# Swinburne University of Technology
# School of Science Technology and Engineering
# Mini Project 3

Lab commences in week 8. Report is due by Monday 15th of May at 4:00 pm in my mailbox on the 7th floor of EN building.

**Preliminary Work:**

1. Write a source VHDL to describe the circuit in Figure 1 (2 marks)
2. Write a source code to generate the 16-bit pseudo random number (PRN) generators in part 2. (3 marks)

## Part 1: Design of a RANDOM-ACCESS MEMORY EMULATOR

The following sequential circuit represents a memory with 8 addresses, where each address holds a 4-bit data. The memory positions are implemented using 4-bit registers. The reset and clock signals are shared by all the registers. Data is written or read onto/from one of the registers (selected by the signal 'address').

- Writing onto memory (wr_rd = 1): The 4-bit input data (D_in) is written into one of the 8 registers. The address signal selects which register is to be written. Here, the 7-segment display must show 0. For example: if address = "101", then D_in is written into register 5.
- Reading from memory (wr_rd = 0): The MUX output appears on the 7-segment display (hexadecimal value). The address signal selects the register from which data is read. For example: If address = "010", then data in register 2 must appear on the 7-segment display. If data in register 2 is '1010', then the symbol 'A' appears on the 7-segment display.

*Hint: please refer to lecture slides on memory elements slides 100-105 in topic 3 Behavioural Modeling.*

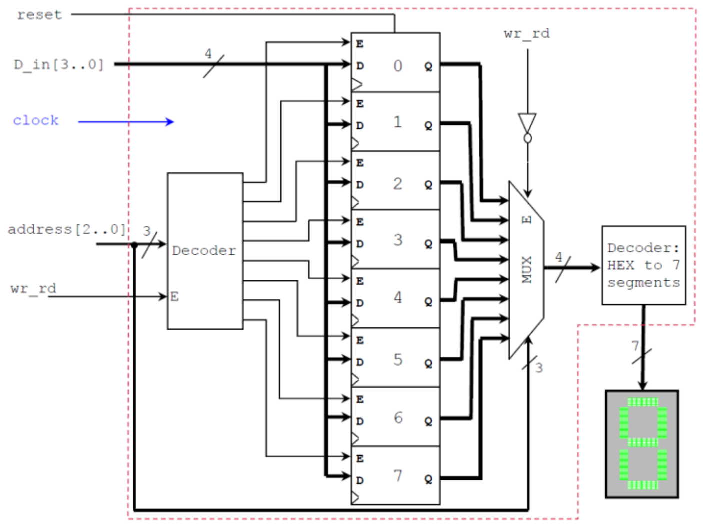


Figure 1

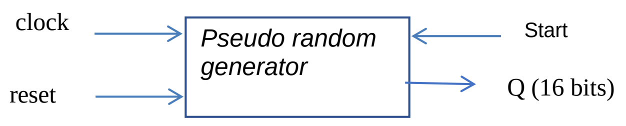
**Lab Work for Part 1**: Verify the operation of the circuit via simulation. Make sure to cover all the functional requirements.

Part 2: **Design of a Pseudo Random Number** (PRN) **Generator**

PURPOSE – In this part of the laboratory you will design and implement two Linear feedback Shift Registers (LFSR) to generate pseudo random number generators.

a. Develop a source VHDL code to design the random number generator that can generate a 16-bit number sequence. (Please refer to the polynomial below). The initial conditions are loaded on reset (initial starting count). If start is set to '1' the then the pseudo random generator generates a" random sequence" and when start is set to zero it stops and holds on to the last value in the sequence002E

**Commented [KMSJK1]:**



**P'**

P1(x) = $\mathbf{x^{16} + x^{15} + x^{11} + x^{9} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x^{1} + 1}$

Reference: Topic: Design for Testability Module: slides 100-127

**Lab Work for Part 2**

(1) Verify via simulation the operation of the pseudo random number (PRN) generator. Note the maximum clock frequency and the hardware requirements.

**Report**

**Hand in**

1. Provide the VHDL source code for part 1. Include the signal generation stimulus, simulation results, hardware requirements, and maximum clock frequency (include the FPGA chip which was used). Ensure that you provide verification of all the functional requirements.

