

ModelSIM으로 MUX simulation 하기

HDL & Verilog

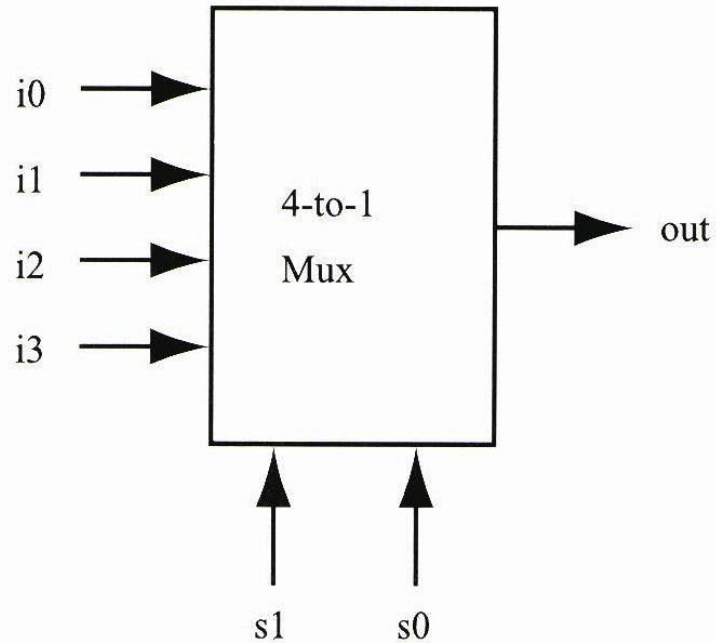
◇ HDL

- ◆ Hardware Description Language
- ◆ C → 범용 프로세서 이용 / HDL → 전용 SoC 또는 IP 설계
- ◆ HDL Design => Simulation & Synthesis & etc. => Chip / FPGA
- ◆ HDL 설계의 장점
 - ◆ C 와 비슷한 친숙한 명령어
 - ◆ Behavioral 하게 설계할 수 있음

◇ Verilog

- ◆ HDL 중에 가장 널리 쓰이는 Language
- ◆ 수많은 Synthesis Tools / Foundry vendors 가 Support
- ◆ Verilog Editing & Simulation Tool : **ModelSim**
- ◆ Synthesis Tool : **ISE Design Suite for FPGA / Design Compiler for Silicon**

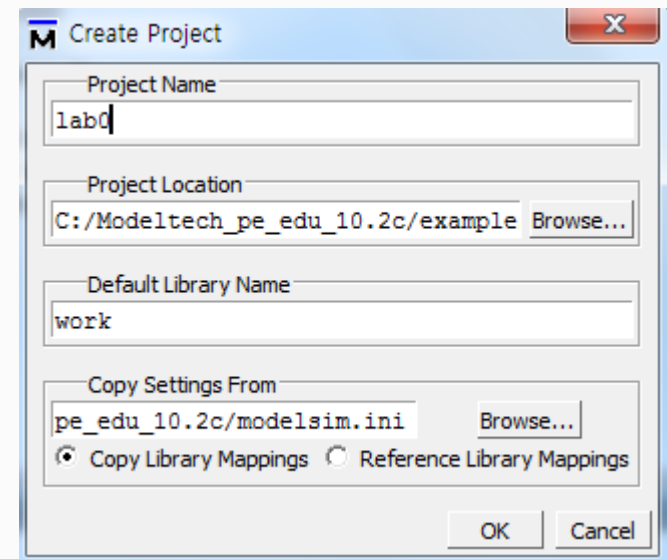
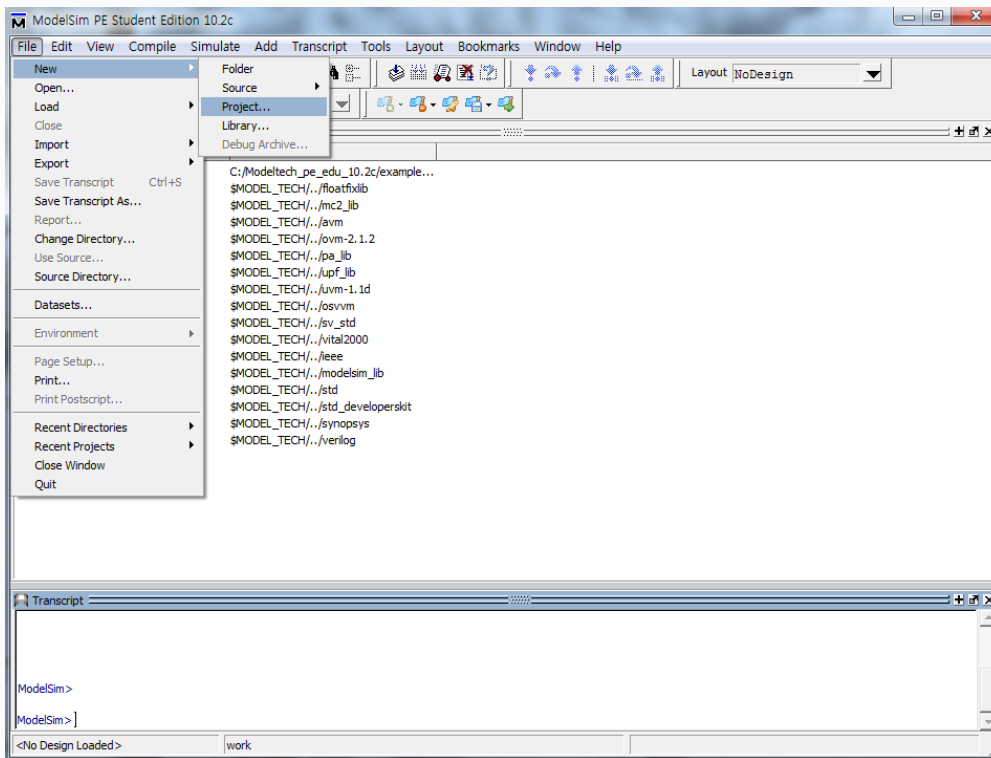
4:1 Multiplexer 설계 및 시뮬레이션



