6주차 결과보고서

전공: 아트앤테크놀로지 학년: 3학년 학번: 20191098 이름: 백승주

**1.**

- half adder와 full adder의 Verilog 소스 코드를 작성하고 그를 토대로 시뮬레이션 결과와 진리표를 확인한다.

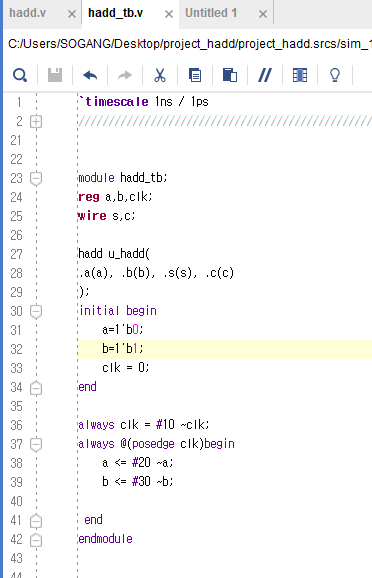
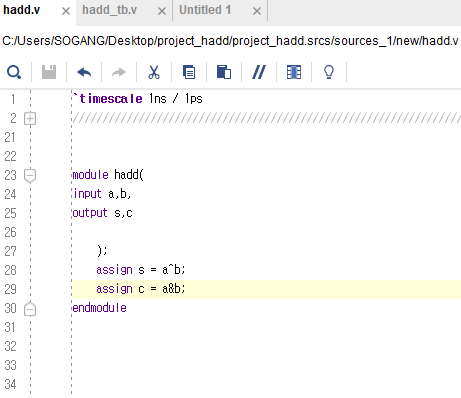
- half subtractor와 full subtractor의 Verilog 소스 코드를 작성하고 그를 토대로 시뮬레이션 결과와 진리표를 확인한다.

- 8421(BCD)-2421 Code converter 의 표를 토대로 진리표를 작성하고 카르노 맵을 통해 수식을 완성해 Verilog 소스 코드를 작성한다. 이후 FPGA를 이용해 구현된 회로의 동작을 직접 확인해본다.

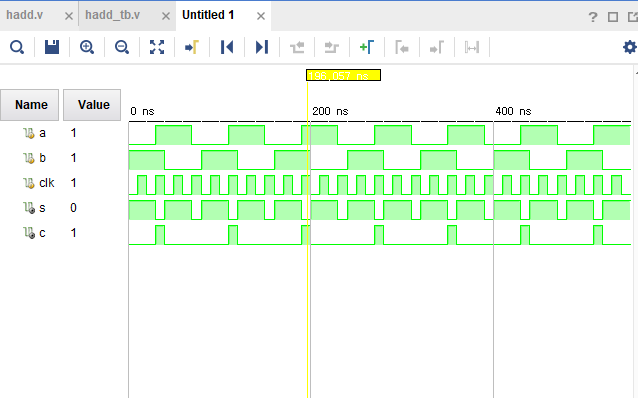
**2-1.**



Half Adder 게이트는 A, B라는 두 개의 input을 받아 sum과 carry라는 결과값을 출력해 덧셈을 계산해주는 회로다. 위의 회로 그림에서는 A와 B의 Xor 게이트를 통해 결과값 s를 출력하고 있으며 A와 B의 And 게이트를 통해 결과값 c를 출력하고 있다. Verilog 코드를 통해 이를 구현하면 다음과 같다.



xor 게이트를 a^b를 통해 표현했고 and 게이트를 a&b를 통해 표현했다. 이러한 코드를 testbench 파일로 나타내면 a와 b의 값들이 서로 다른 시간간격을 두고 바뀌며 그에 따른 결과값들을 시뮬레이션 결과로 나타내게 된다. 시뮬레이션 결과는 다음과 같다.



