NTHU EE

Laboratory for Reliable Computing

Introduction

Hsi-Pin Ma

http://lms.nthu.edu.tw/course/43639

Department of Electrical Engineering

National Tsing Hua University

Outline

Introduction

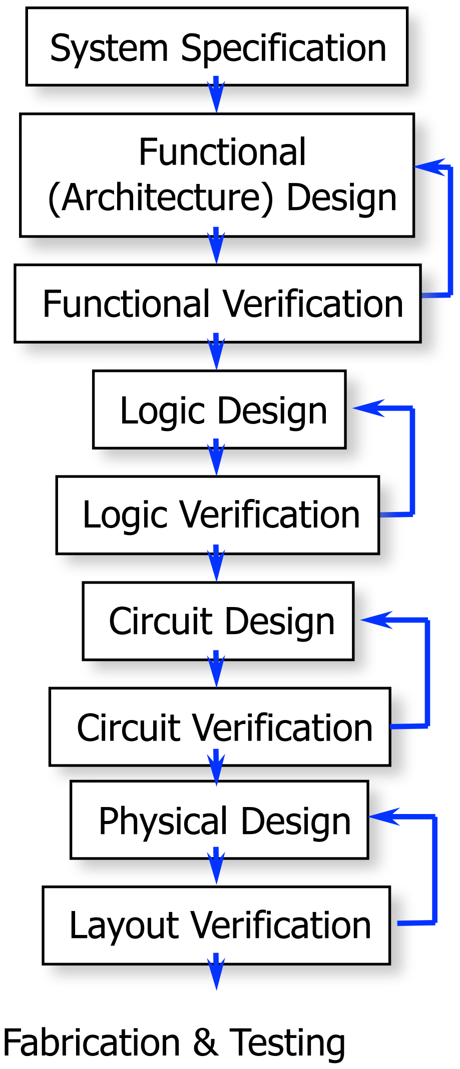
Hardware Description Language

- 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

- 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

VLSI Design Flow



Behavioral Representation (C++,Behavioral Verilog)

Logic (Gate-level) Representation (Gate-level Verilog)

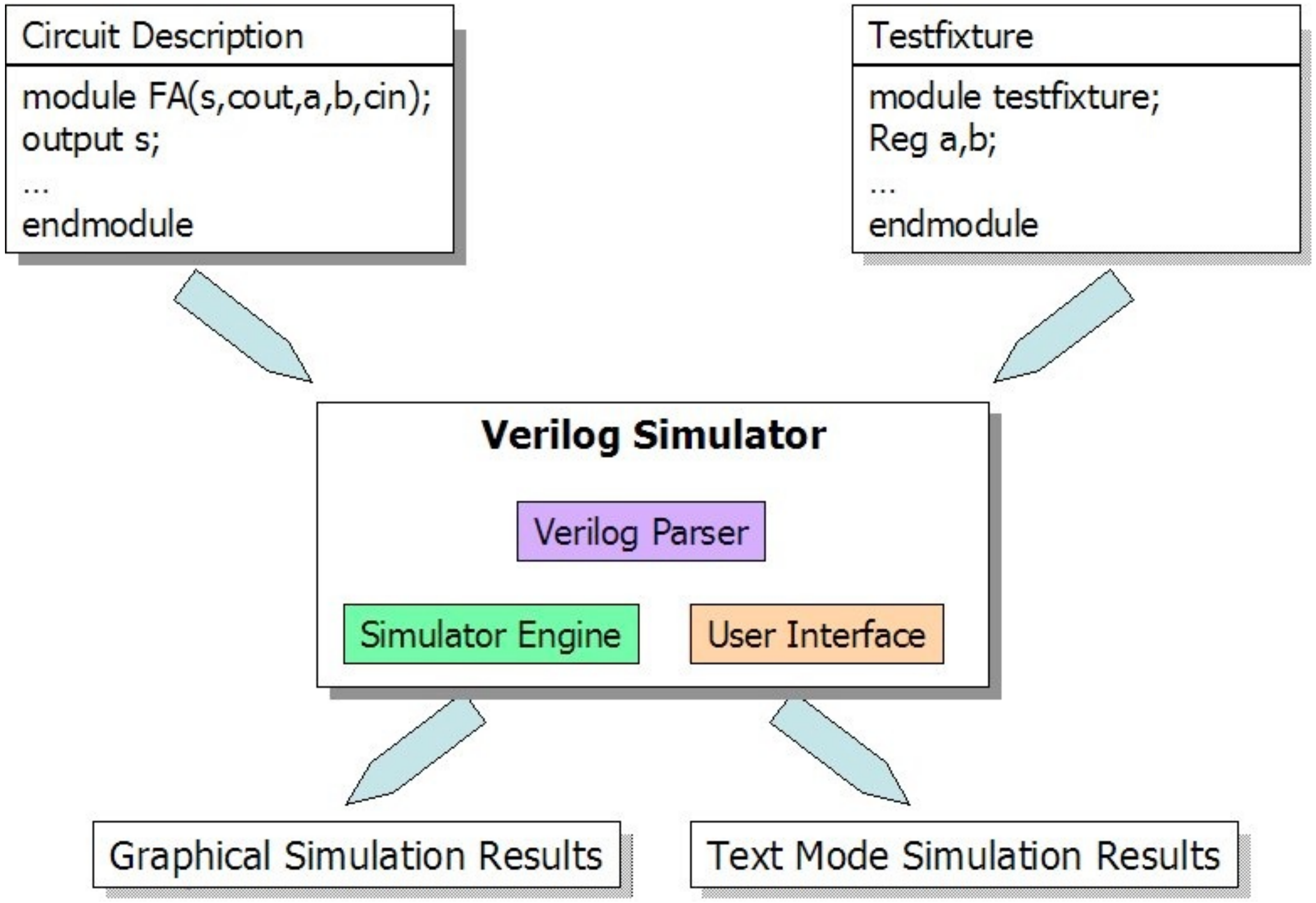
Circuit Representation (Hspice)

