4주차 결과보고서

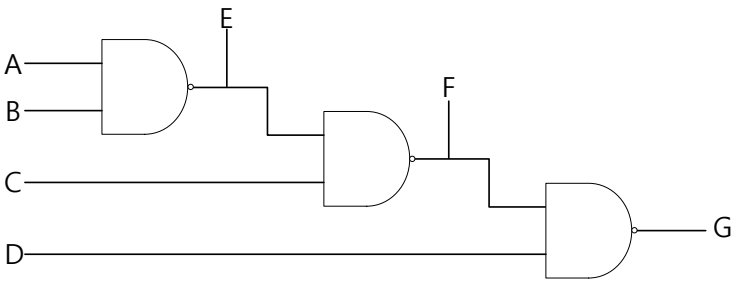
20191621 이민영

1, 실험 목적

NAND/NOR/XOR 게이트가 어떻게 동작되는지 확인한다. Verilog를 사용해서 3-input, 4-input 등 다중입력 NAND/NOR/XOR Gate를 구현한다. 입력 신호 생성 후에 simulation을 통해서 각각의 동작들을 확인해보는 시간을 가진다.

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

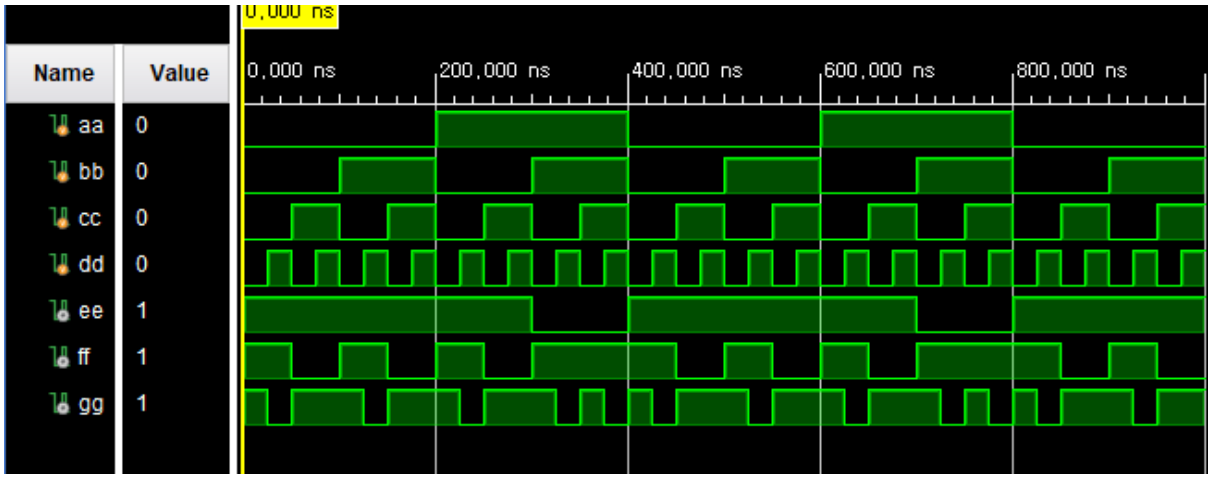
[4장 ppt 6,7 page 참조, 진리표 작성]

