CSED273 Lab1 보고서

20210643 김현준

1. 개요

기초 논리 게이트(and, or, not, nand, nor)로 회로를 구성하여 이를 Verilog로 구현하고, inx Vivado HDL IDE를 이용하여 간단한 Verilog 프로그래밍을 통해 회로를 구현한다. Functionally complete 개념 이해를 위해 (and, not), (or, not), (nand), (nor)만을 이용하여 나머지 게이트를 구성하는 코드를 작성한다. 또한 and gate의 Testbench를 작성하여 조건에 따라서 만든 회로를 테스트한다.

2. 이론적 배경

1) Functionally complete set

Boolean algebra에서, 어떤 논리 연산의 집합으로 모든 것을 표현할 수 있을 때 이 집합이 Functionally complete하다고 말할 수 있다. AND, OR, NOT은 Functionally complete하며, 따라서 어떠한 논리 연산의 set이 AND, OR, NOT을 모두 표현 가능할 때, Functionally complete하다.

2) Universal set

NAND, NOR은 혼자서도 AND, OR, NOT 게이트를 모두 표현 가능하다. 따라서 NAND와 NOR은 그 자체로 Functionally complete하고, 이들을 Universal set라고 한다. 즉, NAND와 NOR은 그 자체의 조합만으로 모든 논리 회로를 표현할 수 있다.

3) HDL, Verilog

HDL(Hardware Description Language)은 논리 회로 구조를 표현하는 언어로, Verilog와 VHDL이 있다. 본 Lab에서 사용되는 Verilog는 하드웨어 동작을 표현하기 위한 언어로, C언어와 유사한 문법 구조를 가지고 있으며 다양한 방식(Behavioral lever, Register-Transfer Level, Gate Level)로 하드웨어를 설계할 수 있다. 본 Lab에서는 Gate Level Modeling을 이용한다. 이는 회로를 Gate끼리 연결해 표현하는 방식으로 Modeling하는 방식이다.

4) Positive Logic / Negative Logic

전자회로로 논리식을 구현할 때, Boolean algebra의 참과 거짓을 HIGH와 LOW에 대응시킬 수 있으며, HIGH와 1, LOW와 0을 대응시킨 것이 Positive Logic, 그 반대가 Negative Logic이다.

5) Testbench

회로에 어떤 신호를 입력할지 정하여 테스트하는 과정으로, Testbench는 작성한 Module을 불러와 적절하게 Input과 Output을 연결하여, 특정 시간에 따른 Input 변화에 Output이 어떻게 변하는지 확인한다.

3. 실험 준비

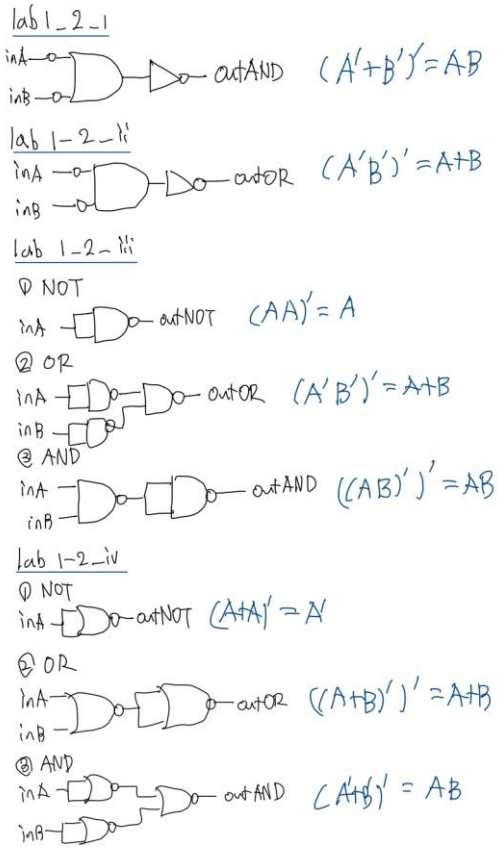
1) NOT, OR, AND, NOR, NAND 진리표

A	not A
0	1
1	0

Truth Table					
A	B	OR	AND	NOR	NAND
0	0	0	0	1	1
0	1	1	0	0	1
1	0	1	0	0	1
1	1	1	1	0	0

다음과 같이 Truth table을 그릴 수 있다.

