Lab. 04

Logic Design Lab.
Fall 2019

Prof. Sungjoo Yoo
(yeonbin@snu.ac.kr)
TA. Hyunsu Kim
(gustnxodjs@gmail.com)
TA. Hyunyoung Jung
(gusdud1500@gmail.com)

Contents

Verilog

Overview

- Hardware description languages (HDLs)

- Types of programming
  - Gate-level design
  - Behavioral design / Register-transfer level design

- Verilog Basics
  - Verilog Notations
  - Verilog Operators
  - Verilog Keywords & Constructs

Overview

- Hardware description languages (HDLs)

- Types of programming
  - Gate-level design
  - Behavioral design / Register-transfer level design

- Verilog Basics
  - Verilog Notations
  - Verilog Operators
  - Verilog Keywords & Constructs

Hardware description languages (HDLs)

- Different kinds of HDLs
  - Abel (circa 1983) - developed by Data-I/O
  - ISP (circa 1977) - research project at CMU
  - **Verilog** (circa 1985) - developed by Gateway (absorbed by Cadence)
  - **VHDL** (circa 1987) - DoD sponsored standard

- Advantages of HDLs
  - IEEE standard
  - Common
  - Flexible – Delay modeling, Matrices …
  - Describe hardware at varying levels of abstraction

    (= Describe hardware to other people briefly)

Overview

- Hardware description languages (HDLs)

- Types of programming
  - Gate-level design
  - Behavioral design / Register-transfer level design

- Verilog Basics
  - Verilog Notations
  - Verilog Operators
  - Verilog Keywords & Constructs

Types of programming

- Describe hardware at varying levels of abstraction
- Gate-level design
  - Textual replacement for schematic
  - Hierarchical composition of logic gates

- Register-transfer level design
  - Specify dataflow between hardware registers

- **(Synthesizable)** Behavioral design
  - Describe what module does, not how
  - Synthesis generates circuit for module

behavioral design해도 되지만,
synthesizable해야 한다. 꼭!
보통 gate level로 짠다?

Overview

- Hardware description languages (HDLs)

- Types of programming
  - Gate-level design
  - Behavioral design / Register-transfer level design

- Verilog Basics
  - Verilog Notations
  - Verilog Operators
  - Verilog Keywords & Constructs

Verilog Notations

- Verilog is:
  - Case sensitive(event-driven)
  - Based on the programming language C

- Comments
  - Single Line

    ```
    //                              [end of line]
    ```
  - Multiple Line

    ```
    /*
                        …………
                        …………
                                                    */
    ```

- List element separator: **,**
- Statement terminator: **;**

Verilog Notations

- Binary Values for Constants and Variables
  - `0`
  - `1`
  - `X,x` - Unknown
  - `Z,z` – High impedance state (open circuit) floating
- Constants
  - n'b[integer]: `1'b1 = 1, 8'b1 = 000000001, 4'b0101 = 0101,`
    `8'bxxxx = 0000xxxx`
  - n'h[integer]: `8'hA9 = 10101001, 16'hf1= 0000000011110001`

    텍스트
- Identifier Examples
  - Scalar: `A,C,RUN,stop,m,n`
  - Vector: `sel[0:2], f[0:5], ACC[31:0], SUM[15:0],`
    `sum[15:0]`

Overview

- Hardware description languages (HDLs)

- Types of programming
  - Gate-level design
  - Behavioral design / Register-transfer level design

- Verilog Basics
  - Verilog Notations
  - Verilog Operators
  - Verilog Keywords & Constructs

Verilog Operators

- Bitwise Operators

  `~` NOT
  `&` AND
  `|` OR
  `^` XOR
  `^~` or `~^` XNOR

  – **Example**:

```
input[3:0] A, B;
output[3:0] Z ;
assign Z = A | ~B;  // Z = A + B'
```

Verilog Operators

- Logical & Relational Operators

  !, &&, | |, = =, !=, >=, <=, >, <, etc.

- Arithmetic Operators

  `+, -,` etc.

- Concatenation & Replication Operators

  {identifier_1, identifier_2, …}

  {n{identifier}}

  – Examples: `{REG_IN[6:0],Serial_in}, {8 {1’b0}}`

  반복하려는 경우 중괄호 사용한다.

Overview

- Hardware description languages (HDLs)
- Types of programming
  - Gate-level design
  - Behavioral design / Register-transfer level design
- Verilog Basics
  - Verilog Notations
  - Verilog Operators
  - Verilog Keywords & Constructs

Verilog Keywords & Constructs

- `module` – fundamental building block for Verilog designs
  - Used to construct design hierarchy
  - Cannot be nested **module - endmodule 꼭 하나다. 여러 개 안됨.**
- `endmodule` – ends a module – not a statement
  => no “;”
- Module Declaration
  - `module` *module_name* (*module_port*, *module_port*, …);
  - Example:

```
module full_adder (A, B, c_in,
            c_out, S);
…
endmodule
```

Verilog Keywords & Constructs

- Input Declaration
  - Scalar
    - input *list of input identifiers*;
    - Example: `input A, B, c_in;`
  - Vector
    - `input[`*range*`]` *list of input identifiers*;
    - Example: `input[15:0] A, B, data;`

- Output Declaration
  - Scalar Example: `output c_out, OV, MINUS;`
  - Vector Example: `output[7:0] ACC, REG_IN, data_out;`

- Wire Declaration
  - Scalar Example: `wire t1, t2;`
  - Vector Example: `wire[7:0] t1, t2;`

Verilog Keywords & Constructs

- `wire` and `reg`

- Values must be retained over time
  - Register type: `reg`
  - The `reg` in contrast to `wire` stores values between executions of the process

- We can set values on `reg`, but cannot on `wire`

value set 불가능 중요

Verilog Keywords & Constructs

- Primitive Gates
  - **buf**, **not**, **and**, **or**, **nand**, **nor**, **xor**, **xnor**
  - **Syntax:**

    gate_operator instance_identifier (*output, input_1, input_2,* ...)

  - **Example:**

output, input1, input2

```
or O1 (t1, A, B)
   O2 (t2, B, C, D);
// t1=A+B, t2=B+C+D
and A1 (OUT, t1, t2);
// OUT = t1•t2  (OUT = (A+B)•(B+C+D))
```

Verilog Keywords & Constructs

- Process
  - The body of a process consists of procedural statement to make desired outputs from inputs as like a common programming
  - Processes are running in parallel

  - `initial` – executes <u>only once</u> beginning at t = 0
  - **Only works on simulation – cannot be synthesized!**
    - **Syntax**:

```
initial Statement;

initial begin
  Statement;
  Statement;
   …
end
```

합성시 다 사라진다. 결국 값 사라짐.***
0초일때 한번만 실행된다.

  - `always` – executes at t = 0 and <u>repeatedly</u> thereafter following repeat conditions.
    - **Syntax**:

조건에 따라 0초일때도 실행 안될수도 있다.

```
always Repeat condition
   Statement;

always Repeat condition begin
  Statement;
  Statement;
   …
end
```

clock?

Verilog Keywords & Constructs

- Timing Control Statement (for repeat conditions)

Type	Syntax	Description
Delay Control	#10	Delay 10 unit time
Event Control	@(a)	Wait until signal 'a' is changed
true가 될때마다이다.	@(posedge a) @(negedge a)	Wait until signal 'a' is changed to '1' Wait until signal 'a' is changed to '0'
Level Control	wait (a==0)	Wait until signal 'a' is equal to '0'

– The body of the process consists of procedural assignments
  - Blocking assignments
    - Example: C = A + B;
    - Execute sequentially as in a programming language
  - Non-blocking assignments
    - Example: C <= A + B;
    - Evaluate right-hand sides, but do not make any assignment until all right-hand sides evaluated. Execute concurrently unless delays are specified.

Verilog Keywords & Constructs

– **Examples:**

```
Always @(*)        *은 모든 상황을 말한다. 항상 실행된다.
   begin
      B   = A;
      C   = B;      결국은 C에 A 대입.
    end
```

- Suppose initially A = 0, B = 1, and C = 2
  After execution, B = 0 and C = 0

```
Always @(*)
   begin
        B <= A;
        C <= B;
   end                                    그냥 처음부터 non blocking으로 사용하잣
```

- Suppose initially A = 0, B = 1, and C = 2
  After execution, B = 0 and C = 1

Verilog Keywords & Constructs

- Conditional constructs
  - The `if-else`

```
If (condition)
  begin procedural statements end
{else if (condition)
  begin procedural statements end}
else
  begin procedural statements end
```

