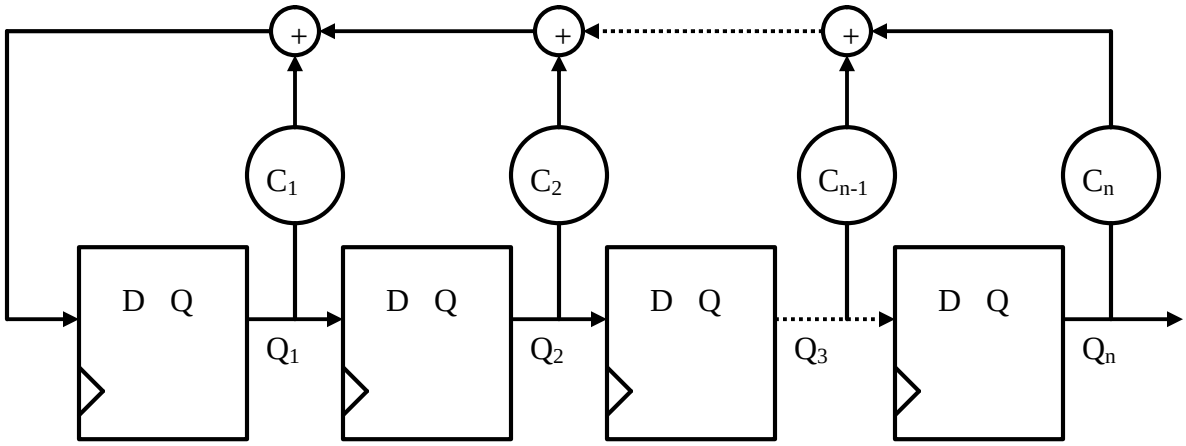
Random Number Generator

PURPOSE – In this lab you will design a Random Number Generator.

---

1. Introduction

In many real applications, such as electronic measurements (signature analyzers), communications, random (or better said pseudo random) number generators are needed. This laboratory is dedicated to designing a random number generator whose block diagram is shown in the figure below:



This circuit is known as an autonomous Linear Feedback Shift Register (LSFR). In particular, the circuit presented here is called type 1 LSFR. There are also other types of LSFR's but they are out of the scope of this lab. It has to be mentioned that this circuit is a pseudo random number generator due to its discrete nature.

The vector **c = c1c2...cn** is called the *seed* of the random number generator. It is fixed and should remain constant over time. It may be provided to the circuit as an input, which does not change over time. **cn** has to be always **'1'**.

ci is a binary constant, and **ci = '1'** implies that a connection exists, while **ci = '0'** implies that no connection exists. Not any binary combination for the vector **c** is a good one. Usually, the vector **c** is taken from existing tables for any particular value of **n**. An encircled '+' symbolizes an XOR gate.

2. Writing the VHDL description

You will design a random number generator that can generate different 8-bit number sequences based on a set of control inputs (ctrl). Each bit of ctrl decides if a XOR gate is used or not.

The following VHDL code describes a regular flip flop. Type it using any text editor (or using the VHDL editor of Project Manager) and save it as `ff.vhd`.

```
-------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity ff is port (
```

```
      clk, start : in std_logic;
      d : in std_logic;
      q : out std_logic);
end ff;

architecture ff_arch of ff is
begin
process (clk, start)
begin
if start = '0' then
      q <= '1';
elsif (clk'event and clk = '1') then
      q <= d;
end if;
end process;
end ff_arch;
```

The following VHDL code describes a clock divider. It is the same divider you used during the previous laboratories. Type it using any text editor (or using the VHDL editor of Project Manager) and save it as `clk_divider.vhd`.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity clk_divider is port (
      CIN : in std_logic;
        COUT : out std_logic);
end clk_divider;

architecture behavioral of clk_divider is
         constant TIMECONST : integer := 84;
         signal count0, count1, count2, count3 : integer range 0 to
1000;
         signal D : std_logic := '0';
begin
process (CIN)
begin
if (CIN'event and CIN = '1') then
    count0 <= count0 + 1;
    if count0 = TIMECONST then
       count0 <= 0;
       count1 <= count1 + 1;
    elsif count1 = TIMECONST then
       count1 <= 0;
       count2 <= count2 + 1;
    elsif count2 = TIMECONST then
       count2 <= 0;
       count3 <= count3 + 1;
    elsif count3 = TIMECONST then
       count3 <= 0;
       D <= not D;
    end if;
end if;
COUT <= D;
end process;
end;
```

The following VHDL code describes the random number generator design. Type it using any text editor (or using the VHDL editor of Project Manager) and save it as `rand.vhd`.

