ELEC 204 Digital Design Lab Report

Lab 2

Name: Atahan Kalaycı

Date: 20/11/2020

1. Introduction and objectives

The objective of this lab was getting familiar with the elementary logic gates by designing a 4-bit magnitude comparator for unsigned and 2's complement binary numbers. During the lab, I practiced my learnings related with combinational design and optimization methodologies.

Since the purpose of the lab was to design a 4-bit magnitude comparator for unsigned and 2's complement binary numbers, my code is implemented in a way to design the comparators. The magnitude comparators had to take two 4-bit binary numbers A and B for unsigned and C and D for signed numbers, compare them and give three output logic bits. The first output is generated for equality resulting in 1 if the inputs are equal and 0 otherwise. The second logic output is 1 if A is greater than B or if C is greater than D, 0 otherwise. Last logic output is 1 if A is less than B or C is less than B, 0 otherwise.

As a result, I generated truth tables for my two magnitude comparators respectively unsigned and 2's complement binary numbers, then I designed my comparator circuits using my truth tables and logic gates to get the desired outputs. My code gave correct logic outputs for A=B, A>B, A<C and C=D, C>D, C<D.

2. Methods

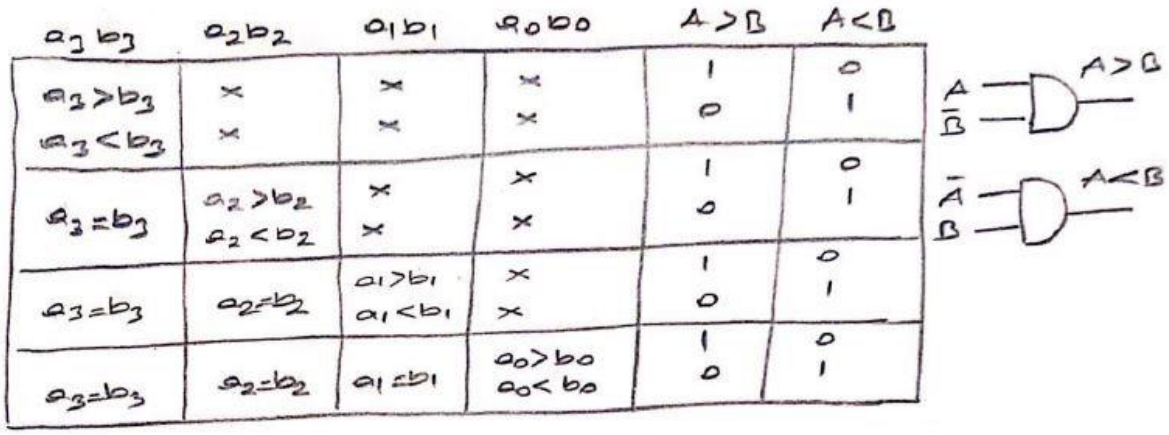
For the MComparator(Part 1), I had 8 inputs A0, A1, A2,A3 and B0, B1, B2, B3.
For the SComparator(Part 2), I also had 8 inputs C0, C1, C2, C3 and D0, D1, D2, D3.
All these inputs are 1-bit.

For the MComparator(Part 1), I had 3 outputs E, G_AB, and G_BA.
For the SComparator(Part 2), I also had 3 outputs E, G_CD, and G_DC.
All these inputs are 1-bit.

My VHDL code for both MComparator and SComparator takes two input as 4-bit binary numbers and compare them. As a result, it produces three logic output. Logic output 1 if equal, greater than, and less than conditions are met and 0 otherwise.

My code work by using XNOR, OR, AND, NOT logic gates. Since XNOR gate is a difference gate, it is used for the equality check. The rest of the is basically build on this equality checker to compare the inputs bit by bit. First, it compares every bit for equality and then it compares the most significant bit to all the way to the least significant bit to compares if the inputs are greater than or less than each other. Lastly, my code produces a 1 or 0 respectively for the outputs if conditions are satisfied.

The figure below illustrates the truth table I generated for the design of my circuits and the implementation for my code.



