**University of Nevada Las Vegas. Department of Electrical and Computer Engineering Laboratories.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class: | **CPE 100L - 1003** | | | Semester: | **Spring 2018** |
|  | | | | | |
| Points |  | Document author: | **Nicholas Poole** | | |
|  | Author's email: | **poolen@unlv.nevada.edu** | | |
|  | | | |
| Document topic: | **Postlab Final Project** | | |
| Instructor's comments: | | | | | |

**1. The Goal**

The goal for the final project was to use elements of sequential design and VHDL code to create a Whack-A-Mole inspired game called Super-Slug-A-Bug. In short, the game requires the player to hit the moles when they pop up, gaining points. The player must hit them quickly before they disappear. The game has a time limit as well, so the player must do the best they could before the time runs out. To create this project required a few elements. The goal was to create a randomizer, a makeshift hammer, to hit the regular bugs and super bugs (which are worth extra points), as well as the rest of the game mechanics. These include a scrolling start screen, a start game button which sets of the timer count down as well as will increment the score as you play, as well as an end screen which reads “good job.” This project required use of almost all aspects of the DE0 board. The LED lights are used as the power bar which is necessary to be completely filled up to kill super bugs. The push buttons are used as the hammer, and you must press it quickly to get points. The 7-segment display is used for many aspects. First, the 7-segment display is used to display the welcome message. Second, the 7-segment display is used to display the bugs which pop on the screen, there are two different types of bugs regular and super and they both look different. Also, the two left-most 7 segments display the score and time remaining during game execution, these can be toggled by a switch. Lastly, the 7 segment is used to display a message after the game is over, and then you must press the right most push button, the new game button, to bring you back to the start menu.

**2. Roles**

The roles for this experiment were as follows:

Week 1-2:

Levi: Randomizer and basic game mechanics (basic I/O).

Nick: Decoders for score, bugs and super-bugs.

Week 3:

Levi: Variable scores and timer.

Nick: “Menu screen” mode, “Game over” message, and switching between states.

Week 4: Debugging, reports, presentation

All the different pieces of the design as mentioned in the beginning are as follows: the randomizer (constructed with several pseudo-random VHDL inputs, as well as a 25 hertz signal), the hammer (which will also light up the LEDs representing the power bar), and the game block which contains the timer, the counter for the score, as well as 3 different states: start, game, and again for displaying the welcome message, the game itself, and the end message.

Nick: Built the hammer module (bugs and super bugs), as well as the scrolling messages for the beginning of the game and the end message (also a scrolling message). Took all the modules and put them together in the game block, to switch between states.

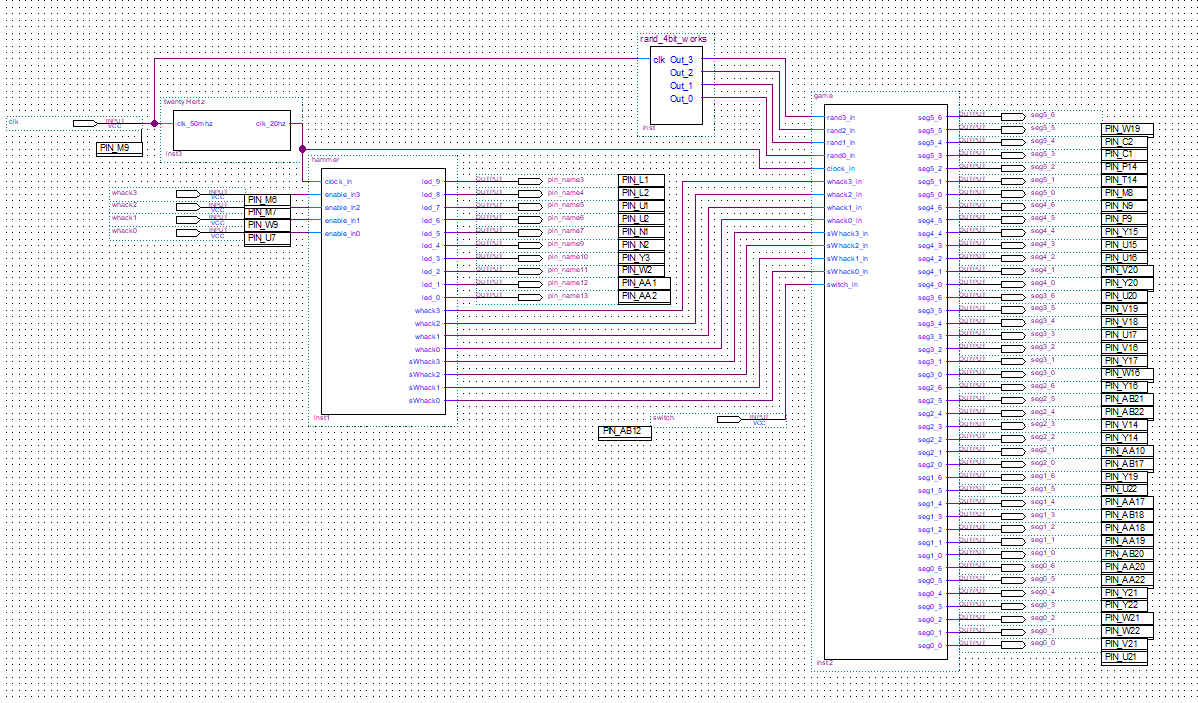
Levi: Built the randomizer (with pseudo random generator), as well as the counter for the score (possibilities 0-99). Created the timer countdown (initially 20 to 0) but lengthened to (40 to 0). Also responsible for creating the presentation and the project report.

**3. Background Theory**

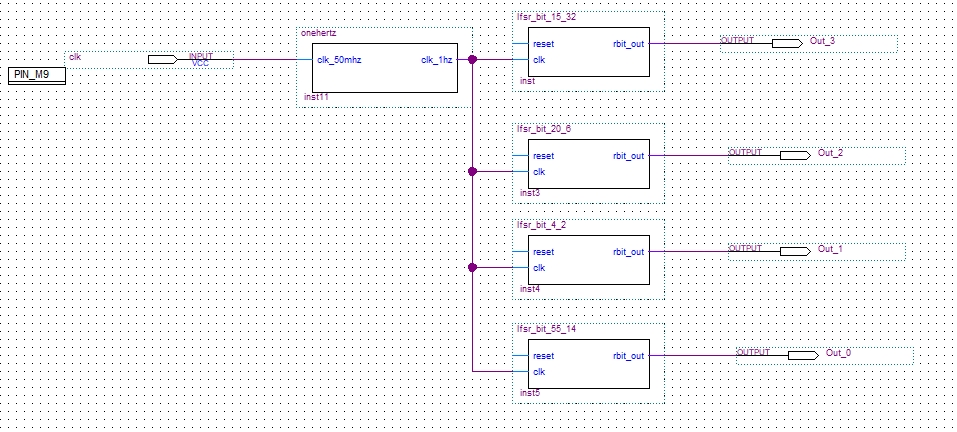
The background for this project was as follows: we set out to create the whack-a-mole game with a couple twists on the DE0 board. Most of the project was created in VHDL code, with a few slices of Verilog code as well. The project was divided into states depending on clock ticks, which is being run by a 25-hertz signal. The idea was to create the different elements necessary for the game and to join it into blocks of VHDL code bound by if statements that are dependent of the timer of the game. This is how we get from one state of the game to another. We have the 25-hertz signal, the randomizer which generates 4 random 1-bit numbers, the hammer which controls the LED lights to light up the longer it is pressed, and the game block which is almost the entire project in one. The game block contains a ton of elements: the timer, the counter, the super bugs and regular bugs, the scrolling messages, the buffers for super bugs and regular bugs, etc.…

**4. Schematics/Diagrams**

**The general schematic:**



**The randomizer:**



**One hertz signal: (Verilog)**

module onehertz(clk\_50mhz, clk\_1hz);

input clk\_50mhz;

output clk\_1hz;

reg clk\_1hz;

reg [24:0] count;

always @ (posedge clk\_50mhz)

begin if(count == 24999999)

begin count <= 0;

