Lab 1. 논리회로 기초

2023. 03. 21.

김지현 (20220302)

1. 개요

Lab 1 은 Vivado 프로그램을 통해 Verilog 라는 HDL (hardware description language) 를 배우게 된다. 회로 코드 작성법과 작성한 코드의 시뮬레이션을 돌릴 수 있게 하는 test bench 작성법도 숙지하여 결과적으로 파형과 회로도를 만들어내는 것을 목표로 한다. 이번 lab 1 에서는 and, or, not gate 를 구현하는 방법과 nand 와 nor gate 가 functionally complete 한지에 대해 알아본다.

2. 이론적 배경

Verilog 를 이용해 회로도를 구현하는 방법에는 크게 두가지 모델링(modeling)이 있다. Behavioral modeling 은 보통 test bench 코드를 나타내는데에 사용되게 되는데 하드웨어가 실행해야 하는 기능을 구현하지만 반드시 합성이 가능하다고는 할 수 없다. Behavioral modeling 구조를 예를 들자면, assign 이나 always @ 등의 블락을 이용해 코드를 작성하게 된다. Gate-level modeling 은 모듈과 게이트 들로만 설명되는 구조이다. Gate-level modeling 은 and, or, not 등의 게이트 구조를 사용해 실제 게이트를 나타낸다. 이번 lab 1 과제에서는 gate-level modeling 을 사용하여 개요에서 설명된 논리 회로들을 구현할 것이다.

먼저, 논리 회로 (logic gate) 란 디지털 회로의 building block 처럼 사용된다. 디지털 신호 input 과 output 은 Boolean algebra (부울 대수)를 기반으로 true(1) 와 false(0) 로 나타난다. 그리고 사용되는 logic gate 의 종류와 입력값에 따라 출력값이 변화하게 된다. 예를 들면 두 입력값이 true 를 가지고 and gate 를 통과한다면 출력값 역시 true 로 나타날 것이다.

위의 논리 회로를 digital electronic system 으로 구현했을 때, 높은 전압/전류가 1 (참)을 나타내고 낮은 전압/전류가 0 (거짓)을 나타낸다면 positive logic 시스템이라고 한다. 반대로, negative logic 시스템의 경우, 높은 전압/전류가 0 (거짓)을 나타내고 낮은

전압/전류가 1 (참)을 나타낸다. 위의 내용을 정리 하자면 positive logic 또는 active-high 일 경우 higher electric signal 이 True 를 나타내고 negative logic 또는 active-low 에서는 lower electric signal 이 True 를 나타내게 된다. 시스템의 결과에 따라 디지털 논리 회로는 positive logic 과 negative logic 으로 분류되는 것이기 때문에 두 회로는 같은 기능을 한다. 하지만 실제로 설계를 할 때는 두가지 논리 중 하나를 정해서 설계를 하는 것이 중요하다.

개요에서 언급한 Verilog 언어를 이용해 위의 회로를 구현하기 때문에 lab 을 진행하기 위해서는 Verilog 의 이해를 필요로 한다. Verilog 은 전자회로의 동작을 기술하는 하드웨어 모델링 언어이고 c 언어와 비슷하게 구현할 수 있어서 비교적 배우기 쉽다는 장점을 가지고 있다.

실험을 진행하기 전 준비 과정에서 NOT, OR, AND, NOR, 그리고 NAND 의 진리표를 작성하게 된다. 진리표란 logic gate function 의 Boolean expression 을 표현한 테이블로, 각 게이트 또는 회로에 대한 가능한 input 조합에 따른 출력값을 나타낸다. 이를 이용해서 작성한 코드의 회로도 또는 파형을 보고 올바른 회로도가 작성되었는지 확인할 수 있다.

개요에서 언급된 목표중 하나는 NOR 과 NAND gate 가 functionally complete 한지 확인하는 것이다. Functionally complete 하기 위해서는 NAND 또는 NOR gate 의 조합을 이용해서 AND/OR/NOT gate 모두 표현할 수 있어야 한다.

