CSE 460: VLSI Design Laboratory Project

| 1st Member: Tasnia Zaman | 2nd Member: Md Shakibul Alam |
| --- | --- |
| *Student ID:20301296* | *Student ID: 20301286* |
| Email: tasnia.zaman@g.bracu.ac.bd | Email: md.shakibul.alam@g.bracu.ac.bd |
|  |  |
| 3rd Member: Md. Abu Bakar Siddique Khan |  |
| *Student ID: 20301041* |  |
| Email: abu.bakar.siddique.khan@g.bracu.ac.bd | |

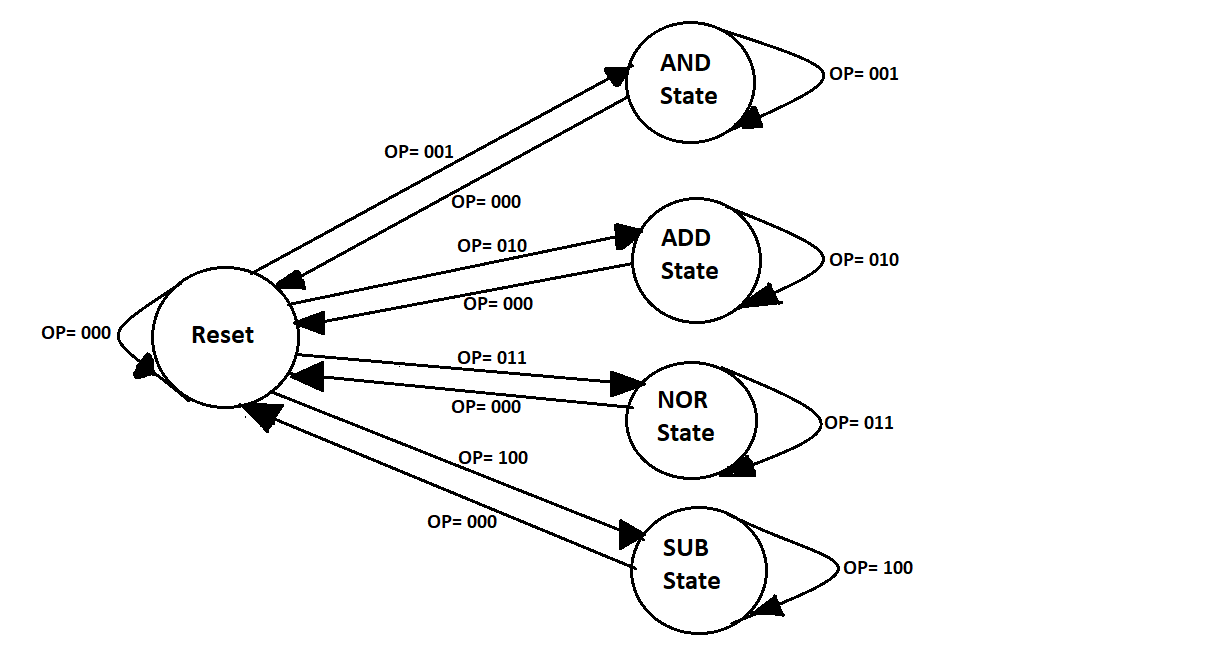
# ***Abstract***— **In this project, Quartus and Verilog HDL are used to develop and implement a 4-bit ALU. Four distinct arithmetic or logical operations, each of which is denoted by a three-bit opcode, can be carried out by the ALU. Based on the selected operation, the ALU receives two four-bit inputs, A and B, and outputs a four-bit result, C. Additionally, the ALU generates three flags: the carry, zero, and sign flags, which depend on the outcome of the operation. The ALU's whole Verilog code is available.**

# 

# INTRODUCTION

Quartus and Verilog HDL are used in the implementation of this 4-bit ALU. Verilog HDL (Hardware Description Language) is a computer language used to describe and simulate electrical systems, whereas Quartus is a software tool used to design and construct digital logic circuits. With the use of these resources, the ALU was created to calculate the intended operation from two four-bit inputs, A and B, and a three-bit opcode. The ALU is capable of performing the reset, AND, ADD, NOR, and SUB operations. The ALU generates a carry, zero, and sign flag in addition to a four-bit output, C, depending on the selected operation. The result of the operation determines these flags, which provide additional information about the output. Overall, this 4-bit ALU is an important and vital component of digital electronics since it makes it possible to manipulate and process binary data.