  - The `case`

```
case expression
  {case expression : statements}
endcase;
```

Lab

Today

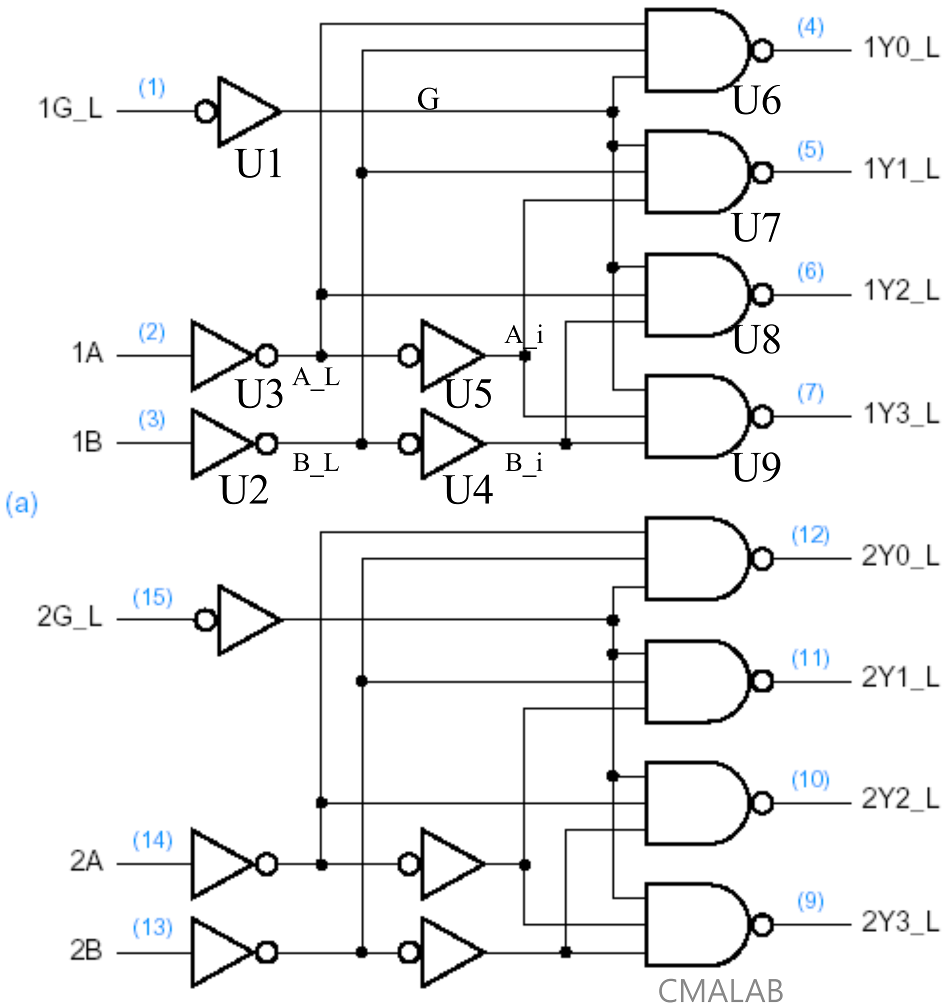
1. **Design a 74x139(dual 2-to-4 decoder) in Verilog**

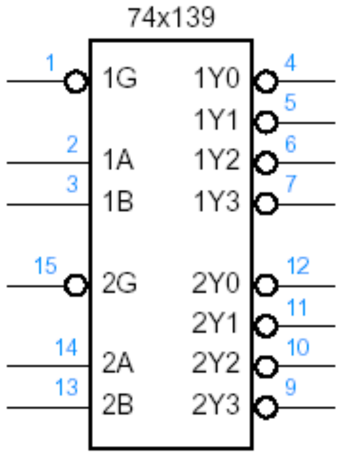
    1) Practice all designing methods for half 74x139 each and simulate them
        1) Gate-level
        2) Behavioral / RTL

    2) Implement a 74x139 and simulate it
        - **Prepare simulations for all possible input combinations**

Designing 74x139(Dual 2-to-4 decoder)

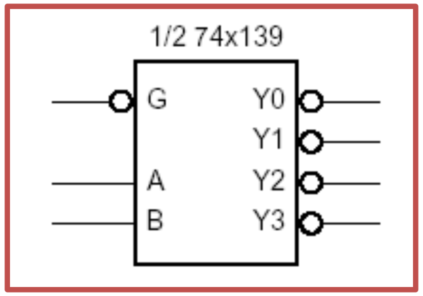
- **Schematic design of a 74x139**





(b)

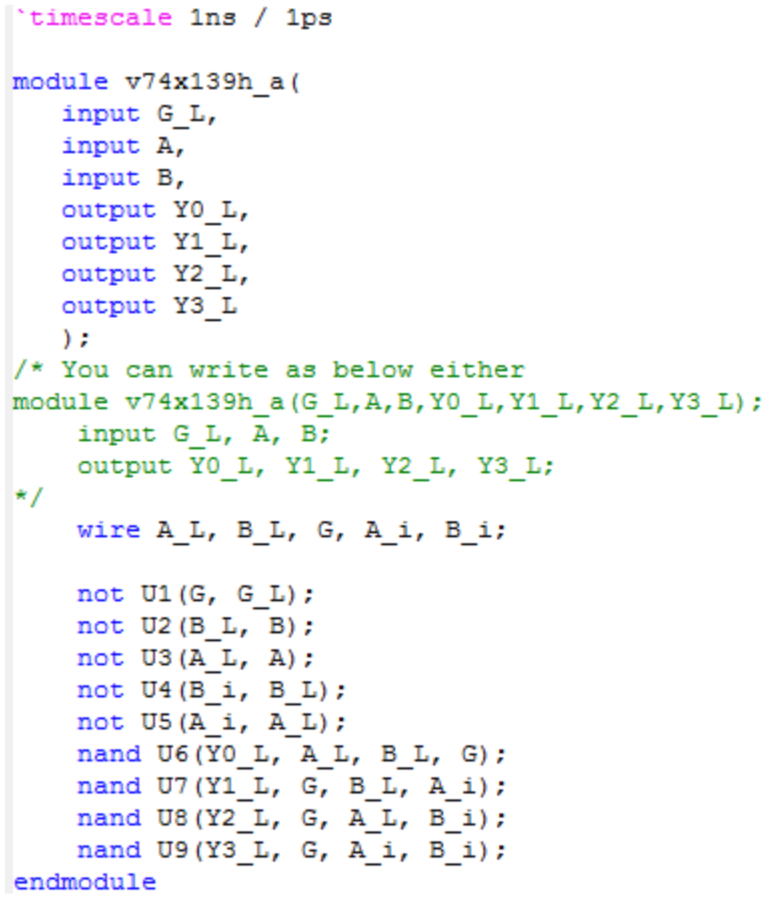
Copyright © 2000 by Prentice Hall, Inc.
Digital Design Principles and Practices, 3/e



(c)

Designing 74x139(Dual 2-to-4 decoder)

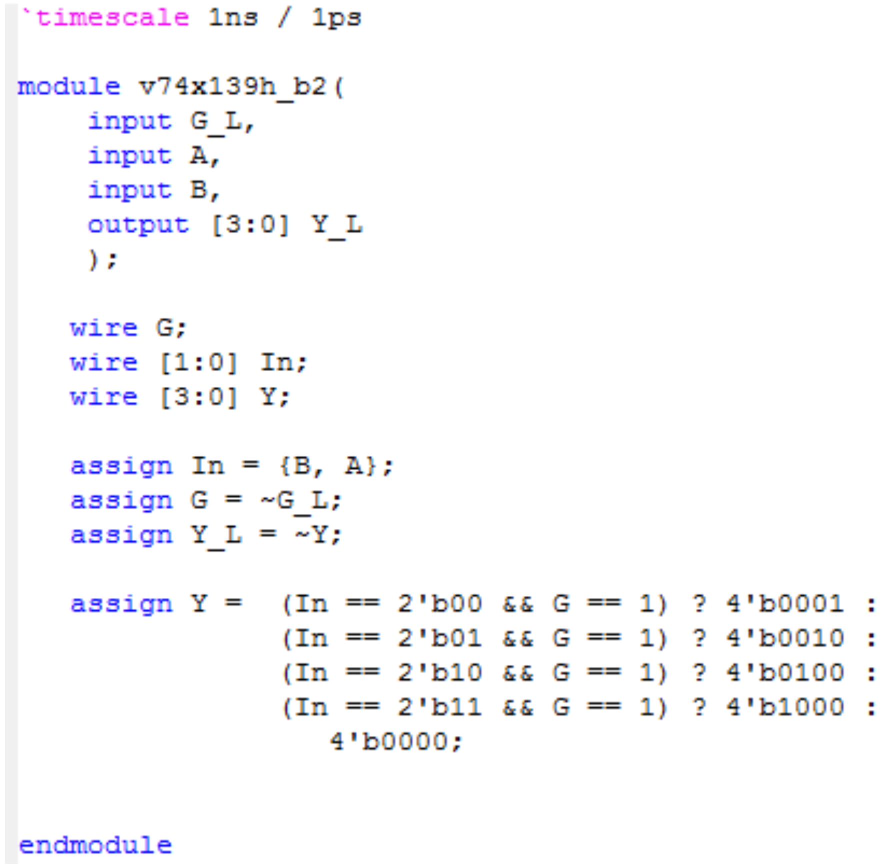
- **Gate level design (Half 74x139)**
  - Circuit function can be described with gate primitives



```
`timescale 1ns / 1ps

module v74x139h_a(
   input G_L,
   input A,
   input B,
   output Y0_L,
   output Y1_L,
   output Y2_L,
   output Y3_L
   );
/* You can write as below either
module v74x139h_a(G_L,A,B,Y0_L,Y1_L,Y2_L,Y3_L);
    input G_L, A, B;
    output Y0_L, Y1_L, Y2_L, Y3_L;
*/
    wire A_L, B_L, G, A_i, B_i;

    not U1(G, G_L);
    not U2(B_L, B);
    not U3(A_L, A);
    not U4(B_i, B_L);
    not U5(A_i, A_L);
    nand U6(Y0_L, A_L, B_L, G);
    nand U7(Y1_L, G, B_L, A_i);
    nand U8(Y2_L, G, A_L, B_i);
    nand U9(Y3_L, G, A_i, B_i);
endmodule
```

Designing 74x139(Dual 2-to-4 decoder)

