B|C|TPB\}$ has an odd number of bits. If this is the case, $RPB = '0'$, otherwise $RPB = '1'$ (signaling an error).
Sketch the schematics of an n-bit comparator of two numbers represented in 2’s complement. Required outputs: \( \text{Age}_B, \text{Al}_B, \text{Ae}_B \). If \( A \geq B \rightarrow \text{Age}_B = 1 \). If \( A < B \rightarrow \text{Al}_B = 1 \). If \( A = B \rightarrow \text{Ae}_B = 1 \). You can use adders (specify the number of bits) and XOR gates. Make sure that your circuit works in all circumstances (consider that \( A \) and \( B \) may cause overflow, you need to avoid it).
**PROBLEM 5 (25 pts)**

- **LUT connected to a bidirectional port:** An LUT4-to-1 (also called LUT4) can implement a 4-input function. The VHDL code is provided below. Note that the values stored in the LUT are constant, hence those values are entered as a parameter in the VHDL code:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity my4to1LUT is
    generic (data: std_logic_vector(15 downto 0):=x"FEAB"); -- LUT 4-to-1 contents
    port ( ILUT: in std_logic_vector (3 downto 0);
           OLUT: out std_logic);
end my4to1LUT;

architecture Behavioral of my4to1LUT is
begin
    with ILUT select
    begin
        O_LUT <= data(0) when "0000",
                 data(1) when "0001",
                 data(2) when "0010",
                 data(3) when "0011",
                 data(4) when "0100",
                 data(5) when "0101",
                 data(6) when "0110",
                 data(7) when "0111",
                 data(8) when "1000",
                 data(9) when "1001",
                 data(10) when "1010",
                 data(11) when "1011",
                 data(12) when "1100",
                 data(13) when "1101",
                 data(14) when "1110",
                 data(15) when "1111",
                 '0' when others;
    end Behavioral;
end Behavioral;
```

- **LUTs with more than four inputs can be built by grouping several LUT 4-to-1 and MUXs. The figure below shows how an LUT 6-to-1 can be built.**
- **An LUT with more than one output can also be built. The figure below show how we can create an LUT 6-to-6 (we just use six LUT 6-to-1).**

You are asked to design a system with an LUT 6-to-6. This will be accomplished in a series of steps:

- Provide the VHDL code of an LUT 6-to-1. You can built it by i) grouping four LUT4-to-1 and MUXs, or ii) using only one VHDL file (similar to `my4to1LUT.vhd`). The entity should look like this:

```vhdl
entity my6to1LUT is
    generic (data: std_logic_vector(63 downto 0):=x"FEAB97CA003E19CC"); -- 64 values
    port ( ILUT: in std_logic_vector (5 downto 0);
           OLUT: out std_logic);
end my6to1LUT;
```
Write the VHDL code for an LUT 6-to-6. You must build it by grouping six LUT 6-to-1. The entity should look like this:

```vhdl
entity my6to6LUT is
    generic (data5: std_logic_vector(63 downto 0):=x"FEAB97CA003E19CC"; -- column 5
        data4: std_logic_vector(63 downto 0):=x"AABBCEBDEC89EA98"; -- column 4
        data3: std_logic_vector(63 downto 0):=x"E595BE6BBEBBE197B"; -- column 3
        data2: std_logic_vector(63 downto 0):=x"FACE090993E3EECAB"; -- column 2
        data1: std_logic_vector(63 downto 0):=x"DECAFF99E32098A"; -- column 1
        data0: std_logic_vector(63 downto 0):=x"ACADE412BAFE125E"); -- column 0
    port ( ILUT: in std_logic_vector (5 downto 0);
        OLUT: out std_logic_vector (5 downto 0));
end my6to6LUT;
```

Important: When instantiating the my6to1LUT component, we use the port map instruction to make interconnections. Now, we also need to provide the correct parameter to each my6to1LUT component. This is done usually with the generic map instruction.

**Final System:** Provide the VHDL code of the circuit depicted below. Use the Structural Description by interconnecting the following components: i) LUT 6-to-6, ii) 6-bit register, and iii) Tri-state buffers. Important: The port 'DATA' can be input or output at different times. Use INOUT in your VHDL code.

**LUT6-to-6 contents:** We want this LUT 6-to-6 to provide the following function: \( OLUT = \lfloor ILUT^{0.95} \rfloor \)

Example: \( ILUT = 35 \ (010011_2) \) \( \Rightarrow \) \( OLUT = [35^{0.95}] = 30 \ (011111_2) \)

Compute the contents of the LUT 6-to-6 and provide each column in hexadecimal format as:

- data5, data4, data3, data2, data1, data0 (generic input parameters)

Complete the Timing diagram shown below. Note that the port DATA is input at some times, and output at other times.
**LUT6-to-6 contents:**

![LUT 6-to-6: 63
0 0 0 0 C A 0 0 0 F C A 0 0 F 0 8 6 0 0 F F E 9 5 0 C 3 3 3 A 0 F 0 8 7 6 0 F 0 F E D 0 F F 0 C A 0 F F F E 9 5 C 3 3 3 3 B F 0 0 8 7 6 F 0 0 F E 9 F 0 F 0 C 2 F 0 F F E 9 5 F C 3 3 3 A F F 0 8 6 5

**VHDL code:** `sysLUT6to6.vhd`

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sysLUT6to6 is
  generic (data5: std_logic_vector(63 downto 0):=x"ffffffc000000000";  -- column 5
            data4: std_logic_vector(63 downto 0):=x"f c0000d000000000";  -- column 4
            data3: std_logic_vector(63 downto 0):=x"0 3ff003ff003ff00";  -- column 3
            data2: std_logic_vector(63 downto 0):=x"8 3e0f83e0f83e0f0";  -- column 2
            data1: std_logic_vector(63 downto 0):=x"6 39ce739ce739ce7398cc";  -- column 1
            data0: std_logic_vector(63 downto 0):=x"5 a5296b5ad6a56aa");  -- column 0
  port (clk, resetn, OE: in std_logic;
        data: inout std_logic_vector (5 downto 0));
end sysLUT6to6;

architecture structure of sysLUT6to6 is

  component my_rege
    generic (N: INTEGER:= 4);
    port ( clock, resetn, E, sclr: in std_logic; -- sclr: Synchronous clear
           D: in std_logic_vector (N-1 downto 0);
           Q: out std_logic_vector (N-1 downto 0));
  end component;

  component my6to6LUT
    generic (data5: std_logic_vector(63 downto 0):=x"FEAB97CA003E19CC";  -- column 5
             data4: std_logic_vector(63 downto 0):=x"AABBCCFFEE99098A";  -- column 4
             data3: std_logic_vector(63 downto 0):=x"E595BEBECAFEDADA";  -- column 3
             data2: std_logic_vector(63 downto 0):=x"FACE090909090909";  -- column 2
             data1: std_logic_vector(63 downto 0):=x"DECAFFE090909090";  -- column 1
             data0: std_logic_vector(63 downto 0):=x"ACADE412BAFE125E");  -- column 0
    port ( ILUT: in std_logic_vector (5 downto 0);
           OLUT: out std_logic_vector (5 downto 0));
  end component;

  signal DI, DO, QR: std_logic_vector (5 downto 0);

begin

  ri: my_rege generic map (N => 6)
    port map (clock => clk, resetn => resetn, E => OE, sclr => '0', D => DI, Q => QR);

  LUT6to6: my6to6LUT generic map (data5 => data5, data4 => data4, data3 => data3, data2 => data2,
                                 data1 => data1, data0 => data0)
    port map (ILUT => QR, OLUT => DO);

  DATA <= DO when OE = '0' else (others => 'Z');
  DI <= DATA when OE = '1' else (others => 'Z');

end structure;
```
• **VHDL code: 'my6to6LUT'**

