CSE460 LAB PROJECT REPORT

1st Rakibul Hassan Hredoy
20101357
CSE Department
Brac University
Dhaka, Bangladesh
rakibul.hassan.hredoy@g.bracu.ac.bd

2nd Weney Hasan Shammo
19101601
CSE Department
Brac University
Dhaka, Bangladesh
weney.hasan.shammo@g.bracu.ac.bd

3rd Khaled Mushahed Hossain
20101297
CSE Department)
Brac University)
Dhaka, Bangladesh
khaled.mushahed.hossain@g.bracu.ac.bd

4th G. M. Refatul Islam
20101482
CSE Department
Brac University)
Dhaka, Bangladesh
gm.refatul.islam@g.bracu.ac.bd

5th Tahzib Azad
19101464
CSE Department
Brac University)
Dhaka, Bangladesh
tahzib.azad@g.bracu.ac.bd

***Abstract*—The aim of this project is to build a complex FSM that will implement a Mealy machine that takes two 4-bit inputs along with a 3-bit opcode as input to perform the operations RESET, NAND, ADD, OR and SUB.**

***Index Terms*—Mealy, FSM, NAND, verilog**

I. INTRODUCTION

A. Problem Statement

Design a 4-bit ALU capable of performing 4 different arithmetic or logical operations using Quartus, implement it using Verilog HDL, and verify it using a timing diagram.The ALU takes two four-bit inputs: A, B, and a three-bit operation code (opcode). Based on the opcode, it performs five different operations i.e. RESET, NAND, ADD, OR and SUB, on A and B and produces a four-bit output, C. Depending on the result of a particular operation, the ALU also produces three flags: carry, zero and sign flag.

II. OPERATIONS

A. State Diagram

B. Timing Diagram

We use the inputs A=0110 and B=1010

1) RESET: If we get 000 in opcode we would reset the values for count and temporary carry value i.e. ‘car’ and we would keep the values for output, C and the values of the flags same as we have to hold the previous states value and display as output.

2) NAND: For NAND operation, we apply the opcode 001. In the timing diagram we see that at first the whole output, C is 0000. After the first clock cycle our ALU calculates (A0 NAND B0) and stores the value in C0.So after the first clock cycle, we can see that the output is updated to 0001. The value of the zero flag, ZF is also updated in this first clock cycle from 1 to 0 since the output, C is not zero. In the second clock cycle, our ALU calculates (A1 NAND B1) and stores the value in C1. So after the second clock cycle, we can see that the output is updated to 0001.Then in the third clock cycle, our ALU calculates (A2 NAND B2) and stores the value in C2. So after the third clock cycle, we can see that the output is updated to 0101. Then in the fourth i.e. the last clock cycle for this operation, our ALU calculates (A3 NAND B3) and stores the value in C2. So after the third clock cycle, we can see that

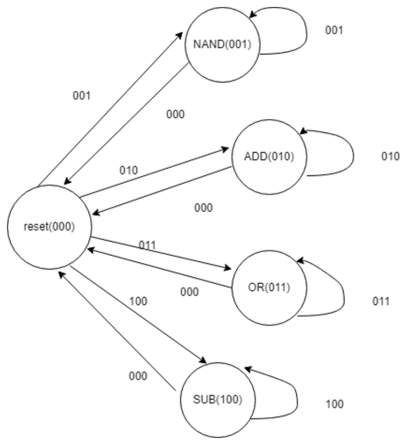

Fig. 1. State Diagram for FSM.

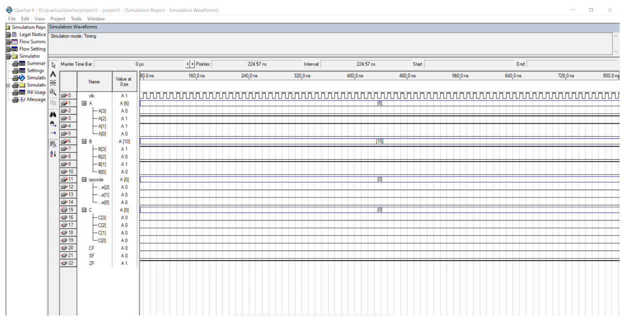

Fig. 2. Timing Diagram for RESET.