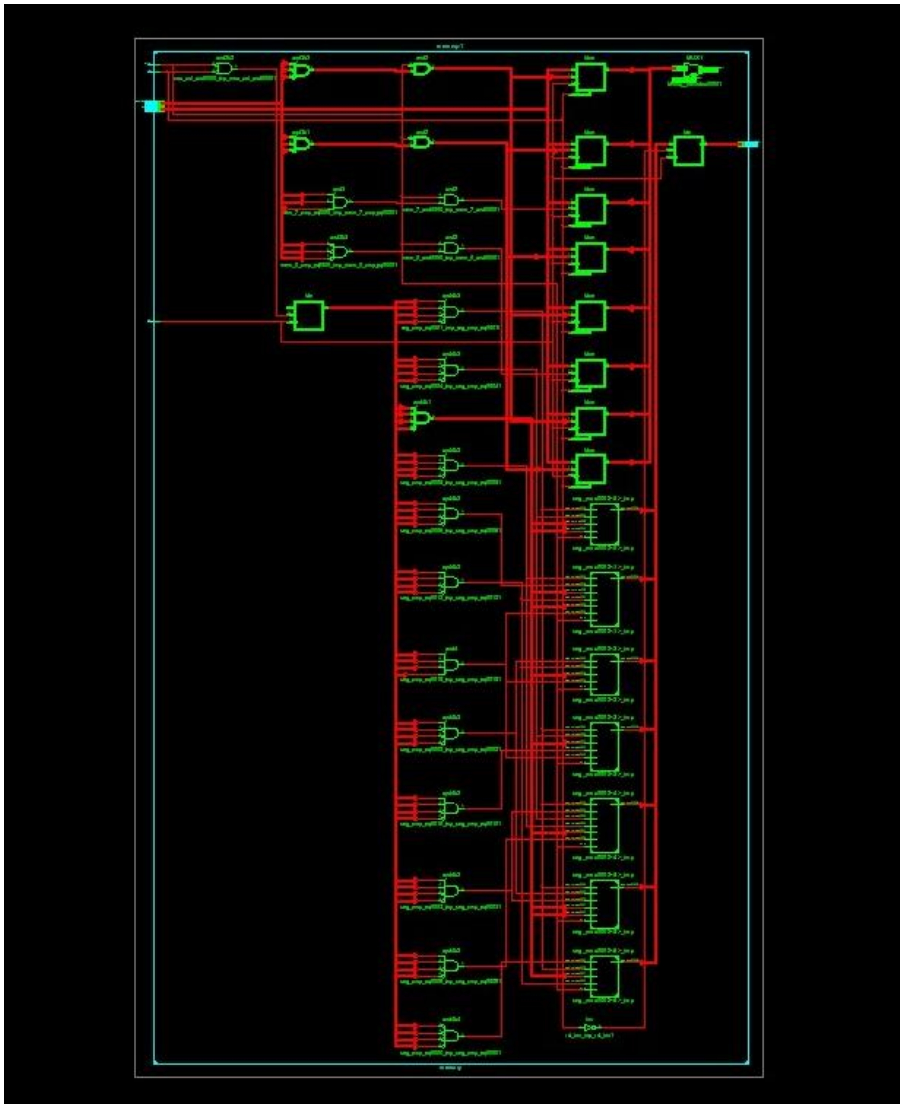
Part 1 VHDL Code

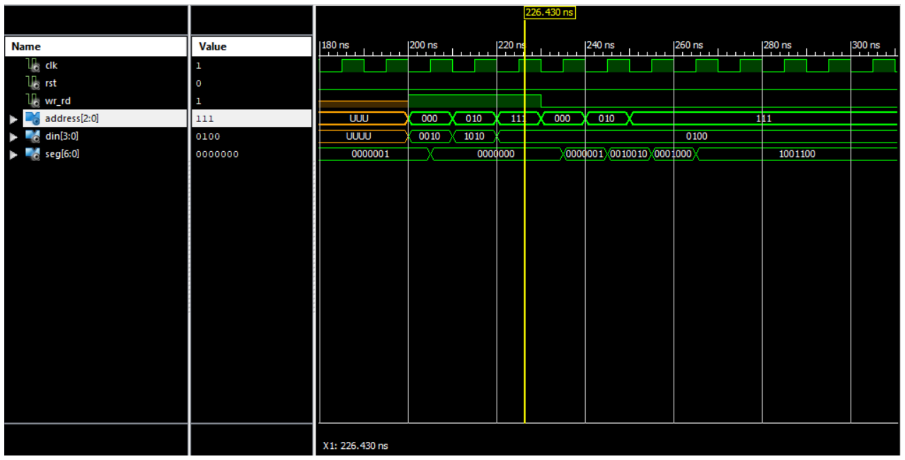
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity memory is
    port (
        clk : in std_logic;
        rst : in std_logic;
        wr_rd : in std_logic;
        address : in std_logic_vector(2 downto 0);
        din : in std_logic_vector(3 downto 0);
        seg : out std_logic_vector(6 downto 0)
    );
end entity;

architecture rtl of memory is
    type memory_array is array (0 to 7) of std_logic_vector(3 downto 0);
    signal mem : memory_array := (others => "0000");
    signal mux_out : std_logic_vector(3 downto 0);
begin
    process (clk, rst)
    begin
        if rst = '1' then
            mem <= (others => "0000");
        elsif rising_edge(clk) then
            if wr_rd = '1' then -- Write mode
                mem(to_integer(unsigned(address))) <= din;
                seg <= "0000000";
            else -- Read mode
                mux_out <= mem(to_integer(unsigned(address)));
                case mux_out is
                    when "0000" => seg <= "0000001"; -- 0
                    when "0001" => seg <= "1001111"; -- 1
                    when "0010" => seg <= "0010010"; -- 2
                    when "0011" => seg <= "0000110"; -- 3
                    when "0100" => seg <= "1001100"; -- 4
                    when "0101" => seg <= "0100100"; -- 5
                    when "0110" => seg <= "0100000"; -- 6
                    when "0111" => seg <= "0001111"; -- 7
                    when "1000" => seg <= "0000000"; -- 8
                    when "1001" => seg <= "0000100"; -- 9
                    when "1010" => seg <= "0001000"; -- A
                    when "1011" => seg <= "1100000"; -- B
                    when "1100" => seg <= "0110001"; -- C
                    when "1101" => seg <= "1000010"; -- D
                when "1110" => seg <= "0110000"; -- E
                when others => seg <= "0111000"; -- F
