컴퓨터공학실험 2 4 주차 결과보고서

전공: 컴퓨터공학과 학년: 2 학번: 20191559 이름: 강상원

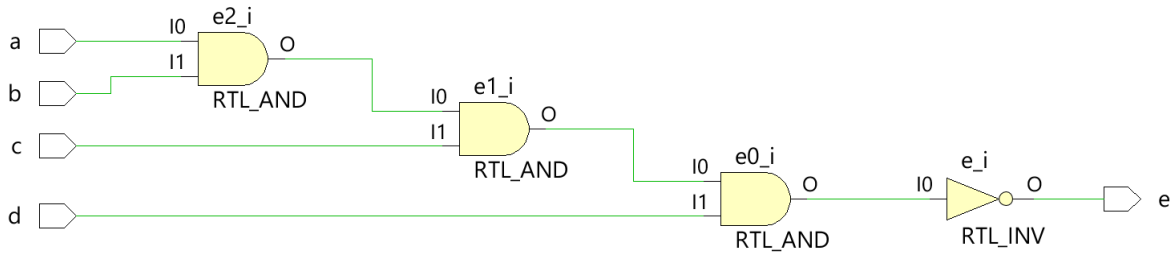
1. 실험 목적

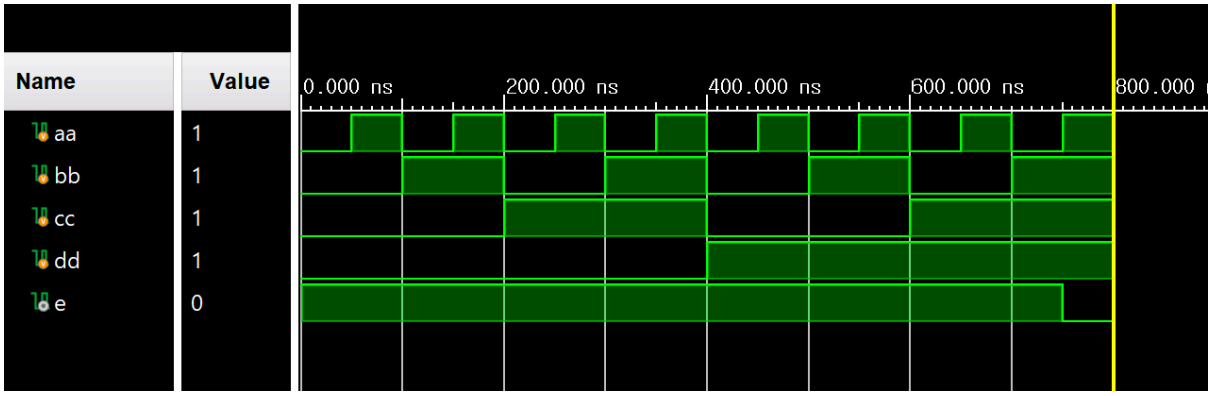
NAND, NOR, XOR gate 의 실제 수행을 확인해본다. Verilog 언어를 이용해 다중 NAND, NOR, XOR gate 를 구현하는 방법을 익힌다. Schematic 구조, Simulation 을 통해 gate 의 구현을 확인한다. 경우의 수를 나누어 각 입력에 대한 gate 의 출력값을 확인해 본다.

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

(A) 는 assign e = ~(a & b & c & d)와 같이 나타낼 수 있다. Schematic 과 Simulation 결과는 아래와 같다.

