

LAB 5. Combinational Logic Design 2

MUX / DEMUX / Tri-State Buffer

2017 Fall Logic Design LAB

Department of Computer Science and Engineering

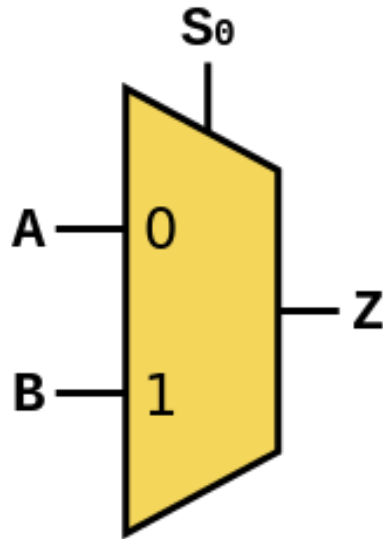
Seoul National University

Outline

1. Multiplexer와 Demultiplexer의 기초
2. Tri-state buffer의 활용방법

Multiplexer와 Demultiplexer의 기초

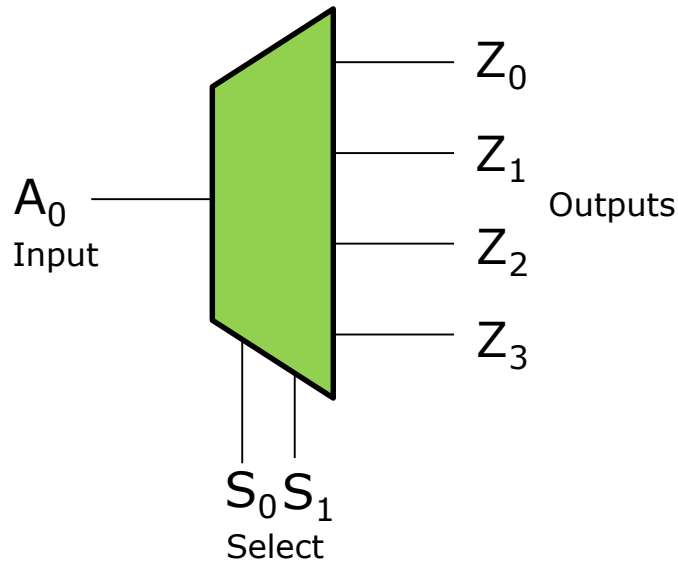
Multiplexer



Input			Output
S_0	A	B	Z
0	0	0	0
	0	1	0
	1	0	1
	1	1	1
1	0	0	0
	0	1	1
	1	0	0
	1	1	1

- S의 신호값에 따라 출력 값을 선택적으로 결정하는 회로
 - $A_0 \dots A_{2^n-1}$ 의 input에 대하여 $s_0 \dots s_{n-1}$ 의 select 신호로 하나의 출력 신호 선택
 - (input 개수):1 MUX 로 표현
 - 2:1 MUX의 경우 하나의 select 신호를 통해 A, B 중 하나의 값을 출력
 - 4:1 MUX의 경우 두 개의 select 신호를 통해 A, B, C, D 중 하나의 값을 출력

Demultiplexer (Decoder)

