University of Nevada, Las Vegas
Department of Electrical and Computer Engineering

CPE 302 - Synthesis and Verification Using Programmable Devices

Final Project Report
Color Match RNG Game

Yanai Avila
Spring 2023

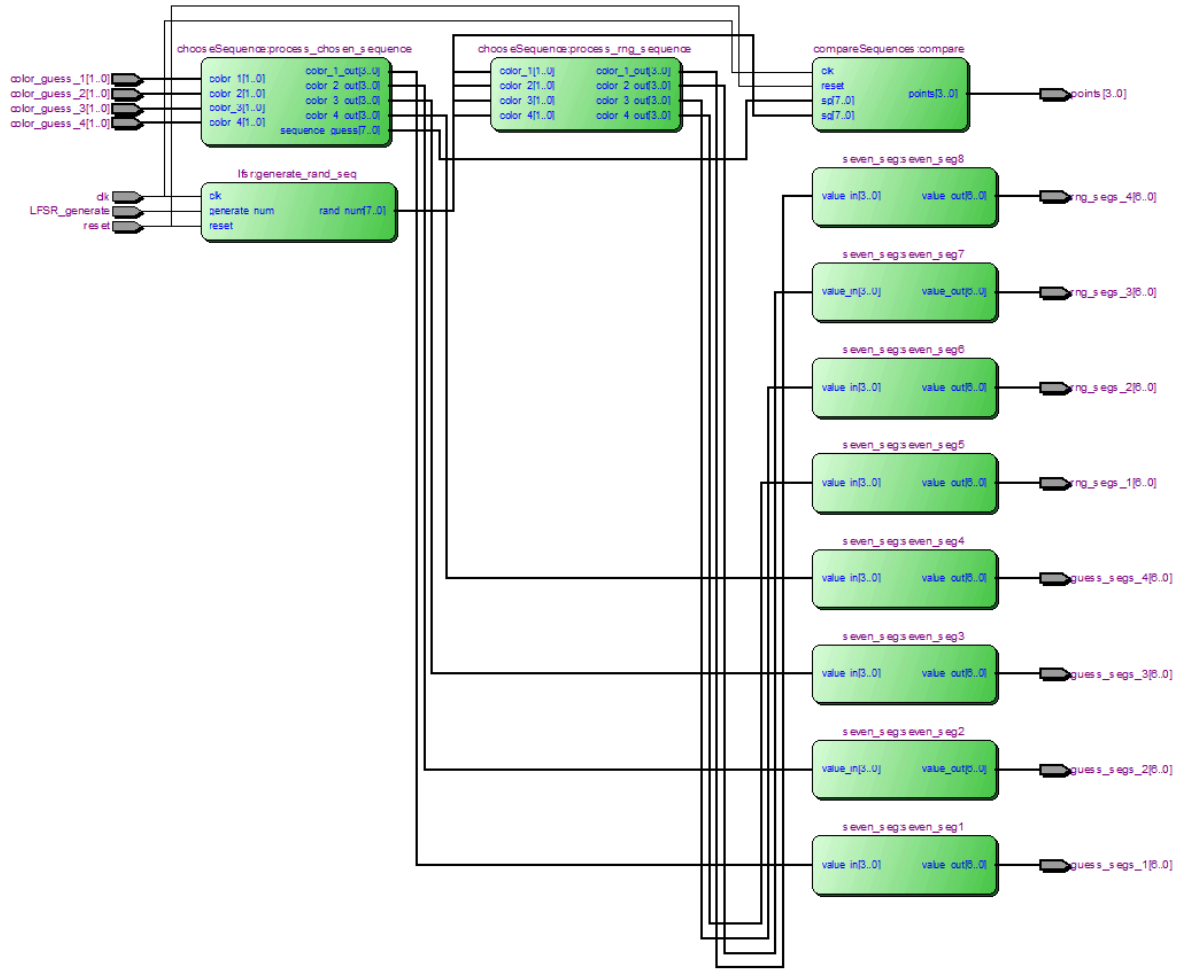
1. Introduction

Board Used: Altera DE2 Cyclone II (board overview)

Language: VHSIC Hardware Description Language (VHDL)

Project Description: For my final project, I implemented a simple mini-game that uses a random number generator. The player chooses a sequence of 4 colors they think will be randomly generated. They input this guess using the switches on the DE2 board. 00, 01, and 10. The 4 colors are randomly generated and displayed on the 7-segment displays. For each color the player guesses correctly, they gain a point.

