Digital Design Lab

ECE 316

Lab/Project 2

Full 4-bit adder

Group 1

Karysse Hay, Nikeem Dunkelly-Allen, Johnny Brown

University of Miami

9/30/22

Overview

The premise of this lab was to familiarize ourselves with the port map mechanism in VHDL, as well as learn how to use components defined in another file. Essentially, port mapping describes sub-components of the circuit as well as interconnections between these sub-components using the "port map" statement. In terms of how VHDL describes hardware dataflow, data flow itself describes how data moves through the circuit, and VHDL describes data flow using concurrent signal assignment (port map). The basis of this lab was to create an Unsigned Full 4-Bit Adder, which adds two 4-bit binary numbers and a carry-in (Cin) input. The output of this gave a 4-bit value of Sum and a single bit Cout which can also be known as Overflow. The essential usage of this 4-Bit Adder is to ripple each carry output to carry the input of the next single bit addition. Each single bit addition was performed with a Full 1-Bit Adder designed. While seemingly complex, this 4-Bit Adder VHDL code can easily be constructed by port mapping 4 1-Bit Full Adders.

Software and Hardware

- ModelSim
-XOR gates
-AND gates

Description

The first portion of this lab was to formulate the code for a 1-bit Full Adder. Using the format of the Full Adder entity in the slides, we were able to complete the behavior. The ports for the two inputs (X and Y) and Cin were already defined likewise as Cout and Sum. As this was a 1-bit Full Adder, we ensured that each input and output were all set to single bits.

For the 1-Bit Full Adder's architecture behavior, two wires were assigned. This was done to facilitate the setting up of the logic. It made for a simpler and more readable code. We first assigned wire A to be the result of the VHDL statement: "X XOR Y". This output was then sent to another XOR gate alongside input Cin. The VHDL statement for this operation was " A XOR Cin" and was assigned to output Sum. Another wire, B, was assigned to the product of A and Cin ("A AND Cin"). After this, the only output left was Cout. Cout was set to the VHDL statement "(X AND Y) OR B".

Included in this process was the editing of the given Do file for the 1-bit Full Adder. This file allowed for automated generated inputs so that we can test the code. For this, we first began by closing any existing simulations. We then compiled our VHDL file, and began the simulation.

After we successfully created and simulated the code for the 1-Bit Full Adder, we moved on to the creation of the 4-bit Full Adder. To begin this process, we first defined the entity of the 4-bit Full Adder, defining the ports, but this time some of them as a set of vectors given the addition of more components. The two inputs, X and Y were converted from 1-bit to 4-bit inputs. Sum was also changed from 1-bit to 4-bits. This allowed for the adder to take two 4-bit numbers and output one 4-bit result. We then moved onto the Architecture structure of the 4-bit Full Adder, first defining it as a component, and then defining the port maps required for it to function. Four port maps of 1-bit Full Adders were added. This ended the architecture structure, and successfully completed the creation of the 4-bit Full Adder code. We finally successfully simulated this 4-bit Full Adder with various values, concluding the experimental process of this lab.

Specifications

Name	Bit Length	Mode	Description
X	1	Input	X operand
Y	1	Input	Y operand
Cin	1	Input	Carry-on Input
Sum	1	Output	Sum of A and B
Cout	1	Output	Carry-on Output

Table 1: Interface Specifications for 1-Bit Full Adder

Name	Bit Length	Mode	Description
A	4	Input	A operand
B	4	Input	B operand
Cin	1	Input	Carry-on Input
Sum	4	Output	Sum of A and B
Cout	1	Output	Carry-on Output

Table 2: Interface Specifications for 4-Bit Full Adder

Design Synthesis

Cin	X	Y	Sum	Cout
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 3: Truth Table for 1-Bit Full Adder

	X				Y				Sum				Carry
Cin	X3	X2	X1	X0	Y3	Y2	Y1	Y0	S3	S2	S1	S0	Cout
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	1	0	0	1	0	0
0	0	0	1	0	0	0	1	0	0	1	0	0	0
0	0	0	1	1	0	0	1	1	0	1	1	0	0
0	0	1	0	0	0	1	0	0	1	0	0	0	0
0	0	1	0	1	0	1	0	1	1	0	1	0	0
0	0	1	1	0	0	1	1	0	1	1	0	0	0
0	0	1	1	1	0	1	1	1	1	1	1	0	0
0	1	0	0	0	1	0	0	0	0	0	0	0	1
0	1	0	0	1	1	0	0	1	0	0	1	0	1
0	1	0	1	0	1	0	1	0	0	1	0	0	1
0	1	0	1	1	1	0	1	1	0	1	1	0	1
0	1	1	0	0	1	1	0	0	1	0	0	0	1
0	1	1	0	1	1	1	0	1	1	0	1	0	1
0	1	1	1	0	1	1	1	0	1	1	0	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 4: Truth Table for 4-Bit Full Adder