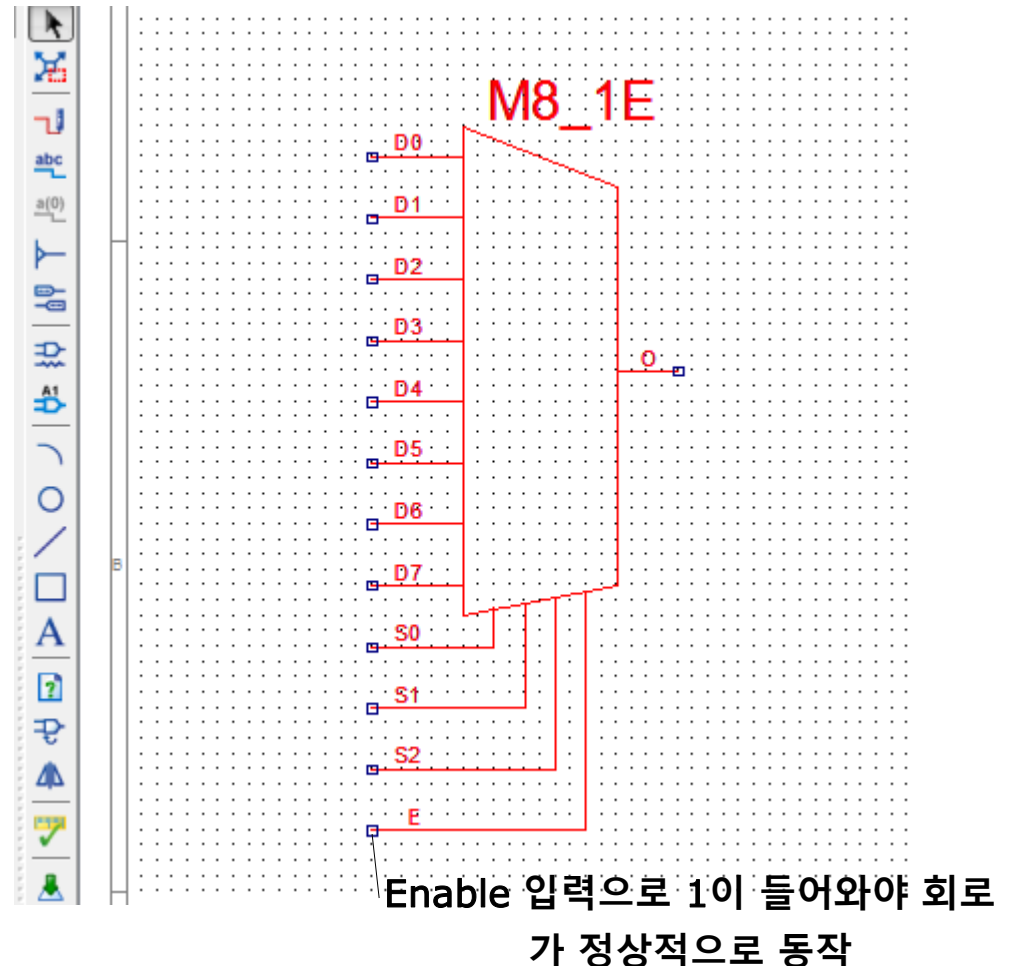
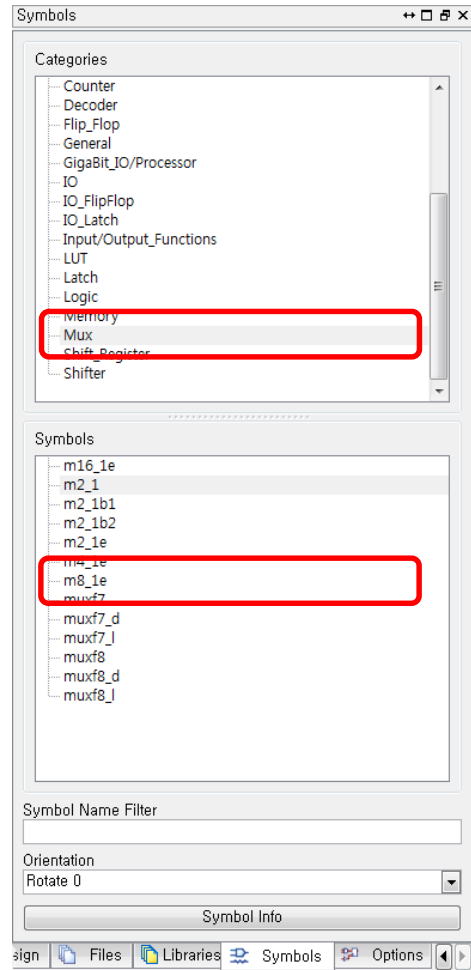


S_0	S_1	Z_0	Z_1	Z_2	Z_3
0	0	A_0	0	0	0
0	1	0	A_0	0	0
1	0	0	0	A_0	0
1	1	0	0	0	A_0

- S 의 신호값에 따라 2^n 개의 출력 중 하나에 입력된 값을 출력하는 회로
 - $Z_0 \dots Z_{2^n-1}$ 의 output 중 A_0 값이 출력될 위치를 $S_0 \dots S_{n-1}$ 의 select 신호로 선택
 - (selector 개수):(output 개수) DEMUX(decoder) 라고 호칭
 - MUX와 반대되는 개념