s_1	s_0	out
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

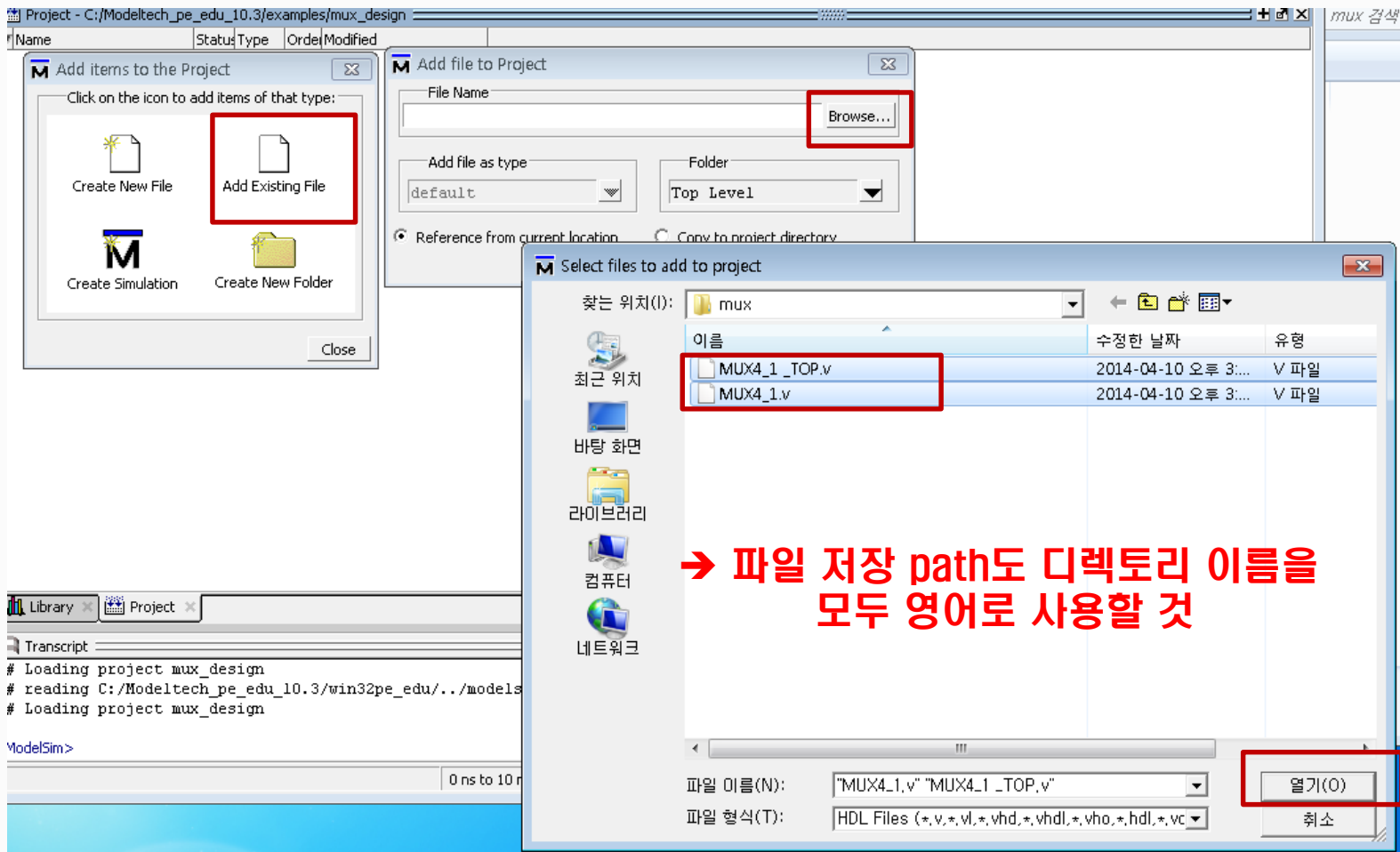
실험 내용

- ◆ ModelSim 을 실행시키고 새 프로젝트를 생성
→ 프로젝트 이름 한글 사용하지 말 것



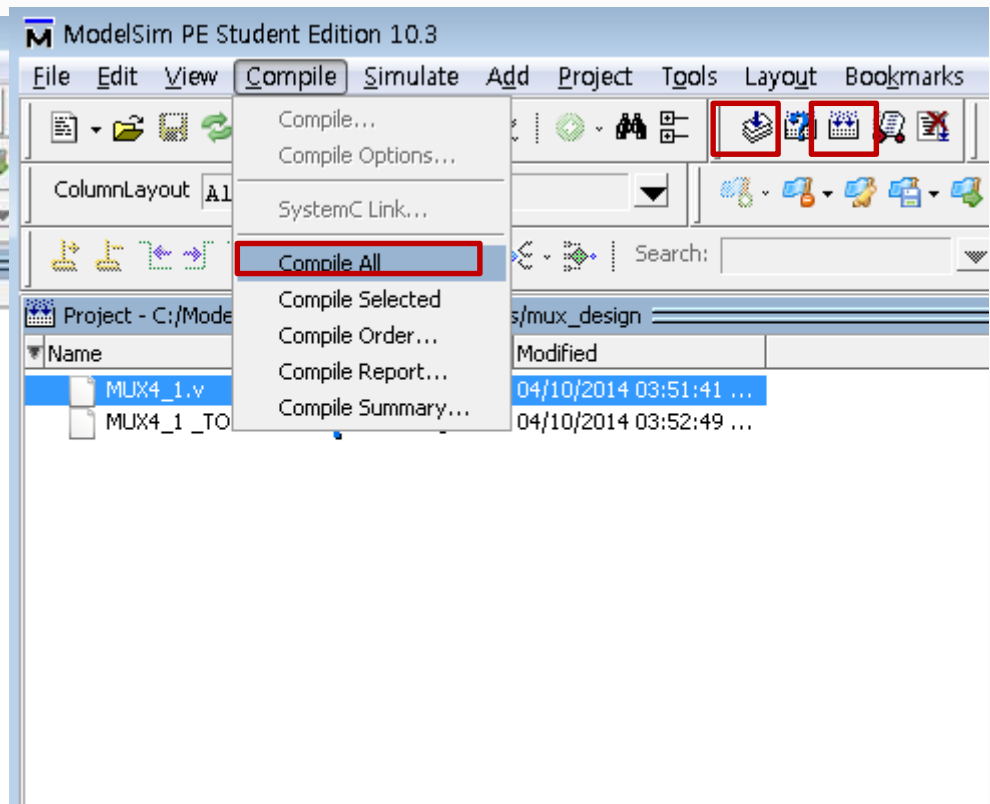
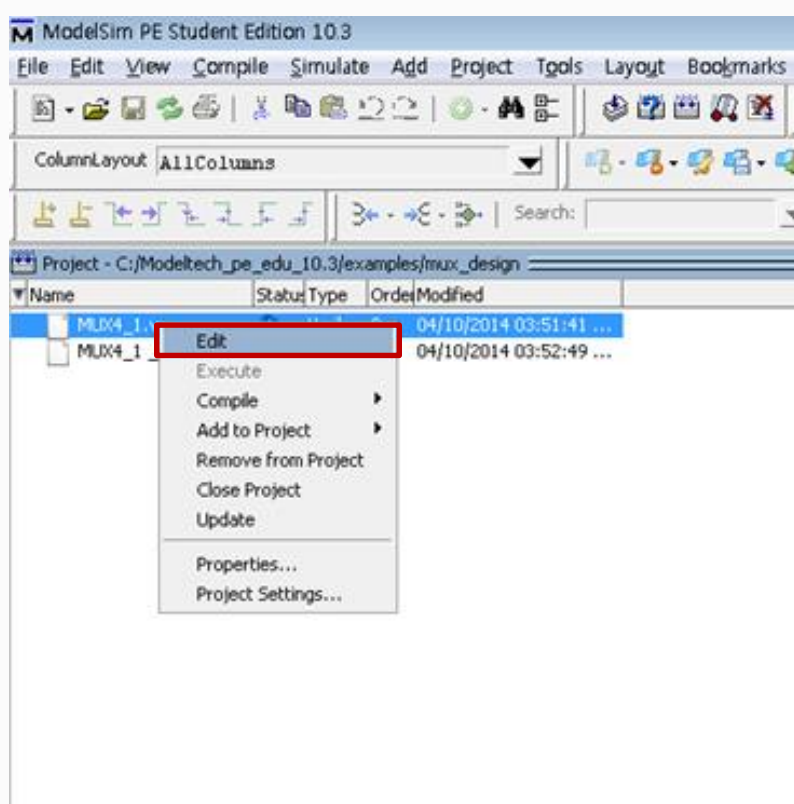
실험 내용 (cont.)

- ◆ Add Existing File → 제공된 'MUX4_1.v', 'MUX4_1_top.v' 를 Open
 - ◆ MUX4_1.v : MUX4_1 Module 을 구현한 파일
 - ◆ MUX4_1_top.v : MUX4_1 Module 을 Test 하기 위한 파일



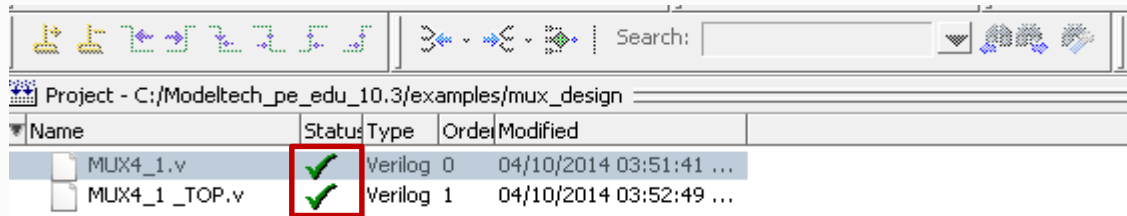
실험 내용 (cont.)

