**ELEC 204 Digital Design Lab Report**

**Lab 3**

**Name: Barış KAPLAN**

**Id Number: 69054**

**Date of Submission: 5 April 2021 (Monday)**

\*Please delete the highlighted lines and write your own parts for the report.

\*\*Reminder: Your lab grade is a weighted average of your performance before, during and after the lab: **Total lab grade = Preliminary work\* (30%) + Lab interview and demo\* (40%) + Lab report (30%)**

**\*\*\*Please make sure the lab report does not exceed 10 A4 pages.**

\*\*\*\*Please make sure you indicate the name of your lab collaborator (if there is any) who you worked together to solve the lab questions. Do not change your lab collaborator throughout the semester.

1. **Introduction and objectives**

**Explain the objectives of the lab (refer to the lab instruction sheet),**

**THE OBJECTIVES OF THE LAB-3**

**1-) Learning how to implement(or write) the code of the 4-bit arithmetic and logic unit.**

**2-) Observing the behavioral structure of the 4-bit ALU by running the simulation code of the 4-bit ALU.**

**3-) Observing the behavior of the 1-bit ALU by running the simulation code of the 1-bit ALU.**

**4-) Learning how to implement(or write) the code of the 1-bit arithmetic and logic unit.**

**5-) Learning how to use the “Port Map” structure in the implementation part of the 4-bit arithmetic and logic unit.**

**6-) Utilizing the basic logic gates such as AND gate, OR gate, XOR gate, and XNOR gate.**

**7-) Observing the output values of the AND gate, OR gate, XOR gate, and XNOR gate for different combinations of the input values.**

**8-) Learning how to write the simulation code (Test Bench) of the 4-bit arithmetic and logic unit.**

**9-) Learning how to write the simulation code (Test Bench) of the 1-bit arithmetic and logic unit.**

**10-) Observing the cases where the end carry is 1 for the changing values of inputs**

**11-) Observing the situations where the end carry is 0 for the changing values of inputs.**

**12-) Learning how to add two bits using bitwise operations.**

**Explain what your code has to do and describe how you did it.**

**For the logic unit part, when M is equal to 0, S1 is equal to 0, and S0 is equal to 0, my code should perform the AND operation to the inputs A and B. Here, A is one bit, and B is one bit. When A is equal to 1, and B is equal to 1, my code should give ‘1’ as the result of the AND operation. When A is equal to 1, and B is equal to 0, my code should give ‘0’ as the result of the AND operation. When A is equal to 0, and B is equal to 1, my code should give ‘0’ as the result of the AND operation. When A is equal to 0, and B is equal to 0, my code should give ‘0’ as the result of the AND operation. When M is equal to 0, S1 is equal to 0, and S0 is equal to 1, my code should perform the OR operation to the inputs A and B. When A is equal to 1, and B is equal to 1, my code should give ‘1’ as the output of the OR operation. When A is equal to 1, and B is equal to 0, my code should give ‘1’ as the output of the OR operation. When A is equal to 0, and B is equal to 1, my code should give ‘1’ as the result of the OR operation. When A is equal to 0, and B is equal to 0, my code should give ‘0’ as the result of the OR operation. When M is equal to 0, S1 is equal to 1, and S0 is equal to 0, my code should perform the XOR operation to the inputs A and B. When A is equal to 1, and B is equal to 1, my code should give ‘0’ as the outcome of the XOR operation. When A is equal to 1, and B is equal to 0, my code should give ‘1’ as the output of the XOR operation. When A is equal to 0, and B is equal to 1, my code should give ‘1’ as the output of the XOR operation. When A is equal to 0, and B is equal to 0, my code should give ‘0’ as the output of the XOR operation. When M is equal to 0, S1 is equal to 1, and S0 is equal to 1, my code should perform the XNOR operation to the inputs A and B. If A is equal to 0, and B is equal to 1, my code should give ‘0’ as the output value of the XNOR operation. If A is equal to 1, and B is equal to 0, my code should give ‘0’ as the output value of the XNOR operation. If A and B are equal to 0, my code should give ‘1’ as the output value of the XNOR operation. If A and B are equal to 1, my code should give ‘1’ as the output value of the XNOR operation. //Explanation of the logic unit of the 1-Bit ALU ends.**

