**ELEC 204 Digital Design Lab Report**

Lab 03

Name: Umur Demircioğlu

Date: 08/11/2019

1. **Introduction and objectives**

In this project we aimed to create a Arithmetic Logic Unit which is in short ALU. We used a divide and conquer method to implement our design. We first designed the Logic Unit and the Arithmetic Unit separately and then combined them. I wrote the truth table which was given using logic gates and the logic behind full adder.

1. **Methods**

I first designed the Logic Unit which was less complicated compared to the Arithmetic

Unit. Logic Unit worked when M(1 bit input) was 0 we had 2 select bits(1 bit inputs) and A(4 bit),

B(4 bit) inputs. We coded the logic for every A and B bit separately. When S1= 0 and S1= 0 we used

logic operator AND for A(0) and B(0) (Same for every other bit). When select bits changed our system

used the logic operator that our truth table asked us to do.

For the Arithmetic part again we designed for every bit separately and the arithmetic part worked

when M=1. From the truth table I realized that the Arithmetic part was grouped 2 by 2 . For example

for Transfer A and Increment A by 1 select bits are the same but C0 is 1 for Increment A by 1. So we

can see that if we use XOR gate to connect C0 to our Transfer A design if C0 is 0 it will transfer a but

if C0 is 1 with Full Adder logic it will add 1 to A. For the next to it is the same logic for S1=0,S0=1 we

when C0=0 we want to add A and B which is XOR(The carry part is different part of code) and the one

below that is A+B+1 which is again adding 1 to our A+B so we use XOR gate with C0 because C0 is

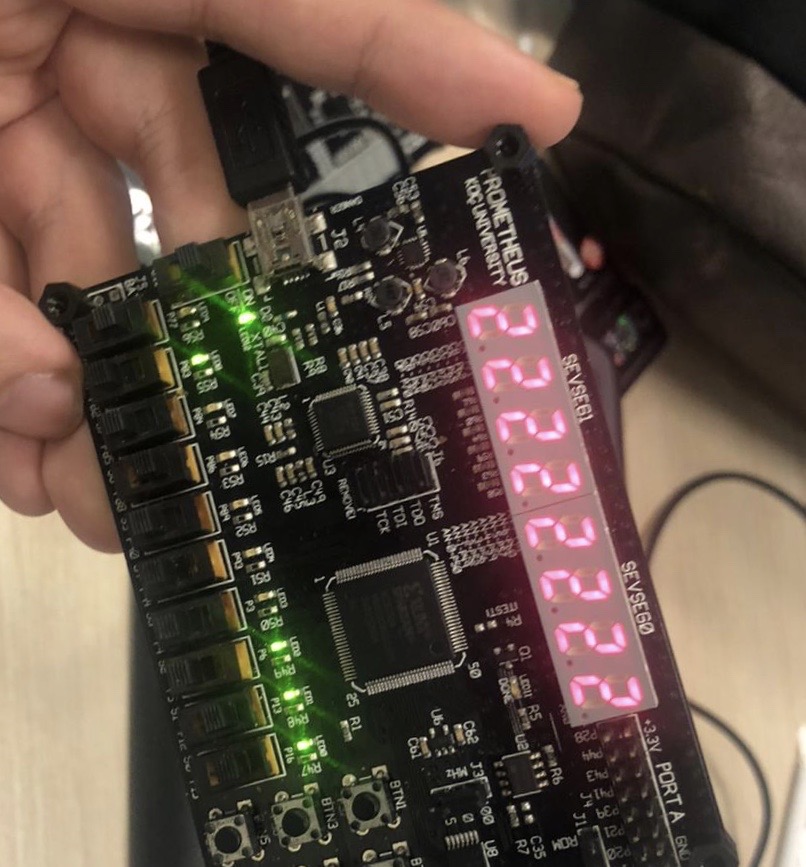
1 for A+B+1. This logic goes for the rest of the truth table. I didn’t mention the carry when doing arithmetic operations for two numbers for example when we add A(0) and B(0) the carry that we will have we have to add to the 2nd bit which is A(1)+B(1)+Cout(0)(Carry of the LSB) again using XOR gate. The last carry that we calculated was our signbit. Then we showed our work on the FPGA Board and seven segment display.

