Lab 1. 논리회로 기초

Digital System Design (CSED273)

20220100 박기현

1. 개요

1) AND Gate 구현

AND Gate 를 Verilog 로 구현하고, Testbench 를 작성해 시뮬레이션을 수행한다. 이때 모듈은 Gata-Level Modeling 으로 구현하며, 시뮬레이션은 총 10ns 동안 수행한다. AND Gate 의 두 입력은 각각 1ns 와 2ns 마다 한 번씩 반전되며, 0ns 일 때 입력의 초기값은 모두 1 이다.

2) Functionally complete 집합 구현

{OR, NOT}, {AND, NOT}, {NAND}, {NOR}, 총 네 가지의 연산을 이용하여 {AND, OR, NOT} 집합을 완성하고, functionally complete set 에 대해 학습한다.

2. 이론적 배경

1) Positive Logic / Negative Logic

전기 신호에는 고전압 상태와 저전압 상태가 존재한다. 논리식을 전자 회로로 구현할 때 부울 대수의 참과 거짓을 고전압 상태와 저전압 상태에 임의로 대응할 수 있다.

Positive Logic 의 경우에는 고전압 상태를 참으로, 저전압 상태를 거짓으로 설정한 것이고, Negative Logic 의 경우에는 반대로 저전압 상태를 참으로, 고전압 상태를 거짓으로 설정한 것이다.

이는 논리 회로를 전기적으로 표현하는 방식만 다를 뿐, 기능에는 차이가 존재하지 않는다. 단, 실제 부품을 사용하여 회로를 구성할 때는 Active-high 와 Active-low 부품이 섞여 있을 수 있으므로 주의해야 한다.

같은 회로에 대하여 Positive logic 에 따랐을 때 함수 F 라고 한다면, Negative logic 에 따랐을 때는 dual 함수 F^D 이다.

A	B	F
0	0	0

0	+V	0
+V	0	0
+V	+V	+V

(1) Positive logic → AND function 과 같다.

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

(2) Negative logic → OR function 과 같다.

A	B	F
1	1	1
1	0	1
0	1	1
0	0	0

2) HDL (Hardware Description Language)

HDL (Hardware Description Language)은 디지털 시스템의 논리 회로 구조를 표현하기 위한 언어이다. 주로 사용하는 언어로는 Verilog, VHDL, Systemverilog 등이 있다. 본 디지털 시스템 설계 과목에서는 Verilog 를 다룬다.

3) Functionally complete set

어떤 논리 연산의 집합으로 모든 부울 대수식을 표현할 수 있다면, 그 집합은 functionally complete 이다. 특정 집합이 functionally complete set 임을 증명하기 위해서는, 포함된 연산을 조합하여 AND, OR, NOT 연산을 만들어낼 수 있음을 보이면 된다.

3. 실험 준비

1) Vivado 사용법과 Verilog 문법, Testbench 작성 방법을 숙지한다.

- Vivado

Xilinx Vivado 는 FPGA 를 설계하기 위해 사용하는 도구이다. HDL 코드를 이용하여 코드 시뮬레이션, 회로도 보기, 합성 후 FPGA 에 쓰기 등이 가능하다.

[File -> Project -> New...]를 통해 Project 를 생성한다. (이때, Project Type 은 RTL Project, Target Language 는 Verilog, Simulator Language 는 Mixed, Default Part 는 xc7a35tcpg236-1 로 설정한다.) Project 를 생성한 이후에는 [Add Sources -> Add or Create Design Sources -> Create File]을 통해 Verilog 타입으로 Design Source 파일을 생성한다. Simulation Source 파일은 [Add Sources -> Add or Create Simulation Sources -> Create File]을 통해 생성한다.

[Run Simulation -> Run Behavioral Simulation]으로 시뮬레이션 수행이 가능하며, 하단의 Tcl Console 을 통해 Pass / Fail 여부 출력 확인이 가능하다. Simulation Runtime 을 설정하면, 시뮬레이션 진행 시간을 제한하여 시뮬레이션 속도를 높일 수 있다.

[Open Elaborated Design -> Schematic]으로 회로도 보기가 가능하다.

- Verilog

Verilog 는 HDL 중 하나로, 하드웨어 동작을 표현하기 위한 언어이다. Behaviorl Level, Register-Transfer Level, Gate Level 등 다양한 방식으로 하드웨어를 설계할 수 있다. 본 실습에서는 Gate-Level Modeling 으로 진행한다.

코드 내 시간 단위와 정밀도를 지정할 수 있으며, *'timescale 1ns / 1ps* 로 선언할 경우, 코드 내 시간 단위는 1ns, 최대 정밀도는 1ps 이다.

