컴퓨터공학실험2 6주차 결과보고서

전공: 컴퓨터공학과 학년: 2 학번: 20191559 이름: 강상원

1. 실험 목적

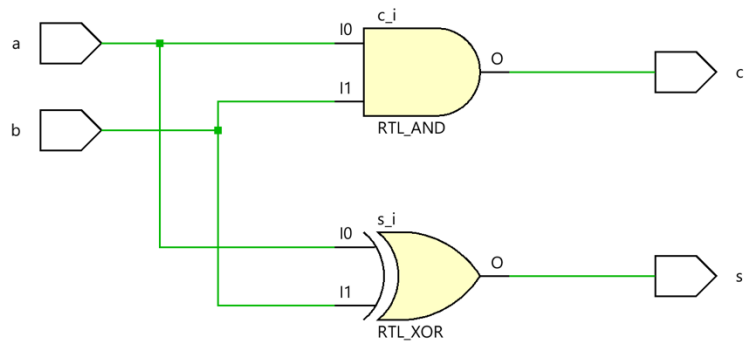
Full/Half Adder, Full/Half Subtractor, BCD Code Converter의 실제 수행을 확인해본다. Verilog 언어를 이용해 Adder(가산기), Subtractor(감산기), BCD Code Converter를 구현하는 방법을 익힌다. Schematic 구조, Simulation을 통해 gate의 구현을 확인한다. 경우의 수를 나누어 각 입력에 대한 gate의 출력값을 확인해 본다.

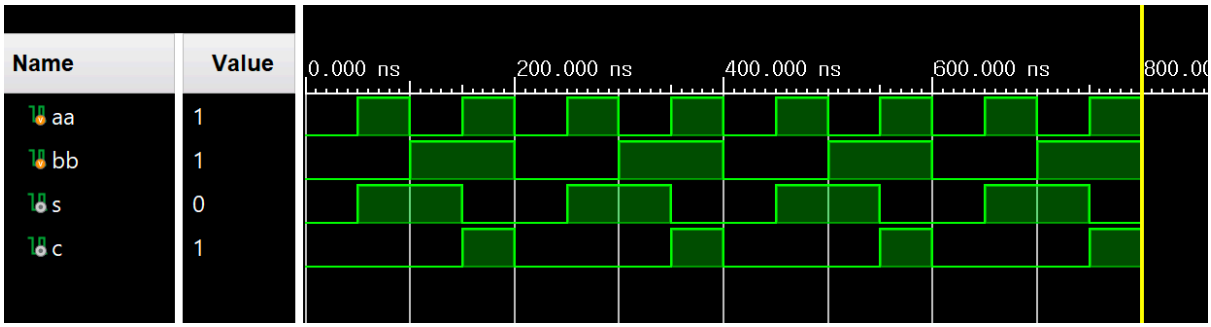
2. Full Adder 및 Half Adder 의 simulation 결과 및 과정에 대해서 설명하시오.

1) Half Adder

assign s = a ^ b, assign c = a & b와 같이 나타낼 수 있다. Schematic과 Simulation 결과는 아래와 같다.

```
1   `timescale 1ns / 1ps
2
3   module half_adder(
4       input a, b,
5       output s, c
6   );
7
8   assign s = a ^ b;
9   assign c = a & b;
10
11  endmodule
12
```





a, b 둘 중에 하나만 입력이 1이어야 s(sum)가 1이 되고, a, b 둘 다 1이어야 c(carry)가 1이 된다.

