Introduction

Hsi-Pin Ma

http://lms.nthu.edu.tw/course/21094
Department of Electrical Engineering
National Tsing Hua University

Outline

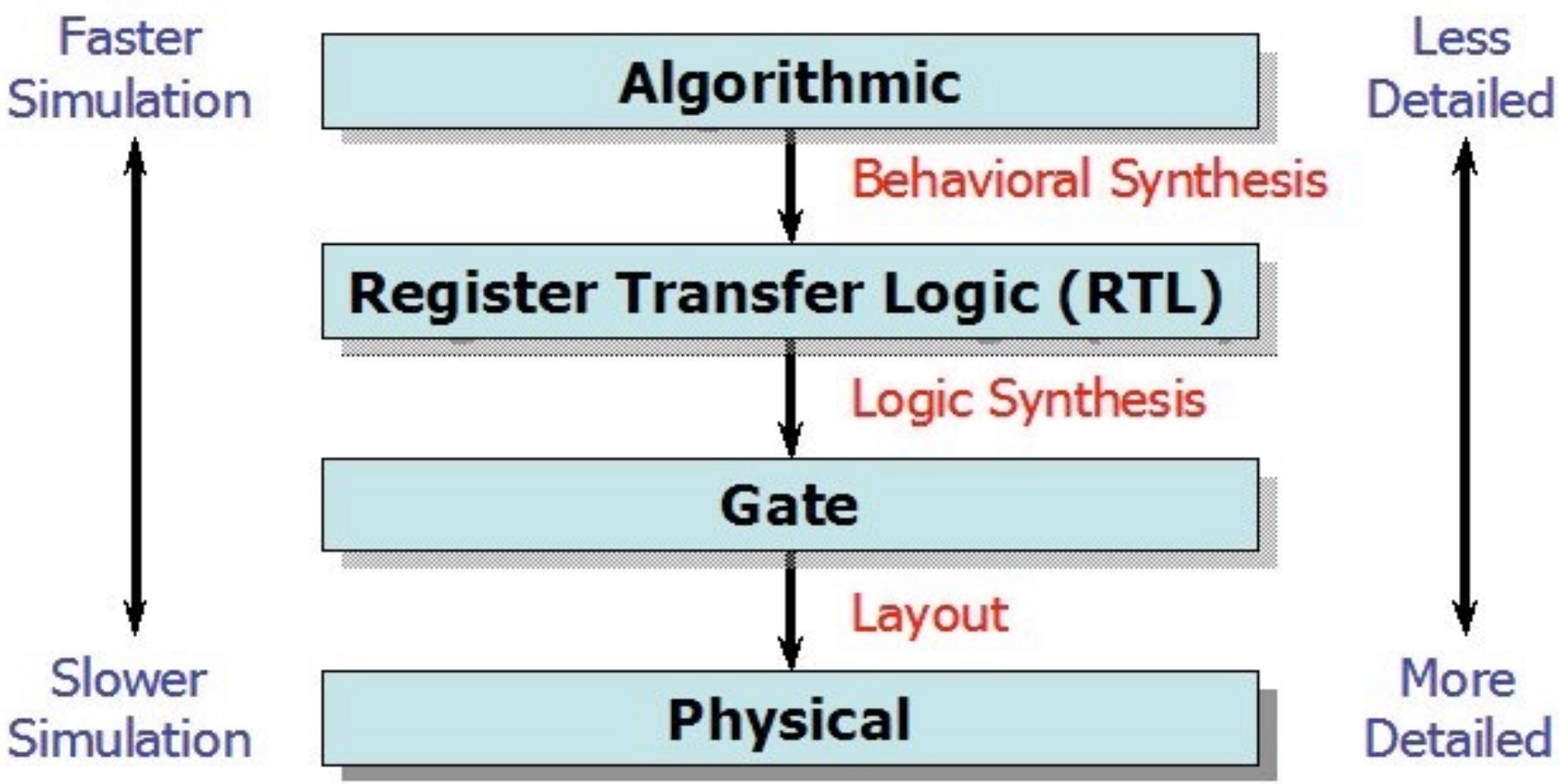
- Introduction
- Sample Design
- Structural Modeling
- RTL Modeling
- Logic Modeling and Simulation Using Xilinx ISE
- A Simple Example

Introduction

Hardware Description Language

- A high-level programming language offering special constructs to model microelectronic circuits
  - Describe the operation of a circuit at various level of abstraction
    - Behavior
    - Function
    - Structure
  - Describe the timing of a circuit
  - Express the concurrency of circuit operation

Levels of Abstraction (1/2)



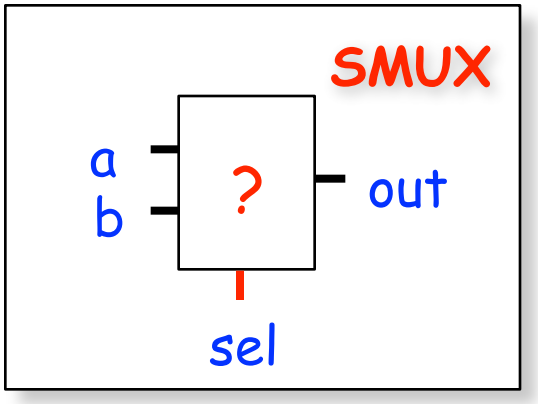
Levels of Abstraction (2/2)

- Behavioral Level (Architectural/Algorithmic Level)
  - Describes a system by the flow of data between its functional blocks
  - Defines signal values when they change
- Register Transfer Level (Dataflow Level)
  - Describe a system by the flow of data and control signals between and within its functional blocks
  - Defines signal values with respect to a clock
  - RTL (Register Transfer Level) is frequently used for the Verilog description with the combination of behavioral and dataflow constructs which is acceptable to logic synthesis tools.
- Gate Level (Structural)
  - A model that describes the gates and the interconnections between them
- Transistor/Switch/Physical Level
  - A model that describes the transistors and the interconnections between them

Behavior Level Abstraction

- Describe the design without implying any specific internal architecture
  - Use high level constructs (@, case, if, repeat, wait, while)
  - Usually use behavioral construct in testbench
  - Synthesis tools accept only a limited subset of these
    - Case 1: assign Z = (S) ? A : B;

```
module SMUX(out, a, b, sel);

output out;
input a,b,sel;
wire out;

assign out = (sel) ? a : b ;

endmodule
```

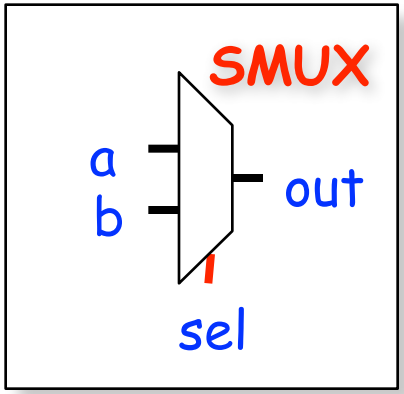


Behavior Level Abstraction

- Case 2:

```
always @(input1 or input2 or ...)
begin
  out1 =
end
```

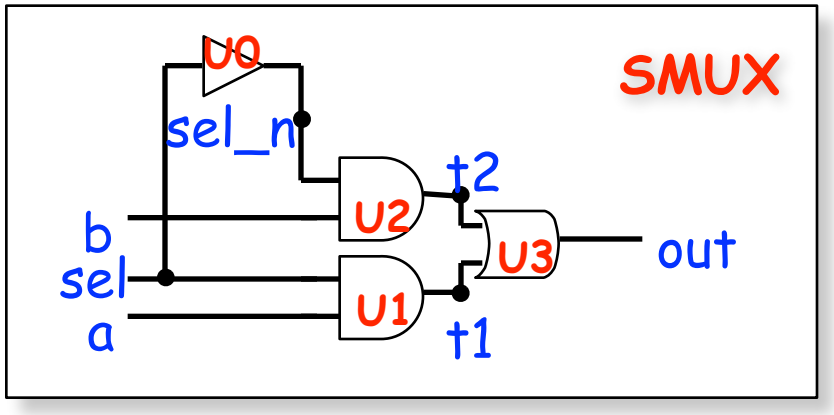
```
module SMUX(out, a, b, sel);

output out;
input a,b,sel;
reg out;

always @(a or b or sel)
  if (sel)
    out=a;
  else
    out=b;
endmodule
```