$dumpfile("f.vcd");

clk\_1hz <= ~clk\_1hz;

end

else begin

count <= count + 1;

end

end

endmodule

**LFSR (Pseudo Random Bit Generator):**

**Each is 8-bit (generates 4 1-bit random numbers) (VHDL):**

library ieee;

use ieee.std\_logic\_1164.all;

entity lfsr\_bit\_15\_32 is

port (

reset : in std\_logic;

clk : in std\_logic;

rbit\_out : out std\_logic

);

end entity;

architecture rtl of lfsr\_bit\_15\_32 is

signal lfsr : std\_logic\_vector (7 downto 0);

signal feedback : std\_logic;

begin

feedback <= not(lfsr(3) xor lfsr(6));

sr\_pr : process (clk)

begin

if (rising\_edge(clk)) then

if (reset = '1') then

lfsr <= (others => '0');

else

lfsr <= lfsr(6 downto 0) & feedback;

end if;

end if;

end process sr\_pr;

rbit\_out <= lfsr(7);

end architecture;

**Now, for these they are all exactly the same besides for feedback <=not(lfsr(3) xor lfsr(6))**

**4 different variations:**

feedback <= not(lfsr(2) xor lfsr(4));

feedback <= not(lfsr(3) xor lfsr(6));

feedback <= not(lfsr(1) xor lfsr(6));

feedback <= not(lfsr(0) xor lfsr(4));

**This is to change the combination of outputs, so the 4 1-bit numbers aren’t synchronized, and we have a random output.**

**25 hertz signal (Verilog) (Yes it is 25 hz despite the fact it says 20 hz)**

module twentyHertz(clk\_50mhz, clk\_20hz);

input clk\_50mhz;

output clk\_20hz;

reg clk\_20hz;

reg [24:0] count;

always @ (posedge clk\_50mhz)

begin if(count == 999999)

begin count <= 0;

$dumpfile("f.vcd");

clk\_20hz <= ~clk\_20hz;

end

else begin

count <= count + 1;

end

end

endmodule

**The Hammer Module: (VHDL)**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

entity hammer is

port(

clock\_in : in std\_logic;

enable\_in3 : in std\_logic;

enable\_in2 : in std\_logic;

enable\_in1 : in std\_logic;

enable\_in0 : in std\_logic;

led\_9 : out std\_logic;

led\_8 : out std\_logic;

led\_7 : out std\_logic;

led\_6 : out std\_logic;

led\_5 : out std\_logic;

led\_4 : out std\_logic;

led\_3 : out std\_logic;

led\_2 : out std\_logic;

led\_1 : out std\_logic;

led\_0 : out std\_logic;

whack3 : out std\_logic;

whack2 : out std\_logic;

whack1 : out std\_logic;

whack0 : out std\_logic;

sWhack3 : out std\_logic;

sWhack2 : out std\_logic;

sWhack1 : out std\_logic;

sWhack0 : out std\_logic

);

end hammer;

architecture rtl of hammer is

signal clock : std\_logic;

signal enable3 : std\_logic;

signal enable2 : std\_logic;

signal enable1 : std\_logic;

signal enable0 : std\_logic;

signal reset : std\_logic;

begin

clock <= clock\_in;

enable3 <= enable\_in3;

enable2 <= enable\_in2;

enable1 <= enable\_in1;

enable0 <= enable\_in0;

process(clock, enable3, enable2, enable1, enable0)

variable temp : std\_logic\_vector (3 downto 0) := "0000";

BEGIN

if enable3 = '0' or enable2 = '0' or enable1 = '0' or enable0 = '0' then

if (rising\_edge(clock)) then

if temp /= "1011" then

temp := temp + '1';

end if;

end if;

elsif enable3 = '1' or enable2 = '1' or enable1 = '1' or enable0 = '1' then

temp := "0000";

end if;

case temp is

when "0001" =>

led\_9 <= '1'; -- '0'

led\_8 <= '0'; -- '1'

led\_7 <= '0'; -- '2'

led\_6 <= '0'; -- '3'

led\_5 <= '0'; -- '4'

led\_4 <= '0'; -- '5'

led\_3 <= '0'; -- '6'

led\_2 <= '0'; -- '7'

led\_1 <= '0'; -- '8'

led\_0 <= '0'; -- '9'

if enable3 = '0' then

whack3 <= '1';

else

whack3 <= '0';

end if;

if enable2 = '0' then

whack2 <= '1';

else

whack2 <= '0';

end if;

if enable1 = '0' then

whack1 <= '1';

else

whack1 <= '0';

end if;

if enable0 = '0' then

whack0 <= '1';

else

whack0 <= '0';

end if;

sWhack3 <= '0';

sWhack2 <= '0';

sWhack1 <= '0';

sWhack0 <= '0';

when "0010" =>

led\_9 <= '1'; -- '0'

led\_8 <= '1'; -- '1'

led\_7 <= '0'; -- '2'

led\_6 <= '0'; -- '3'

led\_5 <= '0'; -- '4'

led\_4 <= '0'; -- '5'

led\_3 <= '0'; -- '6'

led\_2 <= '0'; -- '7'

led\_1 <= '0'; -- '8'

led\_0 <= '0'; -- '9'

whack3 <= '0';

whack2 <= '0';

whack1 <= '0';

whack0 <= '0';

sWhack3 <= '0';

sWhack2 <= '0';

sWhack1 <= '0';

sWhack0 <= '0';

when "0011" =>

led\_9 <= '1'; -- '0'

led\_8 <= '1'; -- '1'

led\_7 <= '1'; -- '2'

led\_6 <= '0'; -- '3'

led\_5 <= '0'; -- '4'

led\_4 <= '0'; -- '5'

led\_3 <= '0'; -- '6'

led\_2 <= '0'; -- '7'

led\_1 <= '0'; -- '8'

led\_0 <= '0'; -- '9'

whack3 <= '0';

whack2 <= '0';

whack1 <= '0';

whack0 <= '0';

sWhack3 <= '0';

sWhack2 <= '0';

sWhack1 <= '0';

sWhack0 <= '0';

when "0100" =>

led\_9 <= '1'; -- '0'

led\_8 <= '1'; -- '1'

led\_7 <= '1'; -- '2'

led\_6 <= '1'; -- '3'

led\_5 <= '0'; -- '4'

led\_4 <= '0'; -- '5'

led\_3 <= '0'; -- '6'

led\_2 <= '0'; -- '7'

led\_1 <= '0'; -- '8'

led\_0 <= '0'; -- '9'

whack3 <= '0';

whack2 <= '0';

whack1 <= '0';

whack0 <= '0';

sWhack3 <= '0';

sWhack2 <= '0';

sWhack1 <= '0';

sWhack0 <= '0';

when "0101" =>

led\_9 <= '1'; -- '0'

led\_8 <= '1'; -- '1'

led\_7 <= '1'; -- '2'

led\_6 <= '1'; -- '3'

led\_5 <= '1'; -- '4'

led\_4 <= '0'; -- '5'

led\_3 <= '0'; -- '6'

led\_2 <= '0'; -- '7'

led\_1 <= '0'; -- '8'

led\_0 <= '0'; -- '9'

whack3 <= '0';

whack2 <= '0';

whack1 <= '0';

whack0 <= '0';

sWhack3 <= '0';

sWhack2 <= '0';

sWhack1 <= '0';

sWhack0 <= '0';

when "0110" =>

led\_9 <= '1'; -- '0'

led\_8 <= '1'; -- '1'

led\_7 <= '1'; -- '2'

led\_6 <= '1'; -- '3'

led\_5 <= '1'; -- '4'

led\_4 <= '1'; -- '5'

led\_3 <= '0'; -- '6'

led\_2 <= '0'; -- '7'

led\_1 <= '0'; -- '8'

led\_0 <= '0'; -- '9'

whack3 <= '0';

whack2 <= '0';

whack1 <= '0';

whack0 <= '0';

sWhack3 <= '0';

sWhack2 <= '0';

sWhack1 <= '0';

sWhack0 <= '0';

when "0111" =>

led\_9 <= '1'; -- '0'

led\_8 <= '1'; -- '1'

led\_7 <= '1'; -- '2'

led\_6 <= '1'; -- '3'

led\_5 <= '1'; -- '4'

led\_4 <= '1'; -- '5'

led\_3 <= '1'; -- '6'

led\_2 <= '0'; -- '7'