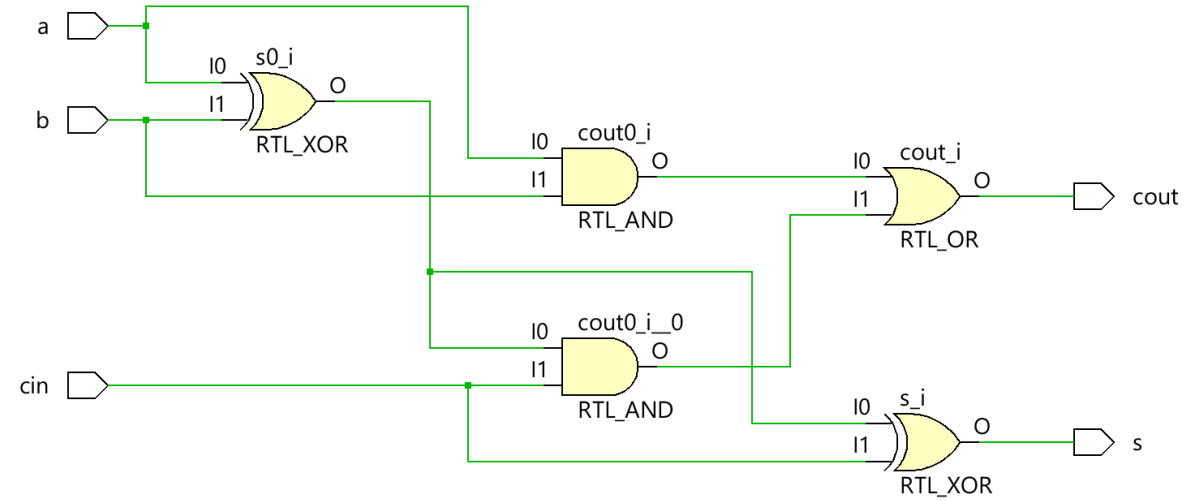
진리표는 다음과 같다.

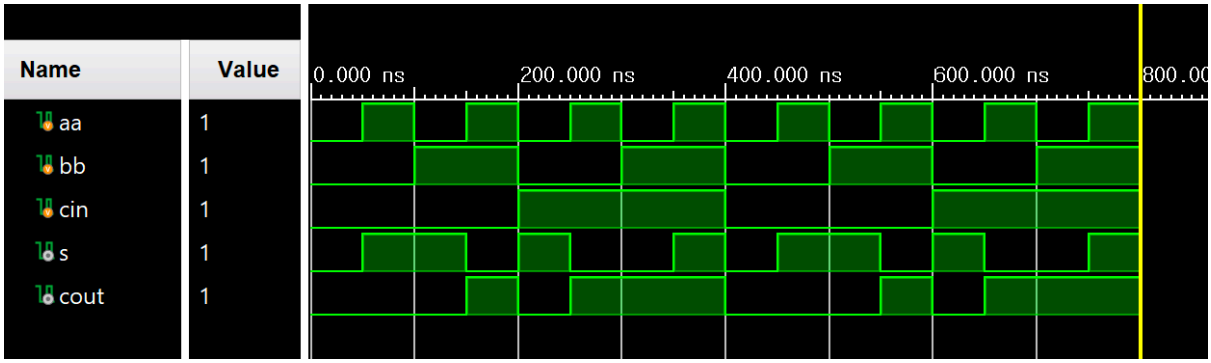
a	b	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

2) Full Adder

assign s = (a ^ b) ^ cin, assign cout = (a & b) | ((a ^ b) & cin)와 같이 나타낼 수 있다. Schematic과 Simulation 결과는 아래와 같다.

```
1  `timescale 1ns / 1ps
2
3  module full_adder(
4      input a, b, cin,
5      output s, cout
6  );
7
8  assign s = (a ^ b) ^ cin;
9  assign cout = (a & b) | ((a ^ b) & cin);
10
11 endmodule
12
```





Full Adder는 Half Adder와 다르게 이전 carry 값까지 고려하여 합을 계산한다. 이에 따라 s(sum)는 a, b, cin 중 1의 대수가 홀수여야 1이 되고, s(sum)은 a AND b에 (a XOR b) AND cin을 OR한 값과 같다.

진리표는 다음과 같다.

