INTEGRATED CIRCUIT DESIGN

State Machine Implementing a Pseudo-Random Number Generator (RNG)

Lecturer: Dr. Christopher Phillips

Student name: Frank Erasmo Cruz Felix

Student ID: 150684871

March 25th, 2019

Contents

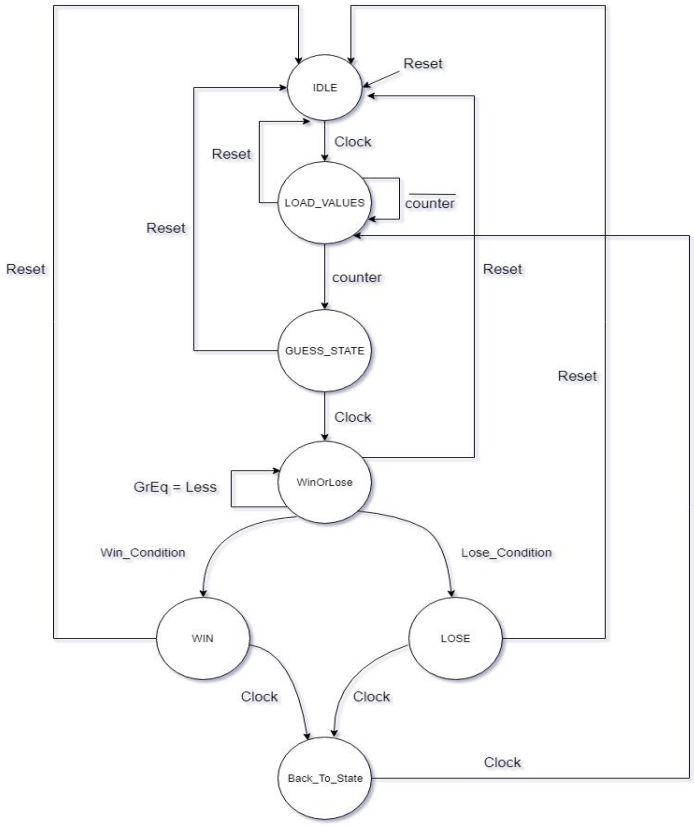
PART A: Compiling and Simulating the VHDL Design

Introduction

The VHDL code written for this Guessing Game, there has been used five different processes: State_clocked, State_counter, State_lfsr (Linear Feed Back Shift Register), LFSR_SR_input (Shift Register) and state_comb (State-transition). The state machine has 12 inputs, where 4 of them are buttons (Clock-1 bit, Reset-1 bit, GrEq-1bit and Less-1bit) and the other ones are the seven bits from the Slider; and 8 outputs (NumberLED – 4 bits, GuessLED – 1 bit, WinLED-1 bit, LoseLED- 1 bit).
Furthermore, It has been implemented additional signals to catch values used later on in the code such as: lfsr_input (8 bits), lfsr_output(1 bit), counter (4 bits), reset_counter (1 bit), seeds_in (1 bit), srone_output (4 bits) and srtwo_output(4 bits).

State_clocked process, when Reset is high the Idle state is assigned to the present state and in the following clock cycles, it will update the value of the Present_State signal. For the State_counter process, if the value of the reset_counter signal is low, then the counter signal will be set to "0000", then after the following clock cycles the counter value will be increased by one. During the State_lfsr process, when Reset is high, the values of lfsr_input, lfsr_output signals are initialised to '0' and the seeds_in signal is '1'. During the next clock cycles, the values of Slider will be assigned to the lfsr_input and seeds_in will be set to '0'. After this the values of the lfsr_input will be shifted, having the third and fourth bits xored wit most significant bit and then these new values are stored in the fourth-bit and fifth-bit respectively. In the LFSR_SF_input process, the values for the two differente sifht registers are initialised to "0000", and for the following clock cycles and if the reset_counter is high, it will process to store and shift to the left one position in either of the corresponding shift registers.