            end case;
        end if;
    end if;
  end process;
end architecture;
```

Part 1 Testbench

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

entity memory_tb is
end entity;

architecture test of memory_tb is
    -- Component declaration
    component memory is
        port (
            clk : in std_logic;
            rst : in std_logic;
            wr_rd : in std_logic;
            address : in std_logic_vector(2 downto 0);
            din : in std_logic_vector(3 downto 0);
            seg : out std_logic_vector(6 downto 0)
        );
    end component;

    -- Signals declaration
    signal clk : std_logic := '0';
    signal rst : std_logic := '0';
    signal wr_rd : std_logic;
    signal address : std_logic_vector(2 downto 0);
    signal din : std_logic_vector(3 downto 0);
    signal seg : std_logic_vector(6 downto 0);

begin

    -- Instantiate the memory component
    uut: memory port map (
        clk => clk,
        rst => rst,
        wr_rd => wr_rd,
        address => address,
        din => din,
        seg => seg
    );

    -- Clock generation
    process
    begin
        clk <= '0';
        wait for 5 ns;
        clk <= '1';
        wait for 5 ns;
    end process;

    -- Stimulus process
    process
    begin
        -- Perform a reset
        rst <= '1';
        wait for 100 ns;
        rst <= '0';
        wait for 100 ns;

        -- Write some values to memory
        wr_rd <= '1';
        address <= "000";
        din <= "0010";
        wait for 10 ns;

        address <= "010";
        din <= "1010";
        wait for 10 ns;

        address <= "111";
        din <= "0100";
        wait for 10 ns;

        -- Read values from memory
        wr_rd <= '0';
        address <= "000";
        wait for 10 ns;

        address <= "010";
        wait for 10 ns;

        address <= "111";
        wait for 10 ns;

        -- End the simulation
        wait;
    end process;

end architecture;
```