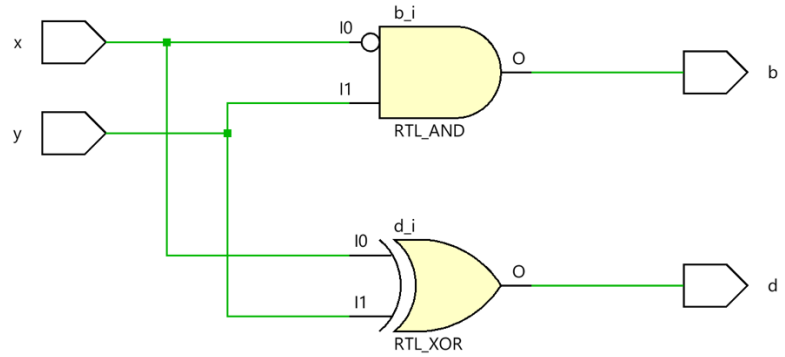
cin	a	b	s	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

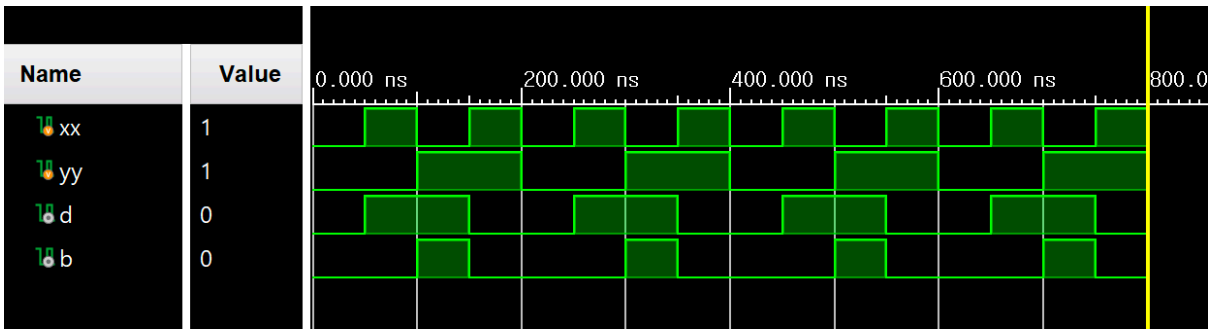
3. Full Subtractor 및 Half Subtractor 의 simulation 결과 및 과정에 대해서 설명하시오.

1) Half Subtractor

assign d = x ^ y, assign b = (~x) & y와 같이 나타낼 수 있다. Schematic과 Simulation 결과는 아래와 같다.

```
1   `timescale 1ns / 1ps
2
3   module half_subtractor(
4       input x, y,
5       output d, b
6   );
7
8   assign d = x ^ y;
9   assign b = (~x) & y;
10
11  endmodule
12
```





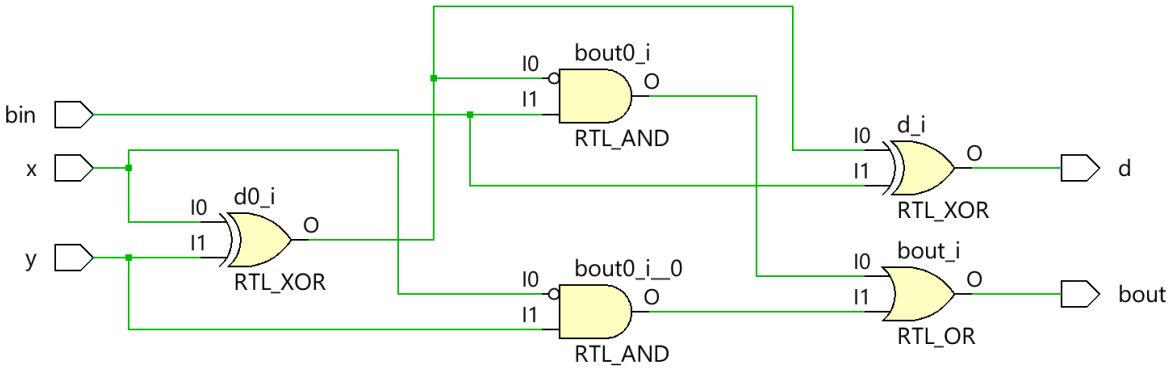
x, y 둘 중에 하나만 입력이 1이어야 d가 1이 되고, !x, y 둘 다 1이어야 b(borrow)가 1이 된다. 진리표는 다음과 같다.

