4주차 결과보고서

전공: 신문방송학과 학년: 4학년 학번: 20191150 이름: 전현길

1. 실험 목적

- NAND, NOR, XOR, AOI gate의 동작을 이해하고 확인한다.
- Verilog를 사용해 다중 입력 NAND/NOR/XOR/AOI gate를 구현한다.
- 입력 신호를 생성한 뒤 simulation으로 구현된 gate의 동작을 확인한다.
- FPGA를 통해서 Verilog로 구현된 회로의 동작을 확인한다.

2. 4-input NAND gate의 simulation 결과 및 과정에 대해서 설명하시오. (3 input-3 output, 진리표 작성)

source code

```
`timescale 1ns / 1ps

module NAND(
    input a, b, c, d,
    output e, f, g
    );

assign e = ~(a&b);
assign f = ~(e&c);
assign g = ~(f&d);

endmodule
```

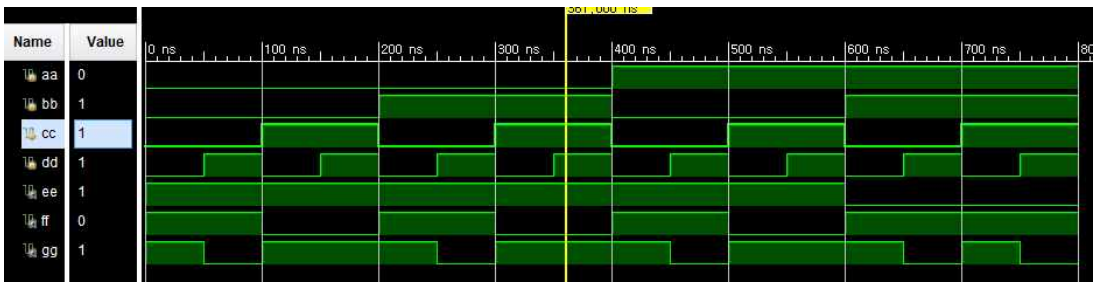
testbench code

```
`timescale 1ns / 1ps

module NAND_tb;

reg aa, bb, cc, dd;
wire ee, ff, gg;

NAND u_test(
.a (aa), .b (bb), .c (cc), .d (dd),
.e (ee), .f (ff), .g (gg)
);

initial begin
    aa = 1'b0;
    bb = 1'b0;
    cc = 1'b0;
    dd = 1'b0;
end

always@(aa or bb or cc or dd) begin
    aa <= #400 ~aa;
    bb <= #200 ~bb;
    cc <= #100 ~cc;
    dd <= #50 ~dd;
end

initial begin
    #800
    $finish;
end

endmodule
```