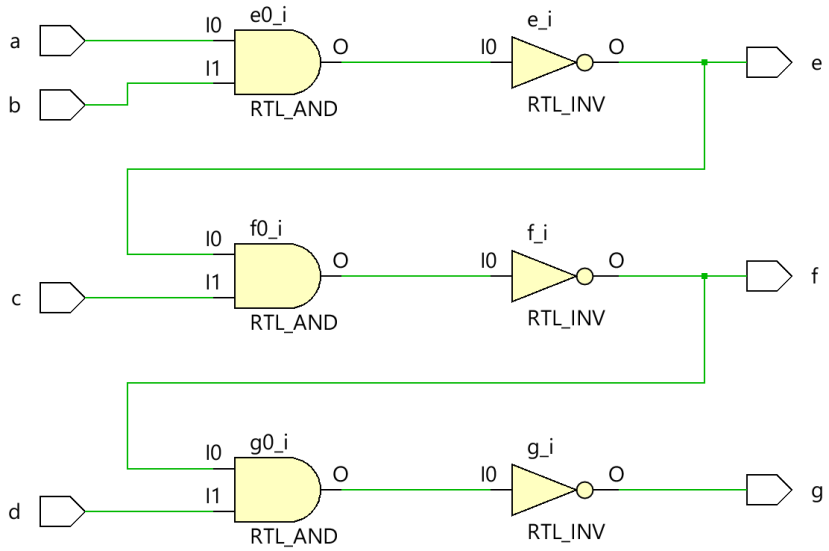
```
1   `timescale 1ns / 1ps
2
3   module four_input_NAND_gate_a(
4       input a, b, c, d,
5       output e
6       );
7
8       assign e = ~(a & b & c & d);
9
0   endmodule
```

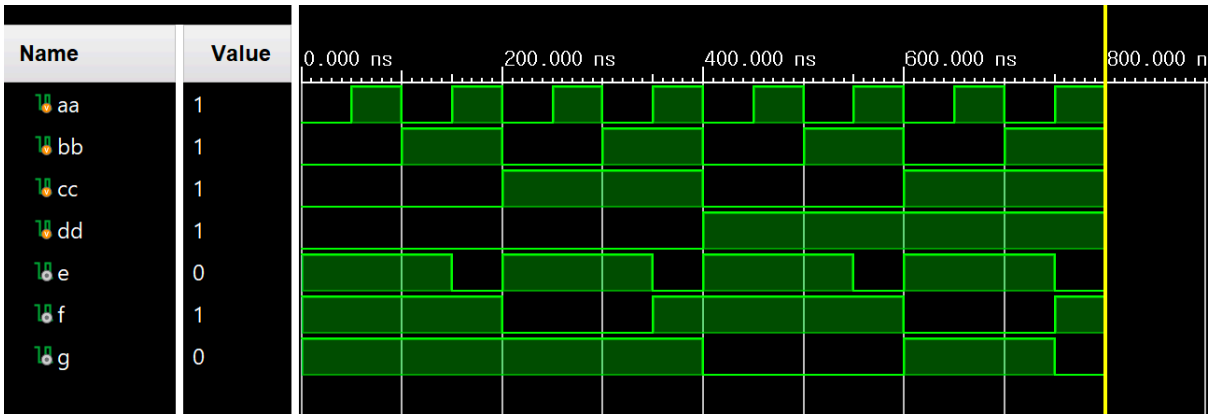




(B) 는 assign e = ~(a & b), assign f = ~(e & c), assign g = ~(f & d)와 같이 나타낼 수 있다. Schematic 과 Simulation 결과는 아래와 같다.

```
1   `timescale 1ns / 1ps
2
3   module four_input_NAND_gate_b(
4       input a, b, c, d,
5       output e, f, g
6       );
7
8       assign e = ~(a & b);
9       assign f = ~(e & c);
10      assign g = ~(f & d);
11
12  endmodule
```





(A)와 (B)의 시뮬레이션 결과를 비교해 보면 최종 결과값이 다르게 나타남을 알 수 있다. 그 이유는, 한번에 e = ~(a & b & c & d)를 한 것과 e = ~(a & b), f = ~(e & c), g = ~(f & d)를 한 것이 다르기 때문이다. (A)의 경우는 4-input AND gate 의 결과를 부정한 것과 같지만, (B)는 2 개를 묶어 NAND 연산을 하고, 다시 다른 입력과 묶어 NAND… 식으로 반복하기에 결과가 다르다.