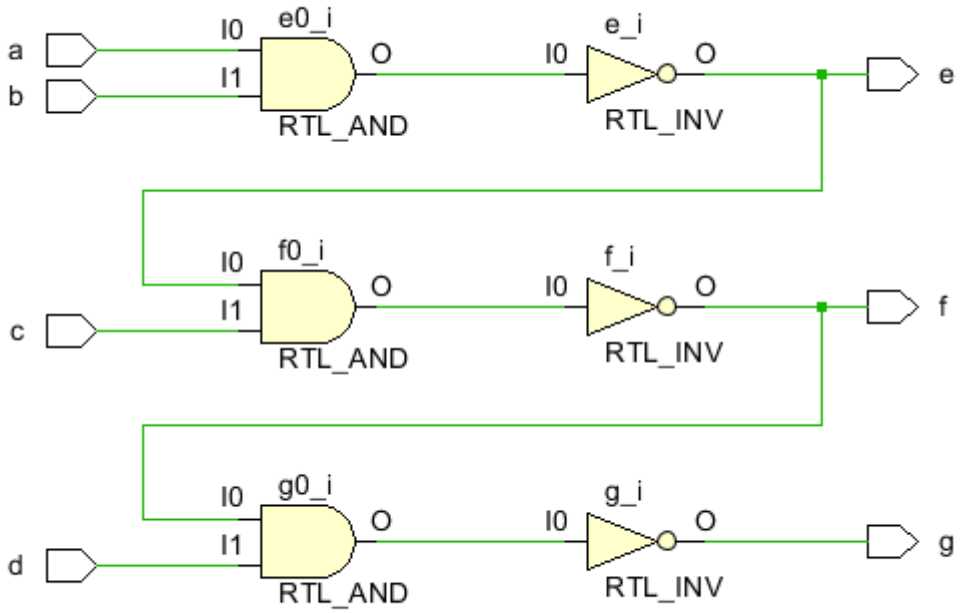


Four_input_NAND_gate_b	Four_input_NAND_gate_b_tb

`timescale 1ns / 1ps
 	
module four_input_NAND_gate_b(
input a,b,c,d,
output e,f,g
);
assign e = ~(a & b);
assign f = ~(e & c);
assign g = ~(f & d);
endmodule
	`timescale 1ns / 1ps
	
	module four_input_NAND_gate_b_tb;
	reg aa;
	reg bb;
	reg cc;
	reg dd;
	wire ee;
	wire ff;
	wire gg;
	
	four_input_NAND_gate_b
	u_four_input_NAND_gate_b(
	.a (aa),
	.b (bb),
	.c (cc),
	.d (dd),

```
.e (ee),
.f (ff),
.g (gg)
);
initial aa = 1'b0;
initial bb = 1'b0;
initial cc = 1'b0;
initial dd = 1'b0;

always aa = #200 ~aa;
always bb = #100 ~bb;
always cc = #50 ~cc;
always dd = #25 ~dd;

initial begin
      #1000
      $finish;
end


endmodule
```



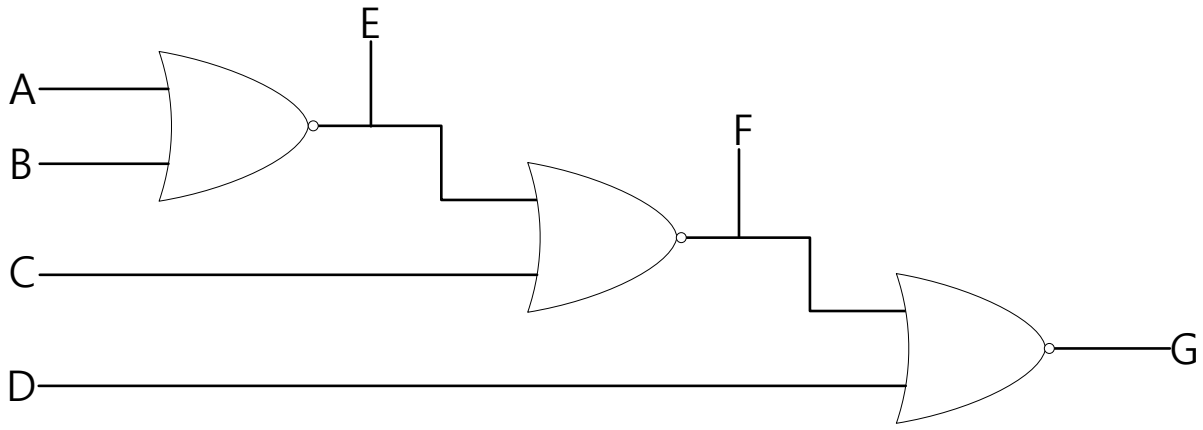


NAND는 AND연산자의 부정결과이다. e에는 a와 b에서의 NAND 값이, f는 e와 c 에서의 NAND값이, g는 f와 d의 NAND값이 들어간다. 이 값을 simulation을 통해서도 판단할 수 있다.