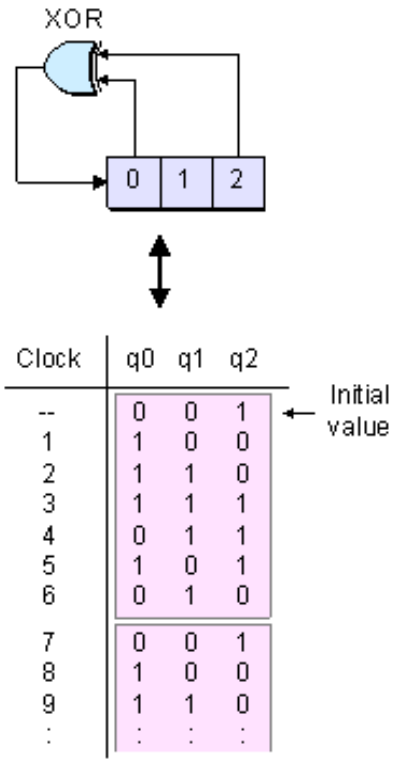
RTL View



Note: Modules are explained in section 2.

2. Modules

LFSR( ) - The main component of my project is a Linear Feeback Shift Register (LFSR). An LFSR is a pseudo-random number generator, and in my project, an LFSR is used to generate a random 8-bit number. This 8-bit output is used in another module to determine how many points the player gains based on how it compares to the 8-bit number that the player inputs as their guess. The figure below shows how XORing the tap bits for an n-bit number can generate a wide range of bit combinations. The LFSR will keep cycling through these combinations until a generate signal goes high. In my module, the generate_num signal has to go high for the rand_num output variable to take in the random combination at that point.



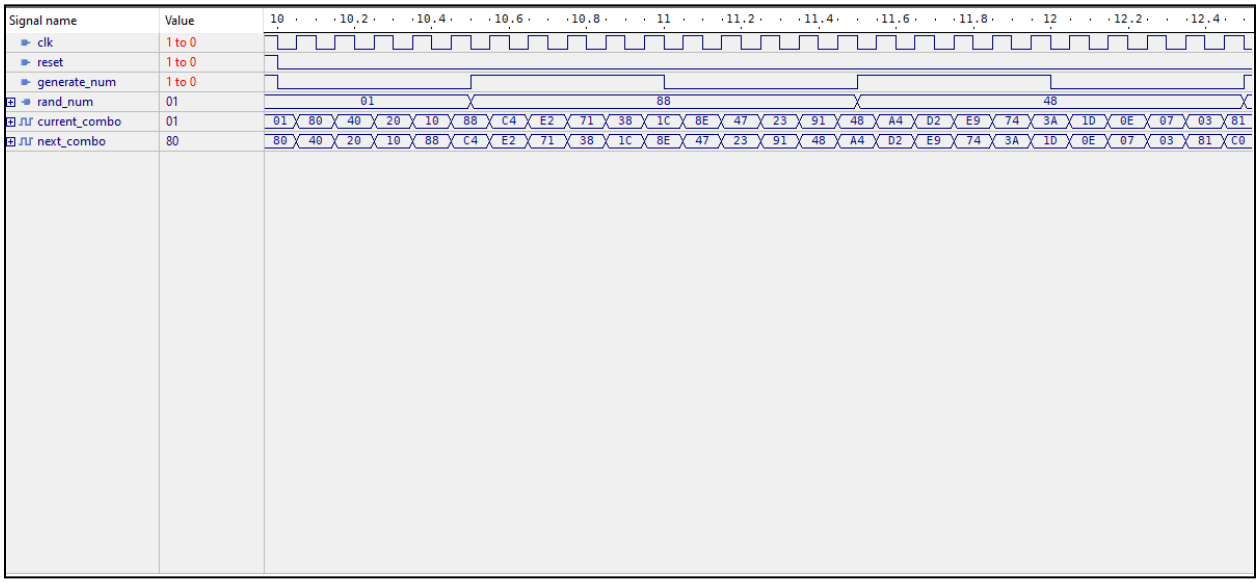
# of Bits	Length of Loop		Taps
2	3	*	[0,1]
3	7	*	[0,2]
4	15		[0,3]
5	31	*	[1,4]
6	63		[0,5]
7	127	*	[0,6]
8	255		[1,2,3,7]
9	511		[3,8]
10	1,023		[2,9]
11	2,047		[1,10]
12	4,095		[0,3,5,11]
13	8,191	*	[0,2,3,12]
14	16,383		[0,2,4,13]
15	32,767		[0,14]
16	65,535		[1,2,4,15]
17	131,071	*	[2,16]
18	262,143		[6,17]
19	524,287	*	[0,1,4,18]
20	1,048,575		[2,19]
21	2,097,151		[1,20]
22	4,194,303		[0,21]
23	8,388,607		[4,22]
24	16,777,215		[0,2,3,23]
25	33,554,431		[2,24]
26	67,108,863		[0,1,5,25]
27	134,217,727		[0,1,4,26]
28	268,435,455		[2,27]
29	536,870,911		[1,28]
30	1,073,741,823		[0,3,5,29]
31	2,147,483,647	*	[2,30]
32	4,294,967,295		[1,5,6,31]