2) 회로도 및 논리식



다음과 같이 구성하고자 한다. lab1_2_1i는 AND, NOT으로 OR을 구현하고, lab1_2_ii는 OR, NOT으로 AND를 구현한다. Lab1_2_iii는 NAND로 AND, OR, NOT을, lab1_2_iv는 NOR로 AND, OR, NOT을 구현한다. 위와 같이 회로를 구성하면, 원하는 결과를 얻을 수 있음을 논리식으로 확인할 수 있다.

3) lab0 및 Orientation 복습

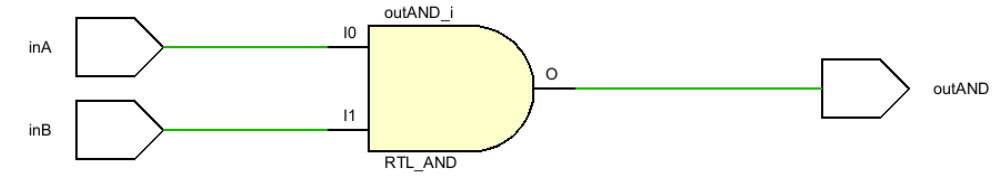
Vivado 통합개발환경을 잘 다루고 Verilog 프로그램 작성을 위해 lab0과 Orientation을 간략하게 다시 살펴보았다.

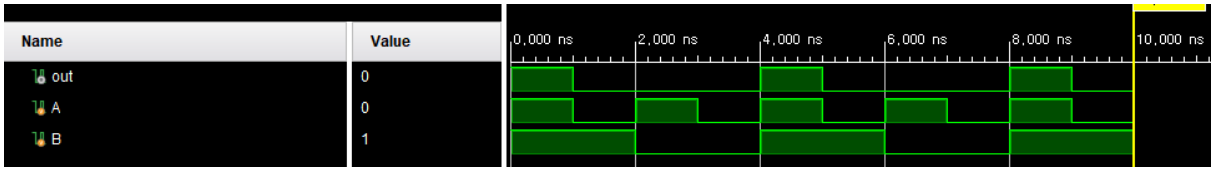
4. 결과

Verilog를 이용하여 각 게이트를 구현하였고, 이때 다른 방식이 아닌 Gate-Level Modeling을 통해 각 게이트를 구현하였다.

1) AND 게이트 구현

Verilog로 AND 게이트를 구현하였다. Testbench를 작성하고 시뮬레이션을 수행한 결과 아래와 같았다. 이때 초기조건은 A=1, B=1및 A는 1ns마다 반전, B는 2ns마다 반전, 총 10ns까지 진행하였다.





Synthesis 결과를 확인하면 제대로 AND 게이트가 형성되었음을 알 수 있고, simulation 결과 A와 B가 1일 때만 OUT이 1이 되는 것을 알 수 있다. 즉, 위의 진리표의 AND와 정확이 같은 기능을 하고, AND 게이트가 제대로 구성되었음을 알 수 있다.

2) Functionally complete 집합 구현

(1) OR, NOT

(3. 실험 준비)의 논리식처럼, OR과 NOT 게이트를 이용하여 AND 게이트를 구현하려면 드 모르간의 법칙에서 (A'+B')'=AB로 구현할 수가 있다. 이를 Verilog로 구현하면 아래와 같다.

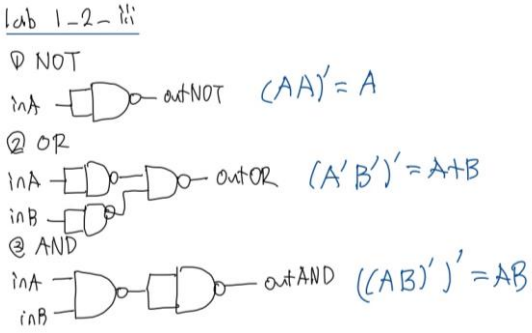
논리식처럼, 입력 A, B를 NOT 게이트에 통과시킨 다음, 이 두 신호를 OR 게이트로 통과시키고 마지막으로 NOT 게이트로 통과시키는 결과를 얻을 수 있었다. (3. 실험 준비) 과정에서 구상한 대로 구현되는 것을 확인할 수 있었다.