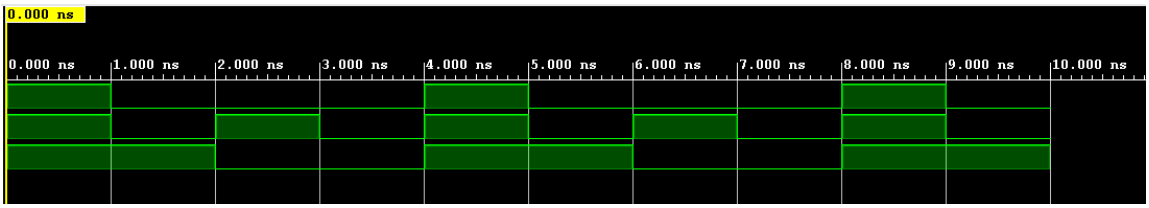
3. 실험 준비

위에서 설명된 gate-level modeling 을 이용해서 회로를 작성할 때는 5 가지 수식 또는 gate 을 사용하게 된다. Lab 1_1 의 경우 and gate 를 사용하고, lab 1_2 부터는 and, or, nand, 그리고 nor gate 를 이용해 functional completeness 를 보이는 설계를 구현한다.

NOT/OR/AND/NOR/NAND 연산 진리표:

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

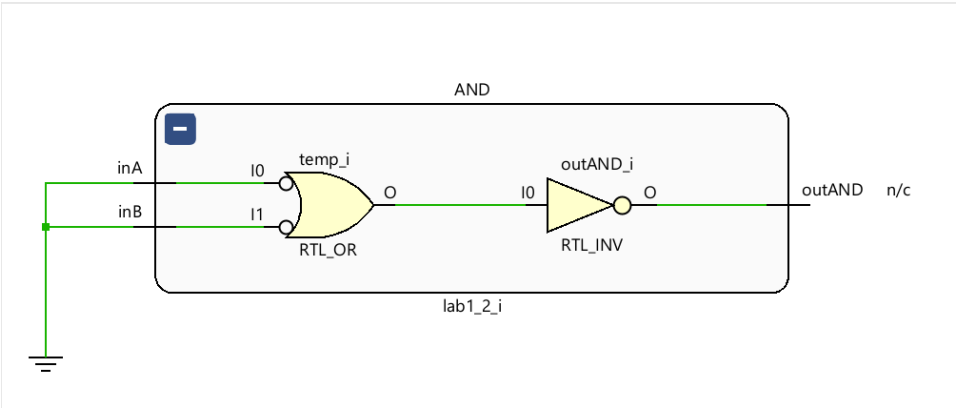
4. 결과



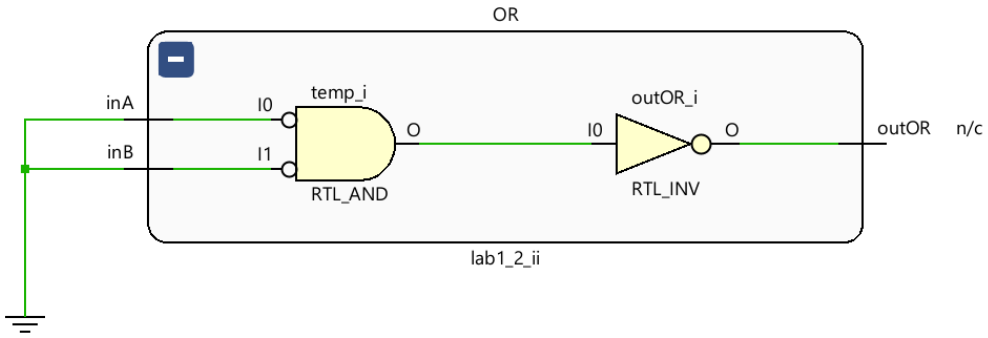
위의 사진은 Lab 1_1 에서 나타난 파형을 캡쳐를 한것이다. 위에서부터 output, input A, input B 를 각각 나타낸 것이고 true(1) 인 경우 초록색으로 표시되어있다. Lab1_1 에서는 and gate 수식인 and(output, input, input)을 사용해 10ns 간의 파형을 살표보았다. 두 입력값 A 와 B 는 시작할 때, 즉 0ns 일때, true (1)으로 시작해서 A 는 1ns 마다 Boolean expression 이 바뀌고 B 는 2ns 마다 Boolean expression 이 바뀌게 된다. 위의 파형을 table 로 나타내면 아래와 같다.