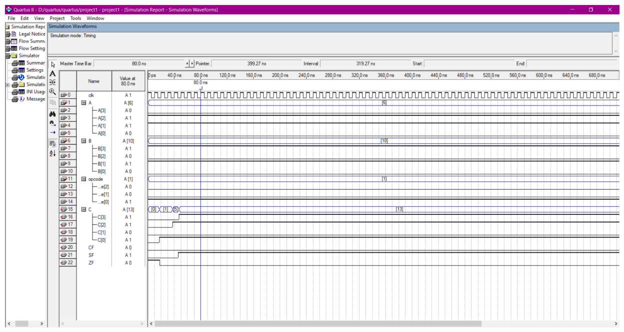

Fig. 3. Timing Diagram for NAND.

the output is updated to 1101(13 in decimal). The value of the sign flag, SF is also updated in this final clock cycle from 0 to 1 since the the MSB i.e. C3 of the output, C is 1.

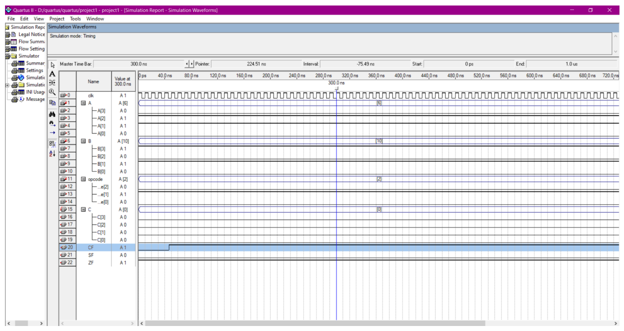

Fig. 4. Timing Diagram for ADD.

3) ADD: For ADD operation, we apply the opcode 010. In the timing diagram we see that at first the output, C is 0000. After the first clock cycle our ALU calculates (A0 ADD B0) and stores the value in C0. So after the first clock cycle, we can see that the output is updated to 0000. In the second clock cycle, our ALU calculates (A1 ADD B1) and stores the value in C1. So after the second clock cycle, we can see that the output is updated to 0000. Then in the third clock cycle, our ALU calculates (A2 ADD B2) and stores the value in C2. So after the third clock cycle, we can see that the output is updated to 0000. Then in the fourth i.e. the last clock cycle for this operation, our ALU calculates (A3 ADD B3) and stores the value in C2. So after the third clock cycle, we can see that the output is updated to 0000.(13 in decimal). The value of the carry flag, CF is updated in this final clock cycle from 0 to 1 since there is a carry value of 1. The value of the zero flag, ZF stays 1 during this entire operation since the the last 4-bits of the output, C is always zero.

4) OR: For OR operation, we apply the opcode 011. In the timing diagram we see that at first the output, C is 0000. After the first clock cycle our ALU calculates (A0 OR B0) and stores the value in C0. So after the first clock cycle, we can see that the output is updated to 0000. In the second clock cycle, our ALU calculates (A1 OR B1) and stores the value in C1. So after the second clock cycle, we can see that the output is updated to 0010. The value of the zero flag, ZF is

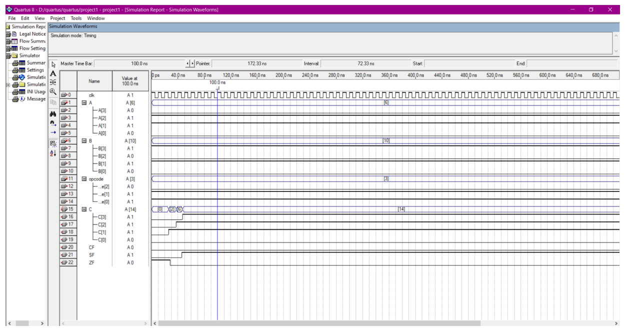

Fig. 5. Timing Diagram for OR.

also updated in this second clock cycle from 1 to 0 since the the last 4-bits of the output, C is not zero. Then in the third clock cycle, our ALU calculates (A2 OR B2) and stores the value in C2. So after the third clock cycle, we can see that the output is updated to 0110. Then in the fourth i.e. the last clock cycle for this operation, our ALU calculates (A3 OR B3) and stores the value in C2. So after the third clock cycle, we can see that the output is updated to 1110.(14 in decimal). The value of the sign flag, SF is also updated in this final clock cycle from 0 to 1 since the the MSB i.e. C3 of the output, C is 1.

