ASIA PACIFIC UNIVERSITY TECHNOLOGY & INNOVATION

EE057-4-2 VLSI

ASSIGNMENT

TITLE	VLSI ASSIGNMENT
NAME & STUDENT ID	PEDRO FABIAN OWONO ONDO MANGUE(TP063251)
INTAKE	APD3F2308CE
LECTURER	DR SOON KIAN LUN
DATE	11/12/2023

Contents

LIST OF FIGURES

INTRODUCTION

Digital VLSI systems have played a key role in this huge move towards small but powerful computing. The core of these developments relies on the digital counter. It is one of the fundamental components that organizes serial logic and timing processes required by multiple digital applications. The aim of this project is to study a highly efficient digital counter which can be used in VLSI systems.

This work presents a detailed description of the design process from initial idea to simulation for a VHDL-based digital counter. The intricacy of my approach involves traversing the details of the design of the counter and its architecture, its functional aspects, and the rationale for choosing each element. I am evaluating the counter's capabilities using simulation tests to make the process more transparent.

The counter is essential to any modern computer VLSI. It is a basis for sophisticated digital circuits with precise timing or event counting. Through this, I hope to provide a road map that gives details on the counter and how it integrates with the VSLSI platform.

This document presents an in-depth discussion of the counter design, including the VLSI system requirements it meets, the VHDL coding specificities, simulation procedures, evaluation of data interpretation, and a critical review of results. Therefore, my effort is to depict a story that is both an educational resource and can be considered a benchmark for subsequent VLSI counter-design of digital circuits.

OBJECTIVE

The main objective of this project is to create a VHDL-based digital counter that is highly efficient and aligns with the performance requirements of modern VLSI systems. I will delve into the design process, architecture, and functionality of the counter, providing a detailed analysis of each aspect. To ensure its effectiveness, I will conduct simulation tests to evaluate its capabilities and integration with larger VLSI systems. Additionally, we will focus on practical applications by designing interfaces with output indicators like seven-segment displays and LEDs. Throughout the design process, I will optimize the counter for power consumption and reliability without compromising accuracy.

LITERATURE REVIEW

The most used recent advances in VLSI (Very Large-Scale Integration) circuit design have led to different methods for designing digital counters each of which offers its advantages and disadvantages. The literature review shows various methods meant to satisfy continuously increasing requirements for speed, size, and power consumption in VLSI counter circuitry.

One innovative design in the field is the sleepy CMOS-sleepy stack (SC-SS) technique that boasts a fast speed and low consumption of energy. Sharma, Sohal, and Kaur (2019) compared SC-SST with a total of eight existing methods, including SKT and FST, showing better results in terms of performance and timing.

The use of LFSR (Linea Feedback Shift Register) counters for big-array applications constitutes the other distinguished approach. Specifically, Morrison et al., 2019, discussed a multistage counter utilizing an LFSR. The authors reported lower power dissipation compared to conventional binary counters. The fact that the decoding logic merely grows logarithmically instead of exponentially as a function of the number of bits shows their superior design for system-on-chip designs.

Recent research has also addressed security in VLSI systems focused on Trojan Horses in PCBs. Pearce et al., (2022) built a detection framework for hardware Trojans by employing multimodal side channels. This work however does not relate directly to the design of counters. It emphasizes security issues in modern VLSI that can affect the integrity of digital counters and other components.

The invention of VLSI counters has greatly enhanced the performance as well as the security in this area. Thus, future studies will involve techniques such as SC-SS or multistage LFSR in their implementations, while simultaneously considering issues like power conservation, efficient area usage, and security measures.

In the last two decades, researchers have come up with several schemes for checking digital systems and their circuits. With increasing levels of complexity in digital logic circuits, the importance of testing these systems has never been greater. Testing for design defects, including manufacturing defects and wear-out defect type is performed in the design phase. For example, a designer’s error, an inborn probabilistic device, or manufacturing variability may cause a failure in digital systems. Automated test generation is a method of creating efficient test cases for validating digital systems with minimal detection time for defective parts. Some other test techniques include the sequential test generation, the scan path testing, and the random test generation. As such, the research domain becomes more mature and grows. Thus, these efforts require a systematic process for the identification, analysis, and categorization of these contributions. We focused on VHDL and from our findings, it is evident that most of the testing procedures for digital circuits concentrate more on the functional/register transfer level. The most widespread method to obtain test cases in some academic papers is HDL model simulation. Although these results point towards increasing interest in this matter, the vast majority are published as conference papers. Only 31% of the methods are realized as software tools and only 63% of all contributions produce executable test cases according to our results.

SYSTEM OVERVIEW

Digital VLSI systems have a lot in them as they help accommodate very high-speed computations on small silicon space. They consist of various subsystems and functional elements each playing their part in the whole system. Indeed, VSLI consists of thousands of logic gates or circuits connected on a small area chip resulting in improvements in computing power and reducing their volume.

The digital counter is an essential component within this intricate web that makes up VLSI design. By counting pulses in succession, helps to measure time intervals which is essential to the generation of timing signals including event count for the proper functioning of such digital systems as computers and other related ones. It is seen in basic timers, clock systems, and sophisticated state machinery that serve as a processor's control unit.

The main counter of our project was designed as the BCD Binary Counter, which finds practical application in counting and interfaces with output indicators. The goal is to detect this phenomenon and then translate it to a human-readable manner on a seven-segment display while lighting up LEDS for representation purposes. It is, therefore, projected to be an independent block that will interface with other system sub-blocks, e.g., a clocking system, a command processor, or even a more comprehensive supervisory mechanism.

The counter must be integrated into a VLSI system. This must be optimized with minimum power usage but should also guarantee a maximum level of dependability and accuracy." In making design choices for the VHDL implementation in the counter, these issues have been considered so that it does not just function but is in line with the efficiency stipulations of VLSI systems. Therefore, in the succeeding chapters, we will discuss the design of a counter and its implementation towards the development of a high-caliber digital VLSI design.

PROPOSED DESIGN

This proposed digital counter for the VLSI system uses a BCD structure that facilitates interfacing with the seven-segment display. In addition, this design is optimized for functionalities to count events as well as interfacing the display within a digital VLSI environment.

Functionality and Features:

Event Counting: For every input pulse, the counter increases its value using the representation of an event.

Display Interface: A seven-segment display provides an easily readable representation of the count value.

Modularity: Moreover, the structure is customary thus making it easy to incorporate into different parts and subsystems within a very large-scale integration system (VLSI).

Power Efficiency: Low power consumption due to the incorporation of power conservation methods into functional operation.

Reset and Control: Synchronous reset at the start of count plus control inputs that enable stopping and restarting count at any time.