1. **Problems encountered, errors and warnings resolved**

In this lab I had some problems implementing the Seven Segment to my project because I wrote the truth table directly with logic gates in one project but I couldn’t understand the Seven Segment which was given.

1. **Conclusion**

In this experiment, I learned how to do arithmetic operations on FPGA board and how to show the results on the Seven Segment Display. I saw the logic behind Arithmetic Logic Unit and got more comfortable using logic operators.

entity main is  
Port ( M : in STD\_LOGIC;  
  
S1 : in STD\_LOGIC;  
  
S0 : in STD\_LOGIC;  
  
A : in STD\_LOGIC\_VECTOR(3 downto 0);  
  
B : in STD\_LOGIC\_VECTOR(3 downto 0);  
  
Cin : in STD\_LOGIC;  
  
CoutFinal : out STD\_LOGIC;  
  
SevenSegment : out STD\_LOGIC\_VECTOR(6 downto 0);  
  
Signbit : out STD\_LOGIC;  
  
L : out STD\_LOGIC\_VECTOR(3 downto 0));  
  
end main;  
  
architecture Behavioral of main is  
  
signal Lout : STD\_LOGIC\_VECTOR(3 downto 0);  
  
signal ARout : STD\_LOGIC\_VECTOR(3 downto 0);  
  
signal Cout0 : STD\_LOGIC;  
  
signal Cout1 : STD\_LOGIC;  
  
signal Cout2 : STD\_LOGIC;  
  