led\_1 <= '0'; -- '8'

led\_0 <= '0'; -- '9'

whack3 <= '0';

whack2 <= '0';

whack1 <= '0';

whack0 <= '0';

sWhack3 <= '0';

sWhack2 <= '0';

sWhack1 <= '0';

sWhack0 <= '0';

when "1000" =>

led\_9 <= '1'; -- '0'

led\_8 <= '1'; -- '1'

led\_7 <= '1'; -- '2'

led\_6 <= '1'; -- '3'

led\_5 <= '1'; -- '4'

led\_4 <= '1'; -- '5'

led\_3 <= '1'; -- '6'

led\_2 <= '1'; -- '7'

led\_1 <= '0'; -- '8'

led\_0 <= '0'; -- '9'

whack3 <= '0';

whack2 <= '0';

whack1 <= '0';

whack0 <= '0';

sWhack3 <= '0';

sWhack2 <= '0';

sWhack1 <= '0';

sWhack0 <= '0';

when "1001" =>

led\_9 <= '1'; -- '0'

led\_8 <= '1'; -- '1'

led\_7 <= '1'; -- '2'

led\_6 <= '1'; -- '3'

led\_5 <= '1'; -- '4'

led\_4 <= '1'; -- '5'

led\_3 <= '1'; -- '6'

led\_2 <= '1'; -- '7'

led\_1 <= '1'; -- '8'

led\_0 <= '0'; -- '9'

whack3 <= '0';

whack2 <= '0';

whack1 <= '0';

whack0 <= '0';

sWhack3 <= '0';

sWhack2 <= '0';

sWhack1 <= '0';

sWhack0 <= '0';

when "1010" =>

led\_9 <= '1'; -- '0'

led\_8 <= '1'; -- '1'

led\_7 <= '1'; -- '2'

led\_6 <= '1'; -- '3'

led\_5 <= '1'; -- '4'

led\_4 <= '1'; -- '5'

led\_3 <= '1'; -- '6'

led\_2 <= '1'; -- '7'

led\_1 <= '1'; -- '8'

led\_0 <= '1'; -- '9'

whack3 <= '0';

whack2 <= '0';

whack1 <= '0';

whack0 <= '0';

if enable3 = '0' then

sWhack3 <= '1';

else

sWhack3 <= '0';

end if;

if enable2 = '0' then

sWhack2 <= '1';

else

sWhack2 <= '0';

end if;

if enable1 = '0' then

sWhack1 <= '1';

else

sWhack1 <= '0';

end if;

if enable0 = '0' then

sWhack0 <= '1';

else

sWhack0 <= '0';

end if;

when "1011" =>

led\_9 <= '0'; -- '0'

led\_8 <= '0'; -- '1'

led\_7 <= '0'; -- '2'

led\_6 <= '0'; -- '3'

led\_5 <= '0'; -- '4'

led\_4 <= '0'; -- '5'

led\_3 <= '0'; -- '6'

led\_2 <= '0'; -- '7'

led\_1 <= '0'; -- '8'

led\_0 <= '0'; -- '9'

whack3 <= '0';

whack2 <= '0';

whack1 <= '0';

whack0 <= '0';

sWhack3 <= '0';

sWhack2 <= '0';

sWhack1 <= '0';

sWhack0 <= '0';

when others =>

led\_0 <= '0'; -- '0'

led\_1 <= '0'; -- '1'

led\_2 <= '0'; -- '2'

led\_3 <= '0'; -- '3'

led\_4 <= '0'; -- '4'

led\_5 <= '0'; -- '5'

led\_6 <= '0'; -- '6'

led\_7 <= '0'; -- '7'

led\_8 <= '0'; -- '8'

led\_9 <= '0'; -- '9'

whack3 <= '0';

whack2 <= '0';

whack1 <= '0';

whack0 <= '0';

sWhack3 <= '0';

sWhack2 <= '0';

sWhack1 <= '0';

sWhack0 <= '0';

end case;

end process;

end rtl;

**The Game Block/Module (VHDL):**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

entity game is

port(

rand3\_in : in std\_logic;

rand2\_in : in std\_logic;

rand1\_in : in std\_logic;

rand0\_in : in std\_logic;

clock\_in : in std\_logic;

whack3\_in : in std\_logic;

whack2\_in : in std\_logic;

whack1\_in : in std\_logic;

whack0\_in : in std\_logic;

sWhack3\_in : in std\_logic;

sWhack2\_in : in std\_logic;

sWhack1\_in : in std\_logic;

sWhack0\_in : in std\_logic;

switch\_in : in std\_logic;

seg5\_6 : out std\_logic;

seg5\_5 : out std\_logic;

seg5\_4 : out std\_logic;

seg5\_3 : out std\_logic;

seg5\_2 : out std\_logic;

seg5\_1 : out std\_logic;

seg5\_0 : out std\_logic;

seg4\_6 : out std\_logic;

seg4\_5 : out std\_logic;

seg4\_4 : out std\_logic;

seg4\_3 : out std\_logic;

seg4\_2 : out std\_logic;

seg4\_1 : out std\_logic;

seg4\_0 : out std\_logic;

seg3\_6 : out std\_logic;

seg3\_5 : out std\_logic;

seg3\_4 : out std\_logic;

seg3\_3 : out std\_logic;

seg3\_2 : out std\_logic;

seg3\_1 : out std\_logic;

seg3\_0 : out std\_logic;

seg2\_6 : out std\_logic;

seg2\_5 : out std\_logic;

seg2\_4 : out std\_logic;

seg2\_3 : out std\_logic;

seg2\_2 : out std\_logic;

seg2\_1 : out std\_logic;

seg2\_0 : out std\_logic;

seg1\_6 : out std\_logic;

seg1\_5 : out std\_logic;

seg1\_4 : out std\_logic;

seg1\_3 : out std\_logic;

seg1\_2 : out std\_logic;

seg1\_1 : out std\_logic;

seg1\_0 : out std\_logic;

seg0\_6 : out std\_logic;

seg0\_5 : out std\_logic;

seg0\_4 : out std\_logic;

seg0\_3 : out std\_logic;

seg0\_2 : out std\_logic;

seg0\_1 : out std\_logic;

seg0\_0 : out std\_logic

);

end game;

architecture rtl of game is

signal clock : std\_logic;

signal rand3 : std\_logic;

signal rand2 : std\_logic;

signal rand1 : std\_logic;

signal rand0 : std\_logic;

signal whack3 : std\_logic;

signal whack2 : std\_logic;

signal whack1 : std\_logic;

signal whack0 : std\_logic;

signal sWhack3 : std\_logic;

signal sWhack2 : std\_logic;

signal sWhack1 : std\_logic;

signal sWhack0 : std\_logic;

signal seg5 : std\_logic\_vector (6 downto 0);

signal seg4 : std\_logic\_vector (6 downto 0);

signal seg3 : std\_logic\_vector (6 downto 0);

signal seg2 : std\_logic\_vector (6 downto 0);

signal seg1 : std\_logic\_vector (6 downto 0);

signal seg0 : std\_logic\_vector (6 downto 0);

signal switch: std\_logic;

begin

clock <= clock\_in;

rand3 <= rand3\_in;

rand2 <= rand2\_in;

rand1 <= rand1\_in;

rand0 <= rand0\_in;

whack3 <= whack3\_in;

whack2 <= whack2\_in;

whack1 <= whack1\_in;

whack0 <= whack0\_in;

sWhack3 <= sWhack3\_in;

sWhack2 <= sWhack2\_in;

sWhack1 <= sWhack1\_in;

sWhack0 <= sWhack0\_in;

seg5\_6 <= seg5(6);

seg5\_5 <= seg5(5);

seg5\_4 <= seg5(4);

seg5\_3 <= seg5(3);

seg5\_2 <= seg5(2);

seg5\_1 <= seg5(1);

seg5\_0 <= seg5(0);

seg4\_6 <= seg4(6);

seg4\_5 <= seg4(5);

seg4\_4 <= seg4(4);

seg4\_3 <= seg4(3);

seg4\_2 <= seg4(2);

seg4\_1 <= seg4(1);

seg4\_0 <= seg4(0);

seg3\_6 <= seg3(6);

seg3\_5 <= seg3(5);