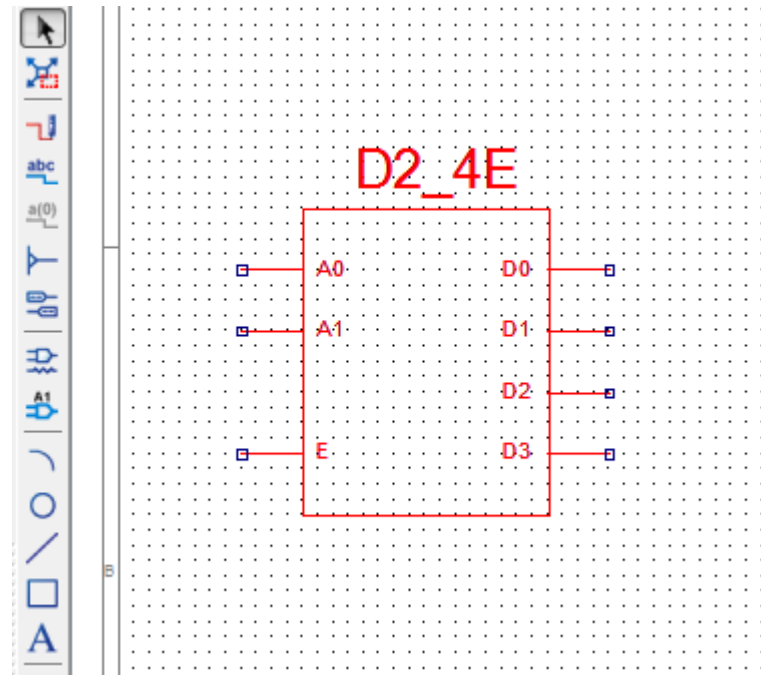
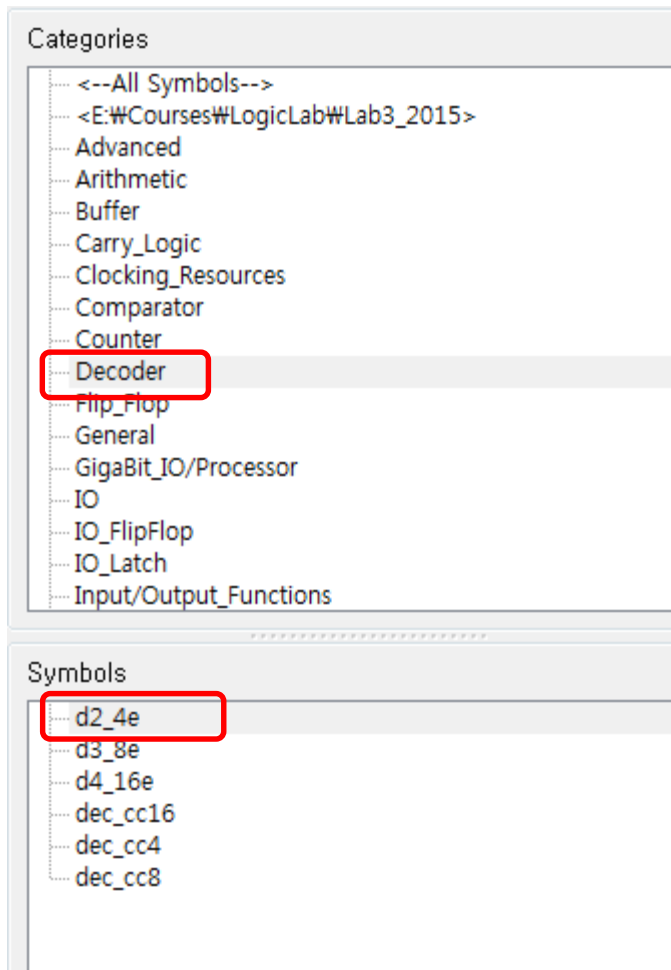
Xilinx에서의 MUX 심볼 사용방법

- Symbol 탭에서 MUX 카테고리를 선택하여 입력
 - 단 4:1 MUX 이상의 기본 심볼의 경우 별도의 enable 신호 입력을 받도록 구성됨



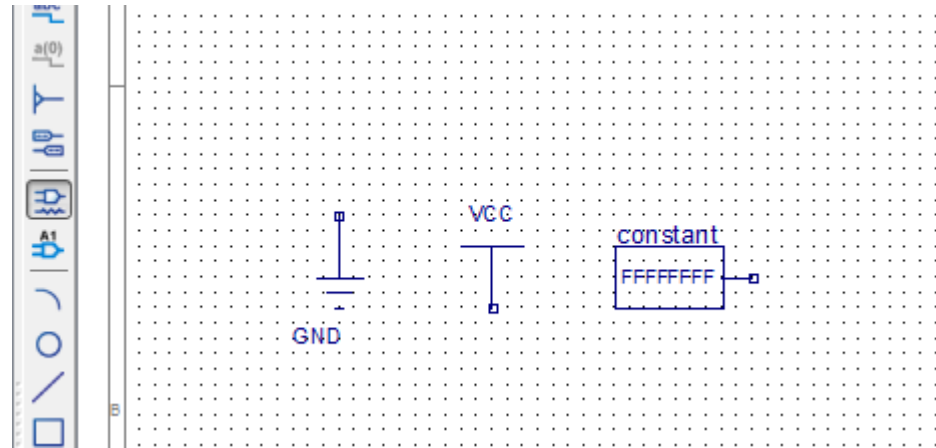
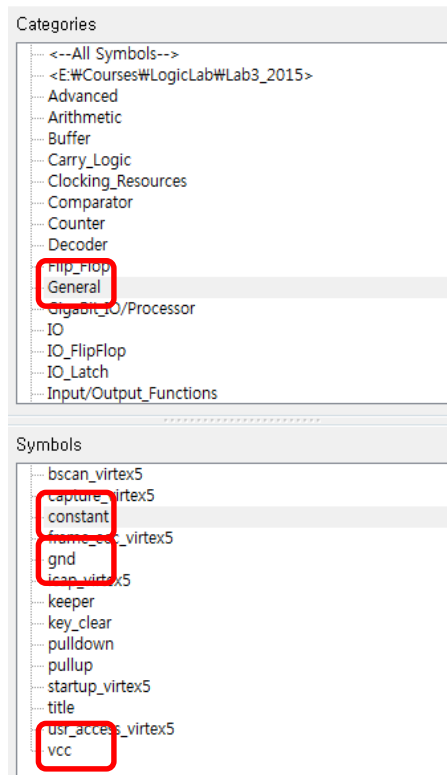
Xilinx에서의 Decoder 심볼 사용방법

