**4-Bit Full Adder**

**CENG 3151**

Submitted by

Michael Lanford

1816337

Computer Engineering

University of Houston-Clear Lake

Houston, Texas 77058

February 20, 2023

# Table of Contents

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

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

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

**4 Prelab………………………………………………………………………………..……….3**

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

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

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

5.3 Testbench**…………………………………………………………………………………....7**

5.4 Waveform / Results**………………………………………………………………………....9**

[**6** **Conclusion** 10](#_Toc126060768)

**Abstract**

# A 4-bit full adder is a circuit where the output depends on the inputs and the carry bit. In this lab, we will be using Xilinx Vivado to build the 4-bit full adder made up of several 1-bit full adders. The major results of this experiment will be a waveform that shows the correct output for each input, which will reflect our truth table in the prelab.

# Introduction

For this project, we will be using Xilinx Vivado to build and test a 4-bit full adder that will accept some input and produce some output.

1. **Requirements**

Design a 4-bit full adder from several 1-bit full adder circuits using a structural implementation style. The circuit has three inputs: one labeled X, one labeled Y, and one labeled CarryIn. The circuit will also have two outputs: S and CarryOut. The figure of this circuit can be seen below:

A picture containing text

Description automatically generated

**Figure 1:** Diagram for the circuit to be designed.

1. **Prelab**

For this prelab, we were required to analyze the working of a 1-bit full adder, write the truth table for a 1-bit full adder, and draw the circuit diagram for the 4-bit full adder in terms of a 1-bit full adder.

Truth table for 1-bit full adder:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Inputs | | | Outputs | |
| X | Y | Carry-In | Sum | Carry-Out |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Table 1:** K-Map for sum

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| AB C | 00 | 01 | 11 | 10 |  |
| 0 | 0 | 1 | 0 | 1 | Sum = X0’X1’X2 + X0’X1X2’ + X0X1X2 + X0X1’X2’ | |
| 1 | 1 | 0 | 1 | 0 |  |

**Table 2:** K-Map for Carry-out

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| AB C | 00 | 01 | 11 | 10 |  |
| 0 | 0 | 0 | 1 | 0 | Y0 = X1X2 + X0X1 + X0X2 | |
| 1 | 0 | 1 | 1 | 1 |  |

4-Bit full adder circuit:

B3 A3

B0 A0

B1 A1

B2 A2

C4

C3 3

C2

C1

C0

Full-adder

Full-adder

Full-adder

Full-adder

S3 3

S2

S1

S0

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 1-bit adder and added the necessary inputs and outputs to it. We then coded the S and Cout equations for the 1-bit adder. After that, we created another design source file for the 4-bit adder and added the necessary inputs and outputs to it along with the component instantiation, the adder logic, and interface signal declarations. Finally, 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**

------1-bit adder

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

--Input and Output declarations

entity Lab4Design is

Port ( X : in STD\_LOGIC;

Y : in STD\_LOGIC;

CIn : in STD\_LOGIC;

S : out STD\_LOGIC;

COut : out STD\_LOGIC);

end Lab4Design;

architecture Behavioral of Lab4Design is

begin

S <= X XOR Y XOR CIn; --Logic for the sum, simplified using XOR gates

COut <= (Y AND CIn) OR (X AND Y) OR (X AND CIn); --Logic for the carry value

end Behavioral;

------4-bit adder

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

--Input and Output declarations

entity Lab4Design2 is

Port ( X : in STD\_LOGIC\_VECTOR (3 downto 0);

Y : in STD\_LOGIC\_VECTOR (3 downto 0);

CIn : in STD\_LOGIC;

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

COut : out STD\_LOGIC);

end Lab4Design2;

architecture Structural of Lab4Design2 is

component Lab4Design is--Instantiate the component

Port ( X : in STD\_LOGIC;

Y : in STD\_LOGIC;

CIn : in STD\_LOGIC;

S : out STD\_LOGIC;

COut : out STD\_LOGIC);

end component;

signal temp: std\_logic\_vector(2 downto 0);--Defined signals

begin

--Calling support design file for adder logic

full\_adder\_0: Lab4Design port map (X(0), Y(0), CIn, S(0), temp(0));

--adder 1 will use temp(0) as its carry, temp(1) is sending the carry over to adder 2

full\_adder\_1: Lab4Design port map (X(1), Y(1), temp(0), S(1), temp(1));

--adder 2 will use temp(1) as its carry, temp(2) is sending the carry over to adder 3

full\_adder\_2: Lab4Design port map (X(2), Y(2), temp(1), S(2), temp(2));

--adder 3 will use temp(2) as its carry, sends COut as the carry output

full\_adder\_3: Lab4Design port map (X(3), Y(3), temp(2), S(3), COut);

--With these adders, we create a 4-bit adder composed of 4 1-bit adders

end Structural;

**4.2 Schematics**

**Diagram

Description automatically generated**

**Figure 2:** 4-Bit Adder Circuit Diagram

**4.3 Testbench**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Lab4Sim is

-- Port ( );

end Lab4Sim;

architecture Behavioral of Lab4Sim is

component Lab4Design2 is--Instantiate the component

Port ( X : in STD\_LOGIC\_VECTOR (3 downto 0);

Y : in STD\_LOGIC\_VECTOR (3 downto 0);

CIn : in STD\_LOGIC;

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

COut : out STD\_LOGIC);

end component;

signal X, Y: std\_logic\_vector(3 downto 0);--Signal declarations

signal CIn: std\_logic;

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

signal COut: std\_logic;

begin

uut: Lab4Design2 PORT MAP(X, Y, CIn, S, COut);--Port maps

process

begin

X <= x"1";

Y <= x"2";

CIn <= '0';

wait for 10ns;--output shuold be 3

X <= x"2";

Y <= x"5";

CIn <= '1';

wait for 10ns;--output should be 8

X <= x"1";

Y <= x"3";

CIn <= '0';

wait for 10ns;--output should be 4

X <= x"0";

Y <= x"4";

CIn <= '1';

wait;--output should be 5

end process;

end Behavioral;

**4.4 Waveform / Results**

The waveform below shows that the code 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.

Graphical user interface

Description automatically generated

**Figure 3:** 4-Bit Adder Waveform

# Conclusion

In this lab, we were able to successfully code a 4-bit adder circuit in Xilinx Vivado by using the circuit we created in our prelab as a base for our code. These programs were made to be able to simulate the circuit and take in the inputs we created and produce the correct output. The result of this simulation is shown in our waveform where you can see that for every X, Y, and Carry inputs we entered, the correct S and COut output appeared.

# 