The signed comparator truth table is similar to this except that first it checks for the most significant bit. If the most significant bit is 1 the inputs is automatically smaller than the other input because one is a negative number and the other is a positive number. If they are equal, every bit is checked one by one similarly.

```
entity MComparator is
    Port ( AO : in  STD_LOGIC;
           A1 : in  STD_LOGIC;
           A2 : in  STD_LOGIC;
           A3 : in  STD_LOGIC;
           BO : in  STD_LOGIC;
           B1 : in  STD_LOGIC;
           B2 : in  STD_LOGIC;
           B3 : in  STD_LOGIC;
           E : out  STD_LOGIC;
           G_AB : out  STD_LOGIC;
           G_BA : out  STD_LOGIC);
end MComparator;

architecture Behavioral of MComparator is

begin

   E <= (A3 XNOR B3) AND (A2 XNOR B2) AND (A1 XNOR B1) AND (AO XNOR BO);

   G_AB <= (A3 AND (NOT B3)) OR
               ((A3 XNOR B3) AND A2 AND (NOT B2)) OR
                  ((A3 XNOR B3) AND (A2 XNOR B2) AND A1 AND (NOT B1)) OR
                     ((A3 XNOR B3) AND (A2 XNOR B2) AND (A1 XNOR B1) AND AO AND (NOT BO));

   G_BA <= ((NOT A3) AND B3) OR
               ((A3 XNOR B3) AND (NOT A2) AND B2) OR
                  ((A3 XNOR B3) AND (A2 XNOR B2) AND (NOT A1) AND B1) OR
                     ((A3 XNOR B3) AND (A2 XNOR B2) AND (A1 XNOR B1) AND (NOT AO) AND BO);
end Behavioral;
```

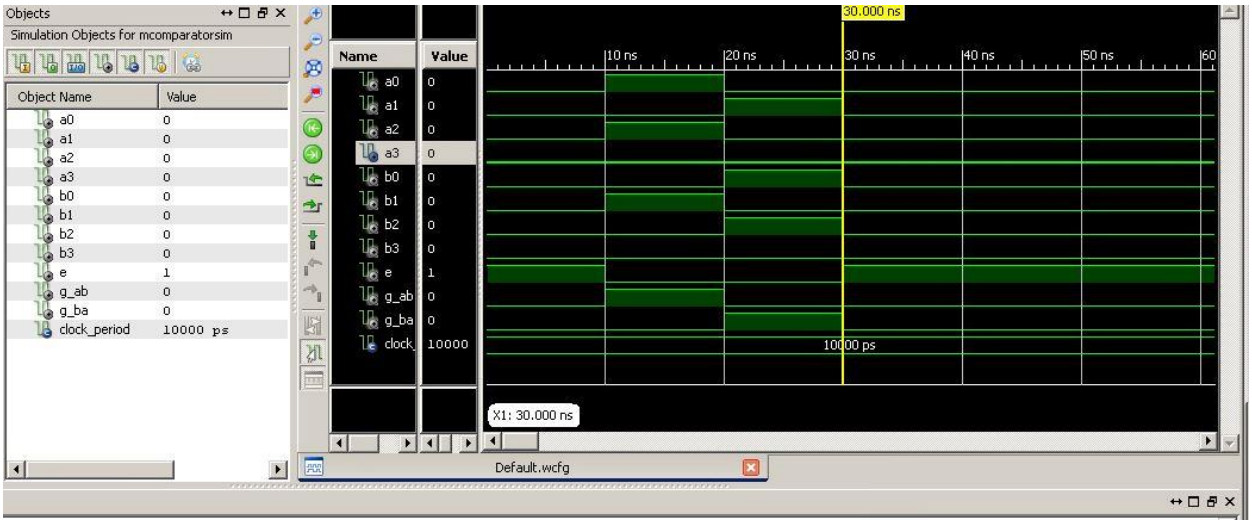
```
entity SComparator is
    Port ( CO : in  STD_LOGIC;
           C1 : in  STD_LOGIC;
           C2 : in  STD_LOGIC;
           C3 : in  STD_LOGIC;
           DO : in  STD_LOGIC;
           D1 : in  STD_LOGIC;
           D2 : in  STD_LOGIC;
           D3 : in  STD_LOGIC;
           E : out  STD_LOGIC;
           G_CD : out  STD_LOGIC;
           G_DC : out  STD_LOGIC);
end SComparator;

architecture Behavioral of SComparator is

begin
   E <= (C3 XNOR D3) AND (C2 XNOR D2) AND (C1 XNOR D1) AND (CO XNOR DO);



   G_CD <= ((D3 XNOR C3) AND (D2 XNOR C2) AND (D1 XNOR C1) AND (DO XNOR CO)) NOR
               ((C3 AND (NOT D3)) OR (D2 AND (NOT C2) AND (D3 XNOR C3)) OR (D1 AND (NOT C1)
                  AND (D3 XNOR C3) AND (D2 XNOR C2)) OR (DO AND (NOT CO) AND (D3 XNOR C3)
                     AND (D2 XNOR C2) AND (D1 XNOR C1)));

   G_DC <= (C3 AND (NOT D3)) OR (D2 AND (NOT C2) AND (D3 XNOR C3)) OR (D1 AND (NOT C1)
               AND (D3 XNOR C3) AND (D2 XNOR C2)) OR (DO AND (NOT CO) AND (D3 XNOR C3)
                  AND (D2 XNOR C2) AND (D1 XNOR C1));


end Behavioral;
```