State-transition Diagram

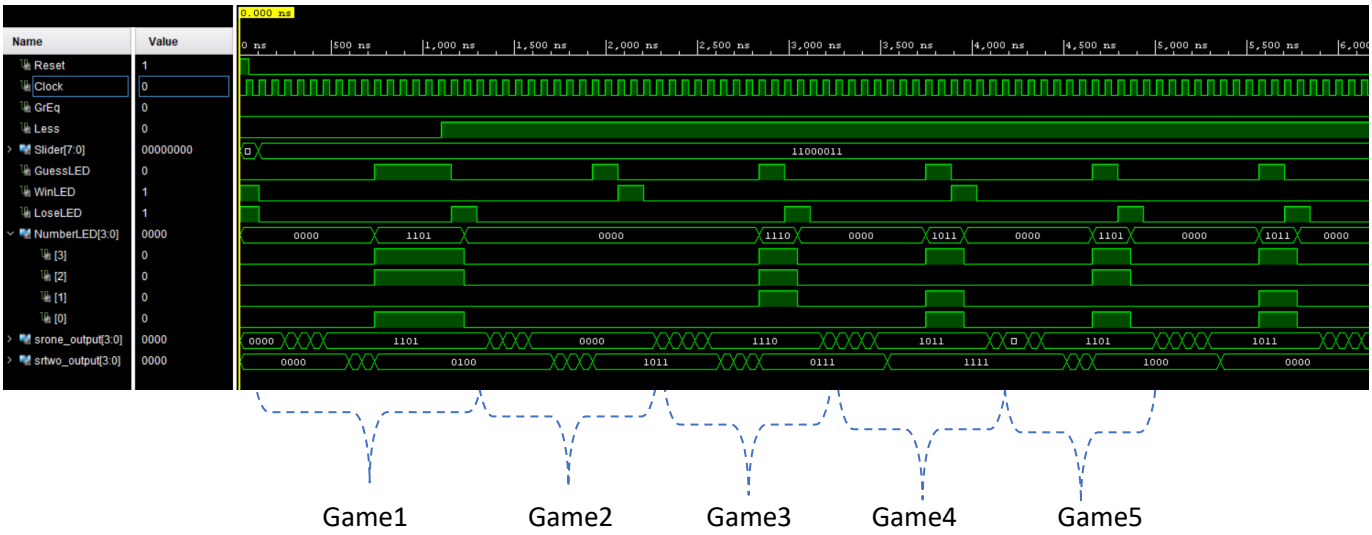


Condition	Value
Reset	'1'
Clock	Clock 'event and Clock = '1'
Win_Condition	(GrEq and SR1>=SR2) or (Less= and SR1<SR2)
Lose_Condition	(GrEq and SR1<SR2) or (Less and SR1>= SR2)
Counter	"1000"
(not) Counter	Other values different than "1000"

The diagram above shows the seven different states used in this software (Idle, Load_values, Guess_state, WinOrLose, Win, Lose, Back_to_State). In the IDLE state, the WinLED and LoseLED are set to high, whereas the other values are set to low. During the Load_values state, the WinLED and LoseLED are still kept low, while the reset_counter signal is set to high. If the value of the counter is different than "1000", then stays in the same state. Otherwise, go to the next state. When the state machine is in Guess_State state, the WinLED, LoseLED and reset_counter variables are set to low ('0'), whereas the GuessLED is set to hight. Furthermore, the value of the first register is loaded in to the NumberLED signal. In the WinOrLose state, the WinLED and LoseLED still keep previous value. To win this guessing game (Win_condition) there are 2 conditions that either has to be satisfied: GrEq is pressed and the value of the first register is greater or equal than the second shift register or the Less signal is pressed and the value of the first register is less than the second register. On the other hand, if the user presses the GrEq button and the value of the first register is less than the second register or the Less button is pressed, and the value of the first register is greater or equal than the second register, then the user will lose.

Screen Capture of the compiled source code – Timing Diagram

The code written was successfully compiled for simulation and the timing diagram below shows the corresponding results after each Game played. As I can be seen, the value of the two compared shift registers are shown to indicate if this code is working correctly.