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

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

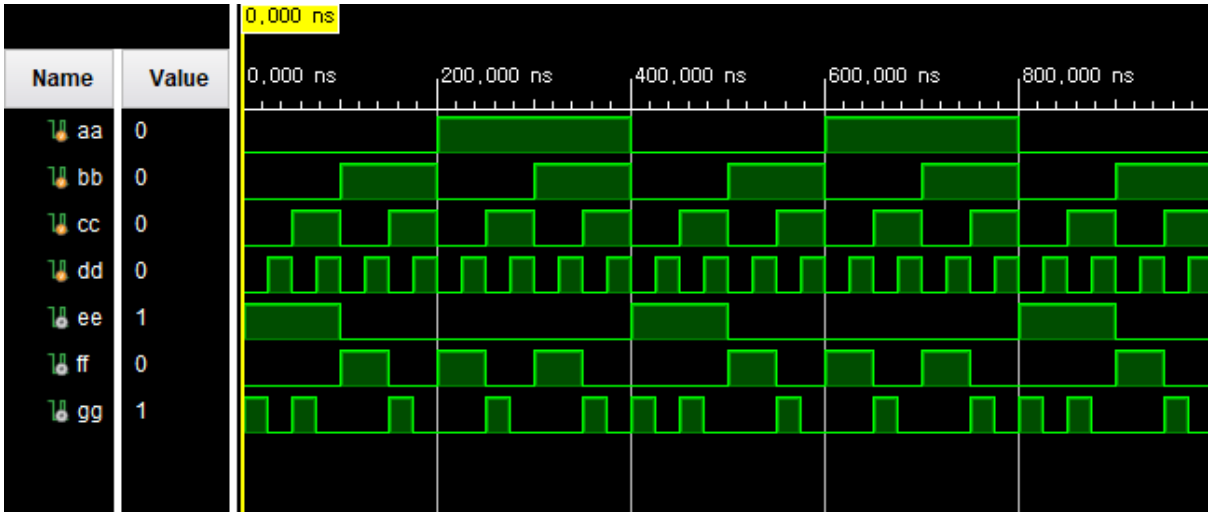
[4장 ppt 8,9 page 참조, 진리표 작성]

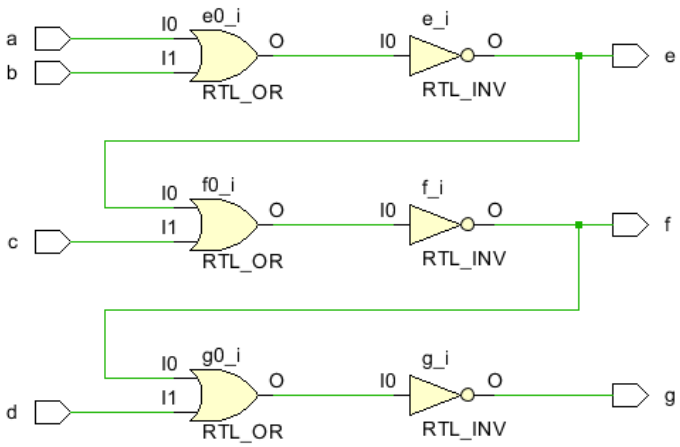


Four_input_NOR_gate_b	Four_input_NOR_gate_b_tb

`timescale 1ns / 1ps
 	
module four_input_NOR_gate_b(
input a,b,c,d,
output e,f,g
);
assign e = ~(a | b);
assign f = ~(e | c);
assign g = ~(f | d);
endmodule
	`timescale 1ns / 1ps
	
	module four_input_NAND_gate_b_tb;
	reg aa;
	reg bb;
	reg cc;
	reg dd;
	wire ee;
	wire ff;
	wire gg;
	