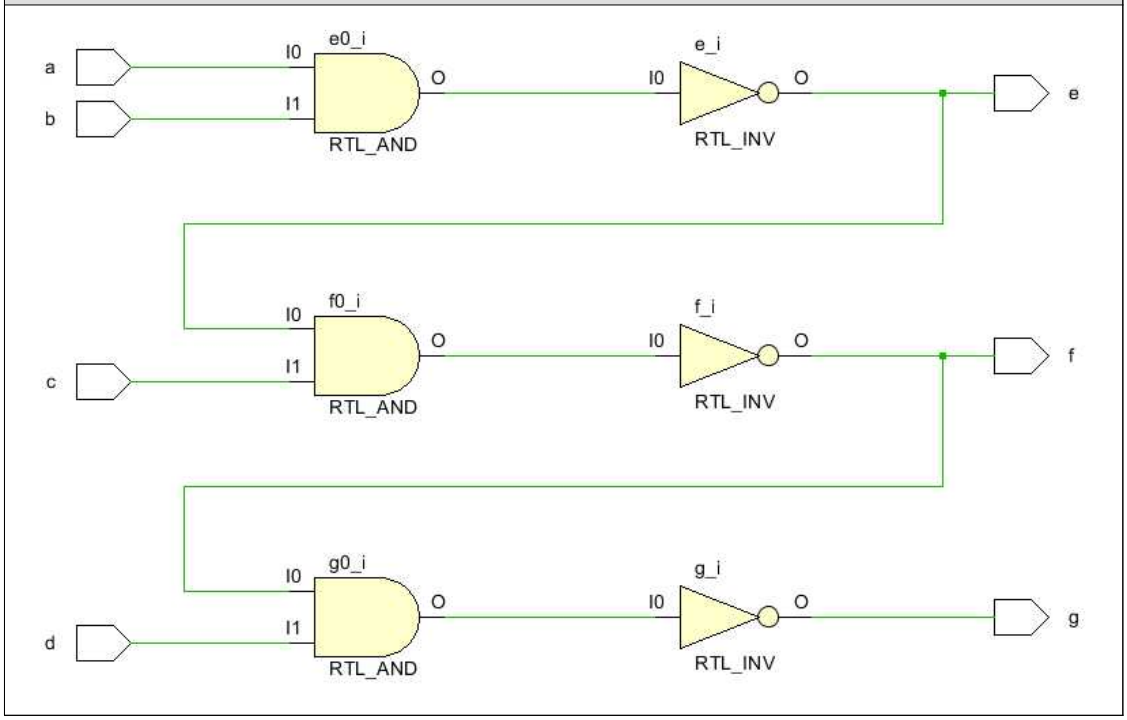
4-3 NAND truth table

input				output		
a	b	c	d	e	f	g
0	0	0	0	1	1	1
0	0	0	1	1	1	0
0	0	1	0	1	0	1
0	0	1	1	1	0	1
0	1	0	0	1	1	1
0	1	0	1	1	1	0
0	1	1	0	1	0	1
0	1	1	1	1	0	1
1	0	0	0	1	1	1
1	0	0	1	1	1	0
1	0	1	0	1	0	1
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	0	1	0
1	1	1	0	0	1	1
1	1	1	1	0	1	0

4-Input NAND gate simulation



4-Input NAND gate circuit design



시뮬레이션의 편의성을 위해, 진리표와 동일한 형식대로 값이 변화하도록 테스트벤치 코드를 작성했다. **e = ~(a&b)**로 할당했기 때문에 a와 b가 모두 1일 때만 0이 되며, 그 외에는 모두 1이 된다. **f = ~(e&c)**로 할당했기 때문에, e와 c가 모두 1일 때에만 0이 되며, 그 외에는 모두 1이 된다. g = **~(f&d)**로 f와 d가 모두 1일 때에만 0이 되며, 그 외에는 모두 1이 된다.

흥미로운 점은 3-input NAND gate의 output은, input 3개(a, b, c)가 모두 1일 때 0이 되어야 하는데, 우리가 만든 회로에서는 a, b, c가 모두 1일 때 f가 1이 된다는 점이다. 하지만 이는 꼼꼼하게 따지면 당연한 일이다.

a, b = 1 → e = 0 → e&c = 0 → ~(e&c) = 1

따라서 3-input NAND gate의 output과 output f의 결과값은 동일하지 않다. 이는 output g도 마찬가지이다. 원래 4-input NAND gate의 output은 input 4개가 모두 1일 때에만 0이 되고 나머지 경우엔 1이 되어야 한다. 하지만 output g는 그 외에도 0이 되는 모습을 볼 수 있다.

3. 4-input NOR gate의 simulation 결과 및 과정에 대해서 설명하시오. (3 input-3 output, 진리표 작성)

source code

```
`timescale 1ns / 1ps

module NOR(
    input a, b, c, d,
    output e, f, g
    );

assign e = ~(a|b);
assign f = ~(e|c);
assign g = ~(f|d);

endmodule
```

