**EENG 2910 Project III – Digital System Design**

Design Project 1

Nathan A Ruprecht

Due 14 September 2016

# Introduction:

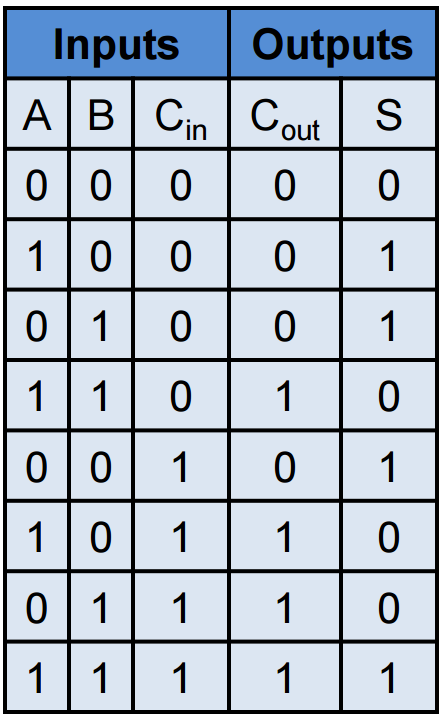
Combinational logic circuits are circuits whose outputs depend only on the present value of their inputs. The lecture on “Combinational Logic Design” introduced us to some of the common combinational logic circuits.

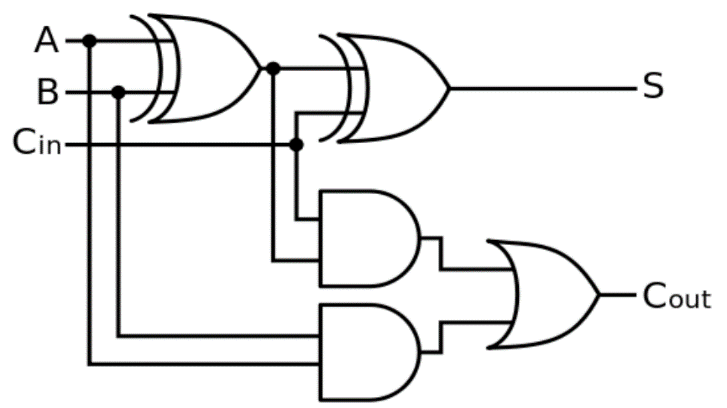
For our first project, we were required to design, implement, and test an 8-bit adder. The idea of an adder is a component that adds 2, 1-bit numbers together. If both inputs are 1 and output should be 10, the answer shows 0 since it holds just 1 bit. So we make a signal inside that is the “carry” for the overflow value. With the carry value used, going both in and out of the adder to account for adders before and after this particular one, we now have a full adder that can successfully show all answers for 2, 1-bit inputs. Put “n” of these full adders in series with the overflow going into the next adder, and we have an n-bit adder.

Since the ALU performs math functions, and simple math is just a matter of adding, this project is a stepping stone to what will come next in the semester. The full adder is also a simple way to be introduced to the style of coding and dealing with this type of class.

# Theory of Operation and Explanation of the Design:

As mentioned in the introduction, the key to this assignment is the full adder. The full adder takes 2, 1-bit digits and can output the 2-bit answer if there is carry over. It’s a matter of copy and pasting the full adder to make an n-bit adder.





Figure

The logic gate above along with the truth table show that the full adder can take 2, 1-bit digits, along with a carry in if applicable, and add them all together to give the appropriate answer.

Table

Now that I have the basis for the lab, I copy and paste the necessary component in series with the carries feeding into the next adder. A visual representation of the 8-bit adder is shown below with ‘a’ and ‘b’ being the inputs, ‘c’ being the carry, and ‘s’ being the sum.