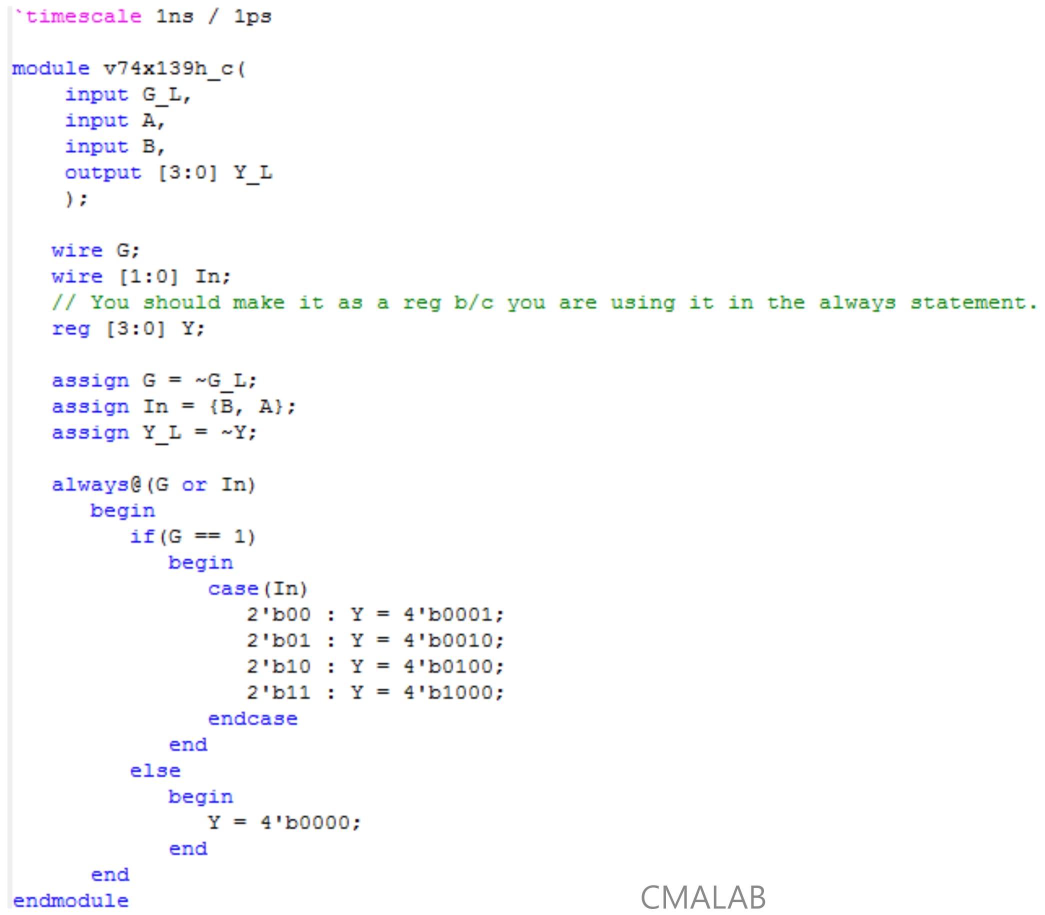
- **RTL design (Half 74x139)**
  - Circuit function can be described by assign(assign/always) statements and the conditional operator with binary combinations as in a truth table



```
`timescale 1ns / 1ps

module v74x139h_b2(
    input G_L,
    input A,
    input B,
    output [3:0] Y_L
    );

   wire G;
   wire [1:0] In;
   wire [3:0] Y;

   assign In = {B, A};
   assign G = ~G_L;
   assign Y_L = ~Y;

   assign Y =  (In == 2'b00 && G == 1) ? 4'b0001 :
               (In == 2'b01 && G == 1) ? 4'b0010 :
               (In == 2'b10 && G == 1) ? 4'b0100 :
               (In == 2'b11 && G == 1) ? 4'b1000 :
                  4'b0000;


endmodule
```

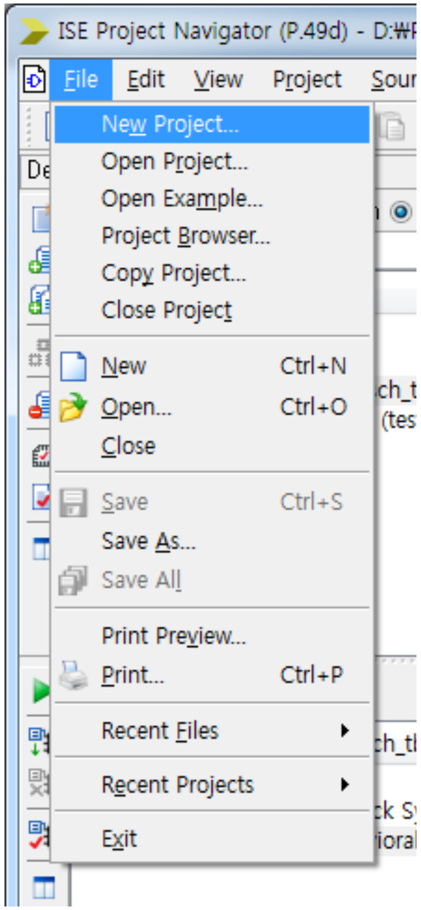
Designing 74x139(Dual 2-to-4 decoder)

- **Behavioral design (Half 74x139)**
  - Circuit function can be described by assign(assign/always) statements and the conditional operator with binary combinations as in a truth table

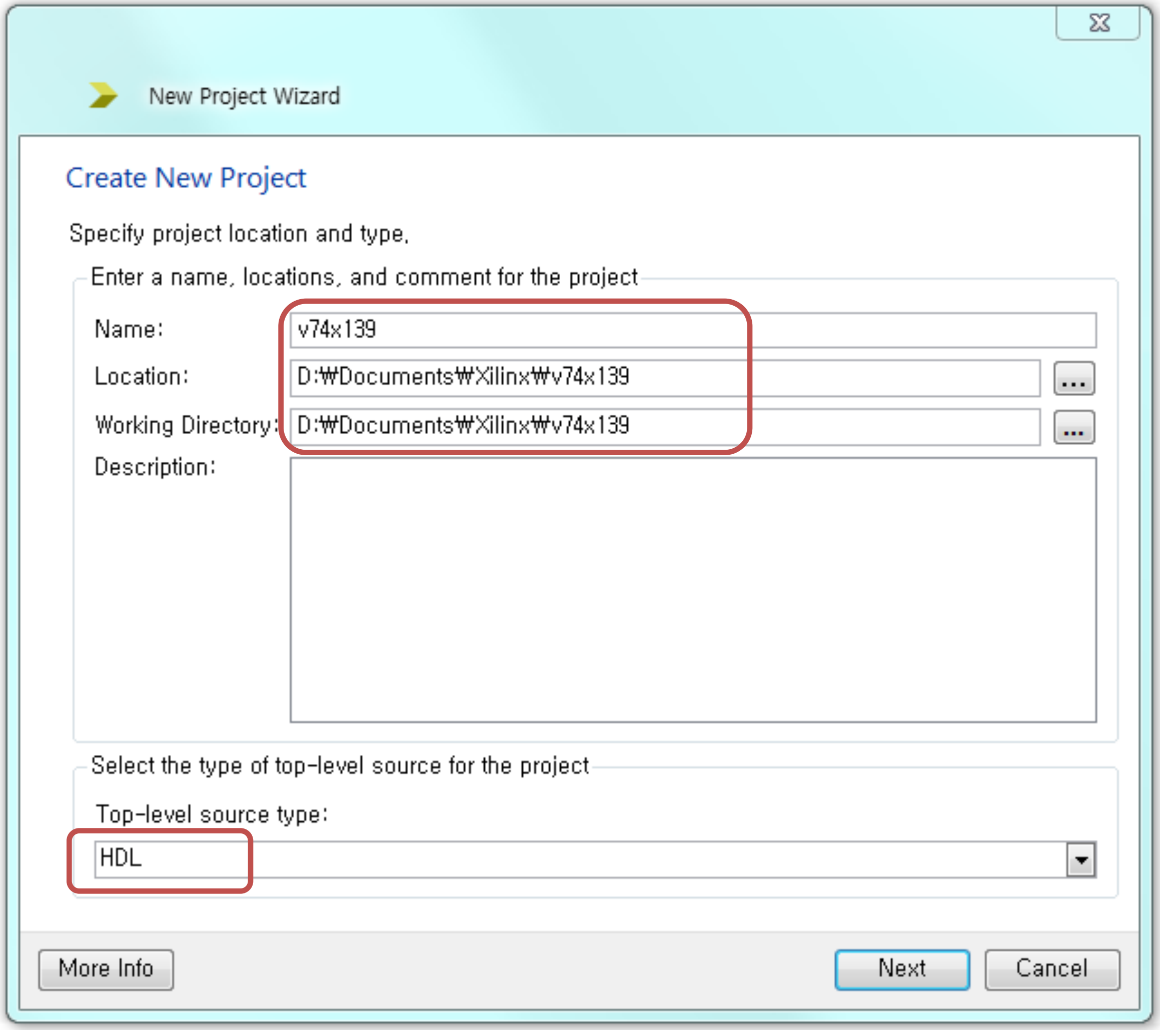


```
`timescale 1ns / 1ps

module v74x139h_c(
    input G_L,
    input A,
    input B,
    output [3:0] Y_L
    );

   wire G;
   wire [1:0] In;
   // You should make it as a reg b/c you are using it in the always statement.
   reg [3:0] Y;

   assign G = ~G_L;
   assign In = {B, A};
   assign Y_L = ~Y;

   always@(G or In)
      begin
         if(G == 1)
            begin
               case(In)
                  2'b00 : Y = 4'b0001;
                  2'b01 : Y = 4'b0010;
                  2'b10 : Y = 4'b0100;
                  2'b11 : Y = 4'b1000;
               endcase
            end
         else
            begin
               Y = 4'b0000;
            end
      end
endmodule
```

Create a new Verilog project



Set the project name, location, type



Create New Project

Specify project location and type.

