Lab. 03

Logic Design Lab.

Spring 2022

Prof. ChangGun Lee

(cglee@snu.ac.kr)

TA. Hayeon Park

TA. Haejoo Jeon

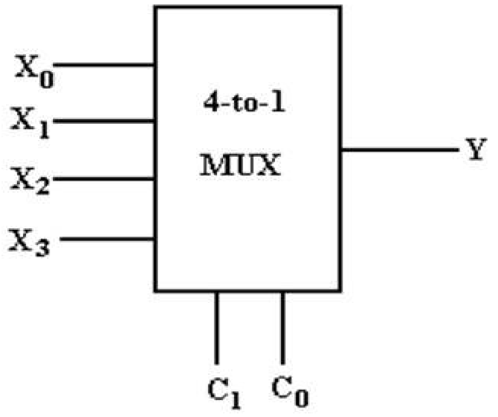
TA. Seohwan Yoo

(rubis.ld.ta@gmail.com)

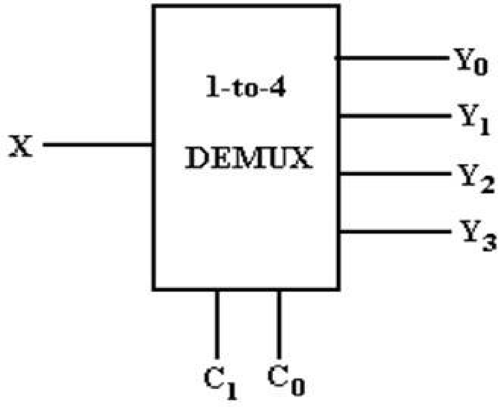
Contents

Multiplexer, Demultiplexer

Multiplexer and Demultiplexer



C_1	C_0	M
0	0	X_0
0	1	X_1
1	0	X_2
1	1	X_3



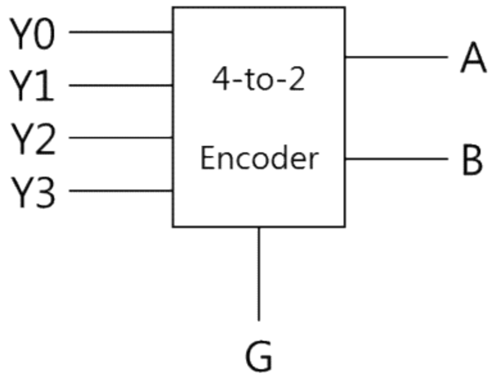
C_1	C_0	Selected Output
0	0	$Y_0 = X$ Other outputs 0
0	1	$Y_1 = X$ Other outputs 0
1	0	$Y_2 = X$ Other outputs 0
1	1	$Y_3 = X$ Other outputs 0

- 2^n inputs, 1 outputs
- Selection lines exist

- 1 inputs, 2^n outputs
- Selection lines exist

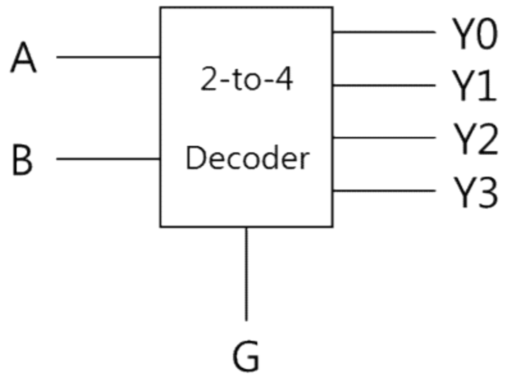
Encoder, Decoder

Encoder, Decoder



G	Y0	Y1	Y2	Y3	A	B
1	1	0	0	0	0	0
1	0	1	0	0	0	1
1	0	0	1	0	1	0
1	0	0	0	1	1	1

- 2^n inputs, n outputs
- Outputs the binary value of the selected input
- No selection lines exist



G	A	B	Y0	Y1	Y2	Y3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

- n inputs, 2^n outputs
- Selects one of 2^n outputs by decoding the binary value on the n inputs
- No selection lines exist

Verilog

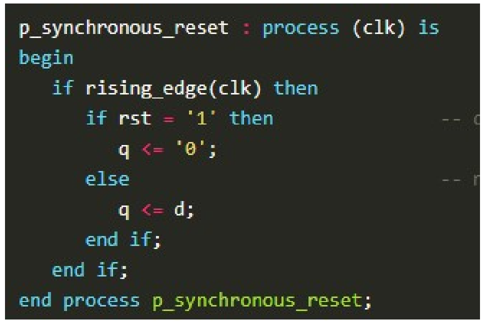
Overview

- Hardware description languages (HDLs)
  - We will focus on **Verilog**
- Verilog Basics
  - Verilog **Notations**
  - Verilog **Operators**
  - Verilog **Keywords & Constructs**
- Types of programming (description)
  - **Structural** description
  - **Data-flow style** description
  - **Behavioral** description

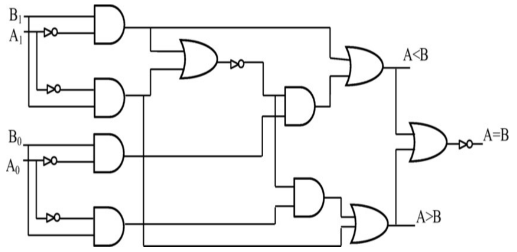
FPGA development process

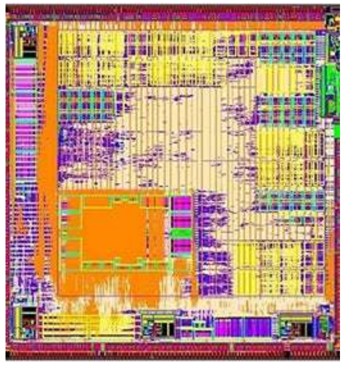
- FPGA(Field-Programmable Gate Array)

$$\begin{tikzcd}
\text{Requirements Specification} \arrow[d] \\
\text{RTL Design} \arrow[d, "\text{Synthesis}"] \\
\text{Gate-level Design} \arrow[d, "\text{Place \& Route}"] \\
\text{Layout} \arrow[d, "\text{Download}"] \\
\text{FPGA}
\end{tikzcd}$$

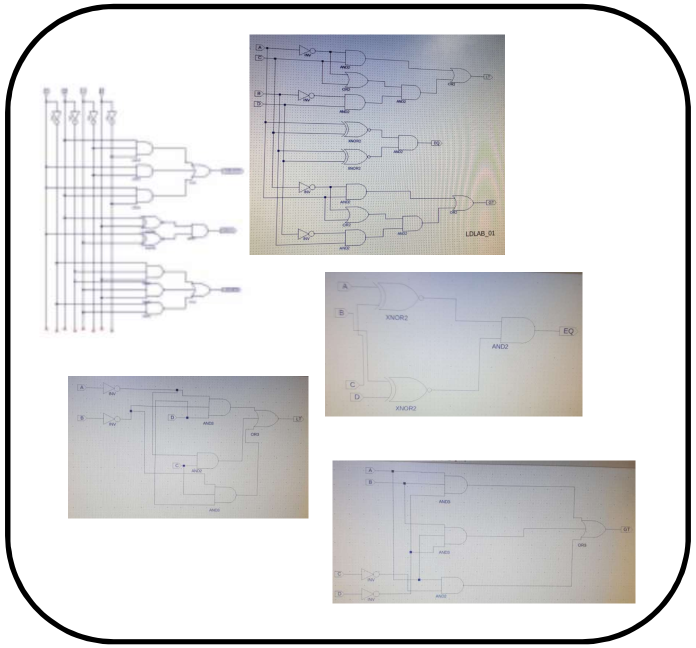


```
p_synchronous_reset : process (clk) is
begin
    if rising_edge(clk) then
        if rst = '1' then                       -- c
            q <= '0';
        else                                    -- r
            q <= d;
        end if;
    end if;
end process p_synchronous_reset;
```





Hardware description languages (HDLs)



2-bit Comparator

- Hard to understand
- Too many papers…

  ➔ Let's make standard HDL !