(A): $E = \overline{A \cdot B \cdot C \cdot D}$, (B): $E = \overline{A \cdot B}, F = \overline{C \cdot E}, G = \overline{D \cdot F}$

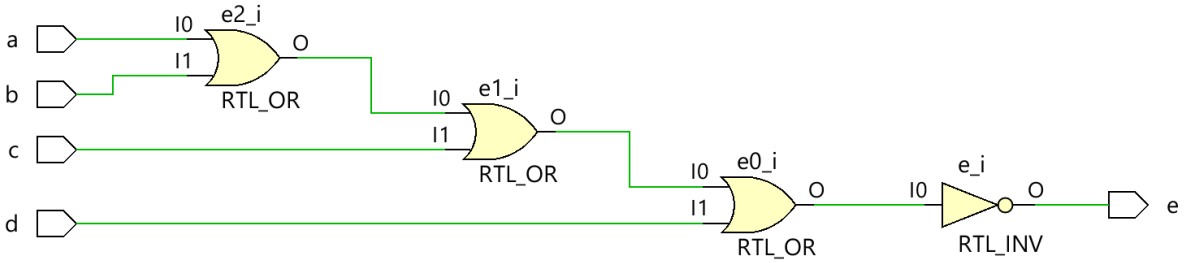
아래는 4-input NAND gate-(B)의 진리표이다.

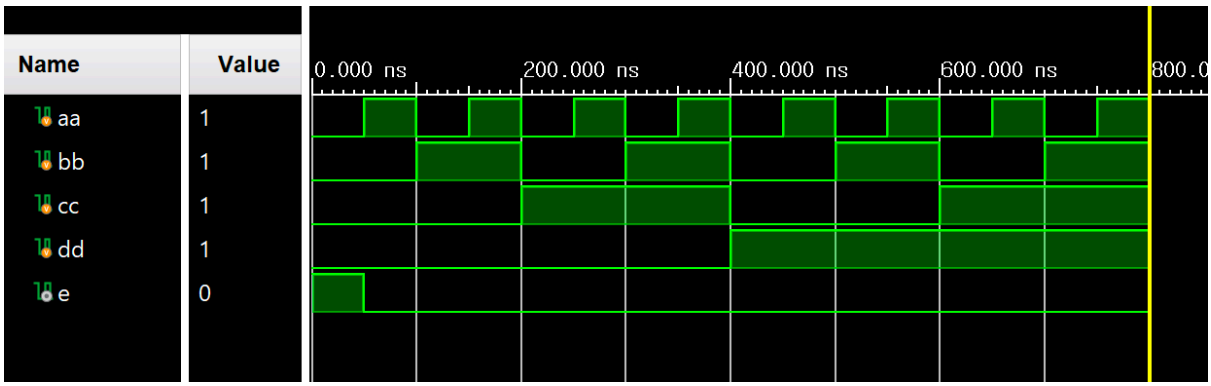
Input a	Input b	Input c	Input d	Output e	Output f	Output g
0	0	0	0	1	1	1
0	0	0	1	1	1	0
0	0	1	0	1	0	1
0	0	1	1	1	0	1
0	1	0	0	1	1	1
0	1	0	1	1	1	0
0	1	1	0	1	0	1
0	1	1	1	1	0	1
1	0	0	0	1	1	1
1	0	0	1	1	1	0
1	0	1	0	1	0	1
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	0	1	0
1	1	1	0	0	1	1
1	1	1	1	0	1	0

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

(A) 는 assign e = ~(a | b | c | d)와 같이 나타낼 수 있다. Schematic 과 Simulation 결과는 아래와 같다.