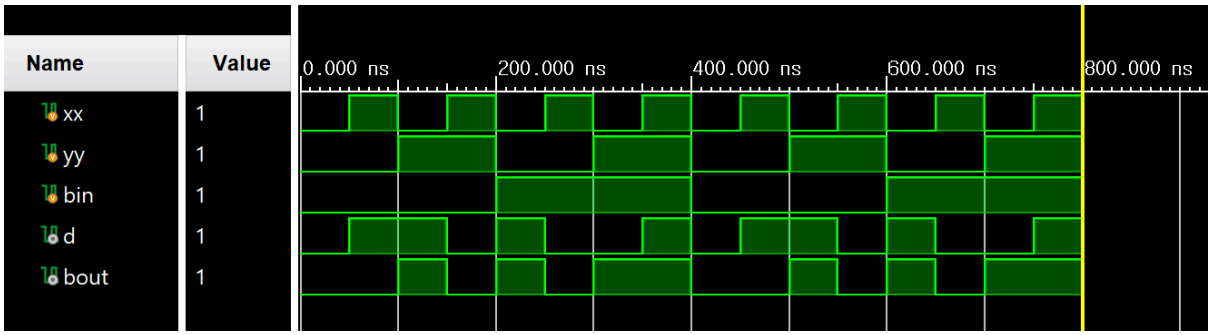
x	y	d	b
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

2) Full Subtractor

assign d = (x ^ y) ^ bin, assign bout = ((~(x ^ y)) & bin) | ((~x) & y)와 같이 나타낼 수 있다. Schematic과 Simulation 결과는 아래와 같다.

```
1   `timescale 1ns / 1ps
2
3   module full_subtractor(
4       input x, y, bin,
5       output d, bout
6   );
7
8   assign d = (x ^ y) ^ bin;
9   assign bout = ((~(x ^ y)) & bin) | ((~x) & y);
10
11  endmodule
12
```





Full Subtractor는 Half Subtractor와 다르게 이전 borrow 값까지 고려하여 합을 계산한다. 이에 따라 d는 x, y, bin 중 1의 대수가 홀수여야 1이 되고, bout(borrow-out)은 !(x XOR y) AND bin에 !x AND y를 OR한 값과 같다.

진리표는 다음과 같다.

bin	x	y	d	bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

4. 8421(BCD)-2421 Code converter simulation 결과 및 과정에 대해서 설명하시오. (진리표 작성 및 카로노맵 SOP form, POS form 포함)

8421-2421 Code Converter의 진리표는 다음과 같다. Don't care는 D로 표현하였다.

w	x	y	z	a	b	c	d
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	1	0	0
0	1	1	1	1	1	0	1
1	0	0	0	1	1	1	0
1	0	0	1	1	1	1	1
1	0	1	0	D	D	D	D
1	0	1	1	D	D	D	D
1	1	0	0	D	D	D	D
1	1	0	1	D	D	D	D
1	1	1	0	D	D	D	D
1	1	1	1	D	D	D	D

위의 진리표를 이용해 a, b, c, d 각각의 카르노 맵을 작성하였다.

yz\wx	00	01	11	10
00	0	0	D	1
01	0	1	D	1
11	0	1	D	D
10	0	1	D	D

↑ A의 카르노 맵

yz\wx	00	01	11	10
00	0	1	D	1
01	0	0	D	1
11	0	1	D	D
10	0	1	D	D

↑ B의 카르노 맵

yz\wx	00	01	11	10
00	0	0	D	1
01	0	1	D	1
11	1	0	D	D
10	1	0	D	D

↑ C의 카르노 맵

yz\wx	00	01	11	10
00	0	0	D	0
01	1	1	D	1
11	1	1	D	D
10	0	0	D	D

↑ D의 카르노 맵

위의 네 카르노 맵을 통해 a, b, c, d를 SOP, POS form으로 도출해낼 수 있다. 그 결과는 다음과 같다.