My VLSI application or the counter that I built can be related to any scenario in a real world, such as a consumer-grade digital clock which is synchronized or used through a seven-segment display (where it is being shown) to display the time in hours and minutes. You can observe that the modularity of my counter design is easier to integrate into the clock system. The power consumption feature can be critical for battery operated devices where the energy efficiency is highly important. In my counter, the synchronous reset feature can be manipulated to set the clock to any time you wish, and the control input allows you to control the sequence of the counter in such a way that you can pause the time display which can specifically be used during a time set-up mode or in power saving modes.

In such a case, the benefits of my design include efficiency in terms of power consumption and modularity, characteristics that are quite significant for consumer electronics devices with long-lasting batteries and a capability of integrating with other system elements. The disadvantage may be a requirement to add another complexity into use of BDC instead of straightforward a binary counter which may not be necessary sometimes. Moreover, they will also be lacking in some of the speed optimisations or measures of security that you would expect to see in a journal paper that describes an application with these requirements.

Here are some comparisons between the proposed designs stated in literature review section and my counter design:

Speed, Size, and Power Consumption: These factors are vital according to the literature review section on digital counter design. The proposed counter design or my counter design takes power efficiency into consideration and ties in with the literature that indicates there is a need for the conservation of power.

Sleepy CMOS-Sleepy Stack (SC-SS) Technique: It emphasizes fast speed and low energy consumption. The literature claims that SC-SS is better, which is a way of saying a newer and faster and lower energy method. Nonetheless, there are no reports in literature review section that suggest if S-S-SC system is similarly modular as yours BCD scheme provides a unique advantage when it comes to system integration.

Linear Feedback Shift Register (LFSR) Counters: The are often called as low-power and high-density designs exhibiting a logarithmic increase in decoding logic. This kind of LFSR architecture is mostly suitable for a large array application and SoC design. However, in terms of scalability for large arrays, my BCD-based counter could not be compared at all while still being preferred in cases when one requires readable output.

Security: While it may not be directly related to counter design, recent research on VLSI security that relates to hardware Trojan detection is emphasizing the significance the secure design practices. Compared to the literature review section, my documentation lacks any specific reference to security measures, and this could probably form another aspect of improvement in the future.

- STATE DIAGRAM

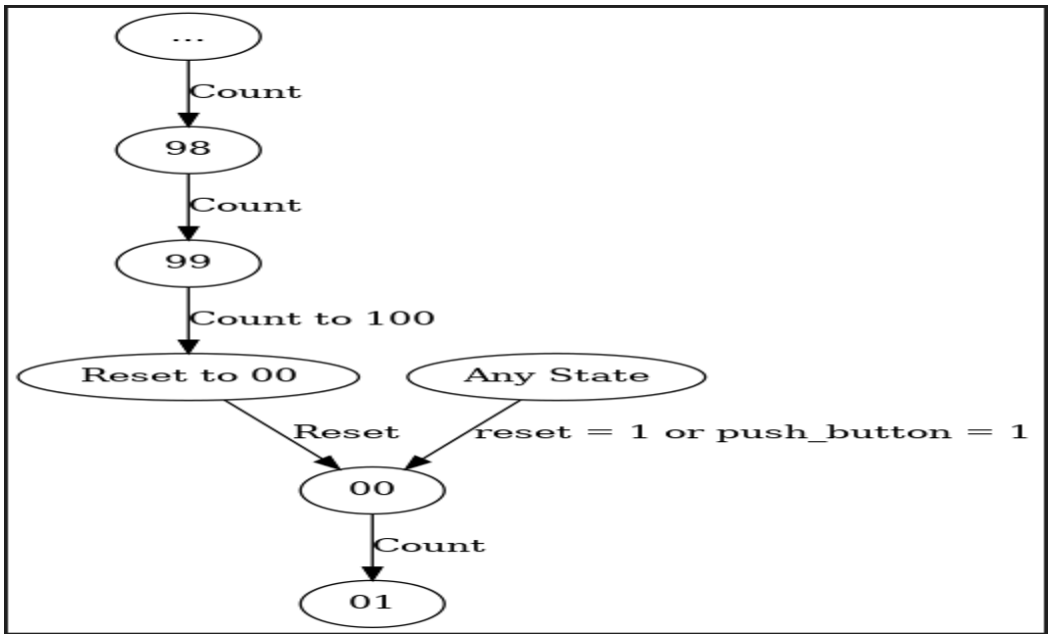

Figure 1: State Diagram-Counter

State diagram offers simple interpretation of how my counter operates. This is my state diagram concentrating on the major transitions and states. You can observe the way in which the counter increases from 00 to 99 and it resets back to 00 in the given diagram. We may also value the presence the push button being active which might lead the counter to be paused or stop. The states in this diagram represent tens and units digits completing 100 stated from 00 to 99.

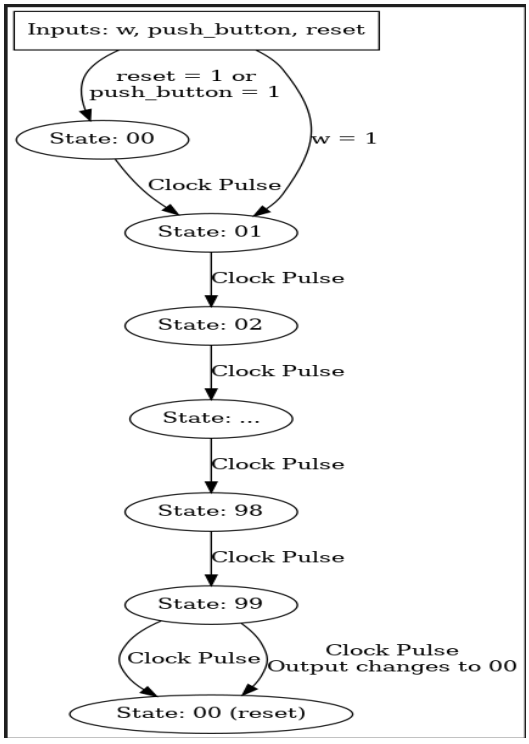

Figure 2: Mealy Machine State Diagram

- MEALY DIAGRAM

In the Mealy machine state diagram, the outputs depend on both the present state and the inputs. The states will represent the values of the counter, so basically, we can say that the control input and the push button can trigger changes in the output. As you can observe in the diagram that there is a clock pulse, so once the control input is activated, the counter will start counting. On the other hand, if we press the reset condition then the clock will be set zero state.

```
entity TopLevelEntity is
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        w : in STD_LOGIC;  -- Control input for event occurrence
        push_button : in STD_LOGIC;  -- Push button to stop the counter
        seg1 : out STD_LOGIC_VECTOR(6 downto 0);  -- Seven-segment display for tens
        seg2 : out STD_LOGIC_VECTOR(6 downto 0);  -- Seven-segment display for units
        led : out STD_LOGIC  -- LED to indicate the occurrence of an event
    );
end TopLevelEntity;
```

