Lab 3 - Random Number Generator

Nate Herbst
A02307138
Nathan Walker
A02364124

Introduction

The purpose of this lab was to implement a pseudo-random number generator (PRNG) using a Linear Feedback Shift Register (LFSR) on a development board. The two buttons on the board were used to reset the system and generate new random numbers. The generated numbers were to be displayed on two of the seven-segment displays in hexadecimal format.

Approach

Our design consists of an LFSR that generates a sequence of pseudo-random numbers based on a feedback polynomial. We configured the LFSR with a 12-bit wide register and implemented the polynomial $x^{12} + x^{11} + x^{10} + x^{4} + 1$ to ensure a maximal-length sequence. The random number is displayed on two seven-segment displays. One button is used to reset the system, while the other is used to trigger the generation of new random numbers.

Challenges Encountered and Solutions

1. Adjusting LFSR Bit Width

Initially, we started with an 8-bit LFSR. However, we realized that the 8-bit range might not cover all the desired numbers, including the hexadecimal number '00'. To ensure a larger sequence and include '00', we increased the bit width of the LFSR to 12 bits. This change gave us a broader range of values while maintaining the maximal-length sequence using the $x^{12} + x^{11} + x^{10} + x^{4} + 1$ feedback polynomial.

Solution: We adjusted the LFSR bit width from 8 to 12 bits and updated the feedback polynomial to reflect the new structure.

2. Syntax Issues in Top-Level Module

When integrating the LFSR into the top-level module, we encountered several syntax issues. Specifically, there were mismatches between the signal names and ports, as well as incorrect mappings of generic parameters. Additionally, the naming convention between `en` and `enable` caused confusion in the instantiation of the LFSR within the top-level module.

Solution: We resolved these issues by ensuring that the signal names between the components were consistently mapped. We also updated the LFSR's port declarations and used proper signal assignments for enable and reset functionalities in the top-level design.

3. Button Logic Inversion

When testing the design on the development board, we noticed that the buttons were not behaving as expected. Pressing the "reset" or "generate" button produced strange results or no result at all. After further investigation, we found that the buttons on the board were logic high by default, meaning they are normally at '1' and switch to '0' when pressed.

Solution: To handle this, we modified our code so that the reset (`rst`) and enable (`en`) signals would trigger on the falling edge (when the button is pressed, logic level changes from '1' to '0') rather than the rising edge. This change resolved the issue and the buttons functioned correctly.

Conclusion

In this lab, we successfully implemented a pseudo-random number generator using a 12-bit LFSR. By resolving issues related to LFSR bit width, syntax errors, and button logic, we were able to complete the design and meet the project requirements. The random numbers were correctly displayed in hexadecimal format, and the buttons operated as intended. This project provided valuable experience in handling VHDL design challenges and working with LFSR-based random number generation.

Note: Physically, since we're not using a debouncer, there is no way to observe this phenomenon directly because of the speed of the clock (though theoretically possible, the chance is very slim). This phenomenon happens because we're using a 12-bit LFSR, which is larger than our 8-bit random number output. As a result, when the LFSR generates a sequence that includes "00" in the lower 8 bits, that state persists across two clock cycles before the LFSR moves to the next state. The LFSR continues shifting through its full 12-bit state, which leads to this momentary repetition of "00" in the output.

Appendix

Simulation Screenshots:

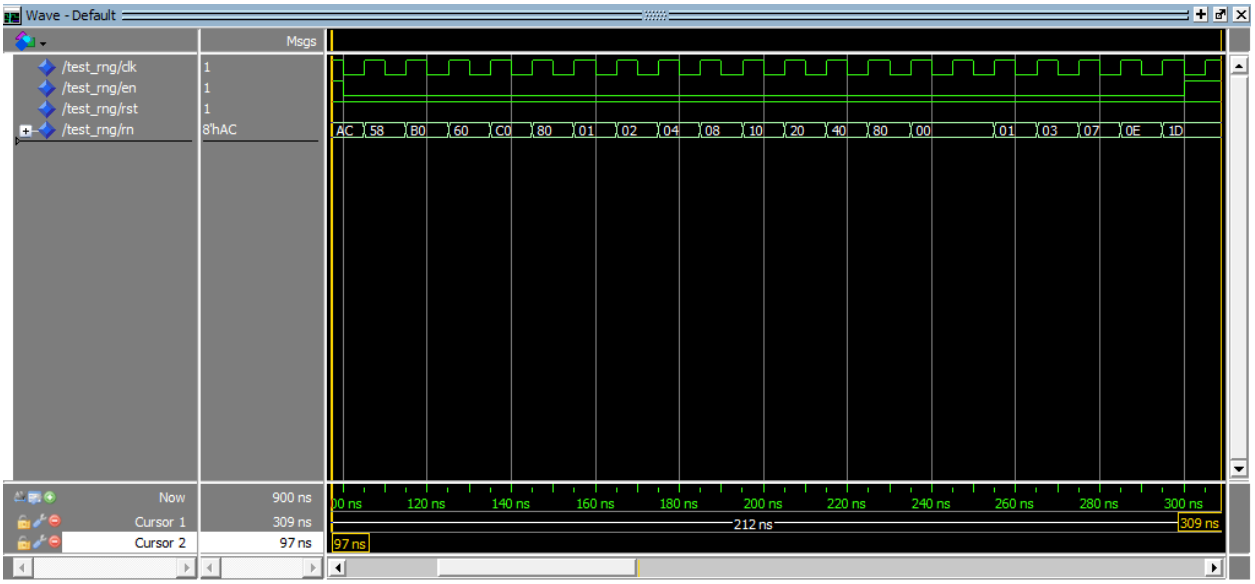

Figure 1: The Zero Case

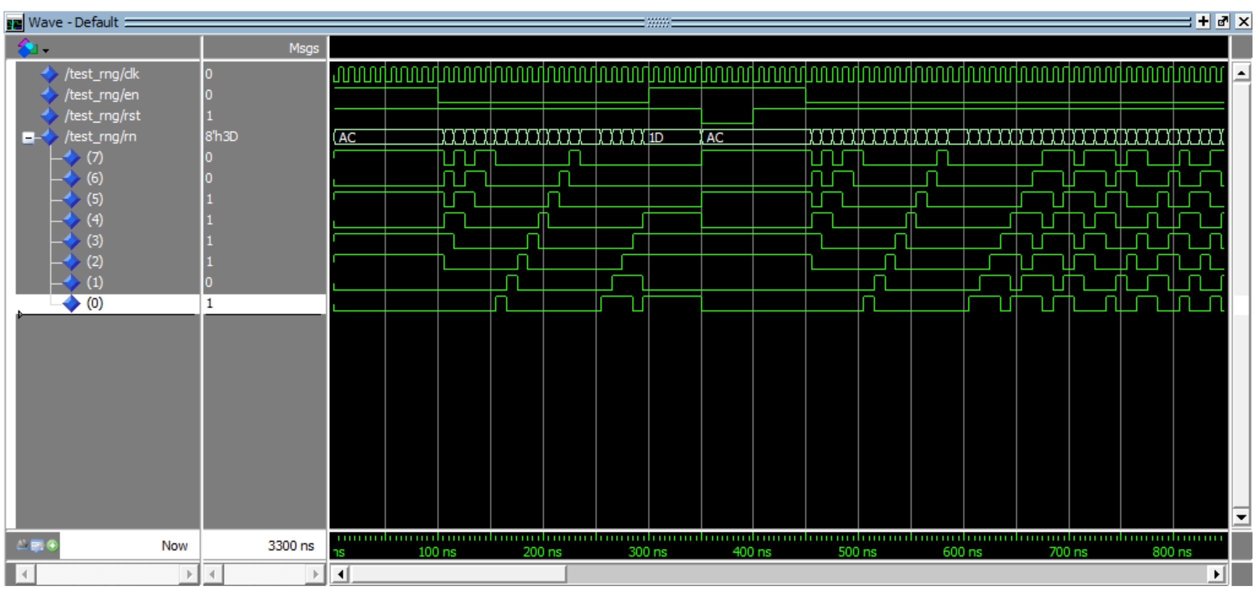

Figure 2: The repeated pattern

Top-Level Module (lab3_RNG.vhdl)