(2) AND, NOT

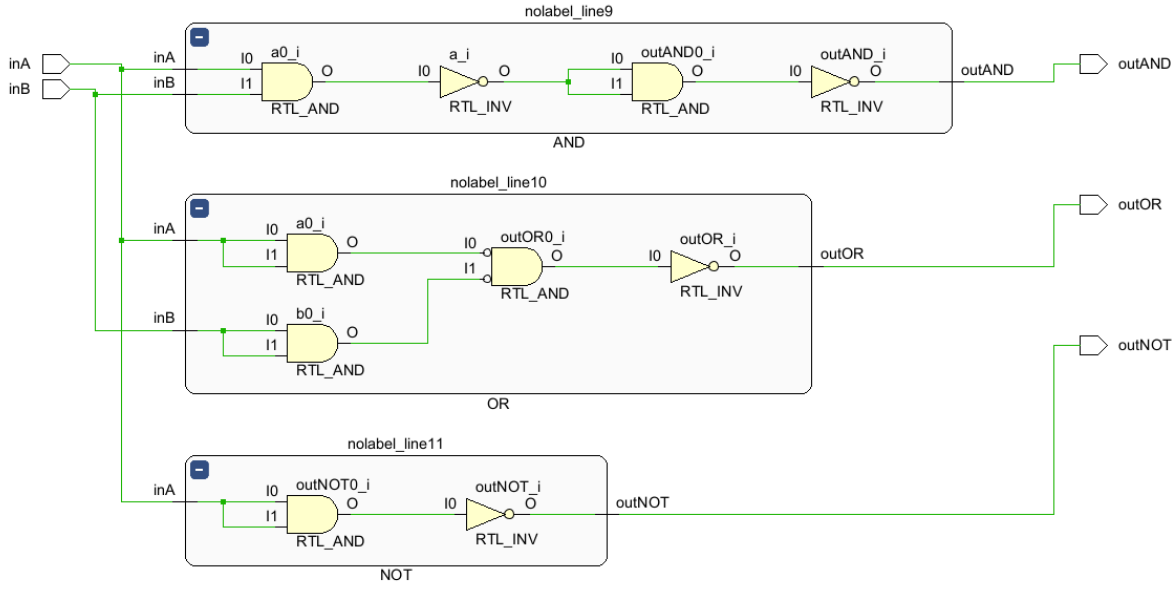
(1)과 마찬가지로 (2)는 논리식 (A'B')'=A+B로 구현할 수 있고, Verilog로 구현한 결과 아래와 같은 Synthesis 결과 그림을 얻었다.

논리식과 같이 입력 A, B를 각각 NOT 게이트에 통과하고, OR 게이트를 통과한 후 NOT 게이트에 통과시킨 형태로 잘 구현되었다. (3. 실험 준비) 과정에서 구상한 그대로 구현되는 것을 확인할 수 있었다.

(3) NAND

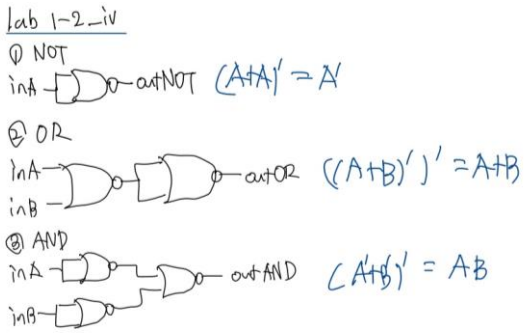


위와 같이 (3. 실험 준비) 과정에서 작성한 회로도와 논리식을 Verilog를 통해 구현한 결과 아래의 결과를 얻을 수 있었다.

