10/01/2024

# Lab 03 Report

Ryan Beck
Jared Bronson

## Objectives

Implement a pseudo-random number generator on the DE-10 Lite development board. The random number generator should be implemented with an appropriately sized LFSR. Taps in LSFR should be chosen to maximize the number sequence. The two on-board buttons will be used to generate and reset. When generate is pressed, a two-digit hexadecimal number, from 00-FF, will appear on two of the seven-segment displays. Pressing reset will revert the sequence back to the seed value.

## Procedures

Using the system builder, create a Quartus Project File to include the on-board clock, seven-segment displays, and push buttons. Open the Verilog file created by the system builder and create a VHDL file with the same name. In the VHDL file, create a random number generator, rng, entity with ports for the on-board clock, seven-segment displays, and push buttons using the same names given in the Verilog file. Only use the 10 MHz clock for this design. In the generic, create a 16 bit logic vector for a seed value, and create a 16 bit hexadecimal value to populate the vector. For this design, the hexadecimal value A58B was used.

Next, create architecture for the design. Initialize two 16-bit vector signals to perform operations on the LFSR. Also, initialize any other needed signals for the design. Create a lookup table with the needed values to display all 16 hexadecimal values on the seven-segment display. Begin a process sensitive to the clock and push buttons. In this process create if statements to for KEY(0) press, KEY(1) press, and a start state before either button is pushed.

Make an else statement for an idle state to continuously update the LFSR to assist randomness. In the KEY(0) press statement, reset the LFSR to the seed value and display it on the display. In the KEY(1) press statement, display the LFSR that is continuously being updated. In the start state, display the original seed value. Create another process sensitive to clock, and turn off the unused seven-segment displays.

Next, create a testbench with the same generic and ports as the rng. Define signals for the clock, buttons, and seven-segment displays as well as a constant for clock period. Start a process to vary the clock between one and zero every half clock period. Start another process to test the behavior of the design for the button pushes. Simulate the design using the testbench. If the simulated behavior matches what is expected, compile and implement the design on the development board.

## Results

The rng file, shown in Figures, successfully implements a random number generator. The design successfully resets to an original seed value by pushing reset, and generates random two digit hexadecimal numbers. The function of the file was verified with a testbench file used to run simulations. Both the testbench and the simulations can be found in Figures.