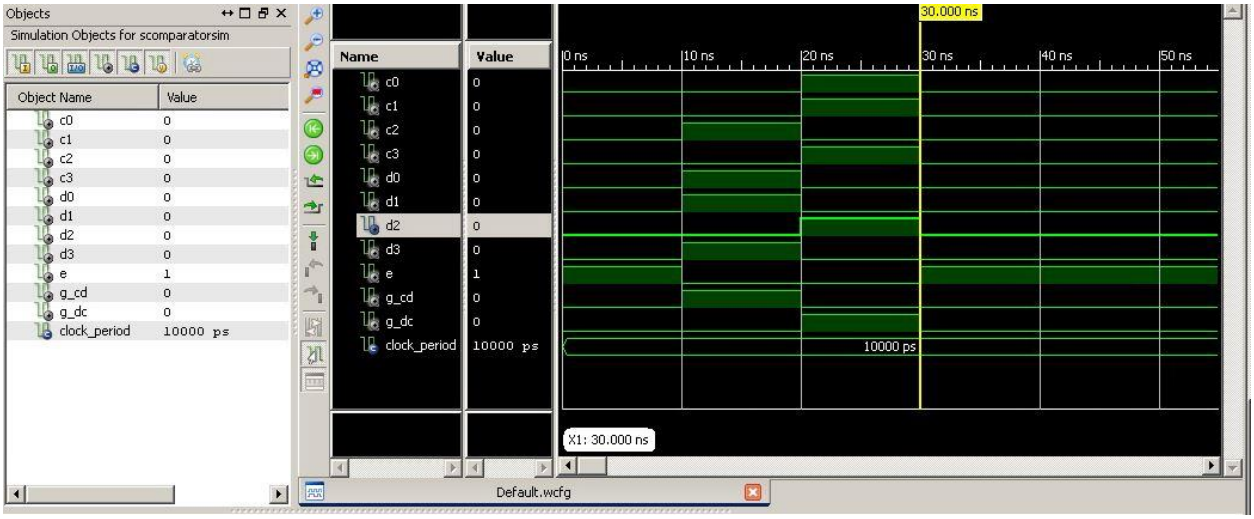
For my simulation, I have used the given test cases.

```
-- Clock process definitions
Clock_process :process
begin
   A3 <= '0'; A2 <= '0'; A1 <= '0'; A0 <= '0'; B3 <= '0'; B2 <= '0'; B1 <= '0'; B0 <= '0'; --(0000, 0000)
   wait for Clock_period;
   A3 <= '0'; A2 <= '1'; A1 <= '0'; A0 <= '1'; B3 <= '0'; B2 <= '0'; B1 <= '1'; B0 <= '0'; --(0101, 0010)
   wait for Clock_period;
   A3 <= '0'; A2 <= '0'; A1 <= '1'; A0 <= '0'; B3 <= '0'; B2 <= '1'; B1 <= '0'; B0 <= '1'; --(0010, 0101)
   wait for Clock_period;
   A3 <= '0'; A2 <= '0'; A1 <= '0'; A0 <= '0'; B3 <= '0'; B2 <= '0'; B1 <= '0'; B0 <= '0'; -- Returning initial state.
wait;
end process;


-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   wait for 100 ns;

   wait for Clock_period*10;

   -- insert stimulus here

   wait;
end process;

';
```

```
  -- Clock process definitions
  Clock_process :process
  begin
     C3 <= '0'; C2 <= '0'; C1 <= '0'; C0 <= '0'; D3 <= '0'; D2 <= '0'; D1 <= '0'; D0 <= '0'; --(0000, 0000)
     wait for Clock_period;
     C3 <= '0'; C2 <= '1'; C1 <= '0'; C0 <= '0'; D3 <= '1'; D2 <= '0'; D1 <= '1'; D0 <= '1'; --(0100, 1011)
     wait for Clock_period;
     C3 <= '1'; C2 <= '0'; C1 <= '1'; C0 <= '1'; D3 <= '0'; D2 <= '1'; D1 <= '0'; D0 <= '0'; --(1011, 0100)
     wait for Clock_period;
     C3 <= '0'; C2 <= '0'; C1 <= '0'; C0 <= '0'; D3 <= '0'; D2 <= '0'; D1 <= '0'; D0 <= '0'; -- Returning initial state
     wait;
  end process;


  -- Stimulus process
  stim_proc: process
  begin
     -- hold reset state for 100 ns.
     wait for 100 ns;

     wait for Clock_period*10;

     -- insert stimulus here

     wait;
  end process;

END;
```