seg3\_4 <= seg3(4);

seg3\_3 <= seg3(3);

seg3\_2 <= seg3(2);

seg3\_1 <= seg3(1);

seg3\_0 <= seg3(0);

seg2\_6 <= seg2(6);

seg2\_5 <= seg2(5);

seg2\_4 <= seg2(4);

seg2\_3 <= seg2(3);

seg2\_2 <= seg2(2);

seg2\_1 <= seg2(1);

seg2\_0 <= seg2(0);

seg1\_6 <= seg1(6);

seg1\_5 <= seg1(5);

seg1\_4 <= seg1(4);

seg1\_3 <= seg1(3);

seg1\_2 <= seg1(2);

seg1\_1 <= seg1(1);

seg1\_0 <= seg1(0);

seg0\_6 <= seg0(6);

seg0\_5 <= seg0(5);

seg0\_4 <= seg0(4);

seg0\_3 <= seg0(3);

seg0\_2 <= seg0(2);

seg0\_1 <= seg0(1);

seg0\_0 <= seg0(0);

switch <= switch\_in;

process(clock)

variable temp3 : integer := 89;

variable temp2 : integer := 89;

variable temp1 : integer := 89;

variable temp0 : integer := 89;

variable scoreT : integer := 0;

variable score3 : integer := 0;

variable score2 : integer := 0;

variable score1 : integer := 0;

variable score0 : integer := 0;

variable timer : integer := 0;

variable trig3 : std\_logic := '0';

variable trig2 : std\_logic := '0';

variable trig1 : std\_logic := '0';

variable trig0 : std\_logic := '0';

variable start : std\_logic := '1';

variable play : std\_logic := '0';

variable again : std\_logic := '0';

variable super3 : std\_logic := '0';

variable super2 : std\_logic := '0';

variable super1 : std\_logic := '0';

variable super0 : std\_logic := '0';

BEGIN

if rising\_edge(clock) then

--start

if start = '1' then

if timer /= 115 then

timer := timer + 1;

elsif timer = 115 then

timer := 0;

end if;

if sWhack3 = '1' or sWhack2 = '1' or sWhack1 = '1' or sWhack0 = '1' then --newGame = '0' then

score3 := 0;

score2 := 0;

score1 := 0;

score0 := 0;

start := '0';

play := '1';

again := '0';

timer := 116;

end if;

end if;

--play

if play = '1' then

if timer /= 1116 then

timer := timer + 1;

elsif timer = 1116 then

start := '0';

play := '0';

again := '1';

--timer := 665;

end if;

end if;

--again

if again = '1' then

if timer /= 1192 then

timer := timer + 1;

elsif timer = 1192 then

timer := 1117;

end if;

if sWhack3 = '1' or sWhack2 = '1' or sWhack1 = '1' or sWhack0 = '1' then

temp3 := 89;

temp2 := 89;

temp1 := 89;

temp0 := 89;

trig3 := '0';

trig2 := '0';

trig1 := '0';

trig0 := '0';

score3 := 0;

score2 := 0;

score1 := 0;

score0 := 0;

timer := 0;

start := '1';

play := '0';

again := '0';

end if;

end if;

--Game Start

--if start = '0' and play = '1' and again = '0' then

--Display[3]

if whack3 = '1' and rand3 = '1' and trig3 = '0' and super3 = '0' then

temp3 := 0;

if score3 /= 99 then

score3 := score3 + 1;

elsif score3 = 99 then

score3 := 0;

end if;

trig3 := '1';

elsif sWhack3 = '1' and rand3 = '1' and trig3 = '0' and super3 = '1' then

temp3 := 0;

if score3 /= 99 then

score3 := score3 + 2;

elsif score3 = 99 then

score3 := 1;

end if;

trig3 := '1';

end if;

if trig3 = '1' then

if temp3 /= 90 then

temp3 := temp3 + 1;

elsif temp3 = 90 then

trig3 := '0';

temp3 := 89;

end if;

end if;

--end if;

--Display[2]

if whack2 = '1' and rand2 = '1' and trig2 = '0' and super2 = '0' then

temp2 := 0;

if score2 /= 99 then

score2 := score2 + 1;

elsif score2 = 99 then

score2 := 0;

end if;

trig2 := '1';

elsif sWhack2 = '1' and rand2 = '1' and trig2 = '0' and super2 = '1' then

temp2 := 0;

if score2 /= 99 then

score2 := score2 + 2;

elsif score2 = 99 then

score2 := 1;

end if;

trig2 := '1';

end if;

if trig2 = '1' then

if temp2 /= 90 then

temp2 := temp2 + 1;

elsif temp2 = 90 then

trig2 := '0';

temp2 := 89;

end if;

end if;

--Display[1]

if whack1 = '1' and rand1 = '1' and trig1 = '0' and super1 = '0' then

temp1 := 0;

if score1 /= 99 then

score1 := score1 + 1;

elsif score1 = 99 then

score1 := 0;

end if;

trig1 := '1';

elsif sWhack1 = '1' and rand1 = '1' and trig1 = '0' and super1 = '1' then

temp1 := 0;

if score1 /= 99 then

score1 := score1 + 2;

elsif score1 = 99 then

score1 := 1;

end if;

trig1 := '1';

end if;

if trig1 = '1' then

if temp1 /= 90 then

temp1 := temp1 + 1;

elsif temp1 = 90 then

trig1 := '0';

temp1 := 89;

end if;

end if;

--Display[0]

if whack0 = '1' and rand0 = '1' and trig0 = '0' and super0 = '0' then

temp0 := 0;

if score0 /= 99 then

score0 := score0 + 1;

elsif score0 = 99 then

score0 := 0;

end if;

trig0 := '1';

elsif sWhack0 = '1' and rand0 = '1' and trig0 = '0' and super0 = '1' then

temp0 := 0;

if score0 /= 99 then

score0 := score0 + 2;

elsif score0 = 99 then

score0 := 1;

end if;

trig0 := '1';

end if;

if trig0 = '1' then

if temp0 /= 90 then

temp0 := temp0 + 1;

elsif temp0 = 90 then

trig0 := '0';

temp0 := 89;

end if;

end if;

scoreT := score3 + score2 + score1 + score0;

case timer is

--start

--0 to 115

when 0 =>

seg5 <= "1111111";

seg4 <= "1111111";

seg3 <= "1111111";

seg2 <= "1111111";

seg1 <= "1111111";

seg0 <= "1111111";

when 1 to 5 =>

seg5 <= "1111111";

seg4 <= "1111111";

seg3 <= "1111111";

seg2 <= "1111111";

seg1 <= "1111111";

seg0 <= "0010010"; -- S

when 6 to 10 =>

seg5 <= "1111111";

seg4 <= "1111111";

seg3 <= "1111111";

seg2 <= "1111111";

seg1 <= "0010010"; -- S

seg0 <= "1100011"; -- u

when 11 to 15 =>

seg5 <= "1111111";

seg4 <= "1111111";

seg3 <= "1111111";

seg2 <= "0010010"; -- S

seg1 <= "1100011"; -- u

seg0 <= "0001100"; -- P

when 16 to 20 =>

seg5 <= "1111111";

seg4 <= "1111111";

seg3 <= "0010010"; -- S

seg2 <= "1100011"; -- u

seg1 <= "0001100"; -- P

seg0 <= "0000110"; -- E

when 21 to 25 =>

seg5 <= "1111111";

seg4 <= "0010010"; -- S

seg3 <= "1100011"; -- u

seg2 <= "0001100"; -- P

seg1 <= "0000110"; -- E

seg0 <= "0101111"; -- r

when 26 to 30 =>

seg5 <= "0010010"; -- S

seg4 <= "1100011"; -- u

seg3 <= "0001100"; -- P

seg2 <= "0000110"; -- E

seg1 <= "0101111"; -- r

seg0 <= "1111111";

when 31 to 35 =>

seg5 <= "1100011"; -- u

seg4 <= "0001100"; -- P

seg3 <= "0000110"; -- E

seg2 <= "0101111"; -- r

seg1 <= "1111111";

seg0 <= "0010010"; -- S

when 36 to 40 =>

seg5 <= "0001100"; -- P

seg4 <= "0000110"; -- E

seg3 <= "0101111"; -- r

seg2 <= "1111111";