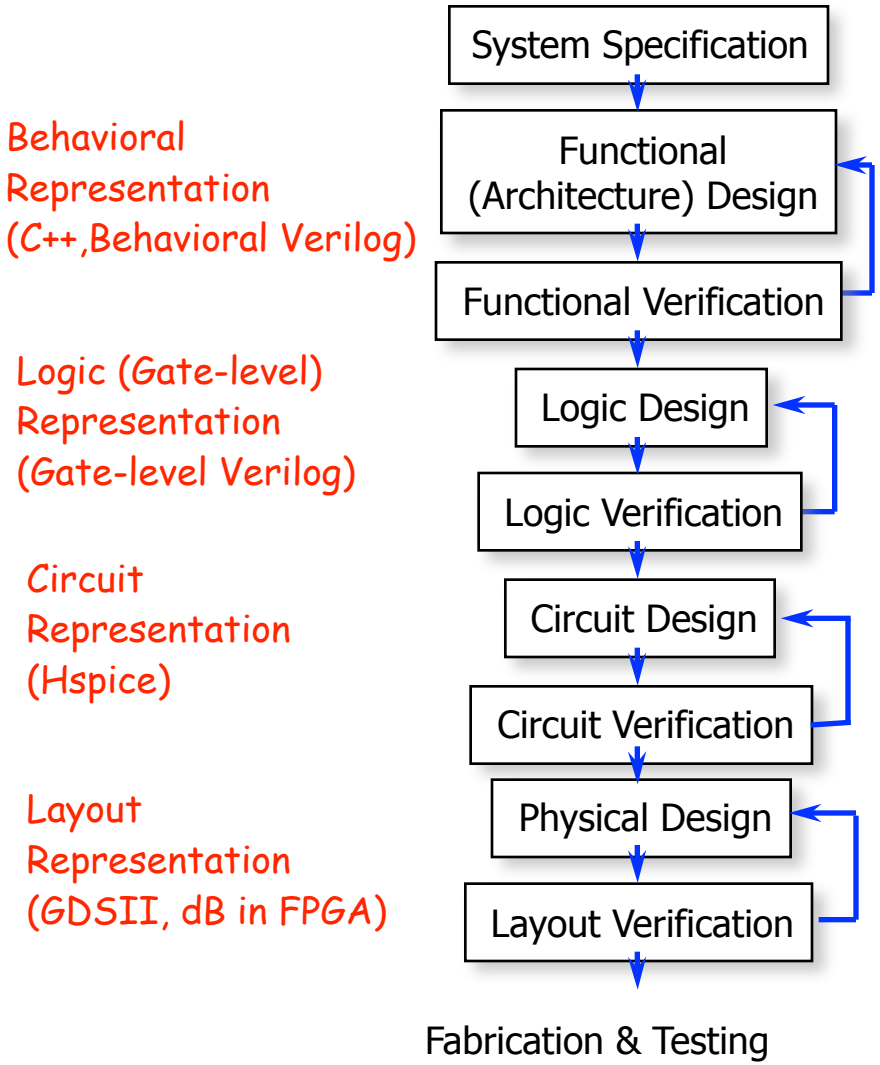
Gate Level Abstraction

- Synthesis tools produce a purely structural design description
  - You must derive and draw the circuit schematics first before writing Verilog codes

```
module SMUX(out, a, b, sel);

output out;
input a,b,sel;
wire sel_n,t1,t2;

not U0(sel_n,sel);
and U1(t1,a,sel);
and U2(t2,b,sel_n);
or  U3(out,t1,t2);

endmodule
```



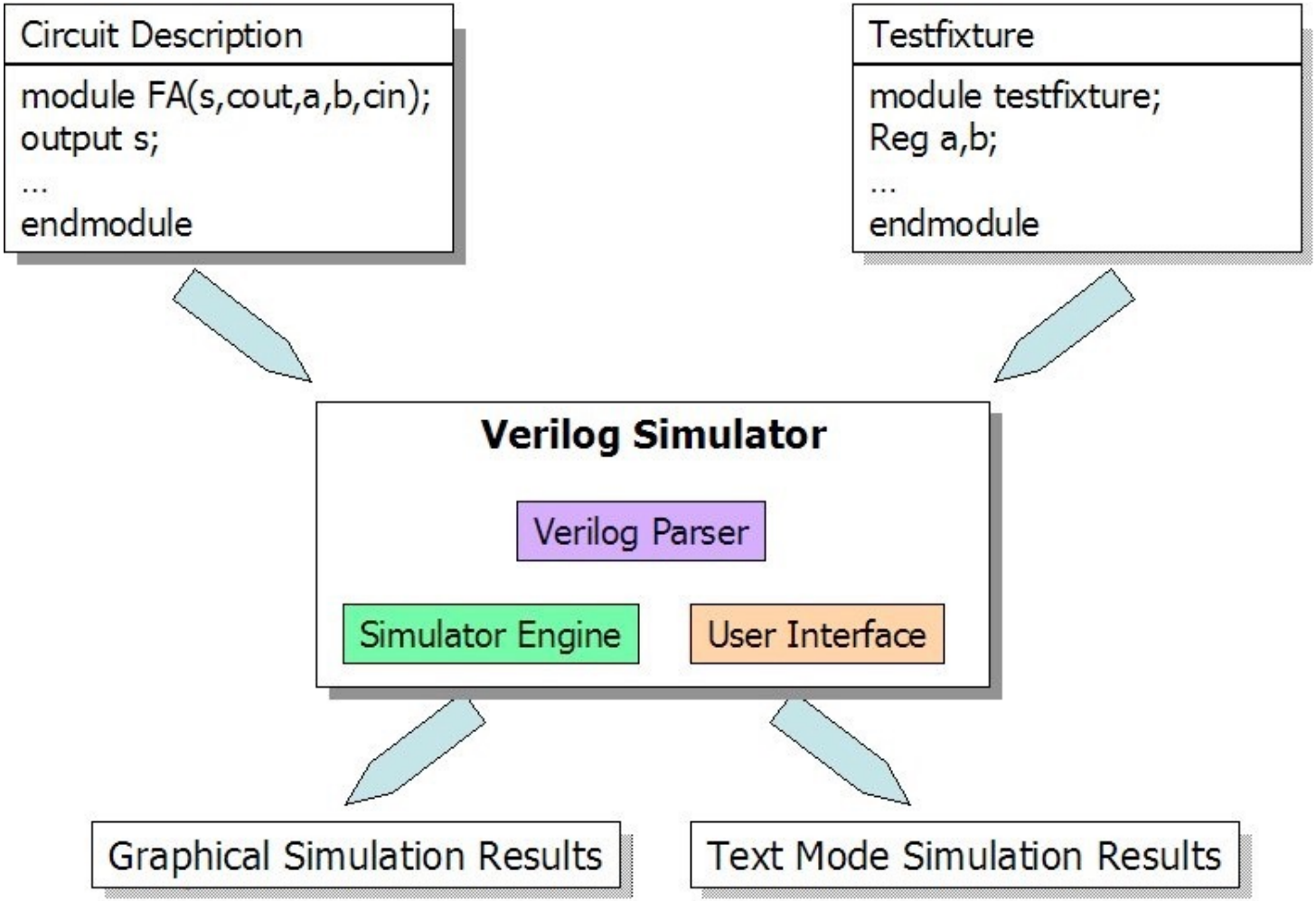
VLSI Design Flow



Event Simulation of a Verilog Model

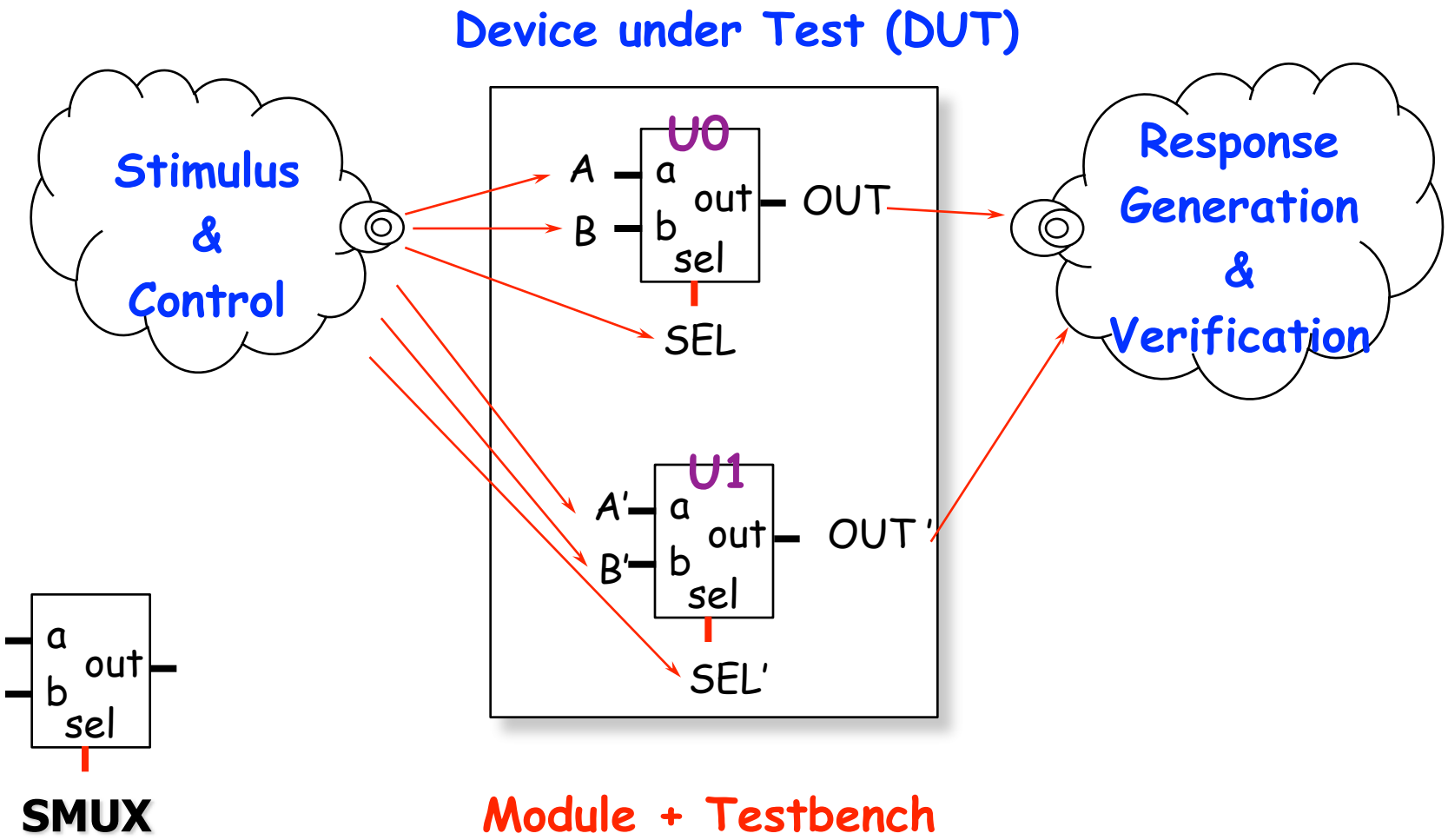
- Compilation
  - Compilation and elaboration
- Initialization
  - Initialize module parameters
    - Set other storage element to unknown (X) state
      - Unknown or un-initialized
    - Set undriven nets to the high-impedance (Z) state
      - Tri-state or floating
- Simulation