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

entity lab3_RNG is
   generic (
       N : integer := 8
   );
   port (
       HEX0 : out std_logic_vector(N - 1 downto 0);
       HEX1 : out std_logic_vector(N - 1 downto 0);
       HEX2 : out std_logic_vector(N - 1 downto 0);
       HEX3 : out std_logic_vector(N - 1 downto 0);
       HEX4 : out std_logic_vector(N - 1 downto 0);
       HEX5 : out std_logic_vector(N - 1 downto 0);
       KEY : in std_logic_vector(1 downto 0);
       ADC_clk_10 : in std_logic;
       MAX10_CLK1_50 : in std_logic;
       MAX10_CLK2_50 : in std_logic
   );
end lab3_RNG;

architecture componentlist of lab3_RNG is
   component RNG
       generic (
           N : integer := 8;
           M : integer := 8
       );
       port(
           clk : in std_logic;
           en : in std_logic;
           rst : in std_logic;
           rn : out std_logic_vector ( N - 1 downto 0)
       );
   end component RNG;

   component HEX_seven_seg_disp
       port (
```

```vhdl
            hex : in std_logic_vector(3 downto 0);
            clk : in std_logic;
            oseg : out std_logic_vector(7 downto 0)
        );
    end component HEX_seven_seg_disp;

    signal rn : std_logic_vector (N - 1 downto 0);


begin

    RNG1 : RNG
    -- Instantiate RNG (pseudo-random number generator)
        generic map (
            N => 8,
            M => 12
        )
        port map (
            clk => ADC_CLK_10,  -- Use the ADC clock
            en => KEY(0),           -- Generate button (enable)
            rst => KEY(1),      -- Reset button
            rn => rn            -- Random number
        );

    -- Display the lower nibble (4 bits) of the random number on HEX0
    disp1 : HEX_seven_seg_disp
        port map (
            hex => rn(3 downto 0),
            clk => ADC_CLK_10,
            oseg => HEX0
        );

    -- Display the upper nibble (4 bits) of the random number on HEX1
    disp2 : HEX_seven_seg_disp
        port map (
            hex => rn(7 downto 4),
            clk => ADC_CLK_10,
            oseg => HEX1
        );
```

```
   -- Turn off unused displays (set to all ones to disable all segments)
   HEX2 <= (others => '1');
   HEX3 <= (others => '1');
   HEX4 <= (others => '1');
   HEX5 <= (others => '1');
end componentlist;
```

RNG Module (RNG.vhdl)

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

entity RNG is
   generic (
       N : integer := 8; -- Output random number size
       M : integer := 12  -- LFSR bit width
   );
   port (
       clk : in std_logic;
       en  : in std_logic;
       rst : in std_logic;
       rn  : out std_logic_vector(N-1 downto 0) -- Random number output
   );
end RNG;

architecture behavioral of RNG is
   component LFSR
       generic (
           M       : integer :=  12 -- LFSR bit width
        );
       port (
           input       : in std_logic;
           clk         : in std_logic;
           enable      : in std_logic;
           rst         : in std_logic;
           lfsr_out    : out std_logic_vector(M-1 downto 0)
       );
```

```vhdl
    end component LFSR;

    signal lfsr_out : std_logic_vector(M-1 downto 0);
    signal bit  : std_logic;

begin
    -- Instantiate the LFSR
    LFSR1: LFSR
        generic map (
            M => 12
        )
        port map (
            clk         => clk,
            input       => bit,
            enable      => en,   -- Map "en" from RNG to "enable" in LFSR
            rst         => rst,
            lfsr_out    => lfsr_out  -- Map "lfsr_out" instead of "data"
        );

    -- Feedback polynomial 8 bits: x^16 + x^14 + x^13 + x^11 + 1
    -- bit <= lfsr_out(0) xor lfsr_out(2) xor lfsr_out(3) xor lfsr_out(5);
    -- Feedback polynomial 12 bits: x^12 + x^11 + x^10 + x^4 + 1
    bit <= lfsr_out(3) xor lfsr_out(9) xor lfsr_out(10) xor lfsr_out(11);


    -- Connect the LFSR output to the random number output
    rn <= lfsr_out(N-1 downto 0);

end architecture behavioral;
```

LFSR Module (LFSR.vhdl)

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

```vhdl
entity LFSR is
   generic (
       M : integer := 12  -- LFSR bit width
   );
   port (
       clk          : in  std_logic;
       input        : in std_logic;
       rst          : in  std_logic;
       enable       : in  std_logic;
       lfsr_out     : out std_logic_vector(M-1 downto 0)
   );
end entity LFSR;

architecture behavioral of LFSR is
   signal lfsr : std_logic_vector(M-1 downto 0) := x"9AC";
begin
   process (clk, rst)
   begin
       if rst = '0' then
           lfsr <= x"9AC";  -- Initial non-zero start state
       elsif rising_edge(clk) then
           if enable = '0' then
               lfsr(M-1 downto 1) <= lfsr(M-2 downto 0);  -- Shift the
register
               lfsr(0) <= input;
           end if;
       end if;
   end process;

   lfsr_out <= lfsr;
end architecture behavioral;
```