## Figures

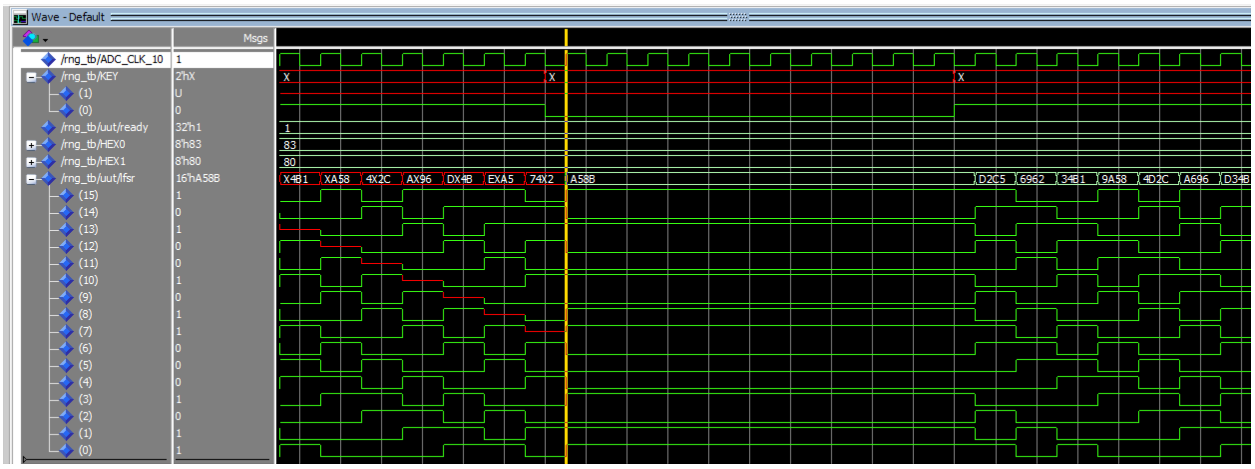


*Figure 1: Initial Reset Behavior*