- ◆ 프로젝트에 추가된 파일 (*.v) 을 우클릭하고 'Edit' 를 선택하면 파일 내용을 확인 및 편집할 수 있음
- ◆ 'Compile' 또는 'Compile All' 을 선택하여 컴파일 수행



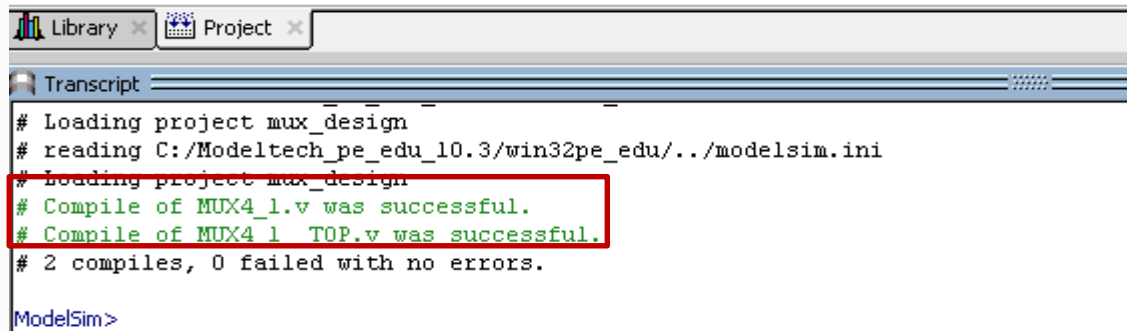
실험 내용 (cont.)

- ◆ 해당 파일에 에러가 없으면 ‘~ successful’ 이란 메시지가 뜨고, 에러가 있으면 ‘~ failed with n errors’ 란 메시지가 뜬다



The screenshot shows the ModelSim project window for 'C:/Modeltech_pe_edu_10.3/examples/mux_design'. It contains a table with columns: Name, Status, Type, Order, and Modified. Two files are listed: MUX4_1.v and MUX4_1_TOP.v. Both have a green checkmark in the Status column, indicating successful compilation. The Status column for both files is highlighted with a red box.

Name	Status	Type	Order	Modified
MUX4_1.v	✓	Verilog	0	04/10/2014 03:51:41 ...
MUX4_1_TOP.v	✓	Verilog	1	04/10/2014 03:52:49 ...



The screenshot shows the ModelSim Transcript window with the following text:

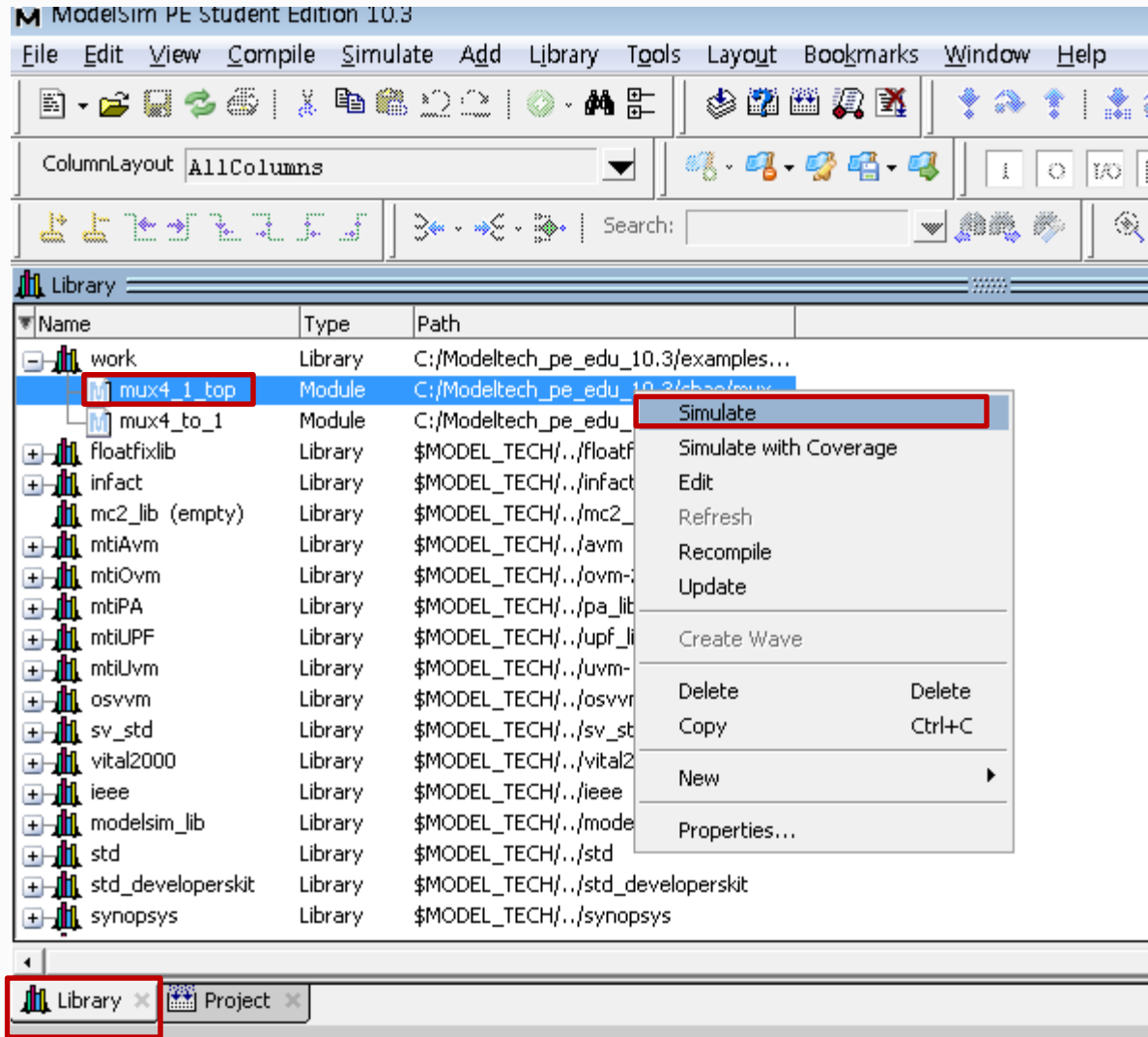
```
# Loading project mux_design
# reading C:/Modeltech_pe_edu_10.3/win32pe_edu/./modelsim.ini
# Loading project mux_design
# Compile of MUX4_1.v was successful.
# Compile of MUX4_1_TOP.v was successful.
# 2 compiles, 0 failed with no errors.

ModelSim>
```

The last two lines of the transcript are highlighted with a red box.

