A 4-bit ALU capable of performing 4 different arithmetic or logical operations implemented using Verilog HDL

Saman Sarker Joy
Department of CSE
BRAC University
Dhaka, Bangladesh
ID: 20101114

Raufar Mostafa
Department of CSE
BRAC University
Dhaka, Bangladesh
ID: 20101312

Maliha Jahan Maisha
Department of CSE
BRAC University
Dhaka, Bangladesh
ID: 20101527

Nowreen Tarannum Rafa
Department of CSE
BRAC University
Dhaka, Bangladesh
ID: 20101329

Ariyan Hossain
Department of CSE
BRAC University
Dhaka, Bangladesh
ID: 20101099

***Abstract*—In this project, two four-bit inputs are used to conduct five different operations inside the ALU, including resetting the state using three-bit opcodes in Quartus using Verilog code. Establishing the clockwise conditions allows bitwise operations to be done. The ALU produces the four-bit result and generates three flags based on the outcome of a given operation: carry, zero, and sign-flag are observed, verifying the timing diagrams with the state diagram and expected outcomes.**

I. Introduction

We use Quartus for the design and Verilog HDL to implement a 4-bit ALU that can perform 4 different arithmetic or logical operations. Here we take two 4-bit inputs- A, B, and a 3-bit operation code (opcode) to perform the operation. From this, we can perform five different operations on A and B and produce a 4-bit output C. After we get the output, ALU produces 3 flags such as- Carry Flag(CF), Sign Flag(SF), Zero Flag(ZF). We get CF=1 when the carry/borrow from the output is 1. SF=1 when the MSB of the output C is 1. ZF=1 when the output C is 0.

When the opcode op0=000 the RESET operation takes place. This means the ALU will remain at an idle state and output C is retained from the previous operation. If op1=001, then XOR operation occurs. From A and B, the ALU executes a bitwise XOR operation at the positive edge of the input clock. When op2=010 SUB operation takes place. ALU subtracts B from A and the MSB's are the sign bits and the remaining bits represent the operands' values. If op=011 NAND operation is performed on A and B. Finally op4=100 the ADD operation is initiated on A and B. We use temp to calculate bitwise operations and carry_bit to keep the carry value. For SUB operation we take borrow_bit when subtracting 1 from 0.

II. Operations

There are five different operations (RESET, XOR, SUB, NAND, ADD) where states have been changed based on the opcodes. The explanations of all these operations are the following:

A. State Diagram

Below is the state diagram:

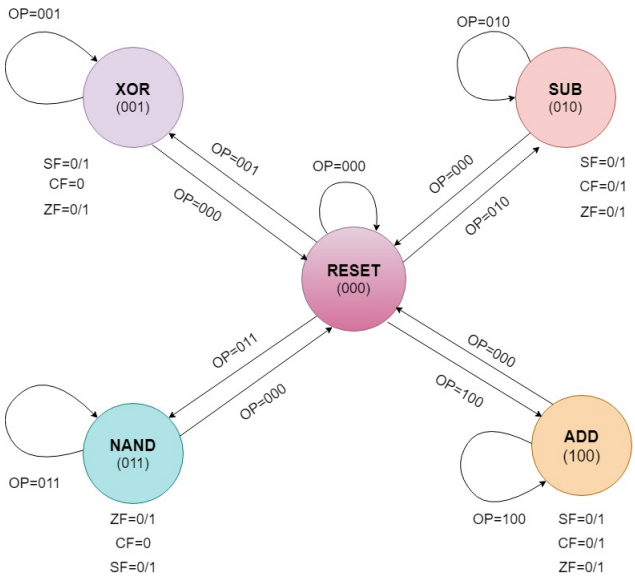

Fig. 1. State Diagram

B. Timing Diagram Explanations of All the Instructions

1) RESET: In the RESET state, we set the bitcount to be 000. Additionally, we also set the values of temp=0, carry_bit and borrow_bit=0. If there was any previous value of the output C, it is retained in the RESET state.

2) XOR: XOR performs xor between two input bits A[n], B[n], and generates one output bit C[n]. This operation is performed bit-by-bit starting from the LSB and progressing to the MSB. For XOR operation at first, we will check if the opcode=001 is matched or not, and then we will perform bitwise XOR. If the bits are the same, the result is 0 and if the bits are different, the result is 1.