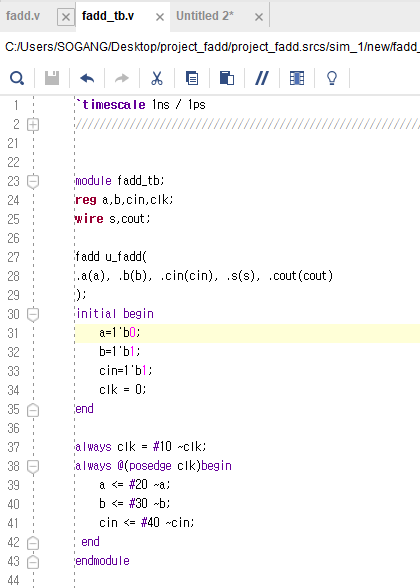
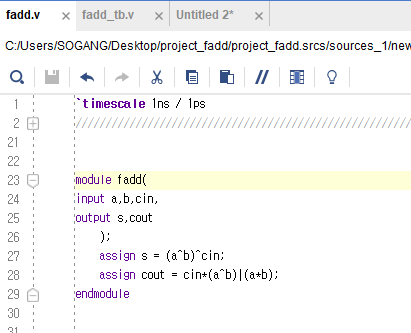
Carry 값은 a와 b 모두 1일 때에만 1을 출력했다. 이에 따른 진리표는 다음과 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **S** | **C** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

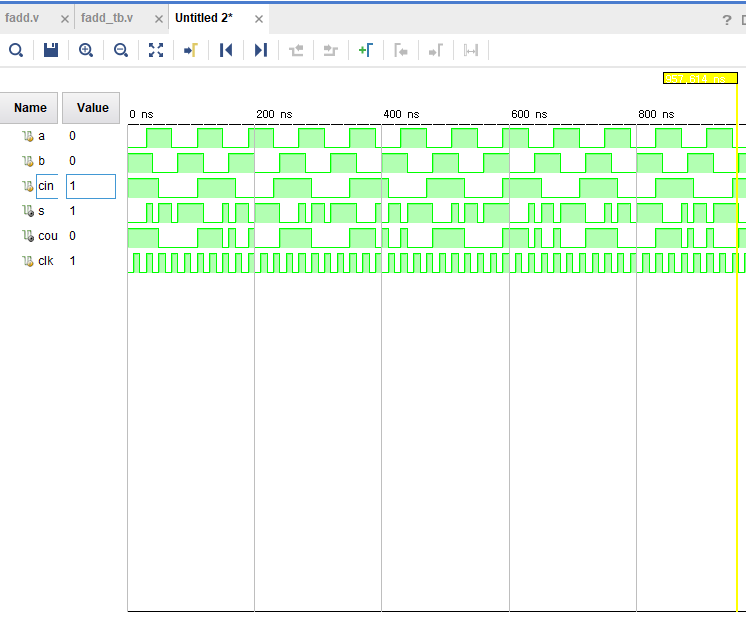
**2-2. Full adder**



Full Adder 게이트는 A, B, carry in이라는 세 개의 input을 받아 sum과 carry out라는 결과값을 출력해 덧셈을 계산해주는 회로이다. 위 그림의 회로를 수식으로 나타내면 Sum은 (A⊕B)⊕Carry in로 표현할 수 있고 Carry out은 Carry in(A⊕B)+AB로 표현할 수 있다. Verilog 소스를 통해 이를 구현하면 다음과 같다.



⊕, 즉 xor게이트는 ^를 통해 표현했으며 or 게이트는 | 기호를 통해 표현했다. 이에 따른 시뮬레이션 결과는 다음과 같다.



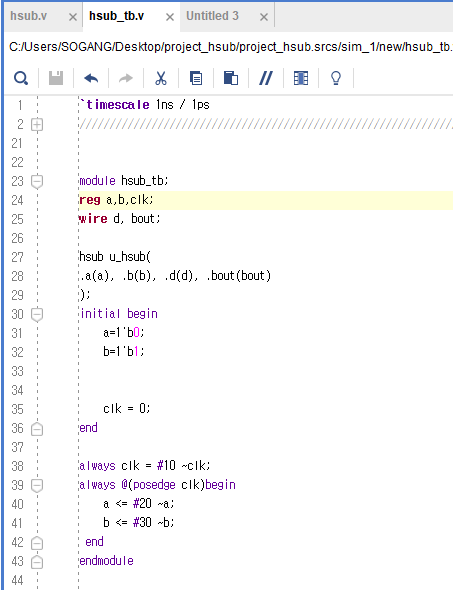
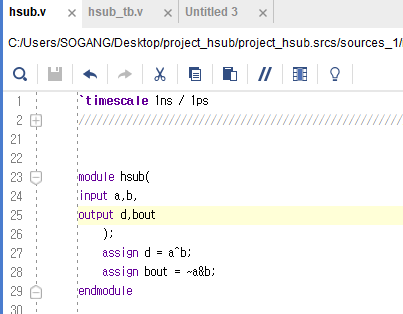
이 결과를 토대로 진리표를 완성하면 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | Cin | S | Cout |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