Verilog Simulation



Sample Design

Scenario


Device under Test (DUT)
Stimulus & Control
Response Generation & Verification
U0
A
B
a
b
out
sel
OUT
SEL
U1
A'
B'
a
b
out
sel
OUT '
SEL'
a
b
out
sel
SMUX
Module + Testbench

Verilog Module

module module_name(port_names);

- Port declaration
- Data type declaration
- Task & function declaration
- Module functionality or structure
- Timing Specification

endmodule

```
module SMUX(out, a, b, sel);

output out;
input a,b,sel;
wire sel_n,t1,t2;

not U0(sel_n,sel);
and U1(t1,a,sel);
and U2(t2,b,sel_n);
or  U3(out,t1,t2);

endmodule
```

Testbench (1/4)

module testfixture;

- Declare signals
- Instantiate modules
- Applying stimulus
- Monitor signals

endmodule

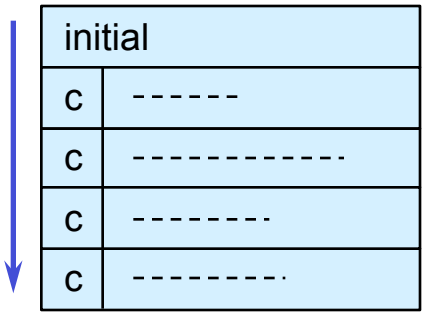
Compare this to a breadboard experiment!

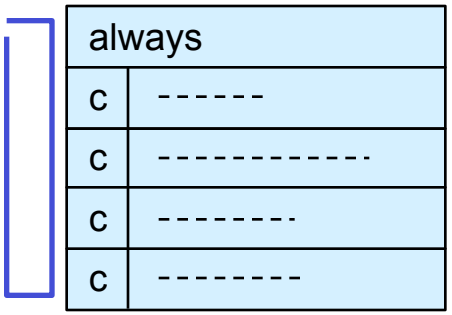
Testbench (2/4)

- Declare signals
  - Test pattern must be stored in storage elements first and then apply to DUT (Device under Test)
    - Use "reg" to declare the storage element
- Instantiate modules
  - Both behavioral level or gate level model can be used.

Testbench (3/4)

- Describing Stimulus
  - The testbench always be described behaviorally.
  - Procedural blocks are bases of behavioral modeling.
  - The simulator starts executing all procedure blocks at time 0 and executes them concurrently.
  - Two types of procedural blocks
    - initial
    - always





Testbench (4/4)

```
module test_SMUX;

reg  A,B,SEL;
wire        OUT;

SMUX U0( .out(OUT), .a(A), .b(B), .sel(SEL));

initial
begin
    A=0;B=0;SEL=0;
    #10     A=0;B=1;SEL=1;
    #10     A=1;B=0;
    #10     SEL=0;
    …..
    #10     SEL=1;
end

endmodule
```

Declare signals

Make an instance

Assign values to storage elements

#10 to specify 10 time unit delay

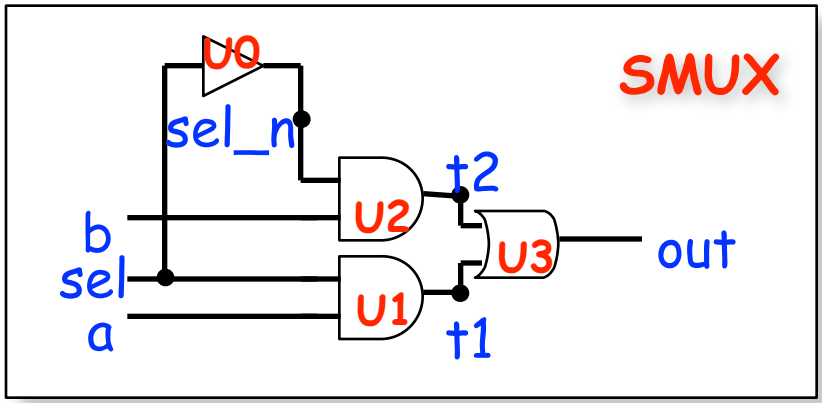
Structural Modeling

Verilog Primitives

- and : Logical AND
- or : Logical OR
- not : Inverter
- buf : Buffer
- xor : Logical exclusive OR
- nand : Logical AND inverted
- nor : Logical OR inverted
- xnor : Logical exclusive OR inverted

Structural Modeling

```
module SMUX(out, a, b, sel);

output out;
input a,b,sel;
wire sel_n,t1,t2;

not U0(sel_n,sel);
and U1(t1,a,sel);
and U2(t2,b,sel_n);
or  U3(out,t1,t2);

endmodule
```



RTL Modeling

Operators (1/3)

Bitwise Operators		
OP	**Usage**	**Description**
~	~m	Invert each bit of m
&	m & n	AND each bit of m with each bit of n
|	m | n	OR each bit of m with each bit of n
^	m ^ n	Exclusive OR each bit of m with n
~^ or ^~	m ~^ n or m ^~ n	Exclusive NOR each bit of m with n

Unary Reduction Operators		
OP	**Usage**	**Description**
&	&m	AND all bits in m together (1-bit result)
~&	~&m	NAND all bits in m together (1-bit result)
|	|m	OR all bits in m together (1-bit result)
~|	~|m	NOR all bits in m together (1-bit result)
^	^m	Exclusive OR all bits in m (1-bit result)
~^ or ^~	~^m or ^~m	Exclusive NOR all bits in m (1-bit result)

Operators (2/3)

Arithmetic Operators		
OP	**Usage**	**Description**
+	m + n	Add n to m
-	m - n	Subtract n from m
-	-m	Negate m (2's complement)
*	m * n	Multiply m by n
/	m / n	Divide m by n
%	m % n	Modulus of m / n

The divisor for divide operator may be restricted to constants and a power of 2

Synthesis not supported

Logical Operators		
OP	**Usage**	**Description**
!	!m	Is m not true? (1-bit True/False result)
&&	m && n	Are both m and n true? (1-bit True/False result)
||	m || n	Are either m or n true? (1-bit True/False result)

Equality Operators (compares logic values of 0 and 1)		
OP	**Usage**	**Description**
==	m == n	Is m equal to n? (1-bit True/False result)
!=	m != n	Is m not equal to n? (1-bit True/False result)

Identity Operators (compares logic values of 0, 1, x, and z)		
OP	**Usage**	**Description**
===	m === n	Is m identical to n? (1-bit True/False result)
!==	m !== n	Is m not identical to n? (1-bit True/False result)

Synthesis not supported

Synthesis not supported

Operators (3/3)

Relational Operators		
OP	**Usage**	**Description**
<	m < n	Is m less than n? (1-bit True/False result)
>	m > n	Is m greater than n? (1-bit True/False result)
<=	m <= n	Is m less than or equal to n? (True/False result)
>=	m >= n	Is m greater than or equal to n? (True/False result)

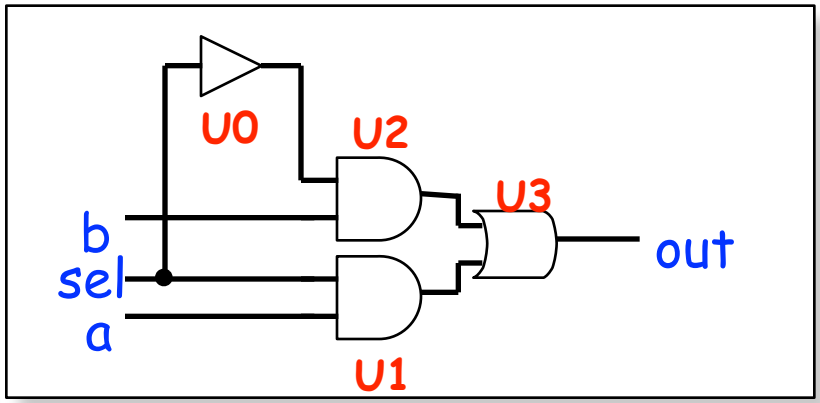
Logical Shift Operators		
OP	**Usage**	**Description**
<<	m << n	Shift m left n-times
>>	m >> n	Shift m right n-times

