Project 2

32-bit Full Adder Design Using VHDL

failure of this course.

Acknowledgment:	I acknowledge all works, including figures, codes, and writings, belong to me and/or persons who are referenced. I understand if any similarities in the code, comments, customized program behavior, report writings, and/or figures are found, both the helper (original work) and the requestor (duplicated/modified work) will be called for academic disciplinary action.

Abheek Mondal

28-Oct-2022

Abheek Mondal

ECE 485

Due Date: October 28, 2022

Dr. Won-Jae Yi

TABLE OF CONTENTS

ABSTRACT

The purpose of Project 2 is to prepare for Project 3, in which we will design and implement a custom 32-bit RISC processor, a stripped-down MIPS processor. The goal of this project is to get familiar with VHDL programming, as well as the simulation environment.

In this project, we will design and implement a 32-bit adder, one of the basic functionalities of an ALU (Arithmetic Logic Unit).

INTRODUCTION

A ripple carry adder (RCA) or carry ripple adder (CRA) is a combinational logic circuit that is used to perform the addition of two n-bit binary numbers. RCA requires n-Full Adders in its circuit to add two n-bit binary numbers. To perform the addition of two 4-bit binary numbers, the two outputs from the 1st Full Adder provides the least significant bit (LSB) of the Bit 0 sum (S) and also a carry (CarryOut) bit which acts as the carry-in (CarryIn) bit of the 2nd Full Adder

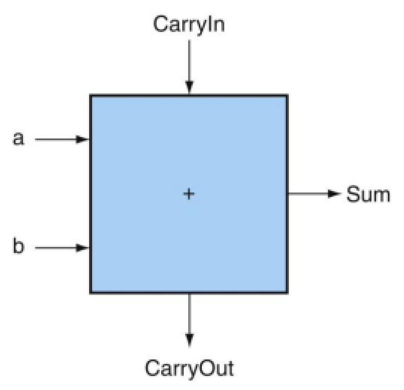


Figure 1. 1-bit Full Adder

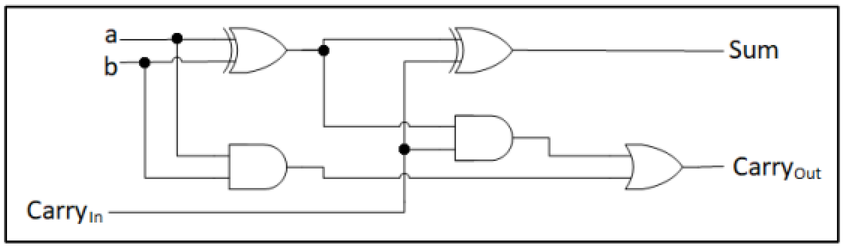
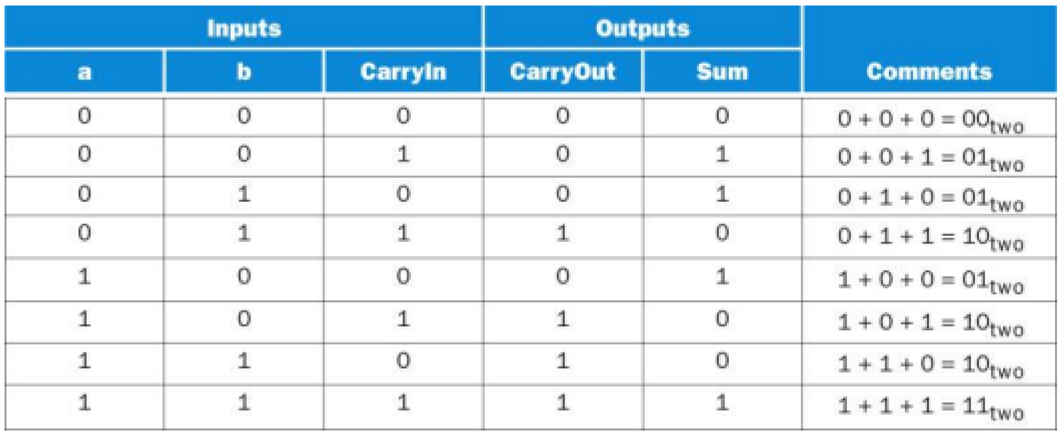