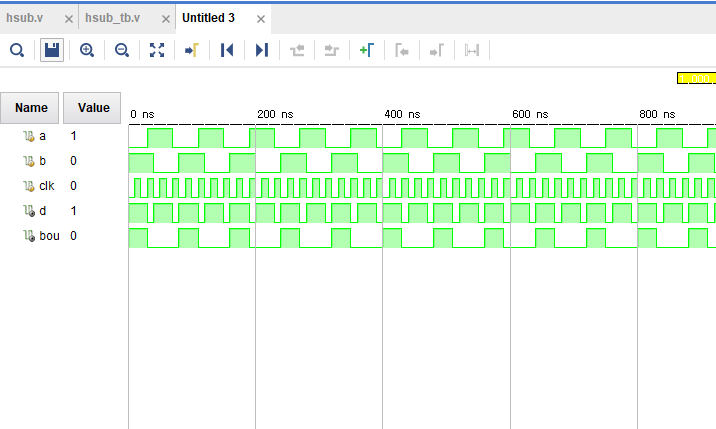
**3-1.**



Half subtractor는 A와 b input을 받아 difference와 borrow를 출력하는 게이트다. 위 회로에서 보다시피 difference는 A⊕B로 표현하고 borrow는 A’&B로 표현이 가능하다. 이를 Verilog 소스 코드를 통해 구현하면 다음과 같다.



~a&b를 통해 A’&B를 표현했으며 a^b를 통해 택 게이트를 표현했다. 이에 따른 시뮬레이션 결과와 시뮬레이션 결과에 따른 진리표는 다음과 같다.

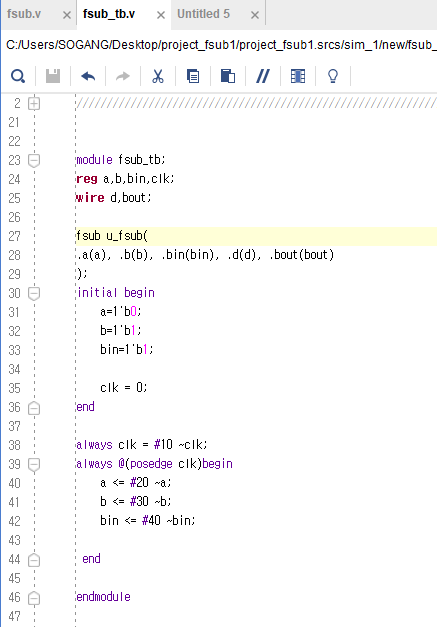
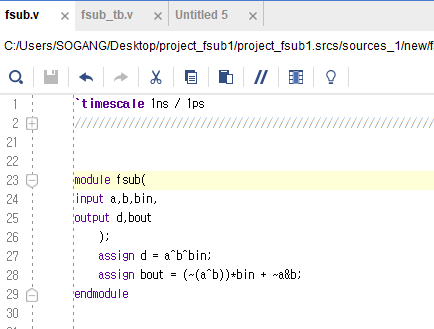


|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **D** | **B** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

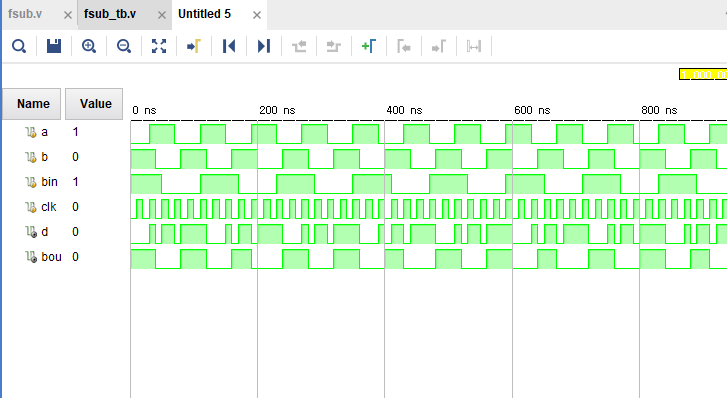
3-2.



위 회로는 full subtractor의 회로다. 위 회로는 half subtractor와 다르게 borrow in 이라는 하나의 input이 추가되었다. 위 회로를 수식으로 나타내면 difference는 A⊕B⊕Borrow in으로 표현할 수 있고 borrow out은 는 (A⊕B)’Borrow\_in + A’B으로 표현이 가능하다. 이를 Verilog 소스 코드로 testbench 소스와 함께 구현하면 다음과 같다.