- Symbol 탭에서 decoder 카테고리를 선택하여 입력



Xilinx에서의 고정된 신호 입력 방법

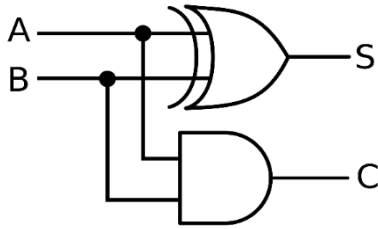
- Symbol 탭의 general 카테고리에서 GND, VCC, constant 심볼 사용
 - GND: 논리값 0이 입력됨
 - VCC: 논리값 1이 입력됨
 - Constant: 고정된 값으로 버스 입력(여러 비트 입력)



Structural vs Behavioral Modeling

Structural modeling

이미 구현된 sub-module들을 재사용하여 큰 단위의 모듈을 구성



Half adder를 구현하기 위해서는 가장 작은 단위의 sub module인 XOR, AND 모듈을 이용.

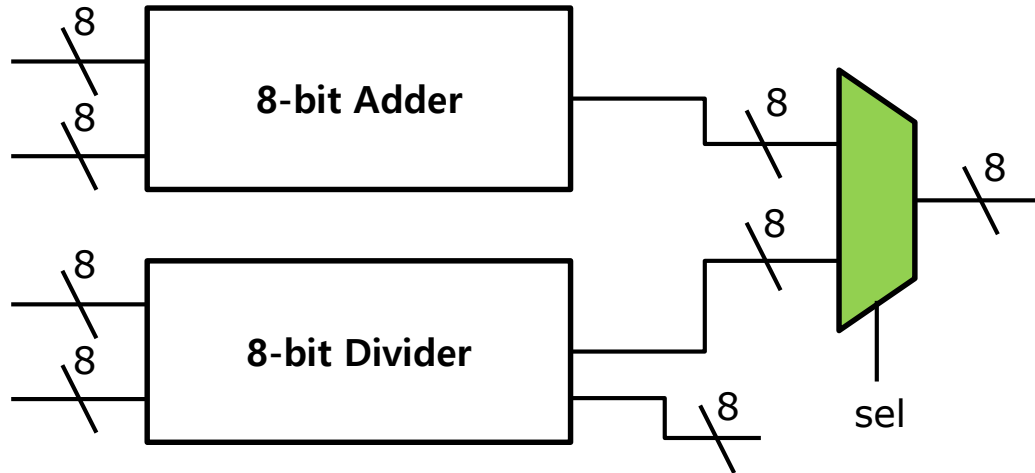
Behavioral modeling

입력되는 신호에 따라 출력 신호를 어떻게 생성할지 입출력의 관계를 묘사

Half adder를 구현하기 위한 또다른 방법으로써, 입력과 출력에 대한 관계를 HDL로 작성

1. $S = A \oplus B$
2. $C = A \& B$
3. A, B가 바뀔 때마다 동작1, 2를 수행

Structural vs Behavioral Modeling 예제



```
module AdderDivider(  
    input [7:0] A,  
    input [7:0] B,  
    input sel,  
    output reg [7:0] C)  
always @ (*)  
begin  
    .....  
    AdderDivider의 행동 구현  
end
```

AdderDivider의 행동 구현




```
//모듈 선언 생략  
wire [7:0] Q, R;  
reg [7:0] sum;  
Divider8bit divider(A, B, Q, R)  
always @ (*)  
begin  
    sum <= a+b;  
    if(sel) C <= sum;  
    else C <= Q;  
end
```

Structural Modeling

Behavioral Modeling

behavioral modeling과
structural modeling을
함께 사용

Verilog를 이용한 4:1 MUX 구현

```
module Mux (  
    input [1:0] sel,  
    input d0,  
    input d1,  
    input d2,  
    input d3,  
    output reg y  Behavioral한 구현을 위해 y 값을 reg로 설정  
);  
  
always @ (*)  
begin  
    case (sel)   
        2'b00: y = d0;   
        2'b01: y = d1;  
        2'b10: y = d2;  
        2'b11: y = d3;  
    endcase  
end  
endmodule
```