Figure 2. 1-bit Full Adder Gate Implementation



Inputs			Outputs		Comments
a	b	CarryIn	CarryOut	Sum	
0	0	0	0	0	$0 + 0 + 0 = 00_{two}$
0	0	1	0	1	$0 + 0 + 1 = 01_{two}$
0	1	0	0	1	$0 + 1 + 0 = 01_{two}$
0	1	1	1	0	$0 + 1 + 1 = 10_{two}$
1	0	0	0	1	$1 + 0 + 0 = 01_{two}$
1	0	1	1	0	$1 + 0 + 1 = 10_{two}$
1	1	0	1	0	$1 + 1 + 0 = 10_{two}$
1	1	1	1	1	$1 + 1 + 1 = 11_{two}$

Figure 3. Truth Table of 1-bit Full Adder

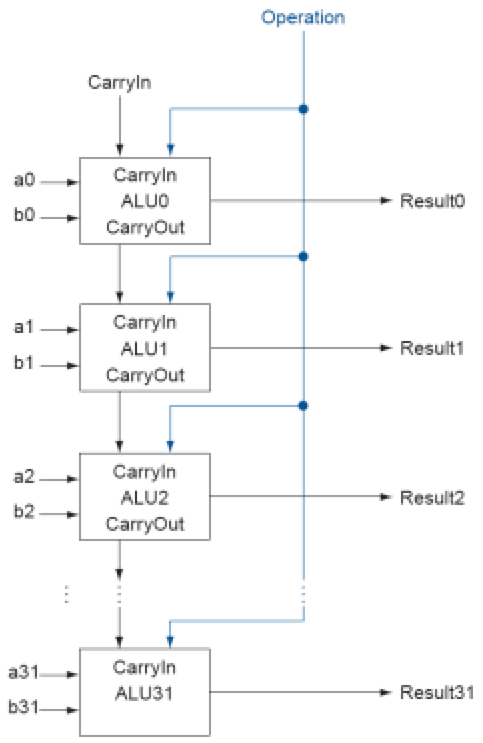


Figure 4. 32-bit ALU

BACKGROUND

For VDHL Structural Modelling, the components of the system are listed and the interconnections between them are specified. In this modeling the designs are described in the form of block diagrams. Components represented by blocks are interconnected by lines representing signals. VHDL Structural modeling code should have the ability to define the list of components, define a set of signals, able to uniquely label the component and specify signals to ports.

For VDHL Behavioral Modelling, it describes how the circuit should behave. It is widely used in test bench design since the test bench design doesn't care about the hardware realization. It is also written in VDHL code.

VDHL Testbench consists of non-synthesizable VHDL code which generate inputs to the design and checks that the outputs are correct. The stimulus block generates the inputs to the FPGA design and a separate block checks the outputs. The architecture of the testbench must contain an instantiation of the design under test (DUT).

SYSTEM DESIGN

In the RCA, each Full Adder's sum can only be determined after the carry generated by the previous stage Full Adder is produced. Therefore, the complete n-bit RCA result will only be ready once the carry has been rippled through from the LSB Full Adder to the MSB Full Adder. This causes a considerable delay in the n-bit RCA output.

The key to speeding up addition is to determine the carry into the higher order bits sooner so that those carry bits would be ready sooner to produce the sum result faster. For faster addition, Carry Lookahead Adder (CLA) can be used where carries are computed in advance, based on the input values.