* I used tap bits **1, 2, 3, 7**

Code for LFSR

```
180  entity lfsr is
181      Port ( clk          : in  std_logic;
182             reset        : in  std_logic;
183             generate_num : in std_logic;
184             rand_num     : out std_logic_vector(7 downto 0));
185  end lfsr;
186
187  architecture behavioral of lfsr is
188  signal current_combo, next_combo: std_logic_vector(7 downto 0);
189  signal feedback: std_logic;
190  begin
191    update: process (clk,reset)
192    begin
193      if (reset = '1') then
194        current_combo <= (0 => '1', others =>'0'); -- if reset, 00000001(cant be 0)
195      -- at rising clk edge
196      elsif (clk = '1' and clk'event) then
197        current_combo <= next_combo;
198      end if;
199    end process;
200
201    -- The feedback bit will be the XORing of differnt bits
202    feedback <= current_combo(4) xor current_combo(3) xor current_combo(2) xor current_combo(0);
203    -- shift current cobinatrion to the right and make the feedback bit the first bit
204    next_combo <= feedback & current_combo(7 downto 1);
205
206    -- When the pushbutton is pressed
207    generat: process(generate_num)
208    begin
209     if(generate_num = '1') then
210       -- the random number will take the value of the current combo(contantly changing)
211       rand_num <= current_combo;
212     end if;
213    end process generat;
214  end behavioral;
```

Simulation for LFSR



NOTE: Every time generate_num goes high, the rand_num (which is the output of the module) takes the value of current_combo. Current_combo changes rapidly at each clock edge, and when the player presses the generate pushbutton, the random number at that clock edge is displayed to the designated 7-segment display for the generated color sequence.

chooseSequence( ) - This module is a lookup table that processes *both* the number that the player inputs using the input switches and the random number generated using the LFSR. This allows the user to input their guess using the designated input switches and see their 4 selected colors on the 7-segment displays as well as see the random 4 colors on some other 7-segment displays. Even though the module is called "choose" sequence, it is also used for the generated number since the processing is the same for both.

This module takes in the 8-bit guess or 8-bit generated number, and processes it to be compatible with the input needed input for the 7-segment display module. This 8 bit input is divided into 2-bit signals for each 7-segment display, so in total there are 4 different combinations (00, 01, 10, 11), each representing a different color. These combinations are as follows:

00 is blue → "8/B" on the 7-segment display
01 is green G → "6/G" on the 7-segment display
10 is purple P → "P" on the 7-segment display
11 is yellow Y → "y" on the 7-segment display

Here we see that 00 is blue, so the output of this sequence is 0000, and this will be the input for the 7-segment display module which outputs the letter B (looks like "8") on the 7-segment display. The same applies to 01, which will output 0001 (corresponds to green), 10, which will output 0010 (corresponds to purple), and 11, which will output 0011(corresponds to yellow).

Note: I could have left the signals to be 2 bits, but I made the input signal for the 7-segment display module to be 4 bits to be able to accommodate more labels/colors for future improvements or additions to the game.