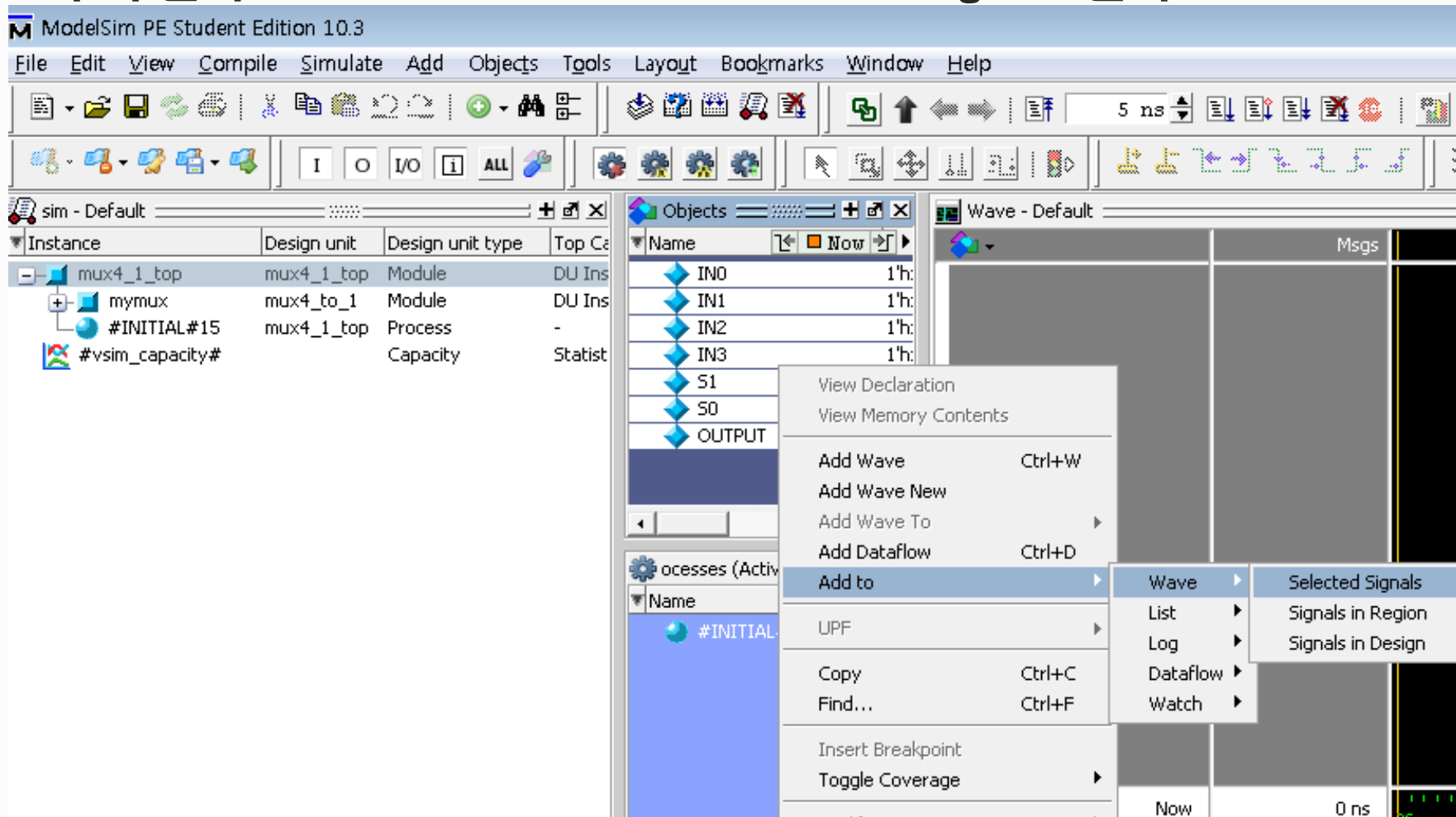
실험 내용 (cont.)

- Library 탭을 선택하고, work 폴더 안의 mux4_1_top 을 우클릭 → Simulate 를 클릭



실험 내용 (cont.)

- ◇ Sim 창에서 시뮬레이션 파형을 보고자 하는 design unit 선택
- ◇ Objects 창에 in/output, 내부 signal 이 뜸. 해당 signals 를 선택한 후 우클릭 → Add to → Wave → Selected Signals 선택



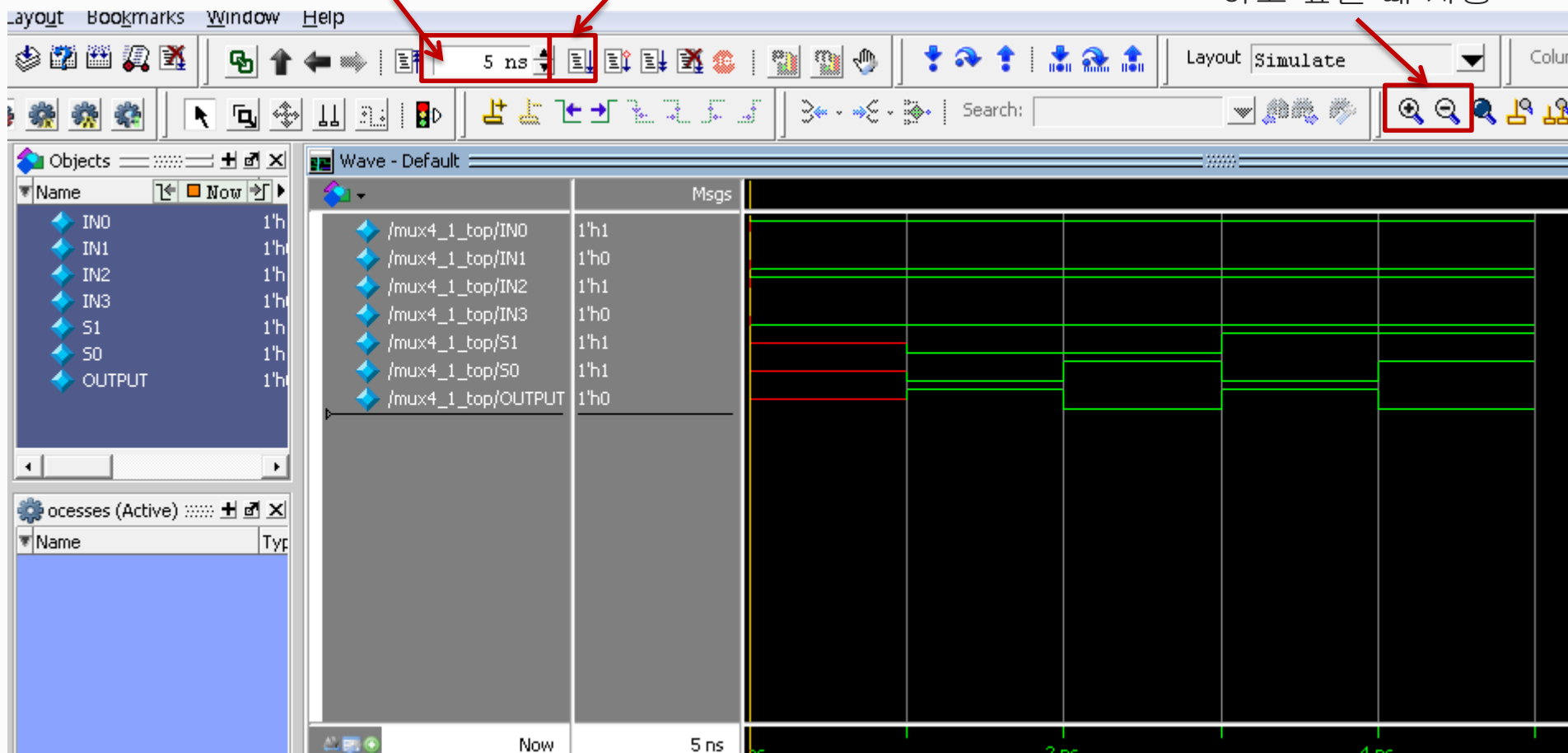
실험 내용 (cont.)