The carry logic equation is

$$c_{i+1}=a_ib_i+(a_i+b_i)c_i$$

where a_ib_i is defined as a *generate signal*, and a_i+b_i is defined as a *propagate signal*. Thus, the carry logic equation can be re-written as

$$c_{i+1}=g_i+p_ic_i$$

For a 4-bit CLA, carries are calculated as the following:

$$c_1=g_0+p_0c_0$$

$$c_2=g_1+p_1c_1=g_1+p_1(g_0+p_0c_0)$$

$$=g_1+p_1g_0+p_1p_0c_0$$

$$c_3=g_2+p_2c_2=g_2+p_2(g_1+p_1g_0+p_1p_0c_0)$$

$$=g_2+p_2g_1+p_2p_1g_0+p_2p_1p_0c_0$$

$$c_4=g_3+p_3c_3=g_3+p_3(g_2+p_2g_1+p_2p_1g_0+p_2p_1p_0c_0)$$

$$=g_3+p_3g_2+p_3p_2g_1+p_3p_2p_1g_0+p_3p_2p_1p_0c_0$$

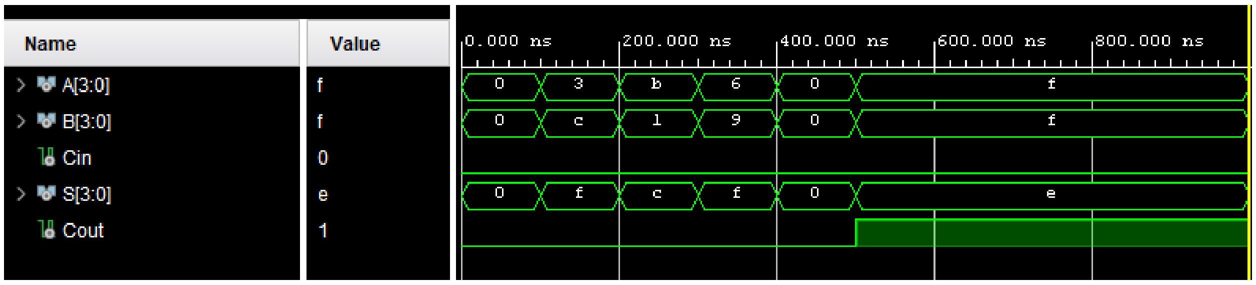
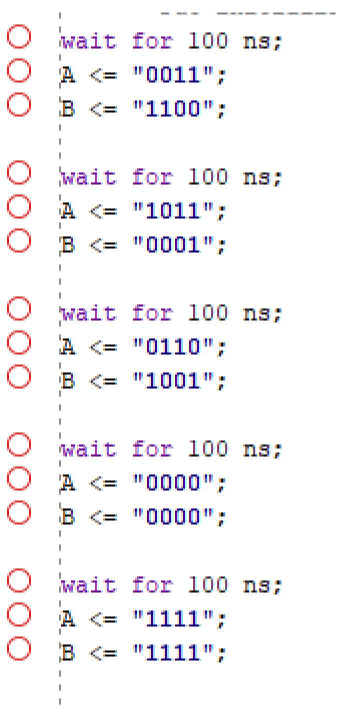


Figure 1 Working Adder



```
wait for 100 ns;
A <= "0011";
B <= "1100";

wait for 100 ns;
A <= "1011";
B <= "0001";

wait for 100 ns;
A <= "0110";
B <= "1001";

wait for 100 ns;
A <= "0000";
B <= "0000";

wait for 100 ns;
A <= "1111";
B <= "1111";
```

Figure 2 Test Cases

From the screenshots above, we can see that my code compiled with no errors and actually works this time.

CONCLUSION

Given the test cases in figure 2, we can see that in Figure 1, the test cases were successful. Which means that the 4-input ripple adder works perfectly fines, which means this project was a great success. I used the same resources as last time to guide me with building the test bench. Besides that, I didn't really encounter many problems, except to figure out how to define the structural behavior of a 1-byte adder.