**For the arithmetic unit, if M is equal to 1; S1, S0, and C0 are equal to 0, my code should give the input ‘A’ as the output. If M is equal to 1; S1 and S0 are equal to 0, and C0 is equal to 1, my code should give the result of the addition of A and 1 as the output(A+1). If M and S0 are equal to 1, S1 and C0 are equal to 0, my code should give the outcome of the addition of A and B as the output(A+B). If M, S0, and C0 are equal to 1, and S1 is equal to 0; then my code should give the result of the addition of A, B, and 1 as the output(A+B+1). If M and S1 are equal to 1, S0 and C0 are equal to 0; then my code should give the result of the addition of A and the one’s complement of B as the output. If M, S1, and C0 are equal to 1, and S0 is equal to 0; then my code should give the result of the addition of A, one’s complement of B and 1(This result is also equal to A-B). If M, S1, and S0 are equal to 1, and C0 is equal to 0; then as the output, my code should give the outcome of the addition of B and the one’s complement of A. If M , S1, S0, and C0 are equal to 1; my code should give the outcome of the addition of B, one’s complement of A, and 1 as the output value(This output value is also equal to B-A).// Explanation of the arithmetic unit of the 1-Bit ALU ends.**

**First, I have implemented the code of the 1-Bit ALU. In my implementation of the 1-Bit ALU code, I have considered all of the cases I have explained for the Logic Unit and the Arithmetic Unit. After the implementation of the 1-Bit ALU, we should insert the 1-Bit ALU component, which includes the inputs of 1-Bit ALU, and the outputs of the 1-Bit ALU, into the 4-Bit ALU implementation code. We should combine 4 of the 1-Bit ALU and design the 4-Bit ALU from these 4 components. For that purpose, I have used the port map structure in the VHDL code of the 4-Bit ALU. For the component of 4-Bit ALU, except the ones I have defined in the 1-Bit ALU component which is in the 4-Bit ALU VHDL code, I have defined some inputs, and some outputs. Their names are different from the names of the inputs and the outputs of the 1-Bit ALU component. Except these inputs and outputs, I have defined 3 carrys named carry1, carry2, and carry3 in my 4-Bit ALU VHDL code. I have put the inputs and the outputs to the Port Map in the order of the definition of the inputs and the outputs. I have put Ain(0), Bin(0), and Fo(0) into the first port map, Ain(1), Bin(1), Fo(1) into the second port map, Ain(2), Bin(2), and Fo(2) into the third port map, and Ain(3), Bin(3), and Fo(3) into the fourth port map. I have given the output carry bit of each map as the input carry bit of the subsequent port map. I have given Co as the output carry bit of the last port map, which indicates the resultant carry bit after the operation among Ain and Bin. In sum, I have divided the problem into the subproblems, which indicates a divide and conquer approach, and I have initially implemented the 1-Bit ALU. By combining 4 of the 1-Bit ALU with the help of the port maps, I have implemented the code of the 4-Bit ALU.**

1. **Methods**

**Explain the inputs (how many bits, names of the inputs),**

**The inputs of the 1-Bit ALU**

**There are 6 inputs in the 1-Bit ALU VHDL code. These are called M, S1, S0, A, B, and C. Each of these inputs is 1 bit. I have defined all of these inputs as STD\_LOGIC variables.**

**The inputs of the 4-Bit ALU**

**There are 6 inputs in the 4-Bit ALU VHDL code. These are called K, S1in, S0in, Ain, Bin, and Cin. K is 1 bit. S1in is 1 bit. S0in is 1 bit. Cin is 1 bit. Ain is 4 bit. Bin is 4 bit. I have defined K, S1in, S0in, and Cin as STD\_LOGIC variables; and Ain and Bin as STD\_LOGIC\_VECTOR(3 downto 0) variables.**

**Explain the outputs (how many bits, names of the outputs),**

**The outputs of the 1-Bit ALU**

**There are 2 outputs in the 1-Bit ALU VHDL code. These are called F and Cout. Each of these outputs is 1 bit. I have defined all of these outputs as STD\_LOGIC variables.**

**The outputs of the 4-Bit ALU**

**There are 2 outputs in the 4-Bit ALU VHDL code. These are called Fo and Co. Fo is 4 bit. Co is 1 bit. I have defined Co as a STD\_LOGIC variable, and I have defined Fo as a STD\_LOGIC\_VECTOR(3 downto 0) variable.**