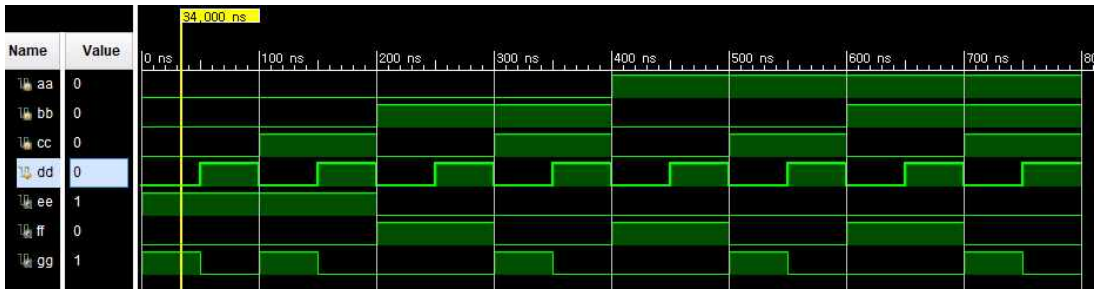
testbench code

```
`timescale 1ns / 1ps

module NOR_tb;

reg aa, bb, cc, dd;
wire ee, ff, gg;

NOR u_test(
.a (aa), .b (bb), .c (cc), .d (dd),
.e (ee), .f (ff), .g (gg)
);

initial begin
    aa = 1'b0;
    bb = 1'b0;
    cc = 1'b0;
    dd = 1'b0;
end

always@(aa or bb or cc or dd) begin
    aa <= #400 ~aa;
    bb <= #200 ~bb;
    cc <= #100 ~cc;
    dd <= #50 ~dd;
end

initial begin
    #800
    $finish;
end

endmodule
```

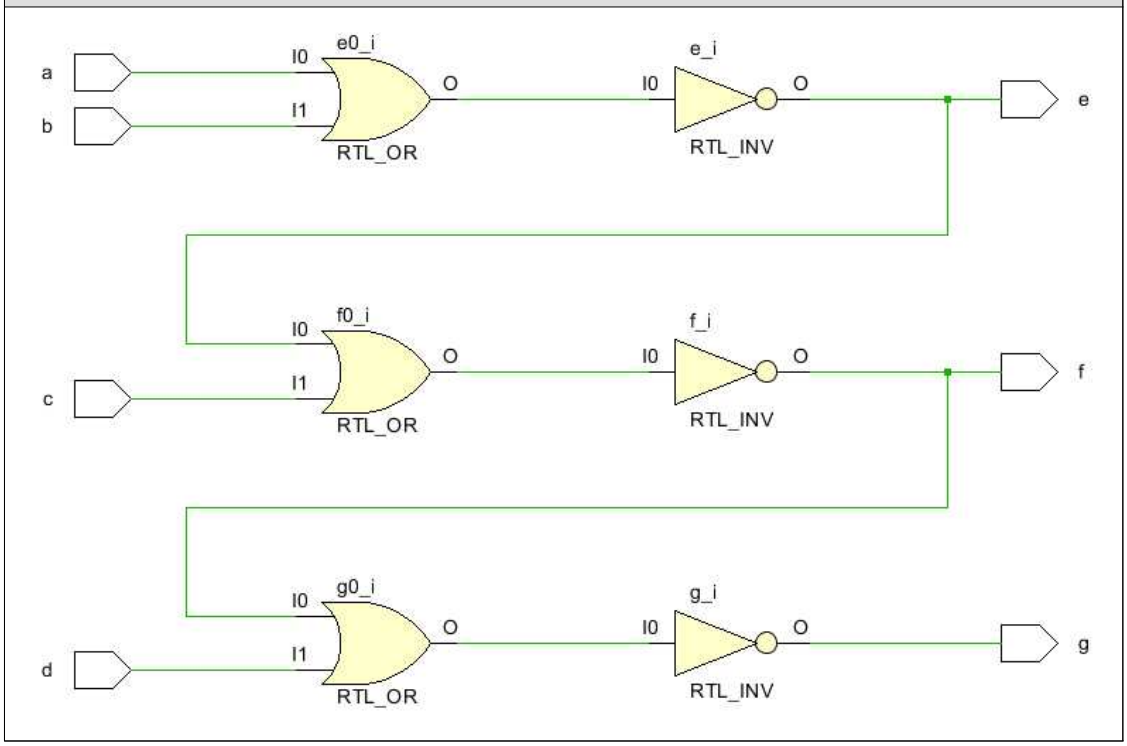
4-3 NOR truth table

input				output		
a	b	c	d	e	f	g
0	0	0	0	1	0	1
0	0	0	1	1	0	0
0	0	1	0	1	0	1
0	0	1	1	1	0	0
0	1	0	0	0	1	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	0	0	1
1	0	1	1	0	0	0
1	1	0	0	0	1	0
1	1	0	1	0	1	0
1	1	1	0	0	0	1
1	1	1	1	0	0	0

