REPORT

보고서 작성 서약서

1. 나는 타학생의 보고서를 복사(Copy)하지 않았습니다.
2. 나는 타학생의 보고서를 인터넷에서 다운로드 하여 대체하지 않았습니다.
3. 나는 타인에게 보고서 제출 전에 보고서를 보여주지 않았습니다.
4. 보고서 제출 기한을 준수하였습니다.

나는 보고서 작성시 위법 행위를 하지 않고,
성.균.인으로서 나의 명예를 지킬 것을 약속합니다.

과 목 : 논리회로설계실험
과 제 명 : Design and Implement XOR Gate
담당교수 : 오 윤 호 교수님
학 과 : 전자 전기 공학부
학 년 : 3 학 년
학 번 : 2 0 1 6 3 1 1 2 9 0
이 름 : 이 지 학
제 출 일 : 2021. 03. 13

XOR Gate 설계

*이지학

*성균관대학교 전자전기공학부 dlwlgkr1@g.skku.edu

Design and Implement XOR Gate

*Ji-Hak Lee

*Dept. of Electronic & Electrical Engineering, Sungkyunkwan University

목 차

1. 요 약 [Brief]

XOR Gate 설계를 통하여 Verilog에서 사용되는 Operator의 활용에 대해 공부하고, 이를 XOR Gate Design에 적용하여 본다.

Verilog Modeling 뿐만 아니라, H/W Design에 전반적으로 사용되는 세 가지의 Modeling 기법 (1. Dataflow Modeling 2. Behavioral Modeling 3. Gate-Level Modeling)의 차이점과 장단점을 이해하고, 각 Modeling 기법을 구분해 설계에 적용하여 본다.

Modeling을 하기에 앞서, 해당 Gate의 Truth Table, Boolean Expression 등을 이론적으로 점검하고 XOR Gate 설계에 어떻게 적용되어 지는지 탐구한다.

2. 이론적 접근 [Theoretical approach]

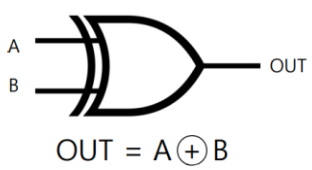
Boolean expression

- 설계 조건 :

(1) Boolean expression must consist of AND, OR, and NOT only.

(2) Using the parentheses is allowed

[XOR Gate]



$0 \oplus 0 = 0$
$0 \oplus 1 = 1$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$

$$A \oplus B = \bar{A} \cdot B + A \cdot \bar{B}$$

A와 B가 같은 경우 OUT 으로 False (0)를 반환한다.
A와 B가 다른 경우 OUT 으로 True (1)를 반환한다.

Truth table

A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	0

$\overline{0}\cdot 1 + 0\cdot\overline{1} = 1\cdot 1 + 0\cdot 0 = 1(\text{True})$
$\overline{1}\cdot 0 + 1\cdot\overline{0} = 0\cdot 0 + 1\cdot 1 = 1(\text{True})$

$A \oplus B = \overline{A}\cdot B + A\cdot\overline{B}$

3. Verilog 구현 [Verilog Implementations]

Behavior modeling

- 설계 조건 : A hybrid implementation is not allowed
- 설계 환경 : Model Sim-Intel FPGA Starter Edition / Verilog HDL
- 실험의 검증 : Logical Verilog Operators가 포함된 XOR Gate의 Data Flow model Sample Code로 XOR Gate의 Behavior Modeling 결과를 검증한다.

- 실험 세부 :

(1) XOR Gate의 logic이 어떻게 동작하는지 논리적으로 점검하고, 특정 조건에 동작하도록 설계
(2) 일반적으로 Behavior modeling은 sequential logic과 복잡한 설계에서 주로 사용된다.
(3) 하지만 간단한 Combinational logic에서도 완벽하게 동작한다.

```
1  module xor_behavioral_gate (a, b, out);
2
3          input a, b;
4
5          output out;
6          reg out;
7
8          always @ (a or b)
9          begin //Fill this out (Instantiation of your modules)
10                 if((a == 1'b0 && b == 1'b1) || (a == 1'b1 && b == 1'b0))
11                 begin
12                         out <= 1'b1;
13                 end
14                 else
15                 begin
16                         out <= 1'b0;
17                 end
18         end
19
20 endmodule
```

reg: 간단한 저장 공간으로, 새로운 값 갱신 혹은 지정된 trigger 동작까지 값을 저장한다.

Sensitivity list : 괄호 안의 event가 trigger 될 때 begin과 end 사이의 문장을 실행한다.

A가 FALSE, B가 TRUE인 경우 혹은(or) A가 TRUE, B가 FALSE인 경우에 실행하라.

out <= 1'b1 : out 포트에 1비트의 Binary number 1을 할당하라.

out <= 1'b0 : out 포트에 1비트의 Binary number 0을 할당하라.

Gate-level modeling

- 설계 조건 :

(1) Use AND, OR, NOT gates only

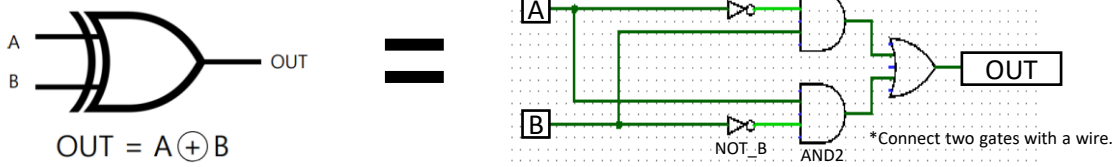
(2) Use the source codes of AND, OR, NOT gates only, which are provided by us.

A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	0

A가 TRUE(1) 이고 B가 FALSE(0)
혹은, A가 FALSE(0) 이고 B가 TRUE(1)인 경우,
OUT이 TRUE(1)가 된다.