**Explain what the VHDL code must do**

**M=1, S1=0, S0=0, C0=0 M=1, S1=1, S0=0, C0=0**

**The output is A. The output for this case is A+B’.**

**If A is 0, the output is 0. (CARRY DOES NOT EXIST)**

**If A is 1, the output is 1.**

**(CARRY DOES NOT EXIST)**

**M=1, S1=0, S0=0, C0=1 M=1, S1=1, S0=0, C0=1**

**The output is (A+1). Here, the output is (A-B).**

**(CARRY EXISTS) (CARRY EXISTS)**

**M=1, S1=0, S0=1, C0=0 M=1,S1=1,S0=1,C0=0**

**The output is (A+B). Here, the output is B+A’.**

**(CARRY DOES NOT EXIST) (CARRY DOES NOT EXIST)**

**M=1, S1=0, S0=1, C0=1 M=1,S1=1,S0=1,C0=1**

**The output is (A+B+1). Here, the output is B-A.**

**(CARRY EXISTS) (CARRY EXISTS)**

**The VHDL code should give the outputs that I have stated for different cases in the Arithmetic Unit. For the Logic Unit part, what the outputs should be for different cases is explained in the “Explain what your code has to and describe how you did it” part of the lab report.**

**Explain how your code works**

**signal sg1: STD\_LOGIC;**

**signal sg2: STD\_LOGIC;**

**signal sg3: STD\_LOGIC;**

**signal sg4: STD\_LOGIC;**

**signal sg5: STD\_LOGIC;**

**signal sg6: STD\_LOGIC;**

**signal sg7: STD\_LOGIC;**

**signal sg8: STD\_LOGIC;**

**signal sg9: STD\_LOGIC;**

**signal sg10: STD\_LOGIC;**

**signal sg11: STD\_LOGIC;**

**signal sg12: STD\_LOGIC;**

**signal mnt: STD\_LOGIC;**

**signal s1nt: STD\_LOGIC;**

**signal s0nt: STD\_LOGIC;**

**signal cnt: STD\_LOGIC;**

**signal ant: STD\_LOGIC;**

**signal bnt: STD\_LOGIC;//Above here, some necessary signals are defined as STD\_LOGIC variables.**

**mnt<=(not M);**

**s1nt<=(not S1);**

**s0nt<=(not S0);**

**cnt<=(not C);**

**ant<=(not A);**

**bnt<=(not B); //assignment of the signal names into some necessary values which will be used in the code**

**//START OF THE LOGIC UNIT CODE**

**sg1<=((mnt) and (s1nt) and (s0nt)) and ((A and B));//M=0, S1=0, and S0=0 case**

**sg2<=((mnt) and (s1nt) and (S0)) and (A or B); //M=0, S1=0, and S0=1 case**

**sg3<=((mnt) and (S1) and (s0nt)) and (A xor B);//M=0, S1=1, and S0=0 case**

**sg4<=((mnt) and (S1) and (S0)) and (A xnor B);//M=0, S1=1, and S0=1 case**

**//END OF THE LOGIC UNIT CODE**

**//START OF THE ARITHMETIC UNIT CODE**

**sg5<=((M) and (s1nt) and (s0nt) and (cnt)) and (A);//M=1, S1=0, S0=0, and C=0 case**

**sg6<=((M) and (s1nt) and (s0nt) and (C)) and (A xor '1');// M=1, S1=0, S0=0, and C=1 case**

**sg7<=((M) and (s1nt) and (S0) and (cnt)) and (A xor B);//M=1, S1=0, S0=1, and C=0 case,**

**//to add A and B, we should take the xor of A and B.**

**sg8<=((M) and (s1nt) and (S0) and (C)) and ((A xor B) xor '1');//M=1, S1=0, S0=1, and C=1 case**

**//to add A, B, AND 1, we should take the xor of A, B, and 1.**

**sg9<=((M) and (S1) and (s0nt) and (cnt)) and (A xor (bnt));//M=1, S1=1, S0=0, and C=0 case**

**//to add A and B’, we should apply the xor operation to A and B’.**

**sg10<=((M) and (S1) and (s0nt) and (C)) and (A xor (bnt) xor '1');//M=1, S1=1, S0=0, and C=1 case**

**sg11<=((M) and (S1) and (S0) and (cnt)) and (B xor (ant));//M=1, S1=1, S0=1, and C=0 case**

**sg12<=((M) and (S1) and (S0) and (C)) and (B xor (ant) xor '1');//M=1, S1=1, S0=1, C= 1 case**

**//END OF THE ARITHMETIC UNIT CODE**

**Cout<=((M) and (s1nt) and (s0nt) and (cnt) and '0')or((M) and (s1nt) and (s0nt) and C and A)or((M) and (s1nt) and (S0) and (cnt) and (A and B))or((M) and (s1nt) and S0 and C and (A or B))or(M and S1 and (s0nt) and (not C) and (A and (bnt))) or (M and S1 and (s0nt) and C and (A or (bnt))) or (M and S1 and S0 and (cnt) and (B and (ant))) or (M and S1 and S0 and C and (B or (ant))); //Expressing the end carry by thinking all possible cases of the carry bits.**

