Lab1 report

20210774 김주은

1. Lab1_1

1) 개요

- 기본 게이트 중 and gate를 verilog로 구현하고, 테스트벤치를 작성하고 시뮬레이션한다.

2) 이론적 배경

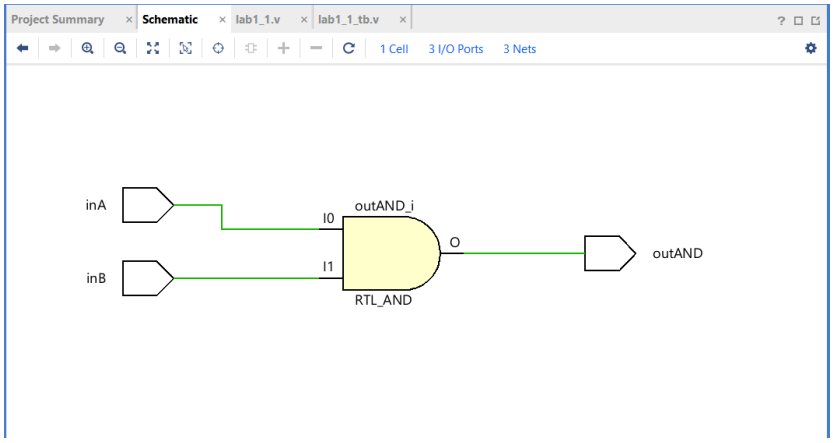
- Verilog로 구현하는 And gate의 경우 input이 x와 y일 때 둘 다 1인 경우 1을 내놓고 아닌 경우는 0을 내놓는 gate에 해당한다.
- And gate를 구현하는 verilog의 경우, HDL 중 하나로 하드웨어 동작을 표현하기 위한 언어이다. 기능에 따라 Module을 구성하고 이를 연결하는 구성으로 되어있다.
- 테스트벤치를 작성하게 되는데, 테스트벤치는 시뮬레이션을 하기 위한 설명서라고 할 수 있으며 작성한 module을 불러와 적절하게 I/O를 연결하고, module에 넘겨준 input과 output의 시간에 따른 state 변화를 확인한다.

3) 실험 준비

- And gate의 truth table이다.

A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

- And gate를 verilog로 구현했을 때 나온 회로이다.



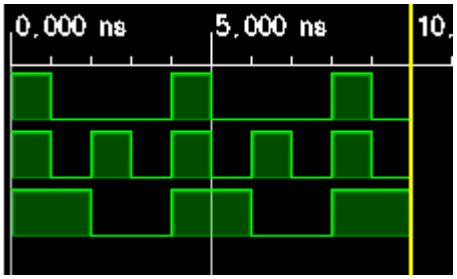
- and gate를 구현하기 위해 and(outAND, inA, inB);로 Verilog 코드에 작성한다.

4) 결과

- 테스트벤치 파일에 A,B 값의 초기화를 위해 A=1; B=1; 을 추가해주고 10ns동안 시뮬레이션을 수행하기 위해서 #10 $finish 을 추가해준다.
- 2ns마다 B를 반전시키기 위해 #2 B <= !B; 를 추가해준다.
- 작성된 테스트벤치 전체 내용은 다음과 같다.

```
`timescale 1ns / 1ps

module lab1_1_tb();
    wire out;
    reg A, B;

    lab1_1 AND(out, A,B);

    /* Initialize A and B */
    initial begin
        ////////////////////////
        A = 1;
        B = 1;
        #10 $finish;
        ////////////////////////
    end

    /* Flip A every 1ns */
    always begin
        #1 A <= !A;
    end
```

- 시뮬레이션 파형을 캡쳐한 결과는 다음과 같다.



- 0에서부터 10ns까지 결과가 나오므로 10ns동안 시뮬레이션 하는 것이 잘 구현됨을 알 수 있으며, A와 B가 1ns, 2ns마다 반전되며 이에 따라 out의 결과도 변하는 것을 발견할 수 있다. 이로써, 시뮬레이션을 통해 구현에 성공했음을 알 수 있다.

5) 논의

- And gate를 verilog를 구현하는 과정에서 verilog 문법을 처음 접해보는 상황에서 처음에 어떻게 해야할 지 막막했지만 lab0에 했었던 문법을 다시 이해해보고 조교님의 설명을 다시 이해해보면서 스스로 다시 구현할 수 있었다고 생각한다. 테스트벤치 작성도 마찬가지로 스스로 구현할 수 있었다.

