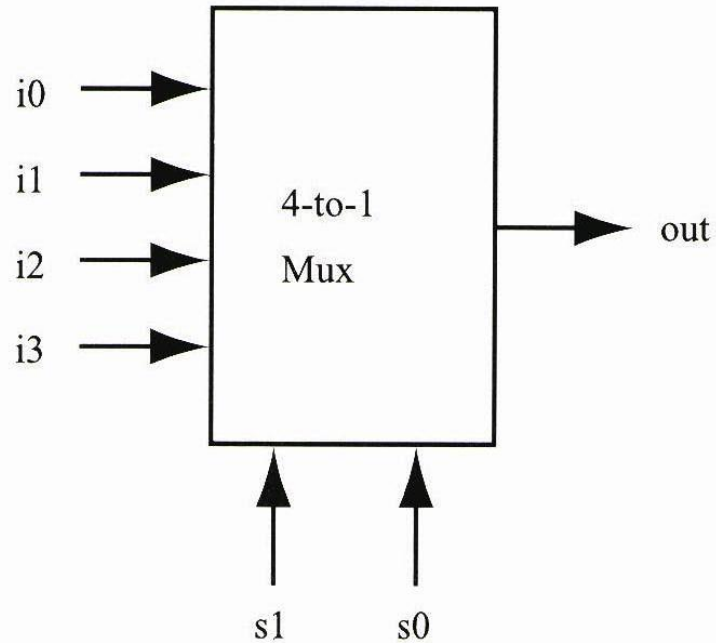


VIVADO로 MUX simulation 하기

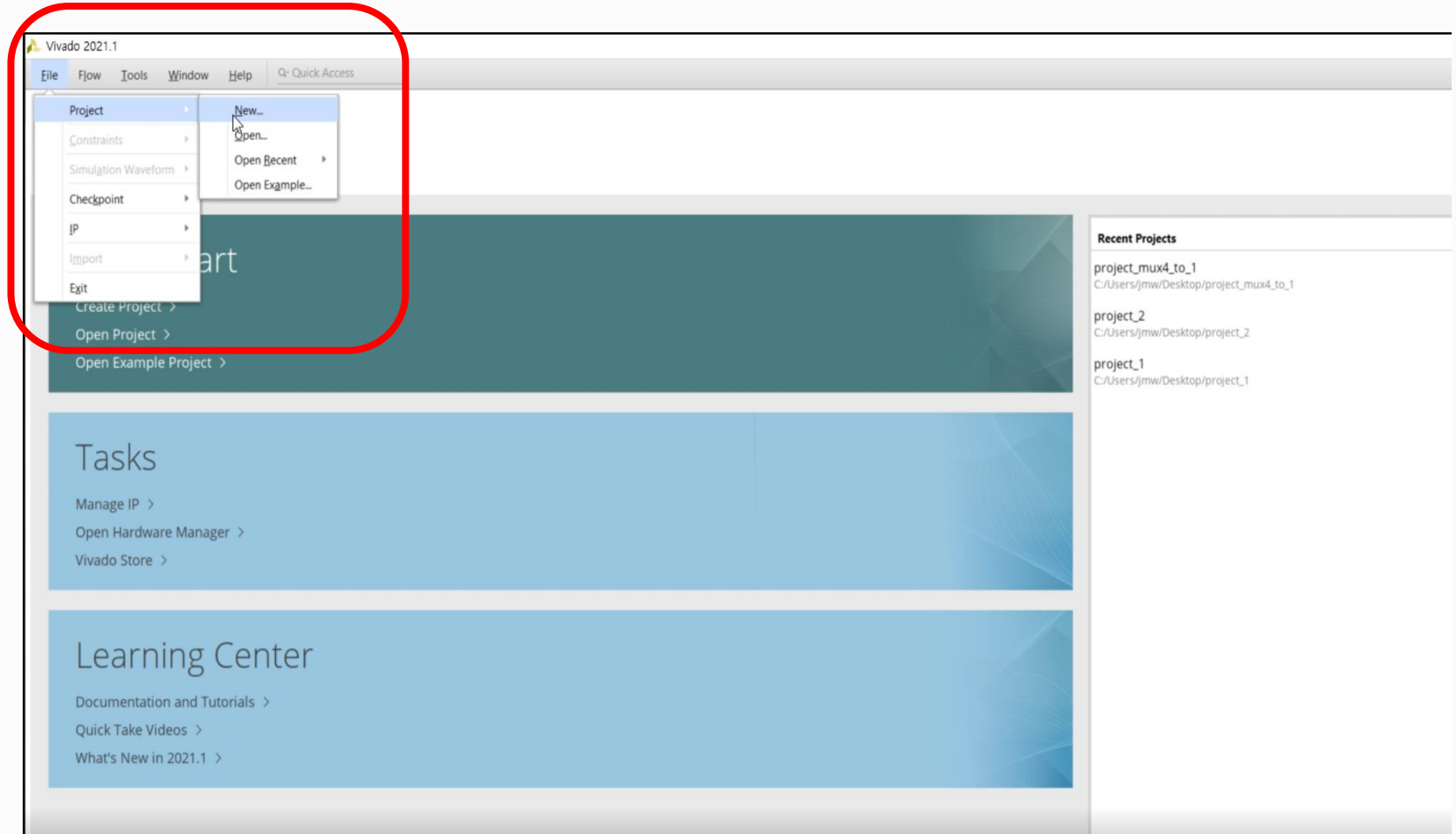
4:1 Multiplexer 설계 및 시뮬레이션



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

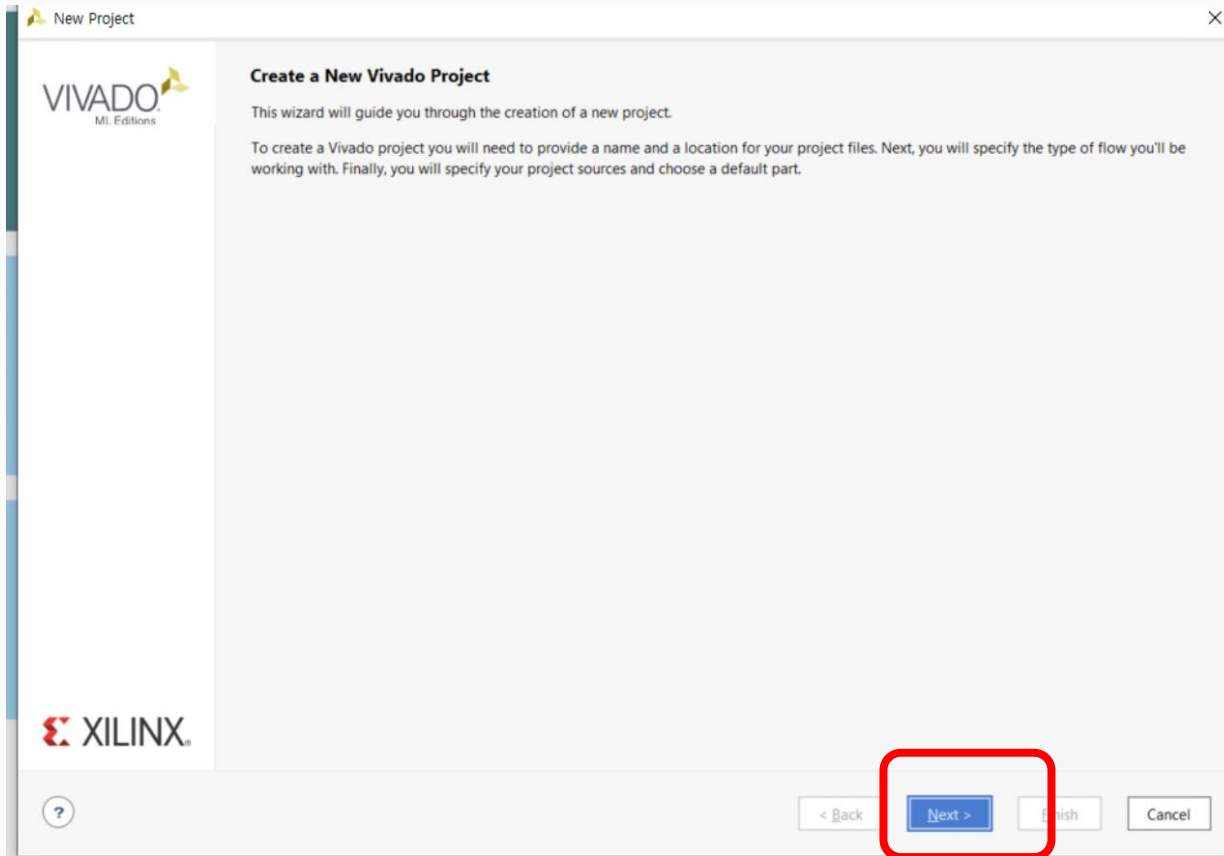
실험 내용

- ◇ VIVADO 을 실행시키고 새 프로젝트를 생성
→ 프로젝트 이름 한글 사용하지 말 것



실험 내용

- ◇ VIVADO 을 실행시키고 새 프로젝트를 생성
→ Next



실험 내용

- ◇ VIVADO 을 실행시키고 새 프로젝트를 생성
→ 프로젝트 이름 한글 사용하지 말 것

New Project

Project Name
Enter a name for your project and specify a directory where the project data files will be stored.

Project name:

Project location:

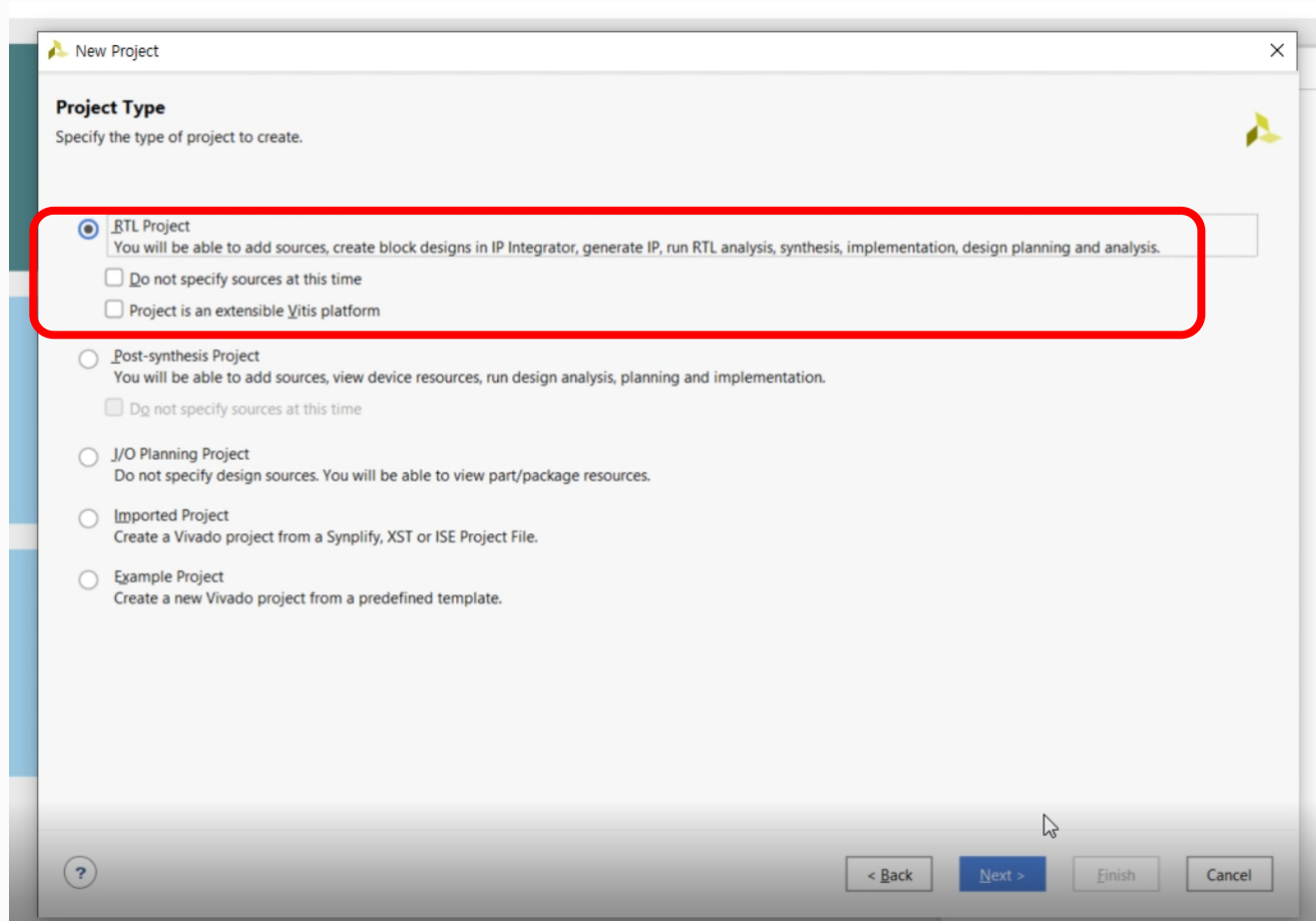
☒ Create project subdirectory

Project will be created at: C:/Users/jmw/Desktop/project_mux

< Back **Next >** Finish Cancel

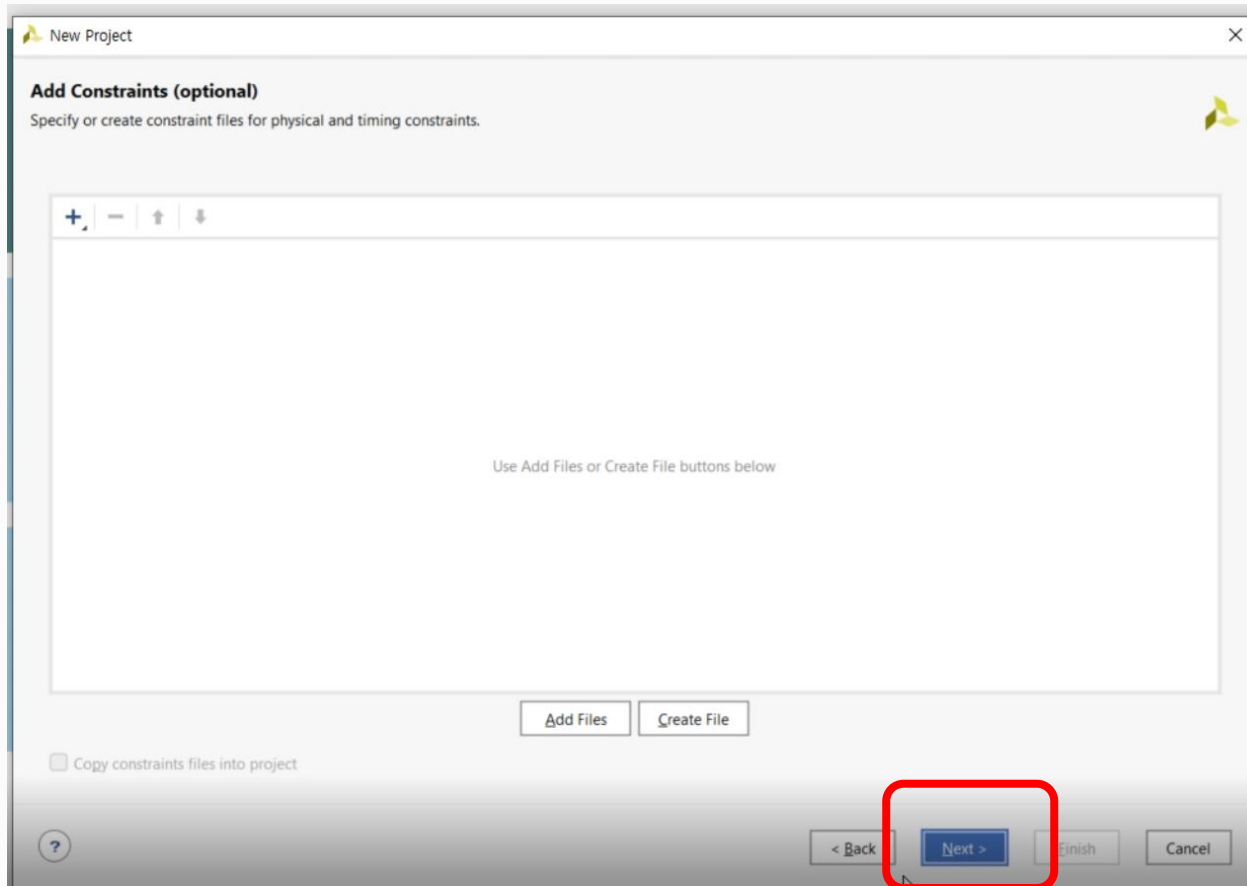
실험 내용

- ◇ VIVADO 을 실행시키고 새 프로젝트를 생성
- ◇ RTL project 체크 → Next



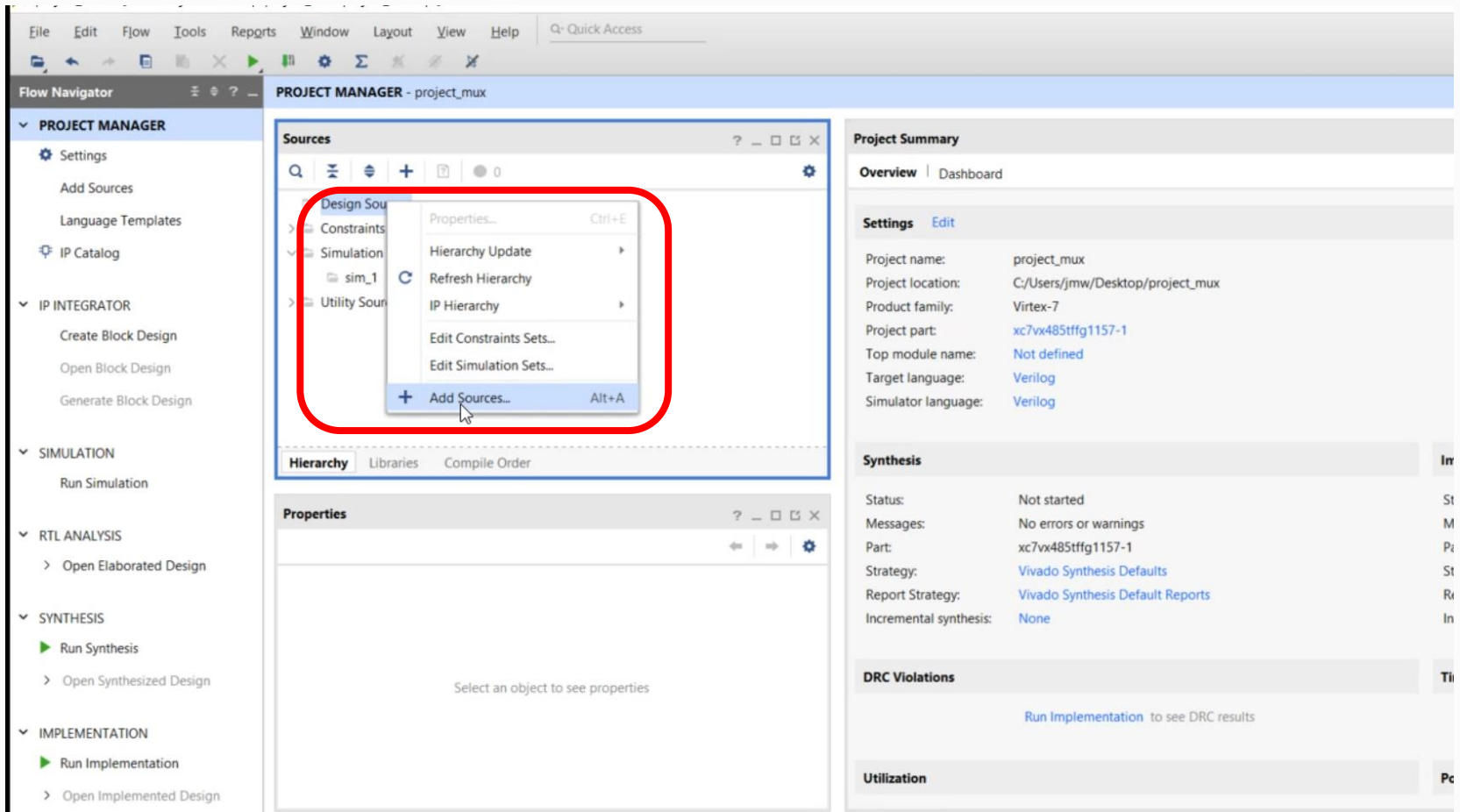
실험 내용

◇ NEXT



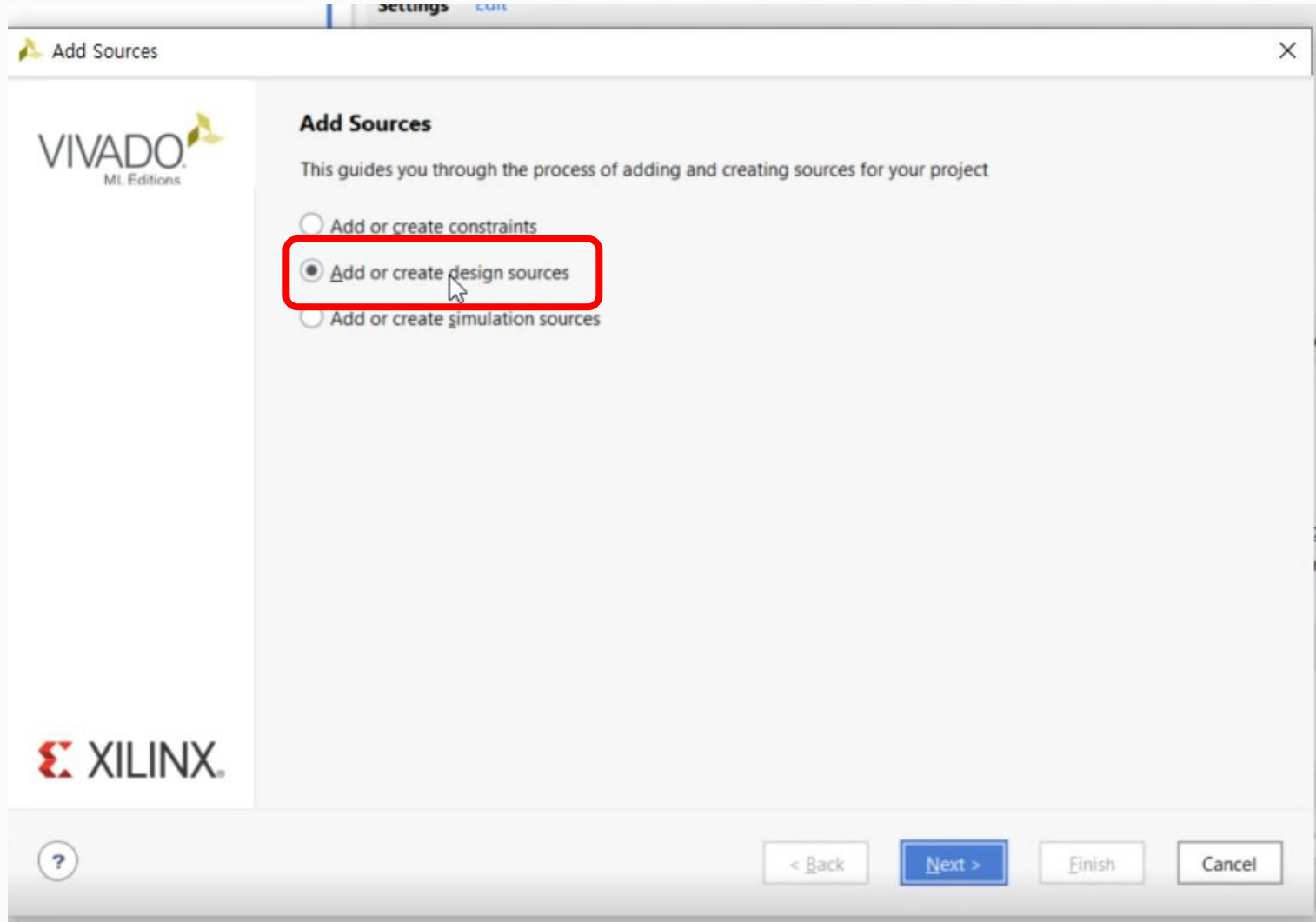
실험 내용

- ◇ Mux4_to_1.v 생성
- ◇ Design sources 우클릭 -> Add Sources..