a = w + xz + xy = (w+x)(w+y+z)

b = w + xy + xy'z' = (w + x)(x + y + z')

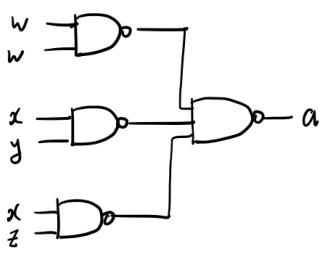
c = w + x'y+xy'z = (w + x + y)(w + y + z)(x' + y')

d = z

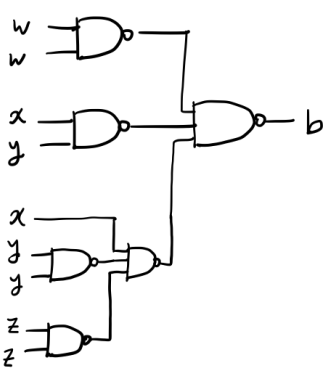
이와 같은 식을 NAND로, 또는 NOR로 표현할 수 있다. (회로 공간, 전기 절약)

<NAND>

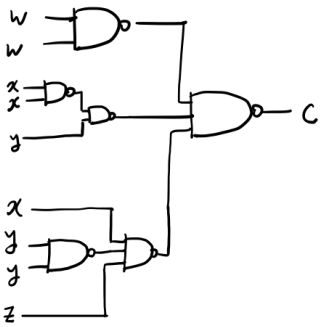
- a -



- b -



- c -

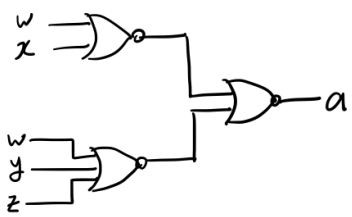


- d -

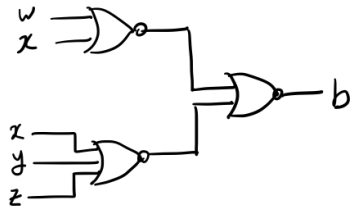
z - d

<NOR>

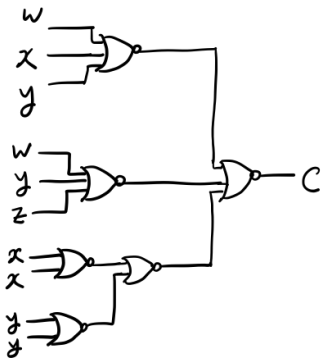
- a -



- b -



- c -

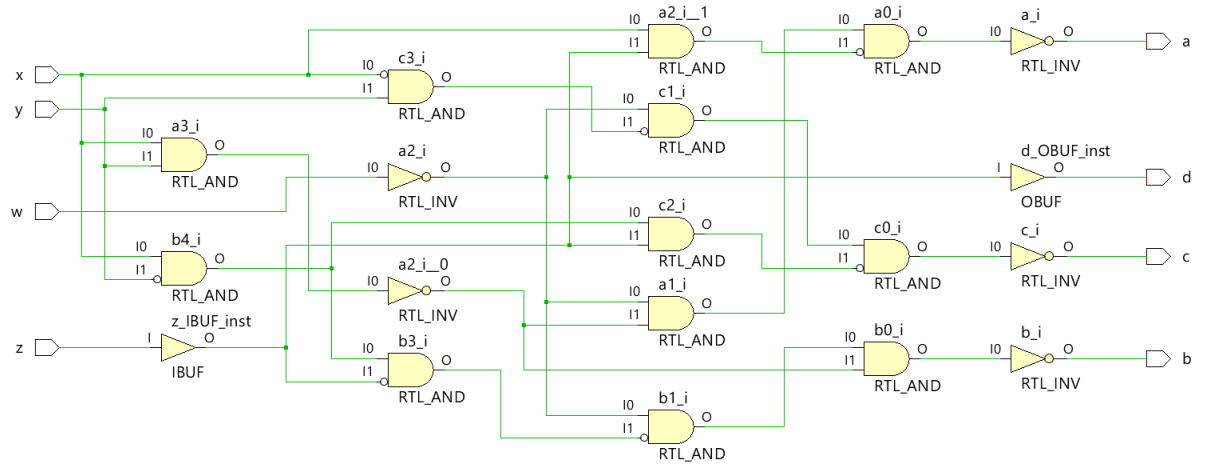


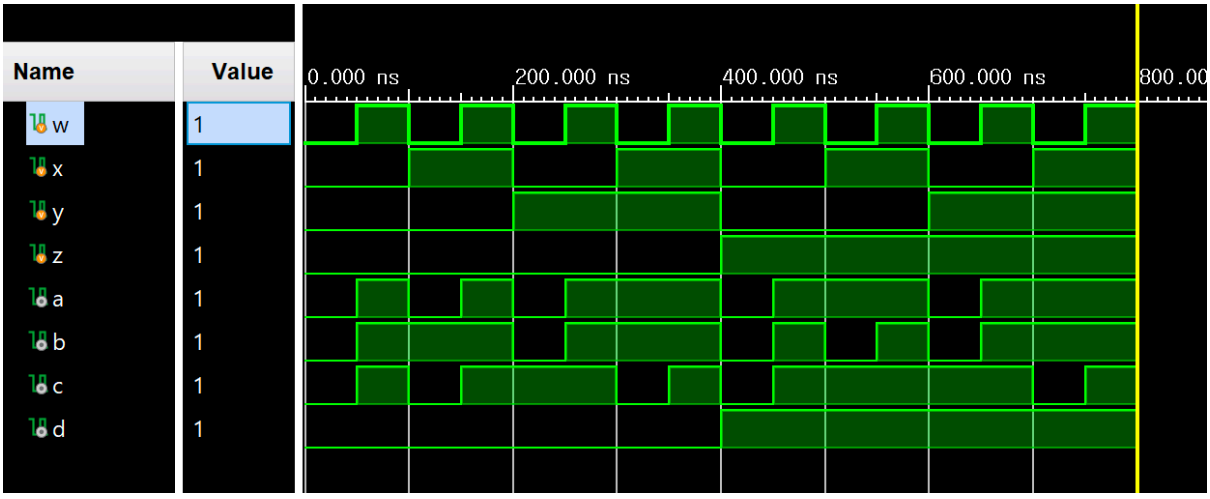
- d -

z - d

이 중 NAND를 이용한 방식을 이용하였다.

```
1  `timescale 1ns / 1ps
2
3  module bcd_code_converter(
4      input w, x, y, z,
5      output a, b, c, d
6  );
7
8  assign a = ~((~w) & (~(x & y)) & (~(x & z)));
9  assign b = ~((~w) & (~(x & (~y) & (~z))) & (~(x & y)));
10 assign c = ~((~w) & (~((~x) & y)) & (~(x & (~y) & z)));
11 assign d = z;
12
13 endmodule
14
```





위의 Simulation 결과와 진리표를 비교해보면 동일하다는 것을 알 수 있다.

5. 결과 검토 및 논의사항.

실험을 통해 Half/Full Adder와 Half/Full Subtractor의 작동을 확인할 수 있었다. Half Adder와 Half Subtractor는 단순히 두 비트를 더하거나 빼는 연산을 수행해 합/차와 carry/borrow를 반환하였지만, Full Adder와 Full Subtractor는 이전 비트의 carry/borrow까지 고려하여 합/차와 출력 carry/borrow를 반환함을 확인할 수 있었다. 또한 실습 전 미리 진리표를 채워보며 Adder와 Subtractor의 작동을 예상하였는데, 실험 결과 예측과 동일하게 나왔다.

