컴퓨터 공학 기초 실험2 보고서

실험제목: 2-to-1 MUX

실험일자: 2021년 09월 12일 (월)

제출일자: 2021년 09월 19일 (일)

학 과: 컴퓨터정보공학부

담당교수: 공진흥 교수님

실습분반: 월요일 0, 1, 2

학 번: 2018202046

성 명: 이준휘

1. 제목 및 목적

A. 제목

2-to-1 MUX

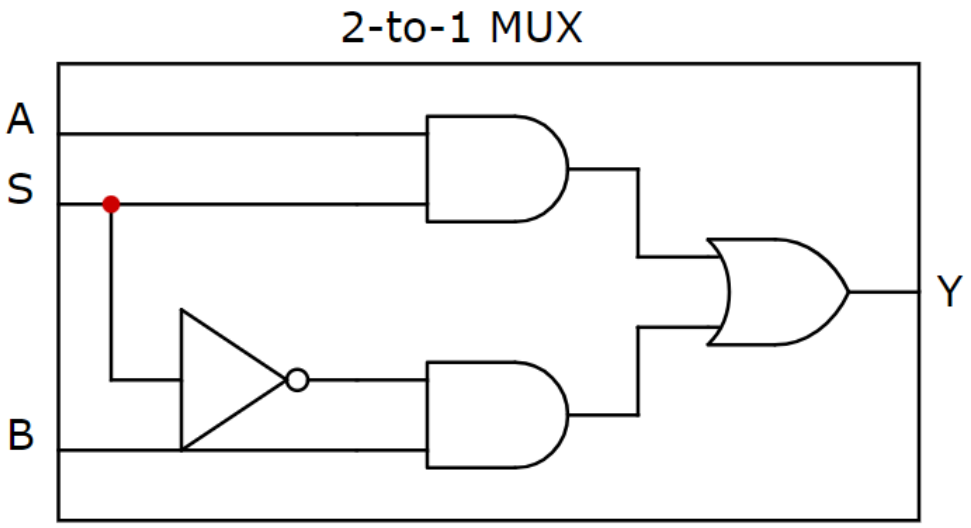
B. 목적

해당 수업을 통해 2-to-1 MUX를 이해한다. 또한 Verilog를 통해 해당 Multiplexer를 설계하며 Verilog의 사용법을 익힌다. 마지막으로 testbench를 통하여 설계한 2-to-1 MUX를 검증하는 방법을 숙달한다.

2. 원리(배경지식)

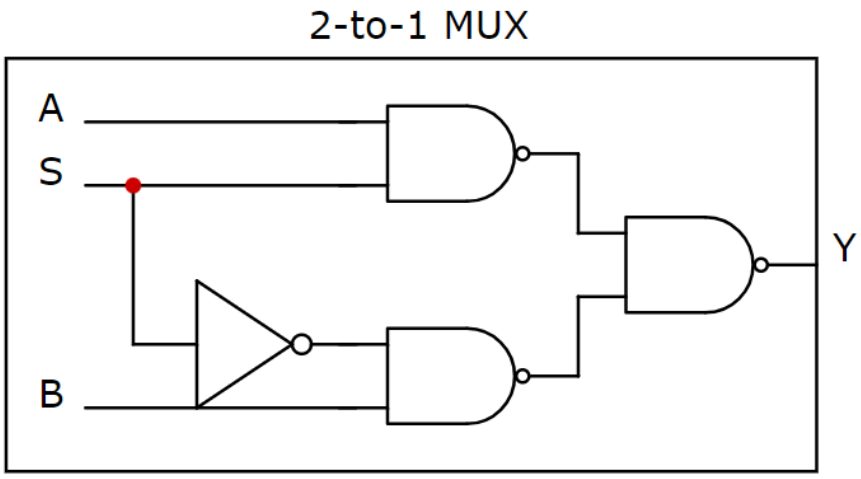
Multiplexer(MUX)란 여러 입력 중 하나를 선택하여 출력하는 회로로, n개의 입력을 받고, 이를 S의 값에 따라 선택적으로 선별한다. 그리고 해당하는 값에 맞는 신호를 출력한다. n개의 신호를 입력 받을 경우 이를 구별하기 위한 S값의 개수는 $\lceil log_2 n \rceil$(해당 수의 올림) 만큼 필요하다.