Layout Representation (GDSII, dB in FPGA)

Event Simulation of a Verilog Model

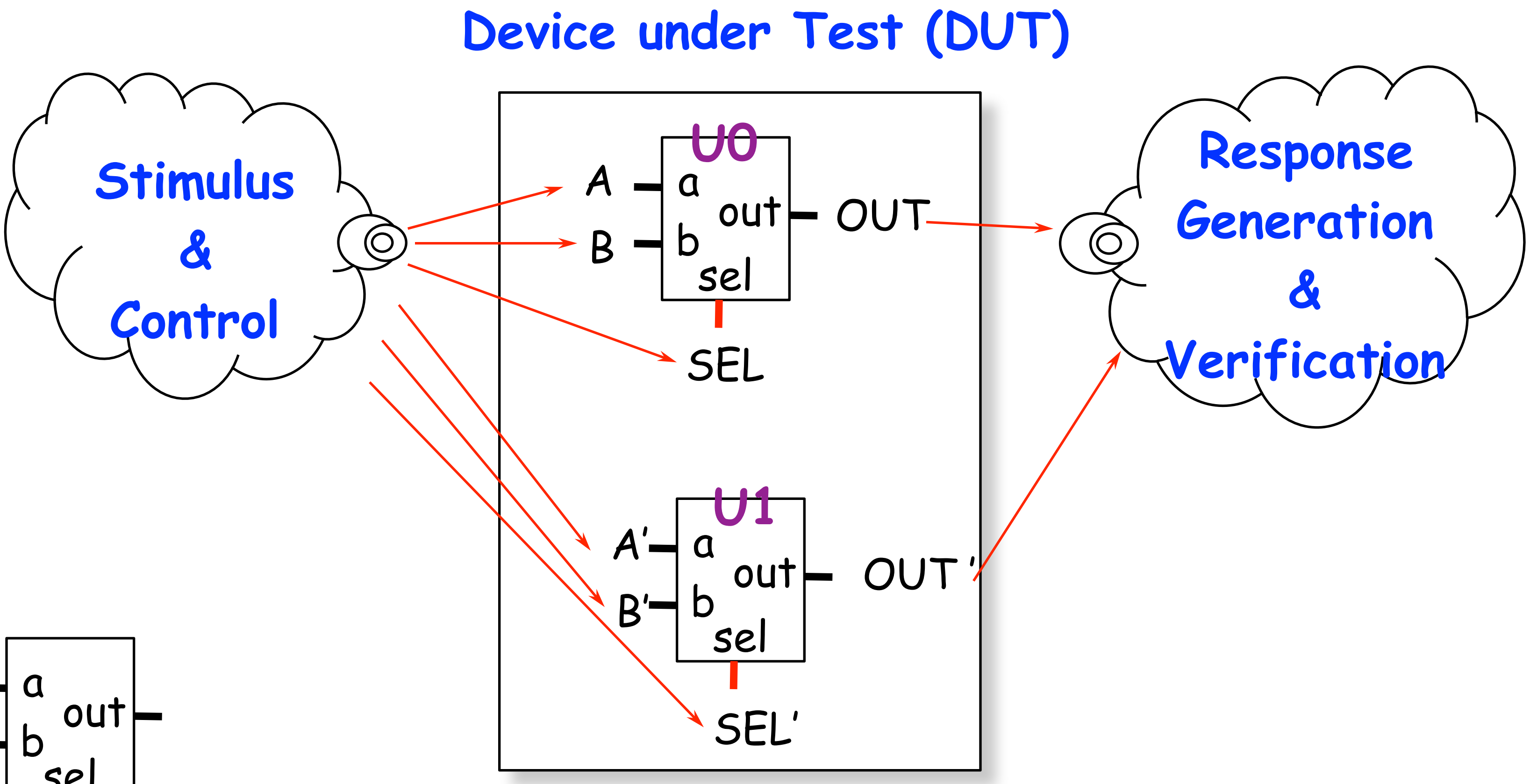
- Compilation
  - Compilation and elaboration
- Initialization
  - Initialize module parameters
    - Set other storage element to unknown (X) state
      - Unknown or un-initialized
    - Set undriven nets to the high-impedance (Z) state
      - Tri-state or floating
- Simulation