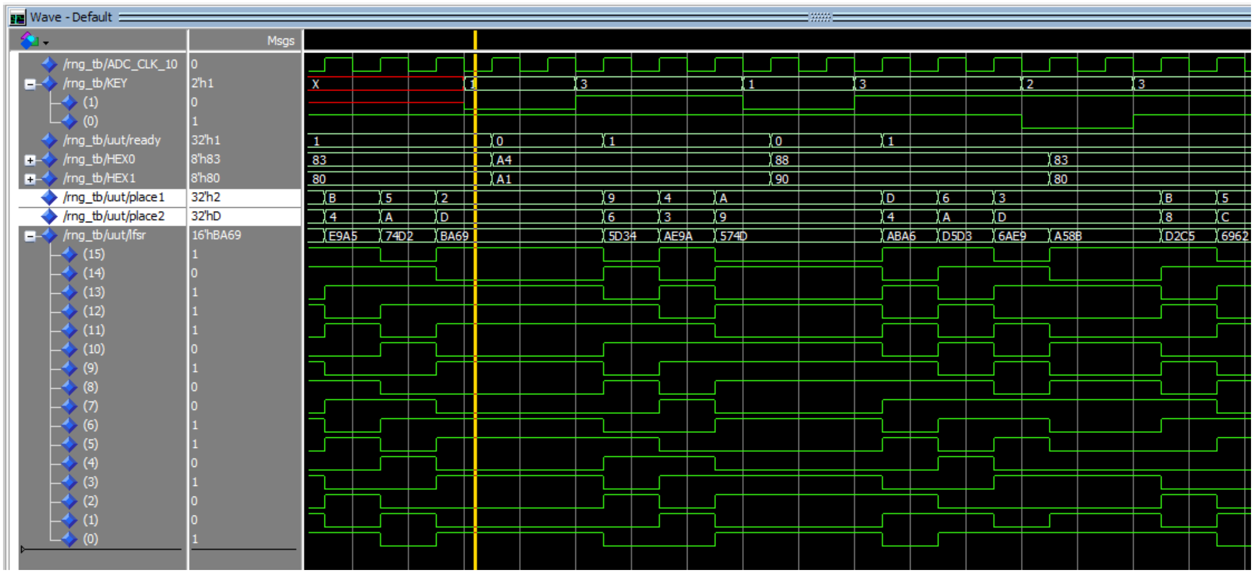
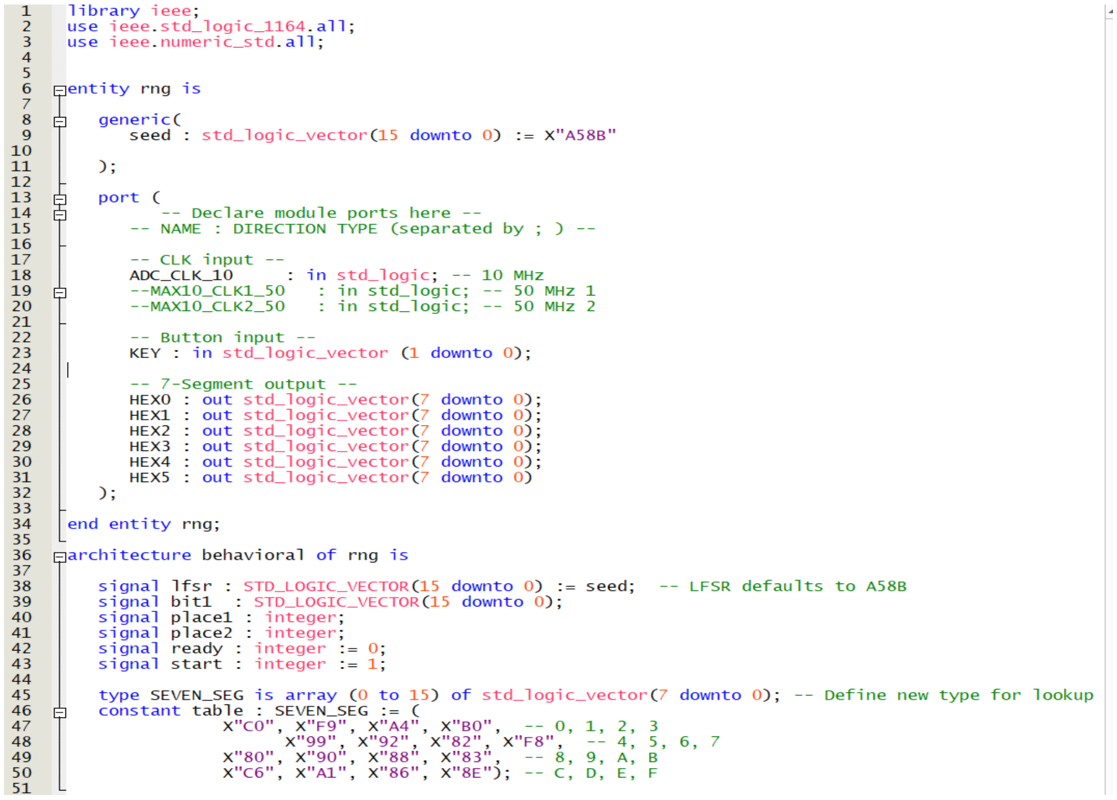
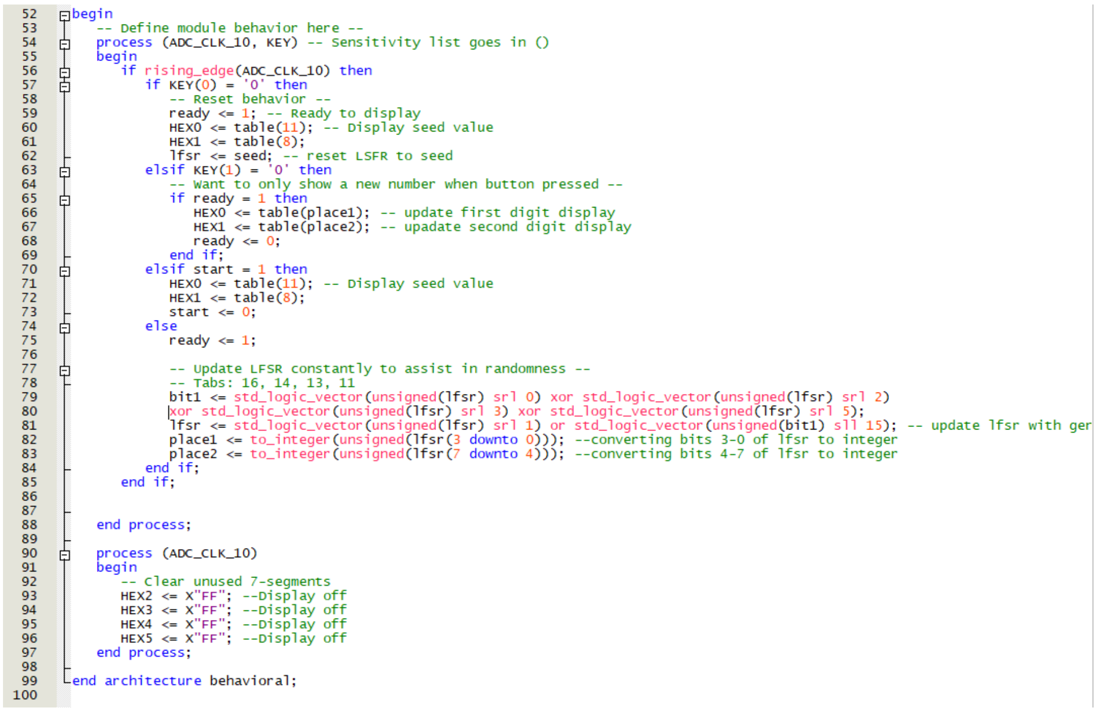