seg1 <= "0010010"; -- S

seg0 <= "1000111"; -- L

when 41 to 45 =>

seg5 <= "0000110"; -- E

seg4 <= "0101111"; -- r

seg3 <= "1111111";

seg2 <= "0010010"; -- S

seg1 <= "1000111"; -- L

seg0 <= "1100011"; -- u

when 46 to 50 =>

seg5 <= "0101111"; -- r

seg4 <= "1111111";

seg3 <= "0010010"; -- S

seg2 <= "1000111"; -- L

seg1 <= "1100011"; -- u

seg0 <= "0010000"; -- g

when 51 to 55 =>

seg5 <= "1111111";

seg4 <= "0010010"; -- S

seg3 <= "1000111"; -- L

seg2 <= "1100011"; -- u

seg1 <= "0010000"; -- g

seg0 <= "0111111"; -- -

when 56 to 60 =>

seg5 <= "0010010"; -- S

seg4 <= "1000111"; -- L

seg3 <= "1100011"; -- u

seg2 <= "0010000"; -- g

seg1 <= "0111111"; -- -

seg0 <= "0001000"; -- A

when 61 to 65 =>

seg5 <= "1000111"; -- L

seg4 <= "1100011"; -- u

seg3 <= "0010000"; -- g

seg2 <= "0111111"; -- -

seg1 <= "0001000"; -- A

seg0 <= "0111111"; -- -

when 66 to 70 =>

seg5 <= "1100011"; -- u

seg4 <= "0010000"; -- g

seg3 <= "0111111"; -- -

seg2 <= "0001000"; -- A

seg1 <= "0111111"; -- -

seg0 <= "0000011"; -- b

when 71 to 75 =>

seg5 <= "0010000"; -- g

seg4 <= "0111111"; -- -

seg3 <= "0001000"; -- A

seg2 <= "0111111"; -- -

seg1 <= "0000011"; -- b

seg0 <= "1100011"; -- u

when 76 to 80 =>

seg5 <= "0111111"; -- -

seg4 <= "0001000"; -- A

seg3 <= "0111111"; -- -

seg2 <= "0000011"; -- b

seg1 <= "1100011"; -- u

seg0 <= "0010000"; -- g

when 81 to 85 =>

seg5 <= "0001000"; -- A

seg4 <= "0111111"; -- -

seg3 <= "0000011"; -- b

seg2 <= "1100011"; -- u

seg1 <= "0010000"; -- g

seg0 <= "1111111";

when 86 to 90 =>

seg5 <= "0111111"; -- -

seg4 <= "0000011"; -- b

seg3 <= "1100011"; -- u

seg2 <= "0010000"; -- g

seg1 <= "1111111";

seg0 <= "1111111";

when 91 to 95 =>

seg5 <= "0000011"; -- b

seg4 <= "1100011"; -- u

seg3 <= "0010000"; -- g

seg2 <= "1111111";

seg1 <= "1111111";

seg0 <= "1111111";

when 96 to 100 =>

seg5 <= "1100011"; -- u

seg4 <= "0010000"; -- g

seg3 <= "1111111";

seg2 <= "1111111";

seg1 <= "1111111";

seg0 <= "1111111";

when 101 to 105 =>

seg5 <= "0010000"; -- g

seg4 <= "1111111";

seg3 <= "1111111";

seg2 <= "1111111";

seg1 <= "1111111";

seg0 <= "1111111";

when 106 to 110 =>

seg5 <= "1111111";

seg4 <= "1111111";

seg3 <= "1111111";

seg2 <= "1111111";

seg1 <= "1111111";

seg0 <= "1111111";

when 111 to 115 =>

seg5 <= "1111111";

seg4 <= "1111111";

seg3 <= "1111111";

seg2 <= "1111111";

seg1 <= "1111111";

seg0 <= "1111111";

--game

--40 seconds

--116 to 1116

when 116 to 1116 =>

--Display[3]

case temp3 is

when 88 =>

if rand3 = '1' then

super3 := '1';

end if;

when 89 =>

if rand3 = '0' then

super3 := '0';

end if;

if rand3 = '1' and (whack3 = '0' or sWhack3 = '0') then

if super3 = '0' then

seg3 <= "0101011";

elsif super3 = '1' then

seg3 <= "1001000";

end if;

elsif rand3 = '0' then

seg3 <= "1111111";

elsif trig3 = '1' then

seg3 <= "1111111";

end if;

when 0 => seg3 <= "1111111";

when 1 =>

if super3 = '0' then

seg3 <= "0101011";

elsif super3 = '1' then

seg3 <= "1001000";

end if;

when 2 => seg3 <= "1111111";

when 3 =>

if super3 = '0' then

seg3 <= "0101011";

elsif super3 = '1' then

seg3 <= "1001000";

end if;

when 4 => seg3 <= "1111111";

when 5 =>

if super3 = '0' then

seg3 <= "0101011";

elsif super3 = '1' then

seg3 <= "1001000";

end if;

when 6 => seg3 <= "1111111";

when 7 =>

if super3 = '0' then

seg3 <= "0101011";

elsif super3 = '1' then

seg3 <= "1001000";

end if;

when 8 => seg3 <= "1111111";

when 9 =>

if super3 = '0' then

seg3 <= "0101011";

elsif super3 = '1' then

seg3 <= "1001000";

end if;

when 10 => seg3 <= "1111111";

when others =>

seg3 <= "1111111";

end case;

--Display[2]

case temp2 is

when 88 =>

if rand2 = '1' then

super2 := '1';

end if;

when 89 =>

if rand2 = '0' then

super2 := '0';

end if;

if rand2 = '1' and (whack2 = '0' or sWhack2 = '0') then

if super2 = '0' then

seg2 <= "0101011";

elsif super2 = '1' then

seg2 <= "1001000";

end if;

elsif rand2 = '0' then

seg2 <= "1111111";

elsif trig2 = '1' then

seg2 <= "1111111";

end if;

when 0 => seg2 <= "1111111";

when 1 =>

if super2 = '0' then

seg2 <= "0101011";

elsif super2 = '1' then

seg2 <= "1001000";

end if;

when 2 => seg2 <= "1111111";

when 3 =>

if super2 = '0' then

seg2 <= "0101011";

elsif super2 = '1' then

seg2 <= "1001000";

end if;

when 4 => seg2 <= "1111111";

when 5 =>

if super2 = '0' then

seg2 <= "0101011";

elsif super2 = '1' then

seg2 <= "1001000";

end if;

when 6 => seg2 <= "1111111";

when 7 =>

if super2 = '0' then

seg2 <= "0101011";

elsif super2 = '1' then

seg2 <= "1001000";

end if;

when 8 => seg2 <= "1111111";

when 9 =>

if super2 = '0' then

seg2 <= "0101011";

elsif super2 = '1' then

seg2 <= "1001000";

end if;

when 10 => seg2 <= "1111111";

when others =>

seg2 <= "1111111";

end case;

--Display[1]

case temp1 is

when 88 =>

if rand1 = '1' then

super1 := '1';

end if;

when 89 =>

if rand1 = '0' then

super1 := '0';

end if;

if rand1 = '1' and (whack1 = '0' or sWhack1 = '0') then

if super1 = '0' then

seg1 <= "0101011";

elsif super1 = '1' then

seg1 <= "1001000";

end if;

elsif rand1 = '0' then

seg1 <= "1111111";

elsif trig1 = '1' then

seg1 <= "1111111";

end if;

when 0 => seg1 <= "1111111";

when 1 =>

if super1 = '0' then

seg1 <= "0101011";

elsif super1 = '1' then

seg1 <= "1001000";

end if;

when 2 => seg1 <= "1111111";

when 3 =>

if super1 = '0' then

seg1 <= "0101011";

elsif super1 = '1' then

seg1 <= "1001000";

end if;

when 4 => seg1 <= "1111111";

when 5 =>

if super1 = '0' then

seg1 <= "0101011";

elsif super1 = '1' then

seg1 <= "1001000";

end if;

when 6 => seg1 <= "1111111";

when 7 =>

if super1 = '0' then

seg1 <= "0101011";

elsif super1 = '1' then

seg1 <= "1001000";

end if;

when 8 => seg1 <= "1111111";

when 9 =>

if super1 = '0' then

seg1 <= "0101011";

elsif super1 = '1' then

seg1 <= "1001000";

end if;

when 10 => seg1 <= "1111111";

when others =>

seg1 <= "1111111";

end case;