- ◆ Wave 창에 선택한 signals 가 추가된 것을 확인
- ◆ 시뮬레이션을 수행할 시간을 설정 (e.g., 5 ns) 하고 run 클릭

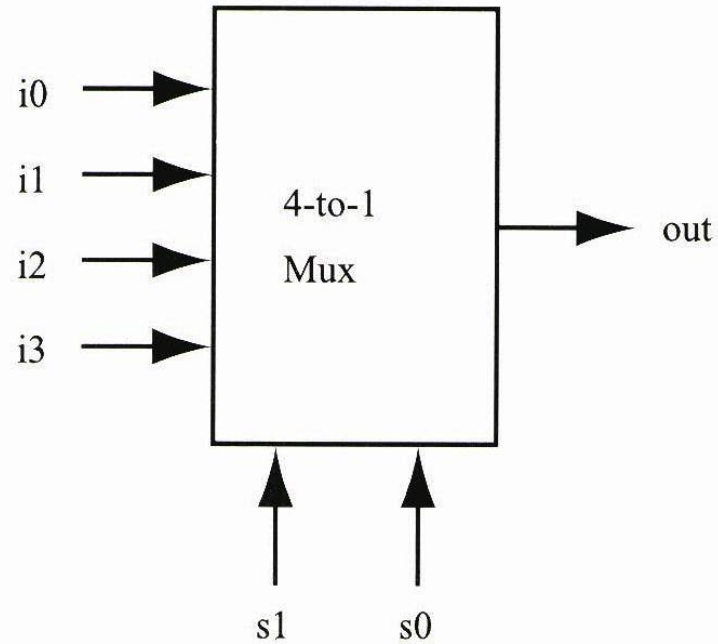
수행시간 설정

run

Waveform을 zoom-in/out
하고 싶을 때 사용

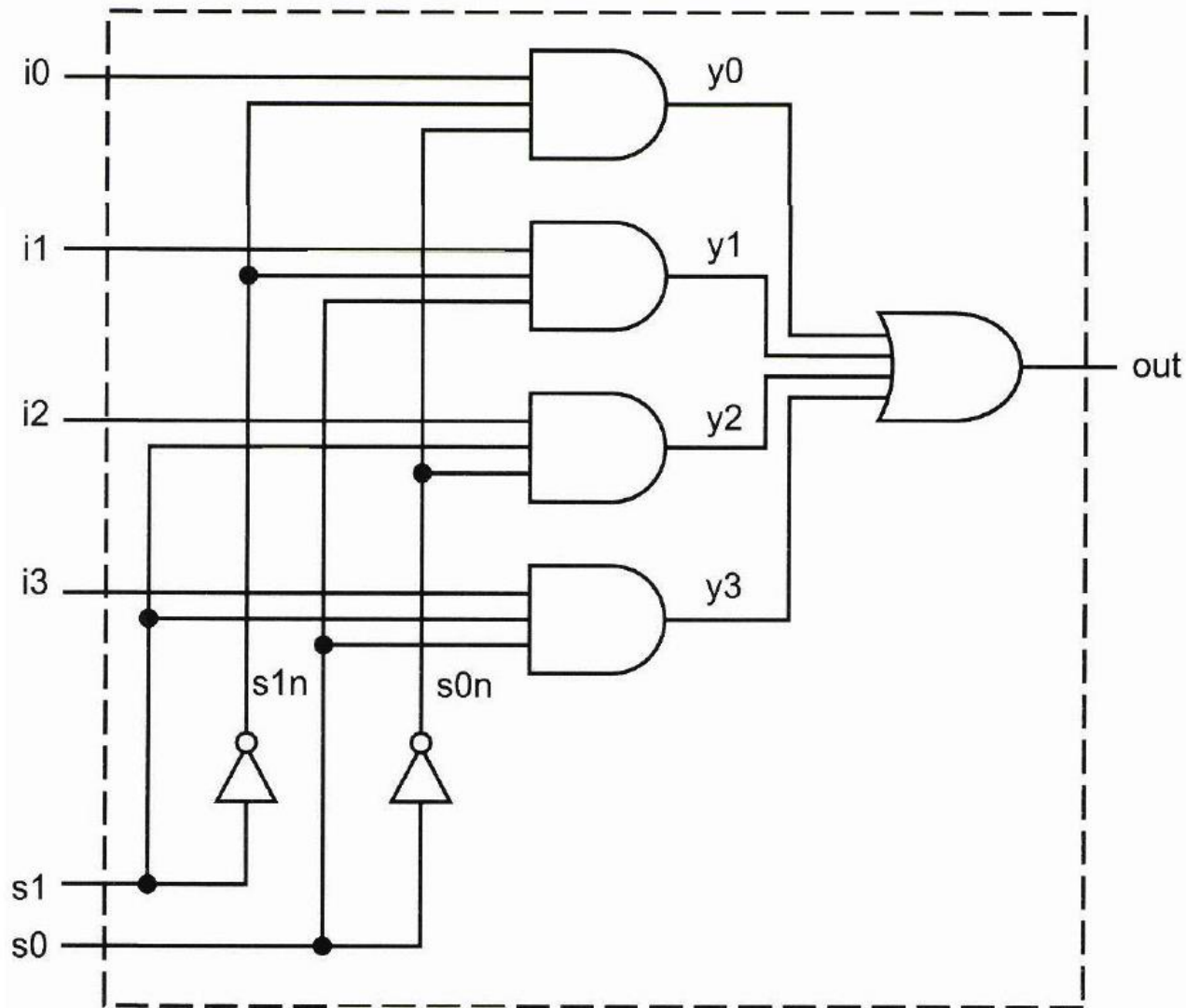


4:1 Multiplexer



s_1	s_0	out
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Gate 수준의 4:1 Multiplexer



Gate 수준의 4:1 Multiplexer Verilog

: MUX4_1.v

// 4:1 멀티플렉서 모듈. 포트 리스트는 I/O 다이어그램에서 직관적으로 알 수 있다.

```
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);
```

// I/O 다이어그램으로부터 포트를 선언.

```
output out;
```

```
input i0, i1, i2, i3;
```

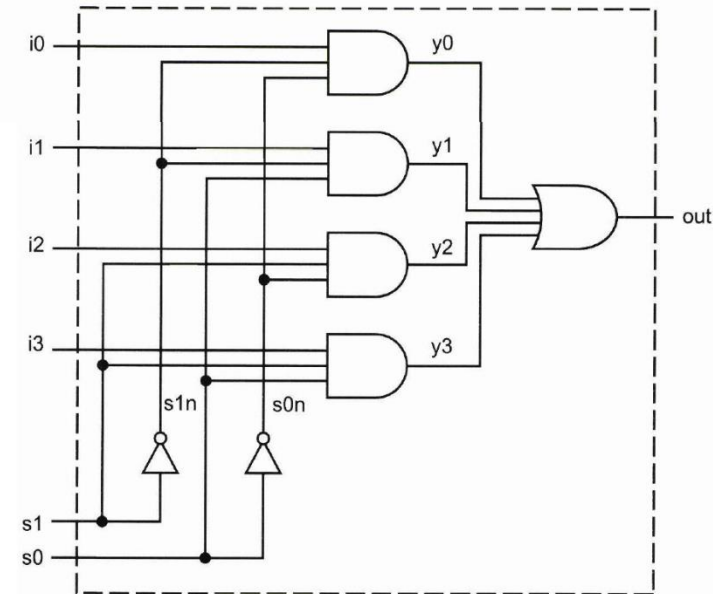
```
input s1, s0;
```

// 내부 wire 선언.

```
wire s1n, s0n;
```

```
wire y0, y1, y2, y3;
```

Module 시작!
필요한 in/out 포트 이름
모두 적기



Gate 수준의 4:1 Multiplexer Verilog

: MUX4_1.v

// 4:1 멀티플렉서 모듈. 포트 리스트는 I/O 다이어그램에서 직관적으로 알 수 있다.

```
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);
```

// I/O 다이어그램으로부터 포트를 선언.

```
output out;
```

```
input i0, i1, i2, i3;
```

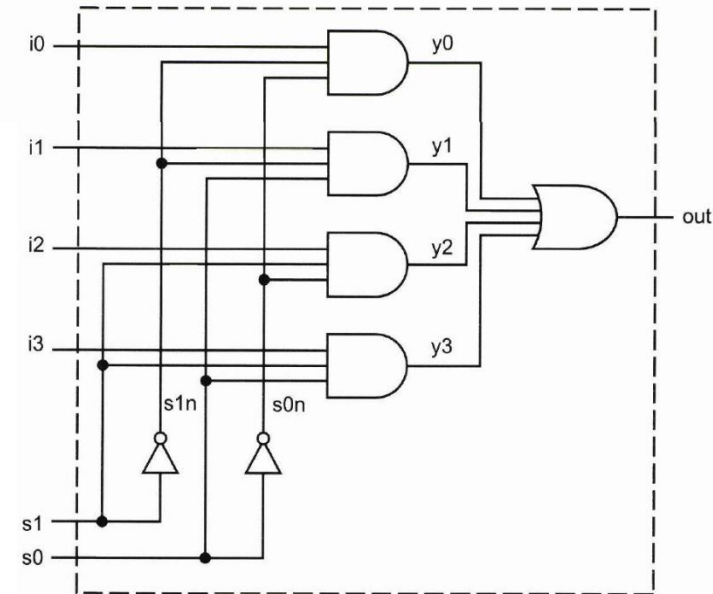
```
input s1, s0;
```

// 내부 wire 선언.

```
wire s1n, s0n;
```

```
wire y0, y1, y2, y3;
```

Input인지 output인지
구별해 주기



Gate 수준의 4:1 Multiplexer Verilog

: MUX4_1.v

// 4:1 멀티플렉서 모듈. 포트 리스트는 I/O 다이어그램에서 직관적으로 알 수 있다.

```
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);
```