*Figure 2: Two number generations, followed by reset to return display to seed value*



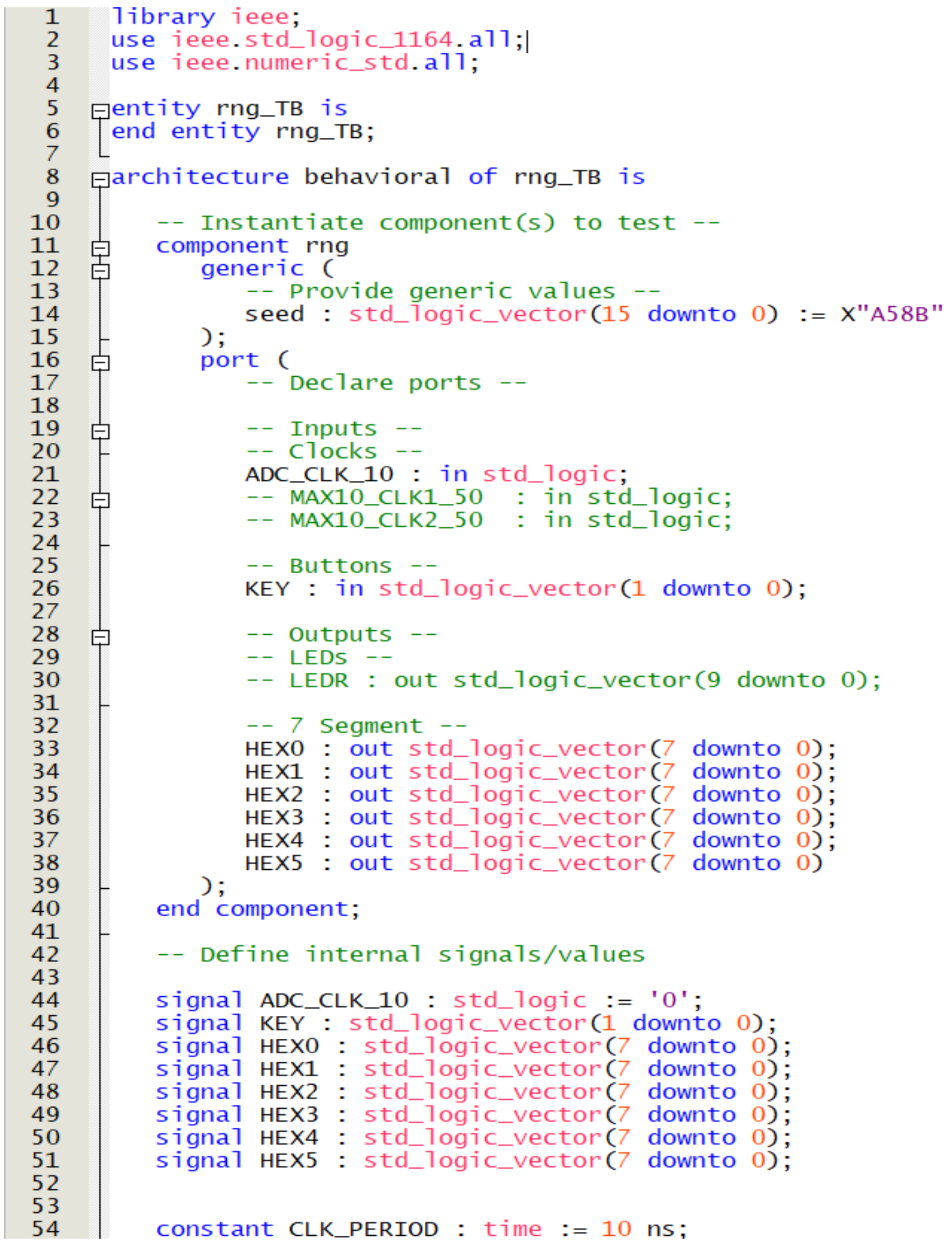
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity rng is

   generic(
      seed : std_logic_vector(15 downto 0) := X"A58B"