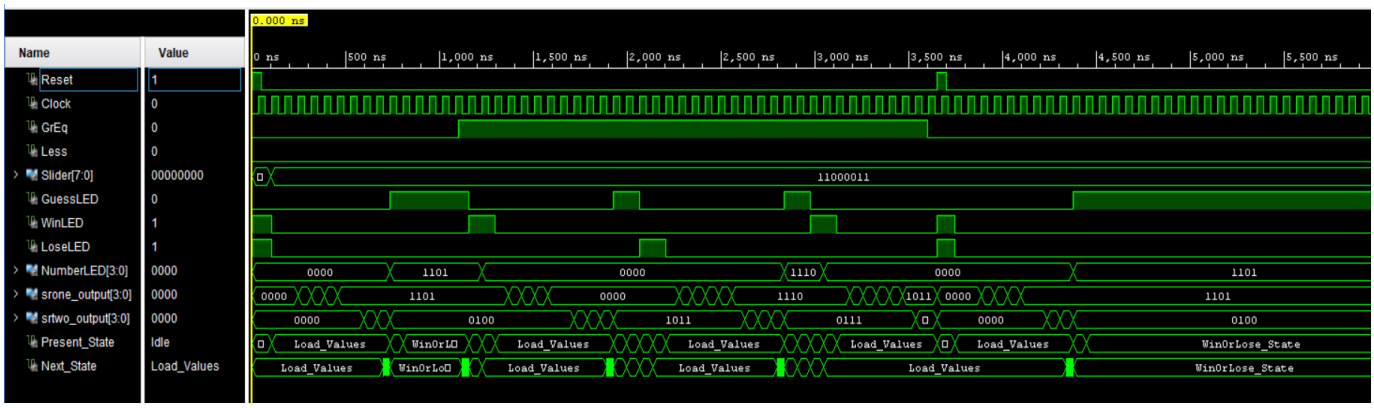


Results (Outputs) for valid and Invalid combinations, and sequences of inputs, satisfactory transition between states caused by (in)correct inputs.

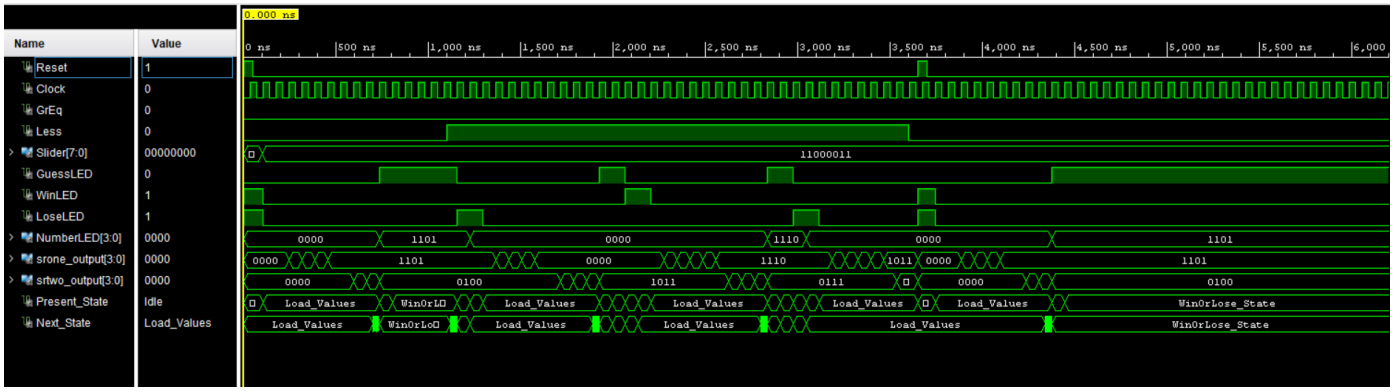
Valid Combinations

The behibour in response to Reset happens at the start of the timing diagram and at 3650 ns, when the reset is high, it resets the program and all signals are back to their initial value (All of them are set to '0' or "0000" depending on the signal). Moreover, it shows the right ouput values, either win (WinLED) or lose(LoseLED) leds switches on depending on the results obtained. Thus, the state-transition generated passing from IDLE→ Load_Values → Guess_state→WinOrLose_state and either one the other states such asWin or Lose → Backto_State→ Load_values and so on. Lastly, the

incorrect outputs are handled such as when no buttons are pressed, so it iterates in the same state until an event happens.