Code segment for the processing of color 1 (guess or generated)

```
-- case for color guessed 1
case(color_1) is
   when "00" => color_1_out <= "0000"; --B (8)
   when "01" => color_1_out <= "0001"; --G
   when "10" => color_1_out <= "0010"; --P
   when "11" => color_1_out <= "0011"; --Y
   when others => color_1_out <= "1111";
end case;
```

Finally, this module combines all these outputs into a sequence_guess variable by concatenating the 2 bit numbers together into an 8 bit number for easier use. This output is used in the compare sequences module to determine how many points the players gain. Again, this is also used for the generated number.

seven_seg( ) - This decoder is used to take a 4-bit value and convert it to a 7-bit signal to be able to display information on the 7-segment displays. The input is taken from the chooseSequence( ) module and the output is mapped to the displays (see chooseSequence( ) for mapping details)

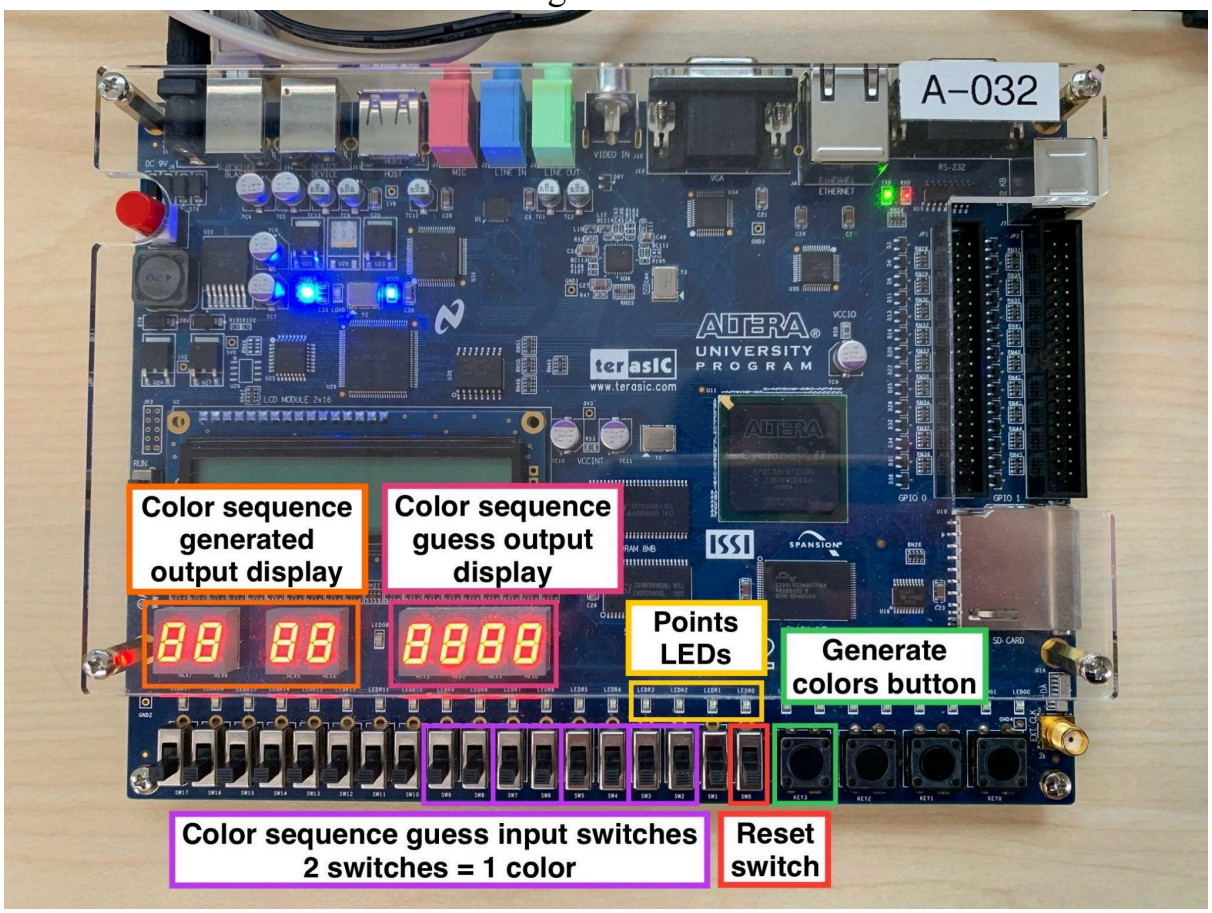
```
library ieee;
use ieee.std_logic_1164.all;
entity seven_seg is
    port(
    value_in : in std_logic_vector(3 downto 0);
    value_out : out std_logic_vector(6 downto 0)
);
end seven_seg;

architecture behavior of seven_seg is
begin
process(value_in)
begin
    case(value_in) is
        when "0000" => value_out <= "0000000"; --B
        when "0001" => value_out <= "0000010"; --G
        when "0010" => value_out <= "0001100"; --P
        when "0011" => value_out <= "0010001"; --Y
        when "0100" => value_out <= "0011001"; --4
        when "0101" => value_out <= "0010010"; --5
        when "0110" => value_out <= "0000010"; --6
        when "0111" => value_out <= "1111000"; --7
        when "1000" => value_out <= "0000000"; --8
        when "1001" => value_out <= "0010000"; --9
        when "1010" => value_out <= "0001000"; --a
        when "1011" => value_out <= "0000011"; --b
        when "1100" => value_out <= "1000110"; --c
        when "1101" => value_out <= "0100001"; --d
        when "1110" => value_out <= "0000110"; --e
        when others => value_out <= "0001110";
    end case;
end process;
end behavior;
```