이 외의 경우, 모두 FALSE(0)가 된다.

- Disassemble



- 실험 세부 :

(1) XOR Gate의 logic이 Gate-Level에서 어떻게 동작하는지 논리적으로 점검한다.

(2) XOR Gate의 Symbol을 AND, NOT, OR 등의 기본 논리 Gate로 분할한다.

(3) 각 Gate의 Input과 Output을 고려하여 연결한다. *각 Gate는 하나의 wire로 연결한다고 가정한다.

```
1   `include "or_gate.v"
2   `include "not_gate.v"
3   `include "and_gate.v"
4
5
6   module xor_gatelevel_gate (a, b, out);
7
8           input a, b;
9
10          output out;
11
12          wire not_a_out, not_b_out;
13          wire and_1_out, and_2_out;
14
15          not_gate not_a (.a(a), .out(not_a_out));
16          not_gate not_b (.a(b), .out(not_b_out));
17          and_gate and_1 (.a(not_a_out), .b(b), .out(and_1_out));
18          and_gate and_2 (.a(not_b_out), .b(a), .out(and_2_out));
19          or_gate or_1 (.a(and_1_out), .b(and_2_out), .out(out));
20
21  //Fill this out
22  endmodule
```

→ Gate와 Gate 사이를 연결할 wire를 정의한다.

→ 각 Gate에 맞는 Input 값과 Output 값을 연결한다.
Ex) AND Gate : 2개의 Input과 1개의 Output을 할당한다.

Test-Bench : 이번 실험에서 설계한 Behavioral, Gate-Level Modeling이 적합한지 검증.

```
module tb_w3;
        reg A, B;

        //This is for the example of nor gare implementations.
        wire OUT_NOR_D, OUT_NOR_G, OUT_NOR_B;

        //Use the following wires for your homework.
        //Each one is for the output ports for your modules
        //with dataflow modeling, gate-level modeling,
        //and behavioral modeling, respectively.
        wire OUT_XOR_D, OUT_XOR_G, OUT_XOR_B;

        nor_dataflow_gate nor_dataflow_module(.a(A), .b(B), .out(OUT_NOR_D));
        nor_gatelevel_gate nor_gatelevel_module(.a(A), .b(B), .out(OUT_NOR_G));
        nor_behavioral_gate nor_behavioral_module(.a(A), .b(B), .out(OUT_NOR_B));


        xor_dataflow_gate xor_dataflow_module(.a(A), .b(B), .out(OUT_XOR_D));
        xor_gatelevel_gate xor_gatelevel_module(.a(A), .b(B), .out(OUT_XOR_G));
        xor_behavioral_gate xor_behavioral_module(.a(A), .b(B), .out(OUT_XOR_B));
        //Fill this out (Instantiation of your modules)

        initial
        begin
                A = 0; B = 0;
                #10 A = 0; B = 0;
                #10 A = 0; B = 1;
                #10 A = 1; B = 0;
                #10 A = 1; B = 1;
        end


endmodule
```

→ Module의 Input, Output 값과, Test-Bench 상의 Input Output 값을 연결한다.
일반적으로 Input 값은 reg. Output 값은 wire로 선언한다.
(Out 에 다른 영향을 미칠 요소를 사전에 차단하기 위해)

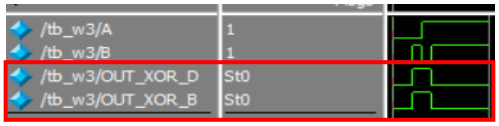
→ 각각의 Modeling 기법에 따른 Out을 서로 다른 wire로 할당한다.

→ 모듈의 각 포트에 Test-Bench의 wire와 reg를 할당한다.

→ Input 값인 A와 B를 초기화 하고,
지정된 delay 동안 지정한 값을 반환하도록 한다.

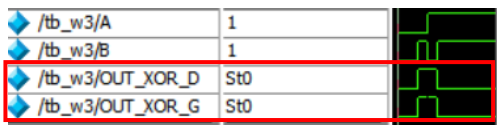
4. 실험 결과 (고찰) [Experiment result]

(1) Behavioral Modeling과 Data Flow Modeling의 시뮬레이션 결과 비교



동일한 파형으로 시뮬레이션 됨을 확인할 수 있다.

(2) Gate-Level Modeling과 Data Flow Modeling의 시뮬레이션 결과 비교



동일한 파형으로 시뮬레이션 됨을 확인할 수 있다.

(3) 고찰

이번 실험에서는 다양한 Modeling 기법을 통해 XOR Gate를 설계하는 방법을 탐구하였다. H/W 설계 기법인 3가지 Modeling 방법에 대한 이해를 기반으로 각 Modeling의 특성 및 장단점을 이해하며 Module 설계를 진행하였고, 검증 Code인 Data flow 코드와 동일한 파형을 가진 결과값을 도출할 수 있었다. H/W 설계에 앞서 선행되어야 하는 Boolean 함수식과 Truth Table을 통해, 설계되는 Module의 순차적인 흐름 및 실행 조건을 이해할 수 있었으며, 논리적 탐구가 선행된 H/W Design의 중요성을 체감할 수 있었다. 실험을 진행하는 도중 Gate가 Open Circuit 혹은 하나의 입력 포트에 두가지 이상의 입력이 들어가는 등의, 회로적 오류 발생 시, Compile은 정상적으로 진행되나 시뮬레이터 동작 오류 (HiZ)가 발생함을 알 수 있었고 이를 해결하기 위하여 정상적인 Port 연결이 필수적임을 알 수 있었다

5. 참 조

오윤호 (2021). *Logic Design Laboratory(ICE2005) Week3*. Sungkyunkwan Univ.