	four_input_NOR_gate_b
	u_four_input_NOR_gate_b(
	.a (aa),
	.b (bb),
	.c (cc),
	.d (dd),
	.e (ee),
	.f (ff),
	.g (gg)
	);
	initial aa = 1'b0;
	initial bb = 1'b0;
	initial cc = 1'b0;
	initial dd = 1'b0;

	always aa = #200 ~aa;
	always bb = #100 ~bb;
	always cc = #50 ~cc;
	always dd = #25 ~dd;
	
	initial begin
	#1000
	$finish;
	end
	
	endmodule



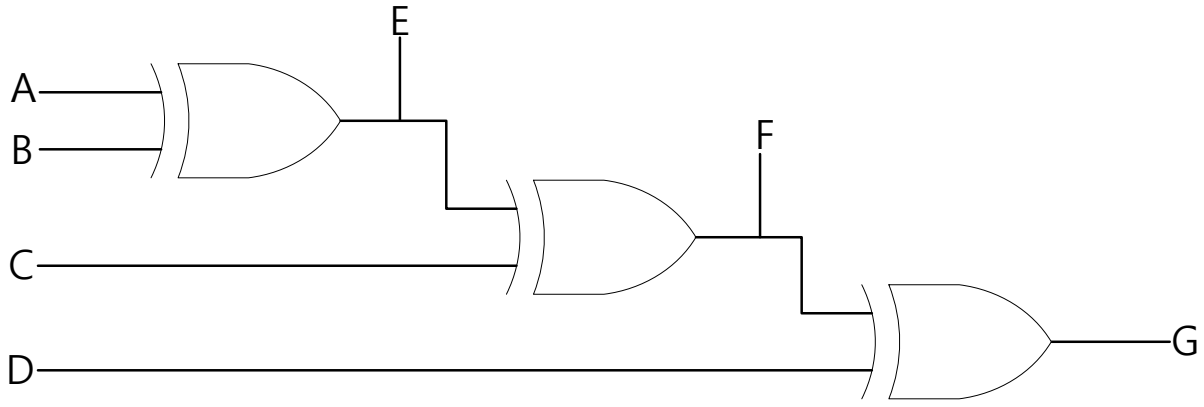


NOR은 or의 값에 NOT을 취한 것이다. 즉 OR의 결과값이 1일 경우에 NOR는 0이 되고, OR의 결과값이 0일 경우에 NOR는 1이 된다. 이러한 결과를 simulation을 통해서도 알 수 있다.

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

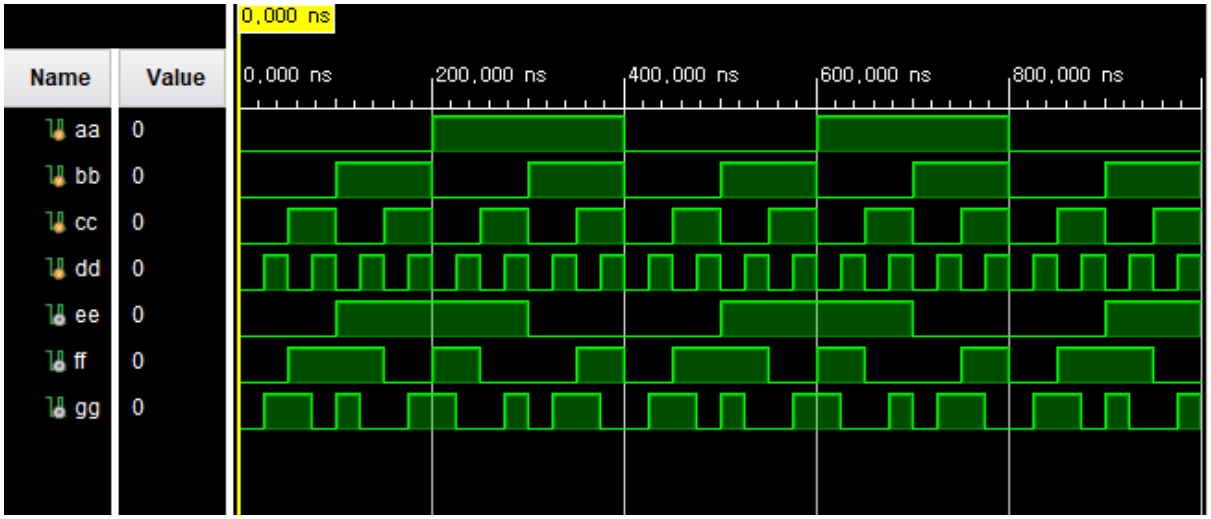
[3장 ppt 10,11 page 참조, 진리표 작성]