# OPERATION

*A. State Diagram:*

*B. Explanation with timing diagram:*

a) *RESET:* The RESET action is carried out when the opcode is set to 000. The timing diagram in this case has two opcode inputs, one of which is AND and the other is ADD. We can see from the timing diagram that following the AND operation result, the results are held in reset mode until the beginning of the following opcode operation, ADD.

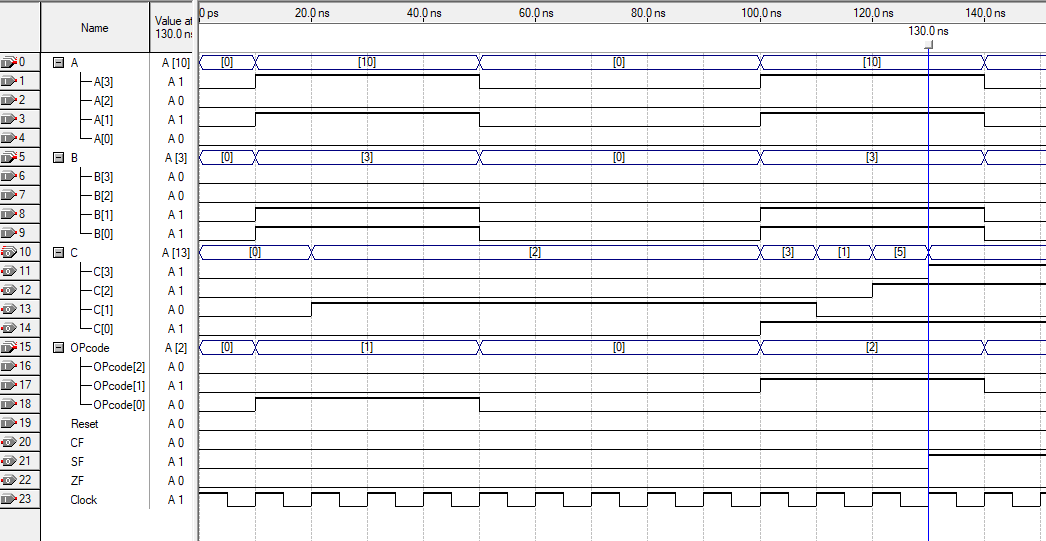


Fig .1. Reset timing example for AND and ADD

b) *AND:* The AND action is carried out when the opcode is set to 001. The output value C is 0010. We've set up a two-bit temp variable to keep track of the carry bit of our bitwise. A count variable has also been set up. This will preserve the bit number track. In addition, there will be a carry that will transport bits from each bitwise and operation. We've used A-1010 and B-0011 as examples. For AND operation, we find that C[0]=0, C[1]=1, C[2]=0, and C[3]=0. The CF, SF, and ZF will be updated once C is determined. All the values will be zero as there is no carry flag and the MSB is 0.