8421-2421 BCD Code Converter는 우선 진리표를 그린 뒤에 이를 보고 a, b, c, d의 카르노 맵을 작성하였다. 작성한 카르노 맵을 통해 a, b, c, d 각각의 SOP / POS form을 나타낼 수 있었다.

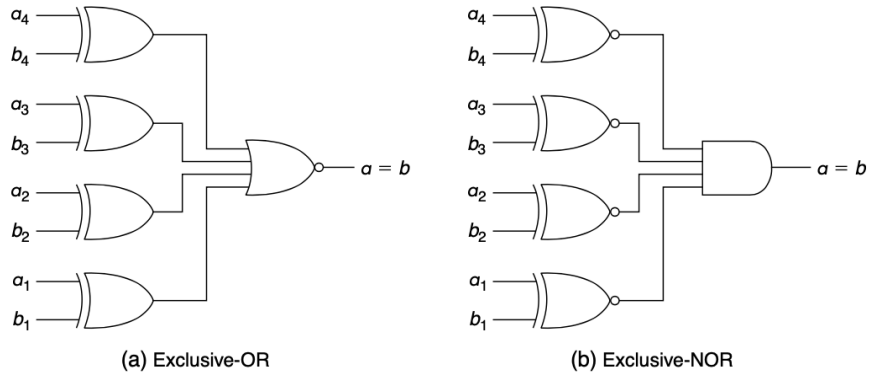
+ 이전에 학습한 NAND, NOR gate를 사용한 회로 최적화를 적용하여 보았다.

이 중 NAND gate를 이용한 회로를 Verilog로 코딩하여 확인해 보았다. Simulation 결과 미리 작성해두었던 진리표와 동일한 출력을 가짐을 볼 수 있었다.

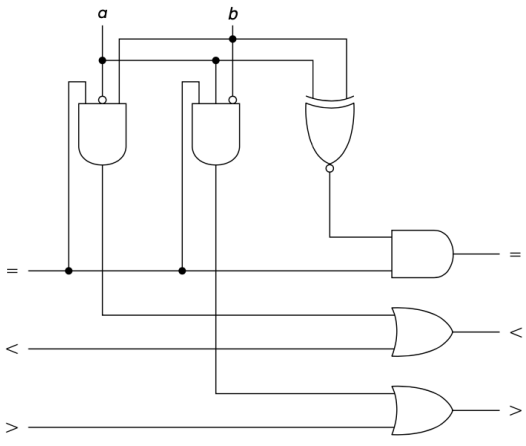
6. 추가 이론 조사 및 작성.

비교기는 두 수를 비교해서, 크기가 같은지 어느 한 쪽이 더 큰지 알려준다. XOR 연산의 경우, 입력이 다른 경우에 1, 같은 경우에 0을 반환한다. 다중 비트 수를 비교할 때는 어느 한 비트라도 일치하지 않으면 다른 수임을 나타내야 한다.

다음 그림은 4비트 비교기를 나타낸 것이다. (우측은 XNOR, AND gate 이용)



대표적인 1비트 비교기는 다음과 같은 형태이다.



두 수의 대소관계까지 알려주는 4비트 비교기를 구현하기 위해서는 다음과 같은 관계를 알아야 한다.

$A > B$ if $A_4 > B_4$ or $(A_4 = B_4$ and $A_3 > B_3)$ or $(A_4 = B_4$ and $A_3 = B_3$ and $A_2 > B_2)$ or $(A_4 = B_4$ and $A_3 = B_3$ and $A_2 = B_2$ and $A_1 > B_1)$

$A < B$ if $A_4 < B_4$ or $(A_4 = B_4$ and $A_3 < B_3)$ or $(A_4 = B_4$ and $A_3 = B_3$ and $A_2 < B_2)$ or $(A_4 = B_4$ and $A_3 = B_3$ and $A_2 = B_2$ and $A_1 < B_1)$

$A = B$ if $A_4 = B_4$ and $A_3 = B_3$ and $A_2 = B_2$ and $A_1 = B_1$