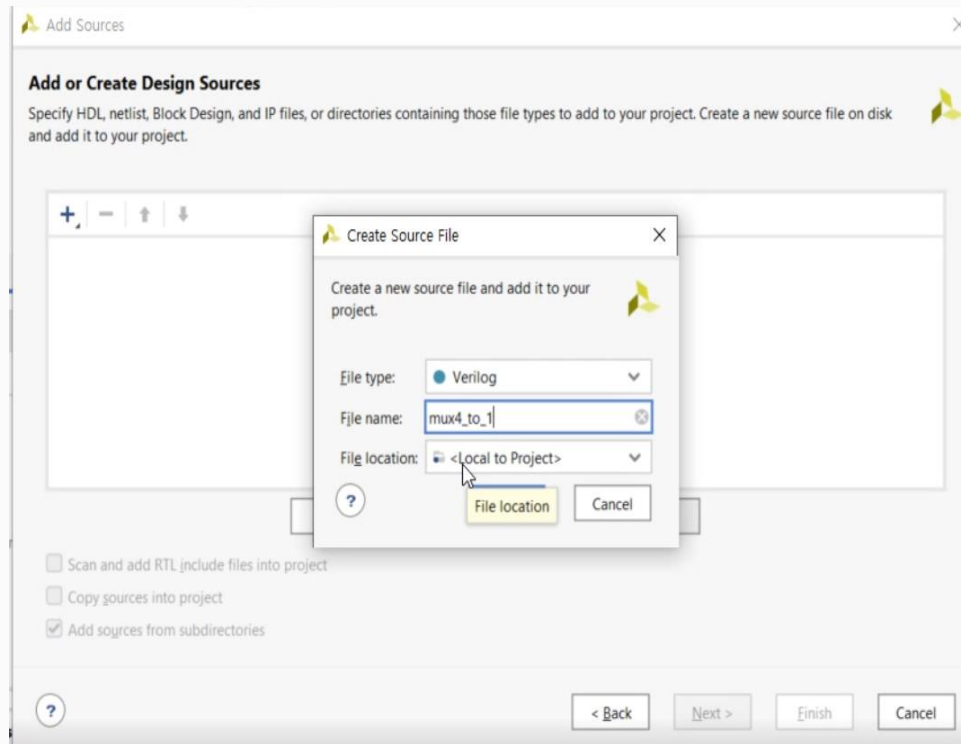
실험 내용

- ◆ Add or create design sources 체크 → Next



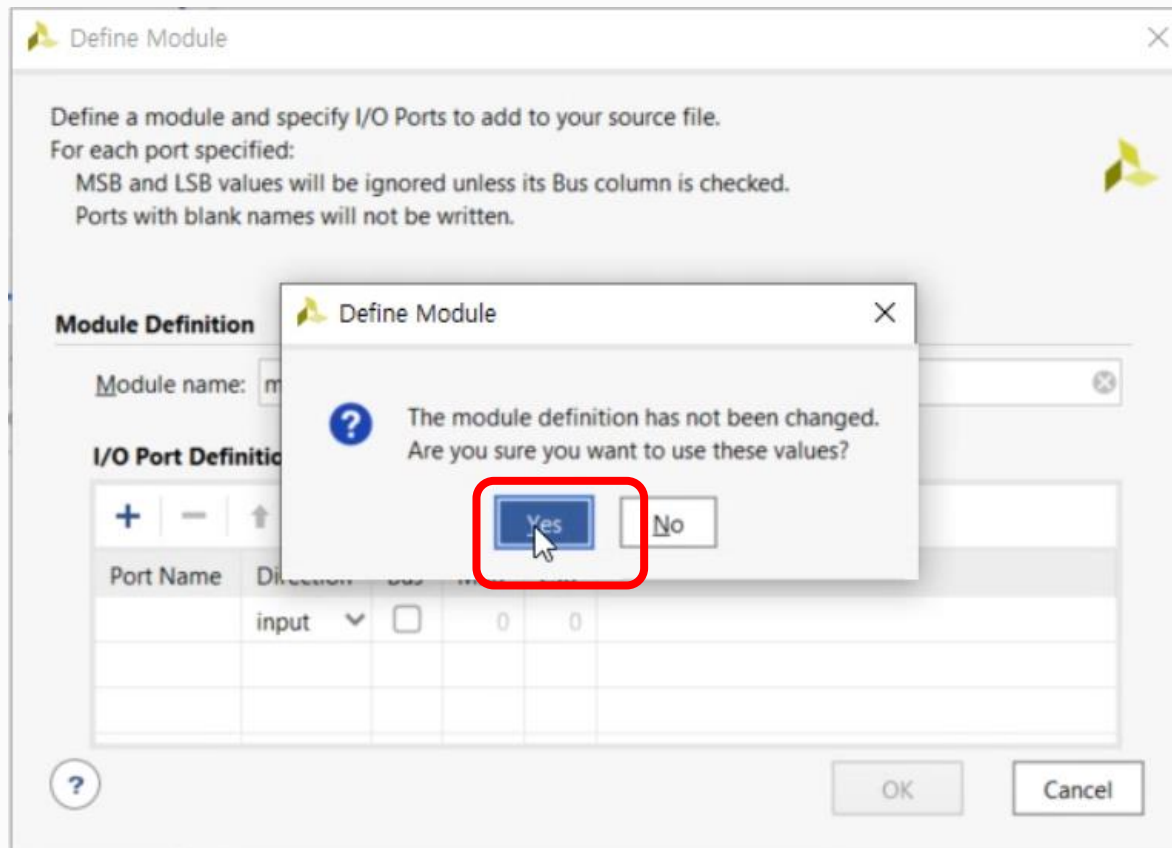
실험 내용

Create File → File name [mux4_to_1] 입력 → ok



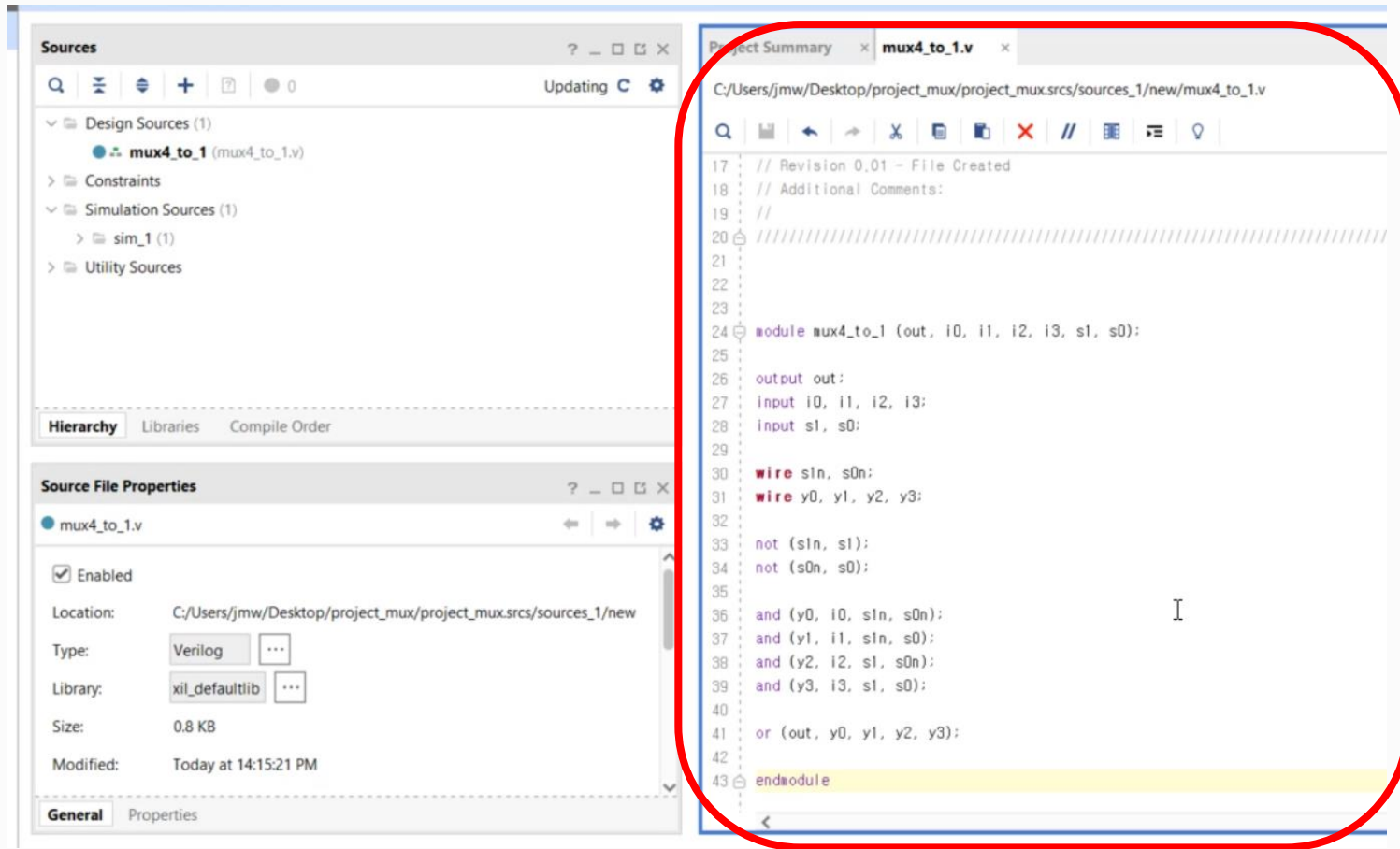
실험 내용

◆ YES



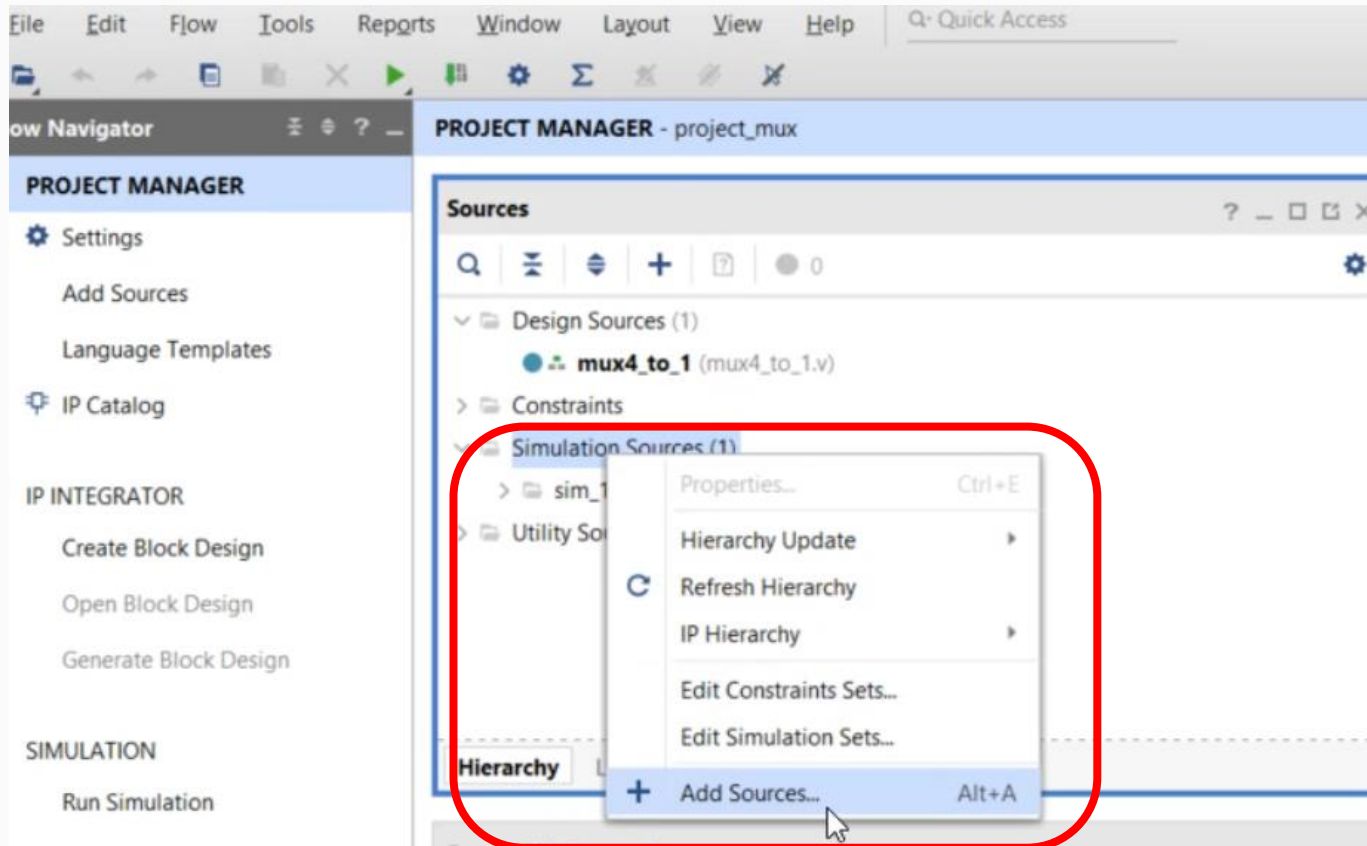