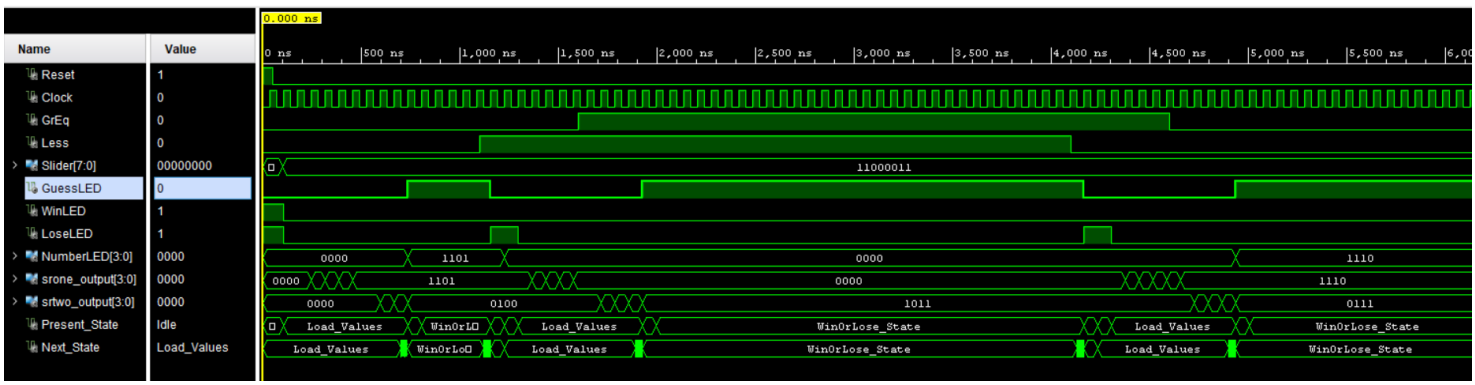
Timing-Diagram showing the outputs when the GrEq button is pressed, when reset is pressed and when no buttons have been pressed.



Timing-Diagram showing the outputs when the Less button is pressed, when reset is pressed and when no buttons have been pressed.

Invalid Combinations

The following diagram shows valid and invalid inputs by the user. As it can be seen in the picture, the Less button was pressend and consequently an output is given. However, If the user tries to press both buttons(GrEq and Less), the state machine will not execute anything and it will remain in the same state iterating until either only of the buttons is actually pressed.