   );

   port (
         -- Declare module ports here --
      -- NAME : DIRECTION TYPE (separated by ; ) --

      -- CLK input --
      ADC_CLK_10      : in std_logic; -- 10 MHz
      --MAX10_CLK1_50    : in std_logic; -- 50 MHz 1
      --MAX10_CLK2_50    : in std_logic; -- 50 MHz 2

      -- Button input --
      KEY : in std_logic_vector (1 downto 0);

      -- 7-Segment output --
      HEX0 : out std_logic_vector(7 downto 0);
      HEX1 : out std_logic_vector(7 downto 0);
      HEX2 : out std_logic_vector(7 downto 0);
      HEX3 : out std_logic_vector(7 downto 0);
      HEX4 : out std_logic_vector(7 downto 0);
      HEX5 : out std_logic_vector(7 downto 0)
   );

end entity rng;

architecture behavioral of rng is

   signal lfsr : STD_LOGIC_VECTOR(15 downto 0) := seed;  -- LFSR defaults to A58B
   signal bit1  : STD_LOGIC_VECTOR(15 downto 0);
   signal place1 : integer;
   signal place2 : integer;
   signal ready : integer := 0;
   signal start : integer := 1;

   type SEVEN_SEG is array (0 to 15) of std_logic_vector(7 downto 0); -- Define new type for lookup
   constant table : SEVEN_SEG := (
               X"C0", X"F9", X"A4", X"B0",  -- 0, 1, 2, 3
                   X"99", X"92", X"82", X"F8",  -- 4, 5, 6, 7
               X"80", X"90", X"88", X"83",  -- 8, 9, A, B
               X"C6", X"A1", X"86", X"8E"); -- C, D, E, F

```

Figure 3: RNG.vhd Pt. 1



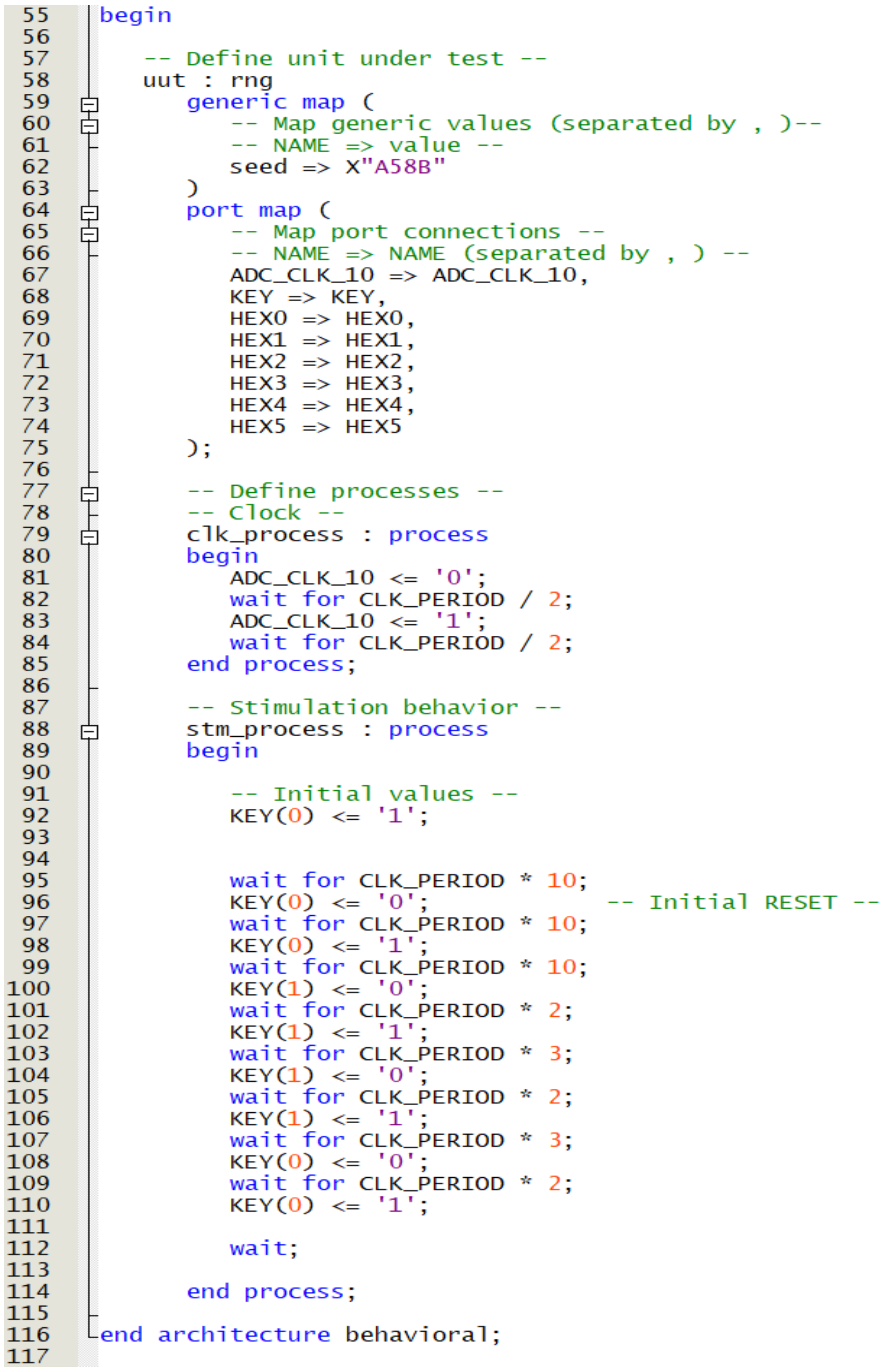
```vhdl
begin
   -- Define module behavior here --
   process (ADC_CLK_10, KEY) -- Sensitivity list goes in ()
   begin
      if rising_edge(ADC_CLK_10) then
         if KEY(0) = '0' then