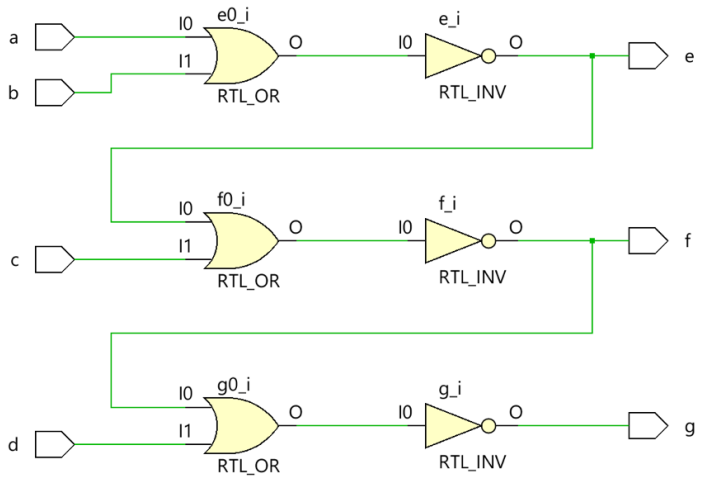
```
1  `timescale 1ns / 1ps
2
3  module four_input_NOR_gate_a(
4      input a, b, c, d,
5      output e
6      );
7
8      assign e = ~(a | b | c | d);
9
10 endmodule
```

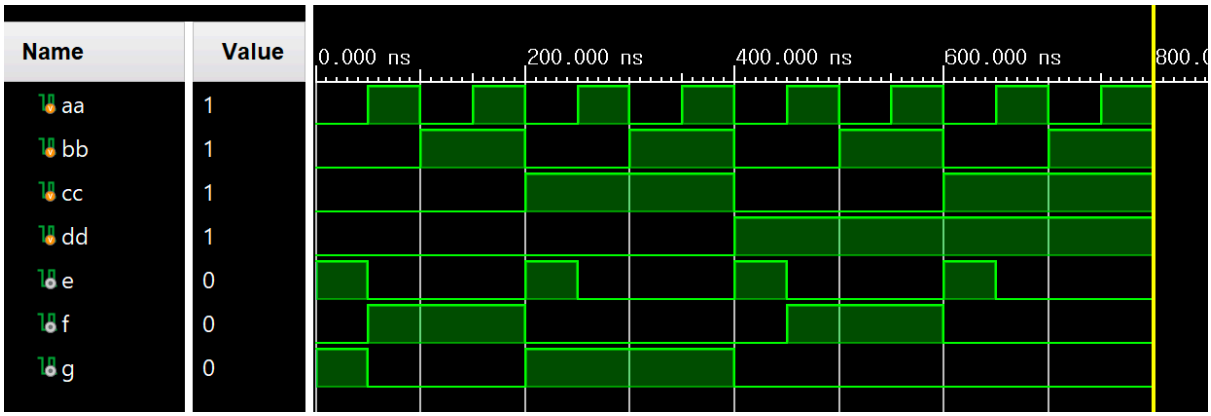




(B) 는 assign e = ~(a | b), assign f = ~(e | c), assign g = ~(f | d)와 같이 나타낼 수 있다. Schematic 과 Simulation 결과는 아래와 같다.

```
1       `timescale 1ns / 1ps
2
3       module four_input_NOR_gate_b(
4           input a, b, c, d,
5           output e, f, g
6           );
7
8           assign e = ~(a | b);
9           assign f = ~(e | c);
0           assign g = ~(f | d);
1
2       endmodule
```





(A)와 (B)의 시뮬레이션 결과를 비교해 보면 최종 결과값이 다르게 나타남을 알 수 있다. 그 이유는, 한번에 e = ~(a | b | c | d)를 한 것과 e = ~(a | b), f = ~(e | c), g = ~(f | d)를 한 것이 다르기 때문이다. (A)의 경우는 4-input OR gate 의 결과를 부정한 것과 같지만, (B)는 2 개를 묶어 NOR 연산을 하고, 다시 다른 입력과 묶어 NOR… 식으로 반복하기에 결과가 다르다.