// I/O 다이어그램으로부터 포트를 선언.

```
output out;
```

```
input i0, i1, i2, i3;
```

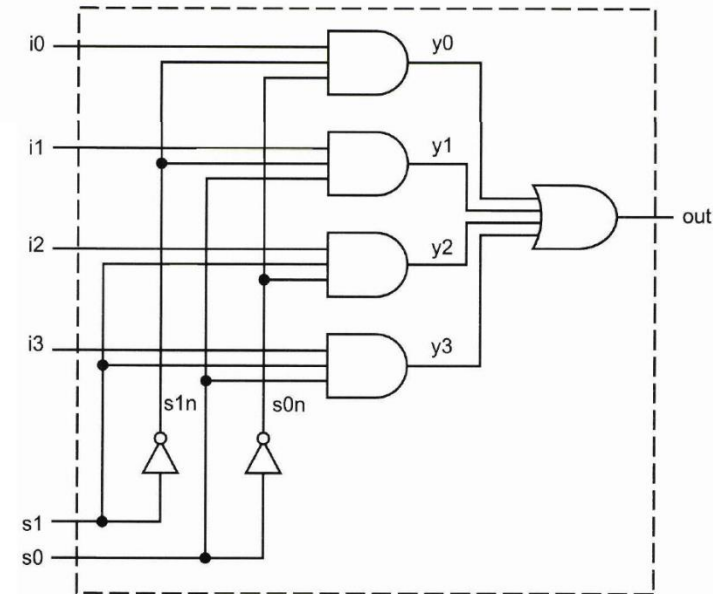
```
input s1, s0;
```

```
// 내부 wire 선언.
```

```
wire s1n, s0n;
```

```
wire y0, y1, y2, y3;
```

중간 연결에 필요한
wire들 마련해 두기



Gate 수준의 4:1 Multiplexer Verilog

: MUX4_1.v

```
// 게이트 파생. s1n, s0n 신호를 생성.
```

```
not (s1n, s1);
```

```
not (s0n, s0);
```

```
// 3-input AND 게이트 파생
```

```
and (y0, i0, s1n, s0n);
```

```
and (y1, i1, s1n, s0);
```

```
and (y2, i2, s1, s0n);
```

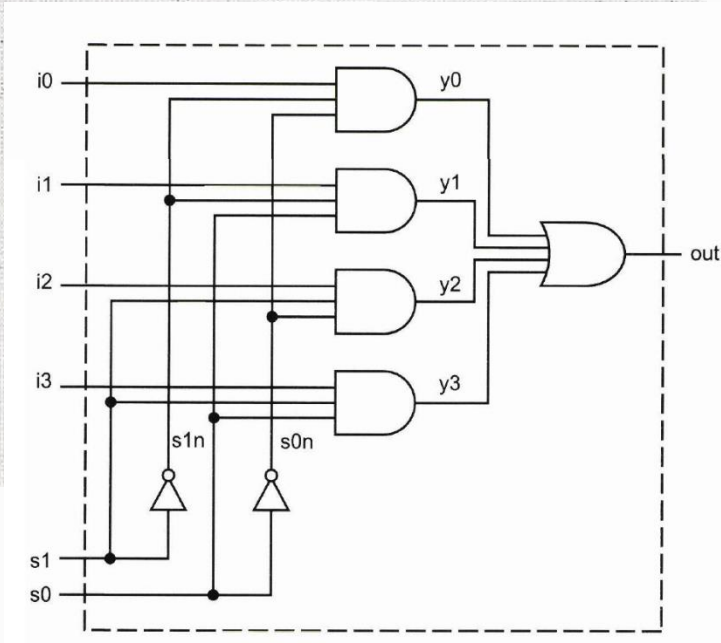
```
and (y3, i3, s1, s0);
```

```
// 4개의 입력을 갖는 OR 게이트 파생
```

```
or (out, y0, y1, y2, y3);
```

```
endmodule
```

not (out, in)



Gate 수준의 4:1 Multiplexer Verilog

: MUX4_1.v

```
// 게이트 파생. s1n, s0n 신호를 생성.
```

```
not (s1n, s1);
```

```
not (s0n, s0);
```

```
// 3-input AND 게이트 파생
```

```
and (y0, i0, s1n, s0n);
```

```
and (y1, i1, s1n, s0);
```

```
and (y2, i2, s1, s0n);
```

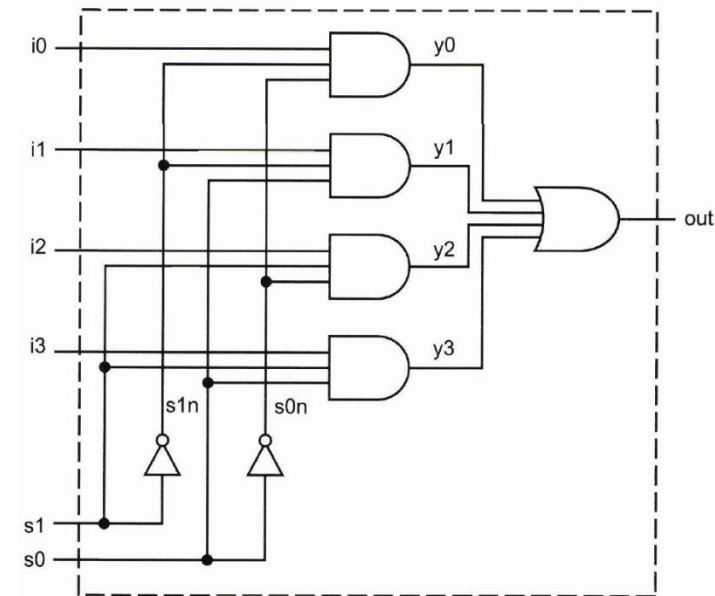
```
and (y3, i3, s1, s0);
```

```
and (out, in1, in2, in3)
```

```
// 4개의 입력을 갖는 OR 게이트 파생
```

```
or (out, y0, y1, y2, y3);
```

```
endmodule
```



Gate 수준의 4:1 Multiplexer Verilog

: MUX4_1.v

```
// 게이트 파생. s1n, s0n 신호를 생성.
```

```
not (s1n, s1);
```

```
not (s0n, s0);
```

```
// 3-input AND 게이트 파생
```

```
and (y0, i0, s1n, s0n);
```

```
and (y1, i1, s1n, s0);
```

```
and (y2, i2, s1, s0n);
```

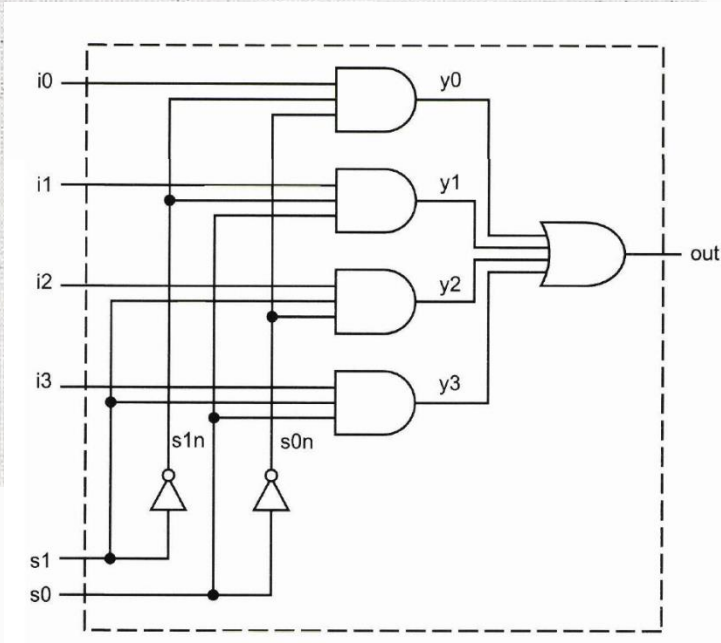
```
and (y3, i3, s1, s0);
```

```
// 4개의 입력을 갖는 OR 게이트 파생
```

```
or (out, y0, y1, y2, y3);
```

```
endmodule
```

or (out, in1, in2, in3, in4)