Enter a name, locations, and comment for the project

Name: v74x139

Location: D:₩Documents₩Xilinx₩v74x139

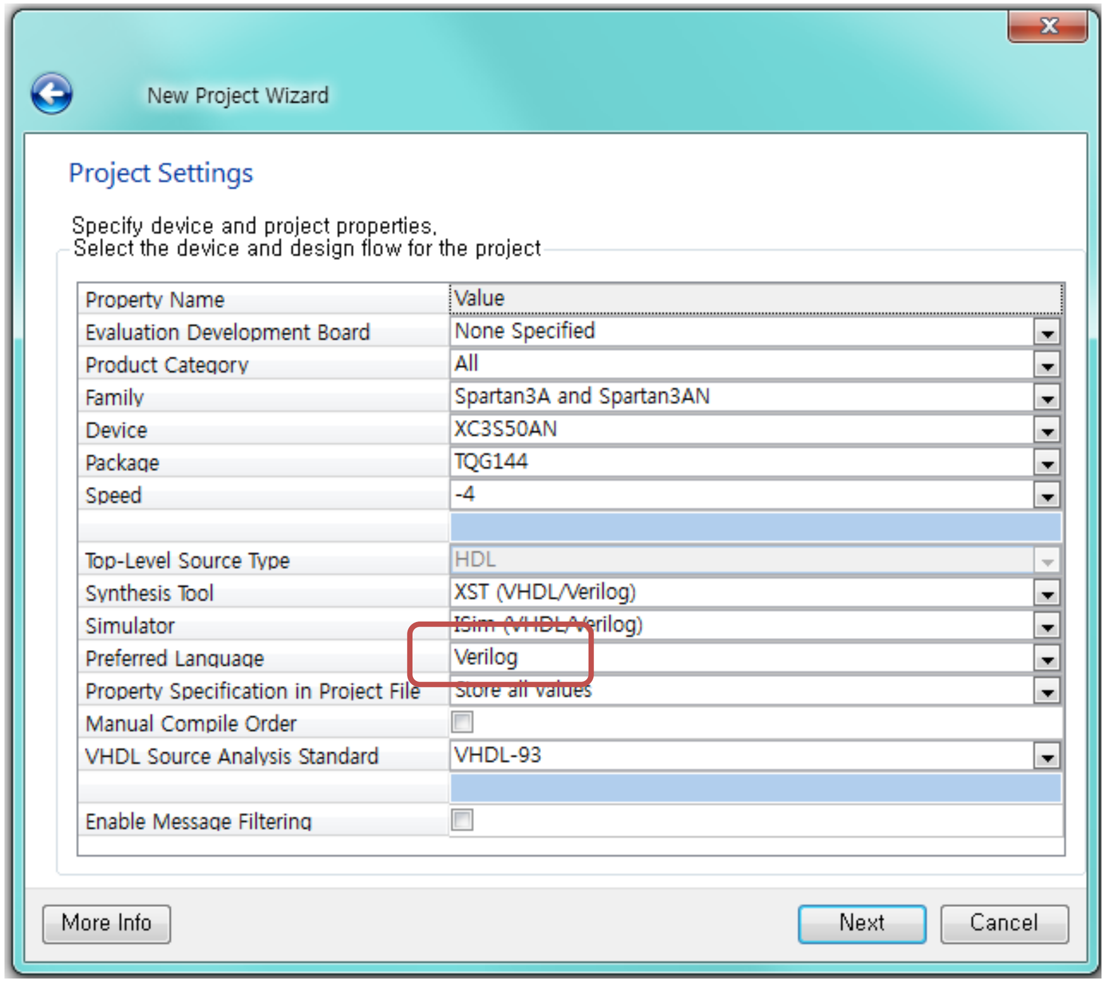
Working Directory: D:₩Documents₩Xilinx₩v74x139

Description:

Select the type of top-level source for the project

Top-level source type: HDL

Set the project name, location, type

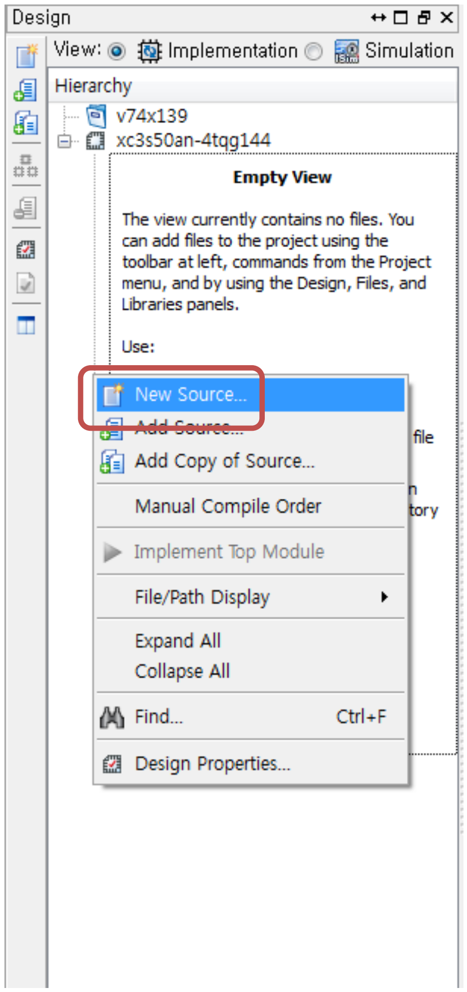


Project Settings

Specify device and project properties.
Select the device and design flow for the project

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan3A and Spartan3AN
Device	XC3S50AN
Package	TQG144
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	Verilog
Property Specification in Project File	Store all values
Manual Compile Order	☐
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	☐

Create a new Verilog source



Design

View: Implementation Simulation

Hierarchy

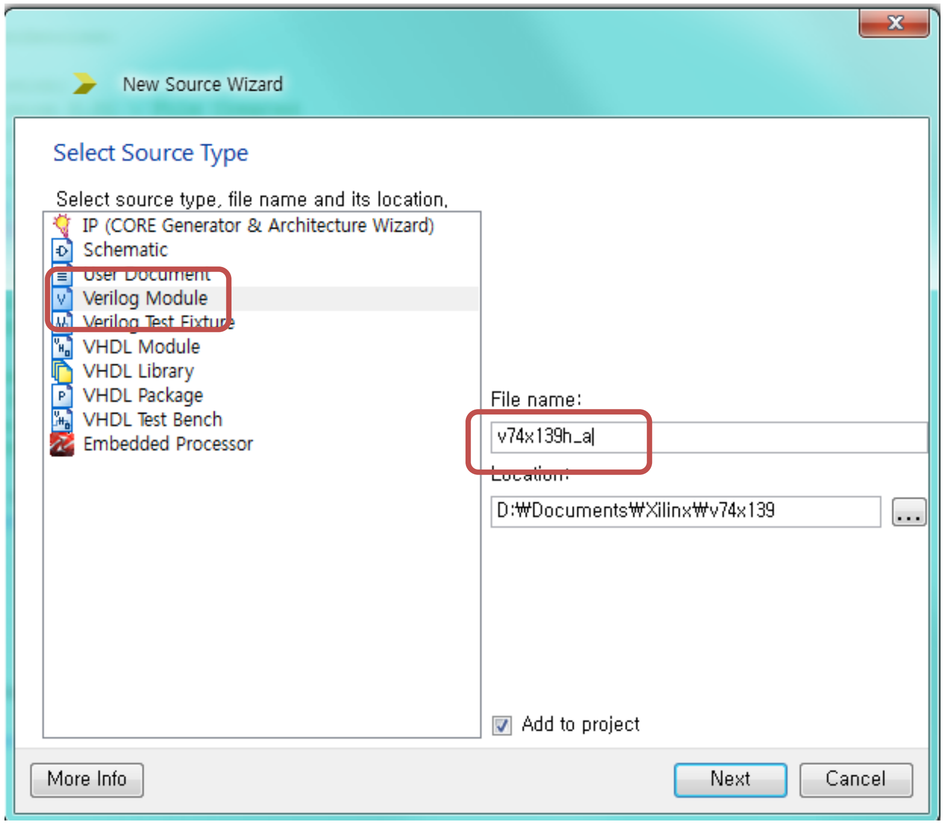
v74x139

xc3s50an-4tqg144

Empty View

The view currently contains no files. You can add files to the project using the toolbar at left, commands from the Project menu, and by using the Design, Files, and Libraries panels.

Use:



New Source Wizard

Select Source Type

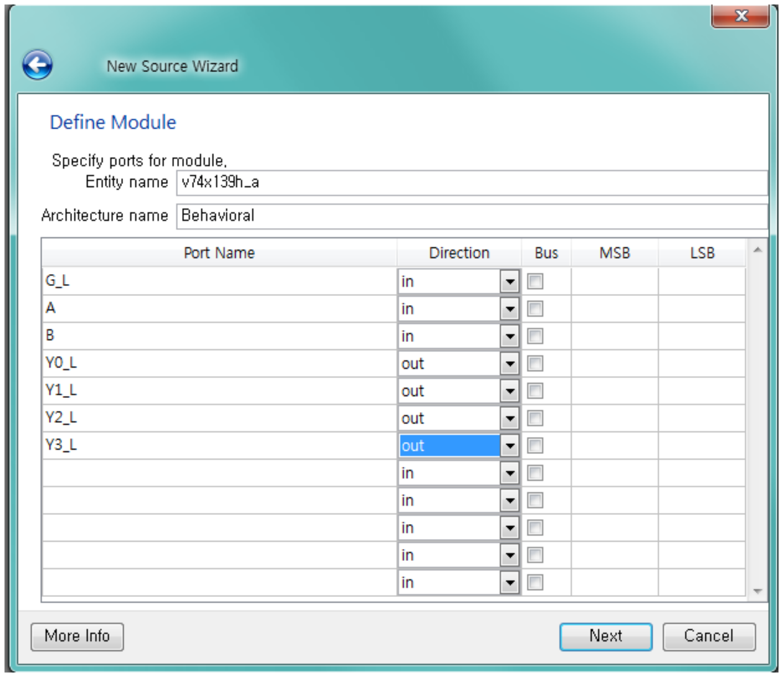
Select source type, file name and its location.