Misc Operators		
OP	**Usage**	**Description**
? :	sel?m:n	If sel is true, select m: else select n
{}	{m,n}	Concatenate m to n, creating larger vector
{{}}	{n{m}}	Replicate m n-times

assign

- **assign** continuous construct
  - combinational logics

```
module SMUX (out,a,b,sel);
output out;
input a,b,sel;

  assign out = (a&sel) | (b&(~sel));

endmodule
```



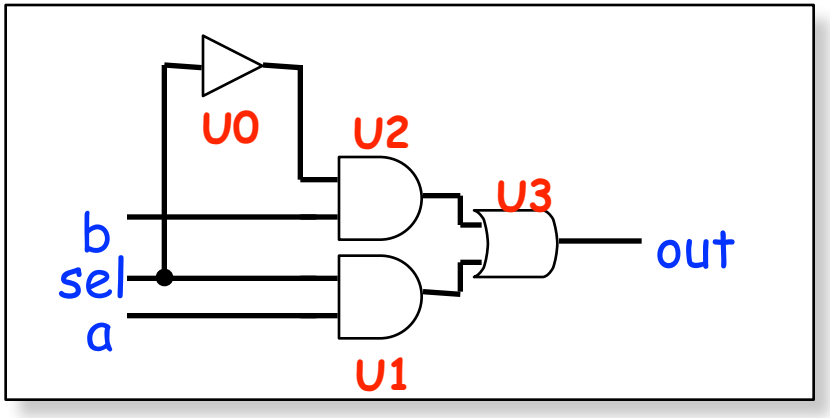
This out has to be declared as “wire” or or “output” data type.
This expression can not be inside always @().

always

- **always** statements

Must have sensitivity list

```
module SMUX (out,s,b,sel);
output out;
input a,b,sel;
reg out;

always @(a or b or sel)
    out = (a&sel) | (b&(~sel));

endmodule
```

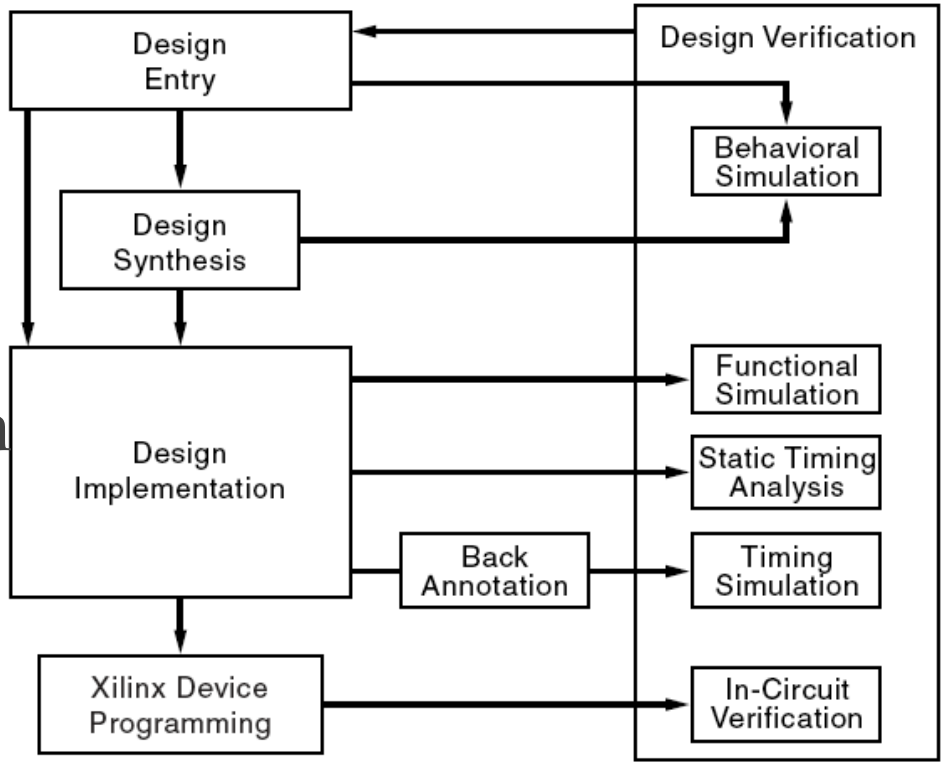


This out has to be declared as “reg” data type.

Logic Modeling and Simulation Using Xilinx ISE

Design Flow

- General design flow
  - Design construction
  - Behavioral simulation
  - Design implementation
  - Timing simulation
- HDL-based design Flow

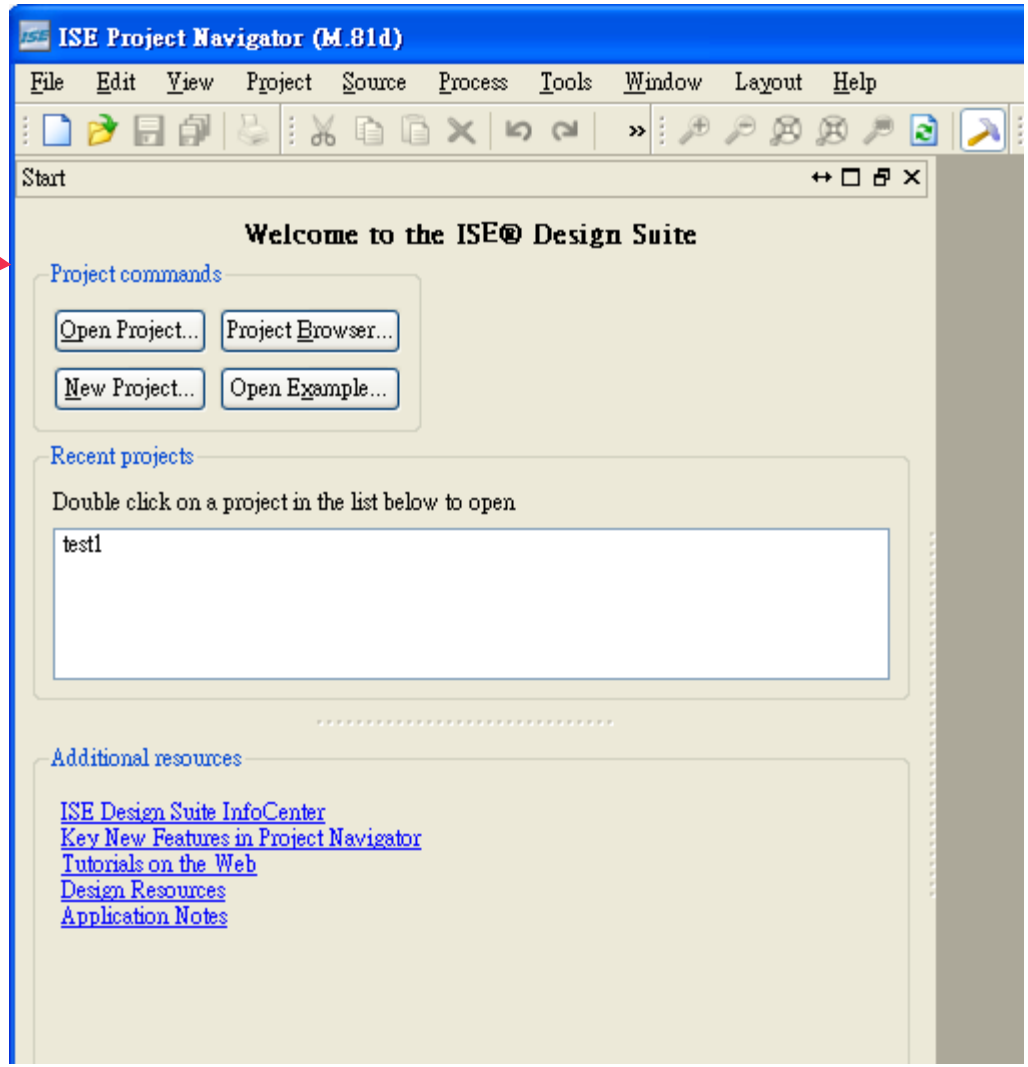


Important Notes

- **Draw schematic first** and then construct Verilog codes.
- Verilog RTL coding philosophy is not the same as C programming
  - Every Verilog RTL construct has its own logic mapping (for synthesis)
  - You should have the logics (draw schematic) first and then the RTL codes
  - You have to write **synthesizable** RTL codes

Open ISE


Xilinx PlanAhe...
Xilinx ISE Design Su...


ISE Project Navigator (M.81d)
File Edit View Project Source Process Tools Window Layout Help
Start
Welcome to the ISE® Design Suite
Project commands
Open Project...
Project Browser...
New Project...
Open Example...
Recent projects
Double click on a project in the list below to open
test1
Additional resources
ISE Design Suite InfoCenter
Key New Features in Project Navigator
Tutorials on the Web
Design Resources
Application Notes

Open New Project (1/4)

ISE Project Navigator (M.81d)

File Edit View Project Source Process Tools Window Layout Help

Start

Welcome to the ISE® Design Suite

Project commands

Open Project... | Project Browser...

New Project... | Open Example...