Module 은 객체지향프로그래밍에서의 객체 개념과 유사하며, module 안에 다른 module instance 를 생성하는 것도 가능하다.

각 Port 는 반점으로 구분하고, *input*, *output* 키워드로 선언하며, 같은 종류의 키워드를 한 번에 선언할 수 있다. data type 과 함께 지정하는 것도 가능하며, data type 을 따로 지정하지 않으면 기본값은 wire 이다. 이는 후에 contents 를 작성할 때 바꿀 수 있다.

Gate-Level Modeling 은 회로를 gate 끼리 연결하여 표현한다. 코드 형태는 *[Gate]([output],[input],[intput])* 와 같다.

- Testbench

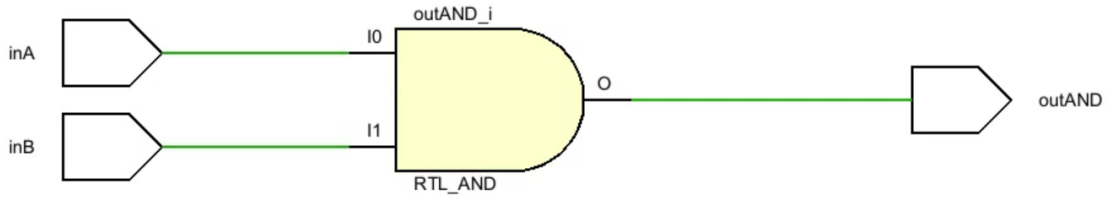
Testbench 는 일반적인 module 과 다르게 합성이 필요하지 않으며, Verilog Module 테스트를 목적으로 한다. 작성한 module 을 불러와 적절하게 I/O 를 연결하고, module 에 넘겨준 input 과 output 의 시간에 따른 state 변화를 확인한다.

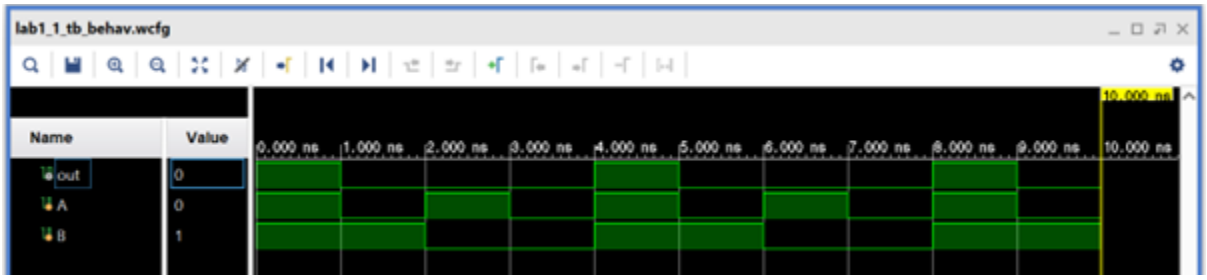
2) NOT, OR, AND, NOR, NAND, 총 다섯 개의 연산 진리표를 작성한다.

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

4. 결과

1) AND Gate 구현





총 10ns 동안 시뮬레이션을 수행하였다.

0ns 일 때, AND gate 의 두 입력의 초기값은 모두 1 이고, 각각 1ns 와 2ns 마다 한 번씩 반전된다.

AND gate 이므로, 두 입력이 모두 1 일 때, 출력값이 1 이 된다.

따라서 A 의 입력값이 반전되기 전인 0ns~1ns 에서 출력값이 1 이 되고, 그후 두 입력이 모두 1 이 되는 4ns~5ns 와 8ns~9ns 에서 출력값이 1 이 된다.

2) Functionally complete 집합 구현

i)

{OR, NOT} 집합은 functionally complete 집합이다.

{(A'+B')}'=AB 를 이용하여 AND gate 를 구현할 수 있다.

따라서 wire C, D, E 를 선언하고, not(C, inA)로 A'을, not(D, inB)로 B'을 구현한 뒤, or(E, C, D), not(outAND, E)를 통해 AB 를 구현한다.

