**DRAG RACE SIMULATOR**

DSD PROJECT

STUDENT: Danci Patricia Ioana

PROJECT SUPERVISOR: Fleger Dan

**TABLE OF CONTENTS**

1. ***Specifications……………………………………………...3***
2. ***Description and Theoretical Approach…………………..4***
3. ***Code Breakdown…………………………………….........8***
4. ***Conclusions……………………………………………....10***
5. ***Further Development…………………………………….11***
6. ***Code………………………………………………………13***
7. **SPECIFICATIONS**

The project involves creating an SSD animation for a drag race using VHDL and Vivado. The animation displays two cars sequentially on eight SSDs. The animation starts with the first car, progresses to the second car when the middle button is pressed, and stops when the button is pressed again. A switch controls the clutch, and the up and down buttons adjust the speed. A counter keeps track of the speed, and at the end, the speeds of both cars are compared to determine the winner. The speed values are stored in memory, and LEDs indicate the winner.

This project focuses on developing an animation on SSDs using Nexys 4 FPGA board and VHDL language. Key elements utilized are:

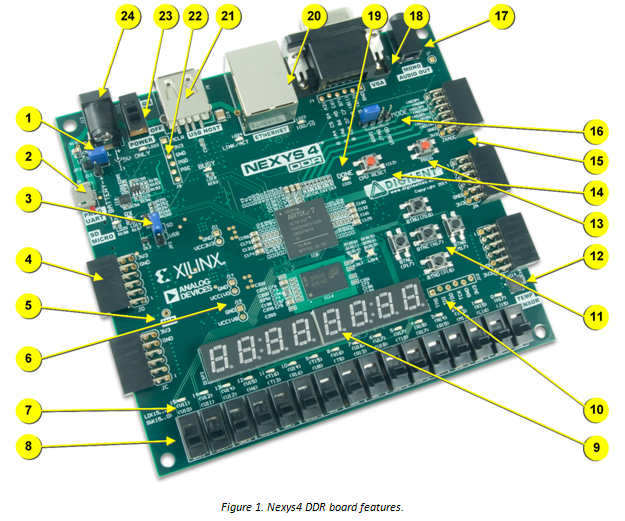
* Nexys 4 FPGA board
* VHDL language
* 7-Segment Display
* Frequency Divider
* Buttons
* Switch(1)
* LEDs

1. **DESCRIPTION AND THEORETICAL APPROACH**

**VHDL** is a hardware description language used for designing digital systems. It allows designers to describe the structure, behavior, and timing of digital circuits, while representing a standardized and concise way to model and simulate complex digital systems, enabling efficient and reliable design methodologies. VHDL has found applications in various domains, including ASIC and FPGA design, system-on-chip (SoC) design, and digital signal processing.

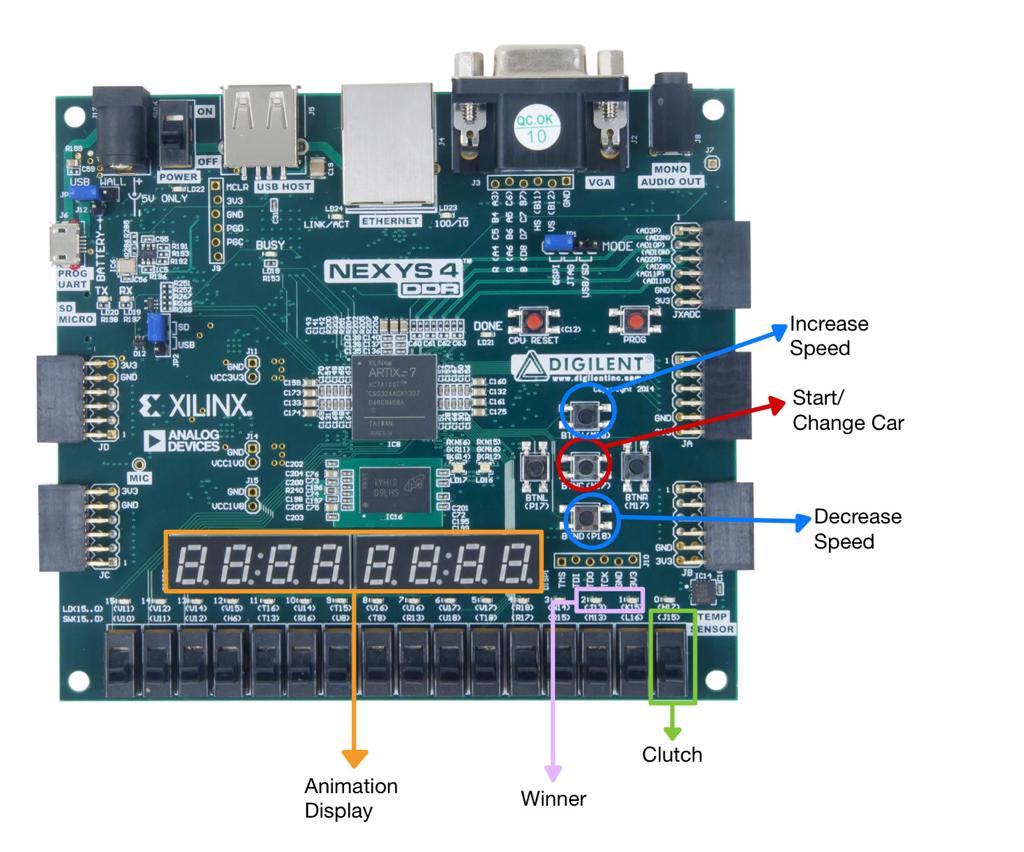
The **Nexys 4** is a field-programmable gate array (FPGA) development board designed by Digilent. It features a Xilinx Spartan-6 FPGA, offering a versatile platform for digital circuit prototyping and development.