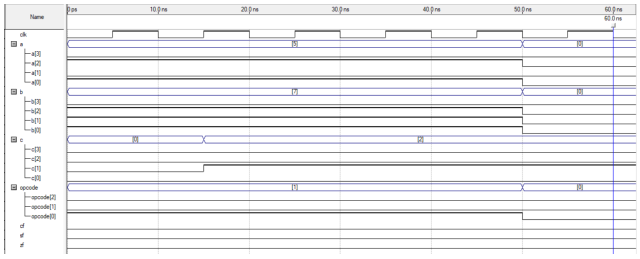

Fig. 2. XOR Timing Diagram

From the timing diagram, we can see A=5(0101) and B=7(0111). Here bitwise operation takes place, as A[0]=1 and B[0]=1 produces C[0]=0, A[1]=0 and B[1]=0 produces C[1]=1, A[2]=1 and B[2]=1 produces C[2]=1 and A[3]=0 and B[3]=0 produces C[3]=0. The final result is C=2(0010). After we do each bitwise operation, we change the current bit to the next bit. The sign bit C[3] is 0 hence Sign Flag, SF is 0 as shown in the timing diagram. As there is no carry bit in this case hence Carry Flag, CF is 0. Lastly, as the output C is a non-zero number hence Zero Flag (ZF) is also 0.

3) SUB: SUB performs addition between two input bits A[n], B_comp[n] where B_comp is the 2's complement of input B and generates one output bit C[n]. Through this, we can get the subtraction of B[n] from A[n]. This operation is performed bit-by-bit starting from the LSB and progressing to the MSB. For SUB operation, at first, we will check if the opcode=010 is matched or not, and then we will perform bitwise SUB.

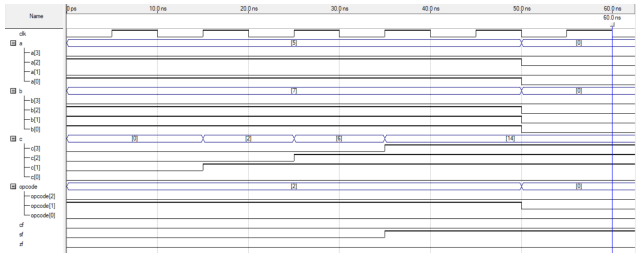

Fig. 3. SUB Timing Diagram

From the timing diagram, we can see A=5(0101) and B=7(0111). Here bitwise operation takes place, as A[0]=1 and B[0]=1 produces C[0]=0, A[1]=0 and B[1]=1 produces C[1]=1, A[2]=1 and B[2]=1 produces C[2]=1 and A[3]=0 and B[3]=0 produces C[3]=1. The final result is C=-2(1110). After we do each bitwise operation, we change the current bit to the next bit. The sign bit C[3] is 1 hence Sign Flag, SF is 1 as shown in the timing diagram. As there is no borrow bit in this case hence Carry Flag, CF is 0. Lastly, as the output C is a non-zero number hence Zero Flag (ZF) is also 0.

4) NAND: NAND performs nand between two input bits A[n], B[n], and generates one output bit C[n]. This operation is performed bit-by-bit starting from the LSB and progressing to the MSB. For NAND operation at first, we will check if the opcode=011 is matched or not, and then we will perform bitwise NAND.

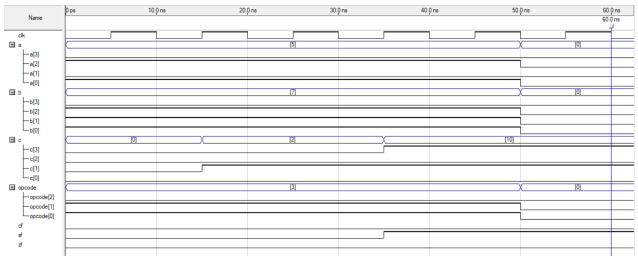

Fig. 4. NAND Timing Diagram

From the timing diagram, we can see A=5(0101) and B=7(0111). Here bitwise operation takes place, as A[0]=1 and B[0]=1 produces C[0]=0, A[1]=0 and B[1]=1 produces C[1]=1, A[2]=1 and B[2]=1 produces C[2]=0 and A[3]=0 and B[3]=0 produces C[3]=1. The final result is C=10(1010). After we do each bitwise operation, we change the current bit to the next bit. The sign bit C[3] is 1 hence Sign Flag, SF is 1 as shown in the timing diagram. As there is no carry bit in this case hence Carry Flag, CF is 0. Lastly, as the output C is a non-zero number hence Zero Flag (ZF) is also 0.