begin  
Lout(0) <=(((not S0) AND (not S1) AND (A(0) AND B(0))) OR ((S0) AND (not S1) AND (A(0) OR B(0)))) OR (((not S0) AND (S1) AND (A(0) XOR B(0))) OR ((S0) AND (S1) AND (A(0) XNOR B(0))));  
Lout(1) <=(((not S0) AND (not S1) AND (A(1) AND B(1))) OR ((S0) AND (not S1) AND (A(1) OR B(1)))) OR (((not S0) AND (S1) AND (A(1) XOR B(1))) OR ((S0) AND (S1) AND (A(1) XNOR B(1))));  
Lout(2) <=(((not S0) AND (not S1) AND (A(2) AND B(2))) OR ((S0) AND (not S1) AND (A(2) OR B(2)))) OR (((not S0) AND (S1) AND (A(2) XOR B(2))) OR ((S0) AND (S1) AND (A(2) XNOR B(2))));  
Lout(3) <=(((not S0) AND (not S1) AND (A(3) AND B(3))) OR ((S0) AND (not S1) AND (A(3) OR B(3)))) OR (((not S0) AND (S1) AND (A(3) XOR B(3))) OR ((S0) AND (S1) AND (A(3) XNOR B(3))));  
ARout(0) <=(((not S0) AND (not S1) AND (A(0) XOR Cin)) OR ((S0) AND (not S1) AND ((A(0) XOR B(0)) XOR Cin))) OR (((not S0) AND (S1) AND ((A(0) XOR (not B(0))) XOR Cin)) OR ((S0) AND (S1) AND (((not A(0)) XOR B(0)) XOR Cin)));  
Cout0 <=(((not S0) AND (not S1) AND ((A(0)AND Cin))) OR ((S0) AND (not S1) AND ((A(0) AND B(0)) OR ((A(0) XOR B(0))AND Cin)))) OR (((not S0) AND (S1) AND ((A(0) AND (not B(0))) OR ((A(0) XOR (not B(0)))AND Cin))) OR ((S0) AND (S1) AND (((not A(0)) AND B(0)) OR (((not A(0)) XOR B(0))AND Cin))));  
ARout(1) <=(((not S0) AND (not S1) AND (A(1) XOR Cout0)) OR ((S0) AND (not S1) AND ((A(1) XOR B(1)) XOR Cout0))) OR (((not S0) AND (S1) AND ((A(1) XOR (not B(1))) XOR Cout0)) OR ((S0) AND (S1) AND (((not A(1)) XOR B(1)) XOR Cout0)));  
Cout1 <=(((not S0) AND (not S1) AND ((A(1) AND Cout0))) OR ((S0) AND (not S1) AND ((A(1) AND B(1)) OR ((A(1) XOR B(1))AND Cout0)))) OR (((not S0) AND (S1) AND ((A(1) AND (not B(1))) OR ((A(1) XOR (not B(1)))AND Cout0))) OR ((S0) AND (S1) AND (((not A(1)) AND B(1)) OR (((not A(1)) XOR B(1))AND Cout0))));  
ARout(2) <=(((not S0) AND (not S1) AND (A(2) XOR Cout1)) OR ((S0) AND (not S1) AND ((A(2) XOR B(2)) XOR Cout1))) OR (((not S0) AND (S1) AND ((A(2) XOR (not B(2))) XOR Cout1)) OR ((S0) AND (S1) AND (((not A(2)) XOR B(2)) XOR Cout1)));  
Cout2 <=(((not S0) AND (not S1) AND ((A(2) AND Cout1))) OR ((S0) AND (not S1) AND ((A(2) AND B(2)) OR ((A(2) XOR B(2))AND Cout1)))) OR (((not S0) AND (S1) AND ((A(2) AND (not B(2))) OR ((A(2) XOR (not B(2)))AND Cout1))) OR ((S0) AND (S1) AND (((not A(2)) AND B(2)) OR (((not A(2)) XOR B(2))AND Cout1))));  
ARout(3) <=(((not S0) AND (not S1) AND (A(3) XOR Cout2)) OR ((S0) AND (not S1) AND ((A(3) XOR B(3)) XOR Cout2))) OR (((not S0) AND (S1) AND ((A(3) XOR (not B(3))) XOR Cout2)) OR ((S0) AND (S1) AND (((not A(3)) XOR B(3)) XOR Cout2)));  
CoutFİNAL <=(((not S0) AND (not S1) AND ((A(3) AND Cout2))) OR ((S0) AND (not S1) AND ((A(3) AND B(3)) OR ((A(3) XOR B(3))AND Cout2)))) OR (((not S0) AND (S1) AND ((A(3) AND (not B(3))) OR ((A(3) XOR (not B(3)))AND Cout2))) OR ((S0) AND (S1) AND (((not A(3)) AND B(3)) OR (((not A(3)) XOR B(3))AND Cout2))));  
  
L(0) <= Lout(0) OR ARout(0);  
L(1) <= Lout(1) OR ARout(1);  
L(2) <= Lout(2) OR ARout(2);  
L(3) <= Lout(3) OR ARout(3);  
signbit <= ARout(3);  
  
process(ARout)  
  
begin  
  
case ARout is  
  
when "0000" => SevenSegment <= "0000001";   
  
when "0001" => SevenSegment <= "1001111";  
  
when "0010" => SevenSegment <= "0010010";   
  
when "0011" => SevenSegment <= "0000110";   
  
when "0100" => SevenSegment <= "1001100";  
  
when "0101" => SevenSegment <= "0100100";   
  
when "0110" => SevenSegment <= "0100000";   
  
when "0111" => SevenSegment <= "0001101";   
  
when "1000" => SevenSegment <= "0000000";   
  
when "1001" => SevenSegment <= "0001101";  
  
when "1010" => SevenSegment <= "0100000";   
  
when "1011" => SevenSegment <= "0100100";   
  
when "1100" => SevenSegment <= "1001100";   
  
when "1101" => SevenSegment <= "0000110";   
  
when "1110" => SevenSegment <= "0010010";   
  
when "1111" => SevenSegment <= "1001111";   
  
when others => SevenSegment <= "1111111";   
  
end case;  
  
end process;  
  
  
end Behavioral;