Figure 3: TopLevelEntity-Description

I created one file called Top Level Entity which is the main body or structure of my counter because it sustains the overall functionality of the whole counter through the integration of lower or submodules implemented into it.

```
entity SimpleCounter is
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        w : in STD_LOGIC;  -- Control input for event occurrence
        push_button : in STD_LOGIC;  -- Push button to stop the counter
        bcd_tens : out STD_LOGIC_VECTOR(3 downto 0);
        bcd_units : out STD_LOGIC_VECTOR(3 downto 0);
        led : out STD_LOGIC  -- LED to indicate the occurrence of an event
    );
end SimpleCounter;
```

Figure 4: Simple Counter Description

There is another VHDL file called Simple Counter which is a submodule, and it is the one in charge of the counting logic and some other operations such as reset and control input.

```
-- Entity declaration of the multiplexer
entity Mux7Seg is
    Port (
        digit0 : in STD_LOGIC_VECTOR(3 downto 0);
        digit1 : in STD_LOGIC_VECTOR(3 downto 0);
        select1 : in STD_LOGIC;
        BCD_out : out STD_LOGIC_VECTOR(3 downto 0) -- Output to BCD_to_7seg
    );
end Mux7Seg;
```

Figure 5: Mux7Seg-Description

The submodule is called Mux7seg or multiplexer to seven-segment display which is a 2 to 1 multiplexer that selects in between the two inputs to output the BCD or binary coded decimal to the seven-segment display.

```
entity BCD_to_7seg is
    Port (
        BCD : in STD_LOGIC_VECTOR(3 downto 0);
        SEG : out STD_LOGIC_VECTOR(6 downto 0)
    );
end BCD_to_7seg;
```

Figure 6: BCD_to7seg-Description

The last submodule I created is the BCD to seven-segment display where I took the BCD from the multiplexer to drive it into a seven-segment display. I should implement this conversion for me to be able to see the count value in a very understandable format.

Top Level module file

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity TopLevelEntity is
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        w : in STD_LOGIC;   -- Control input for event occurrence
        push_button : in STD_LOGIC;   -- Push button to stop the counter
        seg1 : out STD_LOGIC_VECTOR(6 downto 0);   -- Seven-segment display for tens
        seg2 : out STD_LOGIC_VECTOR(6 downto 0);   -- Seven-segment display for units
        led : out STD_LOGIC   -- LED to indicate the occurrence of an event
    );
end TopLevelEntity;

architecture Behavioral of TopLevelEntity is
    component SimpleCounter
        Port (
            clk : in STD_LOGIC;
            reset : in STD_LOGIC;
            w : in STD_LOGIC;
            push_button : in STD_LOGIC;
            bcd_tens : out STD_LOGIC_VECTOR(3 downto 0);
            bcd_units : out STD_LOGIC_VECTOR(3 downto 0);
            led : out STD_LOGIC
        );
    end component;
component Mux7Seg
    Port (
        digit0 : in STD_LOGIC_VECTOR(3 downto 0);
        digit1 : in STD_LOGIC_VECTOR(3 downto 0);
        select1 : in STD_LOGIC;
        BCD_out : out STD_LOGIC_VECTOR(3 downto 0)
    );
end component;

component BCD_to_7seg
    Port (
        BCD : in STD_LOGIC_VECTOR(3 downto 0);
        SEG : out STD_LOGIC_VECTOR(6 downto 0)
    );
end component;

constant refresh_rate : integer := 500;

signal bcd_tens, bcd_units : STD_LOGIC_VECTOR(3 downto 0);
signal mux_output : STD_LOGIC_VECTOR(3 downto 0);
signal select_signal : STD_LOGIC := '0';
signal refresh_counter : integer range 0 to refresh_rate := 0; -- Initialize with refresh_rate
signal seg_tens, seg_units : STD_LOGIC_VECTOR(6 downto 0);
begin
    uut_simple_counter: SimpleCounter Port Map (
        clk => clk,
        reset => reset,
        w => w,
        push_button => push_button,
        bcd_tens => bcd_tens,
        bcd_units => bcd_units,
        led => led
    );

    uut_mux7seg: Mux7Seg Port Map (
        digit0 => bcd_units,
        digit1 => bcd_tens,
        select1 => select_signal,
        BCD_out => mux_output
    );

    uut_bcd_to_7seg: BCD_to_7seg Port Map (
        BCD => bcd_tens,
        SEG => seg_tens
    );

        uut_bcd_to_7seg_units: BCD_to_7seg Port Map (
        BCD => bcd_units,
        SEG => seg_units
    );

         -- Connect the outputs to the seven-segment display signals
    seg1 <= seg_tens;
    seg2 <= seg_units;
     display_refresh_process: process(clk)
     begin
          if rising_edge(clk) then
              if refresh_counter < refresh_rate then
                  refresh_counter <= refresh_counter + 1;
              else
                  refresh_counter <= 0;
                  select_signal <= not select_signal;
              end if;
          end if;
      end process;
end Behavioral;
```

Figure 7: VHDL Implementation Top-Level Entity

This VHDL file implements mechanisms that integrate the 7-segment display and the counter.

Inputs: The inputs are clock, reset, control input, w, and a push button.

Outputs: The 7 segments display seg1 and seg2 are the outputs.

The components that are being instantiated within the Top-Level file are Simple Counter, Mux7Seg, and two BCD_to_7seg components.

Top Level file connects certain internal signals between all the components to achieve the expected functionality.

We can also observe the presence of the display refresh process and the toggling process between tens and units display using the select signal.

Top Level Entity Testbench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY TopLevelEntity_tb IS
END TopLevelEntity_tb;
ARCHITECTURE behavior OF TopLevelEntity_tb IS
    COMPONENT TopLevelEntity
        PORT(
            clk : IN  std_logic;
            reset : IN  std_logic;
            w : IN  std_logic;
            push_button : IN  std_logic;
            seg1 : OUT  std_logic_vector(6 downto 0);
            seg2 : OUT  std_logic_vector(6 downto 0);
            led : OUT  std_logic
        );
    END COMPONENT;
    signal clk : std_logic := '0';
    signal reset : std_logic := '0';
    signal w : std_logic := '0';
    signal push_button : std_logic := '0';
    signal seg1 : std_logic_vector(6 downto 0);
    signal seg2 : std_logic_vector(6 downto 0);
    signal led : std_logic;
    constant clk_period : time := 10 ns;
BEGIN
    uut: TopLevelEntity PORT MAP (
        clk => clk,
        reset => reset,
        w => w,
        push_button => push_button,
        seg1 => seg1,
        seg2 => seg2,
        led => led
    );
    clk_process : process
    begin
        while true loop
            clk <= '0';
            wait for clk_period/2;
            clk <= '1';
            wait for clk_period/2;
        end loop;
    end process;
    stim_proc: process
    begin
        -- Initial reset
        reset <= '1';
        wait for 100 ns;
        reset <= '0';
        wait for 100 ns;
        -- Single Event Triggering
        w <= '1';
        wait for 200 ns;
        w <= '0';
        -- Counter should increment now
        -- Stopping the counter with the push button
        push_button <= '1';
        wait for 100 ns;
        push_button <= '0';
        -- Resetting the counter to 0
        reset <= '1';
        wait for 100 ns;
        reset <= '0';
        wait for 200 ns;
        -- Finish the simulation
        wait;
    end process;
END behavior;
```