**When M=1, S1=0, S0=0, and C=0; the output is A. When A is 1, the carry is 0. When A is 0, the carry is 0. So, we should write ‘0’ to the expression of carry for this case. When M=1, S1=0, S0=0, and C=1; the output is (A+1). When A is 1, the carry is 1. When A is 0, the carry is 0. The value of the A is equal to the value of the carry in both of the cases. So, we can express the carry as ‘A’ for this case.**

**When M=1, S1=0, S0=1, and C=0; the output is A+B. When A and B are both 0, the carry is 0. When they are both 1, the carry is 1. When one of them is 0, and the other one is 1; the carry is 0. So, since the carry is 1 for the case where both of A and B are equal to 1, we can express the carry with an AND operation which is ‘A and B’.**

**When M=1, S1=0, S0=1, C=1; the output is equal to (A+B+1). When both of A and B are equal to 0, the carry is 0. When both of A and B are equal to 1, the carry is 1. When A is equal to 0, and B is equal to 1; the carry is 1. When A is equal to 1, and B is equal to 0, the carry is 1. So, since except the input combination of (0,0) the carry is 1, we can express the carry as (A xor B) for this situation.**

**When M=1, S1=1, S0=0, C=0; the output is (A+B’). When B and A are both 0, the carry is 0. When A and B are both 1, the carry is 0. When A is 1, and B is 0; the carry is 1. When A is 0, and B is 1; the carry is 0. When we try express the carry, we can write A\*B\*C+A\*C\*B’+A’\*B’\*C+A\*B’\*C’ as the boolean expression. Put C=0 here, we will end up with A\*B’ as the simpilified carry expression for this case.**

**When M=1, S1=1, S0=0, and C=1; the output is A+B’+1. When A and B are both 0, the carry is 1(A’\*B’\*C). When A and B are both 1, the carry is 1(A\*B\*C). When A is 1, and B is 0; the carry is 1(A\*B’\*C). When A is 0, and B is 1; the carry is 0(A\*B’\*C’). (A’\*B’\*C) + A\*B\*C+A\*B’\*C+A\*B’\*C’(carry expression). Put C=1 , we will end up with A’B’+AB+AB’= A+A’B’= (A’+A)\*(A+B’)= (A+B’). So, we can use (A or (not B)) for the carry expression in this case.**

**When M=1, S1=1, S0=1, C=0; the output is B+A’. When we replace the A’s with B’s and B’s with A’s in the result of the M=1, S1=1, S0=0, and C=0; we will find the carry expression as B\*A’ (B and (not A)) for this case.**

**Again, for the last case, we should swap A and B in the resultant expression of carry where the output is A+B’+1. So, for the last case, the boolean expression of the carry is (B or (not A)). For each possible case, we should apply the AND operation among the necessary values of M,S1,S0, C with the carry expressions we found in each case.**

**Finally , we should take the OR of the multiplications that we do in order to express the Cout.**

**F<=sg1 or sg2 or sg3 or sg4 or sg5 or sg6 or sg7 or sg8 or sg9 or sg10 or sg11 or sg12;**

**//Expressing the F as the output which is obtained by oring the signals which are sg1, sg2, sg3,….,sg12.**

**//END OF THE 1-BIT ALU VHDL CODE**

**In the 4-Bit ALU VHDL Code, I have combined four 1-Bit ALU into a 4-Bit ALU. I have used port maps in order to combine them.**

**a : BitAlufor1Bit port map (K,S1in,S0in,Ain(0),Bin(0),Cin,Fo(0),carry1);//Port map for the first bit of Ain and for the first bit of Bin.**

**b : BitAlufor1Bit port map (K,S1in,S0in,Ain(1),Bin(1),carry1,Fo(1),carry2);//Port map for the second bit of Ain and the second bit of Bin**

**c : BitAlufor1Bit port map (K,S1in,S0in,Ain(2),Bin(2),carry2,Fo(2),carry3);//Port map for the third bit of Ain and the third bit of Bin.**

**d : BitAlufor1Bit port map (K,S1in,S0in,Ain(3),Bin(3),carry3,Fo(3),Co);//Port map for the last bit of Ain and the last bit of Bin.**

