컴퓨터공학실험2 4주차 예비 보고서

전공: 컴퓨터공학 학년: 2학년 학번: 20201635 이름: 전찬

**0. 목차**

1. 논리 게이트 NAND/NOR/XOR의 Transistor-Level 표현

2. NAND/NOR/XOR의 특성 조사

3. 기본 논리 게이트(AND/OR/NOT)와의 변환 관계

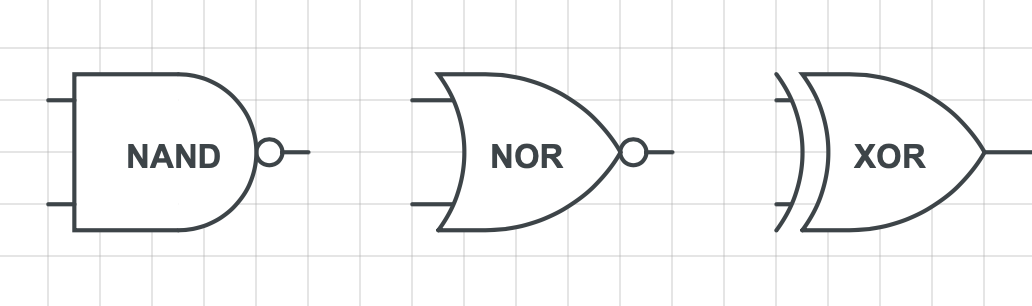
4. AND-OR-INVERT logic의 응용

5. XOR logic의 구현 방법

6. 기타 이론

**1. 논리 게이트 NAND/NOR/XOR의 표현**

NAND/NOR/XOR의 Transistor-Level 표현은 다음과 같다.



<NAND/NOR/XOR의 표현>

**2. NAND/NOR/XOR의 특성**

논리 게이트 NAND는 AND gate에서 output에 NOT gate를 연결한 것과 동일하다. 모든 input이 1이라면 output이 0이며, 이외에는 output이 전부 1인 형태이다. 2-input에서의 진리표는 아래와 같다.

|  |  |  |
| --- | --- | --- |
| 2-input NAND gate | | |
| input | | output |
| A | B | C |
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |

논리 게이트 NOR은 OR gate에서 output에 NOT gate를 연결한 형태이다. 모든 input이 0일 때에만 output을 1로 가질 수 있으며, 그 이외의 경우에는 output을 0으로 갖는다. 2-input에서의 진리표는 아래와 같다.

|  |  |  |
| --- | --- | --- |
| 2-input NOR gate | | |
| input | | output |
| A | B | C |
| 1 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

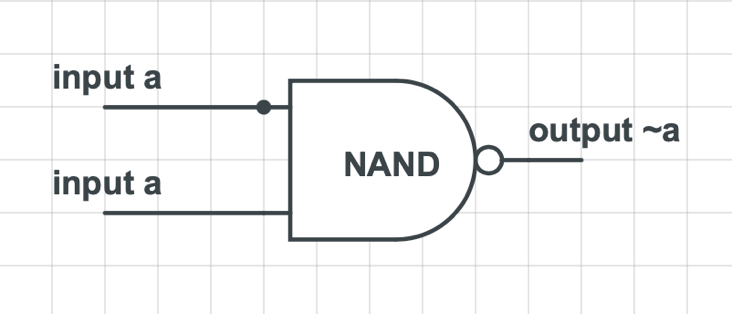
논리 게이트 XOR은 ‘배타적 논리합’으로 두 input이 서로 다를 때에만 output으로 1을 갖는 gate이다. 예를 들면, input1 = 1, input2 = 0 인 형태에서 ouput = 1 이다. 2-input에서의 진리표는 아래와 같다.

|  |  |  |
| --- | --- | --- |
| 2-input XOR gate | | |
| input | | output |
| A | B | C |
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