Part 1 Simulation Results





```
Part 1 Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 2.674ns (Maximum Frequency: 373.972MHz)
   Minimum input arrival time before clock: 3.931ns
   Maximum output required time after clock: 4.040ns
   Maximum combinational path delay: No path found
```

```
HDL Synthesis Report

Macro Statistics
# Registers                                          : 10
 4-bit register                                      : 9
 7-bit register                                      : 1
# Multiplexers                                       : 1
 4-bit 8-to-1 multiplexer                            : 1
```

Verification of working for part 1:

From the simulation we can see the programmed output of the 7seg being “0” (0000001). The later variants of din for the read and write operations, we can see the variation of the 7seg output being correct.

2. Provide the VHDL source code for part 2, Include the signal generation stimulus, simulation results, hardware requirements, and maximum clock frequency ( include the FPGA which was used). Generate simulation results for different initial conditions.

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

entity lfsr is
port (
    reset : in std_logic;
    clock : in std_logic;
    start : in std_logic;
    q_out : out std_logic_vector(15 downto 0)
);
end lfsr;

architecture behavior of lfsr is
  signal reg : std_logic_vector(15 downto 0);
begin

  process(clock, reset)
  begin
    if reset = '1' then
      reg <= (others => '1');
    elsif rising_edge(clock) then
      if start = '1' then
        -- XOR taps
        reg <= reg(14 downto 0) &
                    (reg(15) xor reg(13) xor reg(12) xor reg(4));
      end if;
    end if;
  end process;

  q_out <= reg;

end behavior;
```

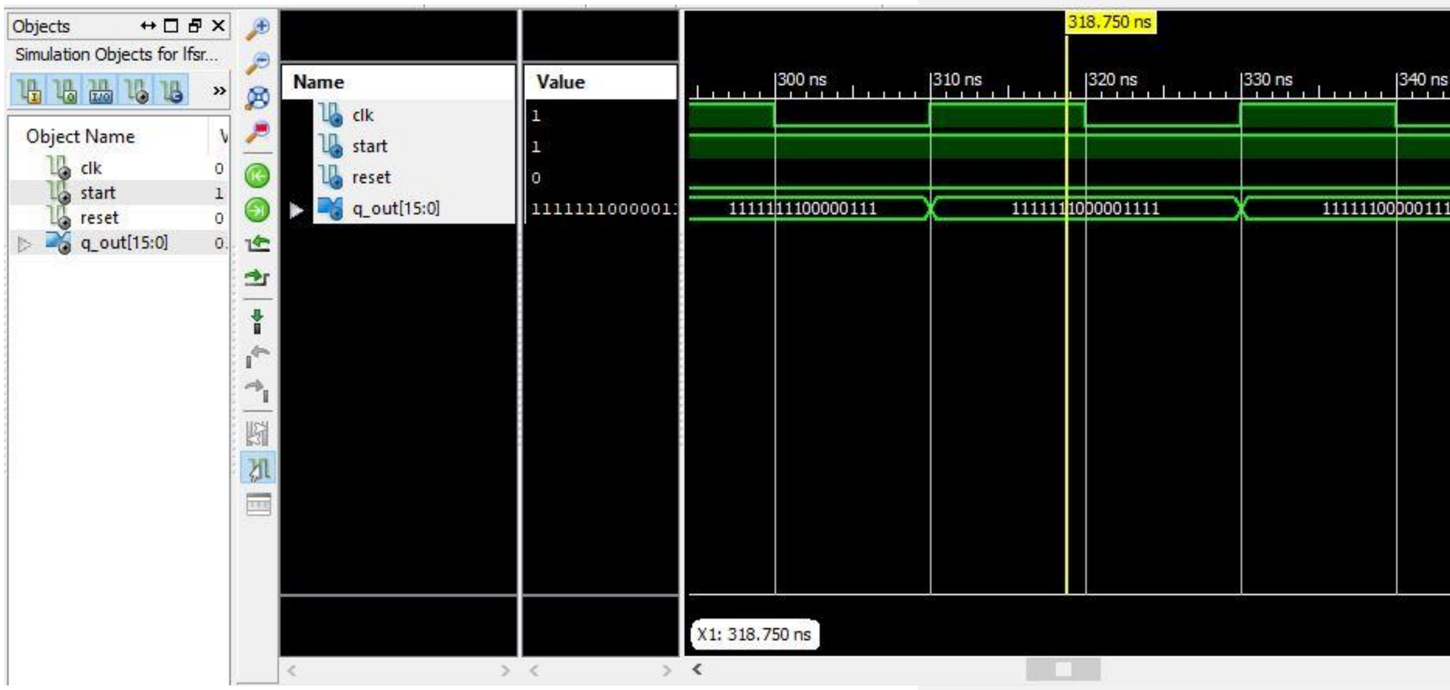
Testbench

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

entity lfsr_tb is
end lfsr_tb;
```

```
architecture testbench of lfsr_tb is
  signal clk       : std_logic := '0';
  signal start     : std_logic := '0';
  signal reset     : std_logic := '1';
  signal q_out     : std_logic_vector(15 downto 0);

begin
  DUT: entity work.lfsr
    port map(
      clock     => clk ,
      reset     => reset ,
      start     => start,
      q_out     => q_out
    );

  clk <= not clk after 10 ns;

  stimulus: process
  begin
    -- Reset the LFSR
    reset <= '1';
    wait for 10 ns;
    reset <= '0';
    wait for 10 ns;

    -- Test 1: Start = '0'
    start <= '0';
    wait for 100 ns;

    -- Test 2: Start = '1', generate 10 cycles
    start <= '1';
    wait for 10 * 10 ns;
    start <= '0';
    wait for 10 ns;

    -- Test 3: Start = '1', generate 100 cycles
    start <= '1';
    wait for 100 * 10 ns;
    start <= '0';
    wait for 10 ns;

    -- Test 4: Reset during operation
    start <= '1';
    wait for 20 * 10 ns;
    reset <= '1';
    wait for 10 ns;
    reset <= '0';
    wait for 30 * 10 ns;
    start <= '0';

    -- Wait for some additional cycles
    wait for 10 ns;
    wait;
  end process;
```