--Display[0]

case temp0 is

when 88 =>

if rand0 = '1' then

super0 := '1';

end if;

when 89 =>

if rand0 = '0' then

super0 := '0';

end if;

if rand0 = '1' and (whack0 = '0' or sWhack0 = '0') then

if super0 = '0' then

seg0 <= "0101011";

elsif super0 = '1' then

seg0 <= "1001000";

end if;

elsif rand0 = '0' then

seg0 <= "1111111";

elsif trig0 = '1' then

seg0 <= "1111111";

end if;

when 0 => seg0 <= "1111111";

when 1 =>

if super0 = '0' then

seg0 <= "0101011";

elsif super0 = '1' then

seg0 <= "1001000";

end if;

when 2 => seg0 <= "1111111";

when 3 =>

if super0 = '0' then

seg0 <= "0101011";

elsif super0 = '1' then

seg0 <= "1001000";

end if;

when 4 => seg0 <= "1111111";

when 5 =>

if super0 = '0' then

seg0 <= "0101011";

elsif super0 = '1' then

seg0 <= "1001000";

end if;

when 6 => seg0 <= "1111111";

when 7 =>

if super0 = '0' then

seg0 <= "0101011";

elsif super0 = '1' then

seg0 <= "1001000";

end if;

when 8 => seg0 <= "1111111";

when 9 =>

if super0 = '0' then

seg0 <= "0101011";

elsif super0 = '1' then

seg0 <= "1001000";

end if;

when 10 => seg0 <= "1111111";

when others =>

seg0 <= "1111111";

end case;

if switch = '0' then

case scoreT is

when 0 =>

seg5 <= "1000000";

seg4 <= "1000000"; --seg4 0

when 1 =>

seg5 <= "1000000";

seg4 <= "1111001"; --seg4 1

when 2 =>

seg5 <= "1000000";

seg4 <= "0100100"; --seg4 2

when 3 =>

seg5 <= "1000000";

seg4 <= "0110000"; --seg4 3

when 4 =>

seg5 <= "1000000";

seg4 <= "0011001"; --seg4 4

when 5 =>

seg5 <= "1000000";

seg4 <= "0010010"; --seg4 5

when 6 =>

seg5 <= "1000000";

seg4 <= "0000010"; --seg4 6

when 7 =>

seg5 <= "1000000";

seg4 <= "1111000"; --seg4 7

when 8 =>

seg5 <= "1000000";

seg4 <= "0000000"; --seg4 8

when 9 =>

seg5 <= "1000000";

seg4 <= "0010000"; --seg4 9

---------------------------------------

when 10 => --10

seg5 <= "1111001";

seg4 <= "1000000";

when 11 =>

seg5 <= "1111001";

seg4 <= "1111001";

when 12 =>

seg5 <= "1111001";

seg4 <= "0100100";

when 13 =>

seg5 <= "1111001";

seg4 <= "0110000";

when 14 =>

seg5 <= "1111001";

seg4 <= "0011001";

when 15 =>

seg5 <= "1111001";

seg4 <= "0010010";

when 16 =>

seg5 <= "1111001";

seg4 <= "0000010";

when 17 =>

seg5 <= "1111001";

seg4 <= "1111000";

when 18 =>

seg5 <= "1111001";

seg4 <= "0000000";

when 19 =>

seg5 <= "1111001";

seg4 <= "0010000";

when 20 => --20

seg5 <= "0100100";

seg4 <= "1000000";

when 21 =>

seg5 <= "0100100";

seg4 <= "1111001";

when 22 =>

seg5 <= "0100100";

seg4 <= "0100100";

when 23 =>

seg5 <= "0100100";

seg4 <= "0110000";

when 24 =>

seg5 <= "0100100";

seg4 <= "0011001";

when 25 =>

seg5 <= "0100100";

seg4 <= "0010010";

when 26 =>

seg5 <= "0100100";

seg4 <= "0000010";

when 27 =>

seg5 <= "0100100";

seg4 <= "1111000";

when 28 =>

seg5 <= "0100100";

seg4 <= "0000000";

when 29 =>

seg5 <= "0100100";

seg4 <= "0010000";

when 30 => --30

seg5 <= "0110000";

seg4 <= "1000000";

when 31 =>

seg5 <= "0110000";

seg4 <= "1111001";

when 32 =>

seg5 <= "0110000";

seg4 <= "0100100";

when 33 =>

seg5 <= "0110000";

seg4 <= "0110000";

when 34 =>

seg5 <= "0110000";

seg4 <= "0011001";

when 35 =>

seg5 <= "0110000";

seg4 <= "0010010";

when 36 =>

seg5 <= "0110000";

seg4 <= "0000010";

when 37 =>

seg5 <= "0110000";

seg4 <= "1111000";

when 38 =>

seg5 <= "0110000";

seg4 <= "0000000";

when 39 =>

seg5 <= "0110000";

seg4 <= "0010000";

when 40 => --40

seg5 <= "0011001";

seg4 <= "1000000";

when 41 =>

seg5 <= "0011001";

seg4 <= "1111001";

when 42 =>

seg5 <= "0011001";

seg4 <= "0100100";

when 43 =>

seg5 <= "0011001";

seg4 <= "0110000";

when 44 =>

seg5 <= "0011001";

seg4 <= "0011001";

when 45 =>

seg5 <= "0011001";

seg4 <= "0010010";

when 46 =>

seg5 <= "0011001";

seg4 <= "0000010";

when 47 =>

seg5 <= "0011001";

seg4 <= "1111000";

when 48 =>

seg5 <= "0011001";

seg4 <= "0000000";

when 49 =>

seg5 <= "0011001";

seg4 <= "0010000";

when 50 => --50

seg5 <= "0010010";

seg4 <= "1000000";

when 51 =>

seg5 <= "0010010";

seg4 <= "1111001";

when 52 =>

seg5 <= "0010010";

seg4 <= "0100100";

when 53 =>

seg5 <= "0010010";

seg4 <= "0110000";

when 54 =>

seg5 <= "0010010";

seg4 <= "0011001";

when 55 =>

seg5 <= "0010010";

seg4 <= "0010010";

when 56 =>

seg5 <= "0010010";

seg4 <= "0000010";

when 57 =>

seg5 <= "0010010";

seg4 <= "1111000";

when 58 =>

seg5 <= "0010010";

seg4 <= "0000000";

when 59 =>

seg5 <= "0010010";

seg4 <= "0010000";

when 60 => --60

seg5 <= "0000010";

seg4 <= "1000000";

when 61 =>

seg5 <= "0000010";

seg4 <= "1111001";

when 62 =>

seg5 <= "0000010";

seg4 <= "0100100";

when 63 =>

seg5 <= "0000010";

seg4 <= "0110000";

when 64 =>

seg5 <= "0000010";

seg4 <= "0011001";

when 65 =>

seg5 <= "0000010";

seg4 <= "0010010";

when 66 =>

seg5 <= "0000010";

seg4 <= "0000010";

when 67 =>

seg5 <= "0000010";

seg4 <= "1111000";

when 68 =>

seg5 <= "0000010";

seg4 <= "0000000";

when 69 =>

seg5 <= "0000010";

seg4 <= "0010000";

when 70 => --70

seg5 <= "1111000";

seg4 <= "1000000";

when 71 =>

seg5 <= "1111000";

seg4 <= "1111001";

when 72 =>

seg5 <= "1111000";

seg4 <= "0100100";

when 73 =>

seg5 <= "1111000";

seg4 <= "0110000";

when 74 =>

seg5 <= "1111000";

seg4 <= "0011001";

when 75 =>

seg5 <= "1111000";

seg4 <= "0010010";

when 76 =>

seg5 <= "1111000";

seg4 <= "0000010";

when 77 =>

seg5 <= "1111000";

seg4 <= "1111000";

when 78 =>

seg5 <= "1111000";

seg4 <= "0000000";

when 79 =>

seg5 <= "1111000";

seg4 <= "0010000";

when 80 => --80

seg5 <= "0000000";