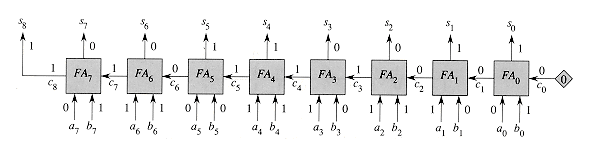


Figure 2

# Experimental Results:

I simply hard coded examples in my test bench to simulate adding 2, 8-bit numbers. I tested it numerous times and made the simulation show the numbers in decimal format just to make it easier to see if it worked or not. Below are screenshots of the simulation.

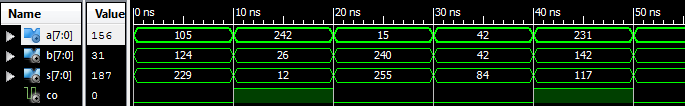


Figure 3

The screenshot above shows the output of the test bench mentioned. “a” being the first 8-bit number, “b” is the second number, and “s” as the sum. Looking at each column, there’s a successful 8-bit addition of ‘a’ and ‘b’. Since it’s an 8-bit number, the max the adder should hit is 255 (2^8 including 0). I tested it by intentionally adding number that would go over 255. My thought proved true when I added 242 with 26. The answer is 268, but it went over 255 so it reset and went over by 12 (the answer given). I did it again later when adding 231 with 142 to show the same thought process.

# Conclusion:

The goal of the lab was to design, implement, and test an 8-bit adder. In the design process, I understood how a half adder was made using logic gates. Put two half adders together gives me a full adder that accounts for carry over. 8, full adders put in series gives me an 8-bit adder. As seen in my code, I copy and pasted the implemented full adder 8 times with the “carry” signal feeding into the next adder. This successfully gave results of a working 8-bit adder.

# References:

Figure 1 and Table 1 from class lecture PowerPoint

Figure 2:

<http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap29.htm>

Figure 3 screenshot from Xilinx test bench

# Source Code:

----------------------------------------------------------------------------------

-- Company:

-- Engineer: Nathan A Ruprecht

--

-- Create Date: 15:38:53 09/07/2016

-- Design Name:

-- Module Name: adder\_8bit - Behavioral

-- Project Name: Design Project 1

-- Target Devices:

-- Tool versions:

-- Description: Design, implement, and test an 8-bit adder

--

-- 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 primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity adder\_8bit is

port (a: in std\_logic\_vector(7 downto 0);

b: in std\_logic\_vector(7 downto 0);

s: out std\_logic\_vector(7 downto 0);

co: out std\_logic

);

end adder\_8bit;

architecture Behavioral of adder\_8bit is

component full\_adder is

Port ( a, b, c\_in : in STD\_LOGIC;

sum, c\_out : out STD\_LOGIC);

end component;

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

begin

ci(0) <= '0';

full\_adder0:full\_adder

port map( a=>a(0), b=>b(0), c\_in=>ci(0), sum=>s(0), c\_out=> ci(1) );

full\_adder1:full\_adder

port map( a=>a(1), b=>b(1), c\_in=>ci(1), sum=>s(1), c\_out=> ci(2) );

full\_adder2:full\_adder

port map( a=>a(2), b=>b(2), c\_in=>ci(2), sum=>s(2), c\_out=> ci(3) );

full\_adder3:full\_adder

port map( a=>a(3), b=>b(3), c\_in=>ci(3), sum=>s(3), c\_out=> ci(4) );

full\_adder4:full\_adder

port map( a=>a(4), b=>b(4), c\_in=>ci(4), sum=>s(4), c\_out=> ci(5) );

full\_adder5:full\_adder

port map( a=>a(5), b=>b(5), c\_in=>ci(5), sum=>s(5), c\_out=> ci(6) );

full\_adder6:full\_adder

port map( a=>a(6), b=>b(6), c\_in=>ci(6), sum=>s(6), c\_out=> ci(7) );

full\_adder7:full\_adder

port map( a=>a(7), b=>b(7), c\_in=>ci(7), sum=>s(7), c\_out=> co );

end Behavioral;