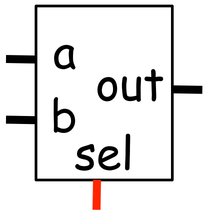
Verilog Simulation



Sample Design

Scenario





SMUX

Module + Testbench

Logic Function in Verilog

- Combinational logic
  - logic function: $\text{out} = a\cdot\text{sel} + b\cdot\text{sel}'$
  - assign out = (a&sel) | (b&(~sel)) ;
- Basic logic operator
  - AND: &
  - OR: |
  - NOT: ~

Verilog Module

module name declaration

I/O definition

module functionality

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

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

endmodule
```

NTHU EE

Laboratory for Reliable Computing

Verilog Testbench

module name declaration

I/O definition

module instantiation

apply stimulus

```
module test_smux;
wire OUT;
reg A,B,SEL;

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

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

endmodule
```

RTL Modeling

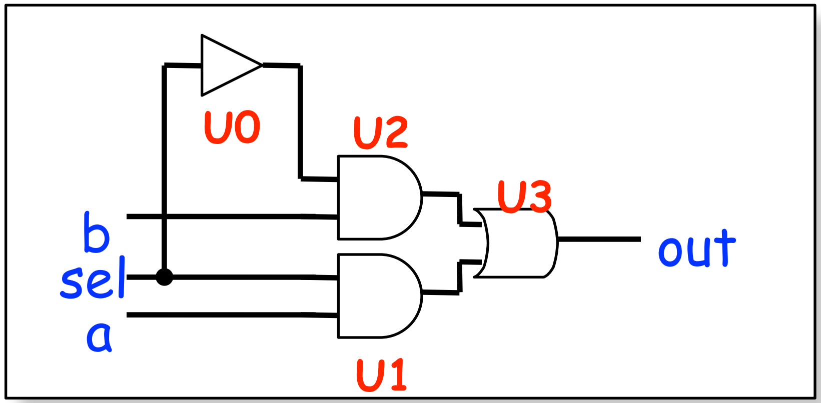
Two Primary Constructs

- continuous assignments
  - begin with an **assign** keyword, and can represent simple combinational logics
- always procedure blocks
  - A procedure block encapsulates one or more lines or programming statements, along with information about when the statements should be executed

Continuous Assignments

- **assign** continuous construct
  - combinational logics

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

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

endmodule
```