Verilog Basics : Verilog Notations (1)

- Verilog is Hardware Description Language
- Verilog is:
  - Case sensitive
  - Based on the programming language C

- Comments:

// this is a comment	Single line
/* this is a comment */	Multiple line

- Statement terminator:
  - Every statement should be terminated with ; (semi-colon)

Verilog Basics : Verilog Notations (2)

- Literals
  - **[bit_size]'[base_format][value]**

    Example > below literals represent same value

    8'b10100001

    8'hA1

    8'd161
- Identifiers
  - **Scalar (always 1 bit)**

    Example > below identifiers are similar with single variable in C

    A, B, myA, myB…
  - **Vector (bit size is specified as [high_bit:low_bit])**

    Example > below identifiers are similar with array variable in C

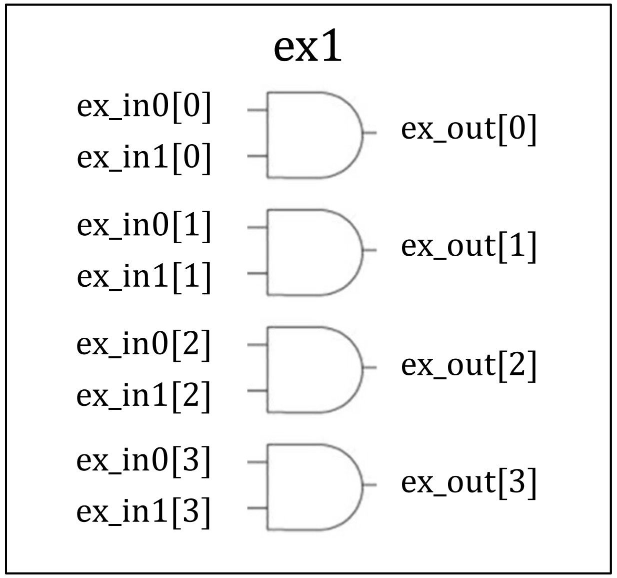
    arrA[7:0], arrB[15:0], arrMyA[7:0], arrMyB[15:0]…

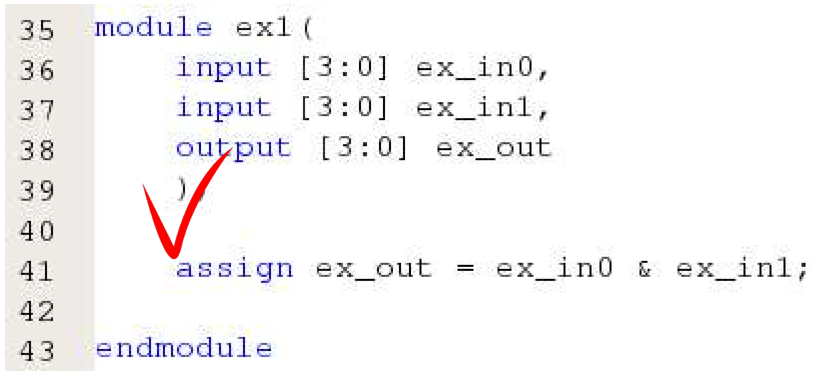
Verilog Basics : Verilog Operators (1)

- Bitwise Operators

~	NOT
&	AND
|	OR
^	XOR

Example >





```
module ex1(
     input  [3:0] ex_in0,
     input  [3:0] ex_in1,
     output  [3:0] ex_out
     );

     assign ex_out = ex_in0 & ex_in1;

endmodule
```

Verilog Basics : Verilog Operators (2)

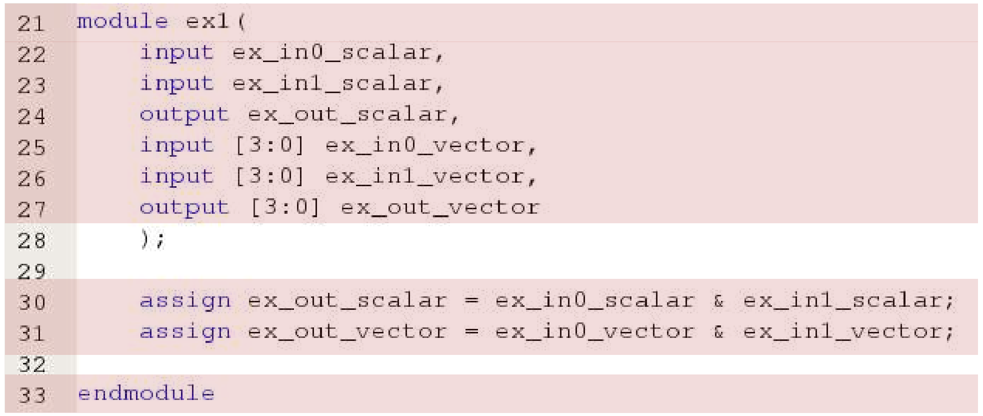
- Logical & Relational Operators
  - &&, | |, = =, !=, >=, <=, >, <, etc.
- Arithmetic Operators
  - +, -, etc.
- Conditional Operators
  - Condition ? expression1 : expression2;
- Concatenation, Shift, Reduction…

Verilog Basics : Verilog Keywords & Constructs (1)

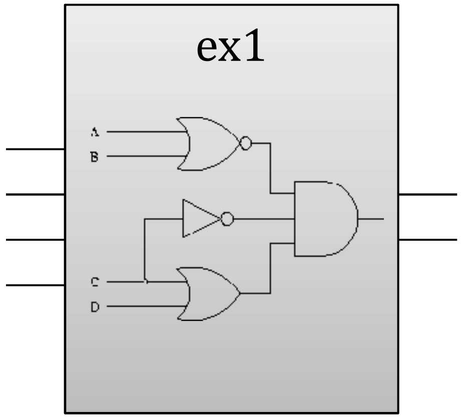
- Module declaration
  - Start with module, end with endmodule

- Module IO port declaration
  - Inputs use keyword, input, Outputs use keyword output.

Example >



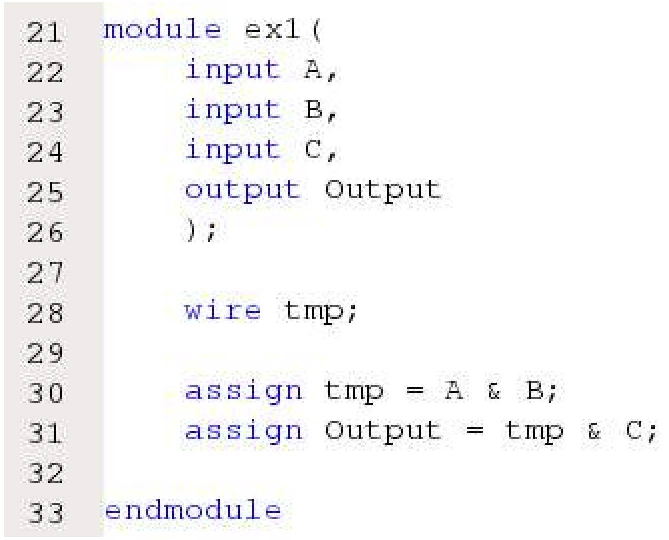
```
module ex1(
    input ex_in0_scalar,
    input ex_in1_scalar,
    output ex_out_scalar,
    input [3:0] ex_in0_vector,
    input [3:0] ex_in1_vector,
    output [3:0] ex_out_vector
    );

    assign ex_out_scalar = ex_in0_scalar & ex_in1_scalar;
    assign ex_out_vector = ex_in0_vector & ex_in1_vector;

endmodule
```



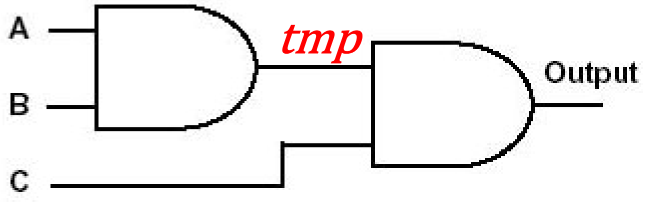
Verilog Basics : Verilog Keywords & Constructs (2)

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

Example >



```
module ex1(
      input A,
      input B,
      input C,
      output Output
      );

      wire tmp;

      assign tmp = A & B;
      assign Output = tmp & C;

endmodule
```

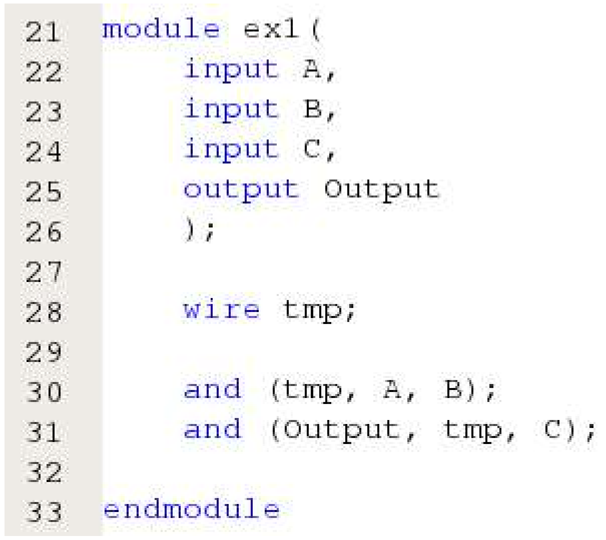


Verilog Basics : Verilog Keywords & Constructs (3)

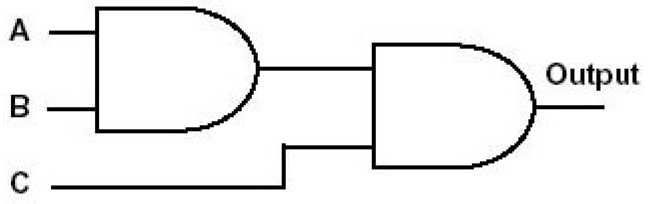
- Primitive Gates
  - not, and, or, nand, nor, xor, xnor
  - Syntax:

    [gate_operator] [instance_identifier] [(output, input_1, input_2, …)]

Example >



```verilog
module ex1(
    input A,
    input B,
    input C,
    output Output
    );

    wire tmp;

    and (tmp, A, B);
    and (Output, tmp, C);

endmodule
```



Verilog Basics : Verilog Keywords & Constructs (4)

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

    - initial – executes <u>only once</u> beginning at t = 0.
      - Syntax:
        initial *Statement;*
      - Example >

        ```
        initial begin
          Statement;
          Statement;
          ...
        end
        ```

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

        ```
        always Repeat condition
          Statement;
        ```
      - Example >

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

Verilog Basics : Verilog Keywords & Constructs (5)

- 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
	@(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 Basics : Verilog Keywords & Constructs (6)

Examples > blocking vs. Non-blocking

```
Always @(*)
    begin
   B  = A;
   C  = B;
  end
```

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

```
Always @(*)
    begin
        B <= A;
        C <= B;
    end
```

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

Verilog Basics : Verilog Keywords & Constructs (7)

- Because of the use of procedural rather than continuous assignment statements, assigned values must be retained over time.
  - Register type: reg
  - The reg in contrast to wire stores values between executions of the process
  - A reg type does not imply hardware storage!