4-Input NOR gate simulation



4-Input NOR gate circuit design



시뮬레이션의 편의성을 위해, 진리표와 동일한 형식대로 값이 변화하도록 테스트벤치 코드를 작성했다. **e = ~(a|b)**로 할당했기 때문에 a와 b가 모두 0일 때만 1이 되며, 그 외에는 모두 0이 된다. **f = ~(e|c)**이므로 e와 c가 모두 0일 때에만 1이 되며, 그 외에는 모두 0이 된다. g = **~(f|d)**이므로 f와 d가 모두 0일 때에만 1이 되며, 그 외에는 모두 0이 된다.

NOR 게이트에서 역시 앞서 살펴본 문제가 동일하게 나타난다. 3-input, 4-input NOR gate와 우리가 만든 회로가 정확히 동일하게 동작하지 않는다.

4. 4-input XOR gate의 simulation 결과 및 과정에 대해서 설명하시오. (3 input-3 output, 진리표 작성)

source code

```
`timescale 1ns / 1ps

module XOR(
    input a, b, c, d,
    output e, f, g
    );

assign e = a^b;
assign f = e^c;
assign g = f^d;

endmodule
```

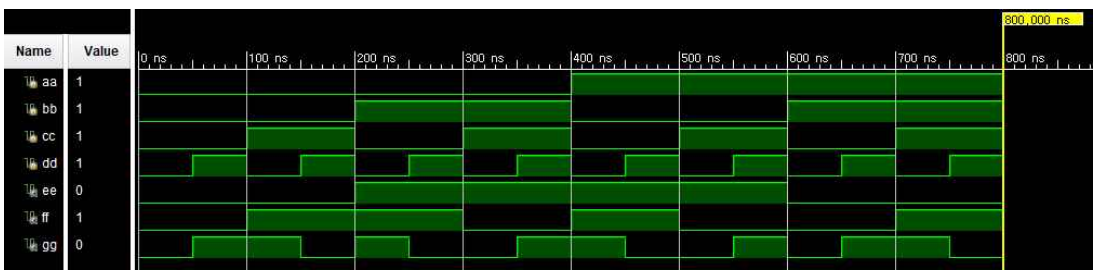
testbench code

```
`timescale 1ns / 1ps

module XOR_tb;

reg aa, bb, cc, dd;
wire ee, ff, gg;

XOR u_test(
.a (aa), .b (bb), .c (cc), .d (dd),
.e (ee), .f (ff), .g (gg)
);

initial begin
    aa = 1'b0;
    bb = 1'b0;
    cc = 1'b0;
    dd = 1'b0;
end

always@(aa or bb or cc or dd) begin
    aa <= #400 ~aa;
    bb <= #200 ~bb;
    cc <= #100 ~cc;
    dd <= #50 ~dd;
end

initial begin
    #800
    $finish;
end

endmodule
```