Complete Diagram

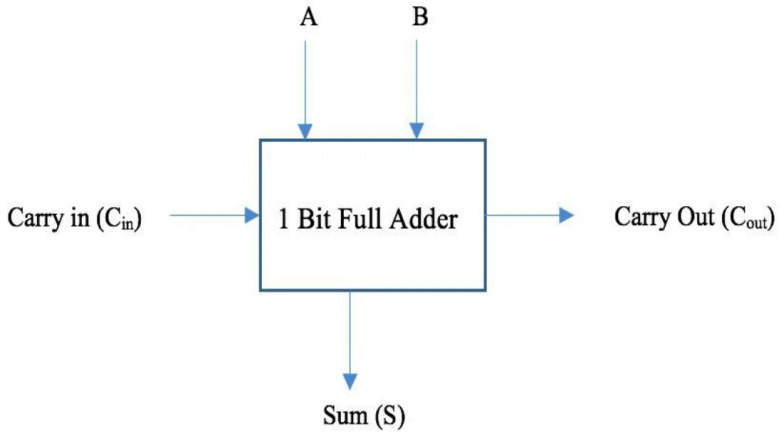

Figure 1: Block Diagram for 1-Bit Adder

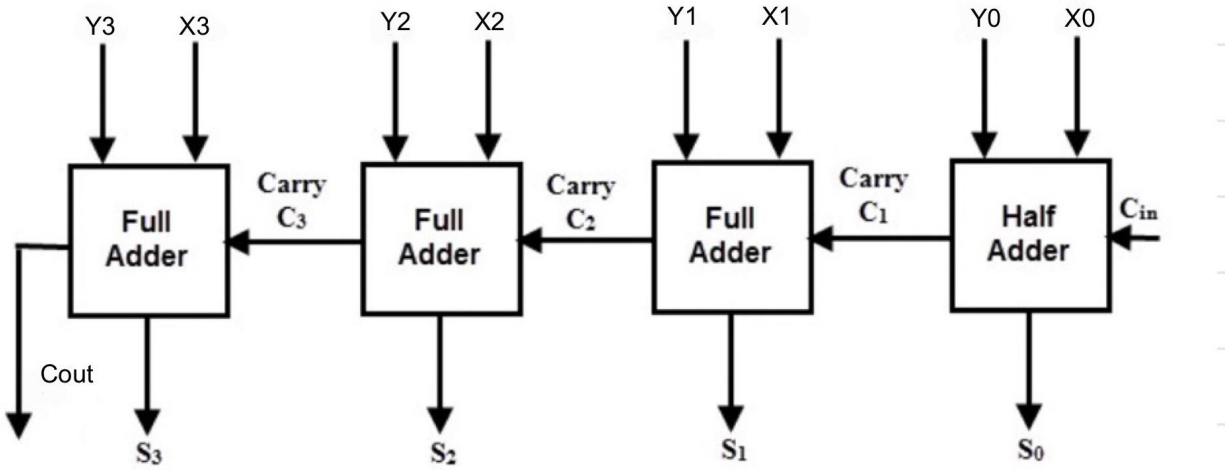

Figure 2: Block Diagram for 4-Bit Adder

Results and Simulations

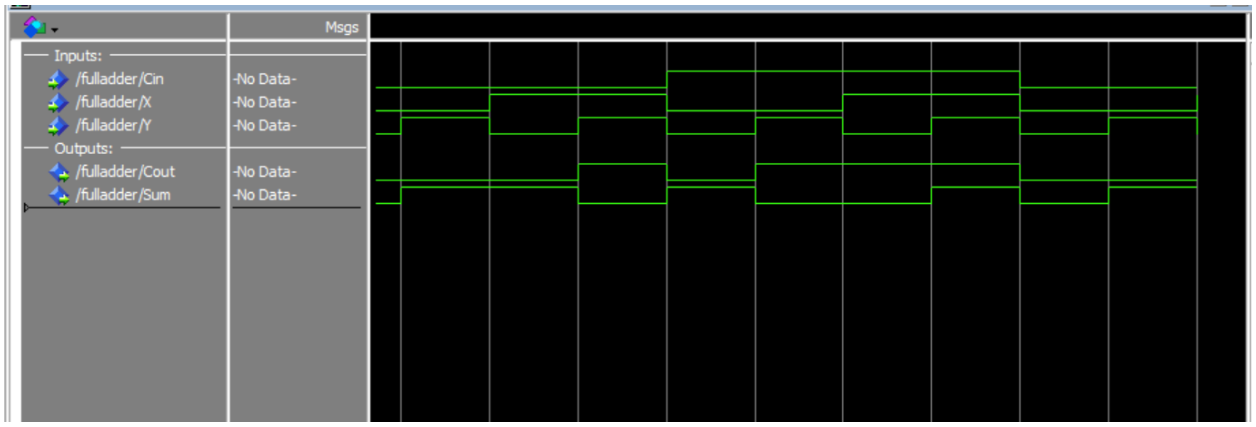

Figure 3: 1-Bit Full Adder Using Do Adder File

In Figure 3, it can be seen that when Cin is forced low, addition will take place. The do file created resulted in the same combinations listed in Table 1.