File name: v74x139h_a

Location: D:₩Documents₩Xilinx₩v74x139

☑ Add to project

Create a new Verilog source



Define Module

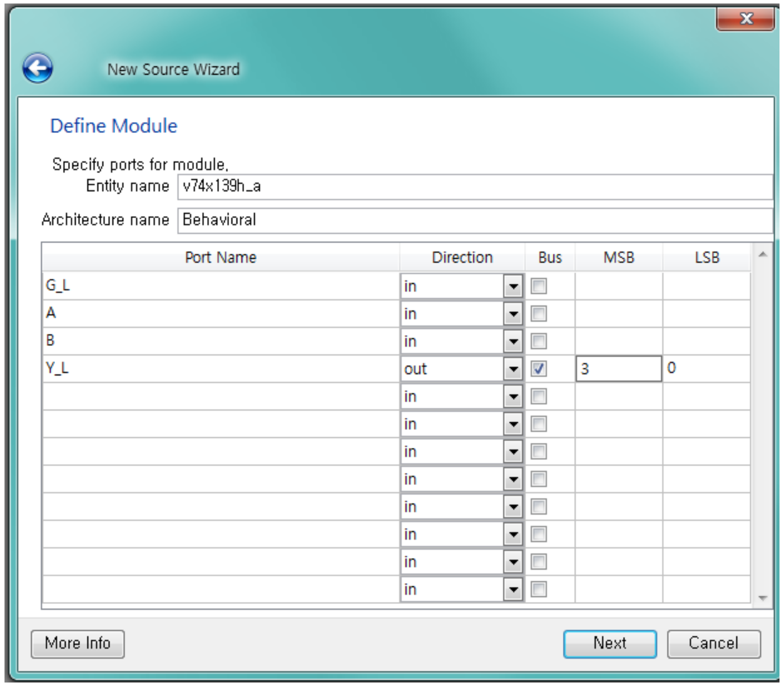
Specify ports for module.

Entity name: v74x139h_a

Architecture name: Behavioral

Port Name	Direction	Bus	MSB	LSB
G_L	in	☐		
A	in	☐		
B	in	☐		
Y0_L	out	☐		
Y1_L	out	☐		
Y2_L	out	☐		
Y3_L	out	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		

or



Define Module

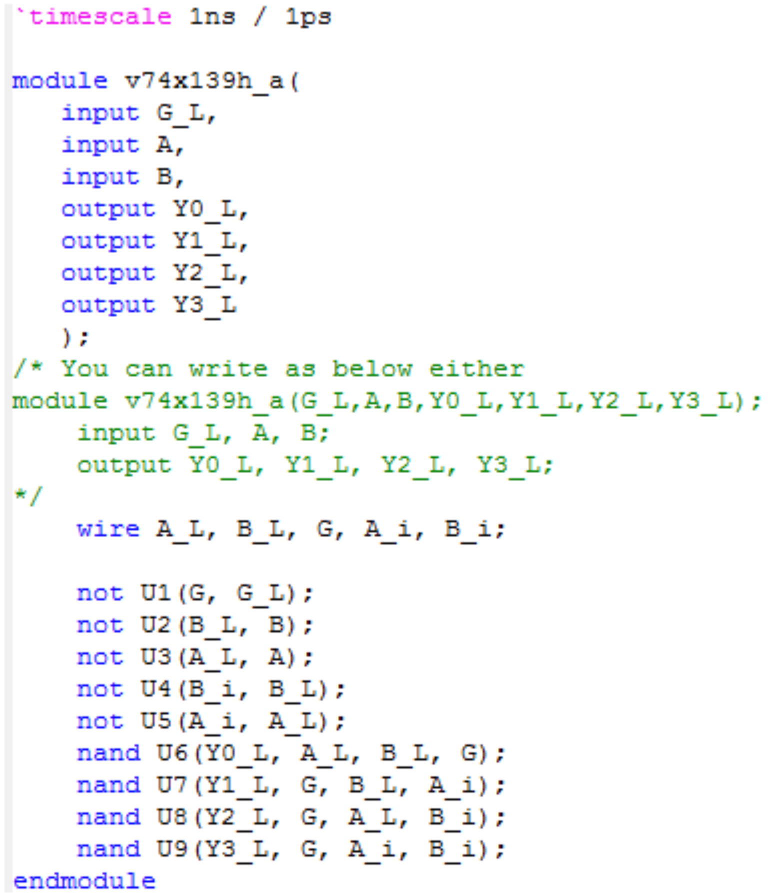
Specify ports for module.

Entity name: v74x139h_a

Architecture name: Behavioral

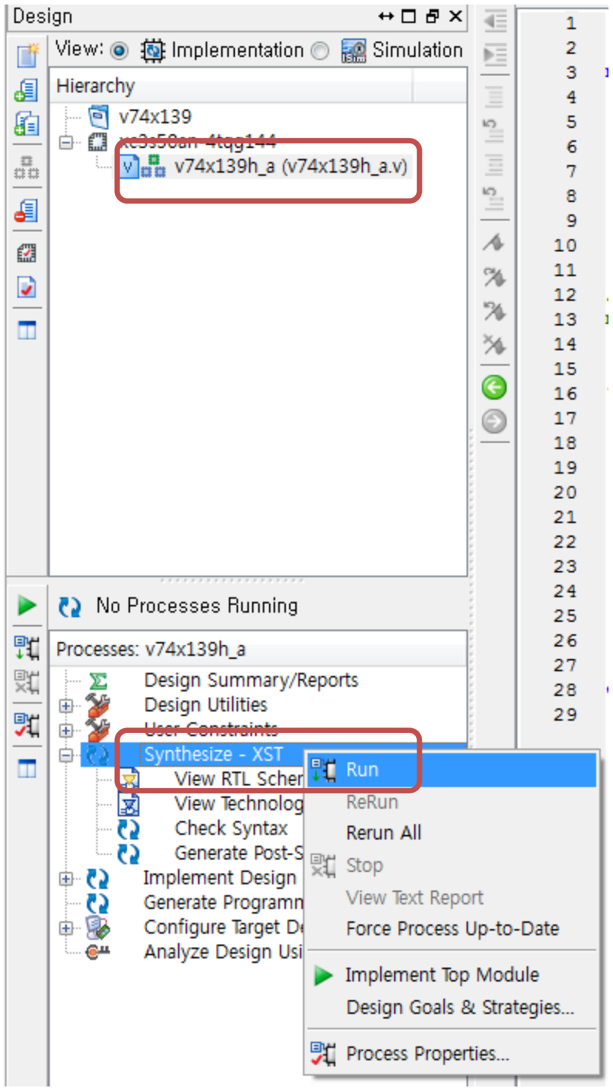
Port Name	Direction	Bus	MSB	LSB
G_L	in	☐		
A	in	☐		
B	in	☐		
Y_L	out	✓	3	0
	in	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		

Write your own Verilog codes

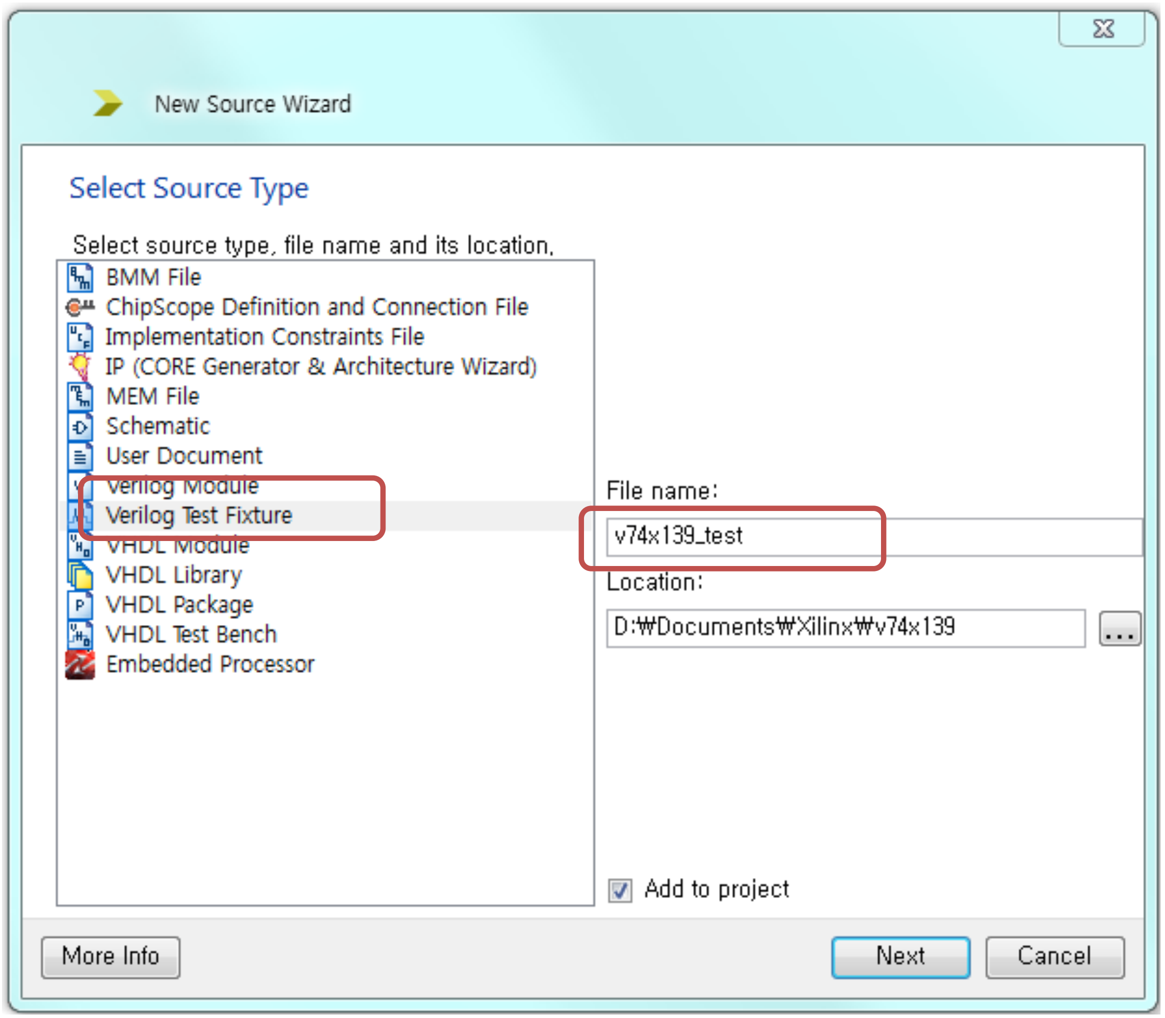


```verilog
`timescale 1ns / 1ps

module v74x139h_a(
   input G_L,
   input A,
   input B,
   output Y0_L,
   output Y1_L,
   output Y2_L,
   output Y3_L
   );
/* You can write as below either
module v74x139h_a(G_L,A,B,Y0_L,Y1_L,Y2_L,Y3_L);
    input G_L, A, B;
    output Y0_L, Y1_L, Y2_L, Y3_L;
*/
    wire A_L, B_L, G, A_i, B_i;

    not U1(G, G_L);
    not U2(B_L, B);
    not U3(A_L, A);
    not U4(B_i, B_L);
    not U5(A_i, A_L);
    nand U6(Y0_L, A_L, B_L, G);
    nand U7(Y1_L, G, B_L, A_i);
    nand U8(Y2_L, G, A_L, B_i);
    nand U9(Y3_L, G, A_i, B_i);
endmodule
```

Compile and check errors



Create a Verilog test bench


New Source Wizard

Select Source Type

Select source type, file name and its location.