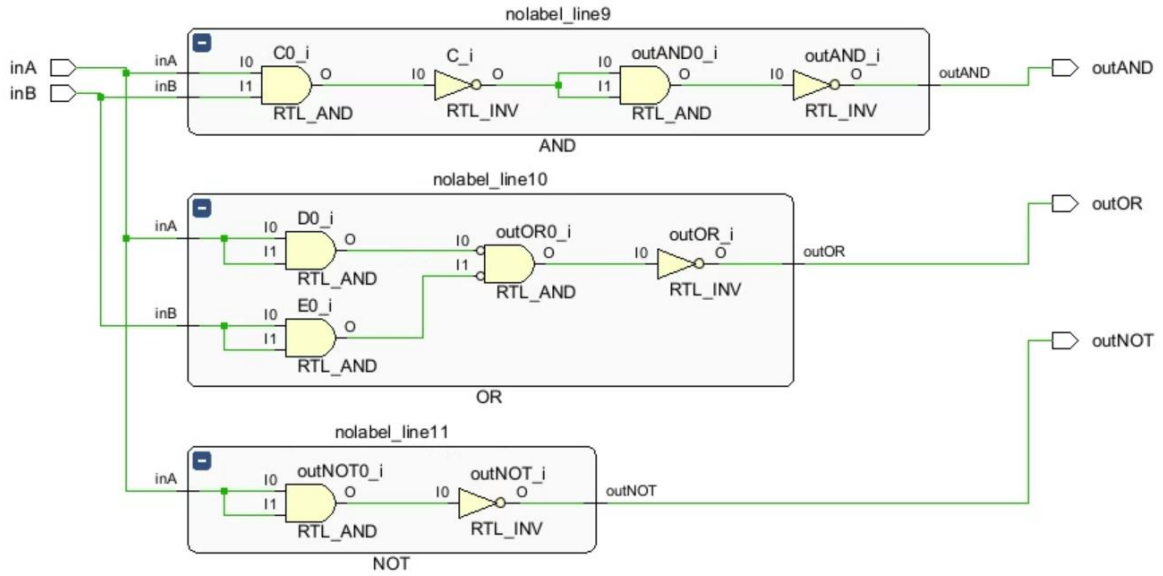
ii)

{AND, NOT} 집합은 functionally complete 집합이다.

(A'B')'=A+B 를 이용하여 OR gate 를 구현할 수 있다.

따라서 wire C, D, E 를 선언하고, not(C, inA)로 A'을, not(D, inB)로 B'을 구현한 뒤, and(E, C, D), not(outOR, E)를 통해 A+B 를 구현한다.

iii)



{NAND} 집합은 functionally complete 집합이다.

(1) {(AB)′(AB)′}′=AB 를 이용하여 AND gate 를 구현할 수 있다.

(AB)′=A′+B′ / {(AB)′(AB)′}′={(A′+B′)(A′+B′)}′=(A′+B′)′+(A′+B′)′=AB+AB=AB

따라서 wire C 를 선언하고, nand(C, inA, inB)로 A′+B′을 구현한 뒤, nand(outAND, C, C)를 통해 AB 를 구현한다.

(2) {(AA)′(BB)′}′=A+B 를 이용하여 OR gate 를 구현할 수 있다.

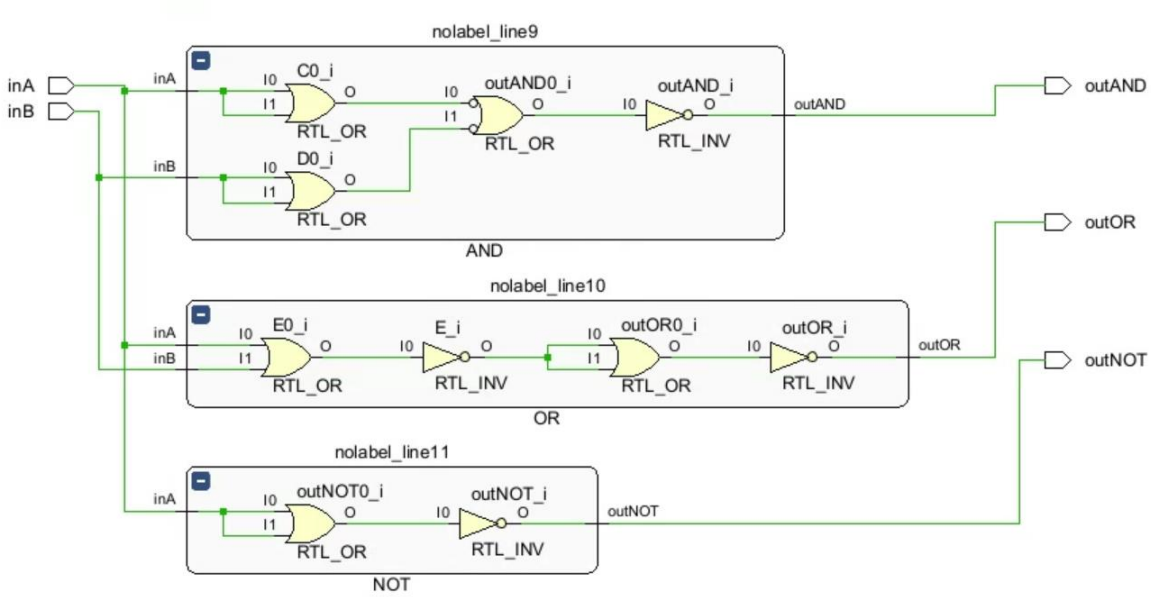
(AA)′=A′, (BB)′=B′ / {(AA)′(BB)′}′=(A′B′)′=A+B

따라서 wire D, E 를 선언하고, nand(D, inA, inA)로 A′을, nand(E, inB, inB)로 B′을 구현한 뒤, nand(outOR, D, E)를 통해 A+B 를 구현한다.