Gate 수준의 4:1 Multiplexer Verilog

: MUX4_1.v

```
// 게이트 파생. s1n, s0n 신호를 생성.
```

```
not (s1n, s1);
```

```
not (s0n, s0);
```

```
// 3-input AND 게이트 파생
```

```
and (y0, i0, s1n, s0n);
```

```
and (y1, i1, s1n, s0);
```

```
and (y2, i2, s1, s0n);
```

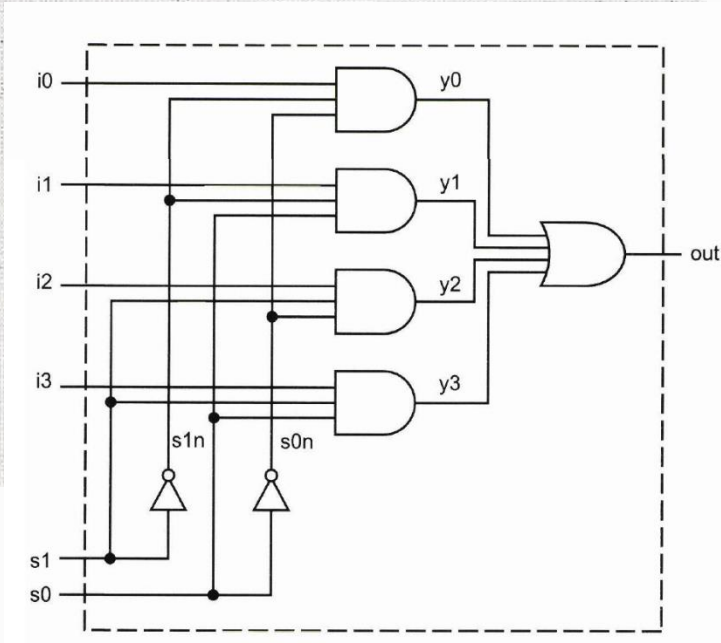
```
and (y3, i3, s1, s0);
```

```
// 4개의 입력을 갖는 OR 게이트 파생
```

```
or (out, y0, y1, y2, y3);
```

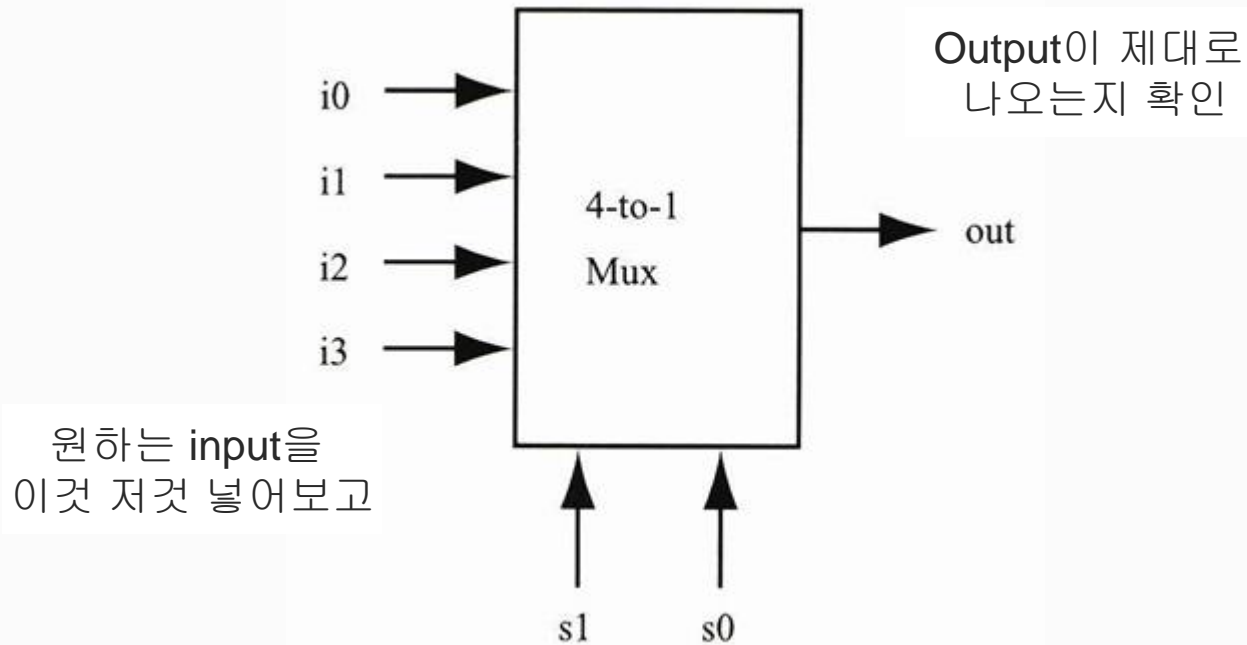
```
endmodule
```

모듈 끝났다고 알려주기



4:1 Multiplexer Verilog 테스트하기

테스트를 위한 모듈



Test Code

: MUX4_1_top.v

```
// 스티뮬러스 모듈을 정의 (포트가 없다).
```

```
module stimulus;
```

포트가 없으면 테스트용 코드라는 것을 알 수 있다.

```
// 입력으로 연결되는 변수들을 정의.
```

```
reg IN0, IN1, IN2, IN3;
```

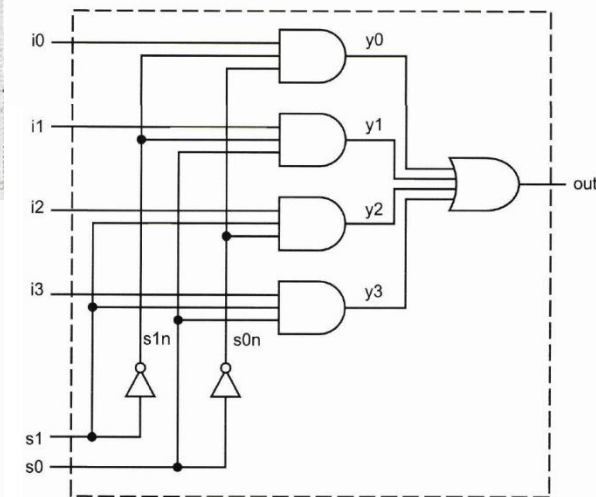
```
reg S1, S0;
```

```
// 출력 wire의 선언.
```

```
wire OUTPUT;
```

```
// 멀티플렉서의 파생.
```

```
mux4_to_1 mymux (OUTPUT, IN0, IN1, IN2, IN3, S1, S0);
```



Test Code

: MUX4_1_top.v

```
// 스티뮬러스 모듈을 정의 (포트가 없다).
```

```
module stimulus;
```

```
// 입력으로 연결되는 변수들을 정의.
```

```
reg IN0, IN1, IN2, IN3;
```

```
reg S1, S0;
```

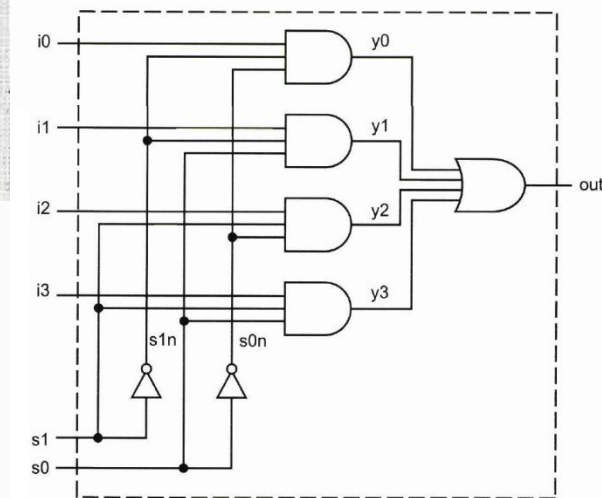
```
// 출력 wire의 선언.
```

```
wire OUTPUT;
```

```
// 멀티플렉서의 파생.
```

```
mux4_to_1 mymux (OUTPUT, IN0, IN1, IN2, IN3, S1, S0);
```