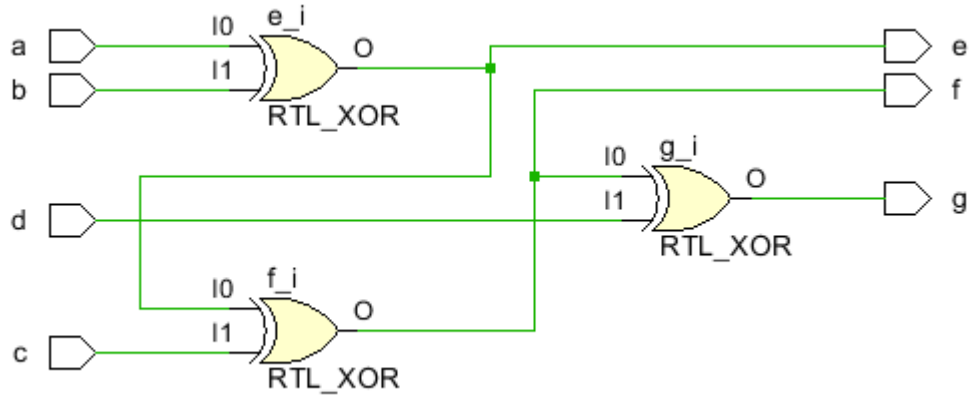


Four_input_XOR_gate_b	Four_input_XOR_gate_b_tb

```
`timescale 1ns / 1ps

module four_input_XOR_gate_b(
input a,b,c,d,
output e,f,g
      );
    assign e = a ^ b;
    assign f = e ^ c;
    assign g = f ^ d;
endmodule
```

```
`timescale 1ns / 1ps

module four_input_XOR_gate_b_tb;
reg aa;
reg bb;
reg cc;
reg dd;
wire ee;
wire ff;
wire gg;

four_input_XOR_gate_b
u_four_input_XOR_gate_b(
.a (aa),
.b (bb),
.c (cc),
.d (dd),
.e (ee),
.f (ff),
.g (gg)
);
initial aa = 1'b0;
initial bb = 1'b0;
initial cc = 1'b0;
initial dd = 1'b0;

always aa = #200 ~aa;
always bb = #100 ~bb;
always cc = #50 ~cc;
always dd = #25 ~dd;

initial begin
      #1000
      $finish;
end


endmodule
```





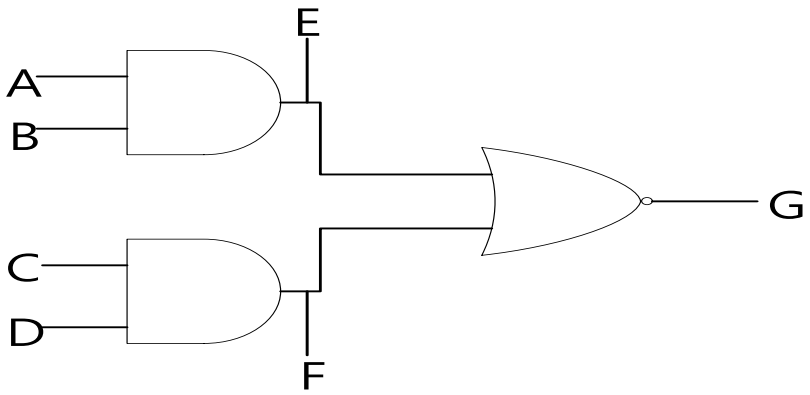
XOR gate는 입력값이 동일한 경우에 출력이 항상 0으로 되고 다를 때 1로 출력된다. 이는 simulation의 결과값을 통해서도 알 수 있다.