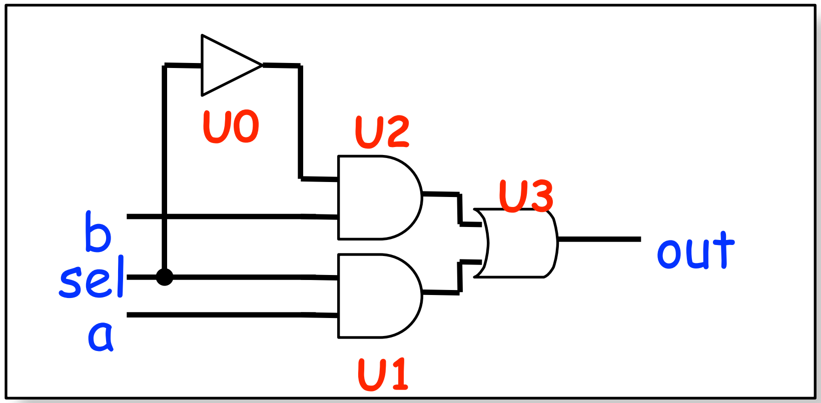


This out has to be declared as "wire" or or "output" data type.
This expression can not be inside always @().

always Procedure Block

- **always** statements

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

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

endmodule
```



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

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)

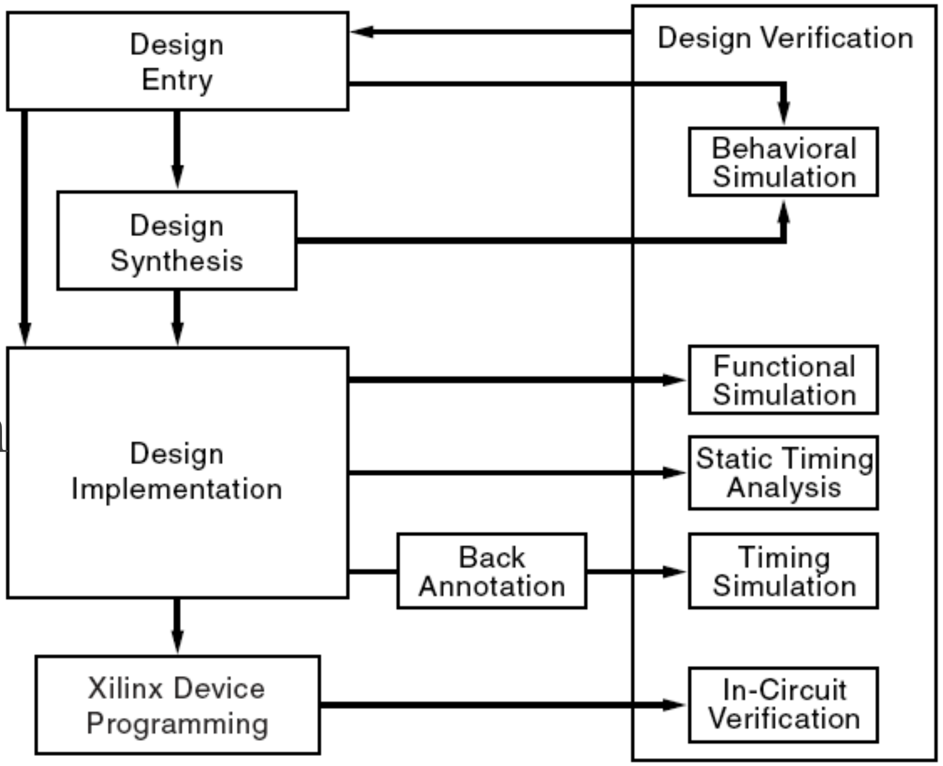
Logical Shift Operators		
OP	**Usage**	**Description**
<<	m << n	Shift m left n-times
>>	m >> n	Shift m right n-times

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