**//Output carry for each port map is the input carry of the subsequent port map.**

**Provide the truth table**

**THE TRUTH TABLE OF THE LOGIC UNIT:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **M** | **S1** | **S0** | **A** | **B** | **Output** |
| **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **0** | **1** | **0** |
| **0** | **0** | **0** | **1** | **0** | **0** |
| **0** | **0** | **0** | **1** | **1** | **1** |
| **0** | **0** | **1** | **0** | **0** | **0** |
| **0** | **0** | **1** | **0** | **1** | **1** |
| **0** | **0** | **1** | **1** | **0** | **1** |
| **0** | **0** | **1** | **1** | **1** | **1** |
| **0** | **1** | **0** | **0** | **0** | **0** |
| **0** | **1** | **0** | **0** | **1** | **1** |
| **0** | **1** | **0** | **1** | **0** | **1** |
| **0** | **1** | **0** | **1** | **1** | **0** |
| **0** | **1** | **1** | **0** | **0** | **1** |
| **0** | **1** | **1** | **0** | **1** | **0** |
| **0** | **1** | **1** | **1** | **0** | **0** |
| **0** | **1** | **1** | **1** | **1** | **1** |

**THE TRUTH TABLE OF THE ARITHMETIC UNIT:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **M** | **S1** | **S0** | **C** | **A** | **B** | **Result** |
| **1** | **0** | **0** | **0** | **1** | **\*** | **1** |
| **1** | **0** | **0** | **0** | **0** | **\*** | **0** |
| **1** | **0** | **0** | **1** | **0** | **\*** | **1** |
| **1** | **0** | **0** | **1** | **1** | **\*** | **0** |
| **1** | **0** | **1** | **0** | **0** | **0** | **0** |
| **1** | **0** | **1** | **0** | **0** | **1** | **1** |
| **1** | **0** | **1** | **0** | **1** | **0** | **1** |
| **1** | **0** | **1** | **0** | **1** | **1** | **0** |
| **1** | **0** | **1** | **1** | **0** | **0** | **1** |
| **1** | **0** | **1** | **1** | **0** | **1** | **0** |
| **1** | **0** | **1** | **1** | **1** | **0** | **0** |
| **1** | **0** | **1** | **1** | **1** | **1** | **1** |
| **1** | **1** | **0** | **0** | **0** | **0** | **1** |
| **1** | **1** | **0** | **0** | **0** | **1** | **0** |
| **1** | **1** | **0** | **0** | **1** | **0** | **0** |
| **1** | **1** | **0** | **0** | **1** | **1** | **1** |
| **1** | **1** | **0** | **1** | **0** | **0** | **0** |
| **1** | **1** | **0** | **1** | **1** | **0** | **1** |
| **1** | **1** | **0** | **1** | **0** | **1** | **1** |
| **1** | **1** | **0** | **1** | **1** | **1** | **0** |
| **1** | **1** | **1** | **0** | **0** | **0** | **1** |
| **1** | **1** | **1** | **0** | **0** | **1** | **0** |
| **1** | **1** | **1** | **0** | **1** | **0** | **0** |
| **1** | **1** | **1** | **0** | **1** | **1** | **1** |
| **1** | **1** | **1** | **1** | **0** | **0** | **0** |
| **1** | **1** | **1** | **1** | **0** | **1** | **1** |
| **1** | **1** | **1** | **1** | **1** | **0** | **1** |
| **1** | **1** | **1** | **1** | **1** | **1** | **0** |

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

Explain what problems you encountered while writing your code.

**I have not encountered any problems while writing my code.**

Explain what synthesis errors and warnings you observed.

**I have not encountered any synthesis errors and warnings.**

Explain what problems you had to solve (or could not) on your board even if your code could be synthesized successfully**. (I have not used my board in this lab.)**

1. **Conclusion**

Provide a 1 paragraph summary of the lab and explain what you learned from this lab.

**In this lab, firstly, I have created a new project called BitAlufor1Bit. After that, I have created a vhdl module for the 1-Bit ALU implementation. I have done my implementation of the 1-Bit ALU into this VHDL module. While I was doing my implementation, I considered all of the possible cases for the arithmetic unit, and for the logic unit. I have synthesized my code of 1-Bit ALU. Subsequently, I have created the VHDL Test Bench of the 1-Bit ALU implementation. I have written some sample inputs into the 1-Bit ALU simulation. I have run the behavioral model, and saw the behavior of the 1-bit ALU. I saw that the values of A and B are changing when I increase or decrease the time from the timing diagram. Then, I have created the VHDL Module for the 4-Bit ALU inside the project that I have initially created. By using the port maps, I have done the implementation of the 4-Bit ALU into this module. I have synthesized my code of 4-Bit ALU. I have inserted the 1-Bit ALU component into the 4-Bit ALU. For two 4-bit inputs, I have used 4 port maps in the 4-Bit ALU VHDL code. The input carry of each map was the output carry of the previous map. Moreover, I have created the VHDL Test Bench of the 4-Bit Alu implementation. In this test bench, I have included some example inputs. After that, I have run the 4-Bit simulation and observed the behavioral model of the 4-Bit ALU.**