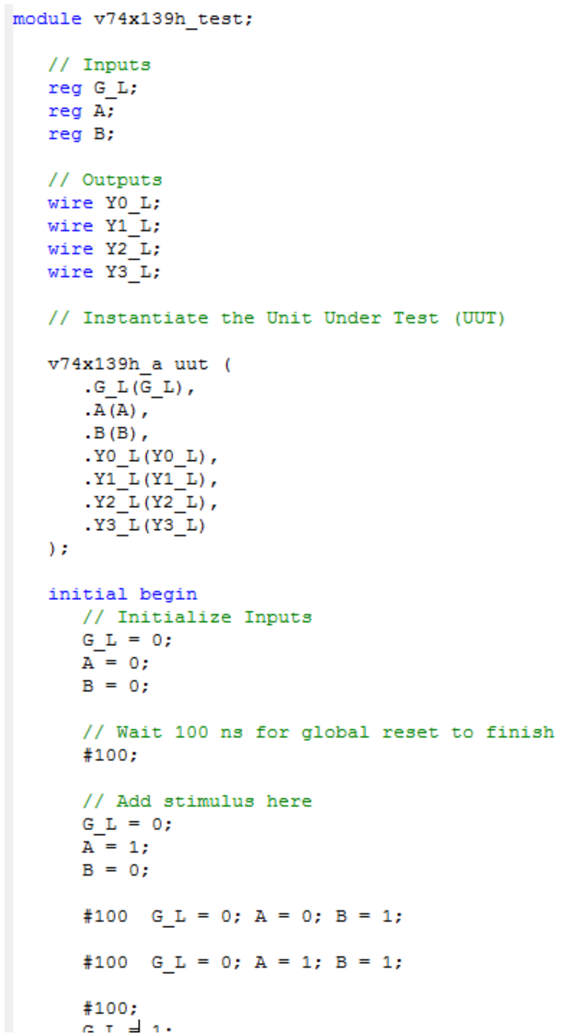
- BMM File
- ChipScope Definition and Connection File
- Implementation Constraints File
- IP (CORE Generator & Architecture Wizard)
- MEM File
- Schematic
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

File name: v74x139_test

Location: D:₩Documents₩Xilinx₩v74x139

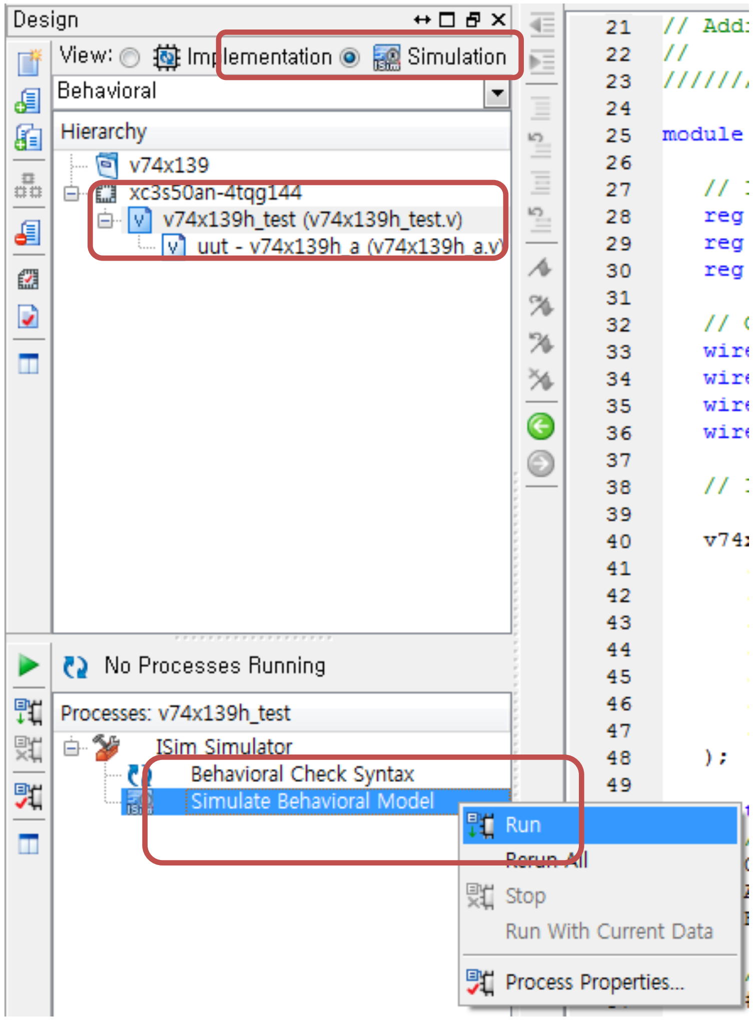
[✓] Add to project

Write a Verilog test bench codes



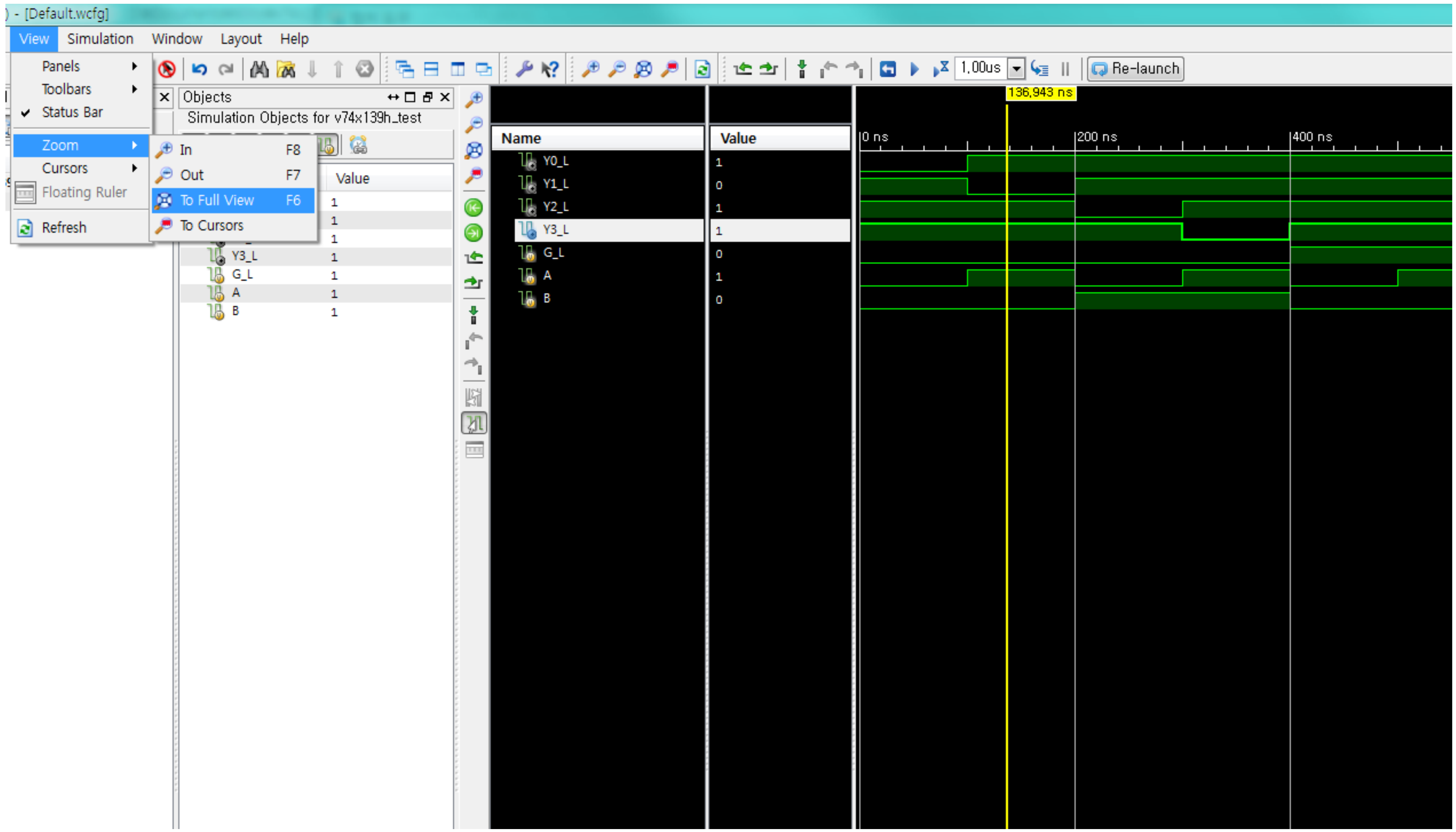
```
module v74x139h_test;

   // Inputs
   reg G_L;
   reg A;
   reg B;

   // Outputs
   wire Y0_L;
   wire Y1_L;
   wire Y2_L;
   wire Y3_L;

   // Instantiate the Unit Under Test (UUT)

   v74x139h_a uut (
      .G_L(G_L),
      .A(A),
      .B(B),
      .Y0_L(Y0_L),
      .Y1_L(Y1_L),
      .Y2_L(Y2_L),
      .Y3_L(Y3_L)
   );

   initial begin
      // Initialize Inputs
      G_L = 0;
      A = 0;
      B = 0;

      // Wait 100 ns for global reset to finish
      #100;

      // Add stimulus here
      G_L = 0;
      A = 1;
      B = 0;

      #100  G_L = 0; A = 0; B = 1;

      #100  G_L = 0; A = 1; B = 1;

      #100;
      G_L = 1;
```