Input/outpour	0ns ~	1ns ~	2ns ~	3ns ~	4ns ~	5ns ~	6ns ~	7ns ~	8ns ~	9ns ~
Input A	1	0	1	0	1	0	1	0	1	0
Input B	1	1	0	0	1	1	0	0	1	1
Output	1	0	0	0	1	0	0	0	1	0

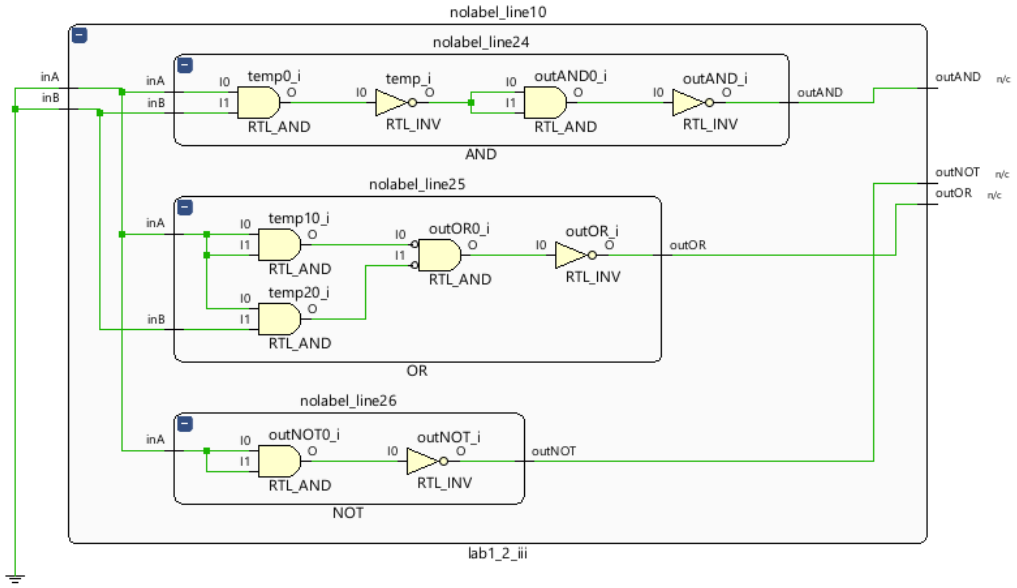
And gate 는 두 입력값이 모두 true(1) 일 때 true 출력값을 나타내기 때문에 회로도가 정확한 결과를 도출한다고 할 수 있다.



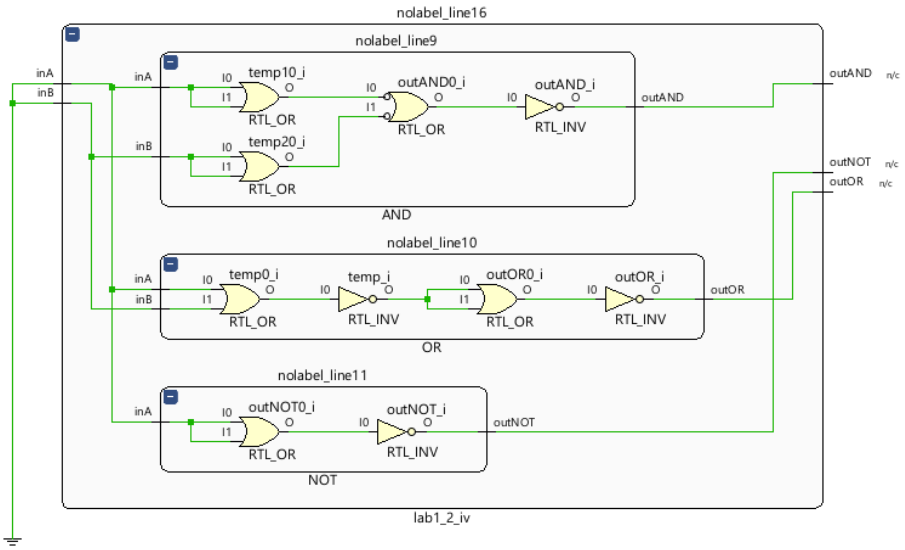
위의 회로도는 lab 1_2_i 에서 진행한 or, not gate 를 이용해 and gate 를 나타낸 것이다. 두 입력값 A,B 를 이용한 회로도를 식으로 나타내자면 De Morgan's law 를 이용했을 때, ~(A*B) = ~A + ~B 이기 때문에 A*B = ~(~A + ~B)) 라고 할 수 있다. 이 식을 회로도로 나타내면 위와 같다. 두 입력 값 A,B 가 각각 not gate 를 지난 후 + 를 의미하는 or gate 를 통과하고, 마지막으로 not gate 를 지나게 된다. 이는 앞서 말했던 식과 동일하기 때문에 올바른 구현이다.