5) ADD: ADD performs addition between two input bits A[n], B[n] and generates one output bit C[n]. Through this, we can get the addition of A[n] with B[n] sequentially. This operation is performed bit-by-bit starting from the LSB and progressing to the MSB. For ADD operation, at first, we will check if the opcode= 100 is matched or not, and then we will perform bitwise ADD.

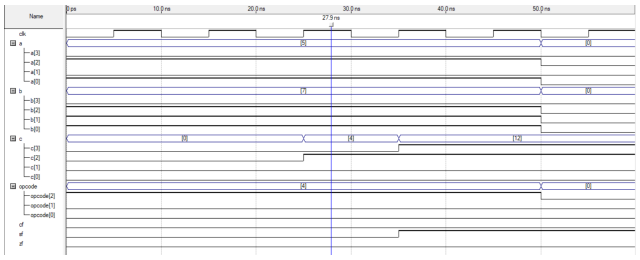

Fig. 5. ADD Timing Diagram

From the timing diagram, we can see A=5(0101) and B=7(0111). Here bitwise operation takes place, as A[0]=1 and B[0]=1 produces C[0]=0, A[1]=0 and B[1]=1 produces C[1]=0 , A[2]=1 and B[2]=1 produces C[2]=1 and A[3]=0 and B[3]=0 produces C[3]=1. The final result is C=12(1100). After we do each bitwise operation, we change the current bit to the next bit. The sign bit C[3] is 1 hence Sign Flag, SF is 1 as shown in the timing diagram. As there is no borrow bit in this case hence Carry Flag, CF is 0. Lastly, as the output C is a non-zero number hence Zero Flag (ZF) is also 0.

III. Conclusion

To conclude this report, the timing diagrams of five distinct operations of XOR, ADD, SUB, NAND, and RESET have been verified and achieved 4-bit produced output C in bit-by-bit operations inside the ALU when the opcode is operating by assisting three-bit opcodes. Depending on the four-bit output C, the three corresponding flag values for the zero flag, sign flag, and carry flag are also set. The state diagram depicts the state transitions after receiving 3-bit opcodes, with RESET remaining as the idle state where no action will occur and output C retaining the result of the previous operation executed.

Lastly, the Timing Diagram is used to verify the outputs with external inputs of A, B, and opcode to perform the operation serially at the positive edge of the input clock i.e. during the first clock cycle, the operation will be performed on the LSBs (A[0], B[0]) storing the result in C[0] and similarly for next clock cycles. Therefore, a 4-bit ALU has been designed which is capable of performing 4 different arithmetic or logical operations using Quartus, implementing it using Verilog HDL, and verifying it using timing diagrams.