Verilog를 이용한 2:4 Decoder 구현(Behavioral)

```
module Decoder (  
    input [1:0] sel,  
    input A,  
    output reg [3:0] y  
);
```

```
always @(*)
```

```
begin
```

```
    y[3] = A & sel[1] & sel[0];
```

```
    y[2] = A & sel[1] & ~sel[0];
```

```
    y[1] = A & ~sel[1] & sel[0];
```

```
    y[0] = A & ~sel[1] & ~sel[0];
```

```
end
```

```
endmodule
```

출력 y에 대한 대입 연산

각 select 신호 값을 AND 하여 조건 설정

조건이 만족될 경우 A의 값에 따라 출력

Verilog를 이용한 2:4 Decoder 구현(Structural)

```
module Decoder (  
    input A,  
    input [1:0] sel,  
    output [3:0] y   
    --- Input의 변화에 따라 자동으로 함께 변화 (reg 사용하지 않음)  
);
```

Assign을 통해 조건에 따라 값을 바꾸지 않고 회로 결과를 바로 연결하여 출력

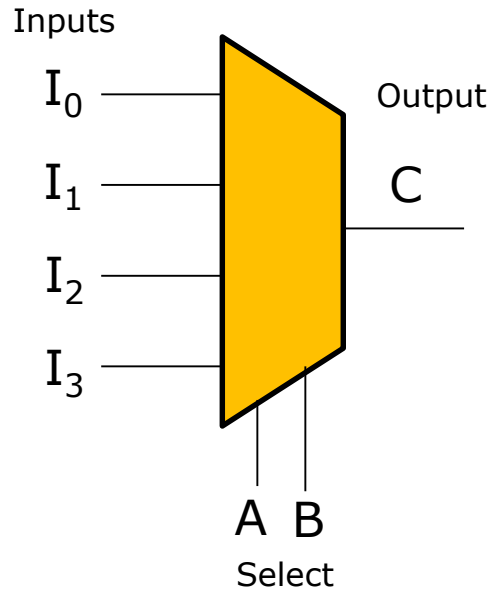
```
assign y[3] = A & sel[1] & sel[0];  
assign y[2] = A & sel[1] & ~sel[0];  
assign y[1] = A & ~sel[1] & sel[0];  
assign y[0] = A & ~sel[1] & ~sel[0];
```

```
endmodule
```

< 같은 동작을 수행하는 코드 >

```
three_input_and and4(A, ~sel[1], ~sel[0], y[0])
```

Multiplexer를 이용한 논리 설계

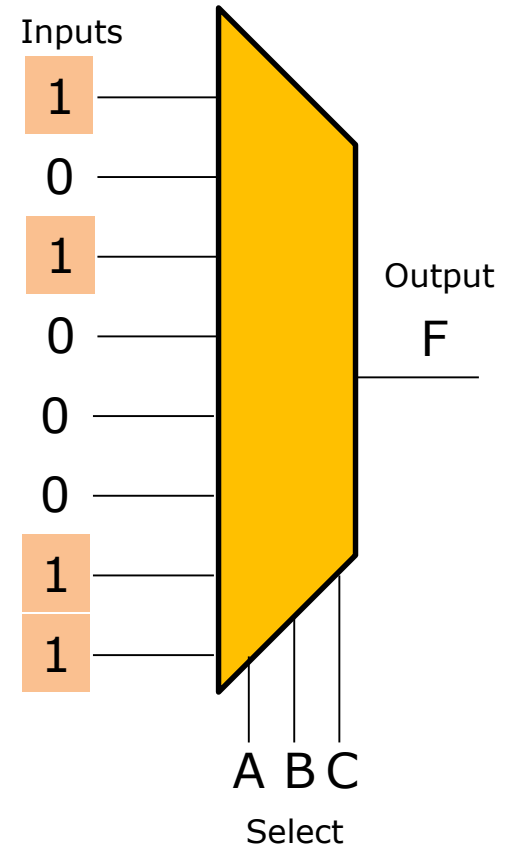
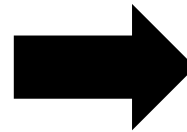


Input		Output
A	B	C
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

- 임의의 N bit 논리 회로를 $2^N : 1$ MUX를 사용하여 구성 가능
 - N개의 입력은 select 신호로 사용
 - 2^N 개의 data input은 해당하는 논리값에 따라 0 또는 1을 입력
 - Lookup table을 회로로 구현하는 것과 동일
 - $F(A,B) = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$