```
library ieee;
use ieee.std_logic_1164.all;
--
entity rand is port (
      clk, start : in std_logic;
      ctrl : in std_logic_vector(7 downto 0);
      led_u7 : out std_logic_vector(6 downto 0);
      led_u8 : out std_logic_vector(6 downto 0));
end rand;
--
architecture rand_arch of rand is

component ff port (
      clk, start : in std_logic;
      d : in std_logic;
      q : out std_logic);
end component;

component clk_divider port (
      CIN : in std_logic;
        COUT : out std_logic);
end component;

signal divided_clk : std_logic;
signal x7, x6, x5, x4, x3, x2, x1, x0 : std_logic;
signal d71, d72, d73, d74, d75, d76, d7 : std_logic;
signal randno_sig : std_logic_vector(7 downto 0);
--
begin
divider : clk_divider port map(clk, divided_clk);
ff7 : ff port map(divided_clk, start, d7, x7);
ff6 : ff port map(divided_clk, start, x7, x6);
ff5 : ff port map(divided_clk, start, x6, x5);
ff4 : ff port map(divided_clk, start, x5, x4);
ff3 : ff port map(divided_clk, start, x4, x3);
ff2 : ff port map(divided_clk, start, x3, x2);
ff1 : ff port map(divided_clk, start, x2, x1);
ff0 : ff port map(divided_clk, start, x1, x0);
--
d71process : process (ctrl, x0, x1)
begin
if ctrl(1 downto 0) = "11" then
      d71 <= x0 xor x1;
else    d71 <= x0;
end if;
end process d71process;
--
d72process : process (ctrl, d71, x2)
begin
if ctrl(2) = '1' then
      d72 <= d71 xor x2;
```

```
else    d72 <= d71;
end if;
end process d72process;
--
d73process : process (ctrl, d72, x3)
begin
if ctrl(3) = '1' then
      d73 <= d72 xor x3;
else    d73 <= d72;
end if;
end process d73process;
--
d74process : process (ctrl, d73, x4)
begin
if ctrl(4) = '1' then
      d74 <= d73 xor x4;
else    d74 <= d73;
end if;
end process d74process;
--
d75process : process (ctrl, d74, x5)
begin
if ctrl(5) = '1' then
      d75 <= d74 xor x5;
else    d75 <= d74;
end if;
end process d75process;
--
d76process : process (ctrl, d75, x6)
begin
if ctrl(6) = '1' then
      d76 <= d75 xor x6;
else    d76 <= d75;
end if;
end process d76process;
--
d7process : process (ctrl, d76, x7)
begin
if ctrl(7) = '1' then
      d7 <= d76 xor x7;
else    d7 <= d76;
end if;
end process d7process;
--
output : process(start, x7, x6, x5, x4, x3, x2, x1, x0)
begin
if (start = '0') then
      randno_sig <= (others => '1');
else
      randno_sig <= (x7 & x6 & x5 & x4 & x3 & x2 & x1 & x0);
end if;
end process;
--
display : process(randno_sig)
begin
case randno_sig(7 downto 4) is
when "0000" =>
```

```
      led_u7 <= "0000001";
when "0001" =>
      led_u7 <= "1001111";
when "0010" =>
      led_u7 <= "0010010";
when "0011" =>
      led_u7 <= "0000110";
when "0100" =>
      led_u7 <= "1001100";
when "0101" =>
      led_u7 <= "0100100";
when "0110" =>
      led_u7 <= "0100000";
when "0111" =>
      led_u7 <= "0001111";

when "1000" =>
      led_u7 <= "0000000";
when "1001" =>
      led_u7 <= "0000100";
when "1010" =>
      led_u7 <= "0001000";
when "1011" =>
      led_u7 <= "1100000";
when "1100" =>
      led_u7 <= "0110001";
when "1101" =>
      led_u7 <= "1000010";
when "1110" =>
      led_u7 <= "0110000";
when "1111" =>
      led_u7 <= "0111000";
when others =>
      led_u7 <= "1111111";
end case;
--
case randno_sig(3 downto 0) is
when "0000" =>
      led_u8 <= "0000001";
when "0001" =>
      led_u8 <= "1001111";
when "0010" =>
      led_u8 <= "0010010";
when "0011" =>
      led_u8 <= "0000110";
when "0100" =>
      led_u8 <= "1001100";
when "0101" =>
      led_u8 <= "0100100";
when "0110" =>
      led_u8 <= "0100000";
when "0111" =>
      led_u8 <= "0001111";

when "1000" =>
      led_u8 <= "0000000";
when "1001" =>
```

```
      led_u8 <= "0000100";
when "1010" =>
      led_u8 <= "0001000";
when "1011" =>
      led_u8 <= "1100000";
when "1100" =>
      led_u8 <= "0110001";
when "1101" =>
      led_u8 <= "1000010";
when "1110" =>
      led_u8 <= "0110000";
when "1111" =>
      led_u8 <= "0111000";
when others =>
      led_u8 <= "1111111";
end case;
end process;
end rand_arch;
-----------------------------------------------
```

3. Correct the code

Read the above VHDL files and convince yourself that they implement the circuit shown in the figure on the first page. Verify the correctness of all assignments of led_u7, and led_u8 within the case statements in the `rand.vhd`. Correct any error you may find. They will have to drive correctly the two 7-segment LEDs available on the XStend board.

4. Simulation

Simulate your design for various values of ctrl and see which ctrl combination gives you the maximum sequence length compared to the correct one: ctrl7=1, ctrl5=1, ctrl4=1, ctrl0=1 and all others (ctrl6, ctrl3, ctrl2, ctrl1) being zero.

5. Implementation and verification

You will have to write a **.ucf** file such that led_u7 will drive the left 7-segment LED and led_u8 will drive the right 7-segment LED available on the XStend board. As clk input you will use the clk signal available via P88. It will be divided inside your rand entity by the clk_divider. As start input you will assign the SPARE push button. As input ctrl (vector on 8 bits) you will use the eight DIP-switches available also on the XStend board.

Synthesize, implement and download your random number generator to the board and **show** to your TA its functioning for full credit.

SUMMARY -- In this lab you designed and verified an 8-bit random number generator.