2-to-1 MUX란 2개의 신호 중 1개의 신호를 선택하여 출력하는 회로를 말한다. 해당 신호는 아래와 같이 구성된다.



위의 회로의 식을 간단히 하면, $Y = (A \cdot S) + (B \cdot \overline{S})$이다. 해당 식을 살펴보면, S가 0일 때 A는 AND Gate에 의해 신호가 전달되지 않고, B는 1과의 AND 연산임으로 B의 값이 나온다. 두 값을 OR로 연산할 경우 B의 값이 출력된다. S가 1일 경우 반대로 B는 AND Gate에 의해 신호가 전달되지 않고, A는 1과의 AND 연산임으로 A의 값이 나온다. 두 값을 OR로 연산할 경우 A의 값이 출력된다.

허나 위의 회로는 3개의 Gate 종류를 사용하였고, AND Gate와 OR Gate는 NAND Gate와 NOR Gate에 보다 트렌지스터를 1.5배 더 사용한다. 이를 해결하기 위해 우리는 해당 회로에서 Bubble Shift를 사용하여 버블을 생성하고 정리하면 아래와 같은 회로가 나온다.



해당 회로의 식을 보면 $Y = \overline{(\overline{(A \cdot S)} + \overline{(B \cdot \overline{S})})}$로 정리하면 위의 식과 같은 것을 알 수 있다. 해당 방식으로 설계했을 경우 사용되는 Gate의 종류를 줄일 수 있고 트렌지스터를 더욱 적게 사용할 수 있다.

Verilog는 module 단위로 코드를 짠다.

module의 사용법은 module module_name(input_output_list); 코드를 통해 모듈의 시작 위치를 선언하고 내용을 입력한 후 endmodule을 통해 module의 종료 위치를 잡아준다.

입력 신호의 경우에는 input을 통해, 출력 신호의 경우에는 output을 통해 선언하고, 중간 길목의 역할이 필요할 경우 wire을 통해 해결한다. 그리고 reg는 가장 처음에 값이 들어가는 통로다. Verilog는 모든 신호를 [to:from]을 통해 from~to의 신호를 단체로 선언할 수 있다. 이는 wire에도 해당되는데 wire는 기본적으로 선언 없이 사용할 수 있지만 묶음으로 선언한 신호에 대해서는 선언 없이 사용할 수 없다.

상수의 경우 <size>'<radix> value로 사용할 수 있으며 10진수, 16진수, 2진수, 8진수를 지원하고, underbar를 사용하여 구분감을 줄 수 있다.

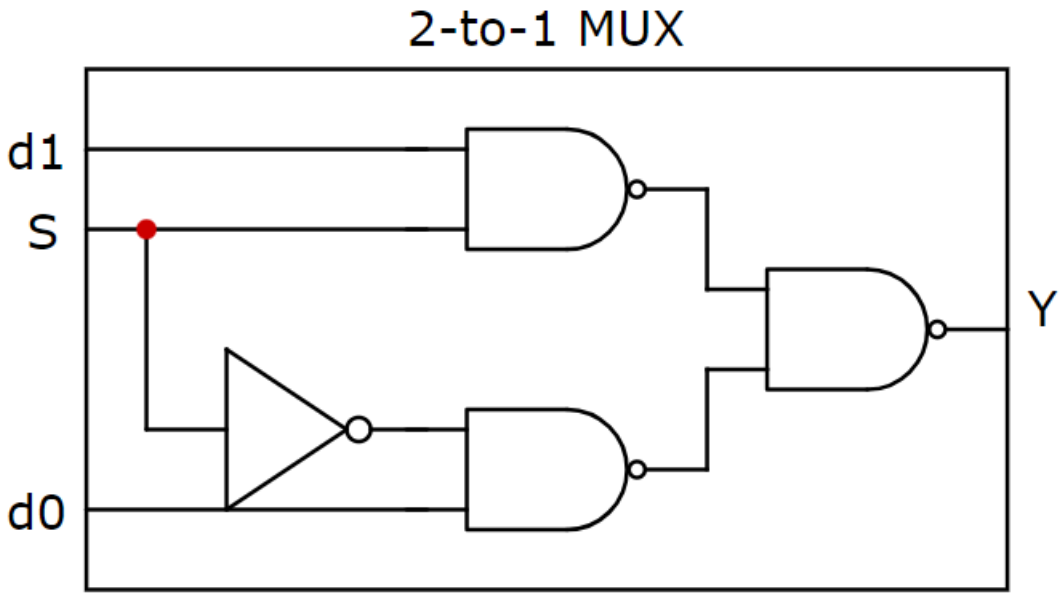
Verilog는 C/C++과 마찬가지로 논리 연산과 Shift 연산, 비트 연산을 지원한다. 단 z나 x 신호로 인해 값을 모를 경우 x로 출력된다.