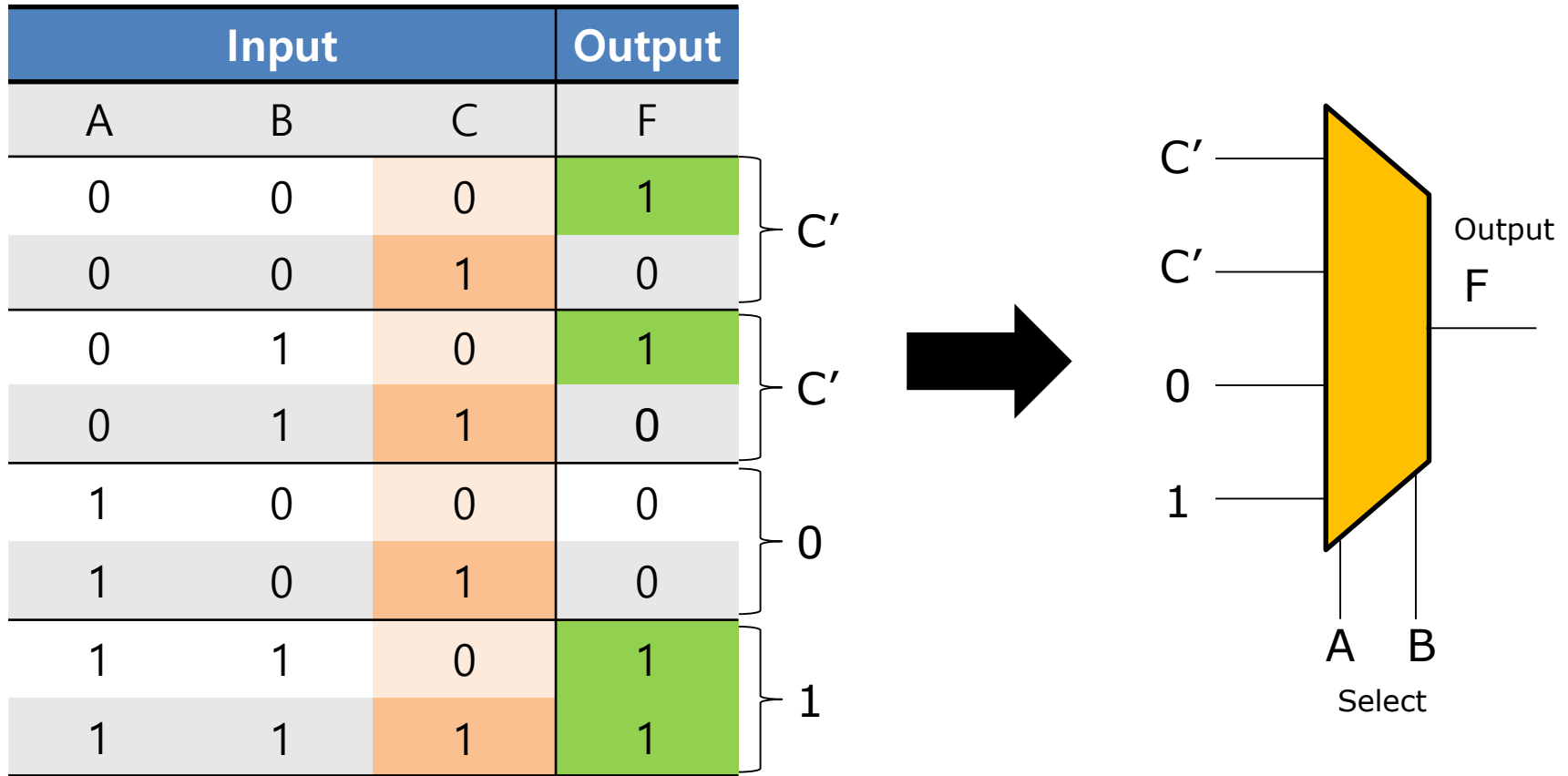
Multiplexer를 이용한 논리 설계의 예

Input			Output
A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



- $F(A,B,C) = m_0 + m_2 + m_6 + m_7$
 $= A'B'C' + A'BC' + ABC' + ABC$

Multiplexer를 이용한 논리 설계 최적화

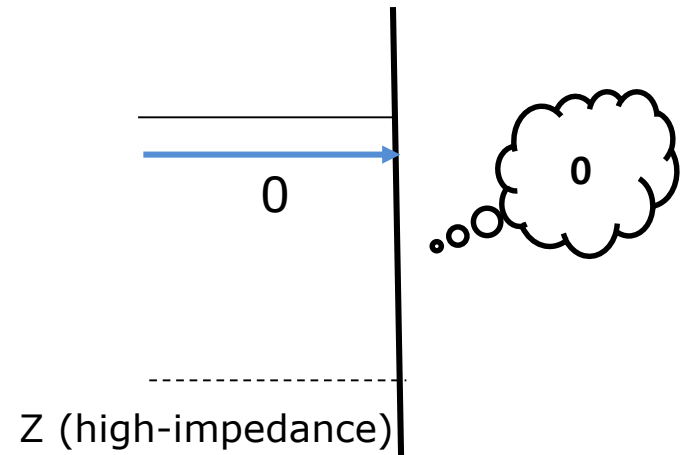
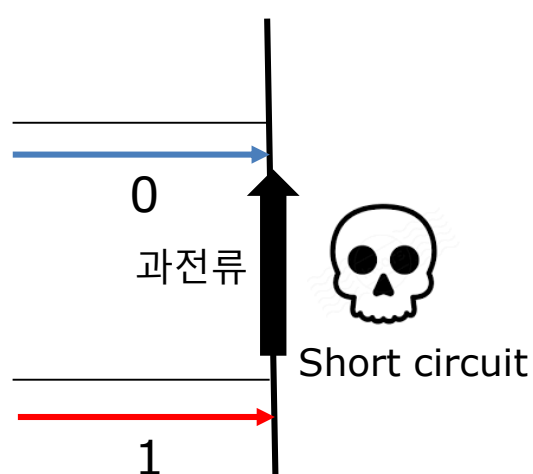


- 3개의 변수 대신 2개의 변수만 select 신호로 사용(4:1 MUX)
 - C, C', 0, 1 네 가지 case중 하나와 대응 가능
 - $F(A,B,C) = A'B'C' + A'BC' + AB(C' + C) = A'B'C' + A'BC' + AB'(0) + AB(1)$

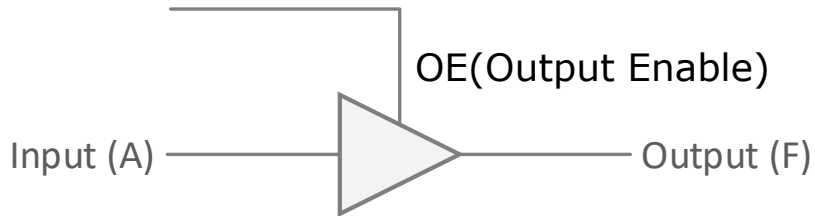
Tri-state Buffer의 활용방법

High Impedance와 Tri-state

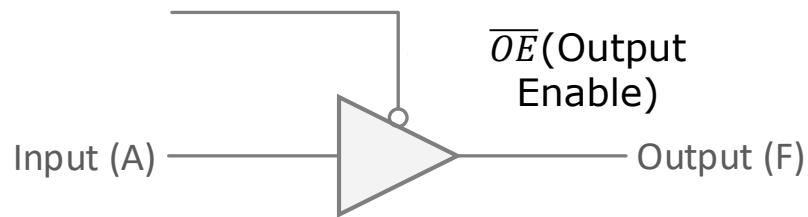
- 하나의 wire에 여러 출력 소자를 동시에 연결하여 선택적으로 출력하고 싶은 경우
 - Computer의 공용 bus와 같은 구조