실험 내용

◇ Mux4_to_1.v 작성 후 저장(ctrl+s)



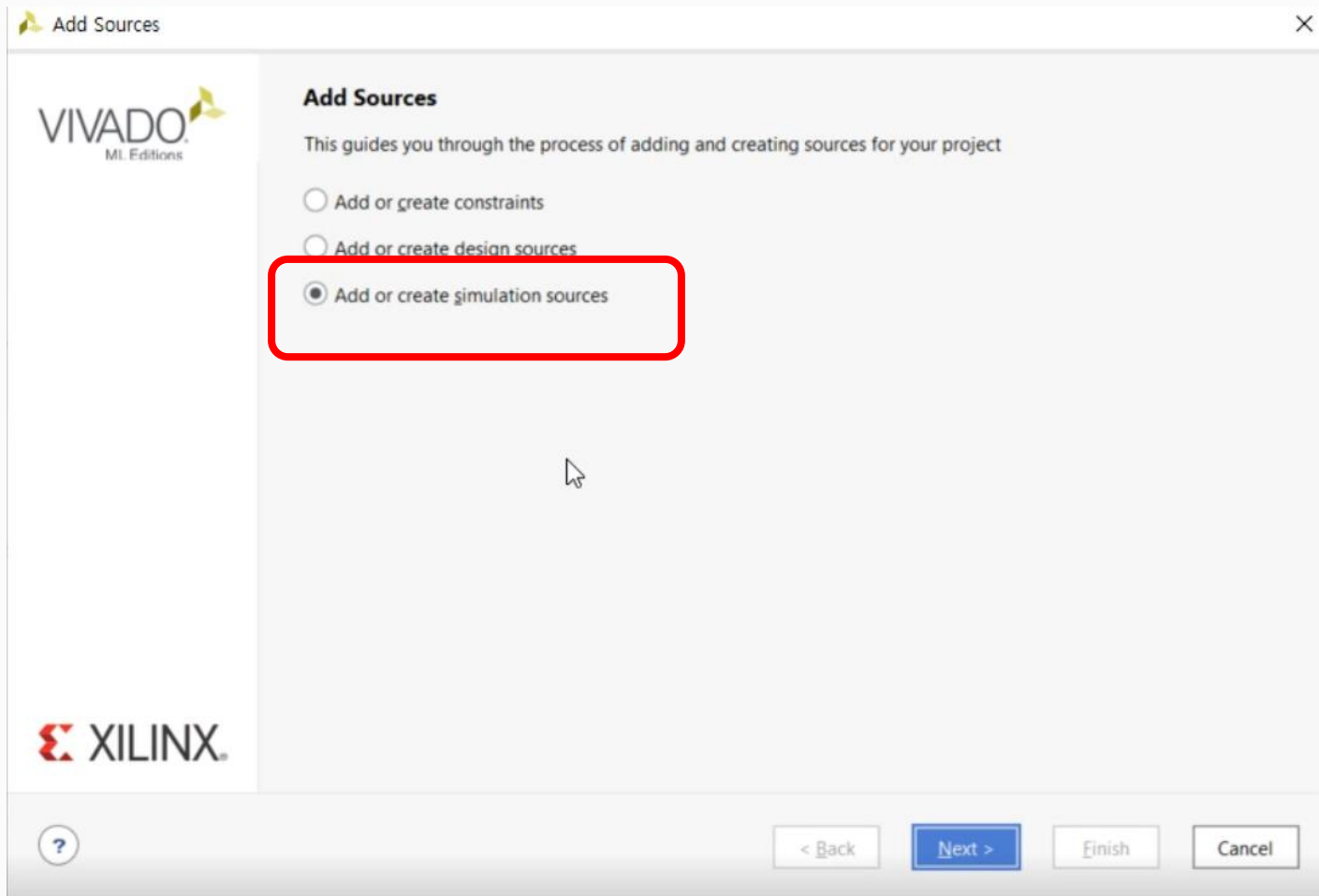
실험 내용

- ◇ Mux4_to_1_tb.v 생성
- ◇ Simulation sources 우클릭 → Add Sources..



실험 내용

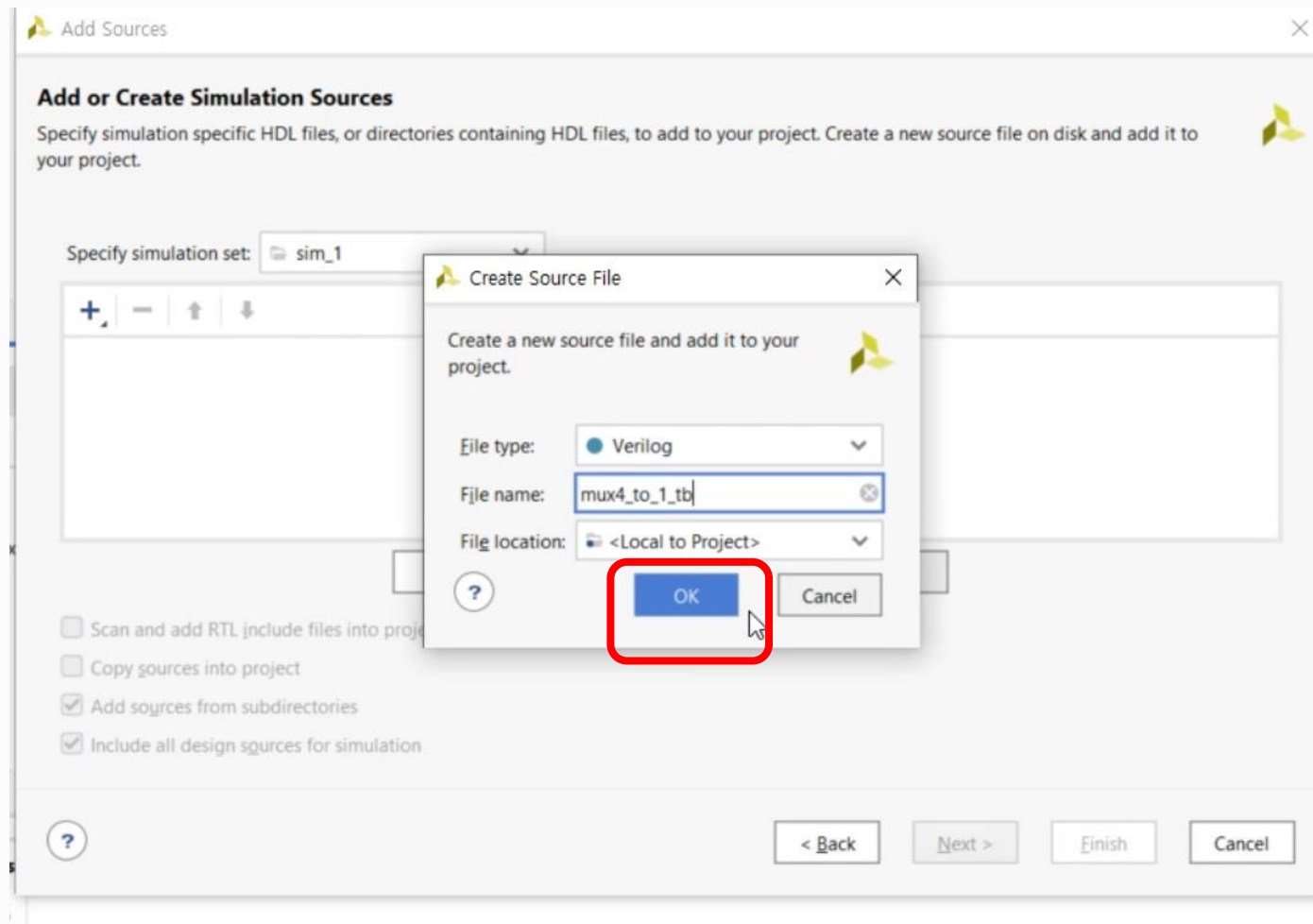
- ◇ Mux4_to_1_tb.v 생성
- ◇ Add or create Simulation sources 체크 → Next