3. 설계 세부사항

윗쪽에서 설명한 2-to-1 MUX 중 필자는 NAND Gate와 NOT Gate를 사용한 MUX를 구현하였다. 해당 MUX의 Karnaugh map은 다음과 같다.

s \ d0, d1	00	01	11	10
0	0	0	1	1
1	0	1	1	0

Karnaugh map에서 공통된 부분을 묶어 식을 정리하면 $Y = (d0 \cdot \overline{S}) + (d1 \cdot S)$ 로 정리가 되고 이 식을 bubble shift를 통해 아래의 도식으로 정리할 수 있다.



해당 칩의 설계를 위해 2개의 .v 파일을 사용하였다.

우선 첫 번째 gates.v에서 NOT Gate인 _inv 모듈과 NAND Gate인 _nand2 모듈을 구현하였다.

_nand2 모듈의 경우 2개의 input 단자(a, b)를 가지며 1개의 output 단자(y)를 가진다. 그리고 assign을 통해 y의 값을 a와 b의 NAND 연산의 결과로 설정하여 연결한다.

_inv 모듈은 1개의 input 단자(a)를 가지며 1개의 output 단자(y)를 가진다. 그리고 assign을 통해 y의 값을 a의 invert된 값으로 설정하여 연결한다.

두 번째 mx2.v 파일에서는 3개의 input 단자(d0, d1, s) 신호와 1개의 output 신호(y)를 사용하여 mx2 모듈을 구현한다.

우선 s의 신호를 _inv 모듈, ni1에 s를 input으로 주고 output을 wire wi1에 연결한다.
이 wi1의 신호와 d0의 신호를 _nand2 모듈, n0에 연결하며 output을 wire w0에 연결한다.
그 후 _nand2 모듈, n0에 d1와 s신호를 input으로 주고 output을 wire w1에 연결한다.
마지막으로 _nand2 모듈을 거쳤던 w0와 w1을 다시 _nand2 모듈, n2의 input으로 주고 그 결과를 y로 연결한다.