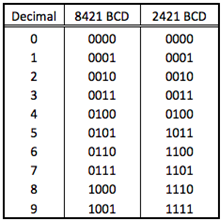


또한 위 식들에 대한 시뮬레이션과 리표는 다음과 같다.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **Bin** | **D** | **Bout** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

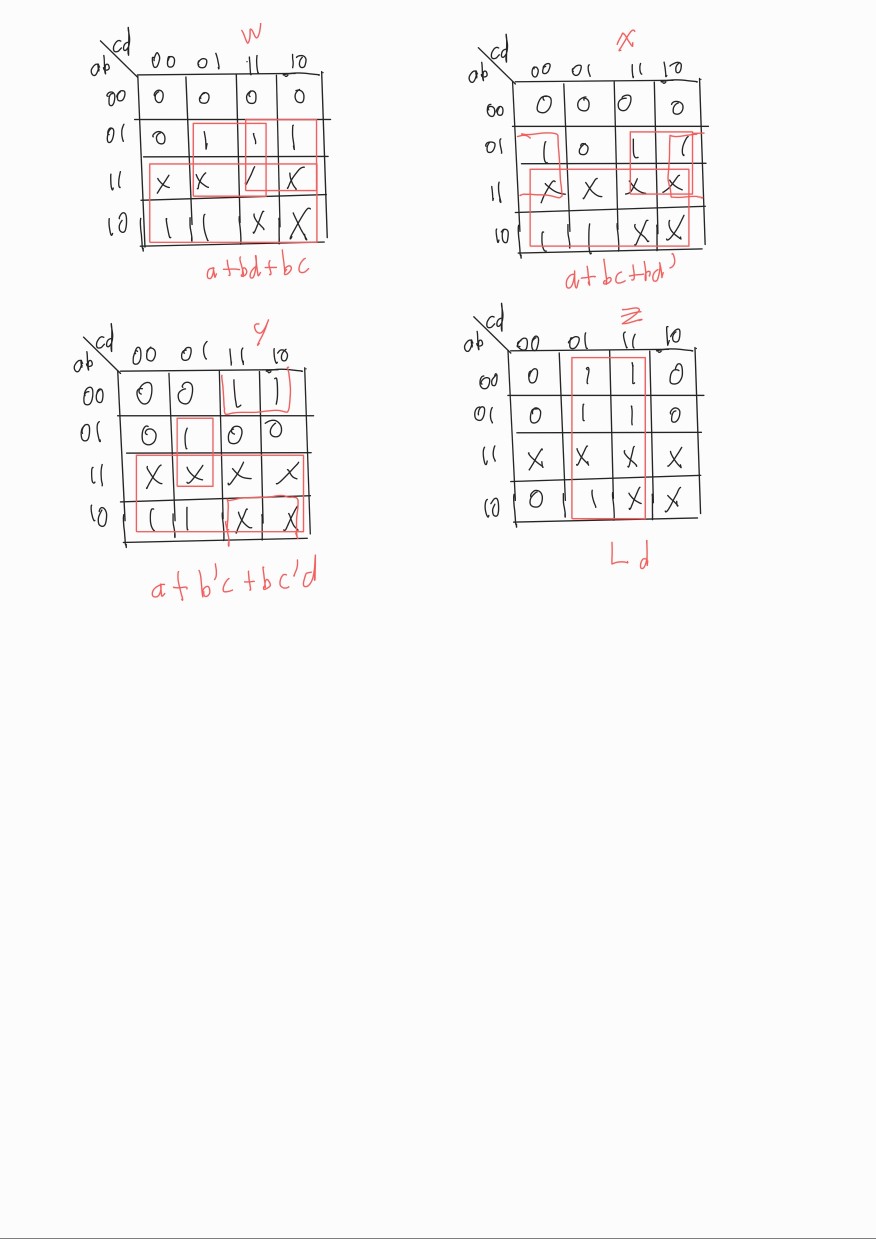
4.



8421(BCD)-2421 Code converter의 경우 강의 자료에 표가 주어졌고 이를 토대로 truth table을 만들 수 있다.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Decimal | 8(a) | 4(b) | 2(c) | 1(d) | 2(w) | 4(x) | 2(y) | 1(z) |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 8 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 9 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

따라서 이 truth table에 따라 카르노 맵을 그려 식을 만들어야 한다. 카르노 맵을 그리면 다음과 같다.