(A): $\mathrm{E} = \overline{\mathrm{A} + \mathrm{B} + \mathrm{C} + \mathrm{D}}$, (B): $\mathrm{E} = \overline{\mathrm{A} + \mathrm{B}}, \mathrm{F} = \overline{\mathrm{C} + \mathrm{E}}, \mathrm{G} = \overline{\mathrm{D} + \mathrm{F}}$

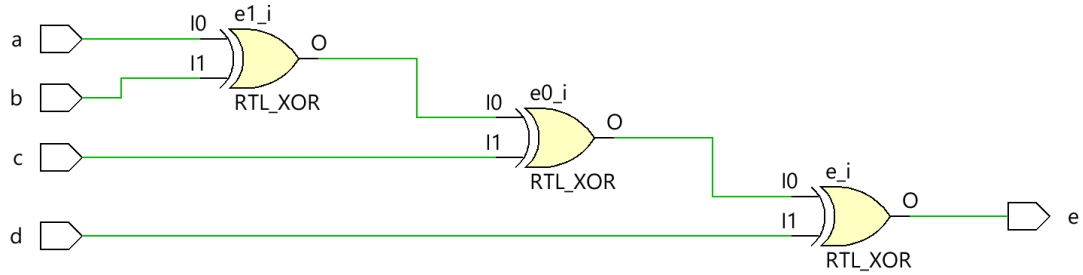
아래는 4-input NOR gate-(B)의 진리표이다.

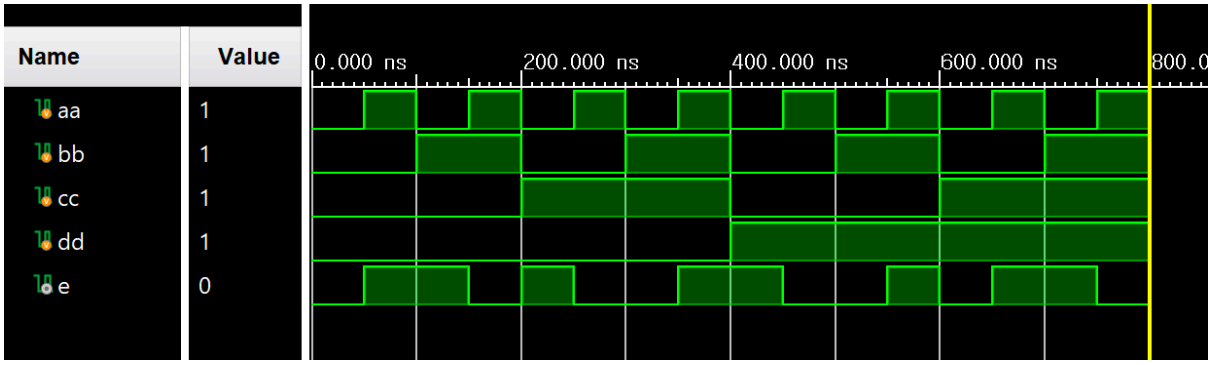
Input a	Input b	Input c	Input d	Output e	Output f	Output g
0	0	0	0	1	0	1
0	0	0	1	1	0	0
0	0	1	0	1	0	1
0	0	1	1	1	0	0
0	1	0	0	0	1	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	0	0	1
1	0	1	1	0	0	0
1	1	0	0	0	1	0
1	1	0	1	0	1	0
1	1	1	0	0	0	1
1	1	1	1	0	0	0

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

(A) 는 assign e = (a ^ b ^ c ^ d)와 같이 나타낼 수 있다. Schematic 과 Simulation 결과는 아래와 같다.

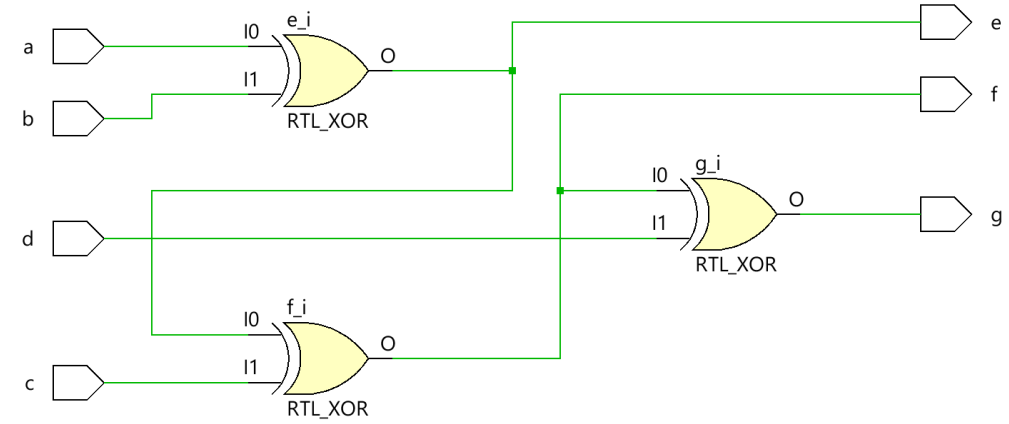
```
1   `timescale 1ns / 1ps
2
3   module four_input_XOR_gate_a(
4       input a, b, c, d,
5       output e
6       );
7
8       assign e = (a ^ b ^ c ^ d);
9
10  endmodule
```