IV. APPENDIX

```
module cse460_project_group3(clk, a, b, c, opcode, zf, cf, sf);

input [3:0]a, b;
input [2:0]opcode;
input clk;

output reg [3:0]c;
output reg sf, cf, zf;

reg [1:0] carry_bit, borrow_bit;
reg [2:0] temp;
reg [3:0] bitcount = 3'b000;
reg [3:0] b_comp;

parameter   op0=3'b000, //reset
          op1=3'b001, //xor
          op2=3'b010, //sub
          op3=3'b011, //nand
          op4=3'b100; //add

always @(posedge clk)
   begin
      case (opcode)
         op0: begin   //reset
                                          bitcount = 3'b000;
                                          temp = 0;
                                          carry_bit = 0;
                                          borrow_bit = 0;
                                         end

                                op1: begin  //xor
                                          if (bitcount == 3'b000)
                                                   begin
                                                    c = 4'b0000;
                                                    sf = 0;
                                                    cf = 0;
                                                    zf = 0;
                                                    c[0] = a[0] ^ b[0];
                                                    bitcount = 3'b001;
                                                   end
                                          else if (bitcount == 3'b001)
                                                   begin
                                                    c[1] = a[1] ^ b[1];
                                                    bitcount = 3'b010;
                                                   end
```

```
        else if (bitcount == 3'b010)
                begin
                 c[2] = a[2] ^ b[2];
                 bitcount = 3'b011;
                end
         else if (bitcount == 3'b011)
                begin
                 c[3] = a[3] ^ b[3];
                 bitcount = 3'b100;

                 if (c == 4'b0000)
                        zf = 1;
                 if (c[3] == 1)
                        sf = 1;
                end
end

op2: begin  //sub
        b_comp = -b;
        if (bitcount == 3'b000)
                begin
                 c = 4'b0000;
                 sf = 0;
                 cf = 0;
                 zf = 0;
                 temp = a[0] + b_comp[0];
                 c[0] = temp[0];
                 carry_bit = temp[1];
                 if(a[0] == 0 && b[0] == 1)
                 begin
                        borrow_bit = 1;
                 end
                 bitcount = 3'b001;
        end

        else if (bitcount == 3'b001)
                begin
                 temp = a[1] + b_comp[1] + carry_bit;
                 c[1] = temp[0];
                 carry_bit = temp[1];
                 if (borrow_bit == 1)
                 begin
                        if(a[0] == 1 && b[0] == 0)
                                begin
                                        borrow_bit = 0;
                                end
                 end
                 else
                 begin
```

```
                        if(a[0] == 0 && b[0] == 1)
                                begin
                                        borrow_bit = 1;
                                end
                end
                bitcount = 3'b010;
                end
        else if (bitcount == 3'b010)
                begin
                temp = a[2] + b_comp[2] + carry_bit;
                c[2] = temp[0];
                carry_bit = temp[1];
                if (borrow_bit == 1)
                begin
                        if(a[0] == 1 && b[0] == 0)
                                begin
                                        borrow_bit = 0;
                                end
                end
                else
                begin
                        if(a[0] == 0 && b[0] == 1)
                                begin
                                        borrow_bit = 1;
                                end
                end
                bitcount = 3'b011;
                end
         else if (bitcount == 3'b011)
                begin
                temp = a[3] + b_comp[3] + carry_bit;
                c[3] = temp[0];
                carry_bit = temp[1];
                if (borrow_bit == 1)
                begin
                        if(a[0] == 1 && b[0] == 0)
                                begin
                                        borrow_bit = 0;
                                end
                end
                else
                begin
                        if(a[0] == 0 && b[0] == 1)
                                begin
                                        borrow_bit = 1;
                                end
                end

                 if (c == 4'b0000)
```

```
                                        zf = 1;
                        if (borrow_bit == 1)
                                        cf = 1;
                        if (c[3] == 1)
                                        sf = 1;
                      end
        end

        op3: begin  //nand
                if (bitcount == 3'b000)
                        begin
                        c = 4'b0000;
                        sf = 0;
                        cf = 0;
                        zf = 0;
                        c[0] = ~(a[0] & b[0]);
                        bitcount = 3'b001;
                        end
                else if (bitcount == 3'b001)
                        begin
                        c[1] = ~(a[1] & b[1]);
                        bitcount = 3'b010;
                        end
                else if (bitcount == 3'b010)
                        begin
                        c[2] = ~(a[2] & b[2]);
                        bitcount = 3'b011;
                        end
                 else if (bitcount == 3'b011)
                         begin
                        c[3] = ~(a[3] & b[3]);
                        bitcount = 3'b100;

                        if (c == 4'b0000)
                                zf = 1;
                        if (c[3] == 1)
                                sf = 1;
                      end
        end

op4: begin  //add
                if (bitcount == 3'b000)
                        begin
                        c = 4'b0000;
                        sf = 0;
                        cf = 0;
                        zf = 0;
                        temp = a[0] + b[0];
                        c[0] = temp[0];
```

```
                        carry_bit = temp[1];
                        bitcount = 3'b001;
                       end
                else if (bitcount == 3'b001)
                       begin
                        temp = a[1] + b[1] + carry_bit;
                        c[1] = temp[0];
                        carry_bit = temp[1];
                        bitcount = 3'b010;
                       end
                else if (bitcount == 3'b010)
                       begin
                        temp = a[2] + b[2] + carry_bit;
                        c[2] = temp[0];
                        carry_bit = temp[1];
                        bitcount = 3'b011;
                       end
                 else if (bitcount == 3'b011)
                       begin
                        temp = a[3] + b[3] + carry_bit;
                        c[3] = temp[0];
                        carry_bit = temp[1];
                        bitcount = 3'b100;

                         if (c == 4'b0000)
                                        zf = 1;
                         if (carry_bit == 1)
                                        cf = 1;
                         if (c[3] == 1)
                                        sf = 1;
                       end
            end
            endcase
      end


endmodule
```