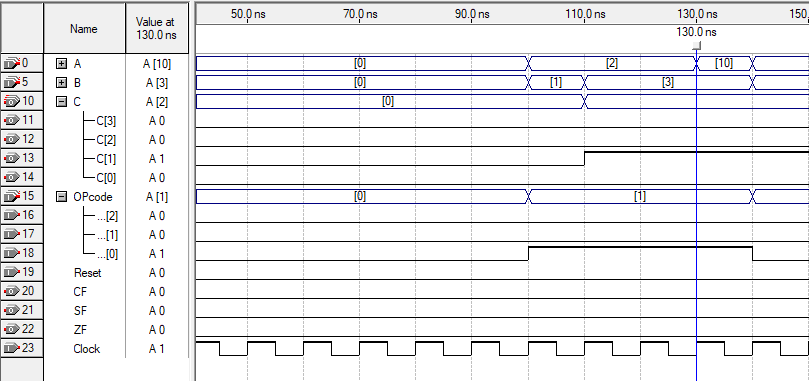


Fig .2. Timing diagram example for AND

c) *ADD*: The ADD action is carried out when the opcode is set to 010. The output value C, which contains the sum of A+B, becomes 1101. We've set up a two-bit temp variable to keep track of the carry bit of our bitwise add. A count variable has also been set up. This will preserve the bit number track. As well as a carry variable that will transport bits from each bitwise addition operation. We've used A-1010 and B-0011 as examples. We can see that for the ADD operation, we have C[0]=1, C[1]=0, C[2]=1, and C[3]=1. The CF, SF, and ZF will be updated once C is determined. Only the SF will be one as the MSB is 1.

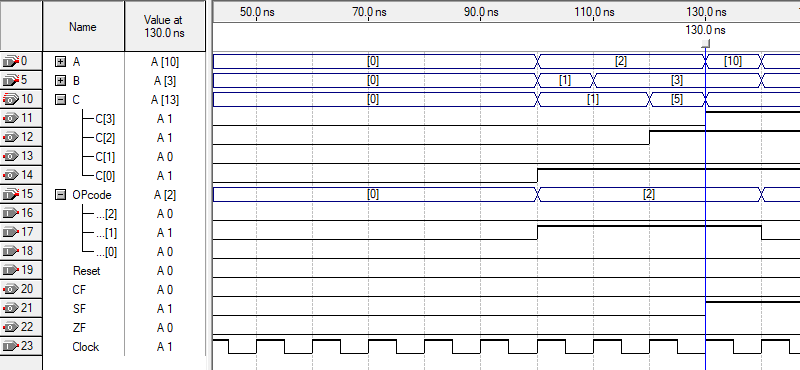


Fig .3. Timing diagram example for ADD

d) *NOR:* The NOR action is carried out when the opcode is set to 011. The output value C is 0100. We've set up a two-bit temp variable to keep track of the carry bit of our bitwise nor. A count variable has also been set up. This will preserve the bit number track. In addition, there will be a carry that will transport bits from each bitwise NOR operation. We've used A-1010 and B-0011 as examples. For AND operation, we find that C[0]=0, C[1]=0, C[2]=1, and C[3]=0. The CF, SF, and ZF will be updated once C is determined. All the values will be zero as there is no carry flag and the MSB is 0.

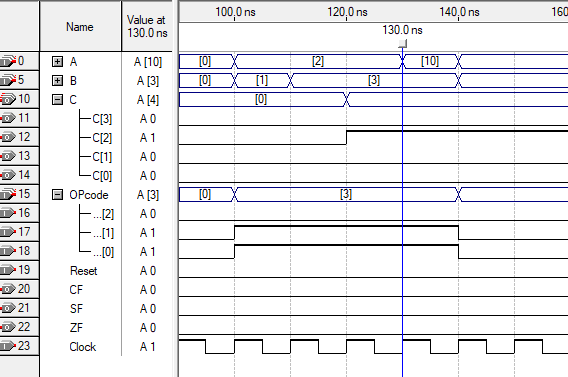


Fig .4. Timing diagram example for NOR

e) *SUB:* The SUB action is carried out when the opcode is set to 100. The output value C is 0111. We've set up a two-bit temp variable to keep track of the carry bit of our bitwise sub. A count variable has also been set up. This will preserve the bit number track. In addition, there will be a carry that will transport bits from each bitwise sub-operation. We've used A-1010 and B-0011 as examples. For SUB operation, we find that C[0]=1, C[1]=1, C[2]=1, and C[3]=0. The CF, SF, and ZF will be updated once C is determined. All the values will be zero as there is no carry flag and the MSB is 0.