Verilog Basics : Verilog Keywords & Constructs (8)

- 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

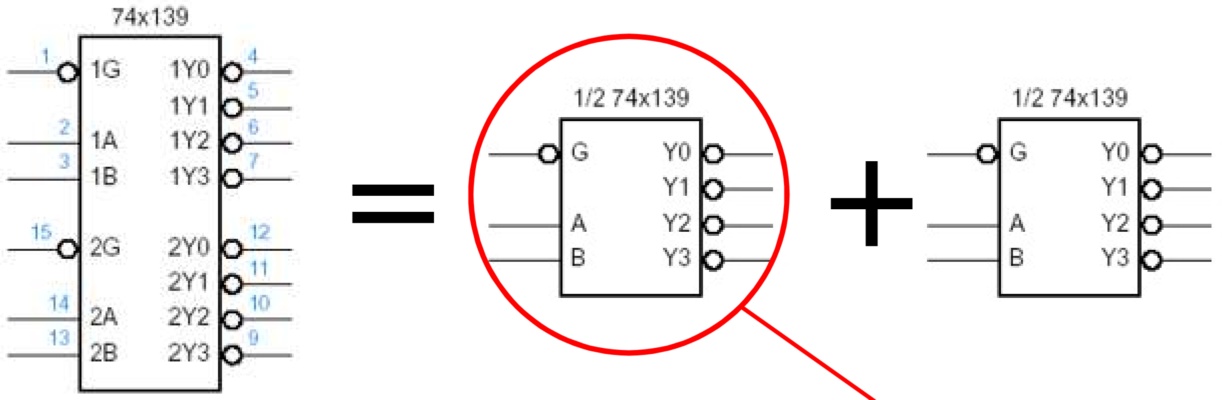
```
case expression
    {case expression : statements}
endcase;
```

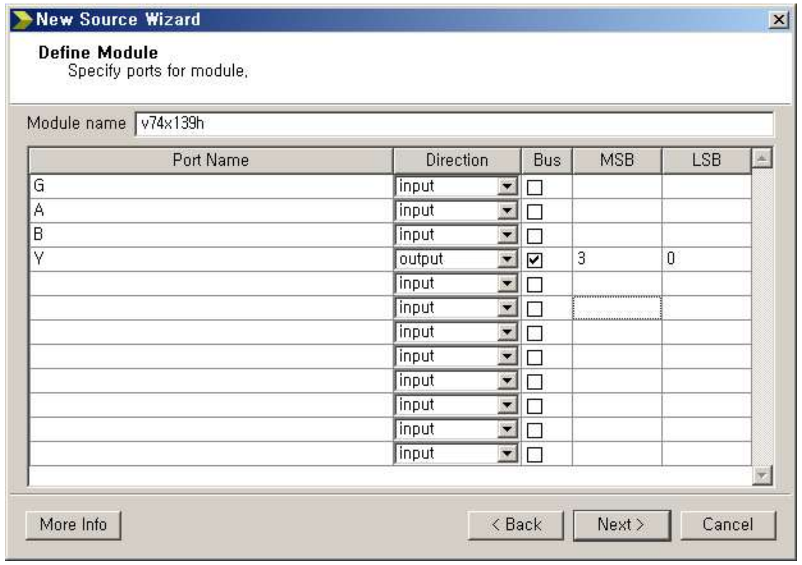
Types of programming (description)

- Describe hardware at varying levels of abstraction
  - **Structural description**
    - Textual replacement for schematic
  - **Data-flow style description**
    - Textual replacement of truth table
  - **Behavioral/functional description**
    - Define just WHEN, WHAT, not HOW

Tutorials

1. Design Structure

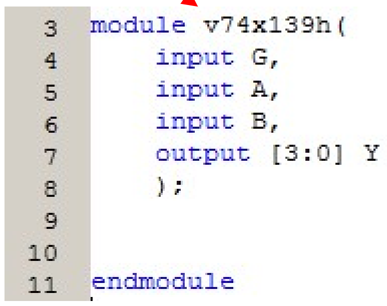




Define Module
Specify ports for module.

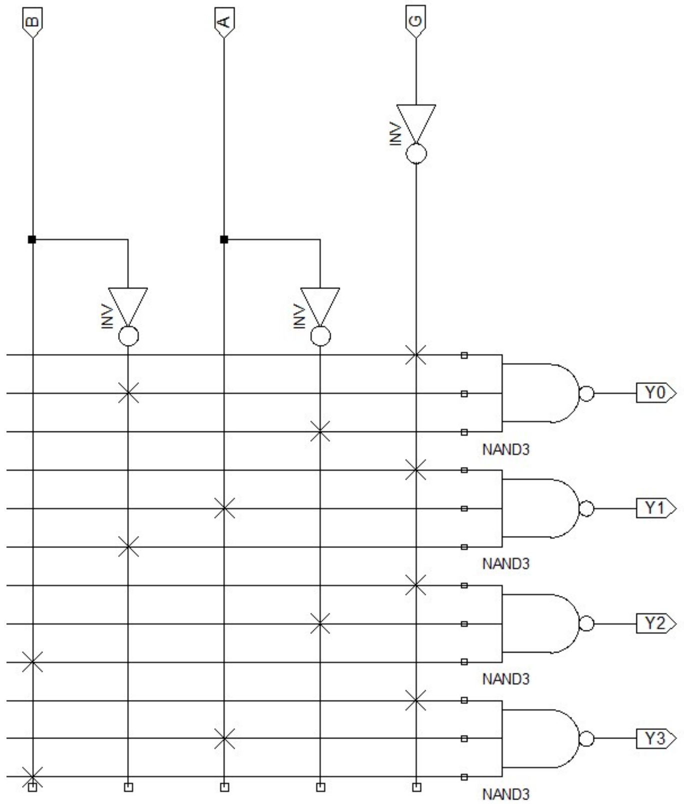
Module name: v74x139h

Port Name	Direction	Bus	MSB	LSB
G	input	☐		
A	input	☐		
B	input	☐		
Y	output	☑	3	0
	input	☐		
	input	☐		
	input	☐		
	input	☐		
	input	☐		
	input	☐		
	input	☐		
	input	☐		

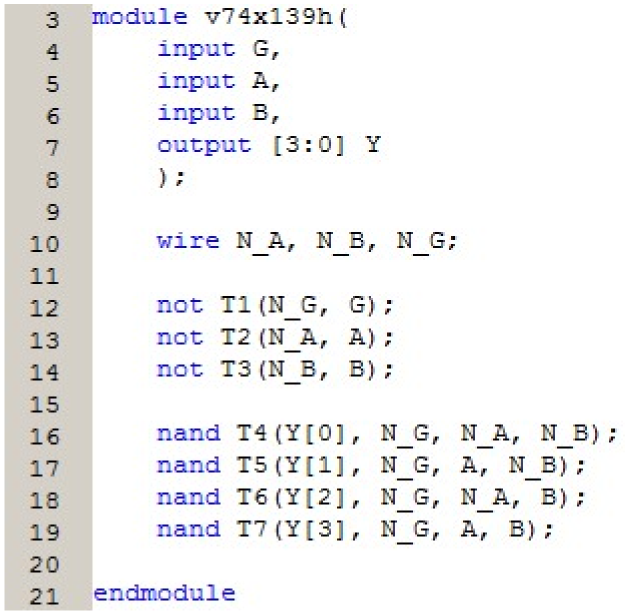


```
module v74x139h(
      input G,
      input A,
      input B,
      output [3:0] Y
      );


endmodule
```