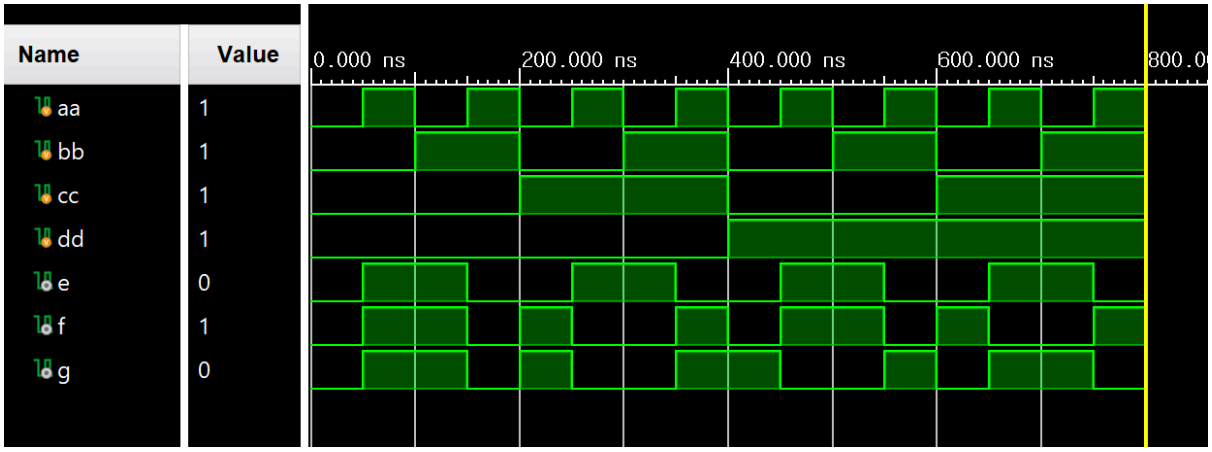




(B) 는 assign e = a ^ b, assign f = e ^ c, assign g = f ^ d 와 같이 나타낼 수 있다. Schematic 과 Simulation 결과는 아래와 같다.

```
1   `timescale 1ns / 1ps
2
3   module four_input_XOR_gate_b(
4       input a, b, c, d,
5       output e, f, g
6       );
7
8       assign e = a ^ b;
9       assign f = e ^ c;
10      assign g = f ^ d;
11
12  endmodule
```





(A)와 (B)의 시뮬레이션 결과를 비교해 보면 최종 결과값이 동일하게 나타남을 알 수 있다

(A): $\mathrm{E} = \overline{\mathrm{A} \oplus \mathrm{B} \oplus \mathrm{C} \oplus \mathrm{D}}$, (B): $\mathrm{E} = \overline{\mathrm{A} \oplus \mathrm{B}}, \mathrm{F} = \overline{\mathrm{C} \oplus \mathrm{E}}, \mathrm{G} = \overline{\mathrm{D} \oplus \mathrm{F}}$

구현 방식에 상관없이, 입력값 중 1(High)의 개수가 홀수이면 1, 짝수이면 0 을 반환한다.

아래는 4-input XOR gate-(B)의 진리표이다.