실험 내용

◇ Mux4_to_1_tb.v 생성

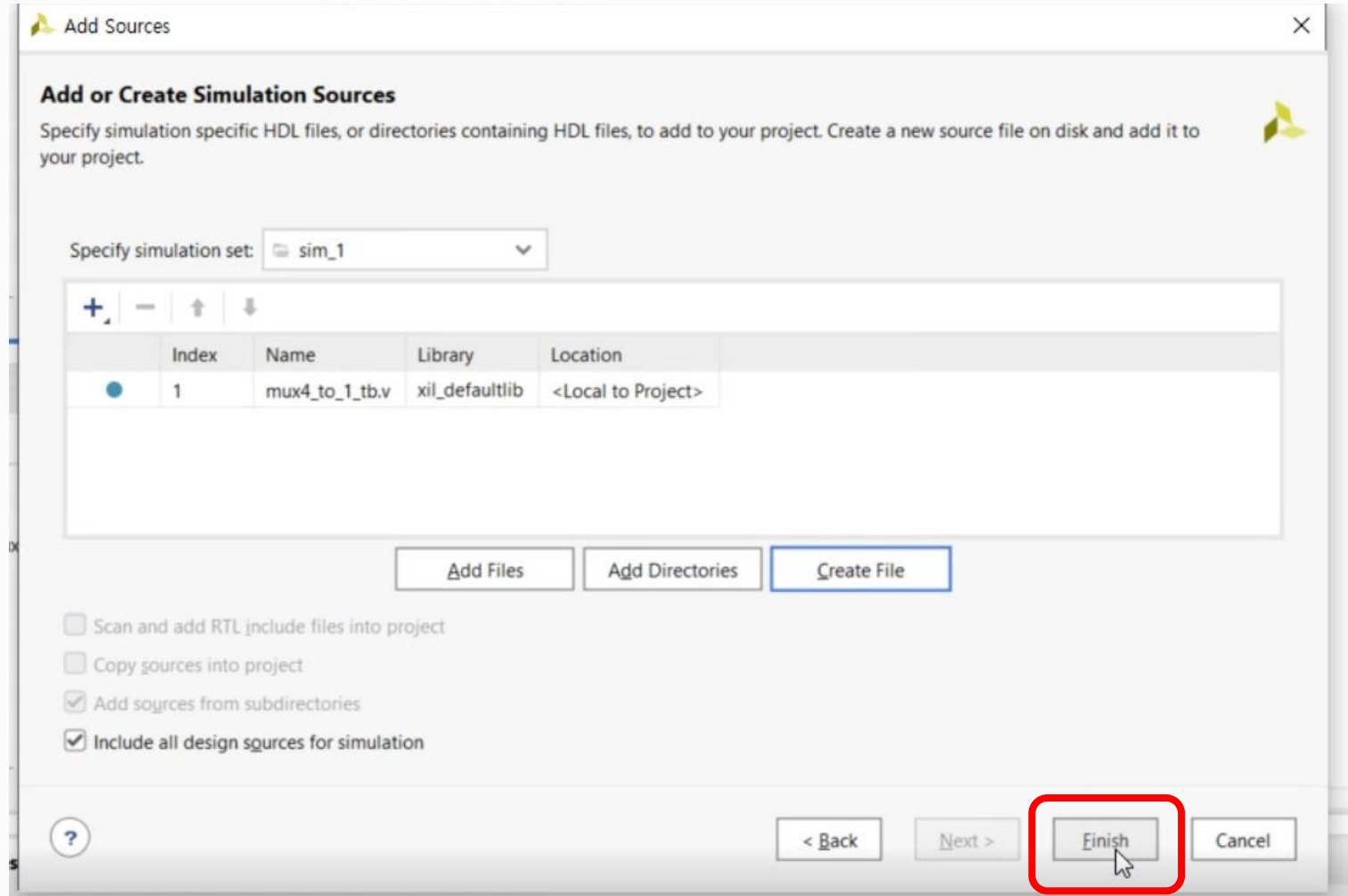
Create File → File name [mux4_to_1_tb] 입력 → ok



실험 내용

◇ Mux4_to_1_tb.v 생성

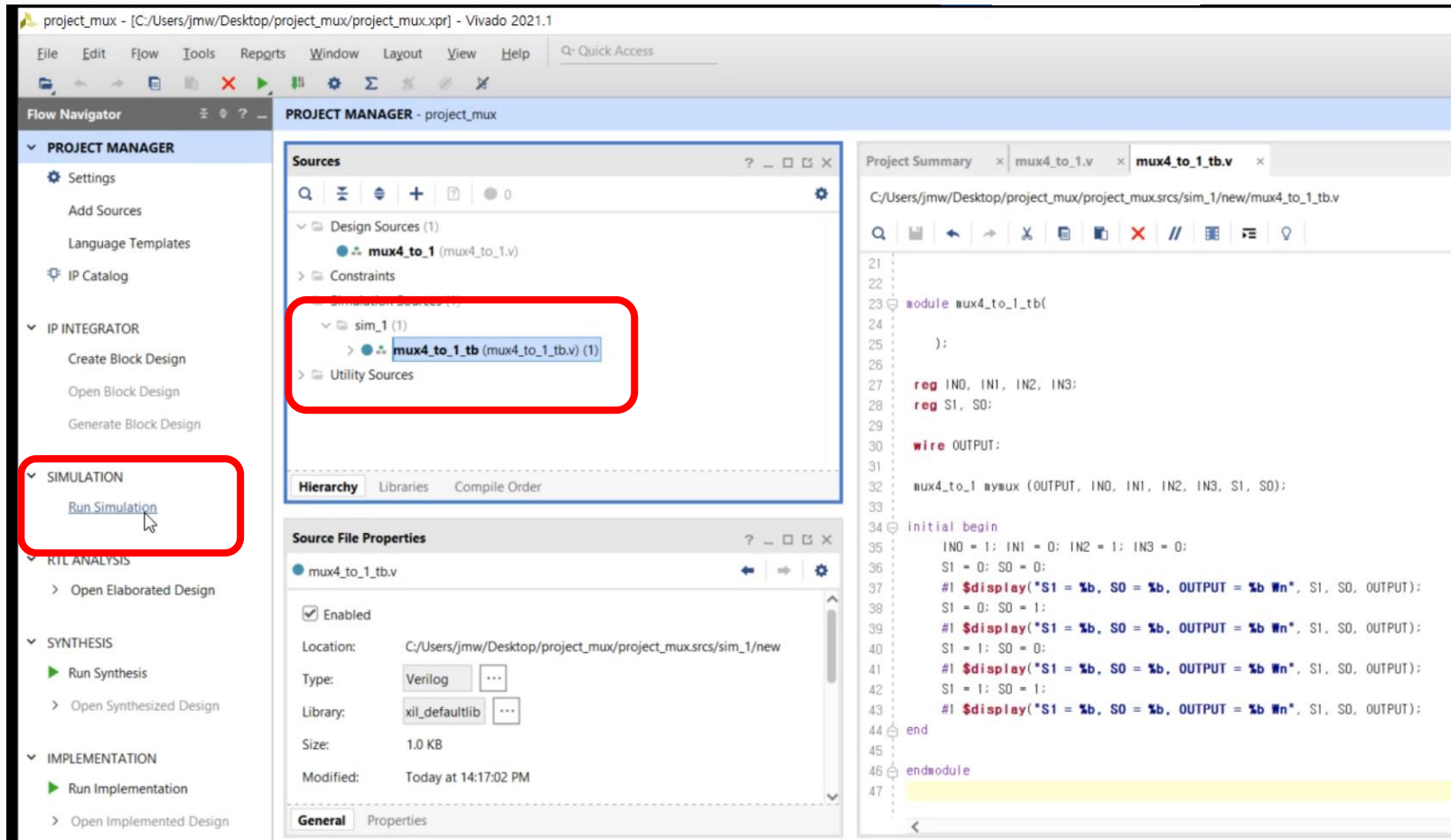
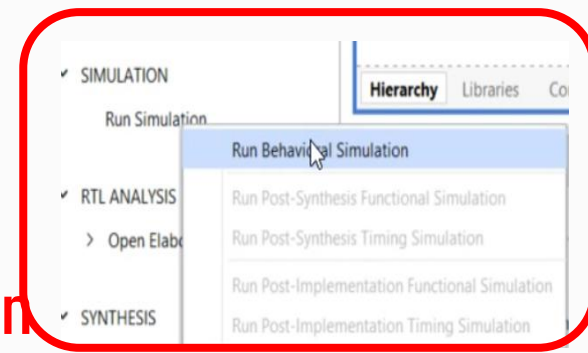
Finish



실험 내용 (cont.)

◇ Mux4_to_1_tb 작성 후

◇ Run Simulation → Run Behavioral Simulation



실험 내용 (cont.)

Run Behavioral Simulation – waveform

The image shows a behavioral simulation interface. The top part displays a waveform window with a table of signal values and a corresponding waveform plot. The bottom part shows a Tcl console with simulation logs and a table of signal values.

Waveform Window:

- Tab: mux4_to_1.v
- Tab: mux4_to_1_tb.v
- Tab: Untitled 1
- Zoom in button (magnifying glass icon) is highlighted with a red box and labeled "Zoom in".
- Table:

Name	Value
IN0	1
IN1	0
IN2	1
IN3	0
S1	0
S0	1
OUTPUT	0

The waveform plot shows a green signal trace over time (0.000 ns to 12.000 ns). A yellow vertical line marks 1.000 ns.

Tcl Console:

- Tab: Messages
- Log
- Search icon
- Run icon
- Stop icon
- Refresh icon
- Copy icon
- Print icon

```
INFO: [USF-XSim-98] *** Running xsim
with args "mux4_to_1_tb_behav -key {Behavioral:sim_1:Functional:mux4_to_1_tb} -tclbatch {mux4_to_1_tb.tcl} -log {simulate.log}
INFO: [USF-XSim-8] Loading simulator feature
Time resolution is 1 ps
source mux4_to_1_tb.tcl
# set curr_wave [current_wave_config]
# if { [string length $curr_wave] == 0 } {
#   if { [llength [get_objects]] > 0 } {
#     add_wave /
#     set_property needs_save false [current_wave_config]
#   } else {
#     send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without a wave window. If you want to o
#   }
# }
# run 1000ns
SI = 0, S0 = 0, OUTPUT = 1
SI = 0, S0 = 1, OUTPUT = 0
SI = 1, S0 = 0, OUTPUT = 1
SI = 1, S0 = 1, OUTPUT = 0
INFO: [USF-XSim-96] XSim completed. Design snapshot 'mux4_to_1_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:05 ; elapsed = 00:00:05 ; Memory (MB): peak = 2350.924 ; gain = 9.527
```

The table of signal values in the Tcl console is highlighted with a red box and labeled "display".

SI	S0	OUTPUT
0	0	1
0	1	0
1	0	1
1	1	0

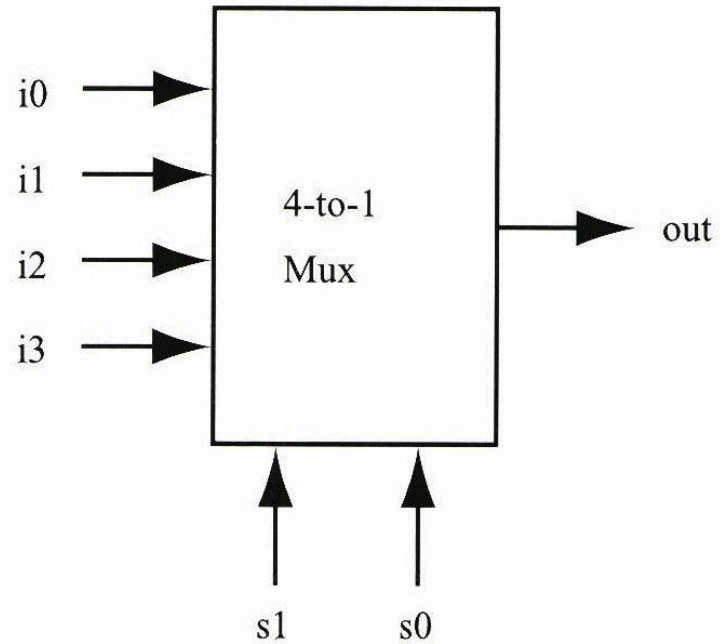
실험 내용 (cont.)