Figure 8: Testbench for Top-Level Entity

The testbench for Top Level Entity ensures that the counter contains the next functionalities:

- Initialize the counting from the reset signal.
- Increment the signal through the control input aligned with the LED.
- It will stop counting once the push button is enabled or activated.
- It will continue or resume its operation after the push button is set to 0.

Module code for BCD to 7seg

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity BCD_to_7seg is
    Port (
        BCD : in STD_LOGIC_VECTOR(3 downto 0);
        SEG : out STD_LOGIC_VECTOR(6 downto 0)
    );
end BCD_to_7seg;

architecture Behavioral of BCD_to_7seg is
begin
    process (BCD)
    begin
        case BCD is
            when "0000" =>
                SEG <= "0000001"; -- 0
            when "0001" =>
                SEG <= "1001111"; -- 1
            when "0010" =>
                SEG <= "0010010"; -- 2
            when "0011" =>
                SEG <= "0000110"; -- 3
            when "0100" =>
                SEG <= "1001100"; -- 4
            when "0101" =>
                SEG <= "0100100"; -- 5
            when "0110" =>
                SEG <= "0100000"; -- 6
            when "0111" =>
                SEG <= "0001111"; -- 7
            when "1000" =>
                SEG <= "0000000"; -- 8
            when "1001" =>
                SEG <= "0000100"; -- 9
            when others =>
                SEG <= "1111111"; -- Blank or other value
        end case;
    end process;
end Behavioral;
```

Figure 9: VHDL Implementation BCD_to_7seg

In BCD to 7-segment coding, we convert a Binary Coded Decimal input or BCD input to 7-segment display output as you can see in the coding.

Input: Only one input that represents BCD.

Output: a 7-bit output that represents a 7-segment display.

We have a process inside the architecture behavioral that is sensitive to any change where BCD input is involved. Inside that process, I implemented a case statement to map the Binary-coded decimal or BCD to a corresponding 7-segment display. As an example, “1001” maps to “0000100” for the display where it lights up all the segments except the middle one. In case the BCD is outside the range from 0 to 9, all segments will be turn off, showing 1111111.

Testbench for BCD to 7seg

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity BCD_to_7seg_tb is
end BCD_to_7seg_tb;

architecture Behavioral of BCD_to_7seg_tb is
    signal BCD_tb : STD_LOGIC_VECTOR(3 downto 0) := "0000";
    signal SEG_tb : STD_LOGIC_VECTOR(6 downto 0);

    -- Instantiate the Unit Under Test (UUT)
         begin
    uut: entity work.BCD_to_7seg
        port map (
            BCD => BCD_tb,
            SEG => SEG_tb
        );

    -- Test process
         process
    begin
                BCD_tb <= "0000";
        wait for 20 ns;
        BCD_tb <= "0001";
        wait for 20 ns;
        BCD_tb <= "0010";
        wait for 20 ns;
        BCD_tb <= "0011";
        wait for 20 ns;
        BCD_tb <= "0100";
        wait for 20 ns;
        BCD_tb <= "0101";
        wait for 20 ns;
        BCD_tb <= "0110";
        wait for 20 ns;
        BCD_tb <= "0111";
        wait for 20 ns;
        BCD_tb <= "1000";
        wait for 20 ns;
        BCD_tb <= "1001";
        wait for 20 ns;
        assert false report "Simulation Complete" severity failure;
       wait;
                end process;
end Behavioral;
```

Figure 10: Testbench for BCD_to_7seg

This testbench is set for BCD to 7seg entity which converts a Binary Coded Decimal to a 7-segment display output.

I declared two signals such as BCD_tb as the input and SEG_tb inside the testbench. As you can see in the testbench, I instantiated the BCD to 7seg. We test many scenarios by changing the input values of the select signal and the output for the cases.

Module code for Multiplexer to 7 segment display or Mux7seg

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Entity declaration of the multiplexer
entity Mux7Seg is
     Port (
          digit0 : in STD_LOGIC_VECTOR(3 downto 0);
          digit1 : in STD_LOGIC_VECTOR(3 downto 0);
          select1 : in STD_LOGIC;
          BCD_out : out STD_LOGIC_VECTOR(3 downto 0) -- Output to BCD_to_7seg
     );
end Mux7Seg;

-- Architecture of the multiplexer
architecture Behavioral of Mux7Seg is
begin
     -- Multiplexer process
     process (digit0, digit1, select1)
     begin
          if select1 = '0' then
               BCD_out <= digit0;
          else
               BCD_out <= digit1;
          end if;
     end process;
end Behavioral;
```

Figure 11: VHDL Implementation Mux7Seg

In this multiplexer, we select between the two digit0 or digit1 which are the 4-bit inputs.

Two 4-bit BCD inputs (digit0 and digit1).

One output is a 4-bit BCD output which is the one going to the BCD_to7seg.

Inside the architecture behavioural we can see that I implemented a process statement where it checks the select1 signal that will define which BCD input to go through the output.

Testbench for Mux7Seg

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mux7Seg_tb is
end Mux7Seg_tb;

architecture Behavioral of Mux7Seg_tb is
    signal digit0_tb : STD_LOGIC_VECTOR(3 downto 0) := "0000";
    signal digit1_tb : STD_LOGIC_VECTOR(3 downto 0) := "0001";
    signal select1_tb : STD_LOGIC := '0';
    signal BCD_out_tb : STD_LOGIC_VECTOR(3 downto 0);

    -- Instantiate the Unit Under Test (UUT)
begin
    uut: entity work.Mux7Seg
        port map (
            digit0 => digit0_tb,
            digit1 => digit1_tb,
            select1 => select1_tb,
            BCD_out => BCD_out_tb
        );

    -- Test process
    process
    begin
        -- Test with initial values
        select1_tb <= '0';
        wait for 20 ns;
        select1_tb <= '1';
        wait for 20 ns;

        -- Change digit0 and digit1 values
        digit0_tb <= "0010"; -- Change digit0
        select1_tb <= '0';
        wait for 20 ns;
        digit1_tb <= "0011"; -- Change digit1
        select1_tb <= '1';
        wait for 20 ns;

        -- Additional tests with new values
        select1_tb <= '0';
        wait for 20 ns;
        select1_tb <= '1';
        wait for 20 ns;

        assert false report "Simulation Complete" severity failure;
        wait;
    end process;
end Behavioral;
```

Figure 12: Testbench for Mux7Seg

Four signals are stated in the testbench from where we have two input BCD signals (digit0_tb, and digit1_tb), a select signal which is select1_tb, and the output which is BCD_out_tb signal.