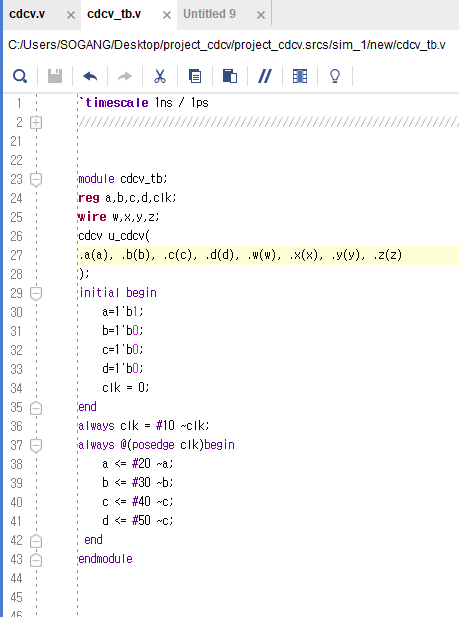
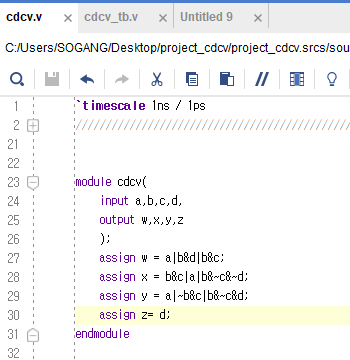
  
 카르노 맵을 이용해 minimum sop를 나타낼 수 있다. 위의 카르노 맵을 이용해 카르노 맵에서 결과값이 1로 나오는 경우들을 묶어 수식으로 간단히 나타낸다. 그 결과 각 결과값들의 sop를 구할 수가 있다. w는 a+ bd +bc, x는 a+bc+bd’ y 는 a+b’c+bc’d, z는 d로 나타낼 수 있다.  
또한 이렇게 구한 sop 형태에 드 모르간 법칙을 적용하면 pos 형태 역시 얻을 수가 있다. 그렇게 얻게 되는 pos 형태는 다음과 같다.

w- (a+c+d)(a+b)

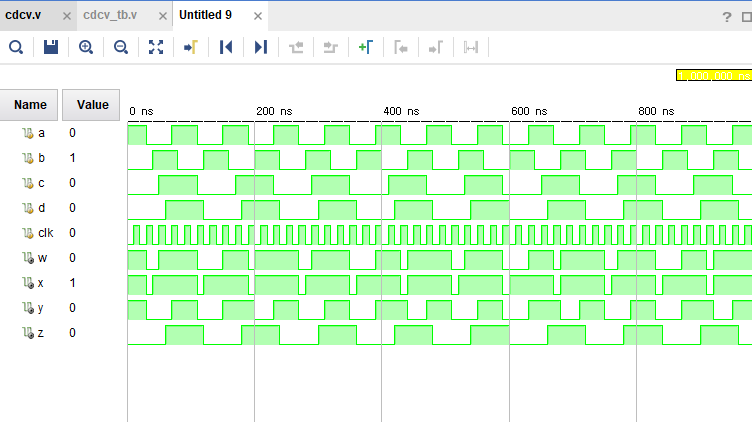
x – (a+c+d’)(a+b)

y – (a+c+d)(a+b+c)(b’+c’)

z - d  
Verilog 코드로 나타내면 다음과 같다.