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

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

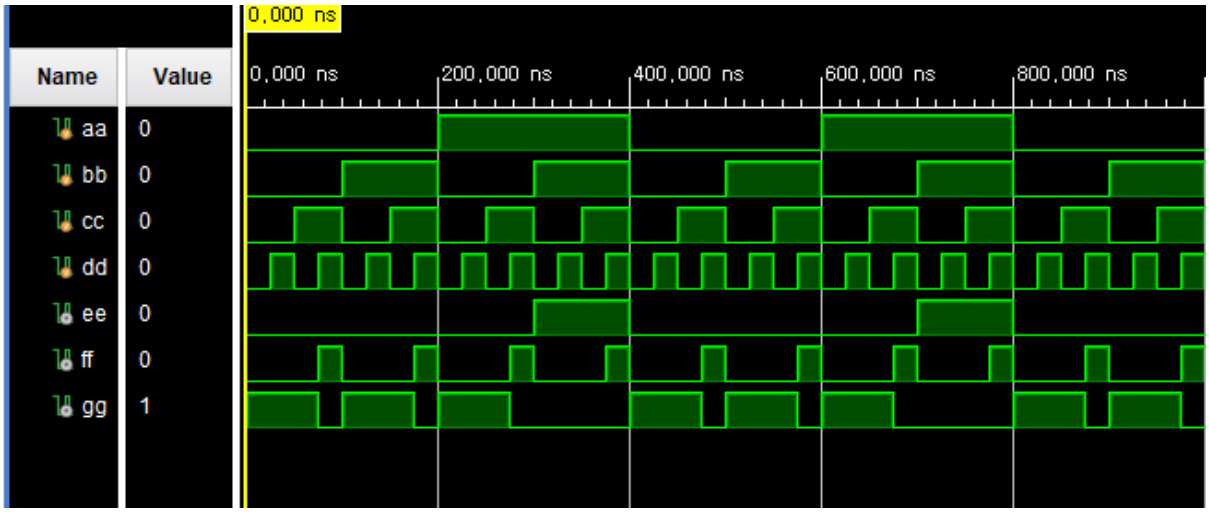
[4장 ppt 12,13 page 참조, 진리표 작성]