The Mux7Seg entity is instantiated and finally, we have implemented various test cases or scenarios where we change the select signal and the input BCD values, observing the output for each case.

Module file for Simple Counter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity SimpleCounter is
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        w : in STD_LOGIC;  -- Control input for event occurrence
        push_button : in STD_LOGIC;  -- Push button to stop the counter
        bcd_tens : out STD_LOGIC_VECTOR(3 downto 0);
        bcd_units : out STD_LOGIC_VECTOR(3 downto 0);
        led : out STD_LOGIC  -- LED to indicate the occurrence of an event
    );
end SimpleCounter;

architecture Behavioral of SimpleCounter is
    signal internal_count : integer range 0 to 99 := 0;
    signal clk_divided : STD_LOGIC := '0';
    signal count_clk : integer range 0 to 1048 := 0;
    signal counting : STD_LOGIC := '0';  -- Indicates if the counter is active

begin
    clock_divider: process(clk)
    begin
        if rising_edge(clk) then
            if count_clk < 1048 - 1 then
                count_clk <= count_clk + 1;
            else
                count_clk <= 0;
                clk_divided <= NOT clk_divided;
            end if;
        end if;
    end process clock_divider;

    counter_process: process(clk)
    begin
        if rising_edge(clk) then
            if reset = '1' then
                internal_count <= 0;
                counting <= '0';
            elsif push_button = '1' then
                counting <= '0';
            elsif w = '1' and counting = '1' then
                if internal_count = 99 then
                    internal_count <= 0;
                else
                    internal_count <= internal_count + 1;
                end if;
            elsif w = '1' then
                counting <= '1';
            else
                counting <= '0';  -- Default state to avoid inferred latches
            end if;
        end if;
    end process counter_process;

    process(internal_count)
    begin
        bcd_tens <= std_logic_vector(to_unsigned((internal_count / 10), 4));
        bcd_units <= std_logic_vector(to_unsigned((internal_count mod 10), 4));
    end process;

    led <= w when counting = '1' else '0';

end Behavioral;
```

Figure 13: VHDL Implementation Simple Counter

In the module code for Simple Counter, you can see a simple counter implementation with multiple inputs and outputs.

Inputs: The inputs are the clock signal, reset signal, control input, and a push button.

Outputs: The outputs are two which are 4-bit BCD outputs, bcd_tens and bcd_units, and an LED signal that is aligned with the control input.

I implemented a clock divider inside the coding where I divided the clock frequency. What it does is that it toggles a clock-divided signal at a certain interval that is controlled by count_clk. Then we have a counter process where we can see the counting logic that is incrementing an internal counter based on the control signals. The counter rests depending on whether the reset signal is high or when it reaches 99. Moreover, we have an internal count process where the bcd_tens and bcd_units are split into tens and units which are then converted or transformed to BCD. Finally, the LED output is turned on whenever the control input is triggered or whenever the counter is counting.

Testbench file for Simple Counter

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY SimpleCounter_tb IS
END SimpleCounter_tb;

ARCHITECTURE behavior OF SimpleCounter_tb IS
    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT SimpleCounter
    PORT(
        clk : IN std_logic;
        reset : IN std_logic;
        w : IN std_logic;
        push_button : IN std_logic;
        bcd_tens : OUT std_logic_vector(3 downto 0);
        bcd_units : OUT std_logic_vector(3 downto 0);
        led : OUT std_logic
    );
    END COMPONENT;
    signal clk : std_logic := '0';
    signal reset : std_logic := '0';
    signal w : std_logic := '0';
    signal push_button : std_logic := '0';
    signal bcd_tens : std_logic_vector(3 downto 0);
    signal bcd_units : std_logic_vector(3 downto 0);
    signal led : std_logic;
    constant clk_period : time := 10 ns;
BEGIN
    uut: SimpleCounter PORT MAP (
        clk => clk,
        reset => reset,
        w => w,
        push_button => push_button,
        bcd_tens => bcd_tens,
        bcd_units => bcd_units,
        led => led
    );
    clk_process :process
    begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
    end process;
     stim_proc: process
     begin
          -- hold reset state for 100 ns.
          reset <= '1';
          wait for 100 ns;
          reset <= '0';
          -- Test Case 1: Normal counting
          w <= '1';  -- Enable counting
          wait for 1 us;  -- Wait for some time
          w <= '0';  -- Disable counting
          -- Test Case 2: Push button stop
          w <= '1';
          wait for 500 ns;
          push_button <= '1';  -- Stop counting
          wait for 100 ns;
          push_button <= '0';
          w <= '0';
          -- Test Case 3: Reset during counting
          w <= '1';
          wait for 500 ns;
          reset <= '1';  -- Reset
          wait for 100 ns;
          reset <= '0';
          -- Test Case 4: Rapid on/off
          w <= '1';
          wait for 100 ns;
          w <= '0';
          wait for 100 ns;
          w <= '1';
          wait for 100 ns;
          w <= '0';
          wait;
     end process;

END behavior;
```

Figure 14: Testbench for Simple Counter

The scenarios that have been tested in the Simple Counter testbench are:

- The counting system or operation counting when the counter is activated.
- The counter stops based on the activation of the push button.
- The counter reset when the reset signal is enabled.

