Computer Architecture HW2

8-bit Addition/Subtraction Module

2012147562 최인호

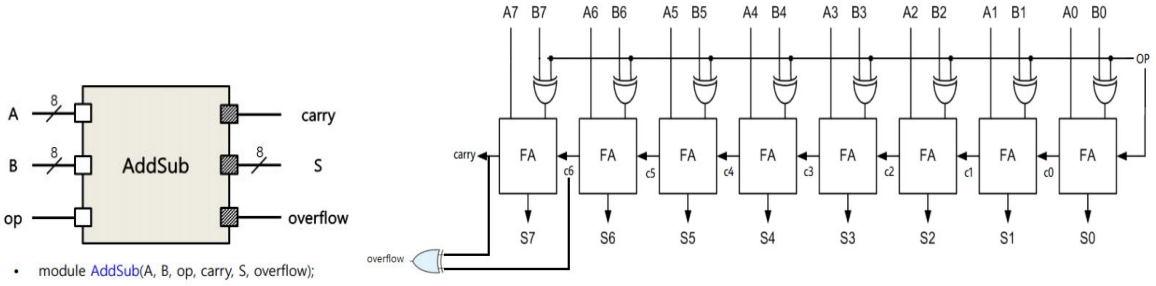
- OS : Windows 10 Enterprise (Administrator 계정)

1) 모듈 구현에 대한 전체적 설명

- 기본 개념

이번 HW2 에서는 기본적인 Full-Adder에 XOR 게이트를 조합하고 조금의 수정을 통하여 8-bit Addition/Subtraction Module을 구현하였다. 사용된 주요 개념은 2's complement 개념이다. 즉, signed binary number에서 어떤 값 B의 음수 값인 -B는 (2's complement of B)를 통하여 구해짐을 이용한다. 이 개념을 사용하면, A - B = A + (2's complement of B) 와 같은 방식으로 subtraction을 빼는 값의 2's complement과의 addition으로 볼 수 있다. 여기서 2's complement는 그 이진수의 모든 자리의 숫자를 반전(0을 1로, 1을 0으로)시킨 뒤 여기에 1을 더하여 얻을 수 있다.

위의 내용을 종합하여 8-bit Addition/Subtraction Module의 circuit을 구성하면 다음과 같다.



[Figure 1 : AddSub module interface 와 circuit]

우선, 이 circuit의 input과 output에 대해 정리한다.
A, B : 8-bit input, Addition이나 Subtraction의 Operands이다.
op : 1-bit input, Addition과 Subtraction중 어떤 연산을 적용할 지 지정해준다. 0이면 Addition, 1이면 Subtraction을 적용한다.
carry : 1-bit output, 연산 결과의 MSB(가장 왼쪽 bit)에서 자리올림이 발생할 경우 on(1) 된다.
S : 8-bit output, binary number로 AddSub 모듈의 연산 결과이다.
overflow : 1-bit output, 연산의 결과값이 8-bit signed binary number가 표현할 수 있는 값의 범위를 넘어가는 경우 on(1)된다.

(* 8-bit input/output port들은 MSB first로 처리한다. ex : input[7:0] A;)

위의 circuit에서 op가 0일때는 A + B(Addition), 1일때는 A - B(Subtraction)이 적용되어야 하며, op가 1일때 그 값을 이용하여 B의 각 bit들과 XOR 연산을 적용하고, LSB(가장 오른쪽 bit)의 Full-Adder 연산시 Carry-in bit로 op값을 넣어 줌으로써 2's complement를 구할 때 1을 더해주는 역할을 한다. 이렇게 op값에 따라 Subtraction일 경우 적절하게 (2's complement of B)를 만들어 주었다면, 각 bit에 Full-Adder를 사용해 Addition을 적용하는 것으로 A - B (Subtraction) 연산을 적용할 수 있는 것이다.

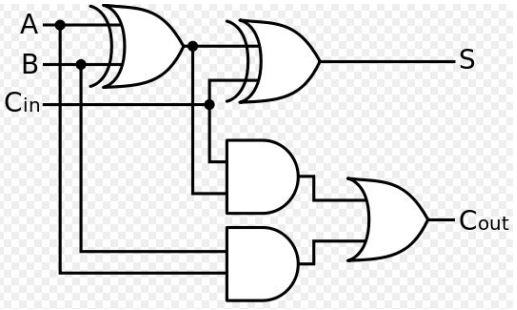
단, 여기서 주의할 점은 Overflow가 발생하는 경우이다. 8-bit signed binary number로 표현할 수 있는 수의 범위는 -128 ~ +127 이다. 따라서, Overflow가 발생하면 연산한 결과의 값이 저 범위를 넘어가고 적절한 결과가 얻어지지 않는다. 이때, Overflow가 발생하는 경우는 MSB 바로 아래 bit의 연산을 수행하는 Full-Adder로부터 발생한 c6값이 부호를 표현하는 MSB에 영향을 줘서 부호가 바뀌는 경우이다. 이것을 식으로 표현하면 overflow = (carry ⊕ c6) 로 나타낼 수 있다.