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity my6to6LUT is
  generic (data5: std_logic_vector(63 downto 0):=x"FEAB97CA003E19CC");  -- column 5
  data4: std_logic_vector(63 downto 0):=x"AABBCCFFEE99098A";  -- column 4
  data3: std_logic_vector(63 downto 0):=x"E595BEBCAFEDADA";  -- column 3
  data2: std_logic_vector(63 downto 0):=x"FACE09093E3EECAB";  -- column 2
  data1: std_logic_vector(63 downto 0):=x"DEAFFF09EA3200";  -- column 1
  data0: std_logic_vector(63 downto 0):=x"ACADE412BAFE125E";  -- column 0
  port ( ILUT: in std_logic_vector (5 downto 0);
        OLUT: out std_logic_vector (5 downto 0));
end my6to6LUT;

architecture Behavioral of my6to6LUT is
  component my6to1LUT
    generic (data: std_logic_vector(63 downto 0):=x"ACCABEBEFACEFEAB");
    port ( ILUT: in std_logic_vector (5 downto 0);
          OLUT: out std_logic);
  end component;

  signal OLUT_l, OLUT_h: std_logic;

begin
  -- 6-to-1 LUT holding contents of column 5:
  r5: my6to1LUT generic map (data => data5)
    port map (ILUT => ILUT, OLUT => OLUT(5));
  -- 6-to-1 LUT holding contents of column 4:
  r4: my6to1LUT generic map (data => data4)
    port map (ILUT => ILUT, OLUT => OLUT(4));
  -- 6-to-1 LUT holding contents of column 3:
  r3: my6to1LUT generic map (data => data3)
    port map (ILUT => ILUT, OLUT => OLUT(3));
  -- 6-to-1 LUT holding contents of column 2:
  r2: my6to1LUT generic map (data => data2)
    port map (ILUT => ILUT, OLUT => OLUT(2));
  -- 6-to-1 LUT holding contents of column 1:
  r1: my6to1LUT generic map (data => data1)
    port map (ILUT => ILUT, OLUT => OLUT(1));
  -- 6-to-1 LUT holding contents of column 0:
  r0: my6to1LUT generic map (data => data0)
    port map (ILUT => ILUT, OLUT => OLUT(0));
end Behavioral;
```

• **VHDL code: 'my6to1LUT'**

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity my6to1LUT is
  generic (data: std_logic_vector(63 downto 0):=x"ACCABEBEFACEFEAB");
  port ( ILUT: in std_logic_vector (5 downto 0);
        OLUT: out std_logic);
end my6to1LUT;

architecture Behavioral of my6to1LUT is
  component my5to1LUT
    generic (data: std_logic_vector(31 downto 0):=x"FACEFEAB");
    port ( ILUT: in std_logic_vector (4 downto 0);
        OLUT: out std_logic);
  end component;

  signal OLUT_l, OLUT_h: std_logic;

begin
  -- 5-to-1 LUT holding contents: data(31 downto 0)
  rl: my5to1LUT generic map (data => data(31 downto 0))
    port map (ILUT => ILUT(4 downto 0), OLUT => OLUT_l);
  -- 5-to-1 LUT holding contents: data(63 downto 32)
  rh: my5to1LUT generic map (data => data(63 downto 32))
    port map (ILUT => ILUT(4 downto 0), OLUT => OLUT_h);

  with ILUT(5) select
    OLUT <= OLUT_l when '0',
            OLUT_h when others;
end Behavioral;
```
- **VHDL code: 'my5to1LUT'**

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity my5to1LUT is
  generic (data: std_logic_vector(31 downto 0):=x"FACEFEAB);
  port ( ILUT: in std_logic_vector (4 downto 0);
         O_LUT: out std_logic);
end my5to1LUT;

architecture Behavioral of my5to1LUT is
  component my4to1LUT
    generic (data: std_logic_vector(15 downto 0):=x"FEAB);
    port ( ILUT: in std_logic_vector (3 downto 0);
           O_LUT: out std_logic);
  end component;
  signal O_LUT_l, O_LUT_h: std_logic;
  begin
    -- 4-to-1 LUT holding contents: data(15 downto 0)
    r1: my4to1LUT generic map (data => data(15 downto 0))
      port map (ILUT => ILUT(3 downto 0), O_LUT => O_LUT_l);