- When the reset condition is active or high (No counting):

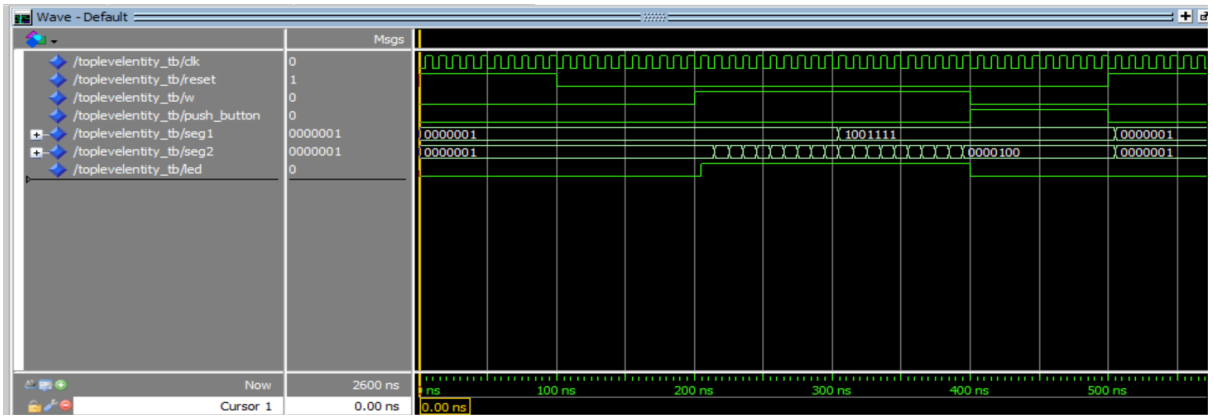

Figure 15: Reset Condition-Waveform

- When the clock is inactive, and the other parameters are set to 0 (No counting):

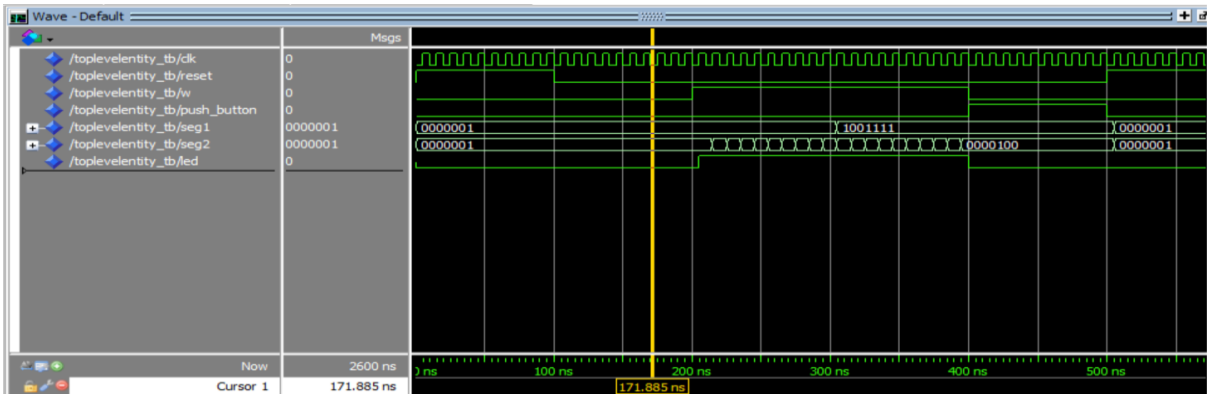

Figure 16: Clock Inactive-Waveform

- When it starts counting, the control input is set to 1 or active (Initial counting):

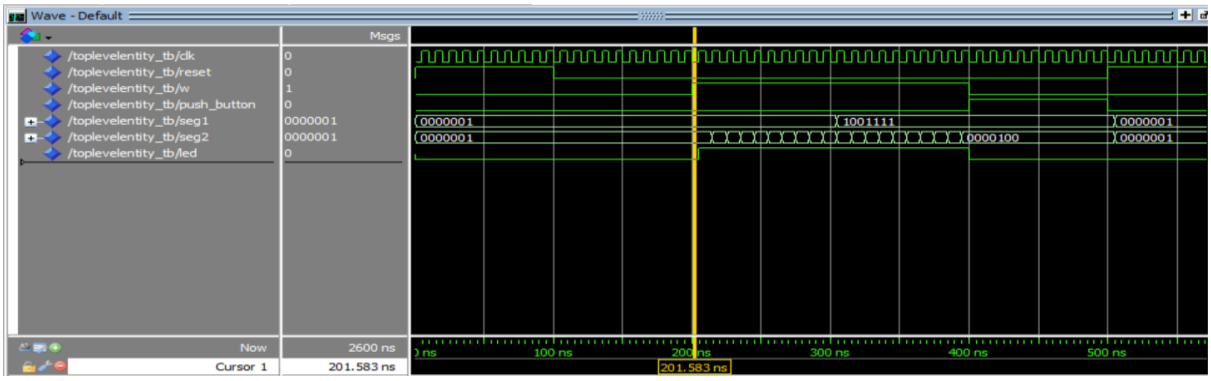

Figure 17: Initial Control Input-Waveform

- When the control input gets triggered and the presence of an event the LED is activated (Counting).

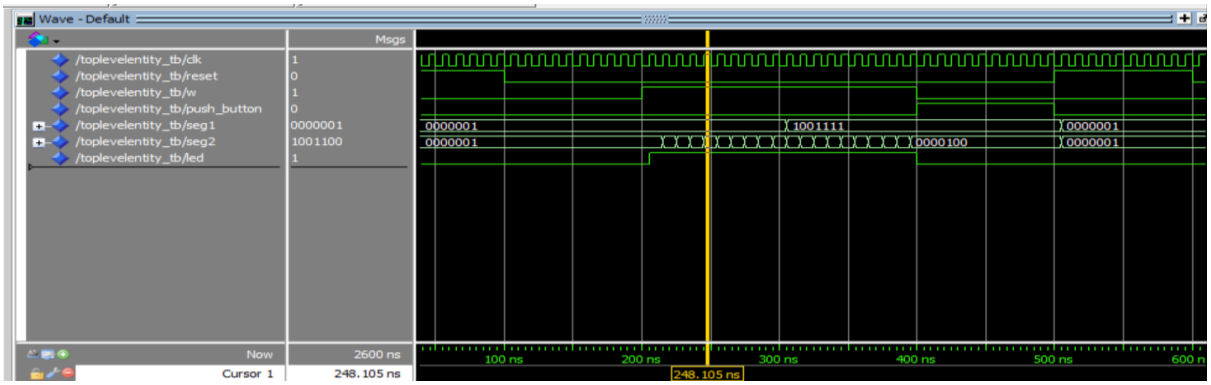

Figure 18: Counting Process Activation-Waveform

- When the counter is proceeding counting.