The Nexys 4 FPGA board features eight **seven-segment displays (SSDs)**. Each SSD consists of seven segments (a, b, c, d, e, f, g) that can be individually controlled to display different characters or patterns. These displays are commonly used to show numeric values and alphanumeric characters. The SSDs on the Nexys 4 are usually multiplexed, meaning only one SSD is active at a time while rapidly switching between them to create the illusion of multiple active displays. By appropriately controlling the segments of each SSD, complex patterns, digits, or letters can be displayed. In the context of the project, the SSDs are used to animate the cars in the drag race, displaying a sequential animation on each SSD to represent the movement and speed of the cars.



**The system is implemented using the following components:**

* **11** -> start/change car/stop button, up button, down button
* **9** -> SSDs for the Animation
* **8** -> one slide switch for the clutch
* **7** -> two leds for determining the winner
* **23** -> power switch
* **24** -> power jack



*Up Button:* The up button is used to increase the speed of the cars in the drag race animation. When pressed, it triggers a signal that increments the speed value, allowing the cars to accelerate.

*Middle Button:* The middle button serves a dual purpose in the project. Firstly, it functions as the start/stop button for the animation. When pressed initially, it starts the animation, initiating the movement of the first car. Pressing it again stops the animation. Secondly, it acts as the button to switch between the first and second car's animation. Each press of the middle button transitions the display between the two cars.

*Down Button:* The down button is used to decrease the speed of the cars in the drag race animation. When pressed, it triggers a signal that decrements the speed value, allowing the cars to slow down.

*Switch:* The switch is used as a clutch control. When the switch is in the "on" position, it enables the ability to adjust the speed using the up and down buttons. By toggling the switch, the user can engage or disengage the clutch functionality.

*LEDs:* The LEDs are used to indicate the winner of the drag race. Depending on the outcome of the race and the winner determination logic, the corresponding LED will be illuminated to signify the victorious car.

*These peripherals provide user interaction and feedback, allowing control over the animation and visualization of the race progress and result.*

1. **CODE BREAKDOWN**

The code features 6 main VHDL files (components): Nexys4.vhd, CarAnimation.vhd, FrequencyDiv.vhd, Memory.vhd, WinnerDet.vhd, SpeedCounter.vhd.

*Nexys4.vhd*

The Nexys4.vhd is the top-level VHDL file, which contains the FPGS’s peripherals, such as switches, buttons, LEDs and seven-segment-displays, in order to connect them to the internal logic defined in the other files.

*FSM.vhd*

The FSM.vhd file contains the main logic of the project. The FSM component is responsible for controlling the animation of the cars on the SSDs, while establishing the playing mode. It sequentially activates segments on each SSD to represent the movement and speed of the cars. It utilizes signals such as start, stop, switch, up, and down to control the animation and adjust the speed.

*FreqDivider.vhd*

The FreqDivider component divides the input clock signal to generate a slower clock frequency. It uses intensity as selection to control the timing and speed of the animation. By adjusting the division ratio, the animation can be slowed down or sped up.