Recent projects

Double click on a project in the list below to open

test1

Additional resources

ISE Design Suite InfoCenter
Key New Features in Project Navigator
Tutorials on the Web
Design Resources
Application Notes

Console

New Project Wizard

Create New Project
Specify project location and type.

Enter a name, locations, and comment for the project

Name: proj1

Location: D:\DATA\test\proj1

Working Directory: D:\DATA\test\proj1

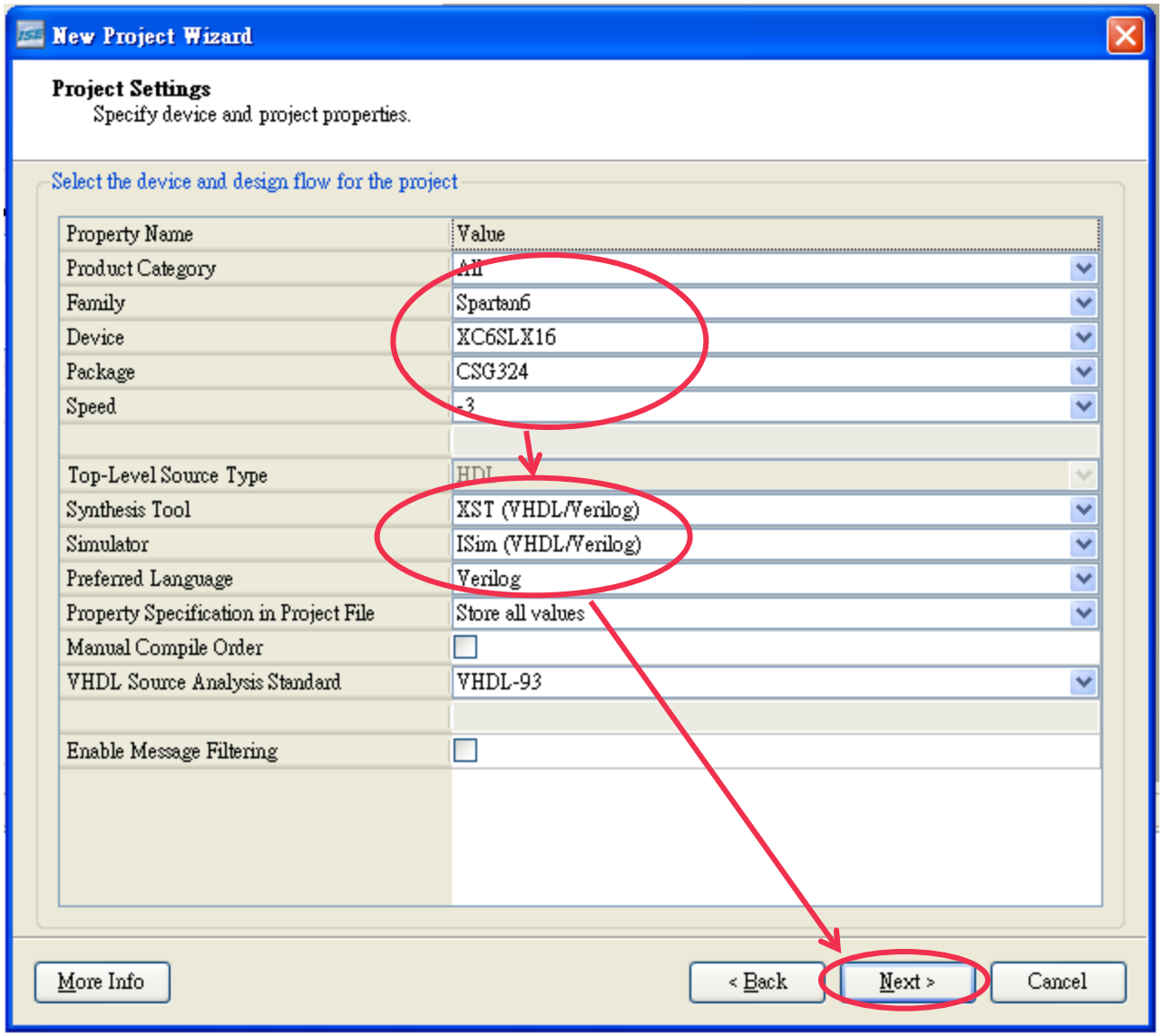
Description: project1 is logic design lab 0

Select the type of top-level source for the project

Top level source type: HDL

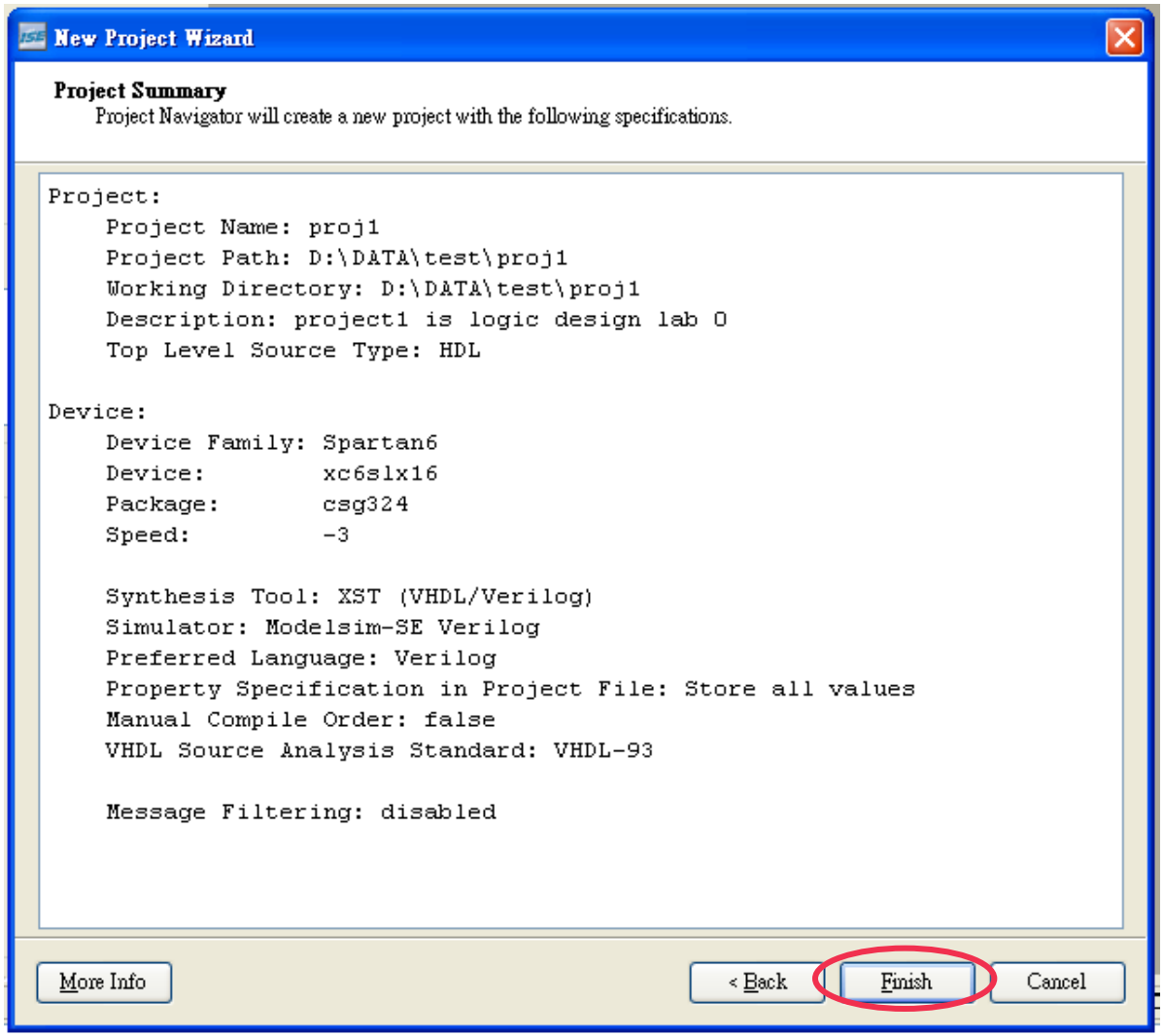
More Info | Next > | Cancel

Open New Project (2/4)



注意: 1. Family 有另外一個Automotive Spartan6，別選到這個錯誤的
2. Simulator 使用ISim

Open New Project (3/4)



Open New Project (4/4)

ISE Project Navigator (M.81d) - D:\DATA\test\proj1\proj1.xise

File Edit View Project Source Process Tools Window Layout Help

Design

View: Implementation Simulation

Hierarchy

proj1

xc6slx16-3csg324

Unassigned User Library Modules

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: To create a new source file.
- Add Source: To add an existing file to the project.
- Add Copy of Source: To copy an existing file to the project directory and add it to the project.