**In this lab, I have learned how to implement the code of the 1-Bit ALU. I have learned how to implement the code of the 4-Bit ALU. I have learned how to write the simulation code of the 1-Bit ALU. I have learned how to write the simulation code of the 4-Bit ALU. I have learned how to use port maps. In addition, for different input pairs, I have learned what the output values of AND gate, OR gate, XOR gate, and XNOR gate are.**

References

1. Please cite any resource (web site, book, youtube video) you used for this lab.

**Youtube video:** <https://www.youtube.com/watch?v=U6GDGzEGRgc>

**Web site:** <https://eem.eskisehir.edu.tr/egermen/EEM%20232/icerik/Week%2011%20ALU.pdf>

**Youtube video:** https://www.youtube.com/watch?v=h8PAobl4xUk

**Appendix 1. Lab source code**

**1-Bit ALU Source Code (Implementation Part)**

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

-- Company:

-- Engineer:

--

-- Create Date:    21:29:19 03/30/2021

-- Design Name:

-- Module Name:    BitAlufor1Bit - 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 primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity BitAlufor1Bit is

    Port ( M : in  STD\_LOGIC;

           S1 : in  STD\_LOGIC;

           S0 : in  STD\_LOGIC;

           A : in  STD\_LOGIC;

           B : in  STD\_LOGIC;

           C : in  STD\_LOGIC;

           F : out  STD\_LOGIC;

           Cout : out  STD\_LOGIC);

end BitAlufor1Bit;

architecture Behavioral of BitAluFor1Bit is

signal sg1: STD\_LOGIC;

signal sg2: STD\_LOGIC;

signal sg3: STD\_LOGIC;

signal sg4: STD\_LOGIC;

signal sg5: STD\_LOGIC;

signal sg6: STD\_LOGIC;

signal sg7: STD\_LOGIC;

signal sg8: STD\_LOGIC;

signal sg9: STD\_LOGIC;

signal sg10: STD\_LOGIC;

signal sg11: STD\_LOGIC;

signal sg12: STD\_LOGIC;

signal mnt: STD\_LOGIC;

signal s1nt: STD\_LOGIC;

signal s0nt: STD\_LOGIC;

signal cnt: STD\_LOGIC;

signal ant: STD\_LOGIC;

signal bnt: STD\_LOGIC;

begin

mnt<=(not M);

s1nt<=(not S1);

s0nt<=(not S0);

cnt<=(not C);

ant<=(not A);

bnt<=(not B);

sg1<=((mnt) and (s1nt) and (s0nt)) and ((A and B));

sg2<=((mnt) and (s1nt) and (S0)) and (A or B);

sg3<=((mnt) and (S1) and (s0nt)) and (A xor B);

sg4<=((mnt) and (S1) and (S0)) and (A xnor B);

sg5<=((M) and (s1nt) and (s0nt) and (cnt)) and (A);

sg6<=((M) and (s1nt) and (s0nt) and (C)) and (A xor '1');

sg7<=((M) and (s1nt) and (S0) and (cnt)) and (A xor B);

sg8<=((M) and (s1nt) and (S0) and (C)) and ((A xor B) xor '1');

sg9<=((M) and (S1) and (s0nt) and (cnt)) and (A xor (bnt));

sg10<=((M) and (S1) and (s0nt) and (C)) and (A xor (bnt) xor '1');

sg11<=((M) and (S1) and (S0) and (cnt)) and (B xor (ant));

sg12<=((M) and (S1) and (S0) and (C)) and (B xor (ant) xor '1');

Cout<=((M) and (s1nt) and (s0nt) and (cnt) and '0')or((M) and (s1nt) and (s0nt) and C and A)or((M) and (s1nt) and (S0) and (cnt) and (A and B))or((M) and (s1nt) and S0 and C and (A or B))or(M and S1 and (s0nt) and (not C) and (A and (bnt))) or (M and S1 and (s0nt) and C and (A or (bnt))) or (M and S1 and S0 and (cnt) and (B and (ant))) or (M and S1 and S0 and C and (B or (ant)));

F<=sg1 or sg2 or sg3 or sg4 or sg5 or sg6 or sg7 or sg8 or sg9 or sg10 or sg11 or sg12;

end Behavioral;