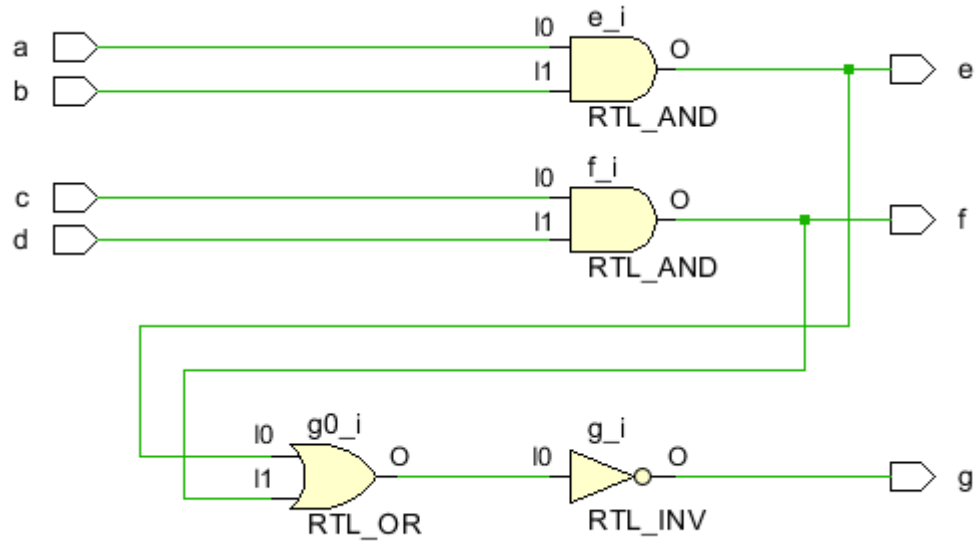


Four_input_AOI_gate_b	Four_input_AOI_gate_b_tb

`timescale 1ns / 1ps
 	
module four_input_AOI_gate_b(
input a,b,c,d,
output e,f,g
);
assign e = a & b;
assign f = c & d;
assign g = ~(e | f);
endmodule
	`timescale 1ns / 1ps
	
	module four_input_AOI_gate_b_tb;
	reg aa;
	reg bb;
	reg cc;
	reg dd;
	wire ee;
	wire ff;
	wire gg;
	
	four_input_AOI_gate_b
	u_four_input_AOI_gate_b(
	.a (aa),
	.b (bb),

```
.c (cc),
.d (dd),
.e (ee),
.f (ff),
.g (gg)
);
initial aa = 1'b0;
initial bb = 1'b0;
initial cc = 1'b0;
initial dd = 1'b0;

always aa = #200 ~aa;
always bb = #100 ~bb;
always cc = #50 ~cc;
always dd = #25 ~dd;

initial begin
      #1000
      $finish;
end


endmodule
```





AOI는 and – or – invert 이다. 즉 and, or를 취한 후 반대를 해주는 경우를 말한다. 이는 simulation의 결과값을 통해서도 알 수 있다.

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

6. 결과 검토 및 논의사항

Simulation 에서 각각의 Output들을 확인할 수 있었다. Input의 값들을 다르게 하기 위해서 시간을 조절하여 필요한 경우의 수들을 만족할 수 있도록 하였다. 실제로 Input 상황에 맞추어서 원하는 output의 결과가 맞는지를 진리표와 simulation을 비교해볼 수 있었다.

또한 여러 input을 주어도 결과적으로 같은 값이 나온다는 것을 알 수 있었다. 이전시간에 조사한 다른 여러 논리 게이트를 조합해서 만든 경우도 실습을 통해 유의미한 결과를 도출해낼 수 있을 것이라는 생각이 들었다.

7. 추가 이론 조사 및 작성

<OAI gate>

OAI gate는 AOI gate를 보완해준다. OAI는 or-and-invert로 or의 연산을 먼저 수행하고 이후에 and의 연산을 수행한다. 이후에 반전을 한다.

<AOI, OAI gate>

AOI 와 OAI gate는 CMOS 회로에서 구현한다. AOI 게이트는 AND, OR, NOT gate가 별도의 다른 함수들로 이루어진 경우보다 트렌지스터의 총 개수가 적게 쓰인다. 이는 속도의 증가, 면적의 감소, 전력 감소, 제조 비용의 감소 등으로 큰 이점이 있다.

<EQV>

EQV는 동치로, 두 명제가 모두 참이거나 모두 거짓일 때 참이 된다. XOR의 부정으로 판단할 수 있어서 부정논리합인 XNOR으로 할 수 있다. 대신 XOR와 달리 결합법칙이 성립하지 않는다는 특징이 있다.

Input a	Input b	Output c
0	0	1
0	1	0
1	0	0
1	1	1

<논리연산의 역사>

논리연산은 19세기 중반 영국의 수학자인 George Boole에 의해서 고안되었다. 불 대수가 고안된 이후에 논리학은 기호논리학의 성향이 강해지기 시작했다. 프로그래밍에서 조건에 의한 분기나

반복을 사용하기 위해서 주로 이용된다. 또한 디지털 회로의 신호의 경우에 0과 1로만 이루어져 있기 때문에 논리연산 (불 연산)이 효율적으로 사용되었다.