            -- Reset behavior --
            ready <= 1; -- Ready to display
            HEX0 <= table(11); -- Display seed value
            HEX1 <= table(8);
            lfsr <= seed; -- reset LSFR to seed
         elsif KEY(1) = '0' then
            -- Want to only show a new number when button pressed --
            if ready = 1 then
               HEX0 <= table(place1); -- update first digit display
               HEX1 <= table(place2); -- upadate second digit display
               ready <= 0;
            end if;
         elsif start = 1 then
            HEX0 <= table(11); -- Display seed value
            HEX1 <= table(8);
            start <= 0;
         else
            ready <= 1;

            -- Update LFSR constantly to assist in randomness --
            -- Tabs: 16, 14, 13, 11
            bit1 <= std_logic_vector(unsigned(lfsr) srl 0) xor std_logic_vector(unsigned(lfsr) srl 2)
            xor std_logic_vector(unsigned(lfsr) srl 3) xor std_logic_vector(unsigned(lfsr) srl 5);
            lfsr <= std_logic_vector(unsigned(lfsr) srl 1) or std_logic_vector(unsigned(bit1) sll 15); -- update lfsr with ger
            place1 <= to_integer(unsigned(lfsr(3 downto 0))); --converting bits 3-0 of lfsr to integer
            place2 <= to_integer(unsigned(lfsr(7 downto 4))); --converting bits 4-7 of lfsr to integer
         end if;
      end if;


   end process;

   process (ADC_CLK_10)
   begin
      -- Clear unused 7-segments
      HEX2 <= X"FF"; --Display off
      HEX3 <= X"FF"; --Display off
      HEX4 <= X"FF"; --Display off
      HEX5 <= X"FF"; --Display off
   end process;