- **코드 구현**

위에서 설명한 개념을 모듈 interface(module AddSub(A, B, op, carry, S, overflow);)를 만족하여 작성한 코드는 다음과 같다.

```
1  module fulladd(sum, c_out, a, b, c_in);
2
3  output sum, c_out;
4  input a, b, c_in;
5
6  wire s1, c1, c2;
7
8  xor (s1, a, b);
9  and (c1, a, b);
10
11 xor (sum, s1, c_in);
12 and (c2, s1, c_in);
13
14 or (c_out, c2, c1);
15
16 endmodule
17
```

fulladd.v



Full-Adder circuit

```
1  module AddSub(A, B, op, carry, S, overflow);
2
3
4  output [7:0] S;
5  output carry;
6  output overflow;
7  input [7:0] A, B;
8  input op;
9
10 wire c0, c1, c2, c3, c4, c5, c6, x0, x1, x2, x3, x4, x5, x6, x7;
11
12 xor(x0, B[0], op);
13 fulladd fa0(S[0], c0, A[0], x0, op);
14
15 xor(x1, B[1], op);
16 fulladd fa1(S[1], c1, A[1], x1, c0);
17
18 xor(x2, B[2], op);
19 fulladd fa2(S[2], c2, A[2], x2, c1);
20
21 xor(x3, B[3], op);
22 fulladd fa3(S[3], c3, A[3], x3, c2);
23
24 xor(x4, B[4], op);
25 fulladd fa4(S[4], c4, A[4], x4, c3);
26
27 xor(x5, B[5], op);
28 fulladd fa5(S[5], c5, A[5], x5, c4);
29
30 xor(x6, B[6], op);
31 fulladd fa6(S[6], c6, A[6], x6, c5);
32
33 xor(x7, B[7], op);
34 fulladd fa7(S[7], carry, A[7], x7, c6);
35
36 xor(overflow, carry, c6);
37
38 endmodule
```

AddSub.v (circuit은 Figure 1 참조)

- **AddSub.v 소스코드 설명**

우선, 과제에서 제시한 interface를 그대로 따르고 있다. AddSub(A, B, op, carry, S, overflow) 에서, 위에서 설명한 것처럼 각각의 input, output을 declaration한다.

```
4   output [7:0] S;
5   output carry;
6   output overflow;
7   input [7:0] A, B;
8   input op;
```

다음은 내부 wire들을 declaration한다. 이 내부 wire들은 input B의 각 bit와 op의 XOR 연산 값이나(x0~x7), 각 Full-Adder에서 나오는 carry bit들(c0~c6)들을 위해 필요하다.

```
10  wire c0, c1, c2, c3, c4, c5, c6, x0, x1, x2, x3, x4, x5, x6, x7;
```

다음은 각 bit들에 적용되는 실제 연산을 구현한 코드이다. 우선 input B의 각 bit 자리와 op의 XOR 연산 값을 구한 후, 그 값과 A의 각 bit 자리를 input으로 하여 fulladd(sum, c_out, a, b, c_in) 모듈을 호출한다. c_in은 LSB를 위한 fulladd 에서는 op, 그 뒤로는 이전 fulladd의 c_out을 전달한다.

```
12  xor(x0, B[0], op);
13  fulladd fa0(S[0], c0, A[0], x0, op);
14
15  xor(x1, B[1], op);
16  fulladd fa1(S[1], c1, A[1], x1, c0);
```

마지막으로 overflow = (carry ⊕ c6) 식에 의해 overflow를 구한다.

```
36  xor(overflow, carry, c6);
```

2) sample input 에 대한 시뮬레이션

Sample Input & Output

Inputs			Output			Decimal
A	B	op	carry	S	overflow	
00001100	00001100	0	0	00011000	0	12+12 = 24
00001100	00001100	1	1	00000000	0	12-12 = 0
00001101	00011001	1	0	11110100	0	13-25 = (-12)
01100100	00110010	0	0	10010110	1	100+50 = 150
10110000	00111100	1	1	01110100	1	(-80)-60 = (-140)

```
module stimulus;

reg [7:0] A, B;
reg op;
wire overflow;
wire [7:0] S;
wire carry;
reg [31:0] STD_ID;

AddSub AddSub1(A, B, op, carry, S, overflow);

initial
begin
        $monitor($time, " A = %b, B = %b, op = %b, --- carry = %b, S = %b, overflow = %b\n", A, B, op, carry, S, overflow);
end

initial
begin
        STD_ID = 32'd2012147562;

        A = 8'b00001100; B = 8'b00001100; op = 1'b0;
        #5 A = 8'b00001100; B = 8'b00001100; op = 1'b1;
        #5 A = 8'b00001101; B = 8'b00011001; op = 1'b1;
        #5 A = 8'b01100100; B = 8'b00110010; op = 1'b0;
        #5 A = 8'b10110000; B = 8'b00111100; op = 1'b1;

end

endmodule
```

stimulus.v

stimulus.v에서 Sample Input에 대한 simulation 코드를 작성하고, simulation한 결과 알맞은 결과가 출력됨을 wave 폼으로 확인할 수 있었다.

$monitor 을 이용하여 실행 결과를 Transcript에 출력하였으며, 그 결과는 다음과 같다.

```
add wave -position insertpoint sim:/stimulus/*
VSIM 7> run -all
#                    0 A = 00001100, B = 00001100, op = 0, --- carry = 0, S = 00011000, overflow = 0
#
#                    5 A = 00001100, B = 00001100, op = 1, --- carry = 1, S = 00000000, overflow = 0
#
#                   10 A = 00001101, B = 00011001, op = 1, --- carry = 0, S = 11110100, overflow = 0
#
#                   15 A = 01100100, B = 00110010, op = 0, --- carry = 0, S = 10010110, overflow = 1
#
#                   20 A = 10110000, B = 00111100, op = 1, --- carry = 1, S = 01110100, overflow = 1
#
```

References :

http://www.labri.fr/perso/strandh/Teaching/AMP/Common/Strandh-Tutorial/circuits-for-binary-arithmetic.html
https://electronics.stackexchange.com/questions/98015/subtraction-using-adder-circuit
https://commons.wikimedia.org/wiki/File:Full-adder.svg