    -- 4-to-1 LUT holding contents: data(31 downto 16)
    r2: my4to1LUT generic map (data => data (31 downto 16))
      port map (ILUT => ILUT(3 downto 0), O_LUT => O_LUT_h);
    with ILUT(4) select
      O_LUT <= O_LUT_l when '0',
               O_LUT_h when others;
  end Behavioral;
```

- **VHDL Testbench**

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use ieee.std_logic_arith.all;

ENTITY tb_sysLUT6to6 IS
END tb_sysLUT6to6;

ARCHITECTURE behavior OF tb_sysLUT6to6 IS
  -- Component Declaration for the Unit Under Test (UUT)
  COMPONENT sysLUT6to6
    PORT (clk : IN  std_logic;
           resetn : IN  std_logic;
           OE : IN  std_logic;
           data : INOUT std_logic_vector(5 downto 0));
  END COMPONENT;

  --Inputs
  signal clk : std_logic := '0';
  signal resetn : std_logic := '0';
  signal OE : std_logic := '0';

  --BiDirs
  signal data : std_logic_vector(5 downto 0);

  -- Clock period definitions
  constant clk_period : time := 10 ns;

  BEGIN
    -- Instantiate the Unit Under Test (UUT)
    uut: sysLUT6to6 PORT MAP (clk => clk, resetn => resetn, OE => OE, data => data);

    -- Clock process definitions
    clk_process :process
    begin
      clk <= '0';
      wait for clk_period/2;
      clk <= '1';
      wait for clk_period/2;
    end process;
```

Instructor: Daniel Llamocca
-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   resetn <= '0'; DATA <= "ZZZZZZ"; wait for 100 ns;
   resetn <= '1'; wait for clk_period;
   OE <= '1'; DATA <= "110001"; wait for 2*clk_period;
   OE <= '0'; DATA <= "ZZZZZZ"; wait for 2*clk_period;
   OE <= '1'; DATA <= "100111"; wait for 2*clk_period;
   OE <= '0'; DATA <= "ZZZZZZ"; wait for 2*clk_period;
   OE <= '1'; DATA <= "100001"; wait for 2*clk_period;
   OE <= '0'; DATA <= "ZZZZZZ"; wait for 2*clk_period;
   OE <= '1'; DATA <= "ZZZZZZ"; wait for 2*clk_period;
   OE <= '0'; DATA <= "ZZZZZZ"; wait for 2*clk_period;
   wait for 4*clk_period;
   OE <= '1';
   lp: for i in 0 to 63 loop
      DATA <= conv_std_logic_vector(i,6); wait for clk_period;
   end loop;
end process;
END;

• Timing diagram:

<table>
<thead>
<tr>
<th>clk</th>
<th>resetn</th>
<th>OE</th>
<th>DATA</th>
<th>DI</th>
<th>DO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>10001</td>
<td>10100</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10001</td>
<td>100111</td>
<td>100001</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>000111</td>
<td>000111</td>
<td>000111</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>100001</td>
<td>011100</td>
<td>011100</td>
</tr>
</tbody>
</table>

EXTRA CREDIT (+10 pts)

• Demonstrate the circuits of Problems 1 and 2 working on the NEXYS3 board.

• UCF File for Problem 1: 'dig_stopwatch.ucf'

# Inputs
NET "clock" LOC = "V10";
NET "resetn" LOC = "T10"; #SW0
NET "pause" LOC = "T9"; #SW1

# Outputs
NET "EN<3>" LOC = "P17"; # anode (1st display from left to right)
NET "EN<2>" LOC = "P18"; # anode (2nd display from left to right)
NET "EN<1>" LOC = "N15"; # anode (3rd display from left to right)
NET "EN<0>" LOC = "N16"; # anode (4th display from left to right)
NET "segs<6>" LOC = "T17"; # a
NET "segs<5>" LOC = "T18"; # b
NET "segs<4>" LOC = "U17"; # c
NET "segs<3>" LOC = "U18"; # d
NET "segs<2>" LOC = "M14"; # e
NET "segs<1>" LOC = "N14"; # f
NET "segs<0>" LOC = "L14"; # g
UCF File for Problem 2: 'lights_pattern.ucf'

# Inputs
NET "clock" LOC = "V10";
NET "resetn" LOC = "T5"; # SW7
NET "stop" LOC = "B8"; # BTNS (center)
NET "x<1>" LOC = "T9"; # SW1
NET "x<0>" LOC = "T10"; # SW0
NET "sel<1>" LOC = "M8"; # SW3
NET "sel<0>" LOC = "V9"; # SW2