위의 회로도는 lab 1_2_ii 에서 진행한 and, not gate 를 이용해 or gate 를 나타낸 것이다. 두 입력값 A,B 를 이용한 회로도를 식으로 나타내자면 De Morgan's law 를 이용했을 때, ~(A + B) = ~A * ~B 이기 때문에 A+B = ~(~A * ~B)) 라고 할 수 있다. 이 식을 회로도로 나타내면 위와 같다. 두 입력 값 A,B 가 각각 not gate 를 지난 후 * 를 의미하는 and gate 를 통과하고, 마지막으로 not gate 를 지나게 된다. 이는 앞서 말했던 식과 동일하기 때문에 올바른 구현이다.



위의 회로도는 lab 1_2_iii 에서 진행한 nand gate 만을 이용해 AND, OR, NOT 연산을 수행시키는 것을 보인다. 이는 nand gate 가 functionally complete 하다는 것을 보이는 것과 같다. 먼저 AND 연산을 하기 위해서 wire temp 라는 임시 와이어를 하나 만든 뒤 두 입력값 A 와 B 가 nand gate 를 지난 후 의 값을 가지도록 했다. 그리고 temp 값 두개가 nand gate 를 지났을 때의 결과값을 출력값인 outAND 에 저장하도록 했다. 이와 같은 방식으로 OR 연산은 입력값인 A 와 B 각각을 nand gate 를 지나도록 한 뒤의 출력값을 각각 wire temp1, temp2 에 저장하고 wire temp1, temp2 가 nand gate 를 지난 후의 출력 값을 outOR 에 저장하도록 했다. 마지막으로 NOT 연산은 입력값은 하나로 가정하고 nand gate 를 지난 출력값을 outNOT 에 저장했다. 이는 위의 회로도와 앞서 설명한 식과 동일하기 때문에 올바른 구현이다.



위의 회로도는 lab 1_2_iv 에서 진행한 nor gate 만을 이용해 AND, OR, NOT 연산을 수행시키는 것을 보인다. 이는 nor gate 가 functionally complete 하다는 것을 보이는 것과 같다. 먼저 OR 연산을 하기 위해서 wire temp 라는 임시 와이어를 하나 만든 뒤 두 입력값 A 와 B 가 nor gate 를 지난 후 의 값을 가지도록 했다. 그리고 temp 값 두개가 nor gate 를 지났을 때의 결과값을 출력값인 outAND 에 저장하도록 했다. 이와 같은 방식으로 AND 연산은 입력값인 A 와 B 각각을 nor gate 를 지나도록 한 뒤의 출력값을 각각 wire temp1, temp2 에 저장하고 wire temp1, temp2 가 nor gate 를 지난 후의 출력 값을 outOR 에 저장하도록 했다. 마지막으로 NOT 연산은 입력값은 하나로 가정하고 nor gate 를 지난 출력값을 outNOT 에 저장했다. 이는 위의 회로도와 앞서 설명한 식과 동일하기 때문에 올바른 구현이다.

5. 논의

이번 실험을 진행하면서 Verilog 언어에 대해 배울 수 있었다. C 언어와 비슷한 점도 많지만 이론에서 언급했다시피 gate-level modeling 과 behavior modeling 이 있어서 둘의 차이점에 대해서 찾아보는 좋은 기회가 되었다. 하지만 실험을 진행하는 도중 Vivado 를 프로그램에서 시뮬레이션을 돌리면서 잦은 오류를 확인하기도 했다. 특히 lab1_2_iii 와 lab1_2_iv 의 test bench 를 만들어서 확인하는 과정에서 test bench 파일에 lab1_2_iii AND (outAND, A, B); lab1_2_iii OR (outOR, A, B); lab1_2_iii NOT (outNOT, A, B); 를 적으면 오류가 생겨서 결국 lab1_2_iii (outAND, outOR, outNOT, A,

B); 를 적어서 시뮬레이션을 돌렸다. 그 결과 schematic 회로도에서 AND/OR/NOT 대신 nolabel_line# 형식으로 나타나게 되었다. 구조상 문제가 생긴 것은 아니였지만 Verilog 언어에 대한 이해도를 높여 nolabel_line# 가 아닌 AND/OR/NOT 으로 나타낼 수 있게 만든다면 더 보기 쉬운 회로도를 구현해낼 수 있을 것이다.