compareSequences( ) - This module compares the sequence that was randomly generated to the sequence that the player guessed. The number of points is outputted to 4 LEDs on the board. The number of points depends on how many colors match. If the player gains 1 point, 1 LED lights up. If 2 points, 2 LEDs light up, and so on.

```
architecture Behavioral of compareSequences is
signal p0, p1, p2, p3: std_logic;
begin
    process(clk, sg)
    begin
        if(sp(7 downto 6) = sg(7 downto 6)) then
            p0 <= '1';
        else
            p0 <= '0';
        end if;

        if(sp(5 downto 4) = sg(5 downto 4)) then
            p1 <= '1';
        else
            p1 <= '0';
        end if;

        if(sp(3 downto 2) = sg(3 downto 2)) then
            p2 <= '1';
        else
            p2 <= '0';
        end if;

        if((sp(1 downto 0) = sg(1 downto 0))) then
            p3 <= '1';
        else
            p3 <= '0';
        end if;
    end process;

    points <= p0 & p1 & p2 & p3;
end Behavioral;
```

3. Pin Assignments

Pin assignments on board



Pin assignments on Active-HDL

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Curr
clk	Input	PIN_N2	2	B2_N1	PIN_N2	3.3-V LV...default)		24mA
color_guess_1[1]	Input	PIN_AE14	7	B7_N1	PIN_AE14	3.3-V LV...default)		24mA
color_guess_1[0]	Input	PIN_P25	6	B6_N0	PIN_P25	3.3-V LV...default)		24mA
color_guess_2[1]	Input	PIN_AD13	8	B8_N0	PIN_AD13	3.3-V LV...default)		24mA
color_guess_2[0]	Input	PIN_AF14	7	B7_N1	PIN_AF14	3.3-V LV...default)		24mA
color_guess_3[1]	Input	PIN_C13	3	B3_N0	PIN_C13	3.3-V LV...default)		24mA
color_guess_3[0]	Input	PIN_AC13	8	B8_N0	PIN_AC13	3.3-V LV...default)		24mA
color_guess_4[1]	Input	PIN_A13	4	B4_N1	PIN_A13	3.3-V LV...default)		24mA
color_guess_4[0]	Input	PIN_B13	4	B4_N1	PIN_B13	3.3-V LV...default)		24mA
guess_segs_1[6]	Output	PIN_V13	8	B8_N0	PIN_V13	3.3-V LV...default)		24mA
guess_segs_1[5]	Output	PIN_V14	8	B8_N0	PIN_V14	3.3-V LV...default)		24mA
guess_segs_1[4]	Output	PIN_AE11	8	B8_N0	PIN_AE11	3.3-V LV...default)		24mA
guess_segs_1[3]	Output	PIN_AD11	8	B8_N0	PIN_AD11	3.3-V LV...default)		24mA
guess_segs_1[2]	Output	PIN_AC12	8	B8_N0	PIN_AC12	3.3-V LV...default)		24mA
guess_segs_1[1]	Output	PIN_AB12	8	B8_N0	PIN_AB12	3.3-V LV...default)		24mA
guess_segs_1[0]	Output	PIN_AF10	8	B8_N0	PIN_AF10	3.3-V LV...default)		24mA
guess_segs_2[6]	Output	PIN_AB24	6	B6_N1	PIN_AB24	3.3-V LV...default)		24mA
guess_segs_2[5]	Output	PIN_AA23	6	B6_N1	PIN_AA23	3.3-V LV...default)		24mA