 - 같은 wire에 여러 전압이 공급되는 출력 값이 동시에 연결될 경우 short circuit 발생
- High impedance 출력 (floating, hi-Z...)
 - 사실상 회로가 끊어진 상태
 - 여러 개의 출력이 같은 wire로 연결될 경우, 이 중 **하나만** 논리값을 출력하고 나머지는 high impedance 상태로 둬으로써 MUX 없이도 선택적으로 출력 가능.



Tri-state buffer



A	OE	F
X	0	Z
0	1	0
1	1	1



A	\overline{OE}	F
0	0	0
1	0	1
X	1	Z

- 출력의 논리값을 0 또는 1이 아닌 high impedance를 출력 시킬 수 있는 소자
 - OE (Output Enable) 입력을 통해 조절
 - High impedance 출력 상태는 Z로 표현
 - High impedance 상태를 이용하여, 여러 소자들의 출력을 같이 묶어서 연결 가능
 - 이 때 원하는 출력 하나를 제외한 나머지 소자들은 Z 상태를 출력

Verilog를 이용한 Tri-state Buffer 구현

```
module Tristate(  
    input OE,  
    input A,  
    output reg y  
);  
  
always @(*)  
begin  
    case (OE)  
        0 : y = 1'bZ;  
        1 : y = A;  
    endcase  
end  
endmodule
```

