**Counter Design**

**CENG 3151**

Submitted by

Michael Lanford

1816337

Computer Engineering

University of Houston-Clear Lake

Houston, Texas 77058

March 31, 2023

# Table of Contents

**1** [**Abstract 3**](#_Toc126060766)

[**2** **Introduction** 3](#_Toc126060767)

**3 Requirements……………………………………………………………………….……......3**

**4 Prelab………………………………………………………………………………..……….4**

**5 Implementation……………………………………………………….…………………......4**

5.1 Design Code / Design Diagrams**……………………………………………………………5**

5.2 Schematics**……………………………………………………………………………….....7**

5.3 Testbench**…………………………………………………………………………………....8**

5.4 Waveform / Results**………………………………………………………………………...11**

[**6** **Conclusion** 12](#_Toc126060768)

**Abstract**

A binary counter is constructed with flip-flops and it takes the outputs of one cell and moves it to the clock input of the next. In VHDL, to convert from integer to std\_logic\_vector you use the to\_signed() function. When you convert from std\_logic\_vector to integer, you use the signed() typecast.

# Introduction

For this project, we will be using Xilinx Vivado to build and test a 4-bit binary counter with asynchronous reset using behavioral implementation style and an 8-bit counter with asynchronous reset using structural implementation style and each of these will accept some input and produce some output.

1. **Requirements**

Design a 4-bit binary counter with 3 inputs: Clock, Count\_Enable, and Reset. This circuit will output 2 values: a 4 bit Count\_Val and Cout.. Design a 8-bit binary counter with 3 inputs: Clock, Count\_Enable, and Reset. This circuit will output 2 values: a 8 bit Count\_Val and Cout. The figures of the circuits can be seen below:

Diagram, PowerPoint

Description automatically generated

**Figure 1:** Diagram for the 4-bit counter to be designed.

Diagram

Description automatically generated

**Figure 2:** Diagram for the 8-bit counter to be designed.

1. **Prelab**

For this prelab, we were required to write about the working of a 4-bit binary counter with behavioral implementation style and an 8-bit binary counter with structural implementation in computer architecture.

A binary counter is constructed with flip-flops and it takes the outputs of one cell and moves it to the clock input of the next. In VHDL, to convert from integer to std\_logic\_vector you use the to\_signed() function. When you convert from std\_logic\_vector to integer, you use the signed() typecast. The block diagram for the 8-bit counter made of 2 4-bit counters is below:

Clock

Clock

Input 1

Input 2

Input 3

Input 4

Input 1

Input 2

Input 3

Input 4

Output

Output

1. **Implementation**

The first step of implementation was to create a new project in Xilinx Vivado. After setting up the project with the correct options, we added a design source file for the 4-bit counter circuit and added the necessary inputs and outputs to it. We then coded the temporary signals and the process for the counter. After that, we created the simulation file and added in the component instantiation, interface signal declarations, instance port map, the clock process, and the test cases to it then tested the waveform. For the 8-bit counter circuit, we created another new project in Xilinx Vivado. After setting up the project with the correct options, we added a design source file and added the necessary inputs and outputs to it. We then added the component instantiation from the 4-bit counter, the temporary signals, and the port maps. After that, we created the simulation file and added in the component instantiation, interface signal declarations, instance port map, the clock process, and the test cases to it then tested the waveform.

**4.1 Design Code / Design Diagrams**

------Part 1

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use ieee.std\_logic\_unsigned.all;

--Input and Output Declarations

entity project9 is

Port ( Clock : in STD\_LOGIC;

Count\_Enable : in STD\_LOGIC;

Reset : in STD\_LOGIC;

Cout : out STD\_LOGIC;

Count\_Val : out STD\_LOGIC\_VECTOR (3 downto 0));

end project9;

architecture Behavioral of project9 is

signal temp: std\_logic\_vector (3 downto 0):= "0000"; --Signal Declarations

signal carry: std\_logic;

begin

process(Clock, Count\_Enable, Reset, temp) -- Process for the 4-bit adder

begin

if(Reset ='1') then--enable reset

temp <= "0000";

carry <= '0';

elsif (rising\_edge(Clock)) then--update at rising edge

if(Count\_Enable = '1') then--if counter is enabled

if (temp = "1111") then--if we encounter overflow

temp <= "0000";--reset to 0

carry <= '1';--signal overflow

else--else, if no overflow, keep counting

carry <= '0';

temp <= temp + "0001";

end if;

end if;

end if;

Count\_val <= temp;--actual counter output

Cout <= carry;--actual carry output

end process;

end Behavioral;

------Part 2

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

--Input and Output Declarations

entity lab9part2 is

Port ( Clock : in STD\_LOGIC;

Count\_Enable : in STD\_LOGIC;

Reset : in STD\_LOGIC;

Cout : out STD\_LOGIC;