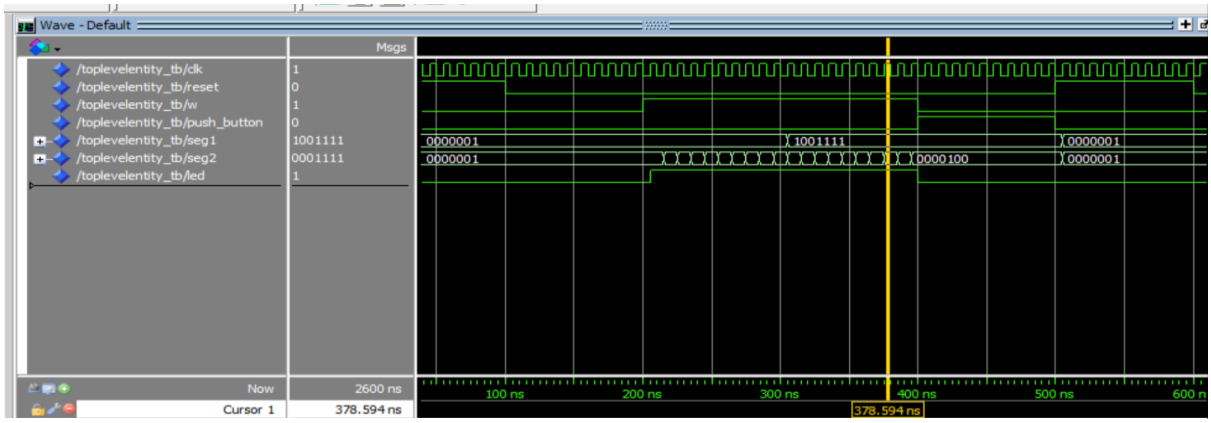

Figure 19: Counting W and LED activated-Waveform

- When the push button is activated, automatically the counter stop counting, so w=led=0

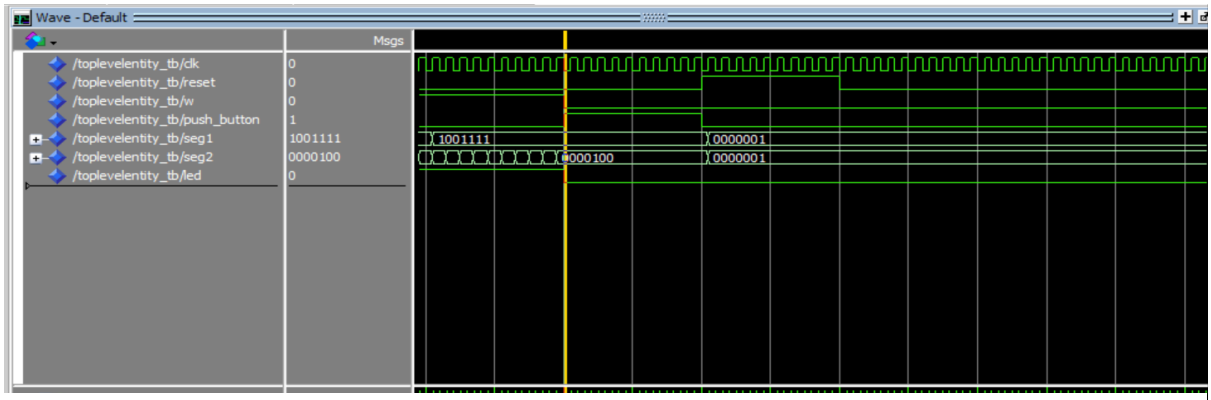

Figure 20: Push Button Activated-Waveform

- I implemented another condition for reset:

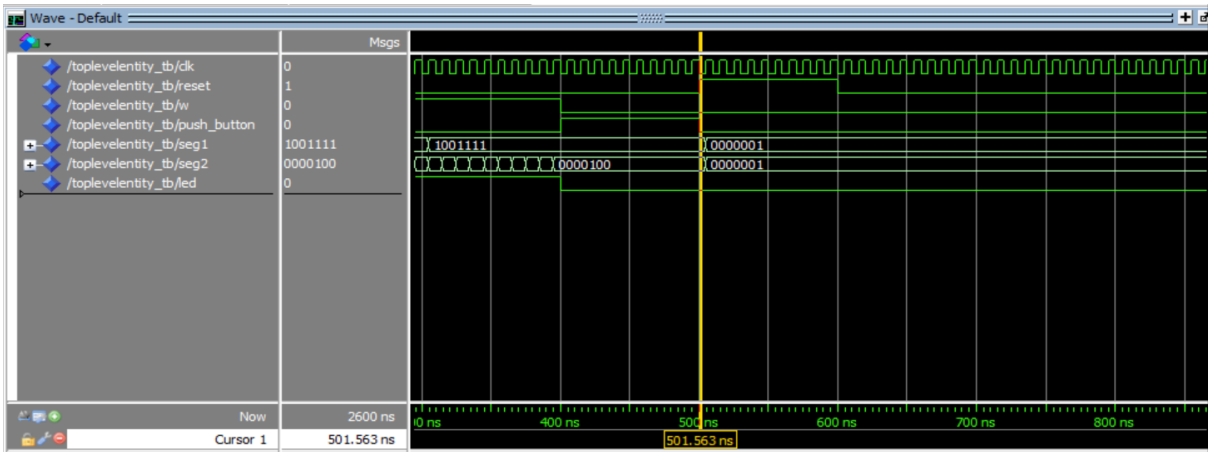

Figure 21: Reset Condition 2-Waveform

End of the counter

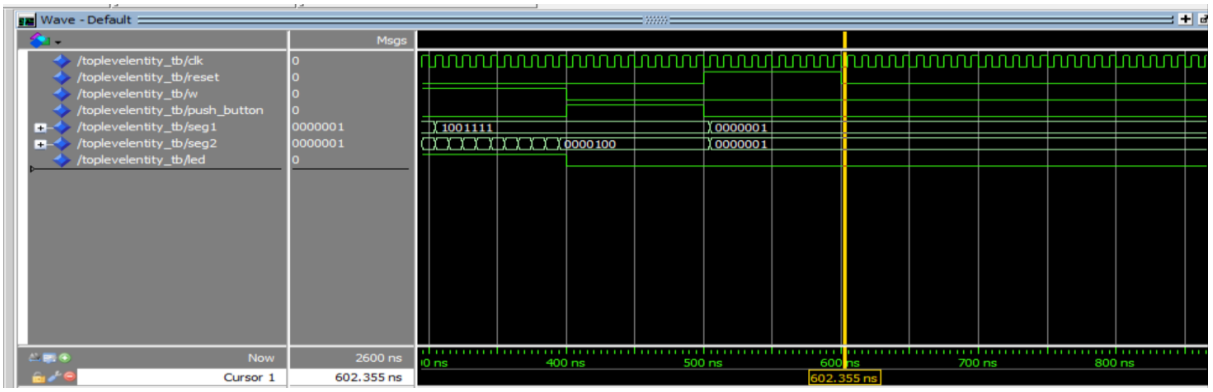

Figure 22: No Counting Process

My counter description.