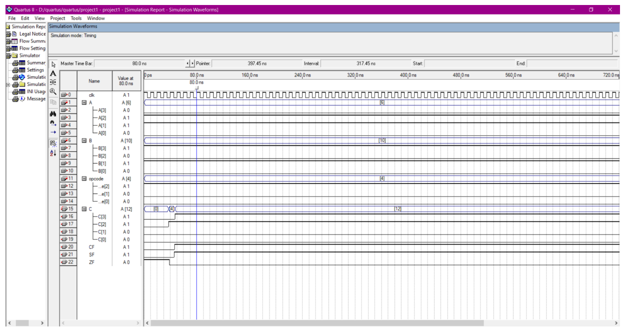

Fig. 6. Timing Diagram for SUB.

5) SUB: For SUB operation, we apply the opcode 011. In the timing diagram we see that at first the output, C is 0000. After the first clock cycle our ALU calculates (A0 SUB B0) and stores the value in C0. So after the first clock cycle, we can see that the output is updated to 0000. In the second clock cycle, our ALU calculates (A1 SUB B1) and stores the value in C1. So after the second clock cycle, we can see that the output is updated to 0000. Then in the third clock cycle, our ALU calculates (A2 SUB B2) and stores the value in C2. So after the third clock cycle, we can see that the output is updated to 0100. The value of the zero flag, ZF is also updated in this third clock cycle from 1 to 0 since the the last 4-bits of the output, C is not zero. Then in the fourth i.e. the last clock cycle for this operation, our ALU calculates (A3 SUB B3) and stores the value in C2. So after the third clock cycle, we can see that the output is updated to 1100.(12 in decimal). The value of the sign flag, SF is also updated in this final clock cycle from 0 to 1 since the the MSB i.e. C3 of the output, C is 1.

C. *Project Methodlogy*

The 3-bit opcodes are set first for each of the different operations. The opcode 000, 001, 010, 011, 100 for RESET, NAND, ADD, OR and SUB respectively. The code is then written using cases for each of the operations and setting conditions and flags accordingly.

III. Conclusion

From the timing diagrams above we can see that the Verilog code successfully executes the operations: RESET, NAND, ADD, OR and SUB, as per the inputs of the opcodes and the variables A and B.

IV. APPENDIX

```
module project1(A,B,C,opcode,clk,ZF,SF,CF);
input [3:0]A,B;
input [2:0]opcode;
input clk;

reg [1:0]sum = 0,count = 0;
reg [3:0]  B2s;
reg car = 0;

output reg [3:0]C;
output reg ZF=0,SF=0,CF=0;

always @(posedge clk)
begin
case(opcode)
3'b000: //RESET
begin
count=0;car=0;
end

3'b001: ///NAND
begin
if(count==0)
begin
C[0]=~(A[0] & B[0]);
count=1;
end
else if(count==1)
begin
C[1]=~(A[1] & B[1]);
count=2;
end
else if(count==2)
begin
C[2]=~(A[2] & B[2]);
count=3;
end
else if(count==3)
begin
C[3]=~(A[3] & B[3]);
count=0;
end
end




3'b010: //ADD
begin
if(count==0)
begin
sum=(A[0]+B[0]);
car=sum[1];
C[0]=sum[0];
count=1;
end
else if(count==1)
begin
sum=(A[1]+B[1]+car);
C[1]=sum[0];
car=sum[1];
count=2;
end
else if(count==2)
begin
sum=(A[2]+B[2]+car);
C[2]=sum[0];
car=sum[1];
count=3;
end
else if(count==3)
begin
sum=(A[3]+B[3]+car);
C[3]=sum[0];
car=sum[1];
count=0;
CF = car==1;
end
end


3'b011: //AND
begin
if(count==0)
begin
C[0]= A[0] | B[0];
count=1;
end
else if(count==1)
begin
C[1]=A[1] | B[1];
count=2;
end
else if(count==2)
begin
```

```
C[2]=A[2] | B[2];
count=3;
end
else if(count==3)
begin
C[3]=A[3] | B[3];
count=0;
end
end

3’b100: //SUB
begin
B2s=-B;
if(count==0)
begin
sum=(A[0]+B2s[0]);
car=sum[1];
C[0]=sum[0];
count=1;
end
else if(count==1)
begin
sum=(A[1]+B2s[1]+car);
C[1]=sum[0];
car=sum[1];
count=2;
end
else if(count==2)
begin
sum=(A[2]+B2s[2]+car);
C[2]=sum[0];
car=sum[1];
count=3;
end
else if(count==3)
begin
sum=(A[3]+B2s[3]+car);
C[3]=sum[0];
car=sum[1];
count=0;
CF = car==1;
end
end
endcase
end
always @(C)
begin
SF = C[3]==1’b1;
ZF = C==4’b0000;
end
endmodule
```