2-1. Structural Description



N_B N_A N_G

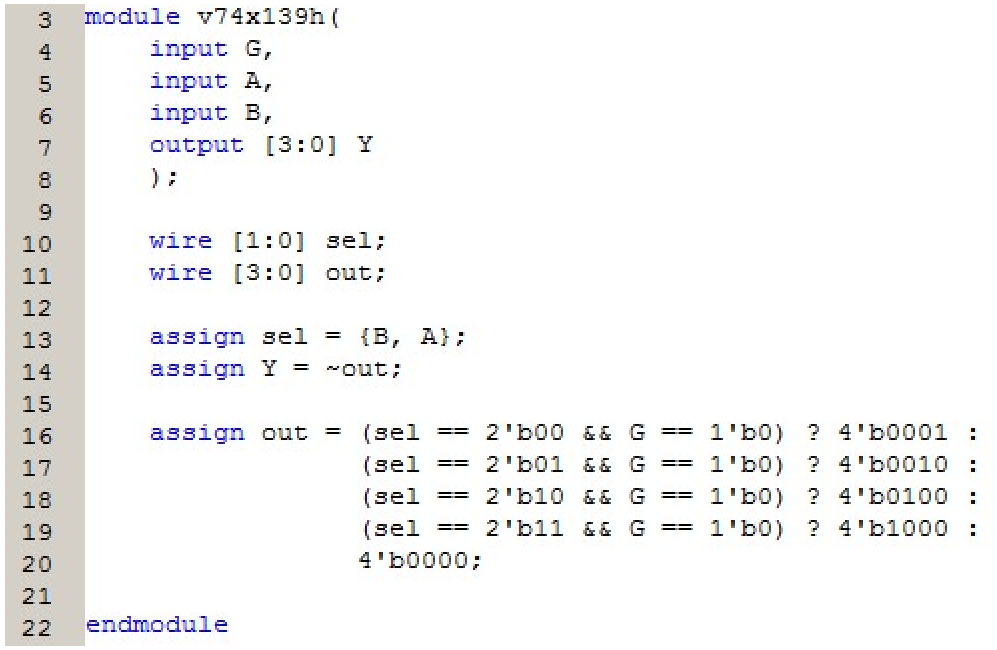


```
module v74x139h(
    input G,
    input A,
    input B,
    output [3:0] Y
    );

    wire N_A, N_B, N_G;

    not T1(N_G, G);
    not T2(N_A, A);
    not T3(N_B, B);

    nand T4(Y[0], N_G, N_A, N_B);
    nand T5(Y[1], N_G, A, N_B);
    nand T6(Y[2], N_G, N_A, B);
    nand T7(Y[3], N_G, A, B);

endmodule
```

Textual representation of schematic

2-2. Data Flow Description

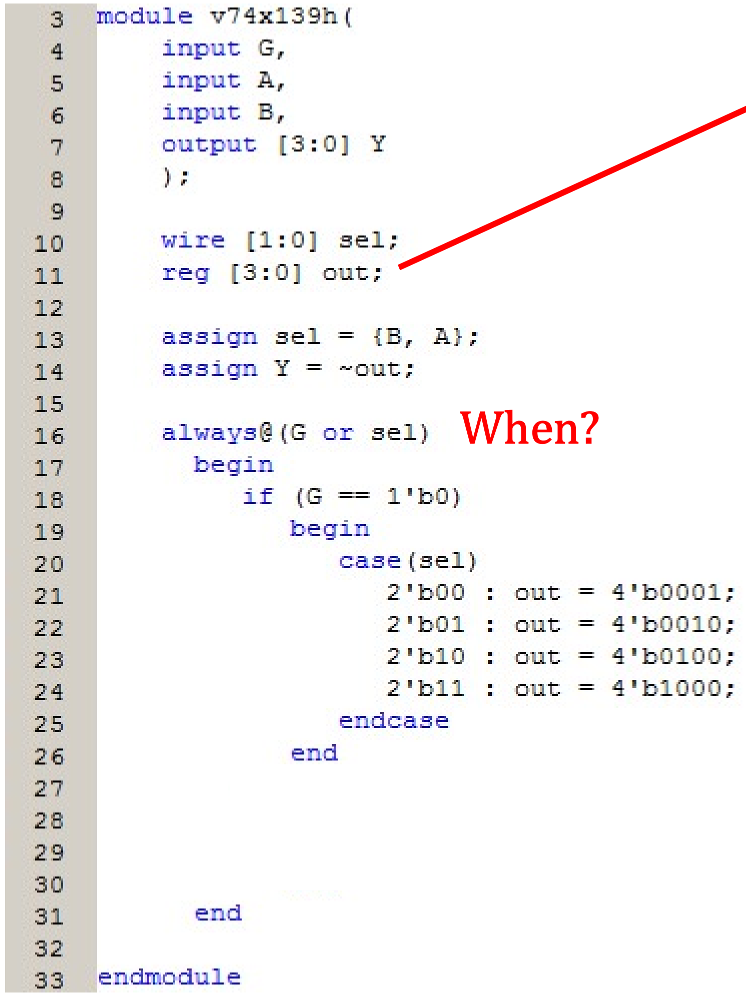
B	A	G	Y[3:0]
0	0	0	1110
0	1	0	1101
1	0	0	1011
1	1	0	0111
0	0	1	1111
0	1	1	1111
1	0	1	1111
1	1	1	1111



```
module v74x139h(
    input G,
    input A,
    input B,
    output [3:0] Y
    );

    wire [1:0] sel;
    wire [3:0] out;

    assign sel = {B, A};
    assign Y = ~out;

    assign out = (sel == 2'b00 && G == 1'b0) ? 4'b0001 :
                 (sel == 2'b01 && G == 1'b0) ? 4'b0010 :
                 (sel == 2'b10 && G == 1'b0) ? 4'b0100 :
                 (sel == 2'b11 && G == 1'b0) ? 4'b1000 :
                 4'b0000;

endmodule
```

Textual representation of truth table

2-3. Behavioral Description



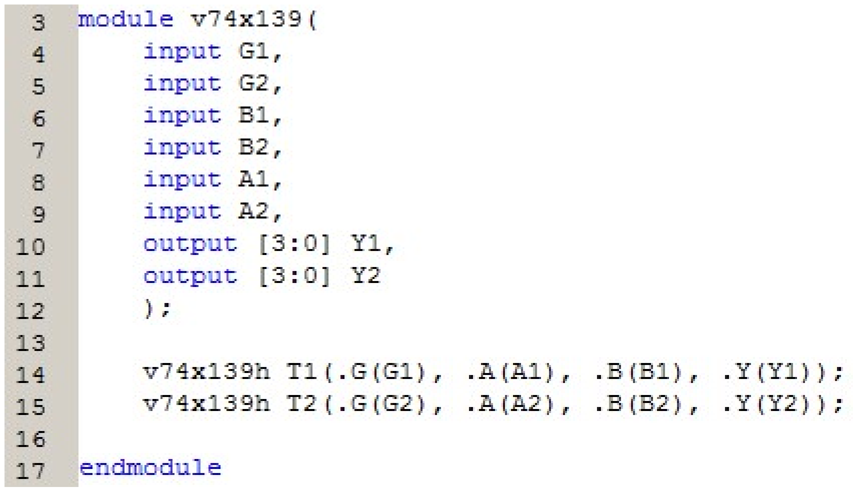
```verilog
module v74x139h(
    input G,
    input A,
    input B,
    output [3:0] Y
    );

    wire [1:0] sel;
    reg [3:0] out;

    assign sel = {B, A};
    assign Y = ~out;

    always@(G or sel)
      begin
         if (G == 1'b0)
            begin
               case(sel)
                  2'b00 : out = 4'b0001;
                  2'b01 : out = 4'b0010;
                  2'b10 : out = 4'b0100;
                  2'b11 : out = 4'b1000;
               endcase
            end




      end

endmodule
```

If there is no else-block,
reg out will be synthesized as storage

When?

Which action?

2-4. Unify Modules



```
module v74x139(
    input G1,
    input G2,
    input B1,
    input B2,
    input A1,
    input A2,
    output [3:0] Y1,
    output [3:0] Y2
    );

    v74x139h T1(.G(G1), .A(A1), .B(B1), .Y(Y1));
    v74x139h T2(.G(G2), .A(A2), .B(B2), .Y(Y2));

endmodule
```