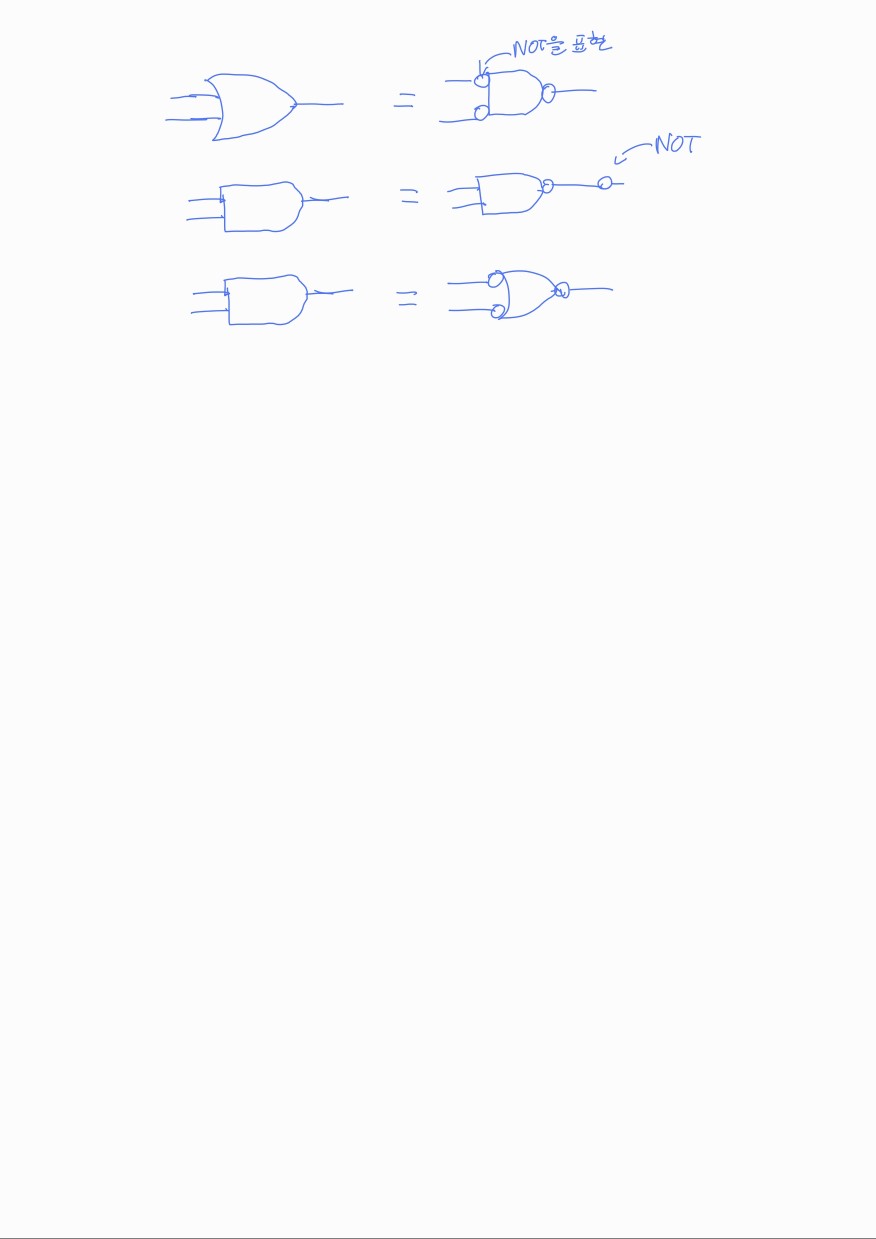
이에 대한 시뮬레이션 결과는 다음과 같다.



위 시뮬레이션 결과는 표를 통해 구한 진리표의 결과와 같다.

**5.** Half Adder, Full Adder, Half Subtracter, Full Subtracter, 8421(BCD)-2421 Code converter의 소스 코드를 작성하고 그에 따른 시뮬레이션 결과를 확인하고 진리표를 작성했다. 또한 8421(BCD)-2421 Code converter의 경우 주어진 진리표를 토대로 카르노 맵을 통해 간소화된 회로를 직접 설계해 그 결과를 확인할 수 있었다. 또한 앞선 4주차 예비보고서에서 Nand 게이트와 Nor 게이트로 AND 게이트와 OR 게이트 , NOT 게이트를 모두 변환해 표현할 수 있음을 배웠는데 마찬가지로 이번에 설계한 8421(BCD)-2421 Code converter의 회로도 마찬가지로 회로를 모두 Nor 게이트 또는 Nand 게이트로 나타낼 수 있을 것이다.

**6.** Nand 게이트와 Nor 게이트로 변환하기 위해서는 기존가지는 회로의 수식을 나타내고 수식 전체에 not을 씌우며 드 모르간 법칙을 사용해 Nand 게이트나 Nor 게이트로 변환했다. 그런데 만일 회로도가 주어졌다면 드 모르간 법칙을 이용한 And 게이트와 or 게이트가 nand 게이트로 변환되는 패턴을 익히고 있으면 회로도에서 쉽게 Nand 게이트나 Nor 게이트로 게이트를 변환할 수 있다.



위 예시는 and 게이트와 or 게이트를 회로도에서 간단하게 nand 게이트로 변환한 것을 나타냈다. 복잡한 회로도를 nand 게이트로 바꿀 때 위 패턴을 이용해 바꿔준 뒤 겹치는 not을 제거해주면 회로도를 nand 게이트만을 이용해 간단하게 나타낼 수 있다.