Count\_Val : out STD\_LOGIC\_VECTOR (7 downto 0));

end lab9part2;

architecture Behavioral of lab9part2 is

component project9 is --Component instantiation

Port ( Clock : in STD\_LOGIC;

Count\_Enable : in STD\_LOGIC;

Reset : in STD\_LOGIC;

Cout : out STD\_LOGIC;

Count\_Val : out STD\_LOGIC\_Vector(3 downto 0));

end component;

signal carry\_1: STD\_LOGIC; --Signal declarations

signal carry\_2: STD\_LOGIC;

begin

Counter0: project9 port map (Clock,Count\_Enable, Reset, carry\_1, Count\_Val(3 downto 0)); --Port maps

Counter1: project9 port map (carry\_1,Count\_Enable, Reset, carry\_2, Count\_Val(7 downto 4));

Cout <= carry\_1 and carry\_2;

end Behavioral;

**4.2 Schematics**

**Diagram, schematic

Description automatically generated**

**Figure 3:** 4-bit counter circuit.

Diagram

Description automatically generated

**Figure 4:** 8-bit counter circuit.

**4.3 Testbench**

--Part 1

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity project9\_tb is

-- Port ( );

end project9\_tb;

architecture Behavioral of project9\_tb is

component project9 is --Component instantiation

Port ( Clock : in STD\_LOGIC;

Count\_Enable : in STD\_LOGIC;

Reset : in STD\_LOGIC;

Cout : out STD\_LOGIC;

Count\_Val : out STD\_LOGIC\_VECTOR (3 downto 0));

end component;

signal Clock, Count\_Enable, Reset: std\_logic := '0'; --Signal declarations

signal Cout: std\_logic;

signal Count\_Val: std\_logic\_vector (3 downto 0);

constant clk\_period: time := 10ns; --Clock constant

begin

uut: project9 port map (Clock => Clock, Count\_Enable => Count\_Enable, Reset => Reset, Cout => Cout, Count\_Val => Count\_Val); --Port map

clock\_process: process --Clock process

begin

wait for Clk\_period/2;

Clock <= not Clock;

end process;

Stim\_proc: process --Test case process

begin

Reset <= '0'; --Test Case 1: Start Counting

Count\_Enable <= '1';

wait for 200ns;

Count\_enable <= '0'; --Test Case 2: Pause Counting

wait for 60ns;

Count\_enable <= '1'; --Test Case 3: Resume Counting

wait for 320 ns;

Reset <= '1'; --Test Case 4: Reset Counting

wait;

end process;

end Behavioral;

--Part 2

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity lab9part2\_tb is

-- Port ( );

end lab9part2\_tb;

architecture Behavioral of lab9part2\_tb is

component lab9part2 --Component instantiation

Port ( Clock : in STD\_LOGIC;

Count\_Enable : in STD\_LOGIC;

Reset : in STD\_LOGIC;

Cout : out STD\_LOGIC;

Count\_Val : out STD\_LOGIC\_VECTOR (7 downto 0));

end component;

signal Clock, Count\_enable, Reset: std\_logic := '0'; --Signal declarations

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

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

constant clock\_period: time :=5 ns; --Clock constant

begin

uut: lab9part2 port map (Clock => Clock, Count\_Enable => Count\_Enable, Reset => Reset, Cout => Cout, Count\_Val => Count\_Val); --Port maps

Clock\_process: process --Clock process

begin

wait for clock\_period/2;

Clock <= not Clock;

end process;

Stim\_proc: process --Test case process

begin

Reset <= '0'; --Test Case 1: Start Counting

Count\_enable <= '1';

wait for 1500 ns;

Count\_Enable <= '0'; --Test Case 2: Pause Count

wait for 20 ns;

Count\_Enable <= '1'; --Test Case 3 Resume Counting

wait for 50 ns;

Reset <= '1'; --Test Case 4 Resume Counting

wait;

end process;

end Behavioral;

**4.4 Waveform / Results**

The waveforms below shows that the four programs we made above in the Testbench and Design Code / Design Diagrams sections was able to produce correct results for each of our inputs that we created. For example, at 140ns on Figure 5 when Clock and Count\_Enable are 1 while Reset is 0, Count\_Val will increment the counter from f to 0. Also, at 1270ns on Figure 6 when Clock and Reset are 0 and Count\_Enable is 1, Count\_Val will increment from 253 to 254.

A screenshot of a computer

Description automatically generated

**Figure 5:** 4-bit counter Waveform.

Graphical user interface

Description automatically generated

**Figure 6:** 8-bit counter Waveform.

# Conclusion

In this lab, we were able to successfully code a 4-bit counter and an 8-bit counter using two of the 4-bit counters in Xilinx Vivado by using the little code snippets given to us in our prelab as a base for our code. These programs were made to be able to simulate how counters work inside a computer, which can be seen in the waveforms due to the correct output being produced for what we inputted.