**1-Bit ALU Simulation Code (Test Bench Part)**

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

-- Company:

-- Engineer:

--

-- Create Date:   21:30:24 03/30/2021

-- Design Name:

-- Module Name:   C:/Test/BitAluFor1Bit/BitAlu1BitSimulation.vhd

-- Project Name:  BitAluFor1Bit

-- Target Device:

-- Tool versions:

-- Description:

--

-- VHDL Test Bench Created by ISE for module: BitAlufor1Bit

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

-- Notes:

-- This testbench has been automatically generated using types std\_logic and

-- std\_logic\_vector for the ports of the unit under test.  Xilinx recommends

-- that these types always be used for the top-level I/O of a design in order

-- to guarantee that the testbench will bind correctly to the post-implementation

-- simulation model.

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

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;

ENTITY BitAlu1BitSimulation IS

END BitAlu1BitSimulation;

ARCHITECTURE behavior OF BitAlu1BitSimulation IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT BitAlufor1Bit

    PORT(

         M : IN  std\_logic;

         S1 : IN  std\_logic;

         S0 : IN  std\_logic;

         A : IN  std\_logic;

         B : IN  std\_logic;

         C : IN  std\_logic;

         F : OUT  std\_logic;

         Cout : OUT  std\_logic

        );

    END COMPONENT;

   --Inputs

   signal M : std\_logic := '0';

   signal S1 : std\_logic := '0';

   signal S0 : std\_logic := '0';

   signal A : std\_logic:= '0';

   signal B : std\_logic:='0';

   signal C : std\_logic:='0';

  --Outputs

   signal F : std\_logic;

   signal Cout : std\_logic;

   -- No clocks detected in port list. Replace <clock> below with

   -- appropriate port name

BEGIN

-- Instantiate the Unit Under Test (UUT)

   uut: BitAlufor1Bit PORT MAP (

          M => M,

          S1 => S1,

          S0 => S0,

          A => A,

          B => B,

          C => C,

          F => F,

          Cout => Cout

        );

   -- Stimulus process

   stim\_proc: process

   begin

  M<='0';

S1<='1';

S0<='0';

A<='1';

B<='1';

      -- hold reset state for 100 ns.

      wait for 100 ns;

M<='0';

S1<='0';

S0<='0';

A<='0';

B<='1';

      wait for 100 ns;

M<='0';

S1<='1';

S0<='1';

A<='0';

B<='0';

      -- insert stimulus here

      wait;

   end process;

END;

**4-Bit ALU Source Code (Implementation Part)**

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

-- Company:

-- Engineer:

--

-- Create Date:    13:48:43 04/01/2021

-- Design Name:

-- Module Name:    BitAlufor4Bit - 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 primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity BitAlufor4Bit is

    Port ( K : in  STD\_LOGIC;

           S1in : in  STD\_LOGIC;

           S0in : in  STD\_LOGIC;

           Ain : in  STD\_LOGIC\_VECTOR (3 downto 0);

           Bin : in  STD\_LOGIC\_VECTOR (3 downto 0);

           Cin : in  STD\_LOGIC;

           Fo : out  STD\_LOGIC\_VECTOR (3 downto 0);

           Co : out  STD\_LOGIC);

end BitAlufor4Bit;

architecture Behavioral of BitAlufor4Bit is

COMPONENT BitAlufor1Bit

Port ( M : in  STD\_LOGIC;

           S1 : in  STD\_LOGIC;

           S0 : in  STD\_LOGIC;

           A : in  STD\_LOGIC;

           B : in  STD\_LOGIC;

           C : in  STD\_LOGIC;

           F : out  STD\_LOGIC;

           Cout : out  STD\_LOGIC);

END COMPONENT;

signal carry1: std\_logic;

signal carry2: std\_logic;

signal carry3: std\_logic;

begin

a : BitAlufor1Bit port map (K,S1in,S0in,Ain(0),Bin(0),Cin,Fo(0),carry1);

b : BitAlufor1Bit port map (K,S1in,S0in,Ain(1),Bin(1),carry1,Fo(1),carry2);

c : BitAlufor1Bit port map (K,S1in,S0in,Ain(2),Bin(2),carry2,Fo(2),carry3);

d : BitAlufor1Bit port map (K,S1in,S0in,Ain(3),Bin(3),carry3,Fo(3),Co);

end Behavioral;

***//To calculate the overflow in the 2’s complement system, we should take the xor of Co and carry3.***

***//For example, add 1111 and 1111. The result of the addition will be 1110. Here, Co is 1, and carry3 is also 1. 1 xor 1 will give 0. For the first example, overflow does not exist. As a second example, add 1000 and 0011. Here, Co is 0, and carry 3 is 0. 0 xor 0 will give 0. For the second example, overflow does not exist.***