seg4 <= "1000000";

when 81 =>

seg5 <= "0000000";

seg4 <= "1111001";

when 82 =>

seg5 <= "0000000";

seg4 <= "0100100";

when 83 =>

seg5 <= "0000000";

seg4 <= "0110000";

when 84 =>

seg5 <= "0000000";

seg4 <= "0011001";

when 85 =>

seg5 <= "0000000";

seg4 <= "0010010";

when 86 =>

seg5 <= "0000000";

seg4 <= "0000010";

when 87 =>

seg5 <= "0000000";

seg4 <= "1111000";

when 88 =>

seg5 <= "0000000";

seg4 <= "0000000";

when 89 =>

seg5 <= "0000000";

seg4 <= "0010000";

when 90 => --90

seg5 <= "0010000";

seg4 <= "1000000";

when 91 =>

seg5 <= "0010000";

seg4 <= "1111001";

when 92 =>

seg5 <= "0010000";

seg4 <= "0100100";

when 93 =>

seg5 <= "0010000";

seg4 <= "0110000";

when 94 =>

seg5 <= "0010000";

seg4 <= "0011001";

when 95 =>

seg5 <= "0010000";

seg4 <= "0010010";

when 96 =>

seg5 <= "0010000";

seg4 <= "0000010";

when 97 =>

seg5 <= "0010000";

seg4 <= "1111000";

when 98 =>

seg5 <= "0010000";

seg4 <= "0000000";

when 99 =>

seg5 <= "0010000";

seg4 <= "0010000";

when others =>

seg5 <= "1111111";

seg4 <= "1111111";

end case;

end if;

if switch = '1' then

if timer >= 116 and timer <= 140 then --40

seg5 <= "0011001";

seg4 <= "1000000";

end if;

if timer >= 141 and timer <= 165 then --39

seg5 <= "0110000";

seg4 <= "0010000";

end if;

if timer >= 166 and timer <= 190 then --38

seg5 <= "0110000";

seg4 <= "0000000";

end if;

if timer >= 191 and timer <= 215 then --37

seg5 <= "0110000";

seg4 <= "1111000";

end if;

if timer >= 216 and timer <= 240 then --36

seg5 <= "0110000";

seg4 <= "0000010";

end if;

if timer >= 241 and timer <= 265 then --35

seg5 <= "0110000";

seg4 <= "0010010";

end if;

if timer >= 266 and timer <= 290 then --34

seg5 <= "0110000";

seg4 <= "0011001";

end if;

if timer >= 291 and timer <= 315 then --33

seg5 <= "0110000";

seg4 <= "0110000";

end if;

if timer >= 316 and timer <= 340 then --32

seg5 <= "0110000";

seg4 <= "0100100";

end if;

if timer >= 341 and timer <= 365 then --31

seg5 <= "0110000";

seg4 <= "1111001";

end if;

if timer >= 366 and timer <= 390 then --30

seg5 <= "0110000";

seg4 <= "1000000";

end if;

if timer >= 391 and timer <= 415 then --29

seg5 <= "0100100";

seg4 <= "0010000";

end if;

if timer >= 416 and timer <= 440 then --28

seg5 <= "0100100";

seg4 <= "0000000";

end if;

if timer >= 441 and timer <= 465 then --27

seg5 <= "0100100";

seg4 <= "1111000";

end if;

if timer >= 466 and timer <= 490 then --26

seg5 <= "0100100";

seg4 <= "0000010";

end if;

if timer >= 491 and timer <= 515 then --25

seg5 <= "0100100";

seg4 <= "0010010";

end if;

if timer >= 516 and timer <= 540 then --24

seg5 <= "0100100";

seg4 <= "0011001";

end if;

if timer >= 541 and timer <= 565 then --23

seg5 <= "0100100";

seg4 <= "0110000";

end if;

if timer >= 566 and timer <= 590 then --22

seg5 <= "0100100";

seg4 <= "0100100";

end if;

if timer >= 591 and timer <= 615 then --21

seg5 <= "0100100";

seg4 <= "1111001";

end if;

if timer >= 616 and timer <= 640 then --20

seg5 <= "0100100";

seg4 <= "1000000";

end if;

if timer >= 641 and timer <= 665 then --19

seg5 <= "1111001";

seg4 <= "0010000";

end if;

if timer >= 666 and timer <= 690 then --18

seg5 <= "1111001";

seg4 <= "0000000";

end if;

if timer >= 691 and timer <= 715 then --17

seg5 <= "1111001";

seg4 <= "1111000";

end if;

if timer >= 716 and timer <= 740 then --16

seg5 <= "1111001";

seg4 <= "0000010";

end if;

if timer >= 741 and timer <= 765 then --15

seg5 <= "1111001";

seg4 <= "0010010";

end if;

if timer >= 766 and timer <= 790 then --14

seg5 <= "1111001";

seg4 <= "0011001";

end if;

if timer >= 791 and timer <= 815 then --13

seg5 <= "1111001";

seg4 <= "0110000";

end if;

if timer >= 816 and timer <= 840 then --12

seg5 <= "1111001";

seg4 <= "0100100";

end if;

if timer >= 841 and timer <= 865 then --11

seg5 <= "1111001";

seg4 <= "1111001";

end if;

if timer >= 866 and timer <= 890 then --10

seg5 <= "1111001";

seg4 <= "1000000";

end if;

if timer >= 891 and timer <= 915 then --09

seg5 <= "1000000";

seg4 <= "0010000";

end if;

if timer >= 916 and timer <= 940 then --08

seg5 <= "1000000";

seg4 <= "0000000";

end if;

if timer >= 941 and timer <= 965 then --07

seg5 <= "1000000";

seg4 <= "1111000";

end if;

if timer >= 966 and timer <= 990 then --06

seg5 <= "1000000";

seg4 <= "0000010";

end if;

if timer >= 991 and timer <= 1015 then --05

seg5 <= "1000000";

seg4 <= "0010010";

end if;

if timer >= 1016 and timer <= 1040 then --04

seg5 <= "1000000";

seg4 <= "0011001";

end if;

if timer >= 1041 and timer <= 1065 then --03

seg5 <= "1000000";

seg4 <= "0110000";

end if;

if timer >= 1066 and timer <= 1090 then --02

seg5 <= "1000000";

seg4 <= "0100100";

end if;

if timer >= 1091 and timer <= 1115 then --01

seg5 <= "1000000";

seg4 <= "1111001";

end if;

if timer = 1116 then --00

seg5 <= "1000000";

seg4 <= "1000000";

end if;

end if;

--again

--1117 to 1192

when 1117 to 1120 =>

seg5 <= "1111111";

seg4 <= "1111111";

seg3 <= "1111111";

seg2 <= "1111111";

seg1 <= "1111111";

seg0 <= "1111111";

when 1121 to 1125 =>

seg5 <= "1111111";

seg4 <= "1111111";

seg3 <= "1111111";

seg2 <= "1111111";

seg1 <= "1111111";

seg0 <= "0010000"; -- g

when 1126 to 1130 =>

seg5 <= "1111111";

seg4 <= "1111111";

seg3 <= "1111111";

seg2 <= "1111111";

seg1 <= "0010000"; -- g

seg0 <= "0100011"; -- o

when 1131 to 1135 =>

seg5 <= "1111111";

seg4 <= "1111111";

seg3 <= "1111111";

seg2 <= "0010000"; -- g

seg1 <= "0100011"; -- o

seg0 <= "0100011"; -- o

when 1136 to 1140 =>

seg5 <= "1111111";

seg4 <= "1111111";

seg3 <= "0010000"; -- g

seg2 <= "0100011"; -- o

seg1 <= "0100011"; -- o

seg0 <= "0100001"; -- d

when 1141 to 1145 =>

seg5 <= "1111111";

seg4 <= "0010000"; -- g

seg3 <= "0100011"; -- o

seg2 <= "0100011"; -- o

seg1 <= "0100001"; -- d

seg0 <= "1111111";

when 1146 to 1150 =>

seg5 <= "0010000"; -- g

seg4 <= "0100011"; -- o

seg3 <= "0100011"; -- o

seg2 <= "0100001"; -- d

seg1 <= "1111111";