◇ .v 파일에 에러가 있으면 TCl console, Messages 살펴보기

The image displays a Verilog IDE interface with two main windows. The left window shows the source code for a Verilog module named `mux4_to_1`. The code includes a revision header, input/output declarations, and a `wire` declaration for `y0, y1, y2, y3`. A red box highlights a syntax error in the `wire` declaration on line 31: `wire y0, y1, y2, y3; s0;`. The right window shows the TCl console output, which includes simulation logs and a warning message: "No top level signals found. Simulator will start without a wave window. If you want to o". Below the TCl console, the Messages window is open, showing a list of errors and warnings. A red box highlights the following messages:

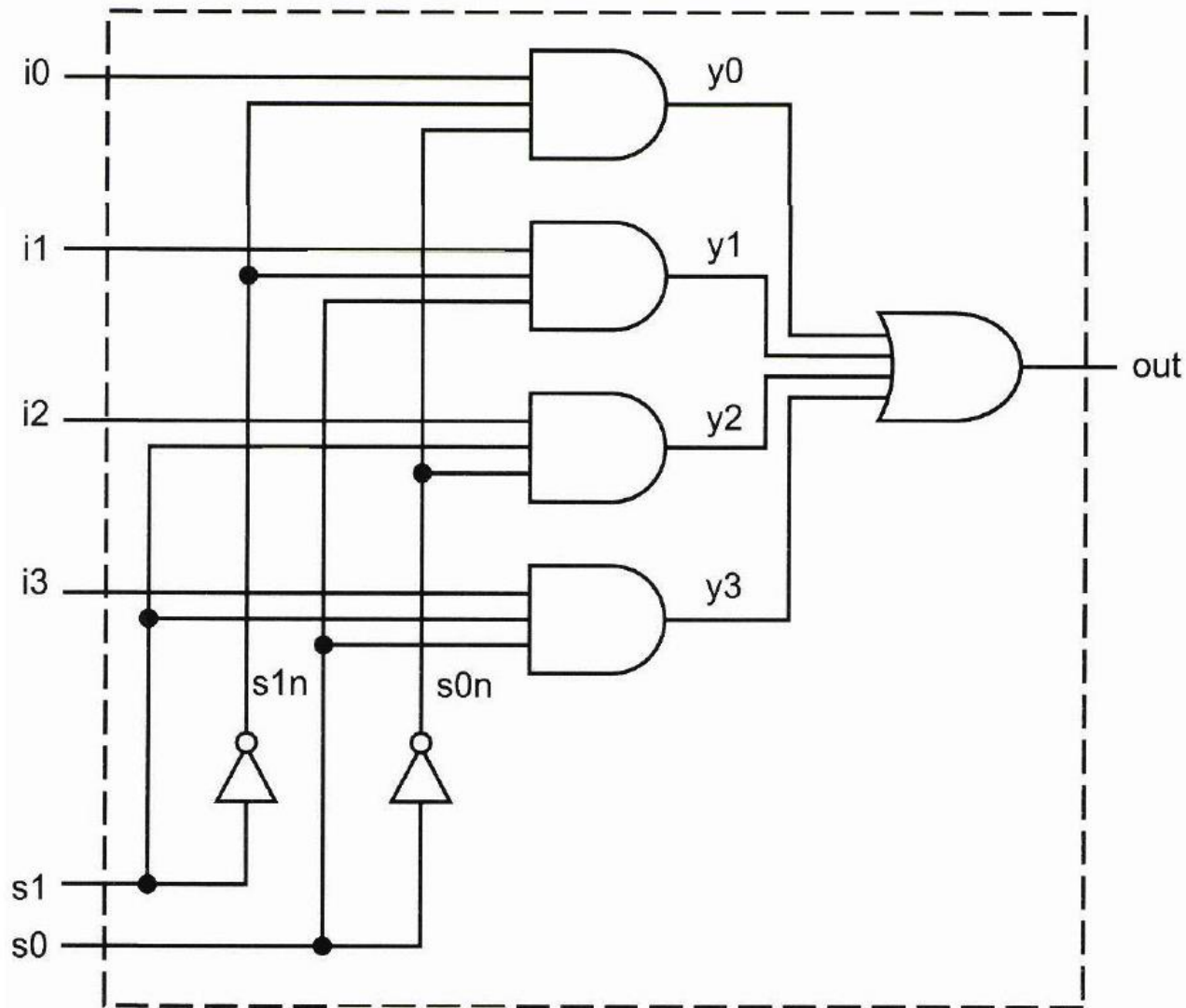
- Analysis Results (2 critical warnings)
- sources_1 (1 critical warning)
- [HDL 9-806] Syntax error near "not". [mux4_to_1.v:33]
- sim_1 (1 critical warning)
- [HDL 9-806] Syntax error near "not". [mux4_to_1.v:33]
- Elaborated Design (2 errors)
- General Messages (2 errors)

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

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

```
    output out;
```

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

```
    input s1, s0;
```

```
    wire s1n, s0n;
```

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

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

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

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

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

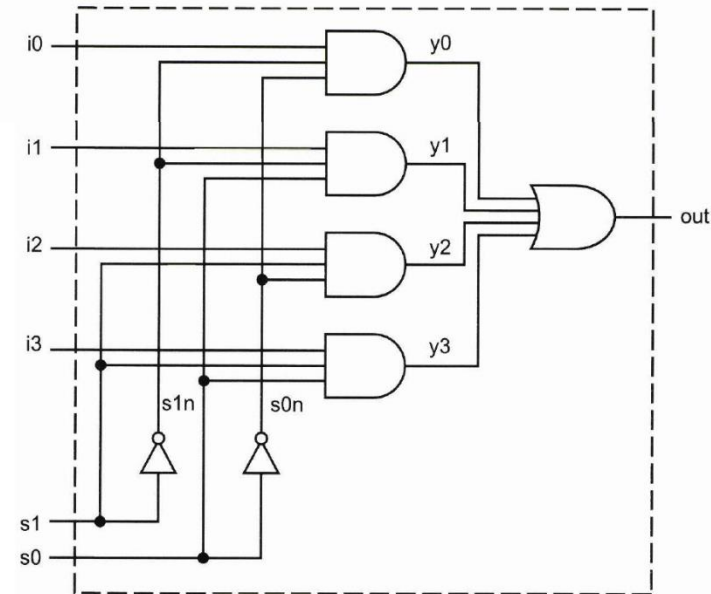
```
    and (y2, i2, s1, s0n);
```

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

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

```
endmodule
```

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;
```

Input인지 output인지
구별해 주기

```
// 내부 wire 선언
```

```
wire s1n, s0n;
```

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

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

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

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

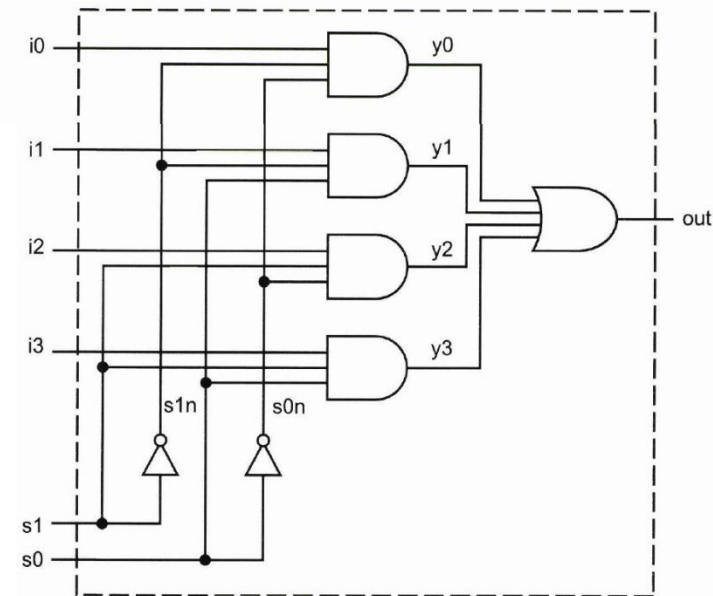
```
and (y1, i1, s1n, s0);
```

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

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

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

```
endmodule
```



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;
```

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

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

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

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

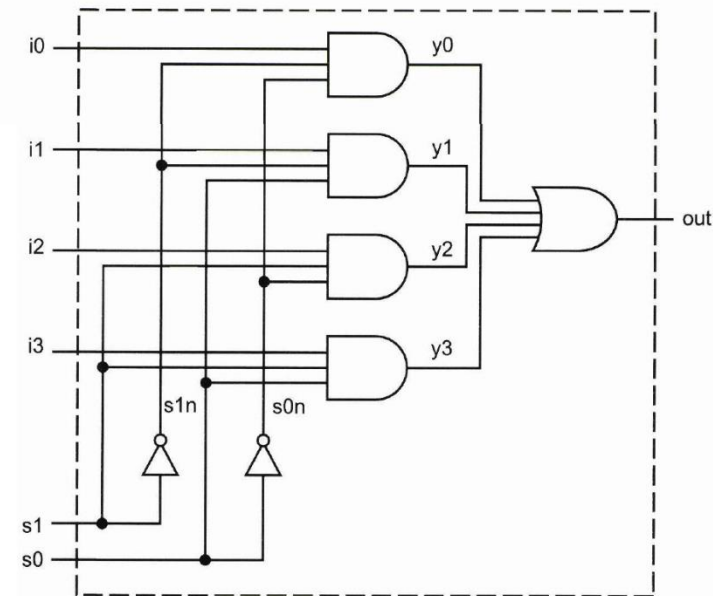
```
and (y2, i2, s1, s0n);
```

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

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

```
endmodule
```

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);
```

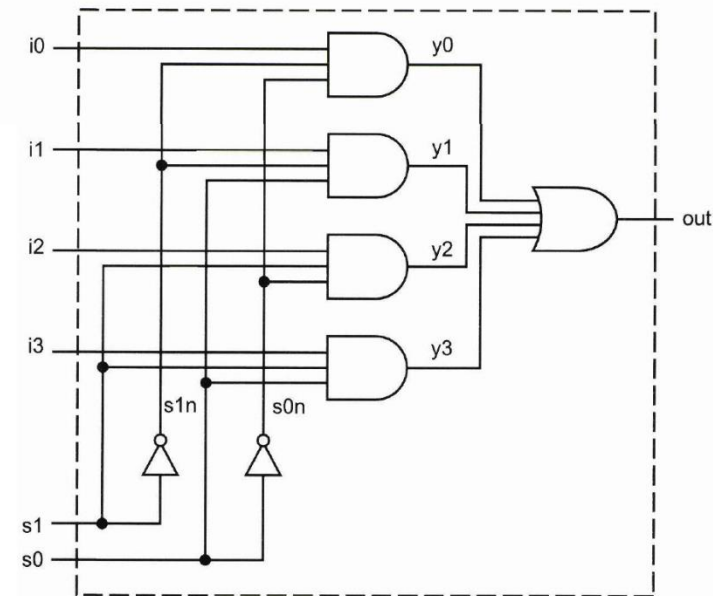
```
and (y3, i3, s1, s0);
```

```
//4개의 입력을 갖는 OR 게이트 파생
```

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

```
endmodule
```

not (out, in) //앞쪽이 출력, 뒤쪽이 입력



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);
```

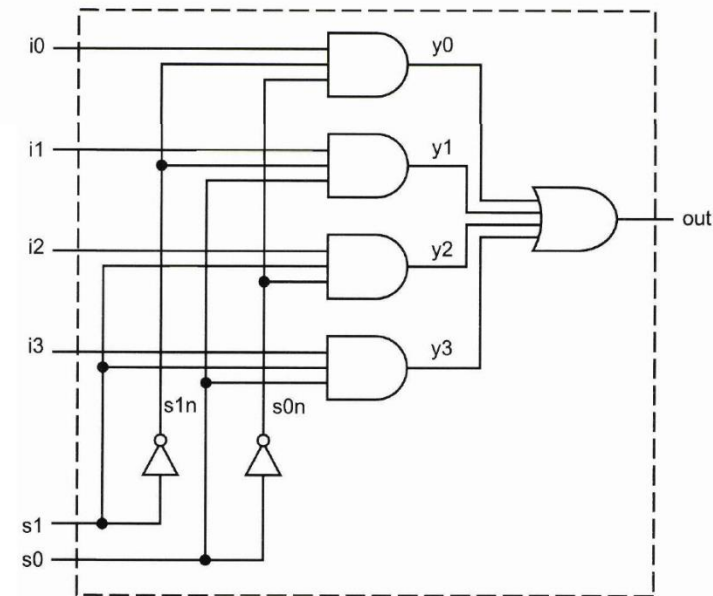
```
and (y3, i3, s1, s0);
```

```
//4개의 입력을 갖는 OR 게이트 파생
```