guess_segs_2[4]	Output	PIN_AA24	6	B6_N1	PIN_AA24	3.3-V LV...default)		24mA
guess_segs_2[3]	Output	PIN_Y22	6	B6_N1	PIN_Y22	3.3-V LV...default)		24mA
guess_segs_2[2]	Output	PIN_W21	6	B6_N1	PIN_W21	3.3-V LV...default)		24mA
guess_segs_2[1]	Output	PIN_V21	6	B6_N1	PIN_V21	3.3-V LV...default)		24mA
guess_segs_2[0]	Output	PIN_V20	6	B6_N1	PIN_V20	3.3-V LV...default)		24mA
guess_segs_3[6]	Output	PIN_Y24	6	B6_N1	PIN_Y24	3.3-V LV...default)		24mA
guess_segs_3[5]	Output	PIN_AB25	6	B6_N1	PIN_AB25	3.3-V LV...default)		24mA
guess_segs_3[4]	Output	PIN_AB26	6	B6_N1	PIN_AB26	3.3-V LV...default)		24mA
guess_segs_3[3]	Output	PIN_AC26	6	B6_N1	PIN_AC26	3.3-V LV...default)		24mA
guess_segs_3[2]	Output	PIN_AC25	6	B6_N1	PIN_AC25	3.3-V LV...default)		24mA
guess_segs_3[1]	Output	PIN_V22	6	B6_N1	PIN_V22	3.3-V LV...default)		24mA
guess_segs_3[0]	Output	PIN_AB23	6	B6_N1	PIN_AB23	3.3-V LV...default)		24mA
guess_segs_4[6]	Output	PIN_W24	6	B6_N1	PIN_W24	3.3-V LV...default)		24mA
guess_segs_4[5]	Output	PIN_U22	6	B6_N1	PIN_U22	3.3-V LV...default)		24mA
guess_segs_4[4]	Output	PIN_Y25	6	B6_N1	PIN_Y25	3.3-V LV...default)		24mA
guess_segs_4[3]	Output	PIN_Y26	6	B6_N1	PIN_Y26	3.3-V LV...default)		24mA
guess_segs_4[2]	Output	PIN_AA26	6	B6_N1	PIN_AA26	3.3-V LV...default)		24mA
guess_segs_4[1]	Output	PIN_AA25	6	B6_N1	PIN_AA25	3.3-V LV...default)		24mA
guess_segs_4[0]	Output	PIN_Y23	6	B6_N1	PIN_Y23	3.3-V LV...default)		24mA
LFSR_generate	Input	PIN_W26	6	B6_N1	PIN_W26	3.3-V LV...default)		24mA
points[3]	Output	PIN_AC22	7	B7_N0	PIN_AC22	3.3-V LV...default)		24mA
points[2]	Output	PIN_AB21	7	B7_N0	PIN_AB21	3.3-V LV...default)		24mA
points[1]	Output	PIN_AF23	7	B7_N0	PIN_AF23	3.3-V LV...default)		24mA
points[0]	Output	PIN_AE23	7	B7_N0	PIN_AE23	3.3-V LV...default)		24mA
reset	Input	PIN_N26	5	B5_N1	PIN_N26	3.3-V LV...default)		24mA
rng_segs_1[6]	Output	PIN_T3	1	B1_N0	PIN_T3	3.3-V LV...default)		24mA
rng_segs_1[5]	Output	PIN_R6	1	B1_N0	PIN_R6	3.3-V LV...default)		24mA
rng_segs_1[4]	Output	PIN_R7	1	B1_N0	PIN_R7	3.3-V LV...default)		24mA
rng_segs_1[3]	Output	PIN_T4	1	B1_N0	PIN_T4	3.3-V LV...default)		24mA
rng_segs_1[2]	Output	PIN_U2	1	B1_N0	PIN_U2	3.3-V LV...default)		24mA
rng_segs_1[1]	Output	PIN_U1	1	B1_N0	PIN_U1	3.3-V LV...default)		24mA
rng_segs_1[0]	Output	PIN_U9	1	B1_N0	PIN_U9	3.3-V LV...default)		24mA
rng_segs_2[6]	Output	PIN_R3	1	B1_N0	PIN_R3	3.3-V LV...default)		24mA
rng_segs_2[5]	Output	PIN_R4	1	B1_N0	PIN_R4	3.3-V LV...default)		24mA
rng_segs_2[4]	Output	PIN_R5	1	B1_N0	PIN_R5	3.3-V LV...default)		24mA