end testbench;

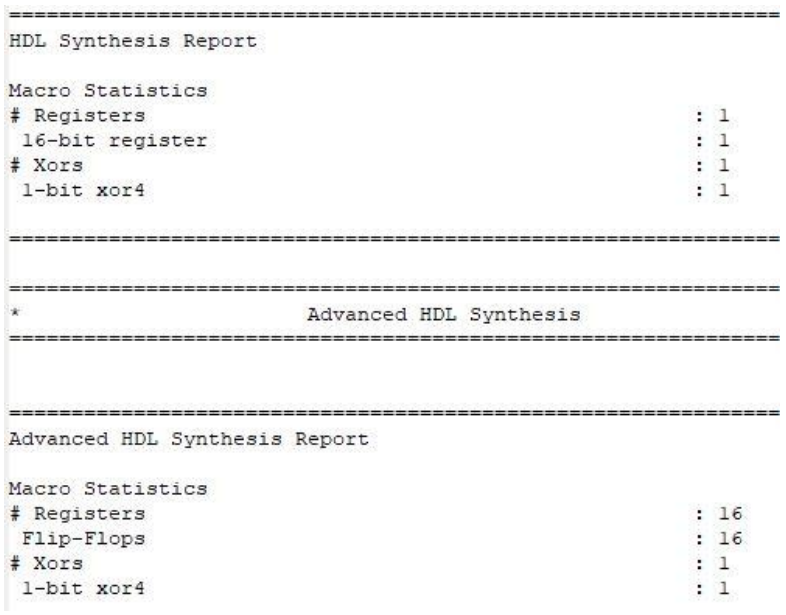
Simulation Results



Timing Summary

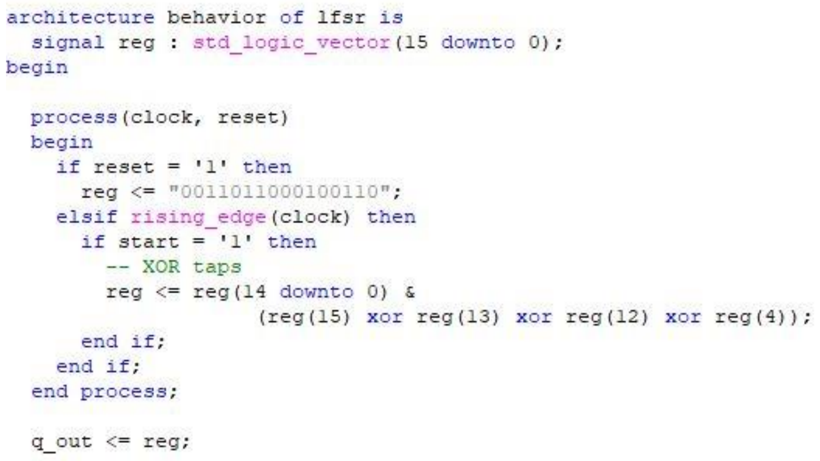
```
Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 2.321ns (Maximum Frequency: 430.765MHz)
   Minimum input arrival time before clock: 2.290ns
   Maximum output required time after clock: 6.306ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)
```

Hardware



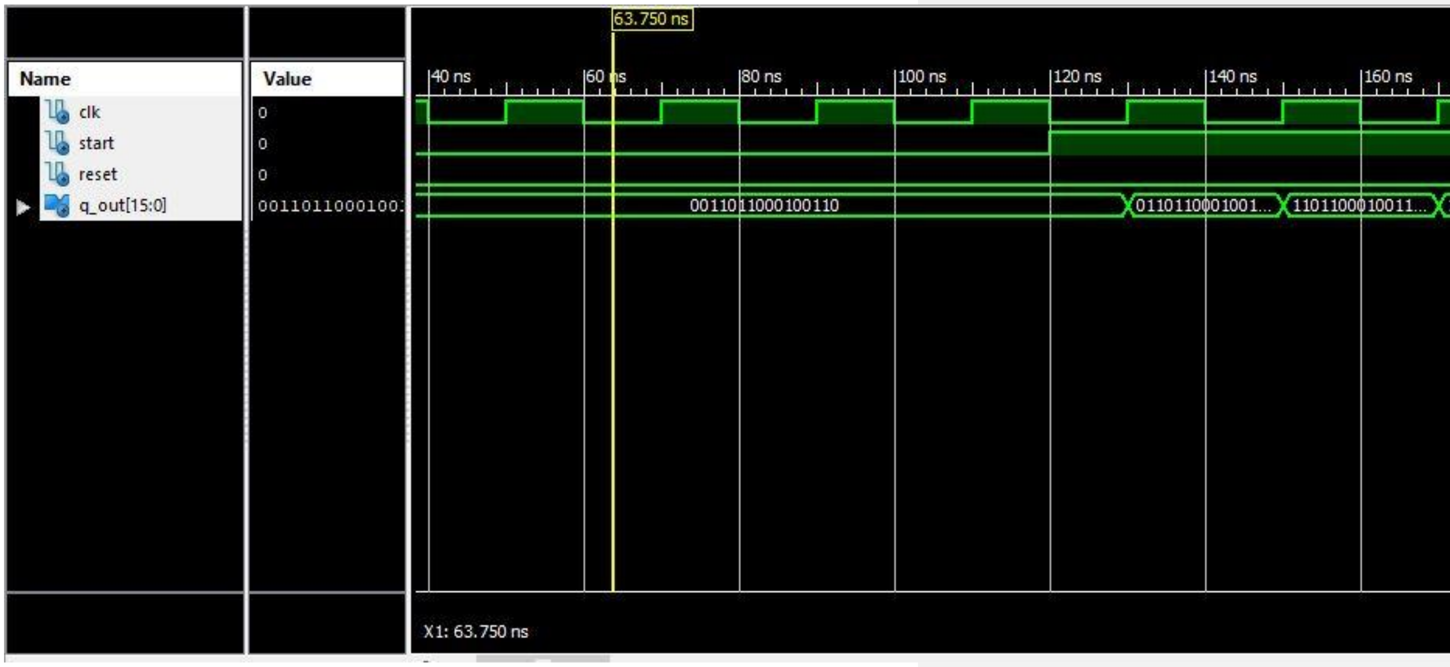
```
=============================================================
HDL Synthesis Report

Macro Statistics
# Registers                                          : 1
 16-bit register                                     : 1
# Xors                                               : 1
 1-bit xor4                                          : 1

=============================================================

=============================================================
*                       Advanced HDL Synthesis
=============================================================


=============================================================
Advanced HDL Synthesis Report

Macro Statistics
# Registers                                          : 16
 Flip-Flops                                          : 16
# Xors                                               : 1
 1-bit xor4                                          : 1
```