No Processes Running

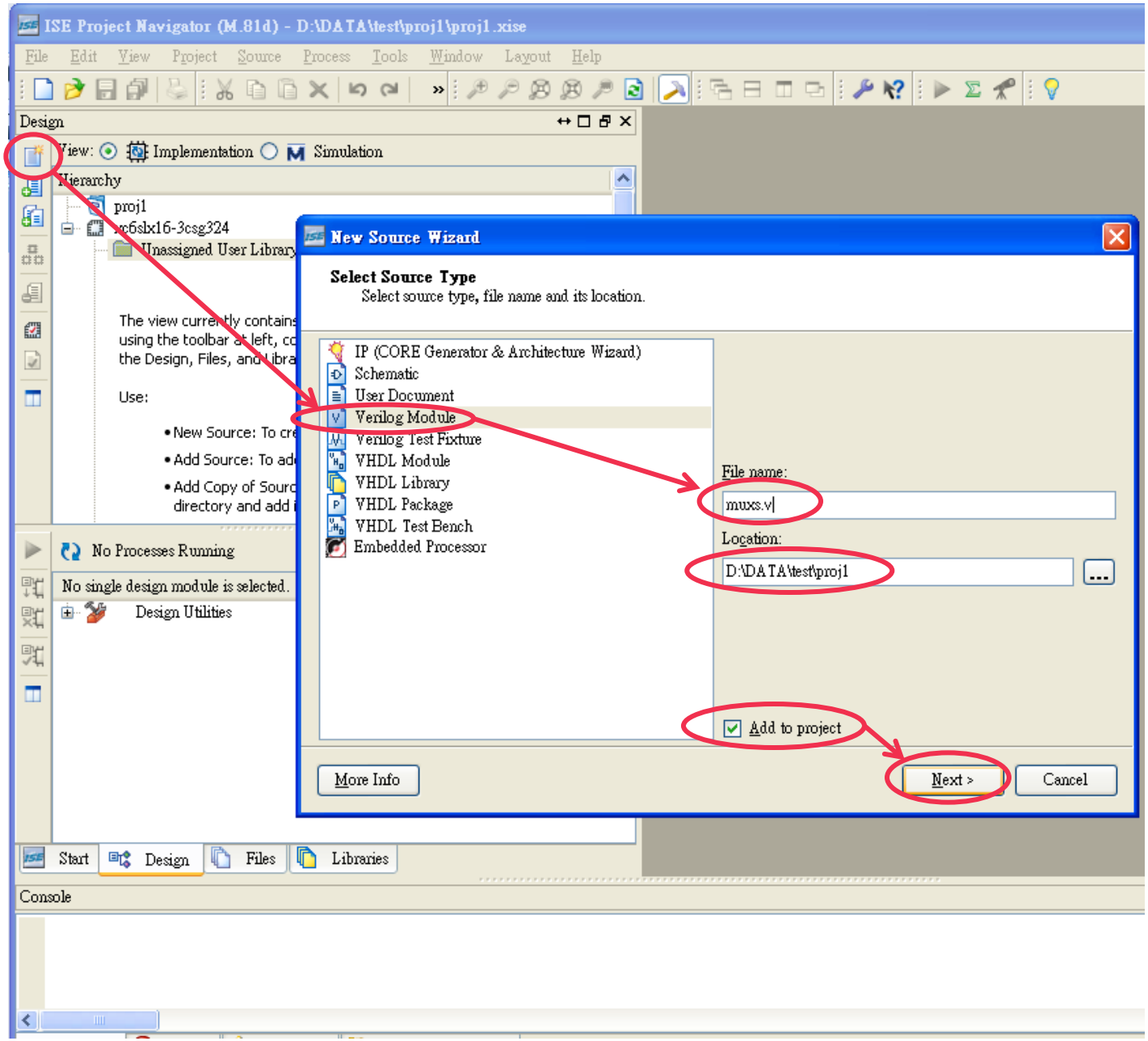
No single design module is selected.

Design Utilities

Start Design Files Libraries

Console

New Source (1/6)



New Source (2/6)

ISE Project Navigator (M.81d) - D:\DATA\test\proj1\proj1.xise - [muxs.v*]

File Edit View Project Source Process Tools Window Layout Help

New Source Wizard

Define Module
Specify ports for module.

Module name: muxs

Port Name	Direction	Bus	MSB	LSB
a	input			
b	input			
sel	input			
z	output			
	input			
	input			
	input			
	input			
	input			
	input			
	input			
	input			

More Info | < Back | Next > | Cancel

```
1  `timescale 1ns / 1ps
2  ////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date:    13:40:34 02/17/2012
7  // Design Name:
8  // Module Name:    muxs
9  // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 ////////////////////////////////////////////////////////////////////////////////
21 module muxs(
22     input a,
23     input b,
24     input sel,
25     output z
26     );
27
28
29
30
31
32 endmodule
```

assign z = sel ? a : b;

Start | Design | Files | Libraries | Design Summary (out of date) | muxs.v*

Console

```
INFO:HDLCompiler:1677 - Analyzing Verilog file \"D:/DATA/test/proj1/muxs.v\" into library work
INFO:ProjectMgmt:656 - Parsing design hierarchy completed successfully.
```

New Source (3/6)

ISE Project Navigator (M.81d) - D:\DATA\test\proj1\proj1.xise - [muxs.v*]

File Edit View Project Source Process Tools Window Layout Help

Design

View: Implementation Simulation

Hierarchy

proj1

New Source Wizard

Define Module

Specify ports for module.

Module name muxs

Port Name	Direction	Bus	MSB	LSB
	input			
	input			
	input			
	input			
	input			
	input			
	input			
	input			
	input			
	input			
	input			
	input			

More Info < Back Next > Cancel

```
1  `timescale 1ns / 1ps
2  ////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date:    13:31:00 02/17/2012
7  // Design Name:
8  // Module Name:    muxs
9  // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 ////////////////////////////////////////////////////////////////////////////////
21 module muxs(
22     );
23
24
25
26
27
28
29
30
31
32 endmodule
```

input a, b, sel;
output z;

assign z = sel ? a : b;

Start Design Files Libraries

Design Summary (out of date) muxs.v*

Console

New Source (4/6)

ISE Project Navigator (M.81d) - D:\DATA\test\proj1\proj1.xise - [muxs.v*]

File Edit View Project Source Process Tools Window Layout Help

Design

View: Implementation Simulation

Behavioral

Hierarchy

proj1

xc6slx16-3csg324

muxs (muxs.v)

```
1 `timescale 1ns / 1ps
2 ////////////////////////////////////////////////////////////////////////////////
3 // Company:
4 // Engineer:
5 //
```

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: proj1_test.v

Location: D:\DATA\test\proj1

Add to project

More Info | Next > | Cancel

No Processes Running

Processes: muxs

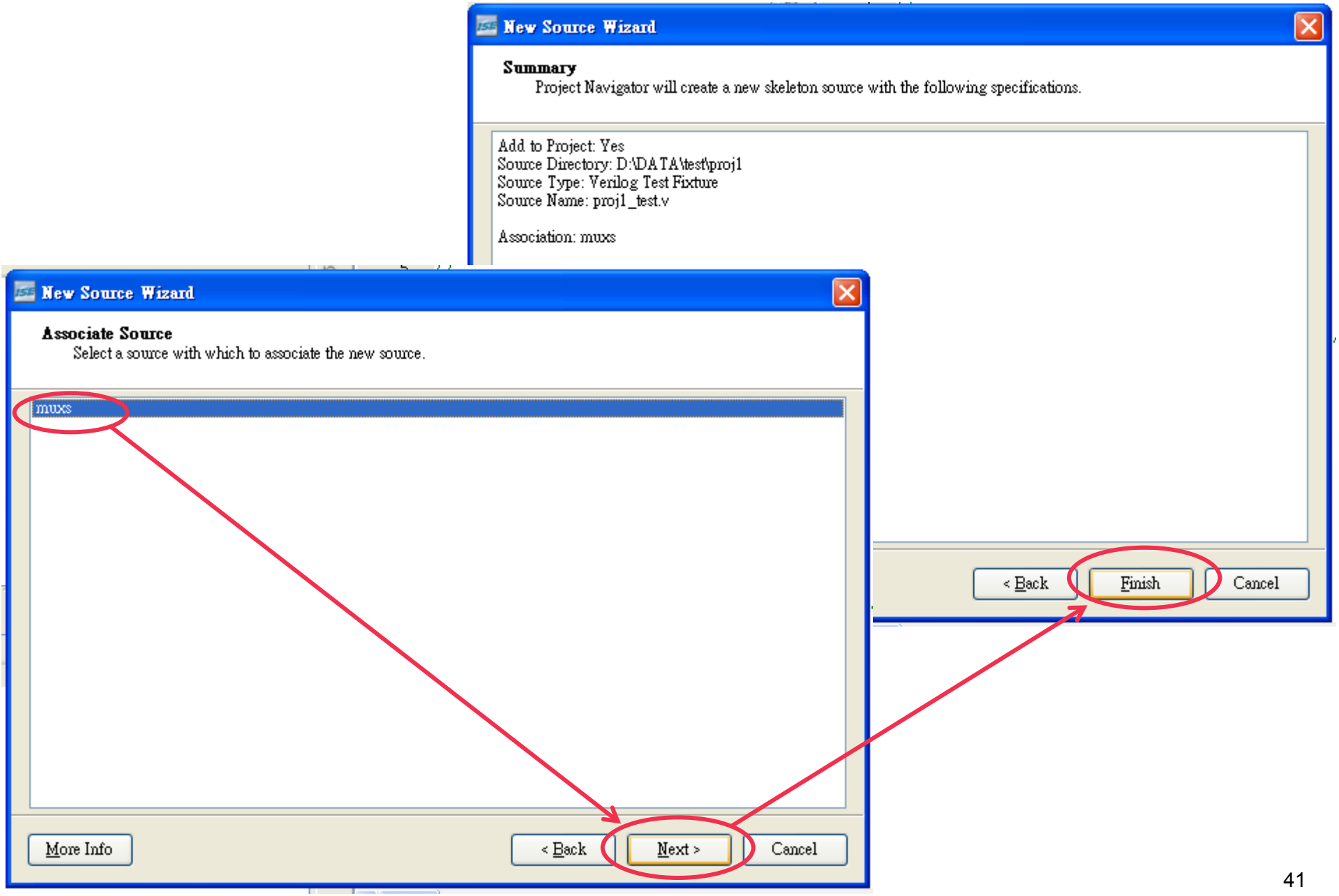
ModelSim Simulator

Start | Design | Files | Libraries | Design Summary (out of date) | muxs.v*

Console

```
INFO:ProjectMgmt:656 - Parsing design hierarchy completed successfully.
INFO:HDLCompiler:1677 - Analyzing Verilog file \"D:/DATA/test/proj1/muxs.v\" into library work
INFO:ProjectMgmt:656 - Parsing design hierarchy completed successfully.
```