6개의 입력에 이것 저것 넣어볼 수 있도록
레지스터 (저장공간) 선언



Test Code

: MUX4_1_top.v

```
// 스티뮬러스 모듈을 정의 (포트가 없다).
```

```
module stimulus;
```

```
// 입력으로 연결되는 변수들을 정의.
```

```
reg IN0, IN1, IN2, IN3;
```

```
reg S1, S0;
```

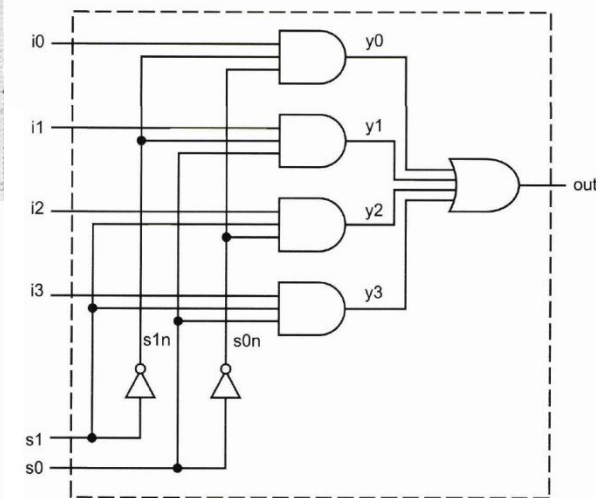
```
// 출력 wire의 선언.
```

```
wire OUTPUT;
```

```
// 멀티플렉서의 파생.
```

```
mux4_to_1 mymux (OUTPUT, IN0, IN1, IN2, IN3, S1, S0);
```

출력은 단순히 확인만 하면 되니까
Wire 연결해 둔다.



Test Code

: MUX4_1_top.v

```
// 스티뮬러스 모듈을 정의 (포트가 없다).
```

```
module stimulus;
```

```
// 입력으로 연결되는 변수들을 정의.
```

```
reg IN0, IN1, IN2, IN3;
```

```
reg S1, S0;
```

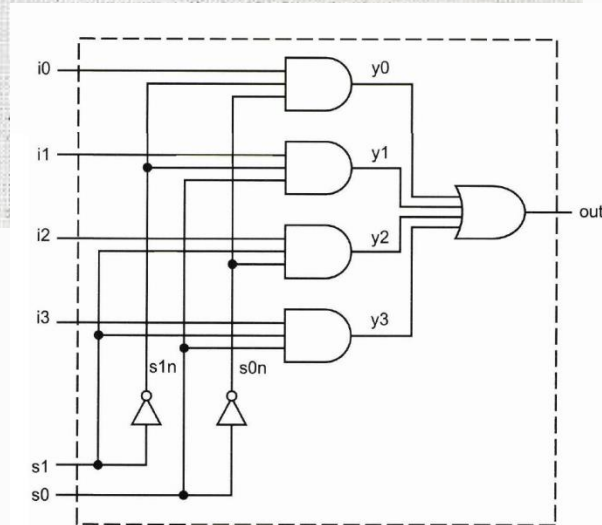
```
// 출력 wire의 선언.
```

```
wire OUTPUT;
```

```
// 멀티플렉서의 피생. instance 이름
```

```
mux4_to_1 mymux (OUTPUT, IN0, IN1, IN2, IN3, S1, S0);
```

내가 정의한
module 이름



Test Code

: MUX4_1_top.v

```
// 스티물러스 모듈을 정의 (포트가 없다).
```

```
module stimulus;
```

```
// 입력으로 연결되는 변수들을 정의.
```

```
reg IN0, IN1, IN2, IN3;
```

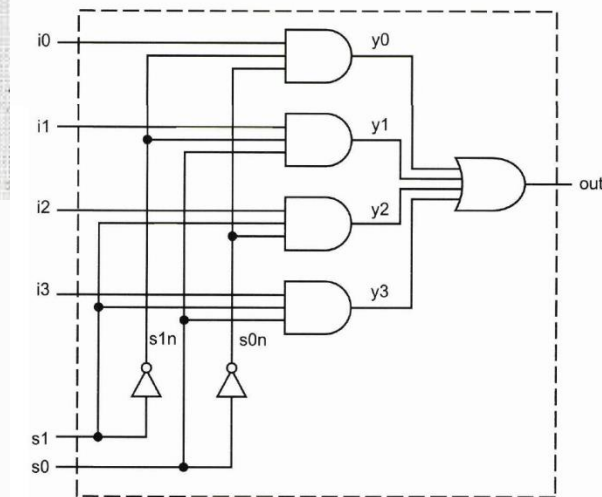
```
reg S1, S0;
```

```
// 출력 wire의 선언.
```

```
wire OUTPUT;
```

```
// 멀티플렉서의 파생.
```

```
mux4_to_1 mymux (OUTPUT, IN0, IN1, IN2, IN3, S1, S0);
```



Test Code

: MUX4_1_top.v

```
// 입력 스티물러스 생성, 스티물러스 모듈 정의(포트 없음).
```

```
initial          Main 함수 처럼 simulation 시작하자마자 initial에서 출발한다.  
begin
```

```
    // 입력 라인을 셋
```

```
    IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
```

```
    #1 $display("IN0= %b, IN1= %b, IN2= %b, IN3= %b\n", IN0, IN1, IN2, IN3);
```

입력 레지스터에 1, 0, 1, 0을 저장하고 1ns 이후에 확인해보기

```
    // IN0를 선택
```

```
    S1 = 0; S0 = 0;
```

```
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);
```

```
    // IN1을 선택
```

```
    S1 = 0; S0 = 1;
```

```
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);
```

```
    // IN2를 선택
```

```
    S1 = 1; S0 = 0;
```


Test Code

: MUX4_1_top.v

```
// 입력 스티물러스 생성, 스티물러스 모듈 정의(포트 없음).
```

```
initial
```

Main 함수 처럼 **simulation** 시작하자마자 **initial**에서 출발한다.

```
begin
```

```
// 입력 라인을 셋
```

```
IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
```

```
#1 $display("IN0= %b, IN1= %b, IN2= %b, IN3= %b\n", IN0, IN1, IN2, IN3);
```

```
// IN0를 선택
```

```
S1 = 0; S0 = 0;
```

```
#1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);
```

입력 레지스터에 **s**에 **0,0**을 저장한 후 **1ns** 이후에 **output** 값을 확인해 보기

```
// IN1을 선택
```

```
S1 = 0; S0 = 1;
```

```
#1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);
```

```
// IN2를 선택
```

```
S1 = 1; S0 = 0;
```


Test Code

: MUX4_1_top.v

```
// 입력 스티물러스 생성, 스티물러스 모듈 정의(포트 없음).
```

```
initial
```

Main 함수 처럼 **simulation** 시작하자마자 **initial**에서 출발한다.

```
begin
```

```
// 입력 라인을 셋
```

```
IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
```

```
#1 $display("IN0= %b, IN1= %b, IN2= %b, IN3= %b\n", IN0, IN1, IN2, IN3);
```

```
// IN0를 선택
```

```
S1 = 0; S0 = 0;
```

```
#1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);
```

```
// IN1을 선택
```

```
S1 = 0; S0 = 1;
```

```
#1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);
```

입력 레지스터에 s에 0,1을 저장한 후 1ns 이후에 output 값을 확인해 보기

```
// IN2를 선택
```

```
S1 = 1; S0 = 0;
```


Test Code

: MUX4_1_top.v

```
// IN3을 선택
```

```
S1 = 1; S0 = 1;
```

```
#1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);
```

```
end
```

입력 레지스터에 **s**에 **0,1**을 저장한 후 **1ns** 이후에 **output** 값을 확인해 보기

```
endmodule
```

Test Code

: MUX4_1_top.v

Initial begin

```
// 테스트하고 싶은 값을  
//이것 저것 넣어보기  
//테스트 다 했으면  
end
```

```
// IN3을 선택
```

```
S1 = 1; S0 = 1;
```

```
#1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);
```

```
end Initial begin 의 내용이 끝났음을 표시
```

```
endmodule 테스트 모듈의  
          끝을 표시
```