Initial condition

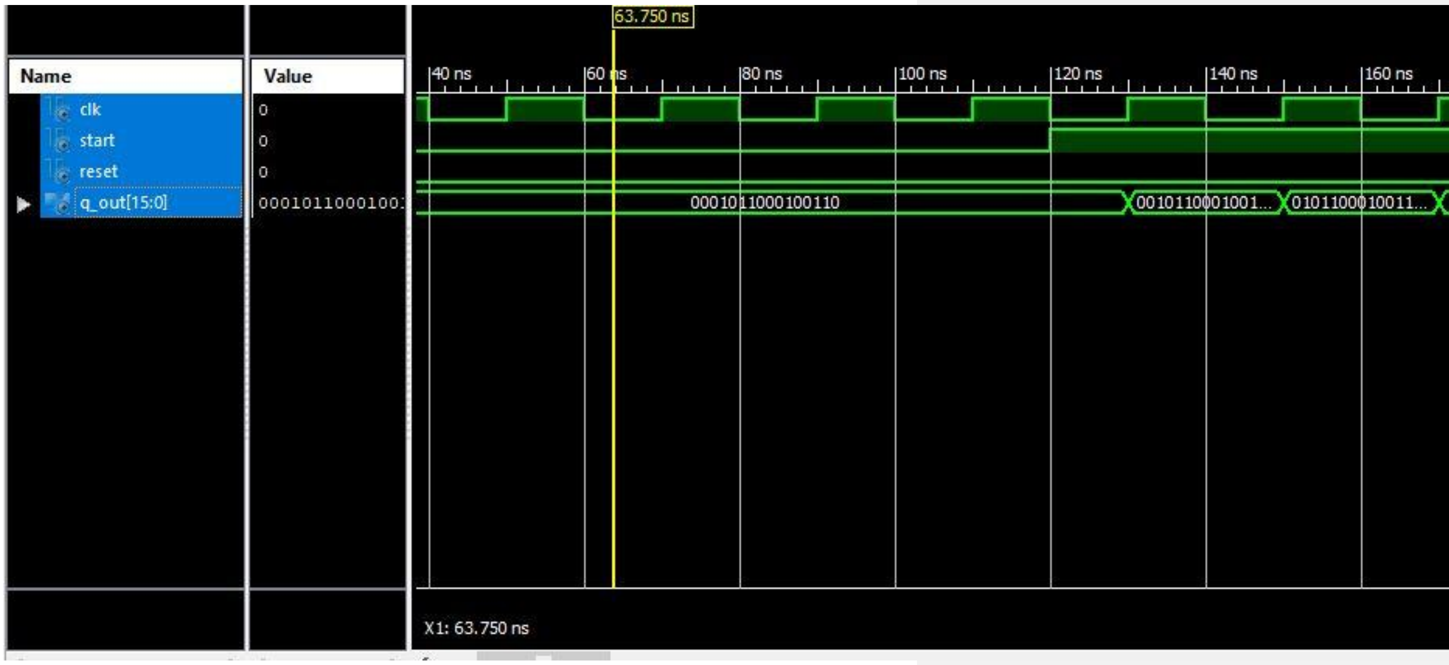


```
architecture behavior of lfsr is
  signal reg : std_logic_vector(15 downto 0);
begin

  process(clock, reset)
  begin
    if reset = '1' then
      reg <= "0011011000100110";
    elsif rising_edge(clock) then
      if start = '1' then
        -- XOR taps
        reg <= reg(14 downto 0) &
                    (reg(15) xor reg(13) xor reg(12) xor reg(4));
      end if;
    end if;
  end process;

  q_out <= reg;
```

We can see that reg is set to a random 16 digit array. When we see the simulation results below, we get:



Similiaryly, for another random initial variation, we get different generated random outputs, proving the ability of the lfsr to create semi-random outputs.



Ismat Hijazin
April 26, 2023