## **SOC Design**

한양대학교 공과대학 컴퓨터소프트웨어학부

최가온(학번: 2019009261)

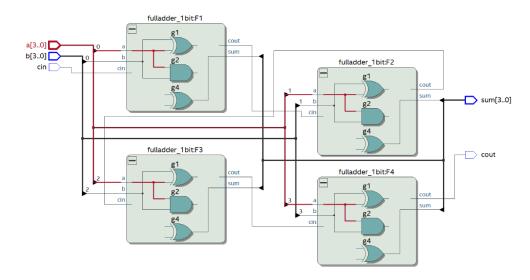
[Homework 1] Build a 4-bit full adder and verify in simulation.

## 1. Code

https://github.com/Gaon-Choi/ITE4003/blob/main/Week04/fulladder\_4bit/tb\_fulladder\_4bit.v

```
module fulladder_4bit(a, b, cin, sum, cout);
    input [3:0] a, b;
     input cin;
    output [3:0] sum;
    output cout;
    // use wire to store carry
    wire [2:0] c;
    fulladder_1bit F1(a[0], b[0], cin, sum[0], c[0]);
    fulladder_1bit F2(a[1], b[1], c[0], sum[1], c[1]);
fulladder_1bit F3(a[2], b[2], c[1], sum[2], c[2]);
fulladder_1bit F4(a[3], b[3], c[2], sum[3], cout);
endmodule
module fulladder_1bit(a, b, cin, sum, cout);
    input a, b, cin;
    output sum, cout;
    wire s1, c1, c2;
    xor g1(s1, a, b);
     and g2(c1, a, b);
    and g3(c2, s1, cin);
    xor g4(sum, s1, cin);
    xor g5(cout, c1, c2);
endmodule
```

RTL Viewer를 통해 위의 구현을 시각적으로 관찰한 결과는 아래와 같다.



## 2. Testbench

```
module tb_fulladder_4bit;
    parameter N = 4;
    reg [N-1:0] a;
    reg [N-1:0] b;
    reg c_in;
    wire [3:0] sum;
    wire c_cout;
    fulladder_4bit fa4(.a(a), .b(b), .cin(c_in), .sum(sum), .cout(c_cout));
    integer i;
    // Test start
    initial
    begin
        // when cin == 0
        c_{in} = 1'b0;
        for(i=0; i<2**(2*N); i=i+1)</pre>
            {b, a} = i; #10;
            $display("a=", a, " b=", b, " sum=", sum, " cout=", c_cout);
        end
        // when cin == 1
        c_{in} = 1'b1;
        for(i=0; i<2**(2*N); i=i+1)
        begin
            {b, a} = i; #10;
            $display("a=", a, " b=", b, " sum=", sum, " cout=", c_cout);
        end
    end
endmodule
```

1) a와 b는 각각 4비트의 크기를 가진다. 하나의 수는 b'000부터 b'111까지 16개의 경우의 수를 가지고 있기에 총 256가지의 테스트케이스를 모두 테스트하였다. 0부터 255까지의 수를 8비트로 나타낸 후 각각 4비트, 4비트로 쪼개어 a, b에 할당하는 방식으로 전체 경우의 수를 실현하였다. 또한 c\_in의 값이 0, 1일 때 각각 for문을 사용하여 총 512가지이다. 파동 결과(waveform result)로 확인이 가능하나, 가시성을 위해 \$display문을 이용하여 각 케이스별로 세부 값들을 출력하였다.

## 2) waveform result

a와 b는 각각 4-bit이지만 둘을 합한 결과는 8-bit 공간에 담아야 오버플로우(overflow) 없이 계산할수 있다. 그러나, sum을 4-bit로 설정하였기에 그것의 결과가 1111이 넘어가게 되면 cout의 값이 1이 되어야 하는 것이 합당하다. 아래의 결과에서 sum=b'1111 이후의 스텝에서 cout이 b'1이 되는 것을 확인할 수 있다. (위에서부터 각각 a, b, cin, sum, cout, i의 값을 나타낸 것이다.)

	0000	0001	0010	0011	0100	0101	0110 (0	111	1000 (1	001	1010	1011	1100	101	1110	1111	0000	(000	1 00	10 (00	011	0100
/tb_fulladder_4bit/b 1111	0000									=							0001					
/tb_fulladder_4bit/c 1			<u> </u>	V		v		$\Rightarrow$		$\Rightarrow$		$\Longrightarrow$		$\rightarrow$	_				$\rightarrow$		$\Rightarrow$	
/tb_fulladder_4bit/s 1111 /tb_fulladder_4bit/c St1	0000	0001	10010	10011	10100	10101	0110 (0	111	1000 11	001	1010	1011	1100 ) 1	101	1110	1111	0001	1,001	0 100	)11   (01	100	0101 ]
tb_fulladder_4bit/i 256		1	12	ĬЗ	14	<u> 1</u> 5	16 Y		8 <u>1</u> 9	-	10 Y	11	12 ) 1	3 [	14	15	16	<u> 17</u>	18	19	9 1	20 ¥21
											•		<u> </u>									
/th fulladder 4hit/a 1111	Y 101	1 110	n Y 110	1 111	0 Y 111	1 1000	0 10001	0010	Y0011	0100	Y0101	0110	Y0111	1000	Ý 100	1 101	0 (10	11 1	1100	1101	1110	Y 1111
// /tb_fulladder_4bit/a 1111	(101	1 110	0 (110	1 111	0 (111	1 000		0010	(0011	0100	(0101	0110	(0111	1000	(100	1 101	0 (10	11 6	1100	1101	1110	) (1111
		1 110	0 (110	1 (11)	0 (111			0010	(0011	0100	(0101	0110	0111	1000	(100	1 101	0 (10	11 (	1100 )	(1101	1110	)(1111
th_fulladder_4bit/b 1111 // tb_fulladder_4bit/c 1 // tb_fulladder_4bit/s 1111	0100					010	1															(0100
/tb_fulladder_4bit/b 1111 //tb_fulladder_4bit/c 1	0100					010	1												0001			