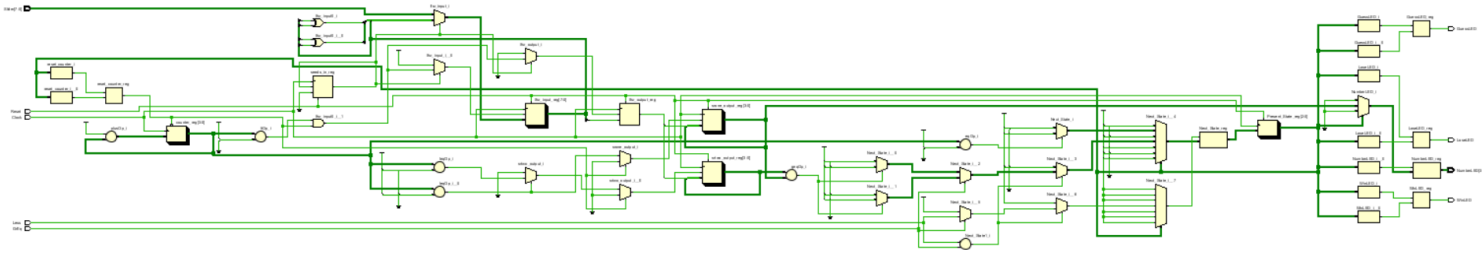


PART B: Synthetisation of VHDL Design

Register Transfer Level (RTL) schematic

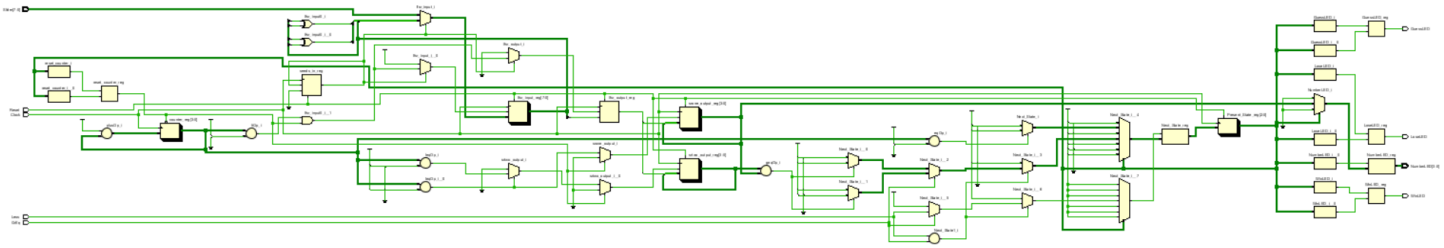
Gray Encoding

The RTL created for the Gray encoding models the synchronous digital circuit. It shows the flow of the digital signal between the registers and the logical operation performed on those signals.



One Hot

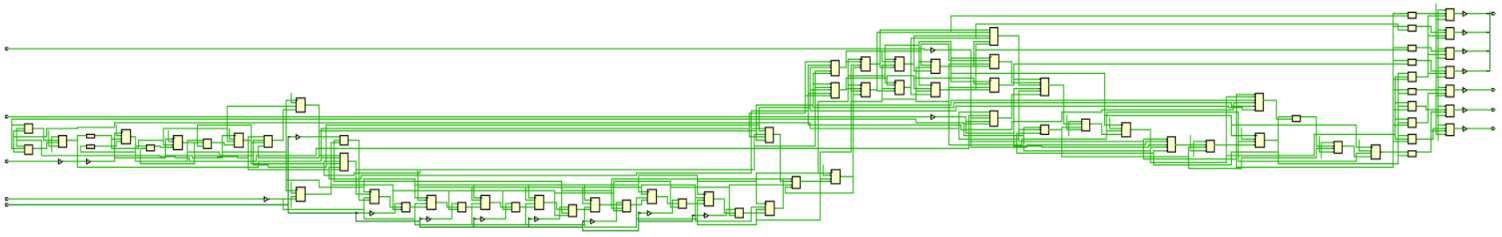
The RTL created for the One Hot encoding models the synchronous digital circuit. It shows the flow of the digital signal between the registers and the logical operation performed on those signals.



Synthesis

Gray Encoded State Machine

- **Schematic**



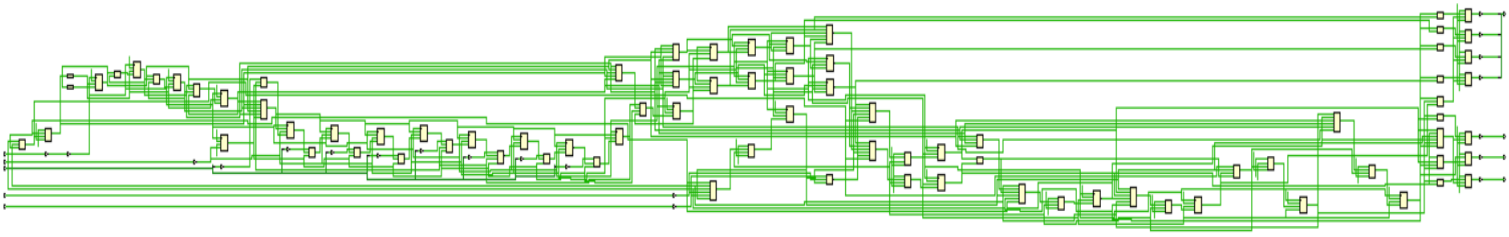
- **Design Report**

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	32	0	53200	0.06
LUT as Logic	32	0	53200	0.06
LUT as Memory	0	0	17400	0.00
Slice Registers	36	0	106400	0.03
Register as Flip Flop	25	0	106400	0.02
Register as Latch	11	0	106400	0.01
F7 Muxes	0	0	26600	0.00
F8 Muxes	0	0	13300	0.00

One Hot Encoded State Machine

- **Schematic**



- **Design Report**

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	33	0	53200	0.06
LUT as Logic	33	0	53200	0.06
LUT as Memory	0	0	17400	0.00
Slice Registers	44	0	106400	0.04
Register as Flip Flop	29	0	106400	0.03
Register as Latch	15	0	106400	0.01
F7 Muxes	0	0	26600	0.00
F8 Muxes	0	0	13300	0.00