Simulate it



```
// Add
//
//////
module
    // 
    reg
    reg
    reg
    // 
    wire
    wire
    wire
    wire
    // 
    v74
    );
```

Simulation result



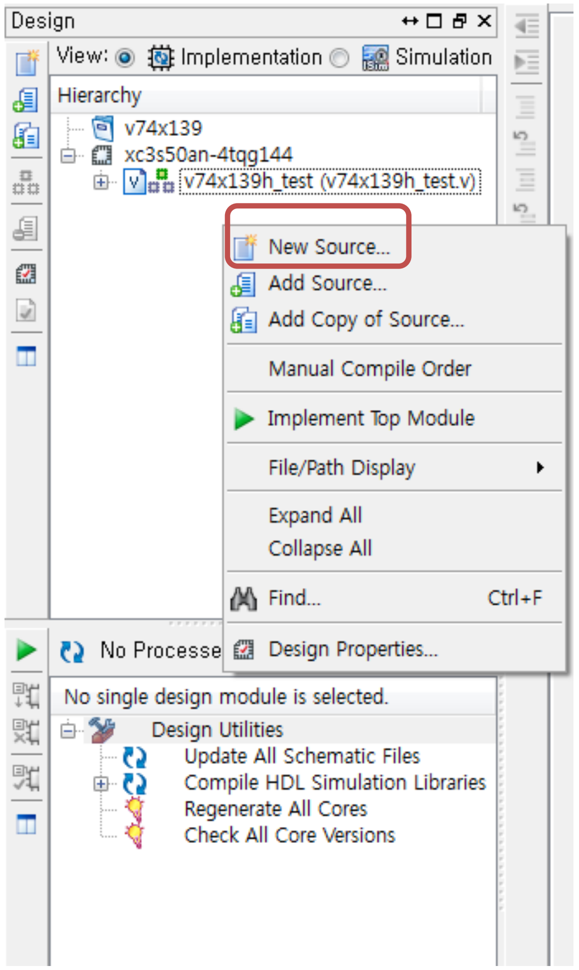
Objects

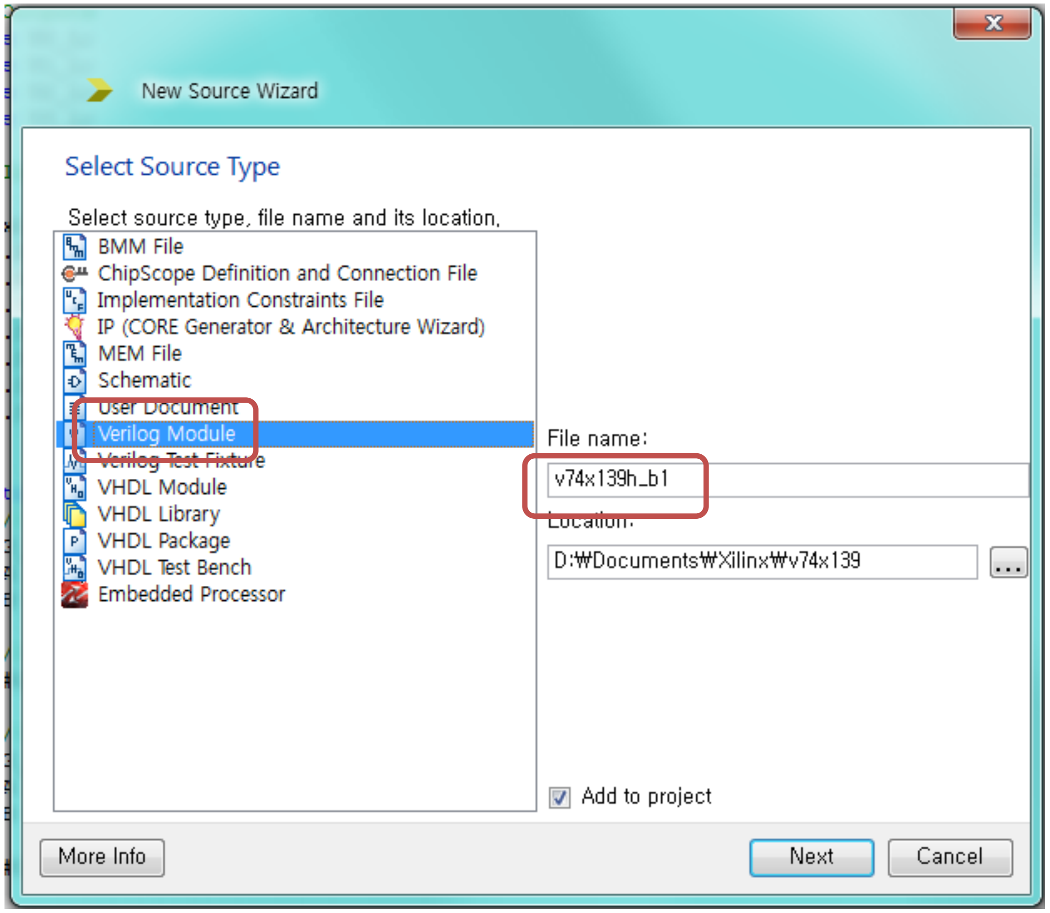
Simulation Objects for v74x139h_test

	Value
	1
	1
	1
Y3_L	1
G_L	1
A	1
B	1

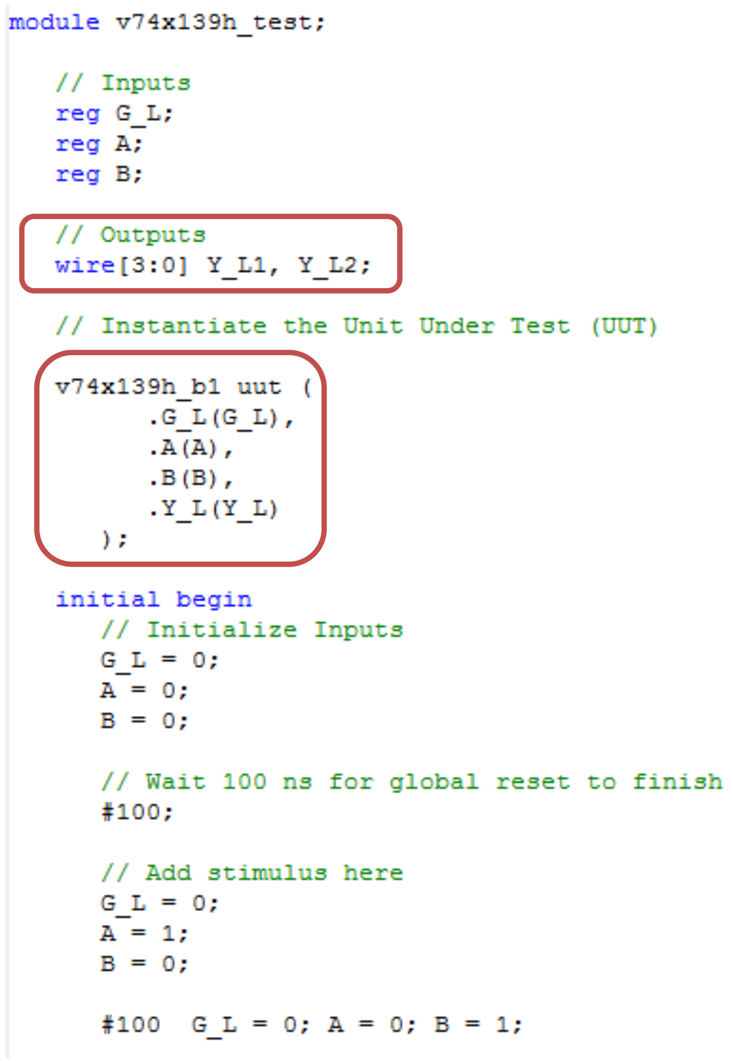
Name	Value
Y0_L	1
Y1_L	0
Y2_L	1
Y3_L	1
G_L	0
A	1
B	0