New Source (5/6)



New Source (6/6)

save

ISE Project Navigator (M.81d) - D:\DATA\test\proj1\proj1.xise - [proj1_test.v]

File Edit View Project Source Process Tools Window Layout Help

Design

View: Implementation Simulation

Behavioral

Hierarchy

proj1

xc6slx16-3csg324

proj1_test (proj1_test.v)

uut - muxs (muxs.v)

No Processes Running

Processes: proj1_test

ISim Simulator

Behavioral Check Syntax

Simulate Behavioral Model

Run

Rerun All

Stop

Run With Current Data

Process Properties...

Right click

Left click to edit simulation duration

```
27    // Inputs
28    reg a;
29    reg b;
30    reg sel;
31
32    // Outputs
33    wire z;
34
35    // Instantiate the Unit Under Test (UUT)
36    muxs uut (
37       .a(a),
38       .b(b),
39       .sel(sel),
40       .z(z)
41    );
42
43    initial begin
44       // Initialize Inputs
45       a = 0;
46       b = 0;
47       sel = 0;
48
49       // Wait 100 ns for global reset to finish
50       #100;
51
52       #100 a=1;                  //  a=1 b=0 sel=0
53       #100 a=0; b=1;             //  a=0 b=1 sel=0
54       #100 a=1;                  //  a=1 b=1 sel=0
55       #100 a=0; b=0; sel=1;      //  a=0 b=0 sel=1
56       #100 a=1;                  //  a=1 b=0 sel=1
57       #100 a=0; b=1;             //  a=0 b=1 sel=1
58       #100 a=1;                  //  a=1 b=1 sel=1
59
60       // Add stimulus here
61
62
63    end
```

Edit your simulation pattern

Start Design Files Libraries

Design Summary proj1_test.v

Console

Simulation (1/2)

ISE Project Navigator (M.81d) - D:\DATA\test\proj1\proj1.xise - [proj1_test.v]

File Edit View Project Source Process Tools Window Layout Help

Design

View: Implementation Simulation

Behavioral

Hierarchy

- proj1
- xc6slx16-3csg324
  - proj1_test (proj1_test.v)
    - uut - muxs (muxs.v)

No Processes Running

Processes: proj1_test

- ISim Simulator
  - Behavioral Check Syntax
  - Simulate Behavioral Model

```
27    // Inputs
58    #100 a=1;          //  a=1 b=1 sel=1
59
```

Process Properties - ISim Properties

Switch Name	Property Name	Value
	Use Custom Simulation Command File	☐
	Custom Simulation Command File	
	Run for Specified Time	☑
	Simulation Run Time	1000 ns
	Waveform Database Filename	D:/DATA/test/proj1/proj1_test_isim_beh.wdb
	Use Custom Waveform Configuration File	☐
	Custom Waveform Configuration File	
	Specify Top Level Instance Names	work.proj1_test

Property display level: Standard ☑ Display switch names

OK Cancel Apply Help

Write simulation duration

Double click to obtain simulation results

Simulation (2/2)

You can use Zoom In/Zoon Out/Scroller to adjust waveform display

Zoon In/Zoon Out

ISim (M.81d) - [Default.wcfg]

File Edit View Simulation Window Layout Help

Source Files: glbl.v, muxs.v, proj1_test.v

Objects — Simulation Objects for proj1_test

Object Name	Value
z	1
a	1
b	1
sel	1

Name	Value
z	1
a	0
b	1
sel	0

0 ns, 500 ns, 1,000 ns, 1,500 ns; X1: 305.833 ns

Scroler Bar

Console

```
INFO: There is another simulation running in the same directory. Using database file name isim2.wdb.
ISim M.81d (signature 0xcb73ee62)
WARNING: A WEBPACK license was found.
WARNING: Please use Xilinx License Configuration Manager to check out a full ISim license.
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
```

A Combinational Logic Example

Design Procedure

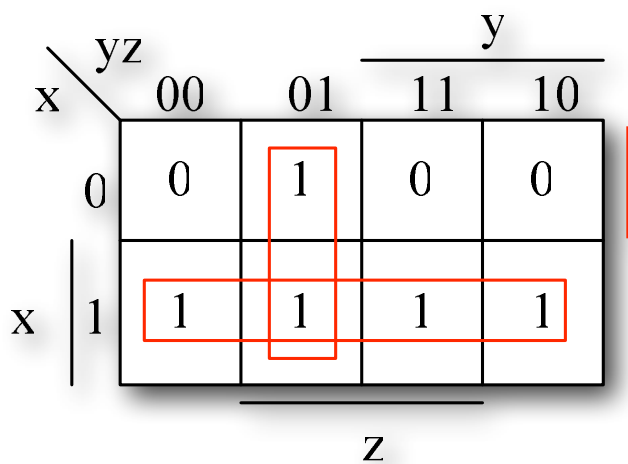
1. From the ***specifications***, determine the inputs, outputs, and their symbols.
2. Derive the ***truth table*** (***functions***) from the relationship between the inputs and outputs
3. Derive the ***simplified Boolean functions*** for each output function.
4. Draw the logic diagram.
5. Construct the Verilog code according to the logic diagram.
6. Write the testbench and verify the design.