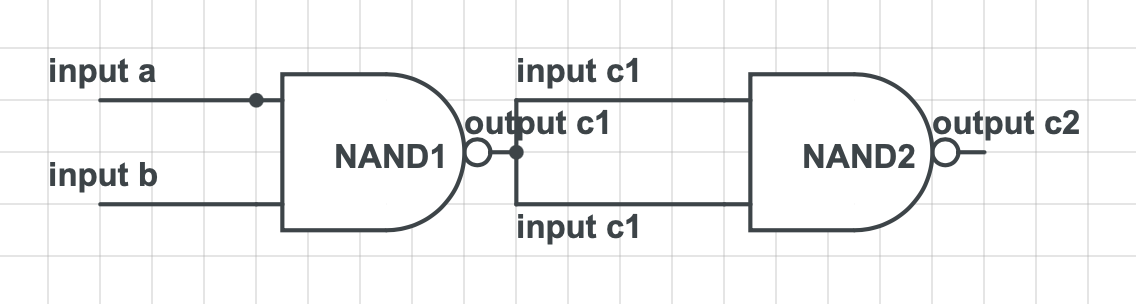
**3. 기본 논리 게이트와의 변환 관계**

거의 모든 복잡한 gate들은 AND, OR, NOT gate의 복잡한 결합으로 이루어지게 된다. 앞에서 설명한 NAND, NOR gate 또한 AND, OR gate의 출력 결과에 NOT gate를 적용시켜서 만들게 된다. 이때 AND, OR, NOT 3개의 gate를 기본 논리 게이트라고 할 수 있다. 하지만 위 3개로 다양한 gate를 만들어내는 것보다, 1가지의 gate로 다양한 gate를 만들 수 있다면 더욱 효율적일 것이며, NAND, NOR gate가 그 역할을 수행해낼 수 있다. 따라서 NAND, NOR gate를 Universial gate 라고 하기도 한다.

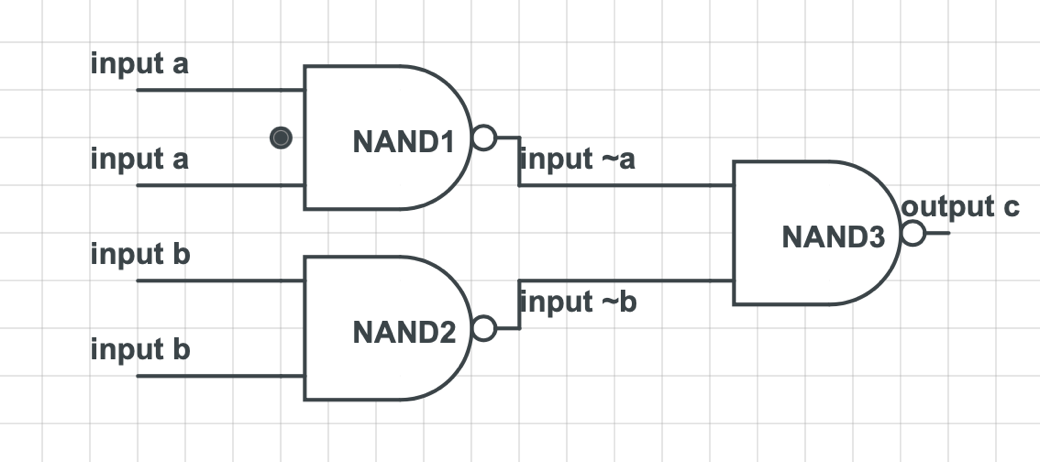
이번엔 NAND gate를 통해서 AND, OR, NOT gate를 구현하는 방법에 대해 알아보자. 우선 제일 먼저 NOT gate를 구현할 것인데, NOT gate는 간단하게 NAND gate의 두 input에 동일한 input NOT에 들어갈 input을 대입해주면 간단하게 NAND gate로 NOT gate를 만들어낼 수 있다. 형태는 아래와 같다.



두 번째로 NAND를 통한 AND 구현은 NAND가 NOT의 부정 형태인 것을 참고해서 마지막에 NOT을 다시 취하는 형태로 만들어줄 수 있다. NAND + NOT = AND가 되는 형태이며, 이때 NOT을 위에서 NAND로 구해낸 형태로 사용해주면 NAND만으로 AND gate를 구현할 수 있으며, 형태는 아래와 같다.



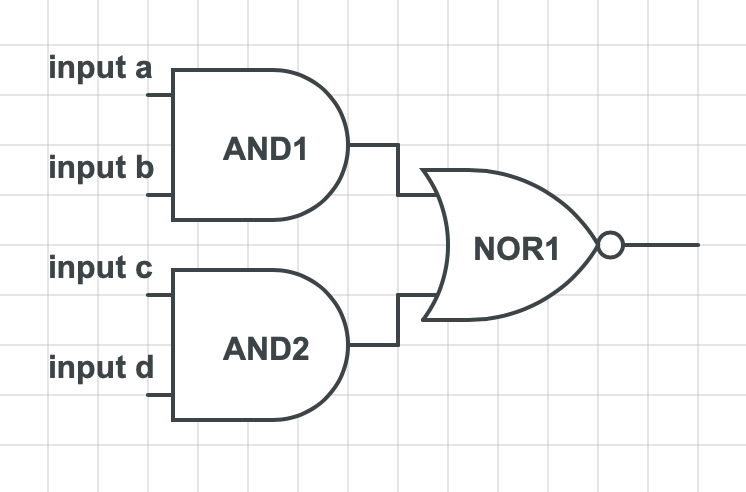
마지막으로 OR gate를 만들기 위해서는 고등학교 때 배운 드모르간의 법칙을 생각해보면 된다. AND의 결과에 NOT을 취하는 형태는 두 input에 NOT을 먼저 취하고 OR을 취하는 것과 동일하다. 따라서 두 input에 다시 한 번 NOT을 취해준다면 AND gate와 동일할 것임을 파악할 수 있다. NAND = NOT(2 input) + OR인데, 이때 2 input에 NOT을 다시 취해준다는 말이다. 그 형태는 아래와 같다.