2. Lab1_2

1) 개요

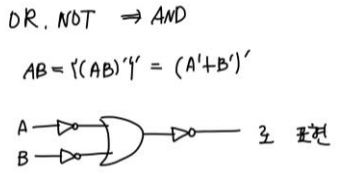
- functionally complete 집합을 구현한다. 즉, {and, or, not} 집합을 완성한다.
- Or과 Not을 통해 AND 게이트를 구현하고, and와 not을 통해 or을 구현한다.
- Nand나 nor만을 사용하여 and, or, not 게이트를 구현한다.
- Schematic의 회로를 통해 잘 구현되었는지 확인한다.

2) 이론적 배경

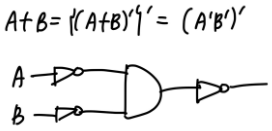
- Functionally complete set은 말그대로 집합으로, 그 집합 안에 속한 게이트들로만 Boolean Expression 표현이 가능한 집합을 말한다.
- 예로는 {AND, OR, NOT}, {AND, NOT}, {OR, NOT}, {NAND}, {NOR}이 존재한다.
- 여기서 functionally complete 함을 증명하기 위해서는 게이트들을 가지고 and, or, not이 만들어짐을 증명하면 된다.
- 특히, NAND와 NOR의 경우는 이 한 게이트만 가지고도 모든 디지털 시스템을 표현할 수 있다는 특징이 있다. 그래서 이들을 universal gate라고 부른다.

3) 실험 준비

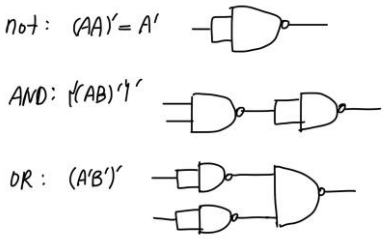
- And gate를 or 과 not gate로 구현할 때 어떻게 해야할 지에 대해 구상해본다.



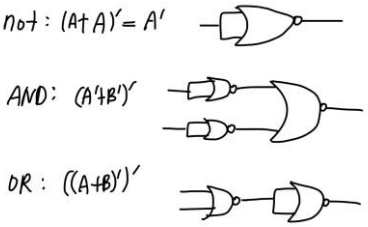
- Or gate를 and와 not gate로 구현하는 것을 구상해본다.



- And or not gate를 nand gate로 구현하는 것을 구상해본다.



- And or not gate를 nor gate로 구현하는 것을 구상해본다.

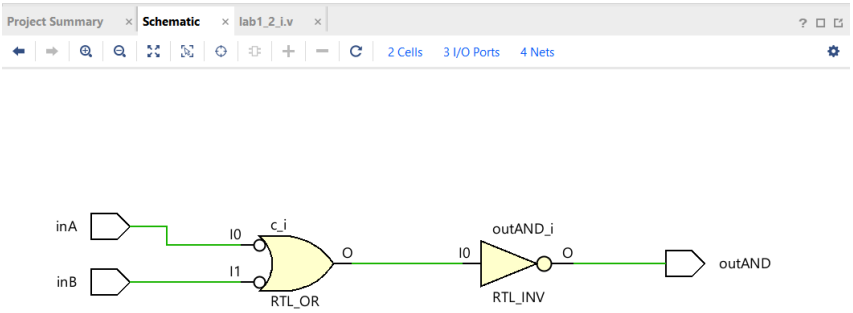


- Truth table들을 다 그려본다.

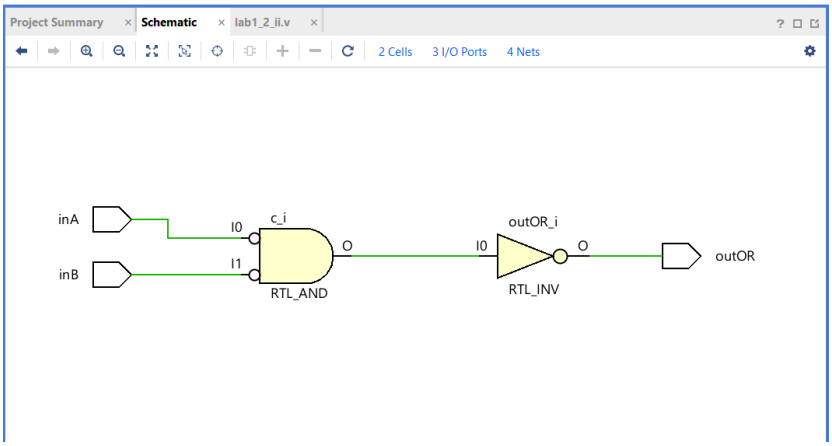
A	B	AND	OR	NOT(A)	NOT(B)	NAND	NOR
0	0	0	0	1	1	1	1
0	1	0	1	1	0	1	0
1	0	0	1	0	1	1	0
1	1	1	1	0	0	0	0