It is same as using gate primitives

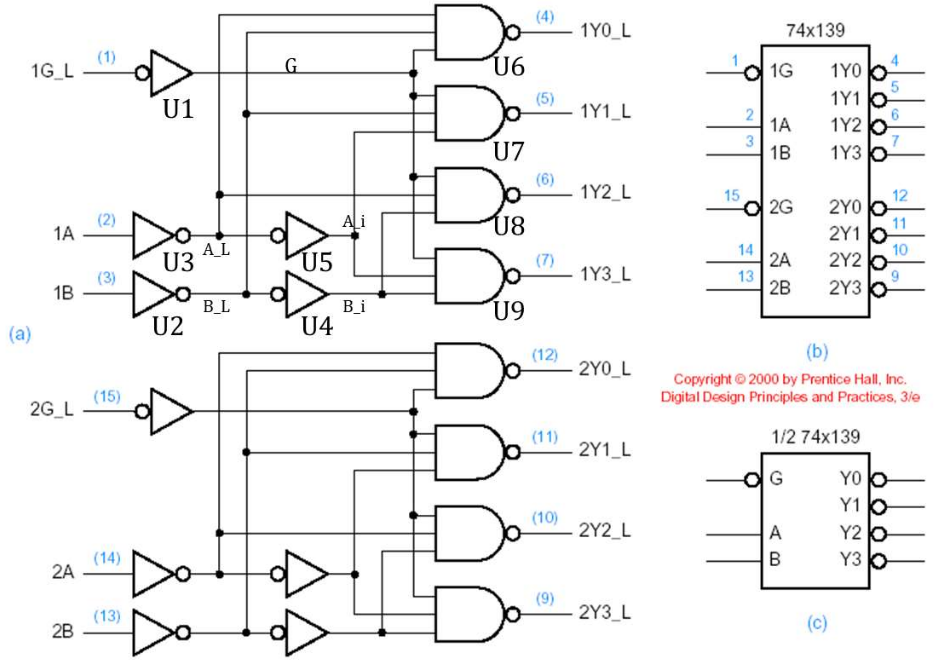
Lab

Today

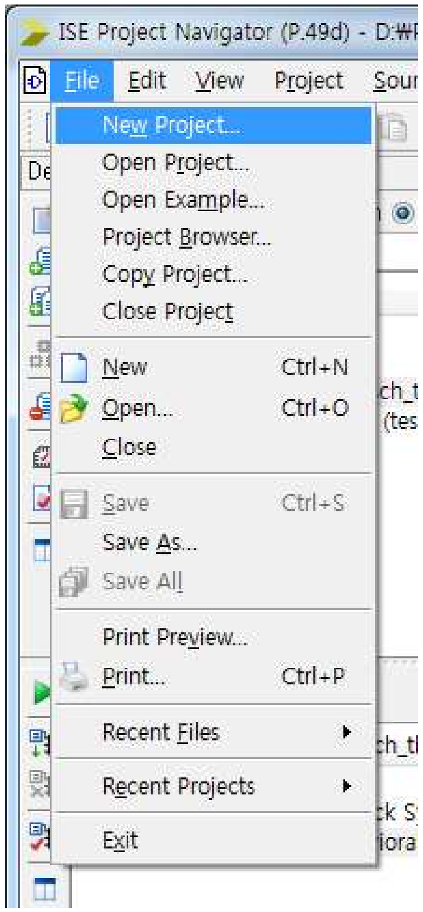
1. Design a 74x139 using Verilog.

1) Practice all the designing methods for half 74x139 each and simulate them
2) Implement a 74x139 and simulate it

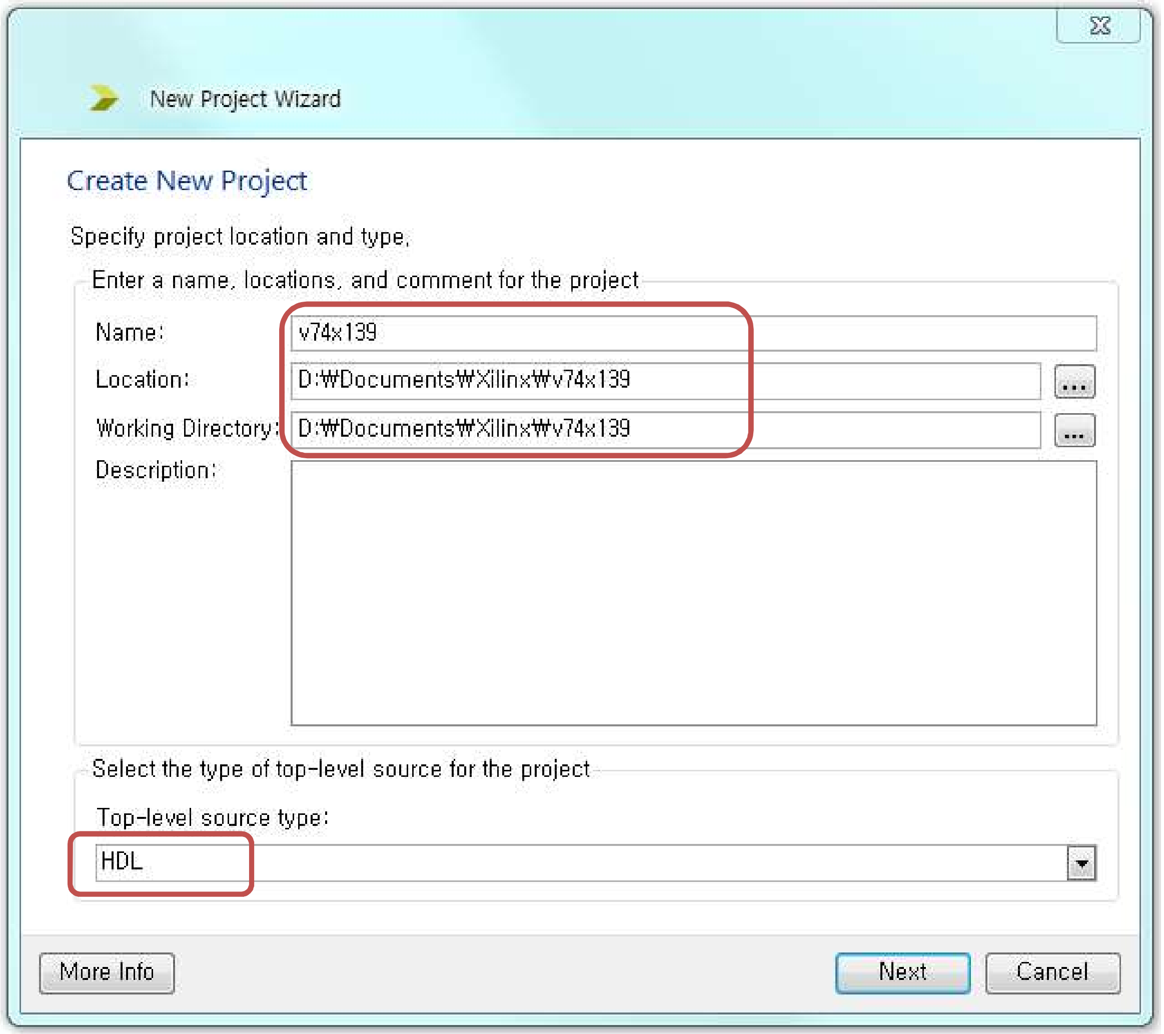
2. Design a 3-to-8 decoder using 2-to-4 decoders only and simulate it.
(one of 3 designing methods)



Create a new Verilog project



Set the project name, location, type



New Project Wizard

Create New Project

Specify project location and type.

Enter a name, locations, and comment for the project

Name: v74x139

Location: D:₩Documents₩Xilinx₩v74x139

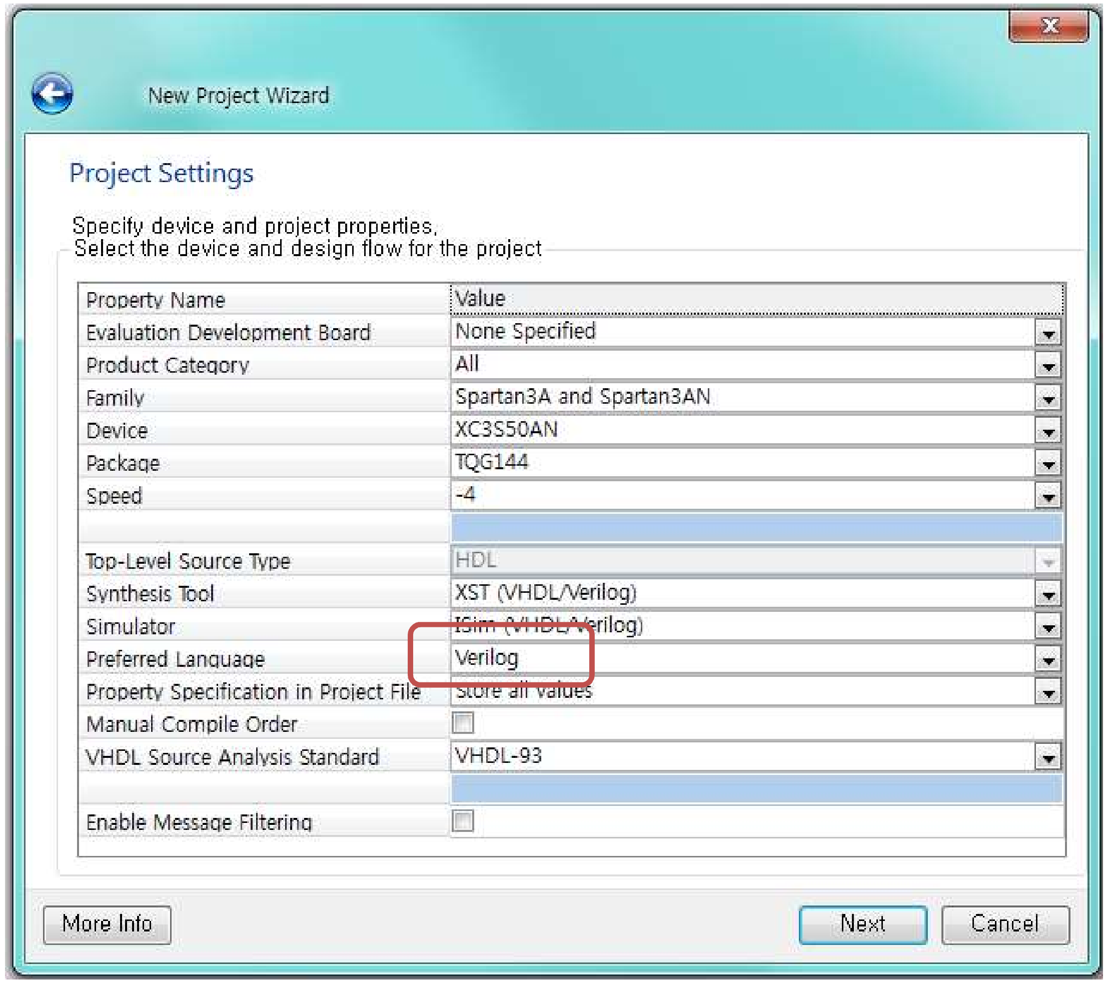
Working Directory: D:₩Documents₩Xilinx₩v74x139

