Report for Lab1: ALU

20140843 Taekang Eom

20150384 Eunyoung Hyung

1. Introduction

In Lab1, we need to implement the Arithmetic Logic Unit in Verilog. There are three inputs; the input A and B are 16-bit signed binary numbers and the input FuncCode represents the operator. This ALU gives two outputs; the output C is the result of the operation and the output OverflowFlag flags whether the result is overflowed. By doing this assignment, we need to learn and the syntax of Verilog better and how the bit operations are done.

2. Design

The things we need to think more carefully are only when the FuncCode is 0000(Signed addition), 0001(Signed subtraction), 0110(NAND), 0111(NOR), 1000(XOR), 1001(XNOR) and 1101(Arithmetic right shift); other operations are just done by using Verilog basic operation.

1) 0000(Signed addition), 0001(Signed subtraction)

In two's complement system, we need to care about the MSB which represents the sign of the number. The result of adding two positive numbers A and B can be negative and also the result of two negative numbers can be positive due to this sign bit and it's called overflow; in this case, the result is the wrong answer so we need to flag it.

We can detect the overflow by checking the MSB and Carry bit. Carry bit part is at the position which is one bit larger than MSB. When the two MSBs of the added numbers A and B are same, but MSB of the result is different from the Carry bit of the result, we can say there is an overflow.

In terms of signed subtraction, we can just make the second operand be negative by negating it and then adding 1. And then we can just do the same thing in Signed addition.

2) NAND, NOR

There is no one single operation for NAND and NOR in Verilog so we need to use "AND", "OR", and "NEGATION". We can just apply the negation to the result of the "AND" and "OR" operations to get NAND and NOR, respectively.

3) XOR, XNOR

There is no single operation for XOR and XNOR in Verilog so we need to use "AND", "OR", and "NEGATION". XOR operates like the table below, so we can say C = ((~A)B + (~B)A). XNOR is simply the negation of the result of XOR.

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

4) 1101(Arithmetic right shift)

In two's complement system, MSB represents the sign of the number. So when we do arithmetic right shift, we need to keep this sign.

3. Implementation

We separate the cases, depending on the input FuncCode.

```
always @(*) begin
        OverflowFlag = 0;
        if (FuncCode == 4'b0000) begin
                ltemp = {1'b1,A[15:0]} + {1'b1,B[15:0]};
                C = ltemp[15:0];
                if (A[15] == B[15] && ltemp[16] != ltemp[15]) begin
                        OverflowFlag = 1;
                end
        end
        else if (FuncCode == 4'b0001) begin
                negB = ~B+1;
                ltemp = {1'b1,A[15:0]} + {1'b1,negB[15:0]};
                C = ltemp[15:0];
                if (A[15] == negB[15] && ltemp[16] != ltemp[15]) begin
                        OverflowFlag = 1;
                end
        end
```

For signed addition and subtraction, we can check the overflow by using the MSB and Carry bit as 2-1) explains.

```
else if (FuncCode == 4'b0010) begin
        C = A;
end
else if (FuncCode == 4'b0011) begin
        C = ~A;
end
else if (FuncCode == 4'b0100) begin
        C = A&B;
end
else if (FuncCode == 4'b0101) begin
        C = A|B;
end
```

These are just basic operations.

```
else if (FuncCode == 4'b0110) begin
        C = ~(A&B);
end
else if (FuncCode == 4'b0111) begin
        C = ~(A|B);
end
else if (FuncCode == 4'b1000) begin
        C = ((~A)&B) | ((~B)&A);
end
else if (FuncCode == 4'b1001) begin
        C = ((~B)|A) & ((~A)|B);
end
```

It's just same as the 2-2) and 2-3) explain. In terms of XNOR, it's negation of the result of XOR; (~A)B + (~B)A. So, C = ~((~A)B + (~B)A) = (A + (~B)) & (B + (~A))

```
        else if (FuncCode == 4'b1010) begin
                C = A << 1;
        end
        else if (FuncCode == 4'b1011) begin
                C = A >> 1;
        end
        else if (FuncCode == 4'b1100) begin
                C = A << 1;
        end
        else if (FuncCode == 4'b1101) begin
                C = A >> 1;
                C = {A[15],C[14:0]};
        end
        else if (FuncCode == 4'b1110) begin
                C = ~A+1;
        end
        else begin
                C = 0;
        end
    end
endmodule
```

As explained in 2-4), we need to care of MSB when we do arithmetic right shift.

4. Discussion

This Lab 1 is quite easy for us, so we don't have many things to discuss. The only thing we needed to think carefully was to care of MSB in signed addition and subtraction to detect the overflow. To do this, we use the carry bit and MSB.

5. Conclusion

We've learned the syntax of the Verilog sufficiently and we totally understand bitwise operations.