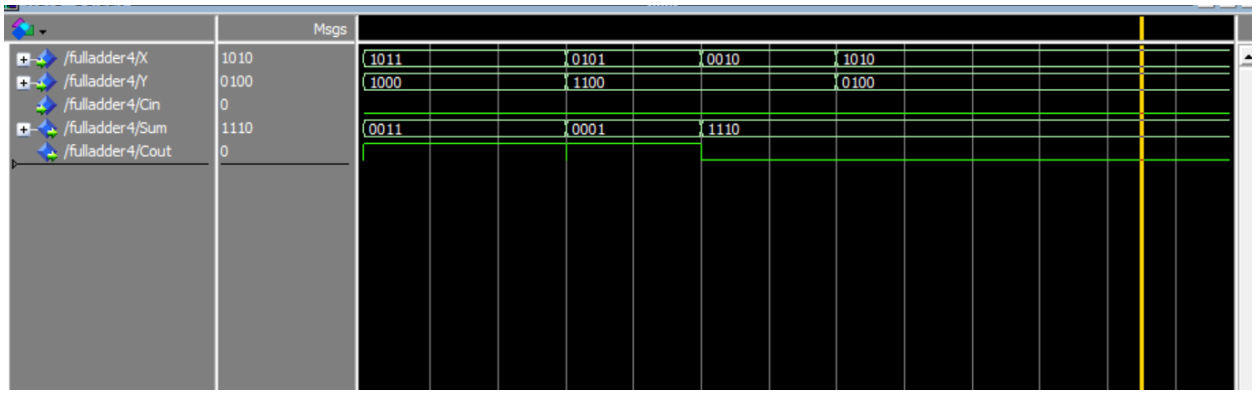

Figure 4: 4-Bit Full Adder

In Figure 4, some combinations for X and Y were set. Firsty, X equals 11_{10} (1011 in binary), Y equals 8_{10} (1000 in binary) and this resulted in the overflow bit being set and Sum equalling 3_{10} (0011 in binary). Another example is the last set. X equals 10_{10} (1010 in binary), Y equals 4_{10} (0100 in binary) which resulted in Sum equalling 14_{10} (1110 in binary) and the overflow bit forced low. For all of these examples Cin was forced low so that it can perform addition successfully..

Conclusion

In conclusion, we were able to successfully create and simulate a Full 1-Bit Adder. This 1-Bit Adder consisted of five gates: AND, XOR and OR. The do adder file provided was edited to suit. This allowed us to test the 1-Bit Adder with all of its combinations. After perfecting this, the 4-Bit Full Adder was designed. This 4-Bit Full adder properly adds two 4-bit binary numbers plus the Cin input, and gives the 4-bit Sum and a Cout output. Each single addition is performed with the full 1-bit adder operation input and output. The 4-Bit Full Adder was constructed by port mapping 4 x 1-Bit Full Adders. Both were successfully simulated with various values.

In the end, all of our final designs successfully simulated the required task. For this lab, we developed our abilities with the concurrent port map mechanism in VHDL, as well as how to use components defined in another file.

Appendix

Code for 1-Bit Full Adder

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


Entity FullAdder is
        port (X, Y, Cin: in bit;
                Cout, Sum: out bit);
End FullAdder;

Architecture behav of FullAdder is
        signal A, B: bit;
begin
        A <= X XOR Y;
        Sum <= A XOR Cin;
        B <= A AND Cin;
        Cout <= (X AND Y) OR B;


End behav;
```

Do File for 1-Bit Full Adder

```
# close any existing simulations
quit -sim

# compile our VHDL file
vcom FullAdder.vhd

# start the simulation - set timescale to nanoseconds
vsim -t ns FullAdder

# add the inputs to the wave
# first add a divider with a label
add wave -divider Inputs:
add wave Cin X Y

# add the outputs to the wave
# first add a divider with a label
add wave -divider Outputs:
add wave Cout Sum

# do initial run with default values - optional
run 10 ns

# force values
force Y 0 , 1 10 ns -r 20 ns
force X 0 , 1 20 ns -r 40 ns
force Cin 0 , 1 40 ns -r 80ns

# run
run 100 ns
```

Code for 4-Bit Full Adder

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

Entity FullAdder4 is
        port (X, Y : in bit_vector (3 downto 0);
                Cin : in bit;
                Sum : out bit_vector (3 downto 0);
                Cout : out bit);
End FullAdder4;
```

```
Architecture struct of FullAdder4 is
        Component FullAdder
                port (X, Y, Cin : in bit;
                        Cout, Sum : out bit);
        End component;

        Signal C : bit_vector(3 downto 1);
        Begin
                FA0 : FullAdder port map (X(0), Y(0), Cin, C(1), Sum(0));
                FA1 : FullAdder port map (X(1), Y(1), C(1), C(2), Sum(1));
                FA2 : FullAdder port map (X(2), Y(2), C(2), C(3), Sum(2));
                FA3 : FullAdder port map (X(3), Y(3), C(3), Cout, Sum(3));
End struct;
```