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

```
endmodule
```

and (out, in1, in2, in3)



Gate 수준의 4:1 Multiplexer Verilog

: MUX4_1.v

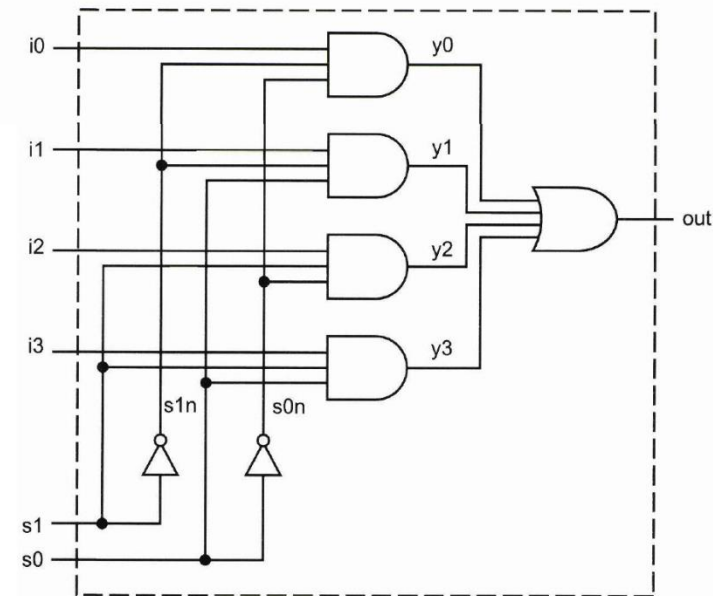
```
//게이트 파생 s1n , s0n 신호를 생성
not (s1n, s1);
not (s0n, s0);

//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

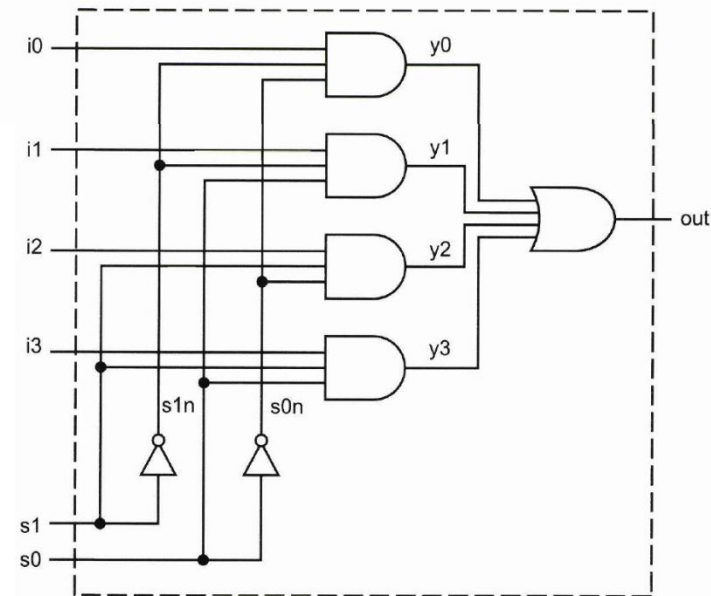
: MUX4_1.v

```
//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

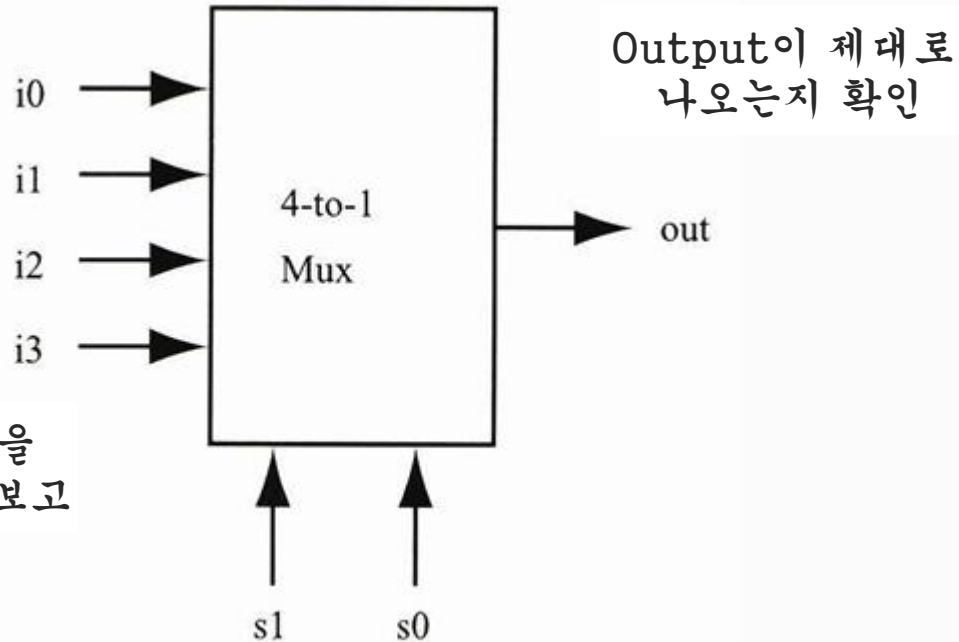


4:1 Multiplexer Verilog 테스트하기

빵판 역할을 하는
모듈에 들고 와서
Test 하기

테스트를 위한 모듈

우리가 방금
설제한 Mux모듈

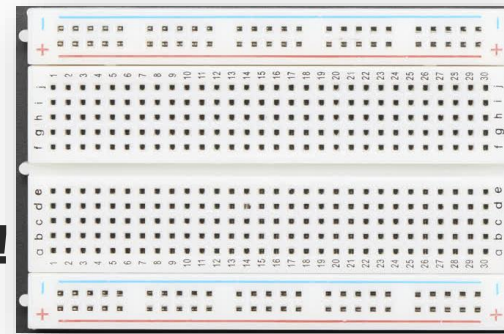


원하는 input을
이것 저것 넣어보고

Test Code

: MUX4_1_tb.v

뺑판 역할!



```
module mux4_to_1_tb(); //포트가 없으면 테스트용 코드라는 것을 알 수 있다.
```

```
//입력으로 연결되는 변수들의 정의
```

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

```
reg S1, S0;
```

```
wire OUTPUT; //출력 wire선언
```

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

```
initial begin
```

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

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

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

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

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

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

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

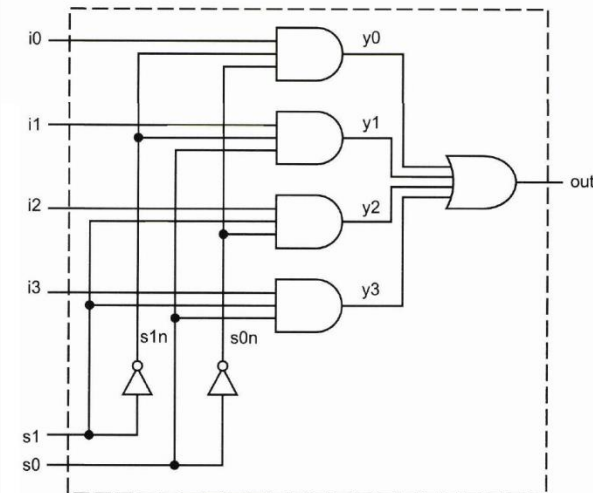
```
    S1 = 1; S0 = 1;
```

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

```
end
```

```
endmodule
```

포트가 없으면 테스트용 코드라는 것을 알 수 있다.



Test Code

: MUX4_1_top.v

```
module mux4_to_1_tb(); //포트가 없으면 테스트용 코드라는 것을 알 수 있다.

    //입력으로 연결되는 변수들의 정의
    reg IN0, IN1, IN2, IN3;
    reg S1, S0;

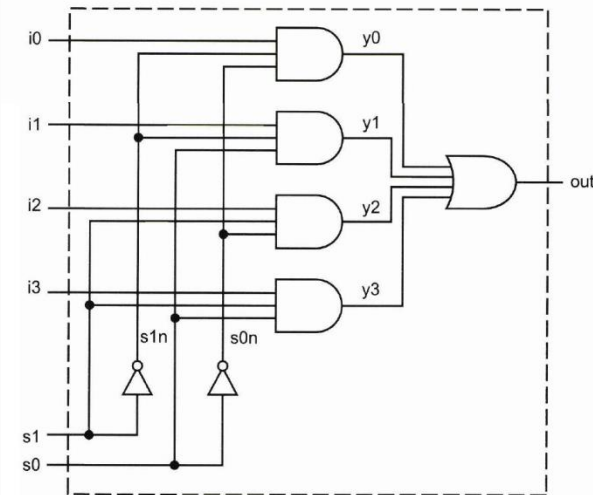
    wire OUTPUT; //출력 wire선언

    mux4_to_1 mymux (OUTPUT, IN0, IN1, IN2, IN3, S1, S0); //멀티플렉서의 파생

    initial begin
        IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
        S1 = 0; S0 = 0;
        #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
        S1 = 0; S0 = 1;
        #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
        S1 = 1; S0 = 0;
        #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
        S1 = 1; S0 = 1;
        #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
    end

endmodule
```

6개의 입력에 이것 저것 넣어볼 수 있도록
레지스터 (저장공간)선언



Test Code

: MUX4_1_top.v

```
module mux4_to_1_tb(); //포트가 없으면 테스트용 코드라는 것을 알 수 있다.

//입력으로 연결되는 변수들의 정의
reg IN0, IN1, IN2, IN3;
reg S1, S0;

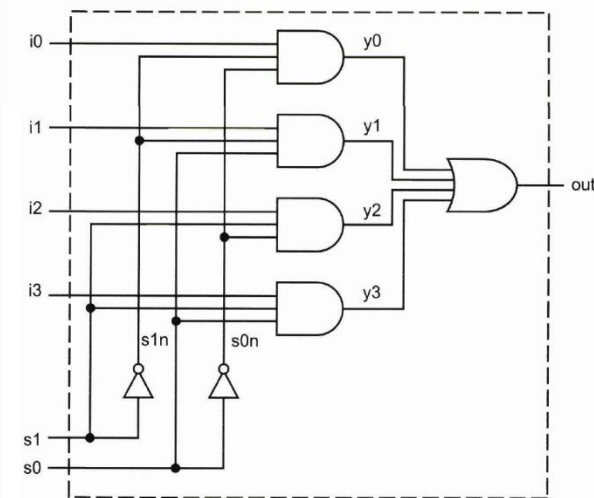
wire OUTPUT; //출력 wire선언

mux4_to_1 mymux (OUTPUT, IN0, IN1, IN2, IN3, S1, S0); //멀티플렉서의 파생

initial begin
    IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
    S1 = 0; S0 = 0;
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
    S1 = 0; S0 = 1;
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
    S1 = 1; S0 = 0;
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
    S1 = 1; S0 = 1;
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
end

endmodule
```

출력은 단순히 확인만 하면 되니까
Wire 연결해 둔다.