REFRENCES

- https://fpgatutorial.com/how-to-write-a-basic-testbench-using-vhdl/
- https://surf-vhdl.com/vhdl-syntax-web-course-surf-vhdl/vhdl-behavioral-modeling-style/
- https://www.doulos.com/knowhow/perl/vhdl-testbench-creation-using-perl/
- https://www.youtube.com/watch?v=ShjXQdKdxsE

APPENDIX

VHDL_PORT_CODE

```
----------------------------------------------------------------------------------
-- Company: Illinois Institute of Technology
-- Engineer: Abheek Mondal
--
-- Create Date: 10/28/2022 06:32:32 AM
-- Design Name: VDHL port code single unit adder
-- Module Name: vhdl_port_code - 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;

-- 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 vhdl_port_code is
--  Port ( );
Port(A, B, Cin : in STD_LOGIC; --inputs
S, Cout : out STD_LOGIC); --Outputs

end vhdl_port_code;
```

```
architecture Behavioral of vhdl_port_code is

begin
process (A, B, Cin) is
begin

S <= (A XOR B) XOR Cin;
Cout <= (A AND B) OR ((A XOR B) AND Cin);

end process;

end Behavioral;
```

Code for Full_Adder

```
----------------------------------------------------------------------------------
-- Company: Illinois Institute of Technology
-- Engineer: Abheek Mondal
--
-- Create Date: 10/28/2022 03:46:11 AM
-- Design Name: 4 bit Ripple Carry Adder VHDL Code
-- Module Name: Full_Adder - Behavioral
-- Project Name: ECE 485 Project 2
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


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 Full_Adder is
--  Port ( );
Port (
A : in STD_LOGIC_VECTOR (3 downto 0); --Input 1
B : in STD_LOGIC_VECTOR (3 downto 0); --Input 2
Cin : in STD_LOGIC; --Carry from previous adder
S : out STD_LOGIC_VECTOR (3 downto 0); -- Sum from each adder
Cout : out STD_LOGIC);  --Carry signal
end Full_Adder;

architecture Behavioral of Full_Adder is

component vhdl_port_code
Port (
A : in STD_LOGIC;
B : in STD_LOGIC;
Cin : in STD_LOGIC;
S : out STD_LOGIC;
Cout : out STD_LOGIC);
end component;

signal c1,c2,c3: STD_LOGIC; -- Immediate Carry (need 3, because 4 adders)


begin
--All this is mapping outputs to ports, the weird kind of recursiveness in VDHL. Define Function then
copy paste
Adder1: vhdl_port_code port map( A(0), B(0), Cin, S(0), c1); --Carry signal in, its the start of the first full
adder
Adder2: vhdl_port_code port map( A(1), B(1), c1, S(1), c2);
Adder3: vhdl_port_code port map( A(2), B(2), c2, S(2), c3);
Adder4: vhdl_port_code port map( A(3), B(3), c3, S(3), Cout); --Carry signal out, its the end of the adder
series


end Behavioral;
```

Code for TESTBENCH

```
-- Testbench created online at:
--   https://www.doulos.com/knowhow/perl/vhdl-testbench-creation-using-perl/
-- Copyright Doulos Ltd

library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity Full_Adder_tb is
end;

architecture bench of Full_Adder_tb is

  component Full_Adder
  Port (
  A : in STD_LOGIC_VECTOR (3 downto 0);
  B : 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;

  --Inputs
signal A : std_logic_vector(3 downto 0) := (others => '0');
signal B : std_logic_vector(3 downto 0) := (others => '0');
signal Cin : std_logic := '0';

--Outputs
signal S : std_logic_vector(3 downto 0);
signal Cout : std_logic;


begin

  uut: Full_Adder port map ( A    => A,
                             B    => B,
                             Cin  => Cin,
                             S    => S,
                             Cout => Cout );

  stimulus: process
  begin

    -- Put initialisation code here
```

```
wait for 100 ns;
A <= "0011";
B <= "1100";

wait for 100 ns;
A <= "1011";
B <= "0001";

wait for 100 ns;
A <= "0110";
B <= "1001";

wait for 100 ns;
A <= "0000";
B <= "0000";

wait for 100 ns;
A <= "1111";
B <= "1111";


    -- Put test bench stimulus code here

    wait;
  end process;


end;
```