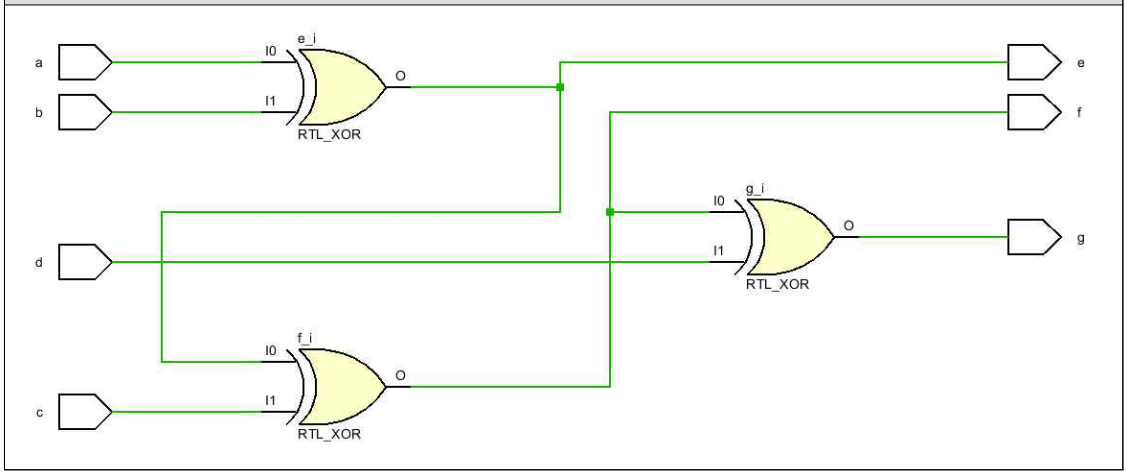
4-3 XOR truth table

input				output		
a	b	c	d	e	f	g
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	0	1	1	0	1	0
0	1	0	0	1	1	1
0	1	0	1	1	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	1
1	0	0	0	1	1	1
1	0	0	1	1	1	0
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	0	0
1	1	0	1	0	0	1
1	1	1	0	0	1	1
1	1	1	1	0	1	0

4-Input XOR gate simulation



4-Input XOR gate circuit design



시뮬레이션의 편의성을 위해, 진리표와 동일한 형식대로 값이 변화하도록 테스트벤치 코드를 작성했다. **e = a^b**로 할당했기 때문에 a와 b가 다를 때만 1이 되며, 같을 때는 0이 된다. **f = e^c**이므로 e와 c가 다를 때엔 1이 되며, 같을 때는 0이 된다. g = **f^d**이므로 f와 d가 다를 때만 1이 되며, 같을 때는 0이 된다.

XOR 게이트에서는 앞서 확인한 NAND, NOR 게이트와 달리, **f = a^b^c와 f = (a^b)^c의 결과가 동일**하고, **g = a^b^c^d의 결과와 g = ((a^b)^c)^d의 결과가 동일**하다. 다중 입력 XOR 게이트는 1을 홀수 개 입력하면 1을 출력하고, 짝수 개 입력하면 0을 출력하는 방식으로 작동되는데, AND, OR 연산처럼 **XOR 연산 간에 결합법칙이 성립하므로 NAND, NOR 연산처럼 결과값이 달라지지 않는다.**

단 XNOR(부정 배타적 논리합) 연산 등은 NAND, NOR 연산과 동일하게 결과값이 달라진다.

5. 4-input AOI gate의 simulation 결과 및 과정에 대해서 설명하시오. (3 input-3 output, 진리표 작성)

source code

```
`timescale 1ns / 1ps

module AOI(
    input a, b, c, d,
    output e, f, g
    );

assign e = a&b;
assign f = c&d;
assign g = ~(e|f);

endmodule
```

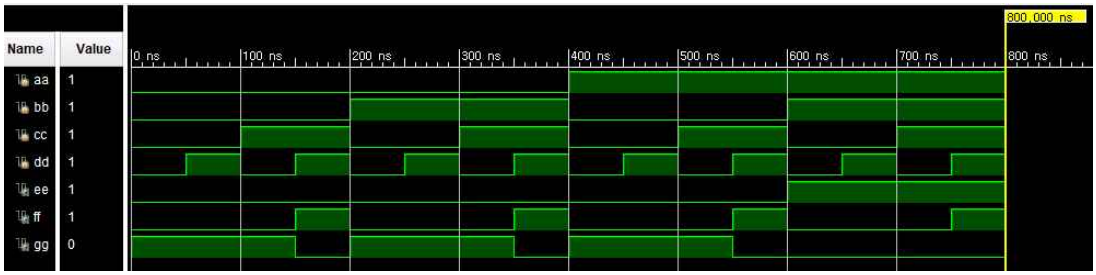
testbench code

```
`timescale 1ns / 1ps

module AOI_tb;

reg aa, bb, cc, dd;
wire ee, ff, gg;

AOI u_test(
.a (aa), .b (bb), .c (cc), .d (dd),
.e (ee), .f (ff), .g (gg)
);

initial begin
    aa = 1'b0;
    bb = 1'b0;
    cc = 1'b0;
    dd = 1'b0;
end

always@(aa or bb or cc or dd) begin
    aa <= #400 ~aa;
    bb <= #200 ~bb;
    cc <= #100 ~cc;
    dd <= #50 ~dd;
end

initial begin
    #800
    $finish;
end

endmodule
```