Create a new Verilog source





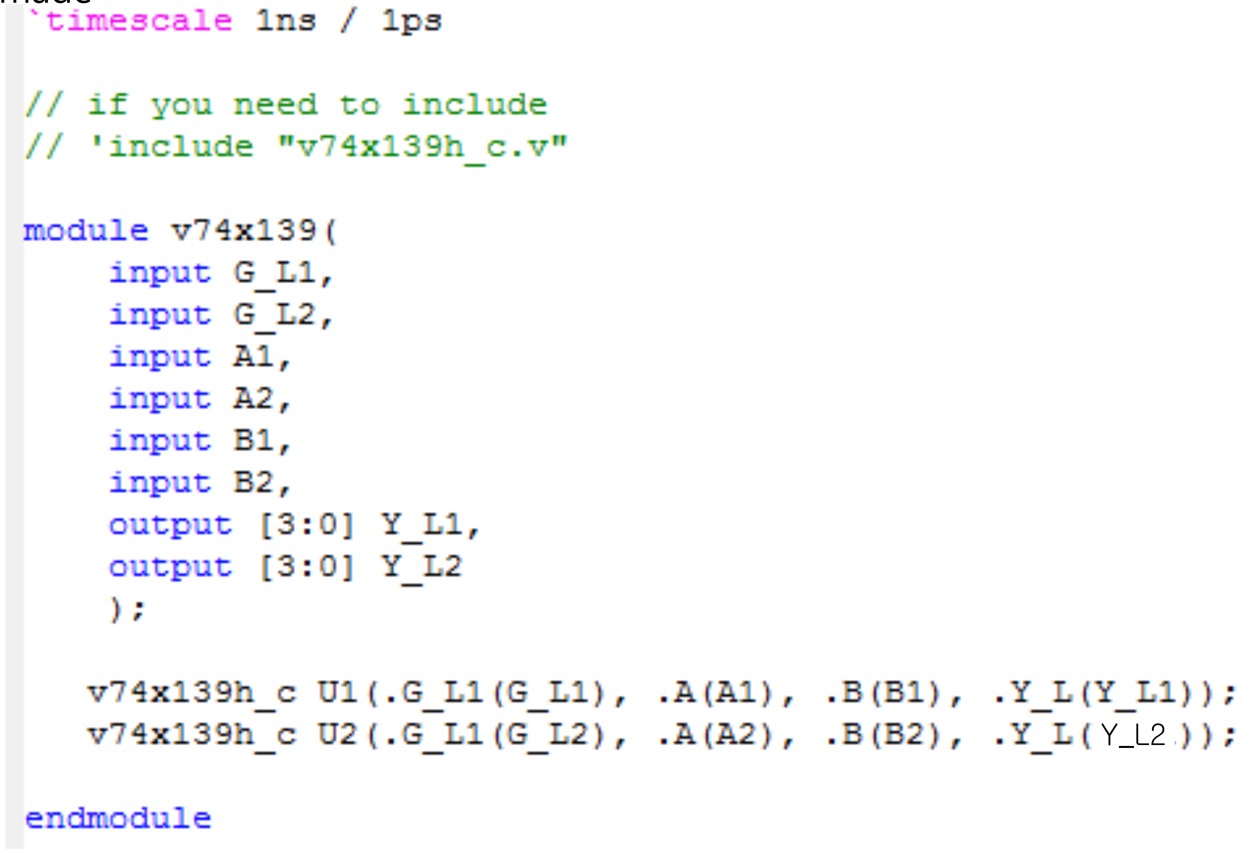
Re-write a Verilog test bench codes



```
module v74x139h_test;

   // Inputs
   reg G_L;
   reg A;
   reg B;

   // Outputs
   wire[3:0] Y_L1, Y_L2;

   // Instantiate the Unit Under Test (UUT)

   v74x139h_b1 uut (
         .G_L(G_L),
         .A(A),
         .B(B),
         .Y_L(Y_L)
      );

   initial begin
      // Initialize Inputs
      G_L = 0;
      A = 0;
      B = 0;

      // Wait 100 ns for global reset to finish
      #100;

      // Add stimulus here
      G_L = 0;
      A = 1;
      B = 0;

      #100  G_L = 0; A = 0; B = 1;
```

Designing 74x139(Dual 2-to-4 decoder)

- **Hierarchical Design (2 × ½ 74x139)**
  - You make higher level module with the modules you already made



```
`timescale 1ns / 1ps

// if you need to include
// 'include "v74x139h_c.v"

module v74x139(
    input G_L1,
    input G_L2,
    input A1,
    input A2,
    input B1,
    input B2,
    output [3:0] Y_L1,
    output [3:0] Y_L2
    );

   v74x139h_c U1(.G_L1(G_L1), .A(A1), .B(B1), .Y_L(Y_L1));
   v74x139h_c U2(.G_L1(G_L2), .A(A2), .B(B2), .Y_L( Y_L2 ));

endmodule
```

Homework

Homework

1. Implement 3-to-8 decoder and simulate it
   - You have to re-use your 2-to-4 decoder implemented in lab

2. Implement 4-to-1 MUX and simulate it
   - Implement in gate and rtl/behavior level
   - Discuss two methods(Pros & Cons, etc.)

3. Implement 16-to-1 MUX and simulate it
   - You have to re-use your 4-to-1 MUX implemented above

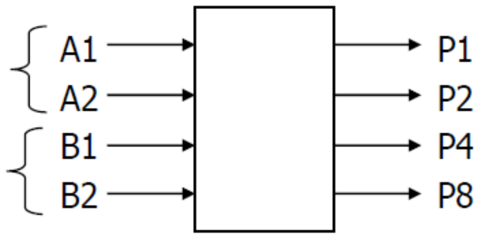
4. Given a four-input Boolean function

   $$F(A,B,C,D) = \Sigma\ m(0,2,4,5,8,10,12,13,14,15),$$

   implement the function using a 16-to-1 MUX
   - You have to re-use YOUR 16-to-1 MUX implemented above

Homework(Cont'd)

5. Design a 2x2-bit multiplier.
   - Implement in Verilog and simulate it



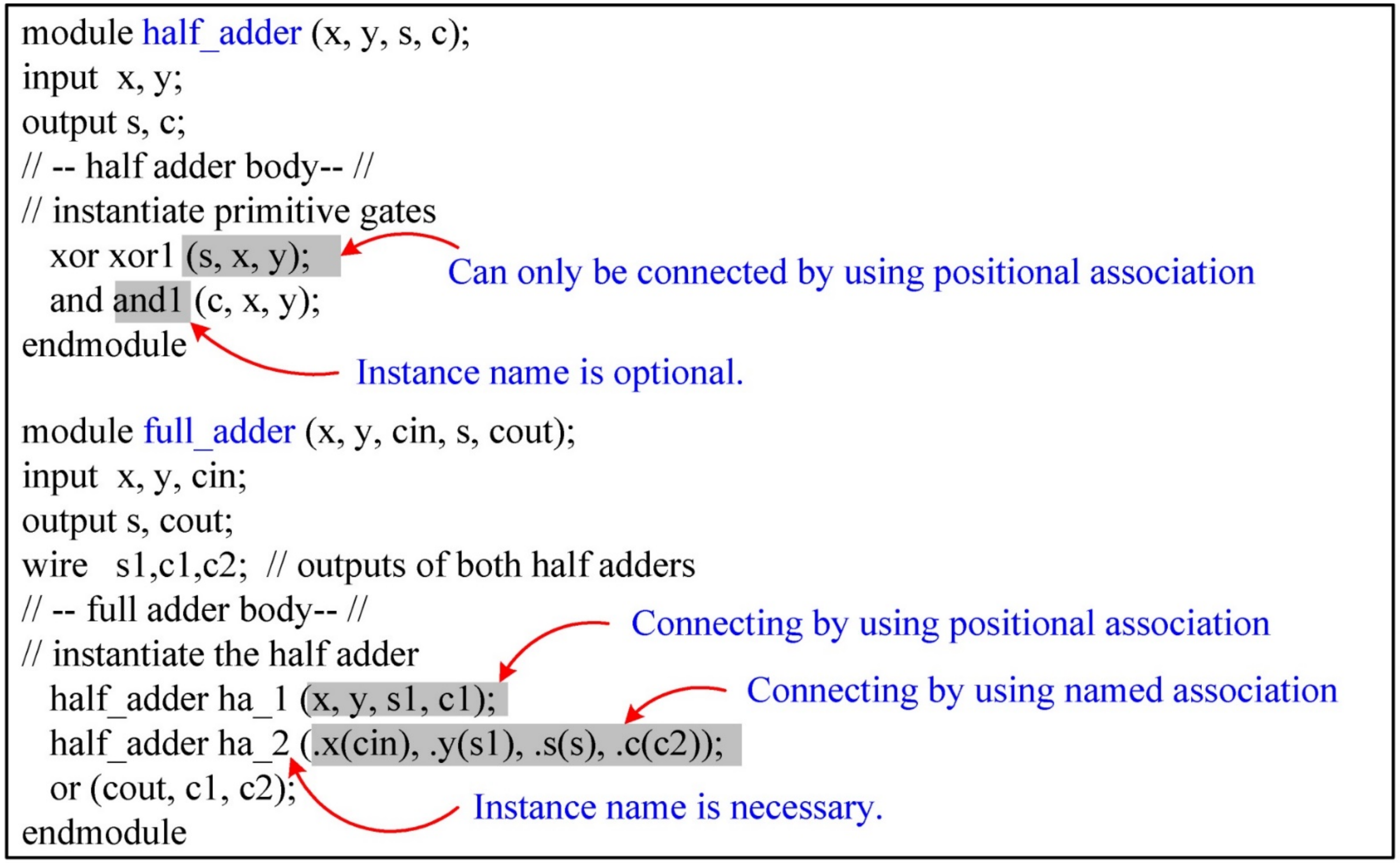
(TIP) This is a 2x2-bit multiplier that generates 4 bit output (whose MSB is P8 and LSB is P1). Note that A2 and B2 are MSBs.

Report

- Write a report
  - # of pages doesn't matter
  - Include **codes** and **simulation result**
  - The file size should not exceed 15MB
  - **Due : 7 Oct. (Before class begin at 7:00pm)**

Appendix

An Example: Port Connection



```
module half_adder (x, y, s, c);
input  x, y;
output s, c;
// -- half adder body-- //
// instantiate primitive gates
   xor xor1 (s, x, y);
   and and1 (c, x, y);
endmodule

module full_adder (x, y, cin, s, cout);
input  x, y, cin;
output s, cout;
wire   s1,c1,c2;  // outputs of both half adders
// -- full adder body-- //
// instantiate the half adder
   half_adder ha_1 (x, y, s1, c1);
   half_adder ha_2 (.x(cin), .y(s1), .s(s), .c(c2));
   or (cout, c1, c2);
endmodule
```

Can only be connected by using positional association

Instance name is optional.

Connecting by using positional association

Connecting by using named association

Instance name is necessary.