반대로 NOR gate로 또한 AND, OR, NOT gate를 구현해낼 수 있는데, 이 과정은 NAND gate로 구현하는 과정과 비슷하므로 생략한다.

**4. AND-OR-INVERT logic의 응용**

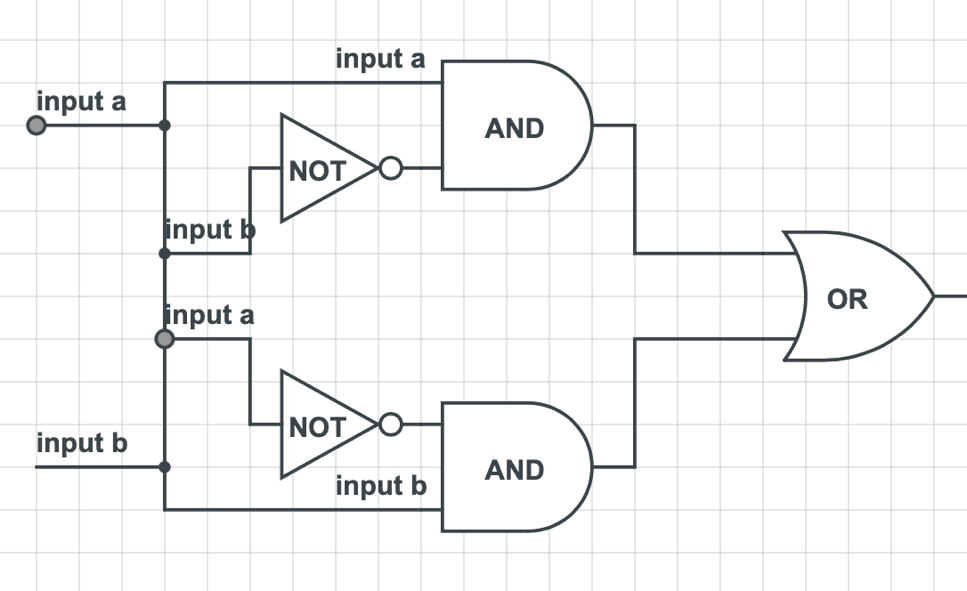
AND-OR-Invert는 4개의 input을 가지며, 2개의 AND gate 이후에 NOR gate를 지나는 형태이다. 실제 형태는 아래와 같다.



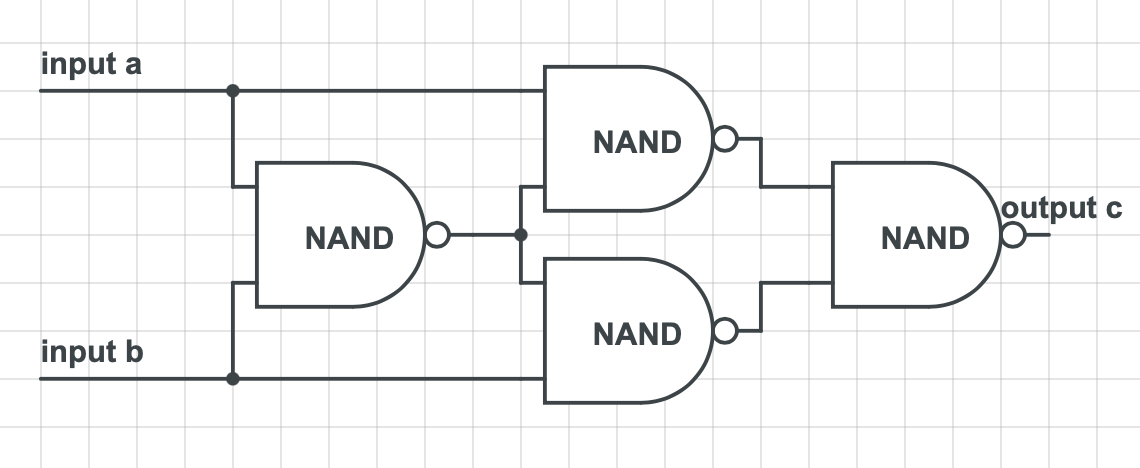
위와 같은 AND-OR-Invert가 중요한 이유는, NAND, NOR gate와 마찬가지로 universal gate로 여길 수 있기 때문이다. 위에서 진행한 과정들처럼 AOI gate를 여러 방법으로 조합하면 AND, OR, NOT gate들을 구현해낼 수 있다.