Test Code

: MUX4_1_top.v

```
module mux4_to_1_tb(); //포트가 없으면 테스트용 코드라는 것을 알 수 있다.
```

```
//입력으로 연결되는 변수들의 정의
```

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

```
reg S1, S0;
```

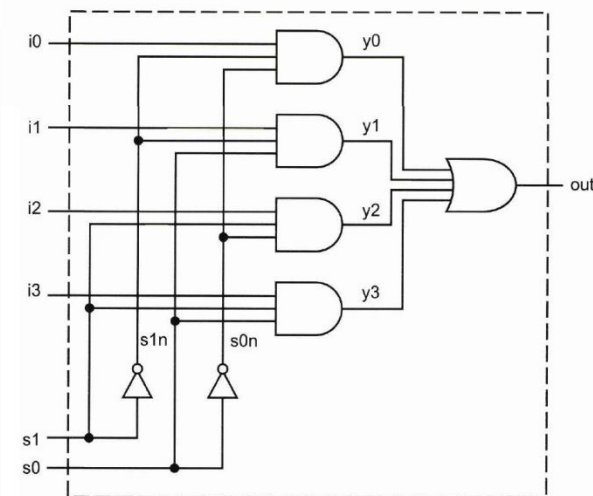
```
wire OUTPUT; //출력 wire선언
```

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

내가 정의한
module이름

instance이름

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



Test Code

: MUX4_1_top.v

```
module mux4_to_1_tb(); //포트가 없으면 테스트용 코드라는 것을 알 수 있다.
```

```
//입력으로 연결되는 변수들의 정의
```

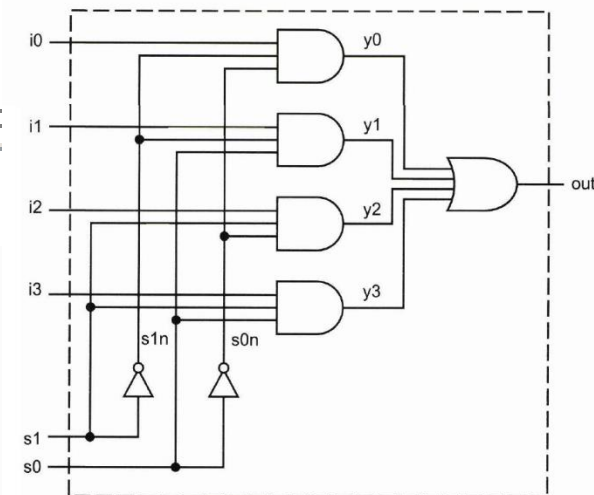
```
reg IN0, IN1, IN2, IN3;
```

```
reg S1, S0;
```

```
wire OUTPUT; //출력 wire선언
```

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

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



Test Code

: MUX4_1_top.v

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

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

```
initial begin
```

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

S1 = 0; S0 = 0; 입력 레지스터에 1, 0, 1, 0을 저장하고 1ns 이후에 확인해보기

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

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

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

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

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

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

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

```
end
```

```
endmodule
```

Test Code

: MUX4_1_top.v

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

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

```
initial begin
```

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

S1 = 0; S0 = 0; 입력 레지스터에 s에 0,0을 저장한 후 1ns 이후에 output 값을 확인해 보기

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

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

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

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

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

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

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

```
end
```

```
endmodule
```

Test Code

: MUX4_1_top.v

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

initial begin Main 함수 처럼 simulation 시작하자마자 initial에서 출발한다.
    IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
    S1 = 0; S0 = 0;
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
    S1 = 0; S0 = 1;
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
    S1 = 1; S0 = 0;
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
    S1 = 1; S0 = 1;
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
end

endmodule
```

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

Test Code

: MUX4_1_top.v

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

initial begin
    IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
    S1 = 0; S0 = 0;
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
    S1 = 0; S0 = 1;
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
    S1 = 1; S0 = 0; 입력 레지스터에 s에 0,1을 저장한 후 1ns 이후에 output 값을 확인해 보기
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
    S1 = 1; S0 = 1;
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
end

endmodule
```

Test Code

: MUX4_1_top.v

Initial begin

// 테스트하고 싶은 값을

//이것 저것 넣어보기

//테스트 다 했으면

end

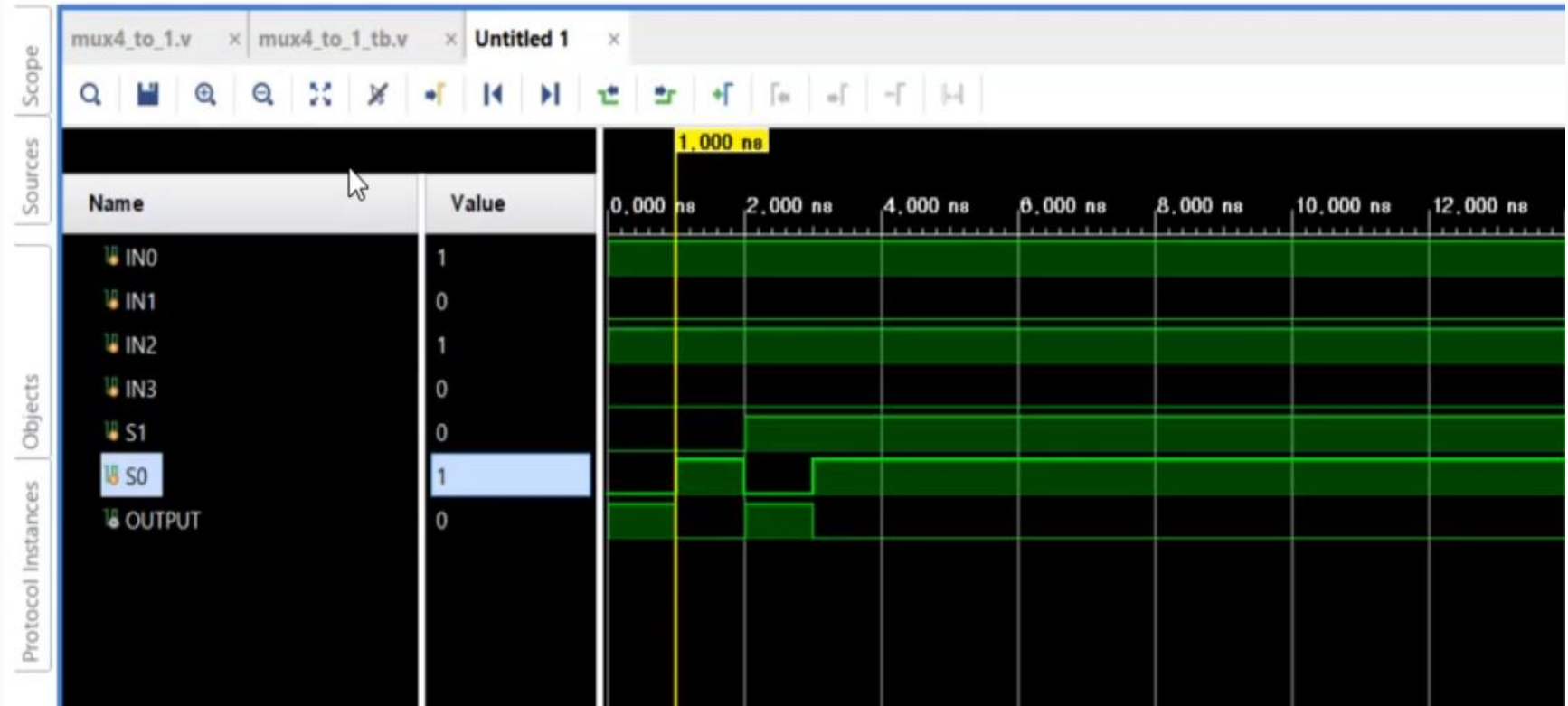
```
mux4_to_1 mymux (OUTPUT, IN0, IN1, IN2, IN3, S1, S0); //멀티플렉서의 파생

initial begin
    IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
    S1 = 0; S0 = 0;
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
    S1 = 0; S0 = 1;
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
    S1 = 1; S0 = 0;
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
    S1 = 1; S0 = 1;
    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b Wn", S1, S0, OUTPUT);
end      Initial begin 의 내용이 끝났음을 표시

endmodule  테스트 모듈의
           끝을 표시
```

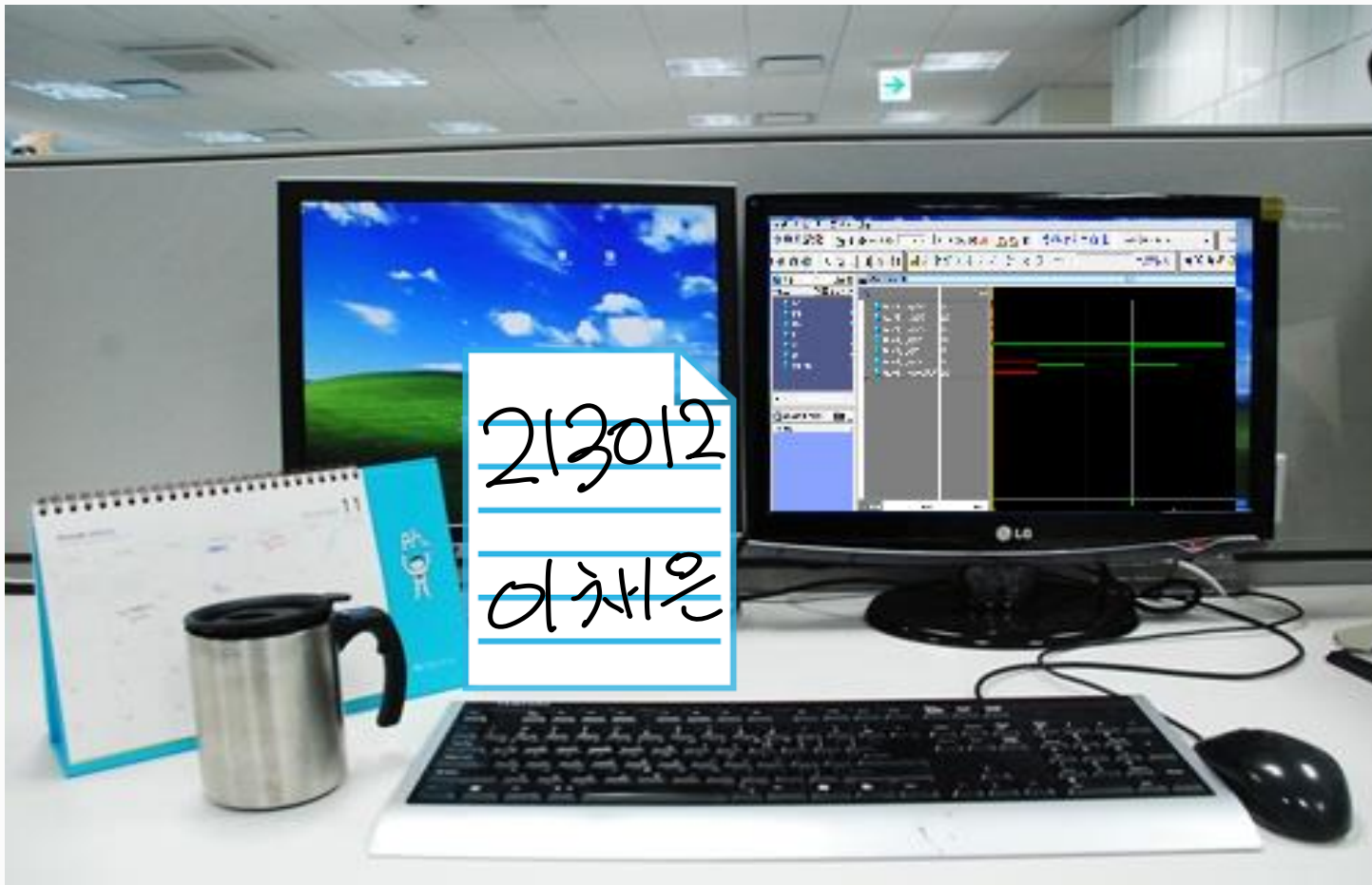
WAVEFORM

: MUX4_1_top.v



과제제출 – 10/30 토요일까지

1. Vivado 를 설치하고 함께 첨부한 verilog file을 실행시켜봅니다.
2. 실행 후 waveform 을 확인한 후
3. 모니터 앞에 학번과 이름을 적은 쪽지를 두고 waveform과 쪽지를 잘 보이게 인증샷을 찍은 후 사진파일을 I-class 에 과제 제출합니다.



213012
이채은