4) 결과

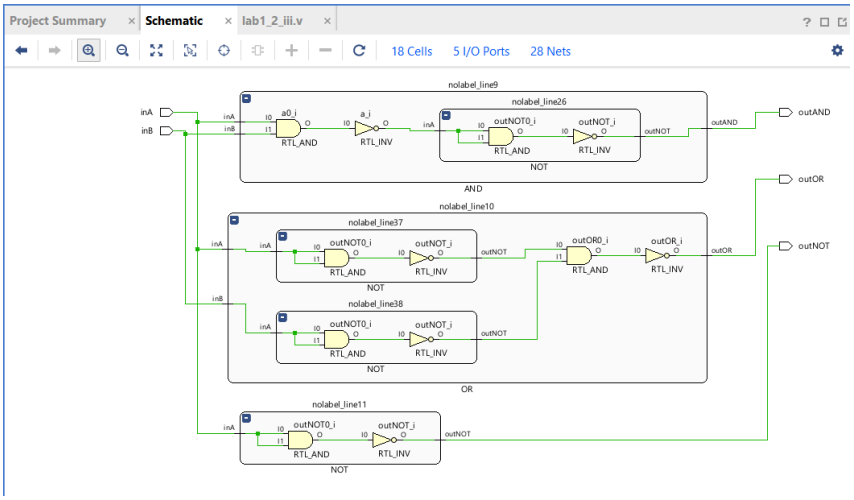
- And gate를 or과 not gate로 구현했을 때 나온 회로이다.



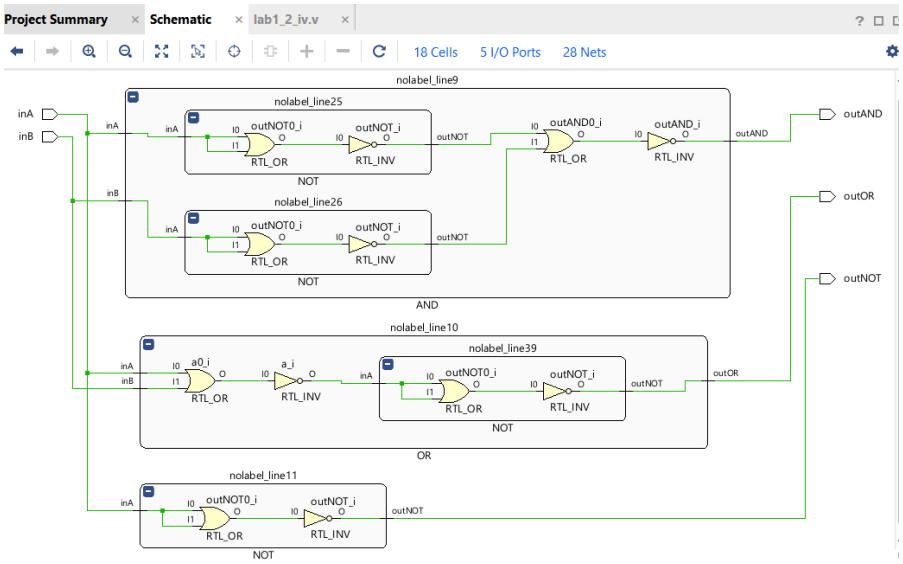
- or gate를 and과 not gate로 구현했을 때 나온 회로이다.



- And or not gate를 nand gate로 구현했을 때 나온 회로이다.



- And or not gate를 nor gate로 구현했을 때 나온 회로이다.



- 3)에서 내가 직접 구상한 회로도와 정확하게 회로도가 일치했고, 동작도 같았음을 이렇게 확인할 수 있다.

5) 논의

- And gate를 구현하는 것과 달리 and not으로 or을 만들거나, or not으로 and를 만드는 경우 처음에는 많이 막막했고 wire을 추가할 생각을 하지도 못했지만 다시 공부해보면서 wire을 추가하고 이 wire가 게이트들을 연결하는 수단으로 이해를 하면서 모든 4개의 실습을 다 완료할 수 있었다. 뿐만 아니라 NOTgate를 nor이나 nand gate로 만든 후 이를 사용하여 나머지 AND나 OR을 구상할 수 있었다. NOT gate를 사용하지 않는다면 조금은 더 복잡하게 NAND나 NOR을 사용하여 다시 구현을 해야 했을 것이다.