Input a	Input b	Input c	Input d	Output e	Output f	Output 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

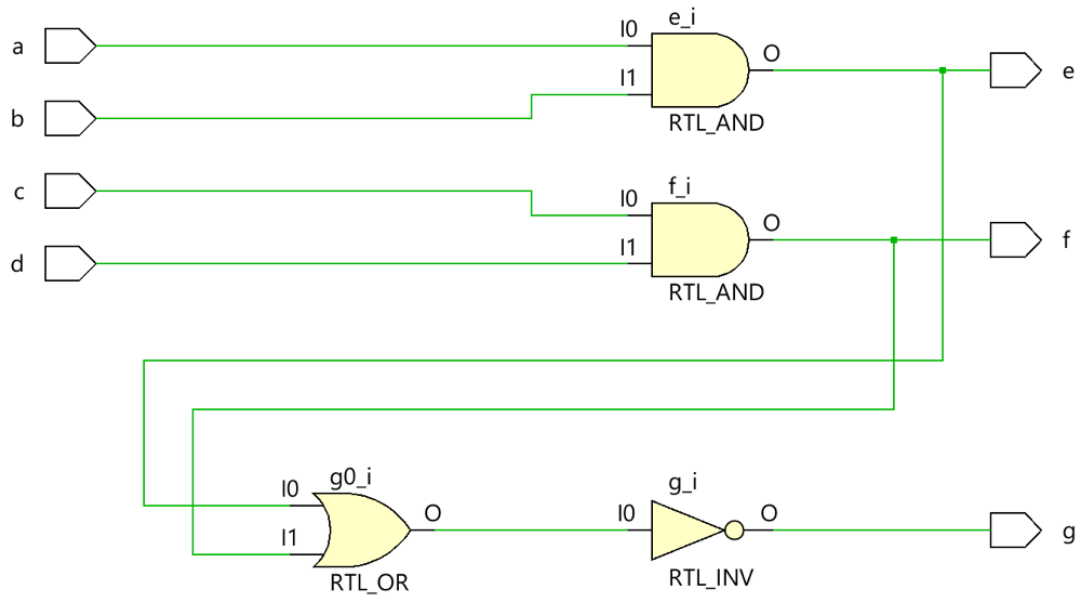
1	0	1	1	1	0	1
1	1	0	0	0	0	0
1	1	0	1	0	0	1
1	1	1	0	0	1	1
1	1	1	1	0	1	0

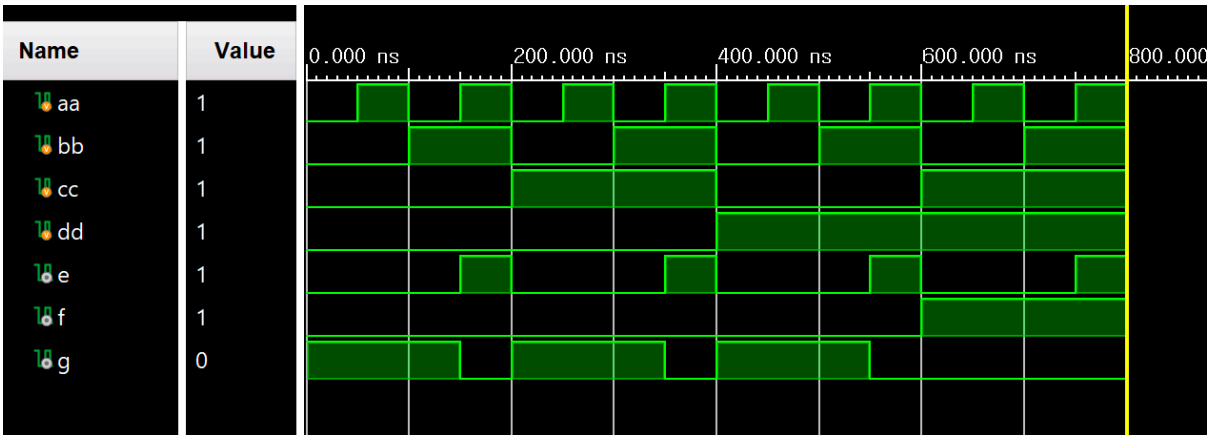
5. 4-input AOI gate 의 simulation 결과 및 과정에 대해서 설명하시오. (4 input, 3 output)