Description:

Select the type of top-level source for the project

Top-level source type: HDL

Set the project name, location, type

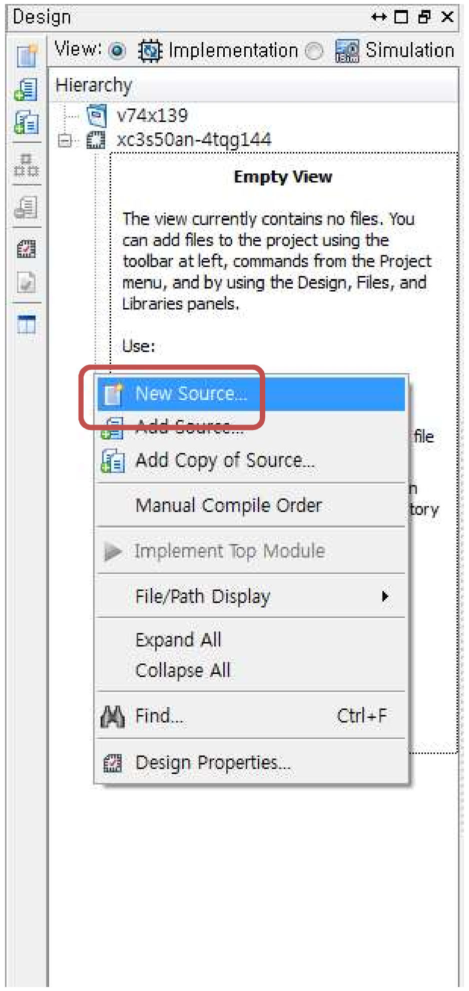


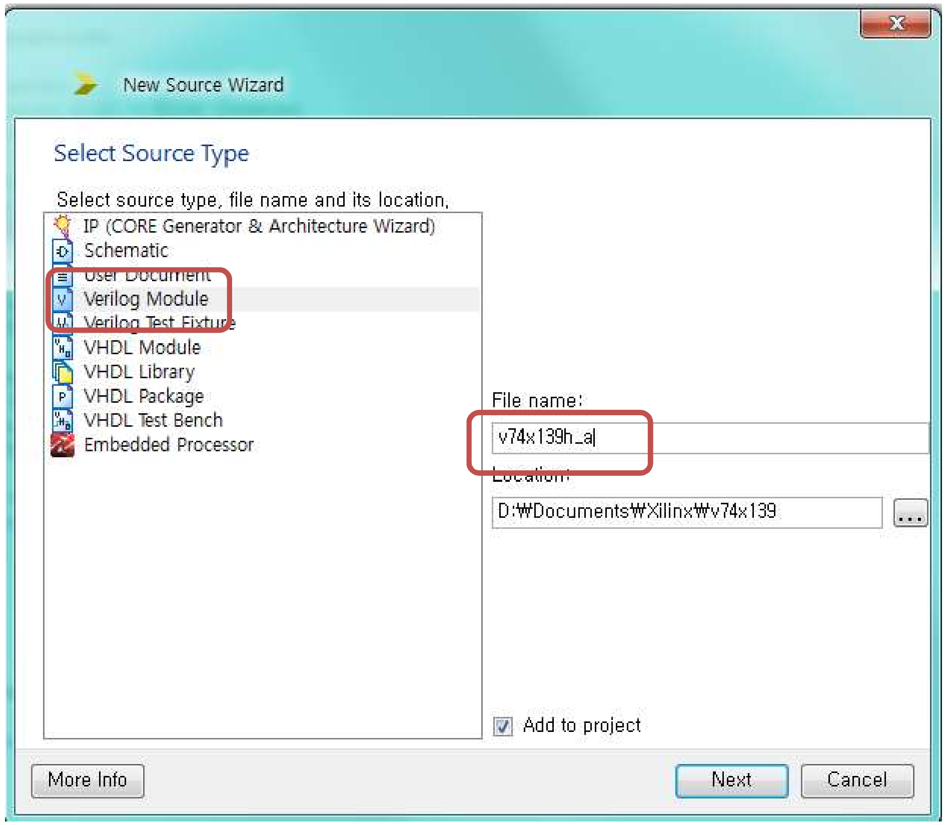
Project Settings

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

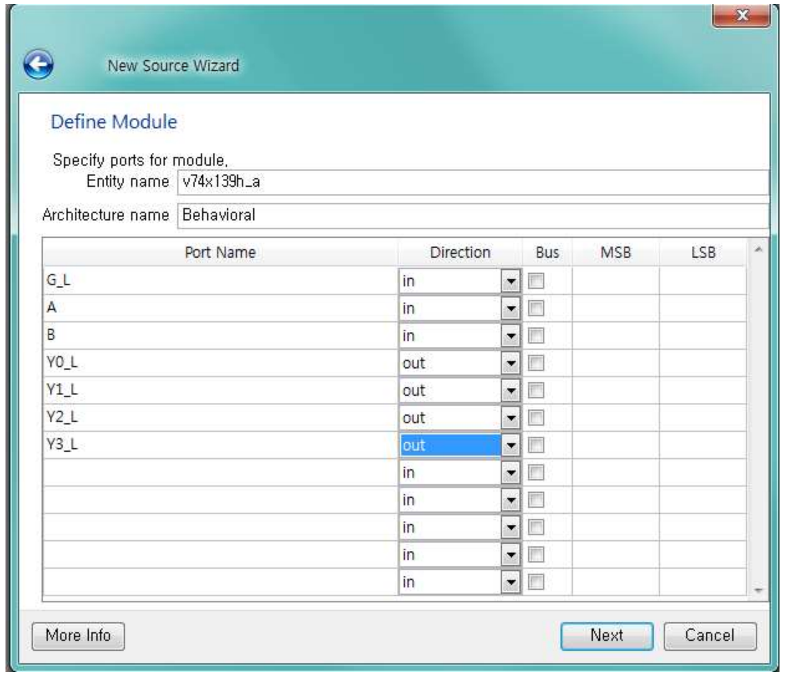
Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan3A and Spartan3AN
Device	XC3S50AN
Package	TQG144
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	Verilog
Property Specification in Project 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...
Add Source...
Add Copy of Source...
Manual Compile Order
Implement Top Module
File/Path Display
Expand All
Collapse All
Find... Ctrl+F
Design Properties...



Create a new Verilog source



Define Module

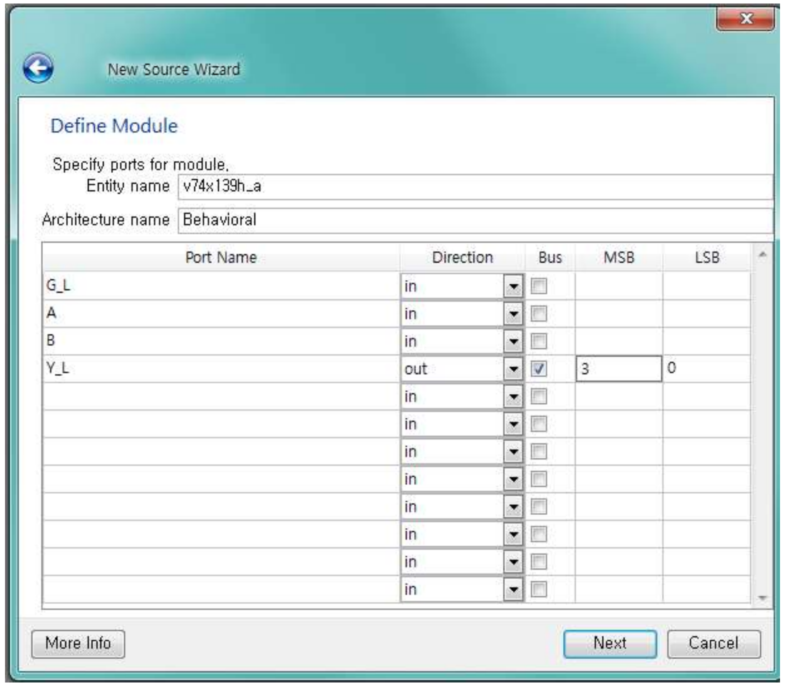
Specify ports for module.