*Timer.vhd*

The Timer component keeps track of the “speed” of the cars. The current speed is stored in a variable and can be used for comparison or display purposes.

*SpeedController.vhd*

*The SpeedController component modifies the speed of the car. The FreqDivider component is based on this component, which utilizes signals such as up and down to modify the speed.*

*SSD.vhd*

The SSD component aims to display the animation, regarding the speed denoted by the FreqDivider component. It utilizes the eight seven-segment-displays, lighting up the middle cathode (‘g’) for each anode as the car moves forward.

*Debouncer.vhd*

The Debouncer component is used for eliminating signal noise and ensuring reliable button input for the buttons. It is used three times in the top level to ensure that all three buttons (up, down, start) work properly.

1. **CONCLUSIONS**

In conclusion, through the successful integration and interconnection of the CarAnimation, FrequencyDivider, SpeedCounter, Memory, and Winner Determination components in the top-level design, a cohesive and functional drag race simulation system was realized. The project demonstrated the versatility and power of VHDL in designing complex digital systems, highlighting its ability to control and manipulate hardware to achieve desired functionalities. This project effectively implemented a drag race animation on the Nexys 4 FPGA board using VHDL and Vivado. The animation featured two cars displayed sequentially on eight seven-segment displays (SSDs), accurately representing their movement and speed. The animation was controlled by user inputs, including start, stop, and speed adjustment buttons, along with a clutch switch. The speed of each car was accurately tracked using a counter and stored in memory for later comparison. At the end of the race, the Winner Determination component accurately determined the winner based on the recorded speeds.

1. **FURTHER DEVELOPMENT**

* Implement simultaneous animation of both cars: Enhance the project by modifying the CarAnimation component to animate both cars at the same time. This would involve using separate counters for each car and updating the animation logic to handle simultaneous movement.
* Individual speed control for each car: Add separate up and down buttons for each car, allowing two players to control the speeds of their respective cars independently. This would enhance the interactivity and competitiveness of the drag race.
* Fault detection and error handling: Implement fault detection mechanisms to identify and handle potential errors or unexpected behavior. For example, you could incorporate checks for invalid input combinations, out-of-range values, or synchronization issues. Displaying error messages or implementing recovery strategies would enhance the overall reliability and robustness of the project.
* Track lap times: Introduce lap time tracking functionality to measure and display the time taken by each car to complete a lap. This would add an additional competitive element to the race and allow for further analysis and comparison between the cars' performance.
* Integrate additional visual indicators: Include visual indicators, such as progress bars or graphical representations of speed, to provide more comprehensive and intuitive feedback to the users. This would enhance the visual appeal and understanding of the drag race dynamics.
* Expand the memory capacity: Increase the memory capacity to store additional race data, such as lap times, previous race results, or player profiles. This would enable the project to support more extensive data logging and analysis capabilities.
* Implement multiple race modes: Develop different race modes, such as time trials, head-to-head races, or championship modes, to provide varied gameplay experiences and challenges.

1. **CODE**

**Nexys4.vhd**

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

-- Company:

-- Engineer:

--

-- Create Date: 06/02/2023 01:53:59 PM

-- Design Name:

-- Module Name: Nexys4 - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

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

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_ARITH.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity Nexys4 is

port (

--INPUT PORTS

CLK100MHZ : in std\_logic;

BTNC : in std\_logic;

BTNU : in std\_logic;

BTND : in std\_logic;

SWITCH : in std\_logic;

--OUTPUT PORTS

CAT : out std\_logic\_vector(6 downto 0);

AN : out std\_logic\_vector(7 downto 0);

LED : out std\_logic\_vector(1 downto 0)

);

end Nexys4;

architecture Behavioral of Nexys4 is

signal up: std\_logic;

signal down: std\_logic;

signal start: std\_logic;

signal clk\_speed : std\_logic;

signal speed : std\_logic\_vector(2 downto 0);

begin

Debouncer\_BTNC: entity WORK.Debouncer

port map(

btn => BTNC,

clk => CLK100MHZ,

en => start

);

Debouncer\_BTND: entity WORK.Debouncer

port map(

btn => BTND,

clk => CLK100MHZ,

en => down

);

Debouncer\_BTNU: entity WORK.Debouncer

port map(

btn => BTNU,

clk => CLK100MHZ,

en => up

);

FSM: entity WORK.FSM

port map(

clk => CLK100MHZ,

btnU => up,

btnD => down,

btnC => start,

switch => SWITCH,

cat => CAT,

an => AN,

led => LED

)

end Behavioral;