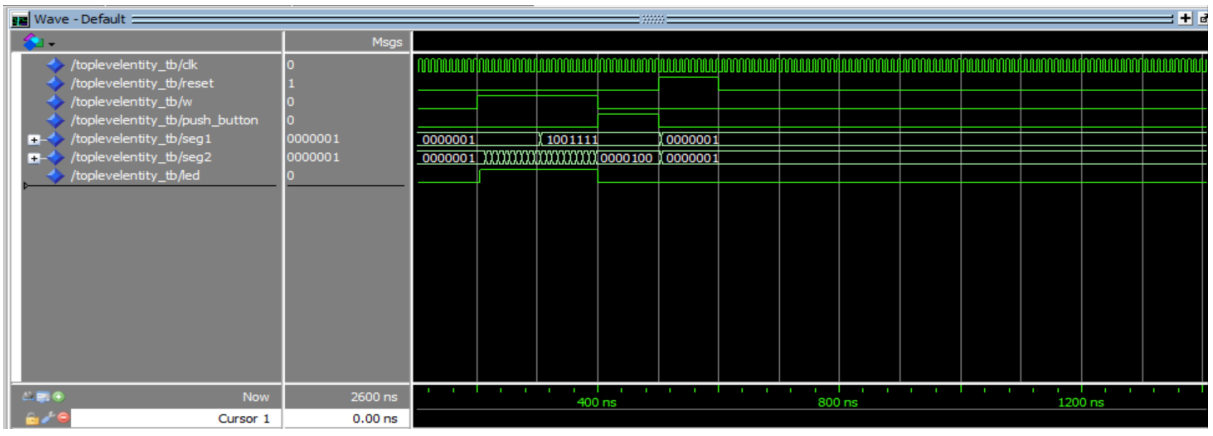

Figure 23: My Counter Waveform

DISCUSSION

I shared multiple waveform snapshots, and this discussion will be based on the conditions that I set starting from Figure 15 until Figure 24 from simulation results section:

- When the reset condition is active or high:

In Figure 15, you can observe the behaviour of the counter when the reset condition is activated which implies a deactivation of the counting in a way that there is no counting process happening due to the activation of the reset. Besides that, during that condition, the counter might initialize or reset with a definite value which is set to 1 or active mode while the other parameters such as the clock, control input, LED, etc. are automatically at zero which indicates that they are not activated. In other words, we can deduce that whenever the reset condition is activated, the output of the counter is forced to zero regardless of the clock, seg1 and seg2, and led signal.

- When the clock is inactive, and the other parameters are set to 0.

In Figure 16, we have a different case which is when the counter is not active which is a clear indication of no counting process as all the parameters are set to zero. Besides that, we can emphasize by stating that the absence of the activation clock signal means that the counter will remain in its current state. Moreover, the counter does not advance, and we can see in the waveform that there is a flat line for the clock signal without rising or falling edges to activate or trigger the counting.

- When it starts counting, the control input is set to 1 or active.

In Figure 17, we can observe in the waveform that the counter is set up as the control input is an active 1. During activation of the control input signal and initiation of the clock pulse the counter is in motion. We can observe that in the waveform referenced the led is not an active 1 yet. This is because although the control input is active, the counter has not started counting properly, and apart from that, we can also say that the presence of an event is not reflected in this case. Lastly, one important point to highlight is that in the waveform whenever the control input is active, the other parameters such as reset, push button, and the clock are not active, and this is because the counter is starting to activate for the counting process.

- When the control input gets triggered and the presence of an event the LED is activated.

Based on the waveform in Figure 18, we can observe a different behaviour of the counter which is when the control input is triggered which leads to the activation of LED as well while other parameters such as push button, reset, etc. are deactivated and this is because the counting process is taking place through the seven-segment display seg1 and seg2. The counter continues its counting after there has been a new external event on a specified input channel, so the LED gets triggered. In the waveform, we can see the clocked on/off behaviour of the outputs and a change in their sequence.

Next, in Figure 19 the counting process carries on sequentially and it is reflected in the seven-segment display. The LED is still high due to the increment of the counter through the control

input signal. As counting goes on, you can see that the waveform depicts a rhythmic pattern of the clock and a progression of the binary in the output of the counter. It is an uninterrupted counting phase because the counter is continuously active over time.

- When the push button is activated, automatically the counter stops counting.

In Figure 20, we can appreciate that we are coming from a counting period until a button called the push button is pressed which is going to interrupt, stop, or pause the counting process, and in the waveform, it is reflected as a sudden halt or a small peak that indicate the push button activation. Besides that, when the push button is pressed the LED, and the control input automatically will be set to low or zero due to a stop process caused by the push button signal. Lastly, it will only pause the counter and read its value without resetting it.

I stated in Figure 21 another condition for reset signal where the counter is set to reset mode, so the push button, control input, and LED will be low or zero.

CONCLUSION

In conclusion, I could successfully complete the project of creating a counter that counts from o to 99 with the use of Quartus software where I implemented the VHDL code needed for the counter. I created four different VHDL files to facilitate or ease the work which are: One VHDL file called Simple Counter where I implemented the counting logic and a synchronous reset function for the counter. Another VHDL file created was the multiplexer to 7-segment display or Mux7Seg where I essentially ensured that only one digit was properly activated ensuring a clear output. The third VHDL file is BCD to 7seg where I implemented the conversion of the digital count into a seven-segment bit pattern so that the display will show a correct sequence of numbers.

A comprehensive simulation test showed the high level of precision of this VHDL implementation for event counting and interfacing display. The operations in the counter are based on the initial reset, event-driven counting, control input, and LED which conform to the requirements stated in this project. This is one of the basic counters which is critical in various systems where precise sequencing and timing are required. Finally, I was able to explore and analyze the waveforms extracted from the counter which played an indispensable role in the functionality of the counter designed for this project.

REFERENCES

1. Sharma, A., Sohal, H., & Kaur, H. (2019). Sleepy CMOS-Sleepy Stack (SC-SS): A Novel High Speed, Area and Power Efficient Technique for VLSI Circuit Design. *Journal of Circuits, Systems, and Computers*, 28, 1950197:1-1950197:25. Sleepy CMOS-Sleepy Stack (SC-SS)
2. Morrison, D., Delic, D., Yuce, M., & Redouté, J.-M. (2019). Multistage Linear Feedback Shift Register Counters With Reduced Decoding Logic in 130-nm CMOS for Large-Scale Array Applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27, 103-115. Multistage Linear Feedback Shift Register Counters
3. Pearce, H., Surabhi, V. R., Krishnamurthy, P., Trujillo, J., Karri, R., & Khorrami, F. (2022). Detecting Hardware Trojans in PCBs Using Side Channel Loopbacks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30, 926-937. Detecting Hardware Trojans in PCBs
4. Vivekananda, A. A., & Enoiu, E. (2020). Automated Test Case Generation for Digital System Designs: A Mapping Study on VHDL, Verilog, and SystemVerilog Description Languages. *Designs*, 4(3), 31. https://doi.org/10.3390/designs4030031