**4-Bit ALU Simulation Code (Test Bench Part)**

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

-- Company:

-- Engineer:

--

-- Create Date:   13:57:14 04/01/2021

-- Design Name:

-- Module Name:   C:/Test/BitAlufor1Bit/BitAlu4BitSimulation.vhd

-- Project Name:  BitAlufor1Bit

-- Target Device:

-- Tool versions:

-- Description:

--

-- VHDL Test Bench Created by ISE for module: BitAlufor4Bit

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

-- Notes:

-- This testbench has been automatically generated using types std\_logic and

-- std\_logic\_vector for the ports of the unit under test.  Xilinx recommends

-- that these types always be used for the top-level I/O of a design in order

-- to guarantee that the testbench will bind correctly to the post-implementation

-- simulation model.

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

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;

ENTITY BitAlu4BitSimulation IS

END BitAlu4BitSimulation;

ARCHITECTURE behavior OF BitAlu4BitSimulation IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT BitAlufor4Bit

    PORT(

         K : IN  std\_logic;

         S1in : IN  std\_logic;

         S0in : IN  std\_logic;

         Ain : IN  std\_logic\_vector(3 downto 0);

         Bin : IN  std\_logic\_vector(3 downto 0);

         Cin : IN  std\_logic;

         Fo : OUT  std\_logic\_vector(3 downto 0);

         Co : OUT  std\_logic

        );

    END COMPONENT;

   --Inputs

   signal K : std\_logic := '0';

   signal S1in : std\_logic := '0';

   signal S0in : std\_logic := '0';

   signal Ain : std\_logic\_vector(3 downto 0) := (others => '0');

   signal Bin : std\_logic\_vector(3 downto 0) := (others => '0');

   signal Cin : std\_logic := '0';

  --Outputs

   signal Fo : std\_logic\_vector(3 downto 0);

   signal Co : std\_logic;

BEGIN

-- Instantiate the Unit Under Test (UUT)

   uut: BitAlufor4Bit PORT MAP (

          K => K,

          S1in => S1in,

          S0in => S0in,

          Ain => Ain,

          Bin => Bin,

          Cin => Cin,

          Fo => Fo,

          Co => Co

        );

   stim\_proc: process

   begin

  K<='0';

S1in<='0';

S0in<='0';

Ain<="1111";

Bin<="0000";

      -- hold reset state for 100 ns.

      wait for 75 ns;

K<='0';

S1in<='0';

S0in<='1';

Ain<="1101";

Bin<="0010";

wait for 75 ns;

K<='0';

S1in<='1';

S0in<='0';

Ain<="1111";

Bin<="0110";

wait for 100 ns;

K<='0';

S1in<='1';

S0in<='1';

Ain<="1111";

Bin<="0111";

      wait for 75 ns;

K<='1';

S1in<='0';

S0in<='0';

Ain<="1111";

Bin<="0011";

Cin<='0';

  wait for 75 ns;

K<='1';

S1in<='0';

S0in<='0';

Ain<="1111";

Bin<="0001";

Cin<='1';

wait for 75 ns;

K<='1';

S1in<='0';

S0in<='1';

Ain<="1010";

Bin<="0001";

Cin<='0';

wait for 75 ns;

K<='1';

S1in<='0';

S0in<='1';

Ain<="0000";

Bin<="0001";

Cin<='1';

wait for 75 ns;

K<='1';

S1in<='1';

S0in<='0';

Ain<="1001";

Bin<="0000";

Cin<='0';

wait for 75 ns;

K<='1';

S1in<='1';

S0in<='0';

Ain<="0011";

Bin<="0000";

Cin<='0';

wait for 75 ns;

K<='1';

S1in<='1';

S0in<='0';

Ain<="0111";

Bin<="1000";

Cin<='1';

wait for 75 ns;

K<='1';

S1in<='1';

S0in<='1';

Ain<="1100";

Bin<="1000";

Cin<='0';

wait for 75 ns;

K<='1';

S1in<='1';

S0in<='1';

Ain<="1111";

Bin<="1000";

Cin<='1';

wait;

   end process;

END;

**RTL SCHEMATIC OF THE BIT ALU FOR 4 BIT**

**Graphical user interface

Description automatically generated with low confidence**

**RTL SCHEMATIC OF THE BIT ALU FOR 1 BIT**

**A picture containing text, clock

Description automatically generated**