2 개의 AND gate 와 1 개의 NOR gate 를 이용해 구현 가능하다. assign e = a & b, assign f = c & d, assign g = ~(e | f)와 같이 나타낼 수 있으며

Schematic 과 Simulation 결과는 아래와 같다.

```
1   `timescale 1ns / 1ps
2
3   module four_input_AOI_gate(
4       input a, b, c, d,
5       output e, f, g
6       );
7
8       assign e = (a & b);
9       assign f = (c & d);
10      assign g = ~(e | f);
11
12  endmodule
```





4-input AOI gate 는 $G = \overline{(A \cdot B) + (C \cdot D)}$로 표현할 수 있다.

아래는 4-input AOI gate 의 진리표이다.

Input a	Input b	Input c	Input d	Output e	Output f	Output 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	0	1
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	0	1
1	1	0	0	1	0	0
1	1	0	1	1	1	0
1	1	1	0	1	0	0
1	1	1	1	1	1	0

6. 결과 검토 및 논의사항.

4-input NAND, NOR, XOR, AOI gate 를 Verilog 코딩하여 시뮬레이션과 Schematic 을 확인해 보았다. gate 의 종류에 따라 양상이 다르게 나타났는데, NAND 와 NOR gate 는 (A)와 (B)가 결과값이 다르게 나온 반면, XOR 은 동일하게 나왔다. 이는 수식으로 표현하여 증명할 수 있었다.

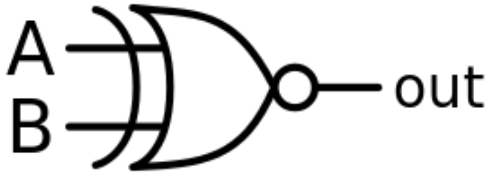
(후술)

~(a & b& c)와 같은 수식은 Schematic 상에서 PPT 의 (A) 모양이 나오지 않는데, 이는 Verilog 에서 기본적으로 3 개를 한꺼번에 연산하는 것으로 나타내지 않고 따로 풀어서 나타내기 때문이다.

- 4-input NAND gate 에서 (A)는 일반적인 2-input NAND gate 처럼 입력값이 모두 0 인 경우에만 0 을 출력하였지만. (B)는 그렇지 않았다. ($\because \overline{A \cdot B \cdot C \cdot D} \neq \overline{\overline{A \cdot B \cdot C} \cdot D}$) -> 수식의 결과가 다름.
- 4-input NOR gate 에서 (A)는 일반적인 2-input NOR gate 처럼 입력값이 모두 0 인 경우에만 1 을 출력하였지만. (B)는 그렇지 않았다. ($\because \overline{A + B + C + D} \neq \overline{\overline{A + B + C} + D}$) -> 수식의 결과가 다름.
- 4-input XOR gate 는 (A)와 (B)의 결과가 같았다. 이는 XOR gate 는 입력값 중 1(High)의 개수에 따라 결과값을 결정짓는다.
- AOI gate 를 구현하고 그 결과를 보았다. 이후 AOI gate 를 활용해 더 복잡한 회로를 설계하는데 이용할 수 있을 것이라는 점을 확인하였다.

7. 추가 이론 조사 및 작성.

3, 4 주차에 걸쳐 AND, OR, NOT, NAND, NOR, XOR, AOI gate 를 확인하였다. 그 이외에도 이를 조합한 다른 gate 들이 존재한다. 그 중 하나는 XNOR gate 이다. XOR gate 에 NOT 을 붙인 것인데, 주의할 점은 NXOR 이 아니라 XNOR gate 로 표현한다는 것이다.



↑XNOR gate 표현법

다음은 XNOR gate 의 진리표이다.

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

진리표를 통해 확인할 수 있듯이, 두 입력값이 같으면 1 을 출력, 다르면 0 을 출력하므로 일치 확인 게이트, 비교 게이트로도 부른다.

$Y = \overline{A \oplus B}$와 같이 표현할 수 있다. 다음은 이를 실제 회로로 구현한 모양이다.