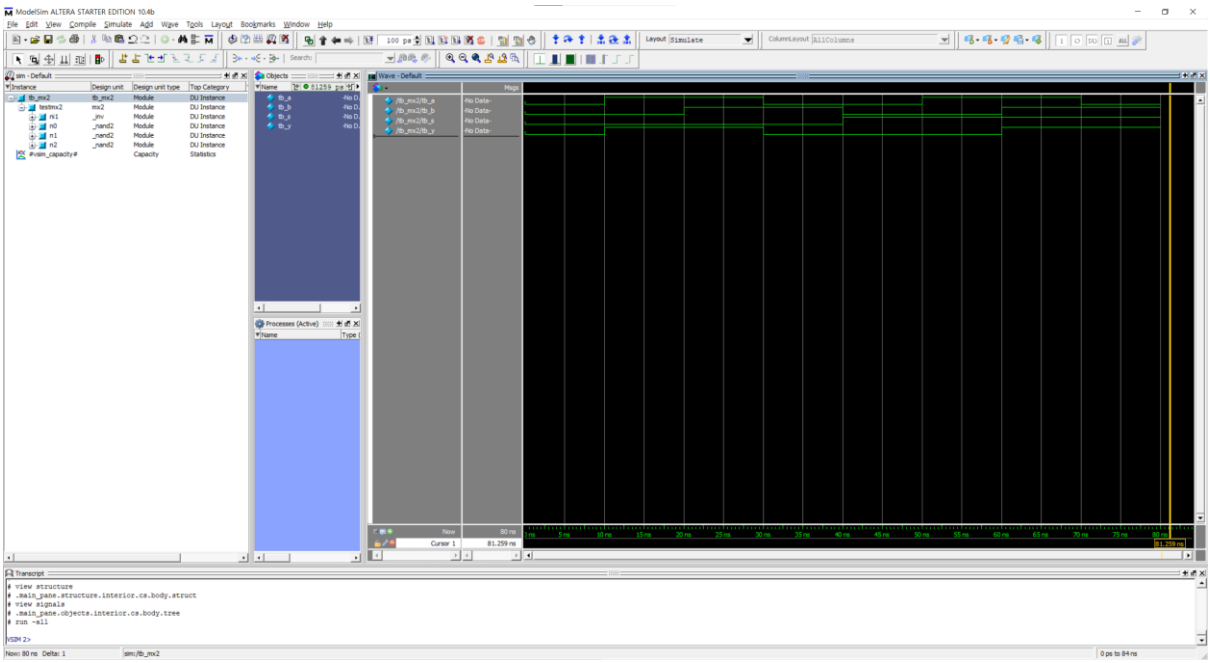
4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

해당 testbench의 경우 가장 기본적인 simple testbench를 사용하였으며, 인자가 적기 때문에 모든 결과를 확인하는 Exhaustive verification을 통해 검증한다.

값을 입력할 reg인 tb_a, tb_b, tb_s를 생성하고 결과를 저장할 wire tb_y를 선언한다.
그리고 mx2 모듈, testmx2를 생성하고 input에 tb_a, tb_b, tb_s를, output에 tb_y를 연결한다.
s가 0일 때 tb_a와 tb_b를 00, 10, 11, 10의 신호를 #10 간격으로 주고 결과를 본다. 그 후 s를 1로 설정하여 위의 작업을 다시 수행하여 결과를 보고 프로그램을 마친다.

B. 합성(synthesis) 결과



결과를 확인해보면 우선 tb_a와 tb_b, tb_s에 따라 tb_y가 바뀌는 것을 알 수 있다.
해당 wave를 표로 정리해서 나타내면 다음과 같다

tb_y tb_a, tb_b	00	01	11	10
0	0	0	1	1
1	0	1	1	0

이 flow summary와 Karnaugh map을 비교해보면 정상적으로 결과가 나온 것을 확인할 수 있다.

C. FPGA board targeting 결과

5. 고찰 및 결론

A. 고찰

과제를 수행하면서 저번 학기 마지막에 조금 해본 Verilog였지만, 간략하게 했던 것이기 때문에 사용하는 데에 익숙하지 않았다. module을 선언할 때 ;를 사용하지 않는 등의 문제를 겪었으나 점차 사용하는 방법에 익숙해졌다. 마지막에는 별 다른 문제없이 과제를 진행하였다.

B. 결론

해당 실험을 통하여 Verilog HDL의 사용법을 익힐 수 있는 시간이었다. 또한 Karnaugh map을 만들고 식으로 이를 정리하며, Bubble shift를 함으로써 지난 학기의 내용을 복기하는 기회였다. 또한 testbench를 진행하면서 simple testbench 말고도 파일을 통해 에러 코드가 나는지 확인하는 self-checking testbench를 해보고 싶었다.

6. 참고문헌

이준환 교수님/디지털논리회로1/광운대학교(컴퓨터정보공학부)/2021

이준환 교수님/디지털논리회로2/광운대학교(컴퓨터정보공학부)/2021

multiplexer/

https://terms.naver.com/entry.naver?docId=1180077&cid=40942&categoryId=32849