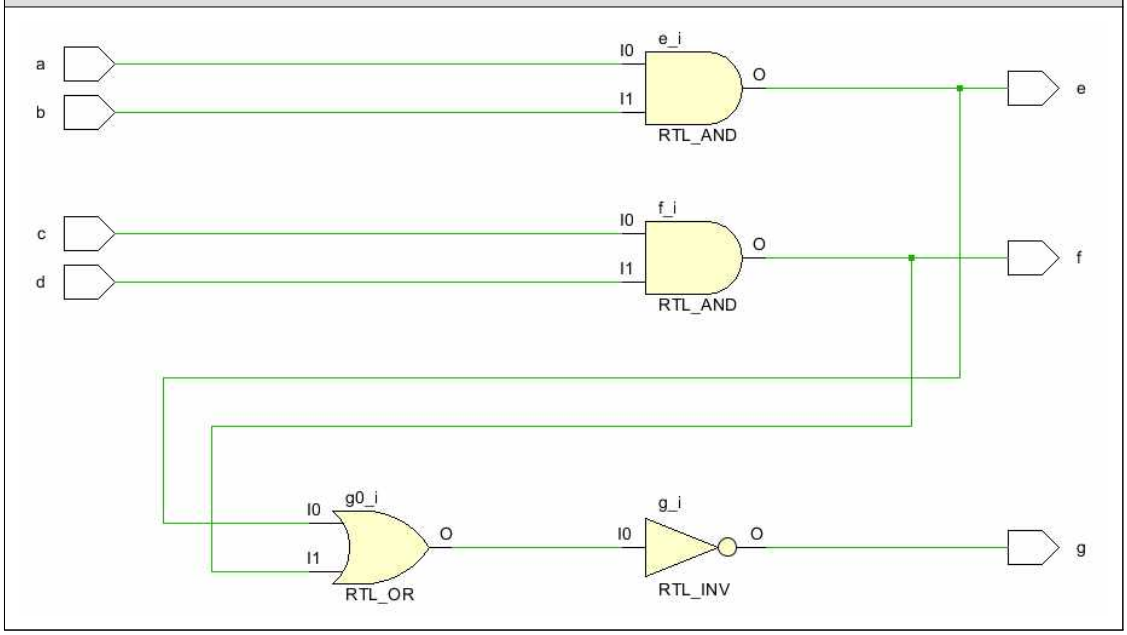
4-3 AOI truth table

input				output		
a	b	c	d	e	f	g
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	0	0	1
0	1	1	0	0	0	1
0	1	1	1	0	1	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	0	1
1	0	1	1	0	1	0
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	1	1	0

4-Input AOI gate simulation



4-Input AOI gate circuit design



시뮬레이션의 편의성을 위해, 진리표와 동일한 형식대로 값이 변화하도록 테스트벤치 코드를 작성했다. **e = a&b, f = c&d, g = ~(e|f)**로 각각 할당하였으므로, e와 f는 일반적인 AND 연산처럼 동작한다. g는 e, f가 모두 0일 때에만 1이며, 그 외에는 모두 1이 된다.

6. 결과 검토 및 논의 사항

이번 실험에서는 **4-input NAND/NOR/XOR/AOI gate를 4-input AND/OR gate를 구현했던 방식으로 구현**해 보았고, **FPGA를 활용해서 직접 결과값 출력을 실험**해 보았다.

실험 결과, 구현한 NAND gate, NOR gate의 output들은 4-input NAND, NOR gate처럼 동작하지 않았다. 이는 **NAND, NOR, XNOR 게이트와 같은 부정 연산들은 AND, OR, XOR 연산들처럼 결합법칙이 성립하지 않기 때문**이다. 따라서 다중 입력 NAND, NOR, XNOR 게이트에서 원하는 결과값을 얻기 위해서는 AND, OR, XOR 연산을 결합한 뒤 마지막에 부정 연산을 덧붙여야 한다.

7. 추가 이론 조사 및 작성

1) XOR, XNOR gate의 transitor-level 구현

XOR gate	XNOR gate