end architecture behavioral;

```

Figure 4: RNG.vhd Pt. 2



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

entity rng_TB is
end entity rng_TB;

architecture behavioral of rng_TB is

    -- Instantiate component(s) to test --
    component rng
        generic (
            -- Provide generic values --
            seed : std_logic_vector(15 downto 0) := X"A58B"
        );
        port (
            -- Declare ports --

            -- Inputs --
            -- Clocks --
            ADC_CLK_10 : in std_logic;
            -- MAX10_CLK1_50  : in std_logic;
            -- MAX10_CLK2_50  : in std_logic;

            -- Buttons --
            KEY : in std_logic_vector(1 downto 0);

            -- Outputs --
            -- LEDs --
            -- LEDR : out std_logic_vector(9 downto 0);

            -- 7 Segment --
            HEX0 : out std_logic_vector(7 downto 0);
            HEX1 : out std_logic_vector(7 downto 0);
            HEX2 : out std_logic_vector(7 downto 0);
            HEX3 : out std_logic_vector(7 downto 0);
            HEX4 : out std_logic_vector(7 downto 0);
            HEX5 : out std_logic_vector(7 downto 0)
        );
    end component;

    -- Define internal signals/values

    signal ADC_CLK_10 : std_logic := '0';
    signal KEY : std_logic_vector(1 downto 0);
    signal HEX0 : std_logic_vector(7 downto 0);
    signal HEX1 : std_logic_vector(7 downto 0);
    signal HEX2 : std_logic_vector(7 downto 0);
    signal HEX3 : std_logic_vector(7 downto 0);
    signal HEX4 : std_logic_vector(7 downto 0);
    signal HEX5 : std_logic_vector(7 downto 0);


    constant CLK_PERIOD : time := 10 ns;
```

*Figure 5: RNG_TB.vhd Pt. 1*



```vhdl
begin

    -- Define unit under test --
    uut : rng
        generic map (
            -- Map generic values (separated by , )--
            -- NAME => value --
            seed => X"A58B"
        )
        port map (
            -- Map port connections --
            -- NAME => NAME (separated by , ) --
            ADC_CLK_10 => ADC_CLK_10,
            KEY => KEY,
            HEX0 => HEX0,
            HEX1 => HEX1,
            HEX2 => HEX2,
            HEX3 => HEX3,
            HEX4 => HEX4,
            HEX5 => HEX5
        );

        -- Define processes --
        -- Clock --
        clk_process : process
        begin
            ADC_CLK_10 <= '0';
            wait for CLK_PERIOD / 2;
            ADC_CLK_10 <= '1';
            wait for CLK_PERIOD / 2;
        end process;

        -- Stimulation behavior --
        stm_process : process
        begin

            -- Initial values --
            KEY(0) <= '1';


            wait for CLK_PERIOD * 10;
            KEY(0) <= '0';                -- Initial RESET --
            wait for CLK_PERIOD * 10;
            KEY(0) <= '1';
            wait for CLK_PERIOD * 10;
            KEY(1) <= '0';
            wait for CLK_PERIOD * 2;
            KEY(1) <= '1';
            wait for CLK_PERIOD * 3;
            KEY(1) <= '0';
            wait for CLK_PERIOD * 2;
            KEY(1) <= '1';
            wait for CLK_PERIOD * 3;
            KEY(0) <= '0';
            wait for CLK_PERIOD * 2;
            KEY(0) <= '1';

            wait;

        end process;

end architecture behavioral;

```

*Figure 6: RNG_TB.vhd file Pt. 2*

## Conclusion

In conclusion, we were able to implement a pseudo-random number generator on the development board. One push button resets the LFSR to the seed value, and the other button displays a random hexadecimal number, 00-FF. We did this by constantly updating the LFSR each clock cycle to improve randomness and displaying it only when generate is pushed. We configured the seven-segment display using a lookup table. Values in the lookup table were used to turn off segments corresponding the appropriate hexadecimal value. Simulations were performed using a testbench to show proper function of the design before final implementation and testing on the development board.