Comparison of Gray and One Hot State Machine Encoding

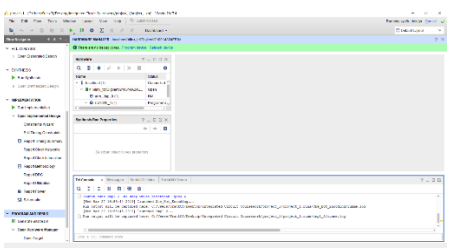
The number of Look Up Tables (LUT) used for Gray Encoding were 32 compared to 30 LUT for the One Hot. Thus, the number of Flip-Flops used for Gray Encoding were 25, whereas, there were 29 FF for the One Hot. One Hot Encoding designs a faster state machine; however, it requires more resources. On the other hand, Gray Encoding uses the same number of FFs as Binary encoding.

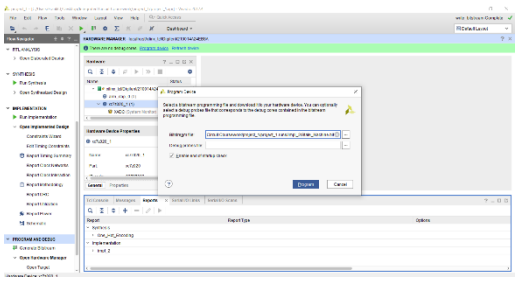
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM
Gray_Encoding (active)	constrs_1	synth_design Complete!								32	25	0.00	0
impl_2 (active)	constrs_1	route_design Complete!	NA	NA	NA	NA	NA	0.226	0	32	25	0.00	0
OneHot	constrs_1	synth_design Complete!								33	29	0.00	0
impl_1	constrs_1	route_design Complete!	NA	NA	NA	NA	NA	0.386	0	30	29	0.00	0

PART C: Implementation of VHDL Design

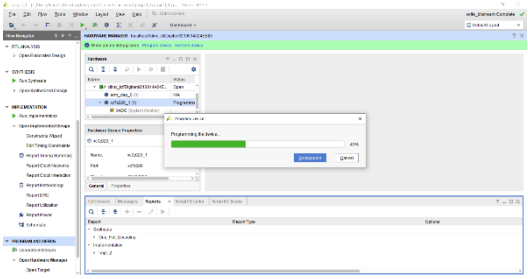
VHDL bitstream

1. Bitstream Generated





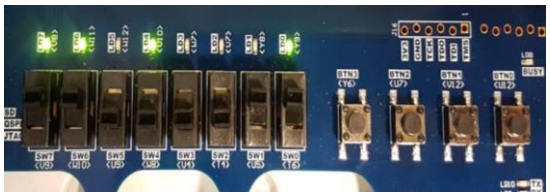
2. Bitstream uploaded to the board



3. Win and Lose leds switch on.

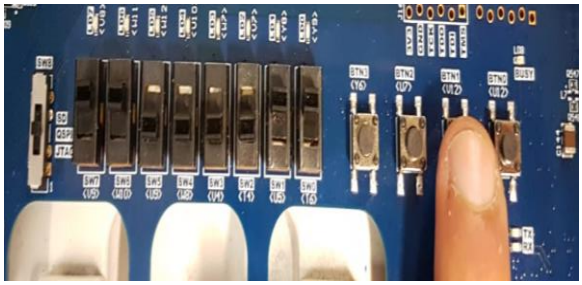
4. After the first Clock cycle the leds are off and the Slider is pass to the lfsr_input.

5. After eight clock cycle, values from the Shift register one are loaded into the NumLed signal. Guess Led is turned on.



6. oosing Less button and generating a clock cycle, and It can be result which is LoseLED on.

7. Back Again: Led signal are switched off and loading values to the Shift Register stars again



Response of the circuit

The only issue found during the development of the code has been when triing to manipulate it on the board. There have been some cases when pressing the button for the counter, it was only counting 7 times before the output were displayed on NumLED. Thus, There was sometimes more than 8 times pressing the cock before it display the value of NumbLED on the leds.