(3) (AA)′=A′을 이용하여 NOT gate 를 구현할 수 있다.

따라서 nand(outNOT, inA, inA)를 통해 A′을 구현한다.

iv)



{NOR} 집합은 functionally complete 집합이다.

(1) {(A+A)'+(B+B)'}'=AB 를 이용하여 AND gate 를 구현할 수 있다.

(A+A)'=A', (B+B)'=B' / {(A+A)'+(B+B)'}'=(A'+B')'=AB

따라서 wire C, D 를 선언하고, nor(C, inA, inA)로 A'을, nor(D, inB, inB)로 B'을 구현한 뒤, not(outAND, C, D)를 통해 AB 를 구현한다.

(2) {(A+B)'+(A+B)'}'=A+B 를 이용하여 OR gate 를 구현할 수 있다.

(A+B)'=A'B' / {(A+B)'+(A+B)'}'=(A'B'+A'B')'=(A'B')'=A+B

따라서 wire E 를 선언하고, nor(E, inA, inB)로 A'B'을 구현한 뒤, nor(outOR, E, E)를 통해 A+B 를 구현한다.

(3) (A+A)'=A'을 이용하여 NOT gate 를 구현할 수 있다.

따라서 nor(outNOT, inA, inA)를 통해 A'을 구현한다.

5. 논의

1) 느낀 점

프로그래밍 언어로는 C, C++, Python 밖에 접해보지 않았었는데, 이번 실습을 통해 HDL 언어인 Verilog 에 대해 알 수 있었다. Gate-level modeling 은 문법이 어렵지 않아 쉽게 gate 를 구현할 수 있었고, Vivado 프로그램 안에서 구현한 gate 코드를 회로도로 볼 수 있었다는 점이 인상깊었다.

수업시간에 배운 positive logic, negative logic과 functionally complete를 직접 적용해볼 수 있어서 그냥 글로써 복습하는 것보다 더 이해하기 수월했다. 또한, Schematic 기능을 활용한다면 회로도를 그리는데 있어 큰 도움이 될 것이라 생각했다.

2) 어려웠던 점 및 해결 방법

시뮬레이션을 수행하고자 했을 때, 에러가 발생하였다. 정확한 원인을 파악하진 못했으나, 한 프로젝트에 design source code를 한꺼번에 넣었던 것이 문제가 되었던 것 같다. 시뮬레이션 코드 파일인 lab1_1_tb.v만 따로 넣어 실행하니 정상적으로 시뮬레이션이 수행되었다.

코드에서 설정한 10ns 만큼의 시뮬레이션을 보고자 하였으나, 화면에는 원하지 않는 범위의 시뮬레이션이 띄워져 있었다. lab1 실습 시간의 수업을 통해 시뮬레이션 구간을 조정하는 방법을 배울 수 있었고, 이를 통해 원하는 10ns 구간을 확인할 수 있었다.

이번 실습에서 가장 어려웠던 점과 미처 해결하지 못했던 점은 Schematic 기능을 활용할 때, Vivado 프로그램이 비정상적으로 종료되는 현상이었다. 아무 오류 메시지 없이 종료되어 원인을 찾을 수 없었고, Vivado 프로그램을 다시 지웠다 깔아보아도 이 문제는 해결되지 않았다. 하지만 다행히 조교님의 도움을 받아 Schematic 기능을 통한 회로도를 볼 수 있었고, 무사히 보고서 작성을 마무리할 수 있었다.

Schematic 기능에서의 강제 종료 현상은 스스로 생각하기에 여러 이유가 있을 것으로 추정된다. 프로젝트 경로의 한글이 들어가지 않는지도 확인하였고, 프로그램이 정상적으로 설치되었는지도 확인하였으나 해결이 되지 않았던 것으로 보아, 가장 큰 의심점은 노트북 자체의 문제라고 판단된다. 하지만 이조차도 정말 다양한 원인이 있을 수 있기에 문제를 해결하는 데에는 큰 시간과 노력이 들 것으로 생각된다.