rng_segs_2[3]	Output	PIN_T9	1	B1_N0	PIN_T9	3.3-V LV...default)		24mA
rng_segs_2[2]	Output	PIN_P7	1	B1_N0	PIN_P7	3.3-V LV...default)		24mA
rng_segs_2[1]	Output	PIN_P6	1	B1_N0	PIN_P6	3.3-V LV...default)		24mA
rng_segs_2[0]	Output	PIN_T2	1	B1_N0	PIN_T2	3.3-V LV...default)		24mA
rng_segs_3[6]	Output	PIN_M4	2	B2_N1	PIN_M4	3.3-V LV...default)		24mA
rng_segs_3[5]	Output	PIN_M5	2	B2_N1	PIN_M5	3.3-V LV...default)		24mA
rng_segs_3[4]	Output	PIN_M3	2	B2_N1	PIN_M3	3.3-V LV...default)		24mA
rng_segs_3[3]	Output	PIN_M2	2	B2_N1	PIN_M2	3.3-V LV...default)		24mA
rng_segs_3[2]	Output	PIN_P3	1	B1_N0	PIN_P3	3.3-V LV...default)		24mA
rng_segs_3[1]	Output	PIN_P4	1	B1_N0	PIN_P4	3.3-V LV...default)		24mA
rng_segs_3[0]	Output	PIN_R2	1	B1_N0	PIN_R2	3.3-V LV...default)		24mA
rng_segs_4[6]	Output	PIN_N9	2	B2_N1	PIN_N9	3.3-V LV...default)		24mA
rng_segs_4[5]	Output	PIN_P9	2	B2_N1	PIN_P9	3.3-V LV...default)		24mA
rng_segs_4[4]	Output	PIN_L7	2	B2_N1	PIN_L7	3.3-V LV...default)		24mA
rng_segs_4[3]	Output	PIN_L6	2	B2_N1	PIN_L6	3.3-V LV...default)		24mA
rng_segs_4[2]	Output	PIN_L9	2	B2_N1	PIN_L9	3.3-V LV...default)		24mA
rng_segs_4[1]	Output	PIN_L2	2	B2_N1	PIN_L2	3.3-V LV...default)		24mA
rng_segs_4[0]	Output	PIN_L3	2	B2_N1	PIN_L3	3.3-V LV...default)		24mA
points[7]	Unknown	PIN_AC21	7	B7_N0		3.3-V LV...default)		24mA
points[6]	Unknown	PIN_AD21	7	B7_N0		3.3-V LV...default)		24mA
points[5]	Unknown	PIN_AD23	7	B7_N0		3.3-V LV...default)		24mA
points[4]	Unknown	PIN_AD22	7	B7_N0		3.3-V LV...default)		24mA
<<new node>>								

4. Video
Video link: https://youtu.be/WUGZxaSniOU

5. Code
Code link: https://github.com/YanaiAvila/cpe-302/blob/main/CPE302-Final-Project-Code.vhd

6. Future Improvements
One of the things I would have liked to figure out with this project is how to implement a sequence on the VGA monitor to provide a visual of the colors instead of the labels on the 7-segment displays. This was the original plan which is why I made this game to have colors. I was unable to figure out what I was doing wrong when trying to display something on the VGA monitor. Because of this, I decided to quickly re-adjust my project into a much simpler one using the 7-segments to demonstrate the concept, but adding a monitor display would not be too difficult if given more time.

Another improvement I would make is to be able to add the points for each round. The way I have it now, the player can only see their points for each round. I would like to have it so that these points accumulate and the player wins at a certain amount of points.