**Debouncer.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_ARITH.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity Debouncer is

port (

signal btn : in std\_logic;

signal clk : in std\_logic;

signal en : out std\_logic

);

end Debouncer;

architecture Behavioral of Debouncer is

signal count\_int1 : std\_logic\_vector(31 downto 0) := x"00000000";

signal Q1 : std\_logic;

signal Q2 : std\_logic;

signal Q3 : std\_logic;

begin

en <= Q2 and (not Q3);

process (clk)

begin

if clk = '1' and clk'event then

count\_int1 <= count\_int1 + 1;

end if;

end process;

process (clk)

begin

if clk'event and clk = '1' then

if count\_int1(15 downto 0) = "1111111111111111" then

Q1 <= btn;

end if;

end if;

end process;

process (clk)

begin

if clk'event and clk = '1' then

Q2 <= Q1;

Q3 <= Q2;

end if;

end process;

end Behavioral;

**FreqDivider.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity FreqDivider is

port (

signal clk : in std\_logic;

signal intensity: in std\_logic\_vector(2 downto 0); --car speed selection

signal new\_clk : out std\_logic

);

end FreqDivider;

architecture Behavioral of FreqDivider is

signal counter : integer := 0;

signal time\_clk: integer := 99999999;

signal clk\_div : std\_logic;

begin

time\_clk <= 99999999 when intensity="000" else -- 100 000 000 1s --0 49999999

74999999 when intensity="001" else -- 75 000 000 0.75s --1 24999999

49999999 when intensity="010" else -- 50 000 000 0.5s --2

24999999 when intensity="011" else -- 25 000 000 0.25s --3

9999999 when intensity="100" else -- 10 000 000 0.1s --4

7499999 when intensity="101" else -- 7 500 000 0.075s --5

499999 when intensity="110" else -- 5 000 000 0.05s --6

249999; -- 2 500 000 0.025s --7

process (clk, time\_clk)

begin

if clk = '1' and clk'event then

if counter < time\_clk then

counter <= counter + 1;

else

counter <= 0;

clk\_div <= not(clk\_div);

end if;

end if;

end process;

new\_clk <= clk\_div;

end Behavioral;

**FSM.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_ARITH.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity FSM is

port (

--Input:

clk : in std\_logic;

btnC : in std\_logic;

btnU : in std\_logic;

btnD : in std\_logic;

switch : in std\_logic;

--Output:

cat : out std\_logic\_vector(6 downto 0);

an : out std\_logic\_vector(7 downto 0);

led : out std\_logic\_vector(1 downto 0)

);

end FSM;

architecture Behavioral of FSM is

type state is (idle, start, player\_1, race1, player\_2, race2, winner);

signal current\_state : state := idle;

signal next\_state : state := idle;

signal timerP1 : std\_logic\_vector(15 downto 0) := (others => '0');

signal timerP2 : std\_logic\_vector(15 downto 0) := (others => '0');

signal enableP1 : std\_logic := '0';

signal enableP2 : std\_logic := '0';

signal speedP1 : std\_logic\_vector(2 downto 0) := (others => '0');

signal speedP2 : std\_logic\_vector(2 downto 0) := (others => '0');

signal speedClkP1 : std\_logic := '0';

signal speedClkP2 : std\_logic := '0';

signal finishP1 : std\_logic := '0';

signal finishP2 : std\_logic := '0';

begin

Timer\_Player1 : entity WORK.Timer

port map(

clk => clk,

enable => enableP1,

timer => timerP1

);

Timer\_Player2 : entity WORK.Timer

port map(

clk => clk,

enable => enableP2,

timer => timerP2

);

SpeedControllerP1 : entity WORK.SpeedController

port map(

clk => clk,

clutch => switch,

up => btnU,

down => btnD,

enable => enableP1,

car\_speed => speedP1

);

SpeedControllerP2 : entity WORK.SpeedController

port map(

clk => clk,

clutch => switch,

up => btnU,

down => btnD,

enable => enableP2,

car\_speed => speedP2

);

FreqDividerP1 : entity WORK.FreqDivider

port map(

clk => clk,

intensity => speedP1, --intensity

new\_clk => speedClkP1 --new\_clk

);

FreqDividerP2 : entity WORK.FreqDivider

port map(

clk => clk,

intensity => speedP2,

new\_clk => speedClkP2

);