**5. XOR logic의 구현 방법**

XOR은 exclusive or의 줄임말로, 2-input에서 input들이 모두 같은 값을 가진다면 output으로 0을 가지는 gate이다.(input이 많아진다면, input의 High(1)개수가 홀수이면 output으로 1, 짝수이면 output으로 0을 가지는 gate이다) Boolean 표현으로는, 두 input이 A, B 라면 output = AB’ + A’B 라고 할 수 있다. 결국 AB’, A’B를 AND로, 이후에 두 output을 OR에 적용시키면 XOR gate를 만들어낼 수 있다. 형태는 아래와 같다.

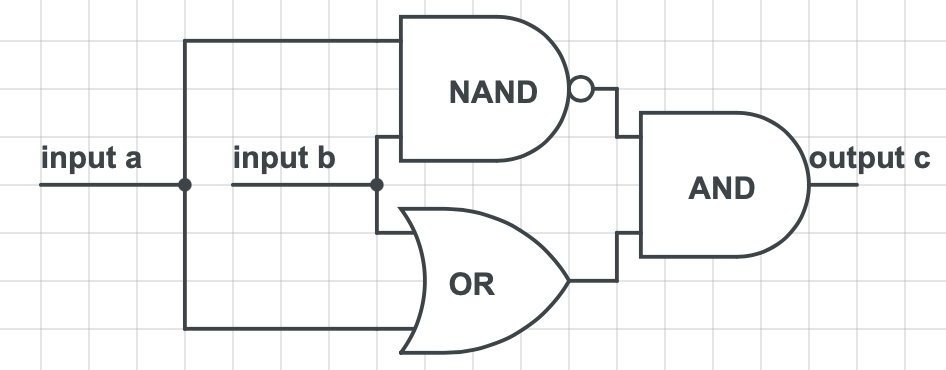


하지만 위에서 설명한 것처럼, AND, OR, NOT 3개의 gate를 사용하는 것보다, NAND나 NOR gate만으로 XOR gate를 만들어낼 수 있다면, 훨씬 효율적일 것이다. 이를 NAND로 a만들어내는 방법은 아래와 같다.



NOR gate 또한 NOR gate만으로 XOR gate를 구현할 수 있으며, 이는 생략하겠다.

NAND gate로 XOR gate를 만들어낼 때에는 4개의 gate를 사용해서 만들어낼 수 있었다. 여기에서 gate 수를 줄여서 효율적으로 만드는 방법이 존재하는데, NAND, AND, OR gate를잘 섞어서 만들어낼 수 있다. 그 형태는 아래와 같다.



**6. 기타 이론**

-Boolean expression을 통해 두 회로가 동일한지 증명하는 방법

위에서 여러 gate로 다른 형태의 gate들을 만들어내는 과정을 진행했는데, 이를 검증하기 위해서 진리표를 직접 만들어볼 수 있을 것이다. 하지만 또 다른 방법으로는 Boolean expression 을 통해서 논리적으로 동일한지 비교할 수도 있다. 이를 바로 위에 있는 NAND, OR, AND를 통해서 XOR을 만들어내는 과정에서 진행해보자.

우선 ~A = A’, ~B = B’, a = A, b = B, c = C라고 전제하고 시작해보자.

A, B에 대해서 NAND를 취하므로, NAND의 output은 (AB)’ = A’+B’ (드모르간의 법칙)

동일하게 OR의 output은 A+B 이다.

이에 대해서 다시 AND를 취해주면 output C = (A’+B’)(A+B) = A’A + A’B + B’A + B’B 이다.

이때 AA’, BB’의 값은 항상 0이므로 output C = A’B+B’A라고 할 수 있으며, 이는 실제 XOR gate의 Boolean expression과 동일하다는 것을 알 수 있다. 따라서 논리적으로 같음을 파악해낼 수 있다.

-3-input NAND, NOR, XOR gate, 3-input XOR gate의 구현

**7. 추가 사항**

모든 회로도 그림은 <https://www.circuitlab.com> 의 회로 그리기 툴을 이용해서 제작했습니다.