As a result, my test cases give correct outputs in the simulation.





3. Problems encountered, errors and warnings resolved

When I implemented my codes and test them, I saw that my circuit design in the preliminary work for the SComparator(Part 2) gave wrong results. Therefore, I had to design a different circuit and implement a different code. Other than this problem, I encountered no synthesis errors or warnings when implementing my code using the instructions in the lab manual.

4. Conclusion

In conclusion, the objective of this lab was getting familiar with the elementary logic gates by designing a 4-bit magnitude comparator for unsigned and 2's complement binary numbers. I have learned how to use XNOR, OR, AND, NOT gates to design a 4-bit magnitude compactor for the unsigned and signed binary numbers.

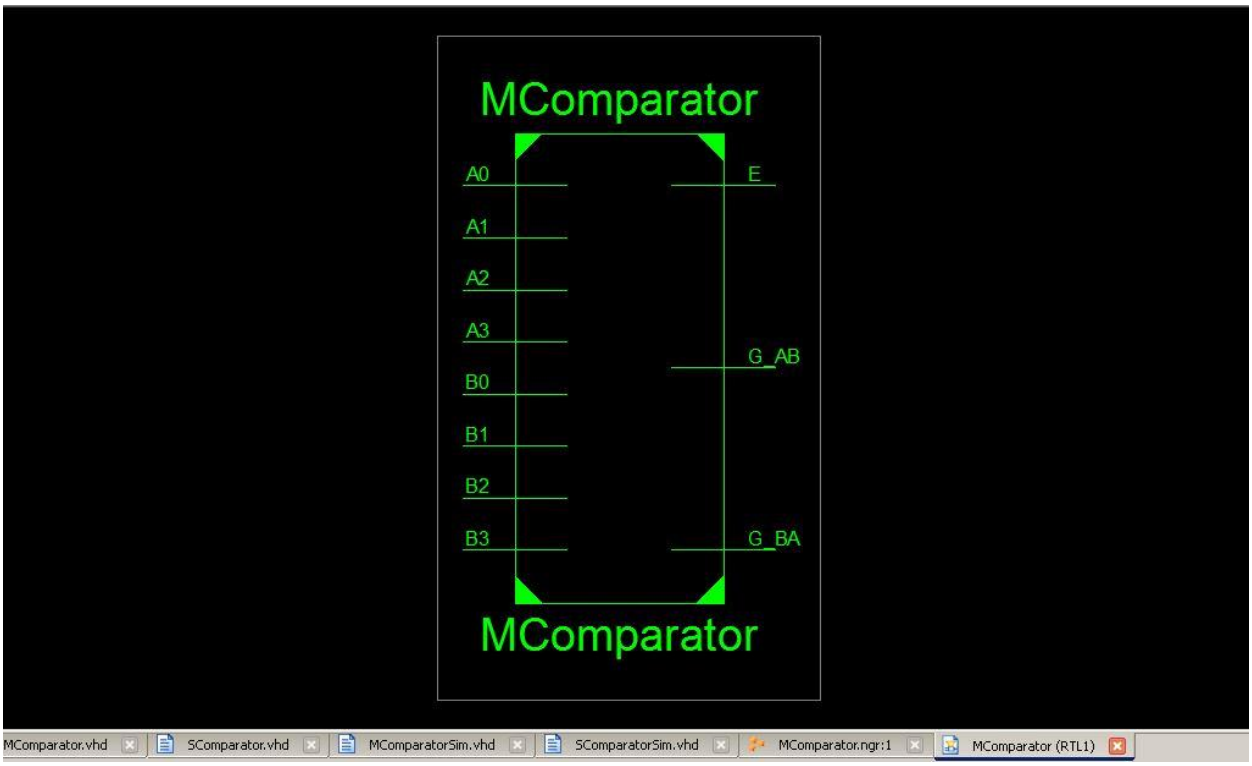
References

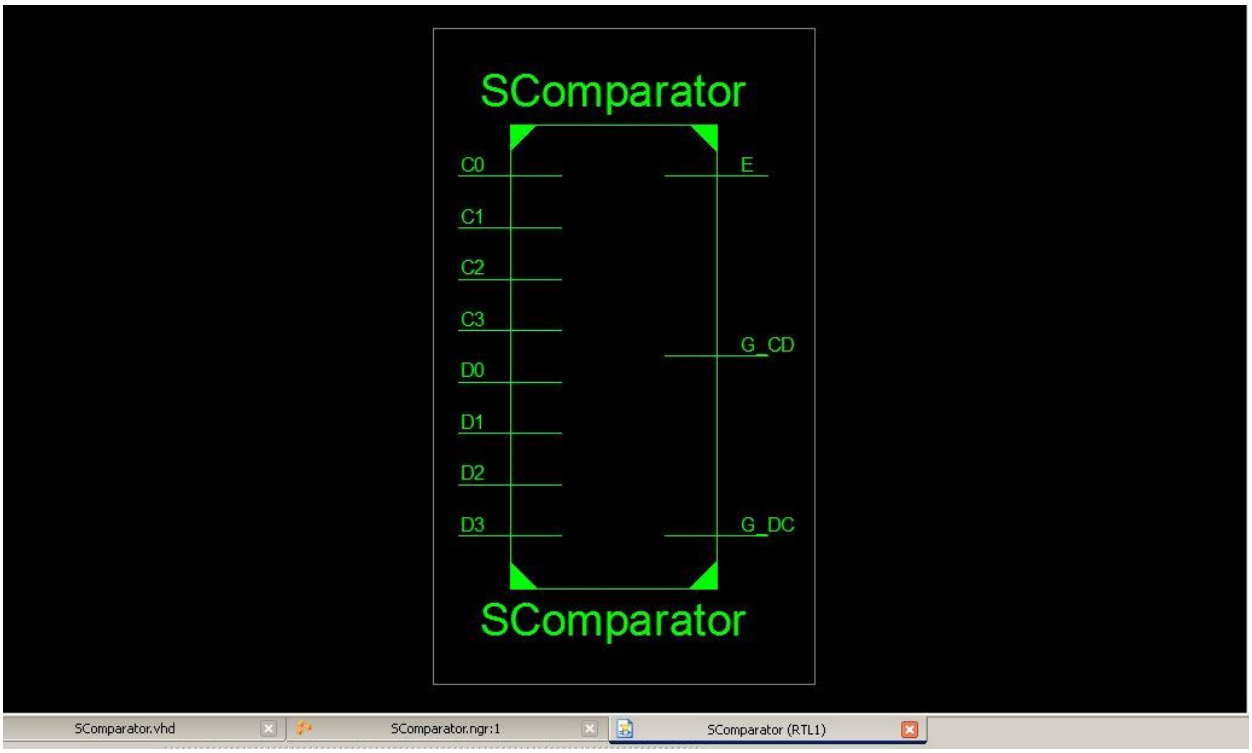
1. Lab 2 Instruction Sheet PDF

Appendix 1. Lab source code

https://drive.google.com/file/d/1o8PH7DFeTjuiJHbMw9YAqz8yQXsOUQwF/view?usp=sharing

Appendix 2. RTL schematics





Appendix 3. FPGA Board photos showing working code