# Outputs
NET "EN<3>" LOC = "P17"; # anode (1st display from left to right)
NET "EN<2>" LOC = "P18"; # anode (2nd display from left to right)
NET "EN<1>" LOC = "N15"; # anode (3rd display from left to right)
NET "EN<0>" LOC = "N16"; # anode (4th display from left to right)
NET "segs<6>" LOC = "T17"; # a
NET "segs<5>" LOC = "T18"; # b
NET "segs<4>" LOC = "U17"; # c
NET "segs<3>" LOC = "U18"; # d
NET "segs<2>" LOC = "M14"; # e
NET "segs<1>" LOC = "N14"; # f
NET "segs<0>" LOC = "L14"; # g

APPENDIX: EXTRA VHDL FILES

VHDL code: 'my_genpulse.vhd'

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.math_real.log2;
use ieee.math_real.ceil;

entity my_genpulse is
    generic (COUNT: INTEGER:= (10**8)/2); -- (10**8)/2 cycles of T = 10 ns --> 0.5 s
    port (clock, resetn, E: in std_logic;
          Q: out std_logic_vector ( integer(ceil(log2(real(COUNT)))) - 1 downto 0);
          z: out std_logic);
end my_genpulse;

architecture Behavioral of my_genpulse is
    constant nbits: INTEGER:= integer(ceil(log2(real(COUNT))));
signal Qt: std_logic_vector (nbits -1 downto 0);
begin
    process (resetn, clock)
    begin
        if resetn = '0' then
            Qt <= (others => '0');
        elsif (clock'event and clock = '1') then
            if E = '1' then
                if Qt = conv_std_logic_vector (COUNT-1,nbits) then
                    Qt <= (others => '0');
                else
                    Qt <= Qt + conv_std_logic_vector (1,nbits);
                end if;
            end if;
        end if;
    end process;
    z <= '1' when Qt = conv_std_logic_vector (COUNT-1,nbits) else '0';
    Q <= Qt;
end Behavioral;
```
### VHDL code: 'my_rege'
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- N-bit Register
-- E = '1', sclr = '0' --> Input data 'D' is copied on Q
-- E = '1', sclr = '1' --> Q is cleared (0)
entity my_rege is
generic (N: INTEGER:= 4);
port ( clock, resetn: in std_logic;
    E, sclr: in std_logic; -- sclr: Synchronous clear
    D: in std_logic_vector (N-1 downto 0);
    Q: out std_logic_vector (N-1 downto 0));
end my_rege;
architecture Behavioral of my_rege is
    signal Qt: std_logic_vector (N-1 downto 0);
begin
    process (resetn, clock)
        begin
            if resetn = '0' then
                Qt <= (others => '0');
            elsif (clock'event and clock = '1') then
                if E = '1' then
                    if sclr = '1' then
                        Qt <= (others => '0');
                    else
                        Qt <= D;
                    end if;
                end if;
            end if;
        end process;
    Q <= Qt;
end Behavioral;
```

### VHDL code: 'sevenseg.vhd'
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

to entity sevenseg is
    port (bcd: in std_logic_vector (3 downto 0);
    sevseg: out std_logic_vector (6 downto 0);
    EN: out std_logic_vector(3 downto 0));
end sevenseg;
architecture structure of sevenseg is
    signal leds: std_logic_vector (6 downto 0);
begin
    leds <= "1111110" when "0000",
            "0110000" when "0001",
            "1111001" when "0100",
            "1011111" when "0111",
            "1011011" when "0011",
            "1111111" when "0010",
            "1110000" when "0001",
            "1111111" when "1000",
            "1111011" when "1001",
            "-------" when others;

    -- There are 4 7-seg displays that can be used. We will use only the first (from left to right):
    -- EN <= "0111"; -- only the first 7-seg display is activated.
    -- EN(3) goes to one 7-seg display. It goes to every LED anode.
    -- To provide a logic '1' to the anode, we need EN(3) to be zero (see circuit)
    sevseg <= not(leds);
end structure;
```