Appendix

```
----------------------------------------------------------------------------------
-- Company: Queen Mary University of London
-- Engineer: Frank Erasmo Cruz Felix
--
-- Create Date: 19.03.2019 22:31:02
-- Design Name: State Machine
-- Module Name: State_Machine - Behavioral
-- Project Name: Integrate Cirucit Design
-- Target Devices: XC7Z020CLG484-1
-- Tool Versions: VHDL 2017.4
-- Description:
--          The software developed inputs a random number (8 bits) and outputs a bit value,
--          which is store in two different shift registers. The value of one of these Shift Registers
--          is stored in NumLed, illuminating the corresponding high bit values. The two register values
--          are compared and depending of the user choice (GrEq or Less), a corresponding Led switches on
--          for WinLed or LoseLed.
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
```

```
-- Libraries used
library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

--State_Machine Port
entity State_Machine is
    PORT (  Reset      : IN  STD_LOGIC;
        Clock      : IN  STD_LOGIC;
        GrEq       : IN  STD_LOGIC;
        Less       : IN  STD_LOGIC;
        Slider     : IN  STD_LOGIC_VECTOR(7 DOWNTO 0); --Actual number
        GuessLED   : OUT STD_LOGIC;
        WinLED     : OUT STD_LOGIC;
        LoseLED    : OUT STD_LOGIC;
        NumberLED  : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
    );
end State_Machine;

architecture Behavioral of State_Machine is

-- Signal used for the software
signal lfsr_input : STD_LOGIC_VECTOR(7 DOWNTO 0);-- gets Slider input
signal lfsr_output: STD_LOGIC;-- gets value from the Linear Feedback Shift Register
signal counter: STD_LOGIC_VECTOR(3 DOWNTO 0);-- gets value of the counter
signal reset_counter: STD_LOGIC;-- rinitialize and resets the counter when it is called
signal seeds_in: STD_LOGIC;-- Allows lfsr_in get the bit values from the
signal srone_output: STD_LOGIC_VECTOR(3 DOWNTO 0);-- Shift Register One
signal srtwo_output: STD_LOGIC_VECTOR(3 DOWNTO 0);-- Shift Register Two

-- States
type StateType is (Idle,Load_Values,Guess_State,WinOrLose_State,Win, Lose,BackTo_State);
signal Present_State, Next_State : StateType;

begin
   -- Starting State
   state_clocked: process(Clock, Reset)
      BEGIN
        If(Reset = '1') THEN  --Initializing values when Reset high
            Present_State <= Idle;
        ELSIF (rising_edge(Clock)) THEN -- Clock event happens and Clock equals '1'
            Present_State <= Next_State;
        END IF;
    END PROCESS;

    --Counter state
    state_counter: process(Clock, Reset, Present_State,reset_counter)
      BEGIN
        IF(reset_counter = '0') THEN --Initializing values when Reset high
           counter <= "0000";
        ELSIF rising_edge(Clock) THEN -- Clock event happens and Clock equals '1'
            counter <= counter +1; -- Increasing value of counter by one
        END IF;
     END PROCESS;

   state_lfsr: process(Clock, Reset,reset_counter)
          BEGIN
             IF(Reset ='1') THEN --Initializing values when Reset high
                 lfsr_input <= "00000000";--Initilization of lfsr_input
                 lfsr_output <= '0';-- Initialiation of lfsr_output
                 seeds_in <= '1';-- making high seeds_in signal
             ELSIF(rising_edge(Clock)) THEN-- Clock event happens and Clock equals '1'
                  IF(seeds_in = '1')THEN
                    lfsr_input <= Slider;-- Gets Slider values
                    seeds_in <= '0';-- making seeds_in signal low
                  -- If counter value is less than 7 and reset_counter is high
                  ELSIF(counter < "0111" and reset_counter = '1')THEN
                     lfsr_output <= lfsr_input(7);
                     lfsr_input(0)<= lfsr_input(7);
                     lfsr_input(1)<= lfsr_input(0);
                     lfsr_input(2)<= lfsr_input(1);
                     lfsr_input(3)<= lfsr_input(2);
                     lfsr_input(4)<= lfsr_input(3) xor lfsr_input(7);--Xor values in order to obtain different values
                     lfsr_input(5)<= lfsr_input(4) xor lfsr_input(7);--Xor values in order to obtain different values
                     lfsr_input(6)<= lfsr_input(5);
                     lfsr_input(7)<= lfsr_input(6);