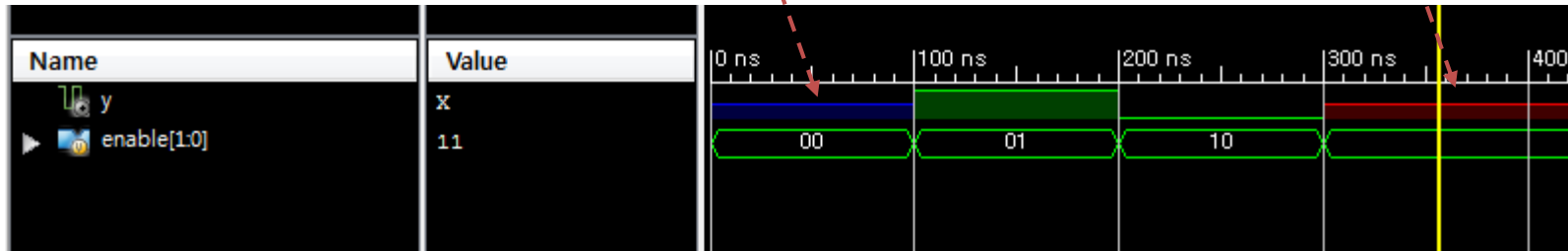
Tri-state 간의 연결

```
module Tritest(  
    input [1:0] enable,  
    output y    Wire assign으로 출력을 생성하므로 reg 붙이지 않음  
);  
  
Tristate tristate1(enable[1], 0, y);  
Tristate tristate2(enable[0], 1, y);  
  
endmodule
```

같은 y 출력 wire에 연결

High impedance 상태
(연결된 신호 없음)

두 가지 입력이 중복으로
wire에 연결되어
short circuit 발생



실습 1 – 4:1 MUX를 이용한 3-input XOR 게이트 구현

■ 목표

- 4:1 MUX 하나를 사용하여 3개의 입력을 받는 3-input XOR 게이트로 동작하는 회로를 구성

■ 실험 내용

- 3변수이므로 앞서 배운 기법을 활용하면 $2^{3-1}:1 = 4:1$ MUX로 구현 가능함
- 4:1 MUX Verilog 모듈을 활용하여 structural modeling으로 구현
- 아래의 3-input XOR 게이트의 진리표를 참고

■ 제출 사항

- 구현한 코드
- 시뮬레이션 결과 스크린샷

A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

실습 2 – 2:4 Decoder를 이용한 4:1 MUX 구현

- 목표
 - 2:4 decoder와 tri-state buffer를 이용하여 4:1 MUX를 구현
- 실험 내용
 - Tri-state buffer는 PPT의 Tristate 모듈을 활용
 - AND, OR 등 기본적인 gate 로직이 필요할 경우 별도 모듈로 구현
 - 2:4 decoder 모듈을 구현한 뒤 structural modeling으로 4:1 MUX 구현
- 제출 사항
 - 구현한 코드
 - 시뮬레이션 결과
 - 입력 1: $I_3I_2I_1I_0 = 0110$, 모든 S_1S_0 에 대한 결과 스크린샷
 - 입력 2: $I_3I_2I_1I_0 = 1001$, 모든 S_1S_0 에 대한 결과 스크린샷

실습 3 – 2-bit comparator 구현

■ 목표

- 두 개의 2 bit 숫자(A_1A_0, B_1B_0)를 입력 받고, 비교 연산 결과를 출력
 - $A_1A_0 \geq B_1B_0$ 일 경우 1을 출력 (예, $A_1A_0=10, B_1B_0=01$ 일 경우 1 출력)
 - $A_1A_0 < B_1B_0$ 일 경우 0을 출력 (예, $A_1A_0=01, B_1B_0=11$ 일 경우 0 출력)

■ 실험 내용

- 모든 입력의 경우의 수 16가지에 대해 진리표를 작성
- 8:1 MUX를 사용하여 구현(MUX는 behavioral modeling으로 구현)
- 먼저 시뮬레이터를 통해 동작을 확인
- SNU Logic Design 보드를 사용하여 구현(조교에게 동작 확인 받기)
 - DIP Switch 1, 2, 3, 4 번을 입력으로 설정
 - LED 0번을 통해 결과값 출력

■ 제출 사항

- 모든 입력의 경우의 수에 대한 진리표
- 구현한 코드

실습 제출 안내

- 제출 항목
 - 실습1 구현한 코드, 시뮬레이션 결과 스크린샷
 - 실습2 구현한 코드, 시뮬레이션 결과 스크린샷
 - 실습3 진리표, 구현한 코드
 - 실습3의 경우 조교에게 보드 동작 검사 필요
- 제출 안내
 - 시뮬레이션 결과 및 진리표를 보고서에 포함(보고서는 PDF 파일 형식)
 - 각 실습에서 작성한 Verilog 소스 코드
 - 작성한 보고서 및 소스코드를 압축하여 하나의 파일로 제출
- 제출 방법 및 기한
 - ETL 과제 게시판에 **팀 별로 제출**
 - **일요일 오후 6시**까지