seg0 <= "1100001"; -- J

when 1151 to 1155 =>

seg5 <= "0100011"; -- o

seg4 <= "0100011"; -- o

seg3 <= "0100001"; -- d

seg2 <= "1111111";

seg1 <= "1100001"; -- J

seg0 <= "0100011"; -- o

when 1156 to 1160 =>

seg5 <= "0100011"; -- o

seg4 <= "0100001"; -- d

seg3 <= "1111111";

seg2 <= "1100001"; -- J

seg1 <= "0100011"; -- o

seg0 <= "0000011"; -- b

when 1161 to 1165 =>

seg5 <= "0100001"; -- d

seg4 <= "1111111";

seg3 <= "1100001"; -- J

seg2 <= "0100011"; -- o

seg1 <= "0000011"; -- b

seg0 <= "1111111";

when 1166 to 1170 =>

seg5 <= "1111111";

seg4 <= "1100001"; -- J

seg3 <= "0100011"; -- o

seg2 <= "0000011"; -- b

seg1 <= "1111111";

seg0 <= "1111111";

when 1171 to 1175 =>

seg5 <= "1100001"; -- J

seg4 <= "0100011"; -- o

seg3 <= "0000011"; -- b

seg2 <= "1111111";

seg1 <= "1111111";

seg0 <= "1111111";

when 1176 to 1180 =>

seg5 <= "0100011"; -- o

seg4 <= "0000011"; -- b

seg3 <= "1111111";

seg2 <= "1111111";

seg1 <= "1111111";

seg0 <= "1111111";

when 1181 to 1185 =>

seg5 <= "0000011"; -- b

seg4 <= "1111111";

seg3 <= "1111111";

seg2 <= "1111111";

seg1 <= "1111111";

seg0 <= "1111111";

when others =>

seg5 <= "1111111";

seg4 <= "1111111";

seg3 <= "1111111";

seg2 <= "1111111";

seg1 <= "1111111";

seg0 <= "1111111";

end case;

end if;

--end if;

end process;

end rtl;

**To sum up:**

**----**Clock receiving a 25-hertz signal, so it spells out when each of the different states start, play, and again begin (according to clock signal ticks and the timer) as well as sets the preliminaries for all the states. When they occur 0-115, 115-1117, and 1117-1185. The button to go from state to state.

**----**This is where the score increments, and the hit-buffer timer is triggered. 2 states hit-buffer state and super state which obviously depends on whether a regular bug or super bug is present. In buffer state the buffer increments to 90, toggles the buffer state (either one depending on whats present) and finally gets reset to 89.

**----**This is part of the start state, where at the beginning there is a scrolling message that displays “Super Slug a Bug”, this is done by repeating the message throughout different screens from 0-115 ticks. It is reset to 0, if the start button isn’t pressed. The other scrolling message works the same way.

----This is the game mechanics. Creates the bugs and super bugs. Super bugs are created (less frequently) due to the fact that if a bug is hit and the signal does not disappear in the time span of approx. 2 and a half seconds a super bug will come in its place), however most bugs are gone before then, so that is how the regular bugs appear more frequently. The timer and the score are also included in here. The score has 100 possibilities (0-99) and the timer counts down from 40 seconds. Both are displayed on the same 2 LED’s which can be toggled by a switch on the board.

----This is part of the again state where it displays a scrolling message using the same strategy as before. It displays the message “Good Job”, the message will keep going because it will keep going back to 1117, until the push button is pressed to go back to the start state, and when pressed again will start the game.

**5. Operation**

This project is created in VHDL code, besides for the 25 hertz and 1 hertz signals which are created in Verilog. The project was divided into a couple blocks. I have mentioned them before as well as what they do, but I will repeat it again for this operations section. We have the 4-bit randomizer block which contains 4 blocks inside of that, which essentially are blocks representing the VHDL code for pseudo random bit generators. As mentioned earlier the difference is here:

feedback <= not(lfsr(2) xor lfsr(4));

feedback <= not(lfsr(3) xor lfsr(6));

feedback <= not(lfsr(1) xor lfsr(6));

feedback <= not(lfsr(0) xor lfsr(4));

This creates different output combinations, so we have 4 different 1-bit random numbers creating our randomizer for bug output.

There is also the hammer block which is what causes the LEDs to light up as you hold down the push buttons. Also, now the signal is sent over to the game block, so it can tell which button just got ‘whacked’ whack0, whack1, whack2, whack3, or if it was a super bug sWhack0, sWhack1 and so on. This differentiation also helps so that you only get 1 point per bug on the regular bugs even if you hold down the button, and you only get points for super whacks if you hold down the button until all the LED’s light up.

The last part of the design is the game block. The game block takes in everything as inputs, the 4 random bits, the hammer whacks both regular and super, and the 25-hertz signal. It also is connected to a switch which is used to either display the time remaining in the game or your score depending on if the switch is up or down. As mentioned earlier, there are three main states inside the game block: start, play, and again. Start displays the scrolling welcome message “Super-Slug-A-Bug” and waits for the user to press the start game button which then changes states to the play state which contains most of the game elements. The timer being displayed (40 to 0) the score (1 point for regular bugs and 2 for super bugs), as well as creating the regular bugs and super bugs, through the hit-buffer timer which increments the score, and creates the super bugs depending on how long it takes for the signal to go away. This, was repeated for each display and push button combination. Lastly, after the timer runs out it switches states to the again state which displays the scrolling end message “Good Job” and waits for the user to hit the new game button which will bring the user back to start screen. Some parts of the code were color coded and explained generally what each piece does:

**----**Clock receiving a 25-hertz signal, so it spells out when each of the different states start, play, and again begin (according to clock signal ticks and the timer) as well as sets the preliminaries for all the states. When they occur 0-115, 115-1117, and 1117-1185. The button to go from state to state.

**----**This is where the score increments, and the hit-buffer timer is triggered. 2 states hit-buffer state and super state which obviously depends on whether a regular bug or super bug is present. In buffer state the buffer increments to 90, toggles the buffer state (either one depending on whats present) and finally gets reset to 89.

**----**This is part of the start state, where at the beginning there is a scrolling message that displays “Super Slug a Bug”, this is done by repeating the message throughout different screens from 0-115 ticks. It is reset to 0, if the start button isn’t pressed. The other scrolling message works the same way.

----This is the game mechanics. Creates the bugs and super bugs. Super bugs are created (less frequently) due to the fact that if a bug is hit and the signal does not disappear in the time span of approx. 2 and a half seconds a super bug will come in its place), however most bugs are gone before then, so that is how the regular bugs appear more frequently. The timer and the score are also included in here. The score has 100 possibilities (0-99) and the timer counts down from 40 seconds. Both are displayed on the same 2 LED’s which can be toggled by a switch on the board.

----This is part of the again state where it displays a scrolling message using the same strategy as before. It displays the message “Good Job”, the message will keep going because it will keep going back to 1117, until the push button is pressed to go back to the start state, and when pressed again will start the game.

**5. Conclusions**

There were many problems encountered throughout the course of this project. First issue was with the randomizer and how to properly implement it so that we did not get a repetitive pattern. Through trial and error, we landed on using a pseudo random bit generator 4 times (corresponding to the 4 push buttons), and the output is ‘wired’ differently on each one so that is how we get a random pattern. The hammer module went pretty smoothly, although we had to write out the code for each possibility. The main issues came in the game module, which makes sense because it contains most of the project. There were issues with every part of this module. Figuring out how to differentiate between regular bugs and super bugs, finding the right timing between the ticks to move from state to state, and especially the output from state to state. Getting the design to only add to the score once and not multiple times for each bug, especially because they don’t disappear right away led us to trigger a counter which goes to 0 and makes the bug shake too, which looks cool on the board because it mimics some video games. This also led to the addition of super bugs at a less frequent pace depending on if the signal was completely gone or not when it reached 0, and if not, a super bug would be made. It was also difficult writing out all 99 possibilities for the score without making a mistake along the line. Overall, it was very difficult connecting all of these different elements together, and it required a lot of if statements and a lot of repetitive statements, for example writing out the same code for each display: display [0], display [1], and so on. Although the project was extremely ambitious and difficult, it was super awesome to see the result. Using VHDL gave us great control over the different states and possibilities which allowed the Super-Slug-A-Bug game to be created.