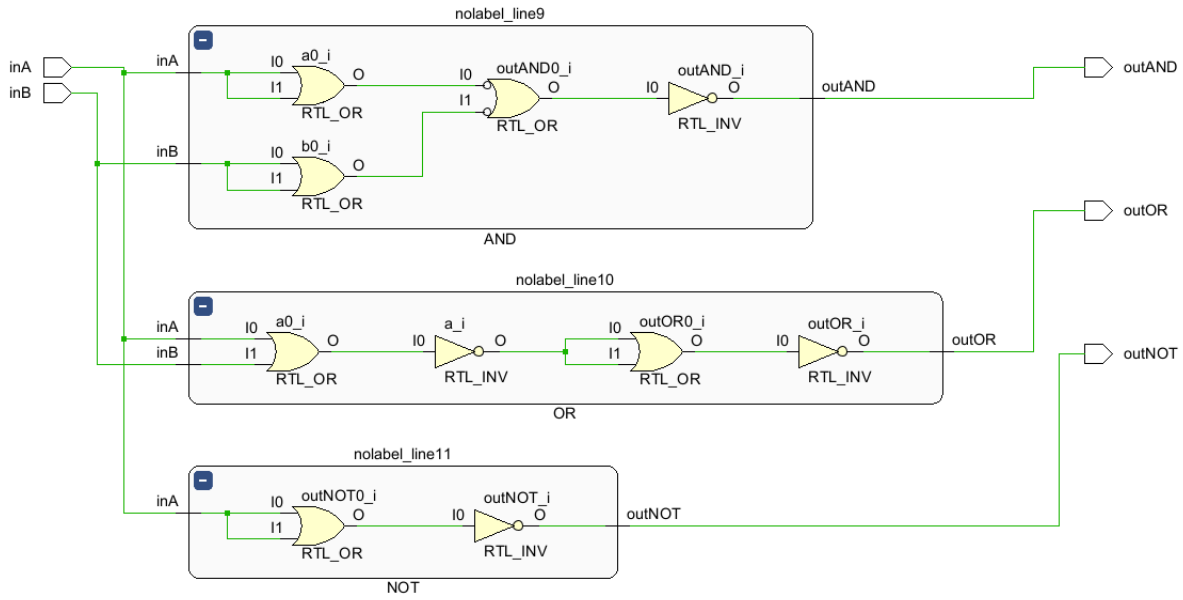


우선 NOT 게이트는 NAND 게이트 하나로 (AA)'=A와 같이 구현할 수 있었고, OR 게이트는 (2)에서 한 것처럼 먼저 만든 NOT게이트 두 개를 통과한 입력을 다시 한 번 NAND 게이트로 통과시켜 구현할 수 있었다. AND 게이트의 경우는 ((AB)')'=AB의 과정으로, NAND게이트를 먼저 만든 NOT 게이트로 통과시킨 형태의 구성으로 만들 수 있었다. 결과적으로 세 게이트 모두, (3. 실험 준비) 단계에서 구상한 대로 잘 구현할 수 있었다.

(4) NOR



위와 같이 (3. 실험 준비) 과정에서 작성한 회로도와 논리식을 Verilog를 통해 구현한 결과 아래의 결과를 얻을 수 있었다.



우선 NOT 게이트는 (A+A)'=A'의 논리식처럼, NOR 게이트 하나로 구현할 수 있었다. 다음으로 OR 게이트는 ((A+B)')'로 NOR 게이트에 전에 만든 NOT 게이트 형태를 연결한 방식으로 구현할 수 있었다. 마지막으로 AND 게이트는 (1)에서 한 것과 같은 형태로, 이미 만든 OR 게이트와 NOT 게이트를 연결한 형태로 구현할 수 있었다. 결과적으로, (3. 실험 준비) 단계에서 구상한 대로 모두 제대로 구현되는 것을 확인할 수 있었다.

논의

구상한 대로 모두 결과를 얻을 수 있었다. 다만, Vivado 환경을 주도적으로 다뤄보는 것이 처음이라 다소 어려움을 겪어 lab0 영상을 다시 복습하면서 공부하였다. lab1을 통해 전체적으로 Verilog를 다뤄볼 수 있었고, testbench 실행하면서 Vivado 사용법을 익힐 수 있었다. 또한, Functionally complete 개념을 직접 확인해보면서 개념을 확고히 할 수 있었다.