SSDPlayer1 : entity WORK.SSD

port map(

clk => speedClkP1, --new\_clk

enable => enableP1,

en => finishP1,

cat => cat,

an => an

);

SSDPlayer2 : entity WORK.SSD

port map(

clk => speedClkP2,

enable => enableP2,

en => finishP2,

cat => cat,

an => an

);

process (clk, btnC)

begin

case current\_state is

when idle =>

if btnC = '1' then

next\_state <= player\_1;

else

next\_state <= idle;

end if;

when player\_1 =>

if btnC = '1' then

next\_state <= race1;

else

next\_state <= player\_1;

end if;

when race1 =>

if (finishP1 = '1') then --when the race is over

enableP1 <= '0';

next\_state <= player\_2;

else

enableP1 <= '1';

next\_state <= race1;

end if;

when player\_2 =>

if btnC = '1' then

next\_state <= race2;

else

next\_state <= player\_2;

end if;

when race2 =>

if (finishP2 = '1') then

enableP2 <= '0';

next\_state <= winner;

else

enableP2 <= '1';

next\_state <= race2;

end if;

when others =>

if btnC = '1' then

next\_state <= idle;

else

if timerP1 > timerP2 then

led(0) <= '1';

elsif timerP2 > timerP1 then

led(1) <= '1';

else

led <= "11";

end if;

next\_state <= winner;

end if;

end case;

end process;

end Behavioral;

**SpeedController.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_ARITH.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity SpeedController is

port (

-- Inputs

clk : in std\_logic;

clutch : in std\_logic;

up : in std\_logic;

down : in std\_logic;

enable : in std\_logic;

-- Outputs

car\_speed : out std\_logic\_vector(2 downto 0)

);

end entity SpeedController;

architecture Behavioral of SpeedController is

signal car\_speed\_reg : std\_logic\_vector(2 downto 0) := (others => '0');

begin

process (clk, enable)

begin

if enable = '1' then

if rising\_edge(clk) then

-- Car speed counting

if clutch = '1' then

if up = '1' then

car\_speed\_reg <= std\_logic\_vector(unsigned(car\_speed\_reg));

car\_speed\_reg <= car\_speed\_reg + 1;

elsif down = '1' then

car\_speed\_reg <= std\_logic\_vector(unsigned(car\_speed\_reg));

car\_speed\_reg <= car\_speed\_reg - 1;

end if;

end if;

end if;

end if;

end process;

-- Output signals assignment

car\_speed <= car\_speed\_reg;

end architecture Behavioral;

**SSD.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_ARITH.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity SSD is

port (

--Input

clk : in std\_logic;

enable : in std\_logic;

--Output

en : out std\_logic;

cat : out std\_logic\_vector(6 downto 0);

an : out std\_logic\_vector(7 downto 0)

);

end SSD;

architecture Behavioral of SSD is

signal selection: std\_logic\_vector(2 downto 0) := (others => '0');

begin

cat <= "0111111";

process (clk, enable)

begin

if (enable = '1') then

if (rising\_edge(clk)) then

if (selection< "111") then

selection<= selection + 1;

else

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

end if;

end if;

case selection is

when "000" => an <= "01111111";

en <= '0';

when "001" => an <= "10111111";

en <= '0';

when "010" => an <= "11011111";

en <= '0';

when "011" => an <= "11101111";

en <= '0';

when "100" => an <= "11110111";

en <= '0';

when "101" => an <= "11111011";

en <= '0';

when "110" => an <= "11111101";

en <= '0';

when others => an <= "11111110";

en <= '1';

end case;

end if;

end process;

end Behavioral;

**Timer.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_ARITH.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity Timer is

port (

clk : in std\_logic;

enable : in std\_logic;

timer : out std\_logic\_vector(15 downto 0)

);

end Timer;

architecture Behavioral of Timer is

signal clk\_div : std\_logic := '0';

signal counter : integer := 0;

signal cnt: std\_logic\_vector(15 downto 0) := (others => '0');

begin

process (clk)

begin

if clk = '1' and clk'event then

if counter < 24999999 then

counter <= counter + 1;

else

counter <= 0;

clk\_div <= not(clk\_div);

end if;

end if;

end process;

process (clk\_div, enable)

begin

if (enable = '1') then

if (rising\_edge(clk\_div)) then

if (cnt < "1111111111111111") then

cnt <= cnt + 1;

else

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

end if;

end if;

end if;

end process;

timer <= cnt;

end Behavioral;