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

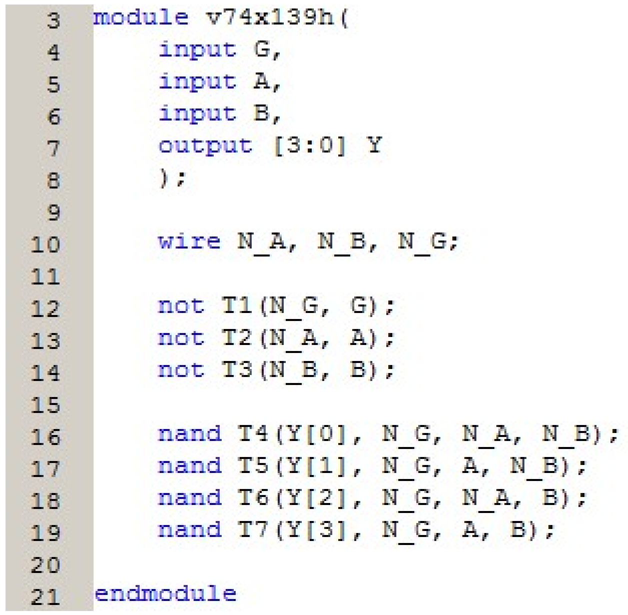
Specify ports for module.

Entity name: v74x139h_a

Architecture name: Behavioral

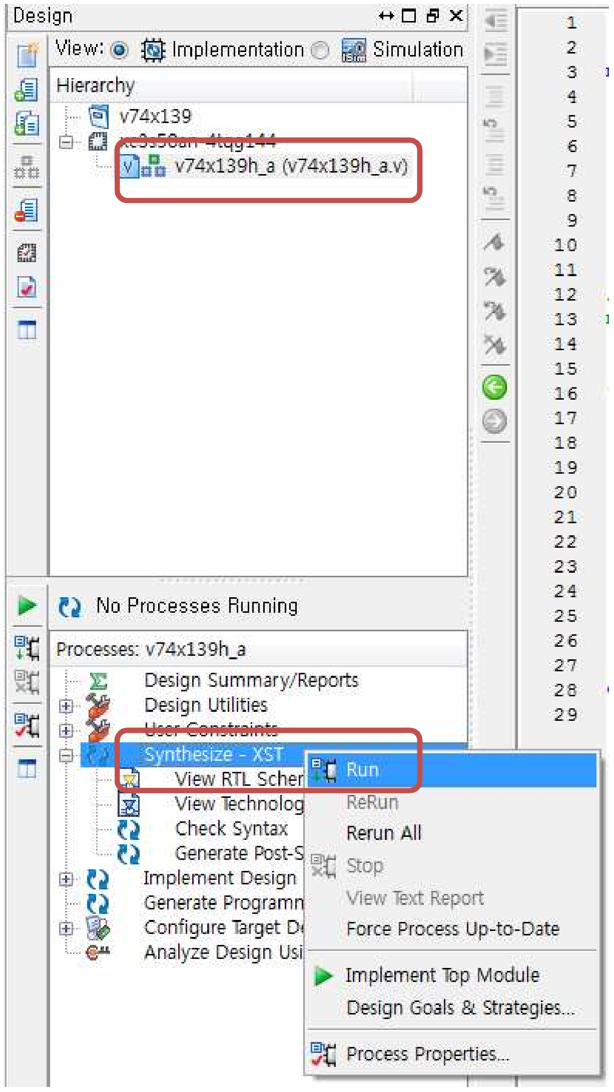
Port Name	Direction	Bus	MSB	LSB
G_L	in	☐		
A	in	☐		
B	in	☐		
Y_L	out	✓	3	0
	in	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		

Write your own Verilog codes

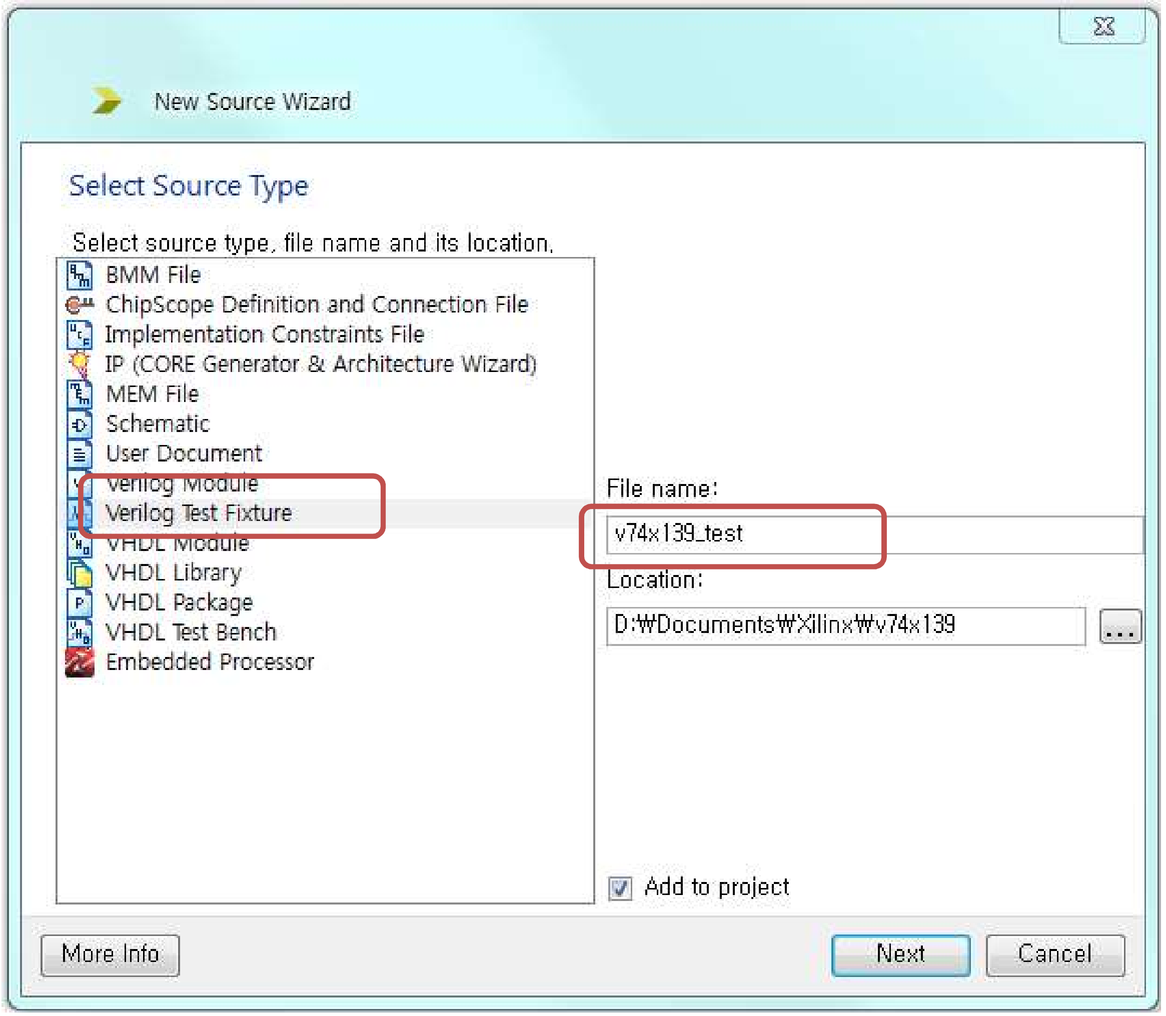


```
module v74x139h(
    input G,
    input A,
    input B,
    output [3:0] Y
    );

    wire N_A, N_B, N_G;

    not T1(N_G, G);
    not T2(N_A, A);
    not T3(N_B, B);

    nand T4(Y[0], N_G, N_A, N_B);
    nand T5(Y[1], N_G, A, N_B);
    nand T6(Y[2], N_G, N_A, B);
    nand T7(Y[3], N_G, A, B);

endmodule
```

Compile and check errors



Create a Verilog test bench


New Source Wizard

Select Source Type

Select source type, file name and its location.