                  END IF;
             END IF;
        END PROCESS;

  -- Shift Register outputs
    LFSR_SF_input: process(Clock, Reset,reset_counter)
     BEGIN
         IF(Reset ='1')THEN  --Initializing values when Reset high
```

```
            srone_output <= "0000";
            srtwo_output <= "0000";
        -- Clock event happens and Clock equals '1'
        ELSIF(rising_edge(Clock))THEN
            IF(reset_counter = '1')THEN
                IF(counter <= "0100")THEN--If counter value is less than 5
                    srone_output(0)<= lfsr_output;-- srone_output gets the bits from the lfrs
                    srone_output(3 DOWNTO 1)<= srone_output(2 DOWNTO 0);-- Values are shifted to the right
                ELSIF(counter <= "1000")THEN -- If counter value is less than 9
                    srtwo_output(0)<= lfsr_output;-- srtwo_output gets the bits from the lfrs
                    srtwo_output(3 DOWNTO 1)<= srtwo_output(2 DOWNTO 0);-- Values are shifted to the right
                END IF;
            END IF;
        END IF;

    END PROCESS;
  -- States Process
  state_comb: process(Clock,Reset,Present_State,reset_counter)
    BEGIN
      --Case statement
      CASE present_state is
          --IDLE state
          WHEN Idle =>
              WinLED <= '1';-- WinLED on
              LoseLED <= '1';-- LoseLED on
              GuessLED <= '0';-- GuessLED off
              NumberLED <= "0000";-- Initial state for NumberLED
              reset_counter <= '0';-- Making reset counter low
              Next_State <= Load_values;-- Go to next State
          -- Load_values State
          WHEN Load_Values =>
             WinLED <= '0';-- WinLED off
             LoseLED <= '0';-- LoseLED off
             reset_counter <= '1';-- Making reset counter high
             IF(counter = "1000")THEN-- If counter equals 10 (decimal)
               Next_State <= Guess_State;-- Go to next State
             ELSE
               Next_State <= Load_Values;-- Go to next State
             END IF;
          WHEN Guess_State =>-- Guess_State state
               WinLED <= '0';-- WinLED off
               LoseLED <= '0';-- LoseLED off
               GuessLED <= '1';-- GuessLED on
               reset_counter <= '0';--reset counter
               NumberLED <= srone_output;-- NumberLED gets the value of the first Shift Register
               Next_State <= WinOrLose_State;-- Go to next State
          WHEN WinOrLose_State => -- WinOrLose_State state
               WinLED <= '0';-- WinLED off
               LoseLED <= '0';-- LoseLED off
             IF(Less = GrEq)THEN-- If the buttons have the same value then get back to the same state
                 Next_State <= WinOrLose_State;-- Go back to the same state
             ELSE
                 IF(GrEq = '1')THEN-- If GrEq is pressed
                     --If values of first SR is greater or equal to the second SR
                     IF(srone_output >= srtwo_output)THEN
                         Next_State <= Win;-- Go to Win state
                     ELSE
                        Next_State <= Lose;-- Go to Lose state
                     END IF;
                 ELSIF(Less = '1')THEN
                     --If values of first SR is greater or equal to the second SR
                     IF(srone_output >= srtwo_output)THEN
                        Next_State <= Lose;-- Go to Lose state
                     ELSE
                        Next_State <= Win;-- Go back to Load_Values state
                     END IF;
                 END IF;
             END IF;
           WHEN Win => --Win State
               WinLED <= '1';-- WindLED on
               LoseLED <= '0'; -- LosedLED on
               GuessLED <= '0'; -- GuessLED off
               Next_State <= BackTo_State;-- Go back to Load_Values state
           WHEN Lose=> -- Lose State
               WinLED <= '0';-- WindLED on
               LoseLED <= '1'; -- LosedLED on
               GuessLED <= '0'; -- GuessLED off
               Next_State <= BackTo_State;-- Go back to Load_Values state
           WHEN BackTo_State =>
               NumberLED <= "0000";-- Initial state for NumberLED
               Next_State <= Load_Values;-- Go back to Load_Values state
       END CASE;
   END PROCESS;

end Behavioral;
```