$$F(x,y,z) = \sum(1,4,5,6,7) = f$$

1 input: x,y,z output: f

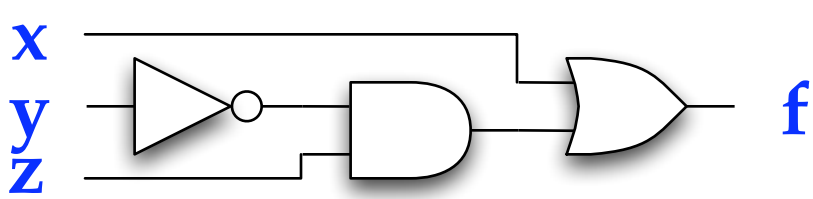
2

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



3

f=F(x,y,z)=x+y'z

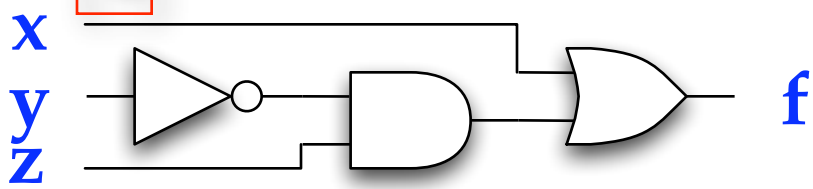


4

$$F(x,y,z) = \sum(1,4,5,6,7) = f$$

f=F(x,y,z)=x+y'z

4



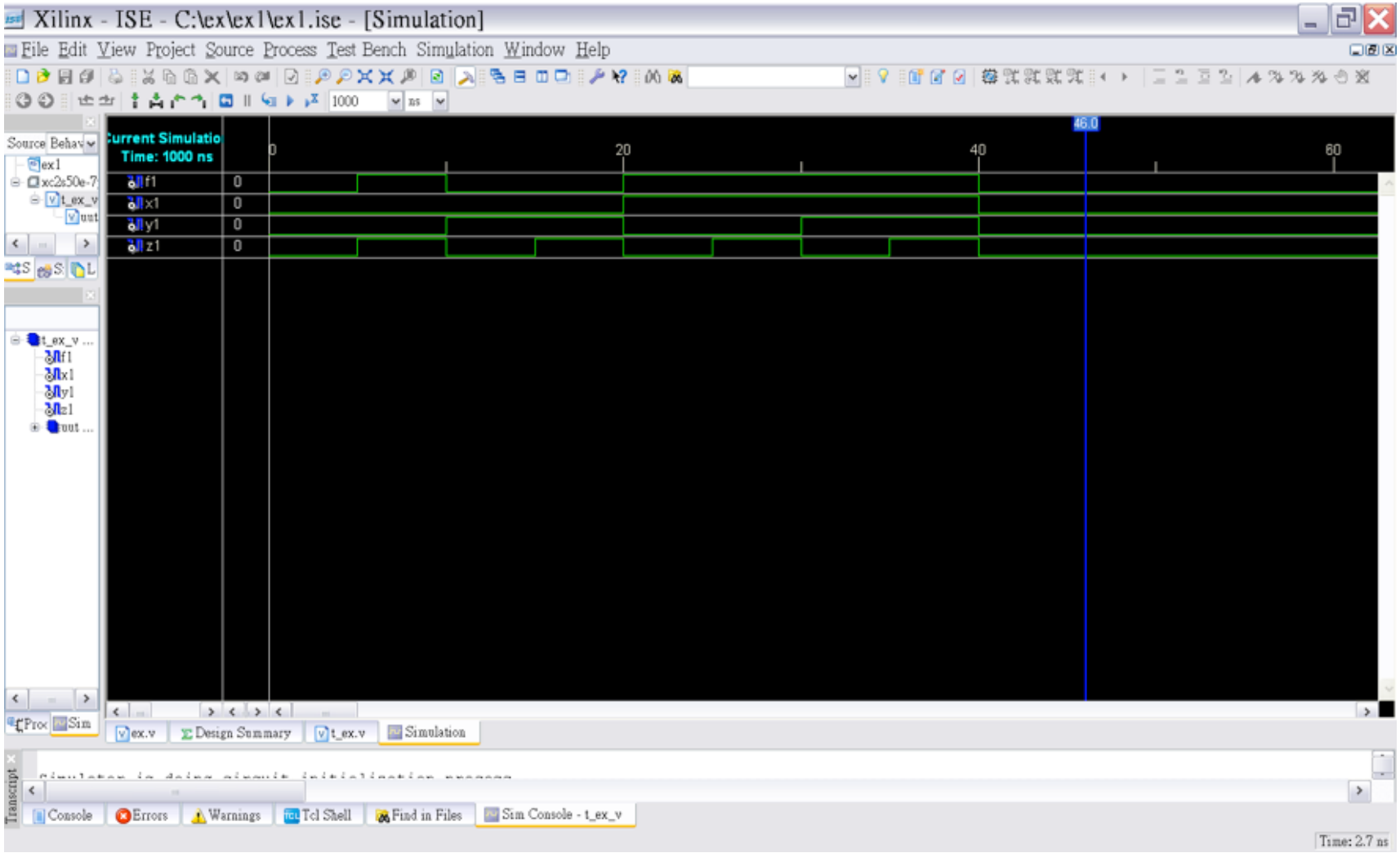
5

```
module ex(f,x,y,z);
output f;
input x,y,z;

  assign f = x | ((~y)&z);

endmodule
```

6

```
module t_ex;
wire f1;
reg x1,y1,z1;

ex U0(.f(f1),.x(x1),.y(y1),.z(z1));

initial
begin
  x1=0;y1=0;z1=0;
  #5 x1=0;y1=0;z1=1;
  #5 x1=0;y1=1;z1=0;
  #5 x1=0;y1=1;z1=1;
  #5 x1=1;y1=0;z1=0;
  #5 x1=1;y1=0;z1=1;
  #5 x1=1;y1=1;z1=0;
  #5 x1=1;y1=1;z1=1;
  #5 x1=0;y1=0;z1=0;
end

endmodule
```

$$F(x, y, z) = \sum(1, 4, 5, 6, 7) = f$$



Decimal Adders (1/3)

- Addition of 2 decimal digits in BCD
  - $\{C_{out},S\}=A+B+C_{in}$
    - 1 $S=S_8S_4S_2S_1$, $A=A_8A_4A_2A_1$, $B=B_8B_4B_2B_1$
  - A digit in BCD cannot exceed 9, add 6 (0110) for final correction.

10		10000
8_{10}	A	1000_2
9_{10}	B	1001_2
17_{10}	KZ	10001_2
		0110_2
		00010111_2

2 3

binary coded results

if >9, add 6

BCD coded results

Decimal symbol	BCD digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110

Decimal Adders (2/3)

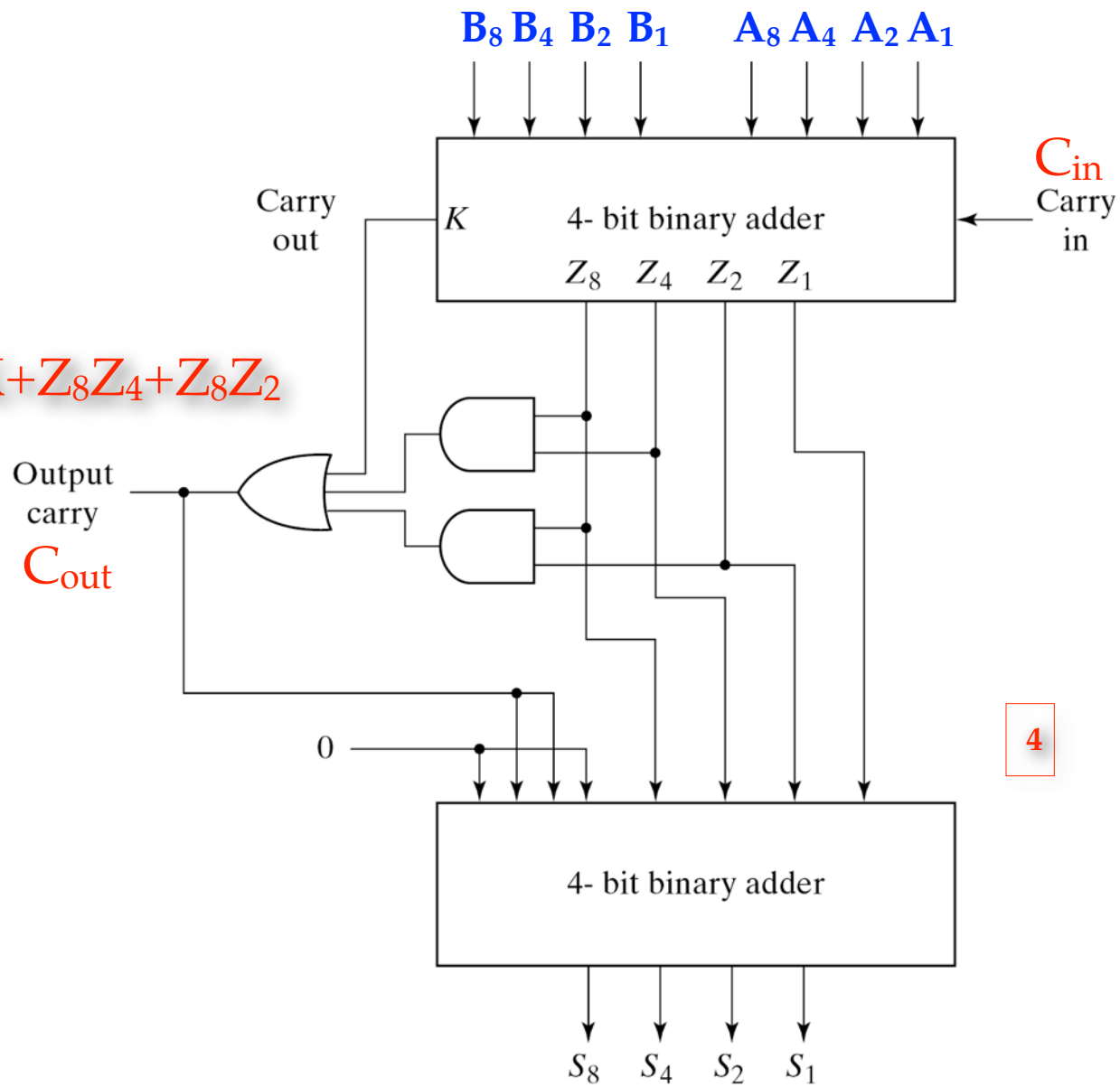
2 3

Z_8	Z_4	Z_2	Z_1
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Z_8Z_4

Z_8Z_2

Decimal Adders (3/3)



3 $C_{out}=K+Z_8Z_4+Z_8Z_2$

4

Verilog Construction

- 1. Use direct mapping of figure from P52
- 2. Use definition
  - two additions
    - $kz_3z_2z_1z_0 = a_3a_2a_1a_0 + b_3b_2b_1b_0$
    - $kz_3z_2z_1z_0 + 00110$
  - selection
    - output = $kz_3z_2z_1z_0$ (if $kz_3z_2z_1z_0 <= 6$)
    - output = $kz_3z_2z_1z_0 + 00110$ (if $kz_3z_2z_1z_0 > 6$)

Verilog Module Construction (1/2)

- Separate flip-flops with other logics (two types)
  - flip-flops (edge-triggered with clock, reset)
  - combinational logics (level sensitive)
- Combinational logics
  - simple logics (AND, OR, NOT)
  - coder/decoder (mapping, addressing)
  - comparison (conditional/equality test)
  - selection (select correct results, MUX)
  - arithmetic functions and superposition (+,-,*,binary shift)
- Finite state machine (FSM)

Verilog Module Construction (2/2)

- Separate flip-flops with other logics
  - For a D-type flip-flop

```
always @(posedge clk or negedge rst_n)
  if (~rst_n)
    q<=0;
  else
    q<=1;
```

  - For a 2-to-1 MUX

```
always @*
  if (select==1'b1)
    out=a;
  else
    out=b;
```

```
assign out = (select==1'b1) ? a : b;
```