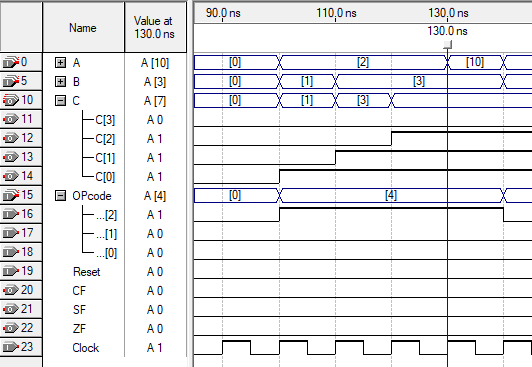


Fig .5. Timing diagram example for SUB

# CONCLUSION

With the use of five opcodes, we have created a 4-bit arithmetic and logic unit (ALU) that can carry out the following five operations: RESET, AND, ADD, NOR, and SUB. Four bits from each of A and B make up the input, and the ALU creates output that is placed in C after five consecutive clock cycles. To build the design in Quartus, we used the Verilog Hardware Description Language (HDL). From the least significant bit to the most significant bit, the output value of C is shown bit by bit. Additionally, the ALU generates the Zero Flag (ZF), Sign Flag (SF), and Carry Flag (CF) outputs. When output C is set to 0, SF is set to 1, when output C's most significant bit is set to 1, and CF is set to 1, when the output carry is set to 1, ZF is set to 1. We have confirmed that the ALU performs each of the five operations as intended and generates the anticipated output.

1. APPENDIX

*Verilog Code:*

module project\_lab(input Clock, input Reset,input[3:0]A,input[3:0]B,input[2:0]OPcode,output reg[3:0]C,output reg ZF, output reg CF, output reg SF);

reg[1:0]temp;

reg carry;

initial carry=0;

reg [2:0] count;

initial temp=0;

initial count=0;

parameter [2:0] RESET = 3'b000, AND = 3'b001, ADD = 3'b010, NOR = 3'b011, SUB = 3'b100;

always @(posedge Clock)

begin

if (Reset==1)

begin

temp=0;

count=1;

end

else

begin

if (OPcode==RESET)

begin

temp=0;

count=1;

end

else if (OPcode==AND)

begin

if (count==1)

begin

C[0]=A[0]&B[0];

count=2;

end

else if (count==2)

begin

C[1]=A[1]&B[1];

count=3;

end

else if (count==3)

begin

C[2]=A[2]&B[2];

count=4;

end

else if (count==4)

begin

C[3]=A[3]&B[3];

if (C==0)

ZF=1;

else

ZF=0;

SF=C[3];

CF=0;

count=0;

end

end

else if (OPcode==ADD)

begin

if (count==1)

begin

temp=A[0]+B[0];

C[0]=temp[0];

carry=temp[1];

count=2;

end

else if (count==2)

begin

temp=A[1]+B[1]+carry;

C[1]=temp[0];

carry=temp[1];

count=3;

end

else if (count==3)

begin

temp=A[2]+B[2]+carry;

C[2]=temp[0];

carry=temp[1];

count=4;

end

else if (count==4)

begin

temp=A[3]+B[3]+carry;

C[3]=temp[0];

carry=temp[1];

if (C==0)

ZF=1;

else

ZF=0;

SF=C[3];

CF=carry;

carry=0;

count=0;

end

end

else if (OPcode==NOR)

begin

if (count==1)

begin

C[0]=~(A[0]|B[0]);

count=2;

end

else if (count==2)

begin

C[1]=~(A[1]|B[1]);

count=3;

end

else if (count==3)

begin

C[2]=~(A[2]|B[2]);

count=4;

end

else if (count==4)

begin

C[3]=~(A[3]|B[3]);

if (C==0)

ZF=1;

else

ZF=0;

SF=C[3];

CF=0;

count=0;

end

end

else if (OPcode==SUB)

begin

if (count==1)

begin

temp=A[0]-B[0];

C[0]=temp[0];

carry=temp[1];

count=2;

end

else if (count==2)

begin

temp=A[1]-B[1]-carry;

C[1]=temp[0];

carry=temp[1];

count=3;

end

else if (count==3)

begin

temp=A[2]-B[2]-carry;

C[2]=temp[0];

carry=temp[1];

count=4;

end

else if (count==4)

begin

temp=A[3]-B[3]-carry;

C[3]=temp[0];

carry=temp[1];

if (C==0)

ZF=1;

else

ZF=0;

SF=C[3];

CF=carry;

carry=0;

count=0;

end

end

end

end

endmodule