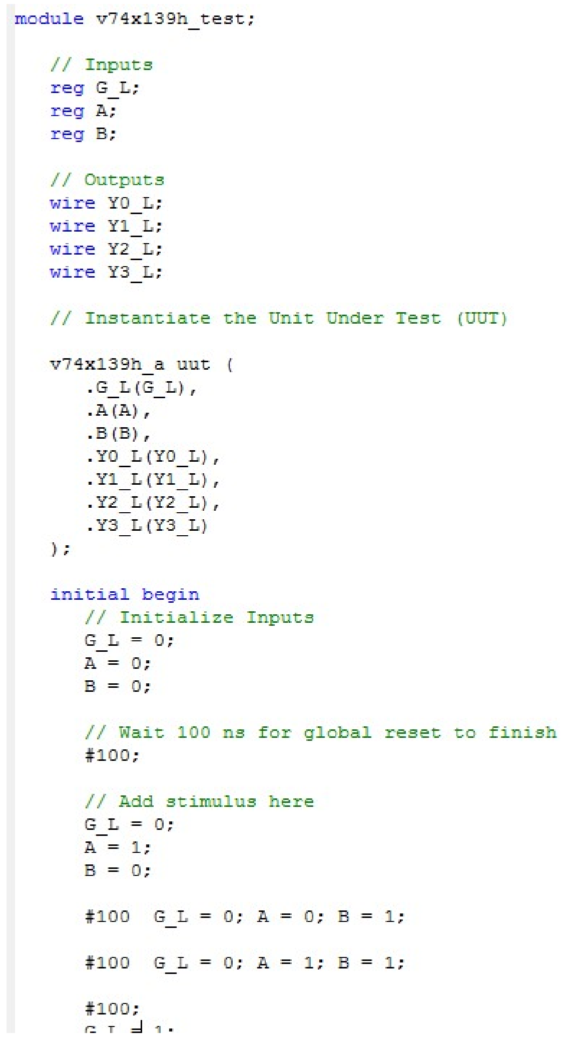
- BMM File
- ChipScope Definition and Connection File
- Implementation Constraints File
- IP (CORE Generator & Architecture Wizard)
- MEM File
- Schematic
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

File name: v74x139_test

Location: D:₩Documents₩Xilinx₩v74x139

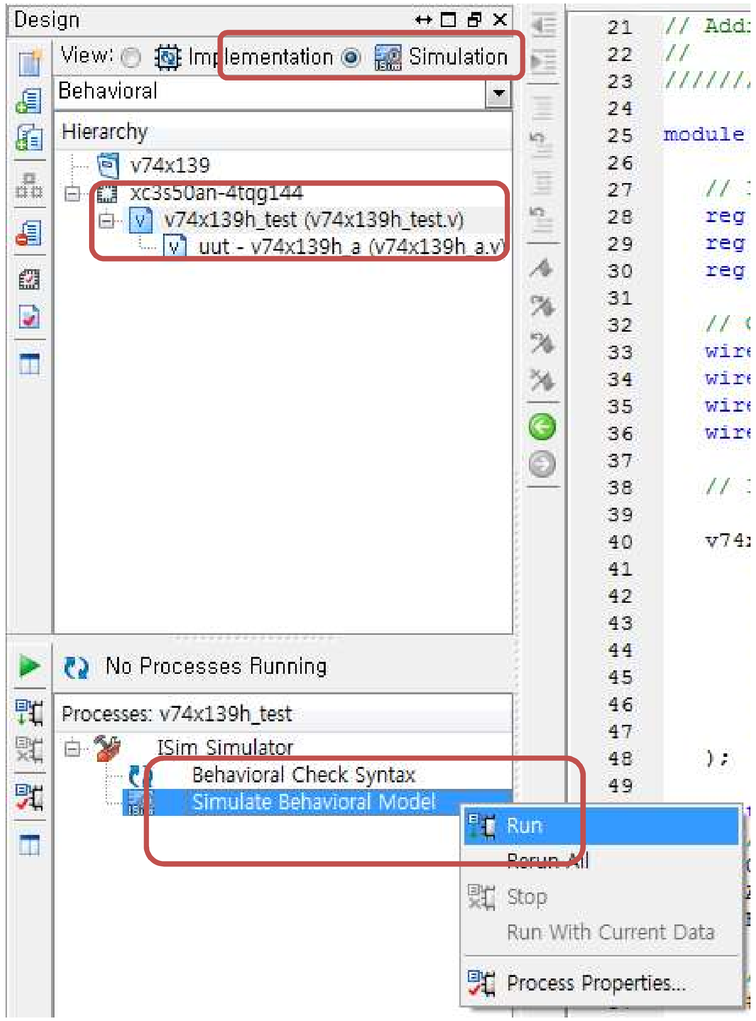
☑ Add to project

Write a Verilog test bench codes



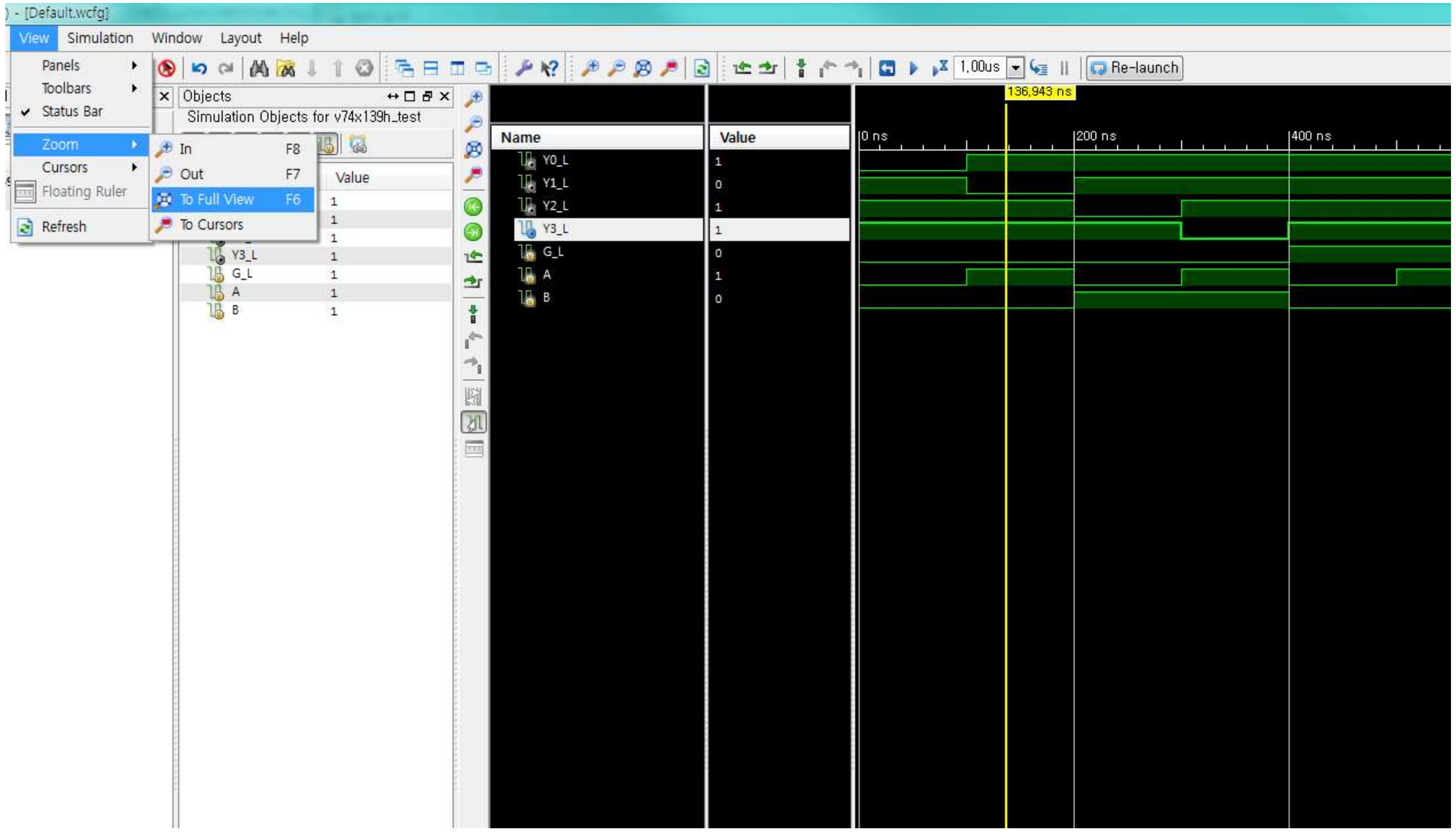
```
module v74x139h_test;

   // 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



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

Homework

Homework

(1) Lab Practice
- Implement 74x139 with 3 method and simulate it
- Implement a 3-to-8 decoder using 2-to-4 decoders only and simulate it (one of 3 designing methods)

(2) Implement 4-to-1 MUX with all the designing methods that we practiced and Simulate it.

(3) Discuss about each type of descriptions. (Pros & Cons, etc.)

(Optional) study part of the Verilog grammar and discuss about why it was designed that way
Ex) for loop, while loop, blocking, non-blocking assignments, module based feature, etc.

Report

- Write a report
  - Either in Korean or in English
  - Your report should include
    - discussion
    - Homework (if there is any)
  - # of pages doesn't matter
  - Documents should be submitted as **PDF file(less than 25Mb**)
  - **Attach source code and waveform screenshot**
  - Due :
    Class 001 - April, 11th (Before class begin at 7:00pm)
    Class 002 - April, 12th (Before class begin at 7:00pm)
    Class 003 - April, 13th (Before class begin at 7:00pm)