Logic Modeling and Simulation Using Xilinx Vivado

Design Flow

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

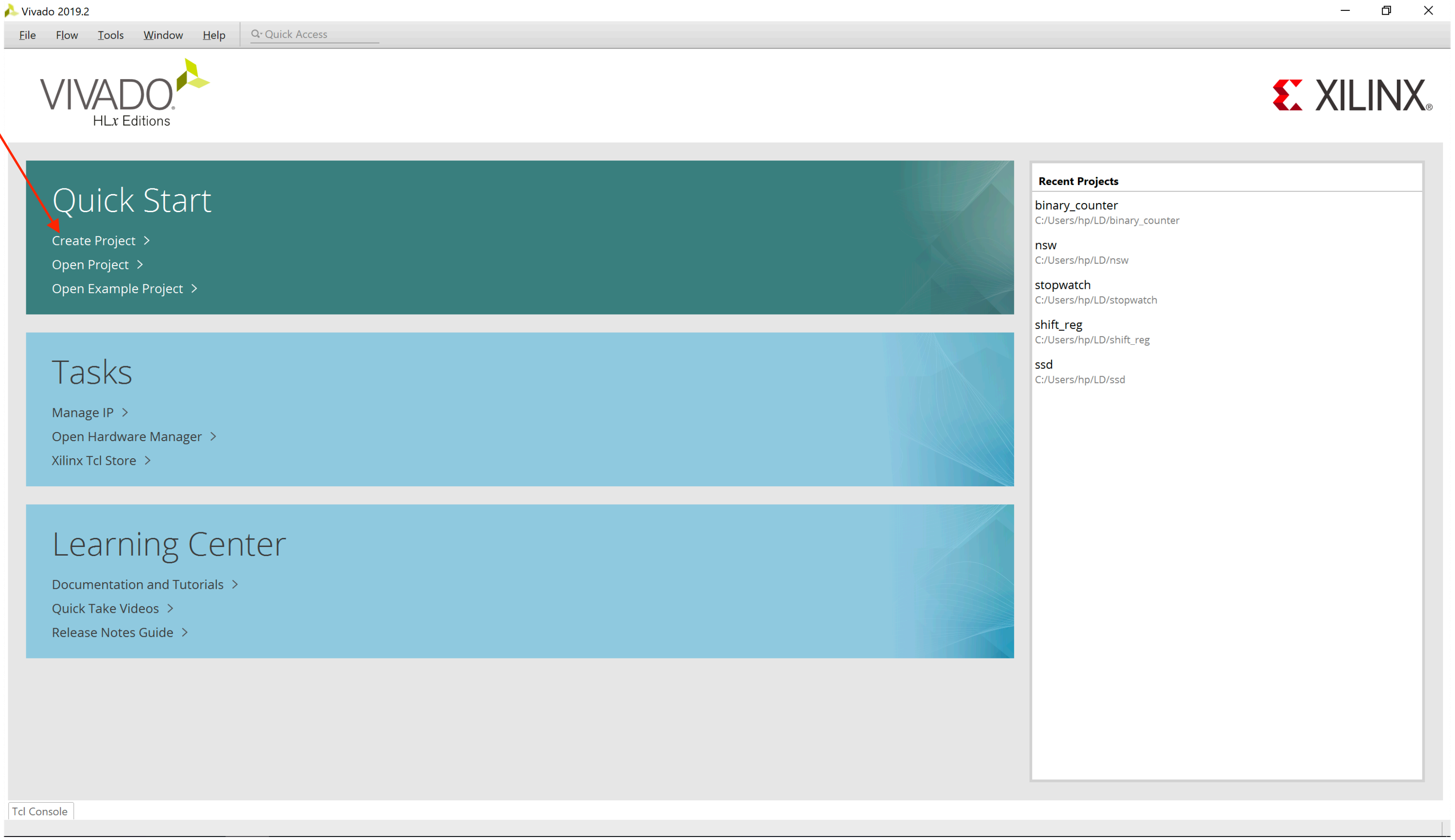


Important Notes

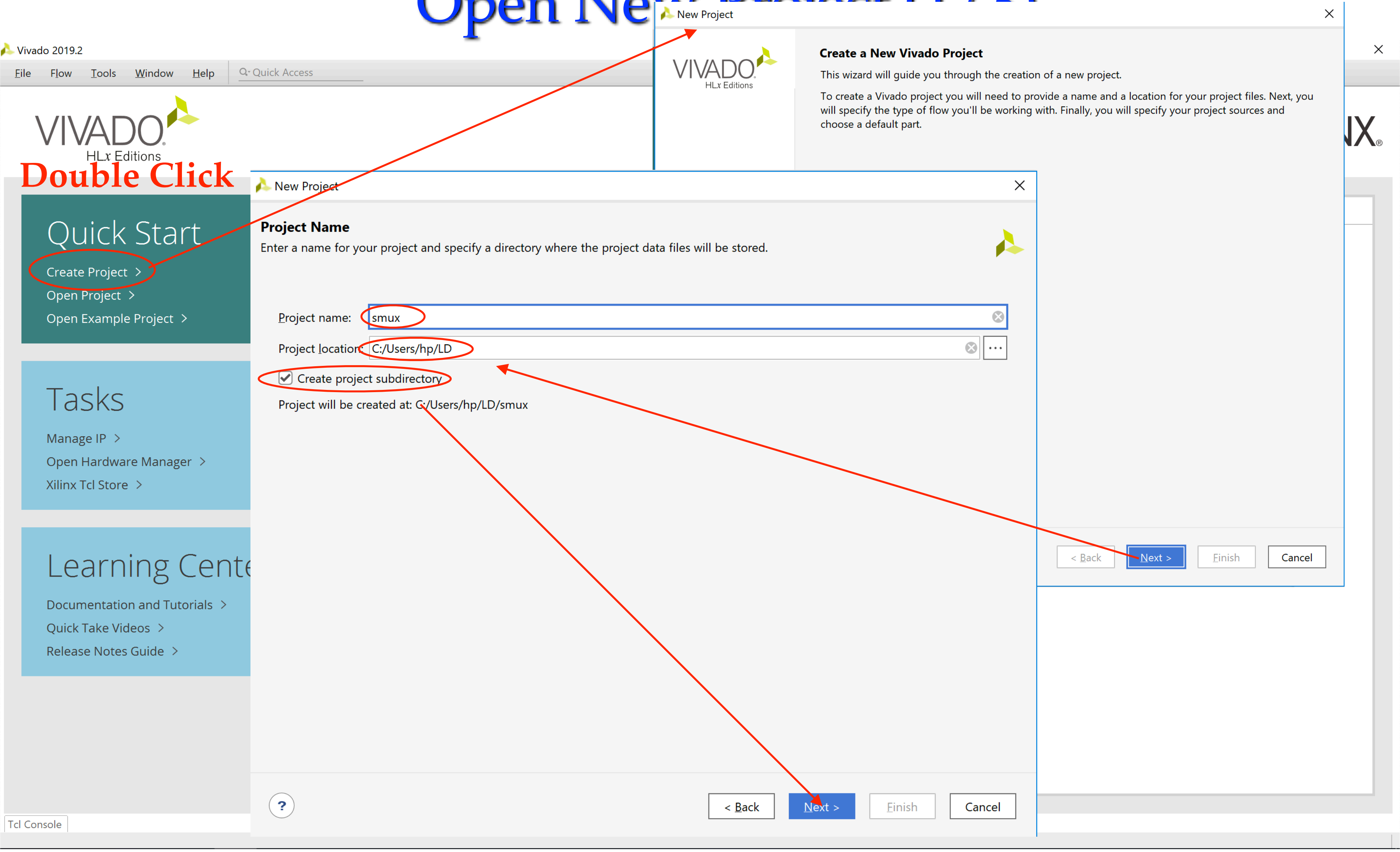
- **Draw schematic first** and then construct Verilog codes.
- Verilog RTL coding philosophy is not the same as C programming
  - Every Verilog RTL construct has its own logic mapping (for synthesis)
  - You should have the logics (draw schematic) first and then the RTL codes
  - You have to write **synthesizable**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

Open Vivado

Vivado


Vivado 2019.2
File Flow Tools Window Help
Quick Access
VIVADO HLx Editions
XILINX
Quick Start
Create Project >
Open Project >
Open Example Project >
Tasks
Manage IP >
Open Hardware Manager >
Xilinx Tcl Store >
Learning Center
Documentation and Tutorials >
Quick Take Videos >
Release Notes Guide >
Recent Projects
binary_counter
C:/Users/hp/LD/binary_counter
nsw
C:/Users/hp/LD/nsw
stopwatch
C:/Users/hp/LD/stopwatch
shift_reg
C:/Users/hp/LD/shift_reg
ssd
C:/Users/hp/LD/ssd
Tcl Console

Open New Project (1/3)



Create a New Vivado Project

This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

Project Name

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

Project name: smux

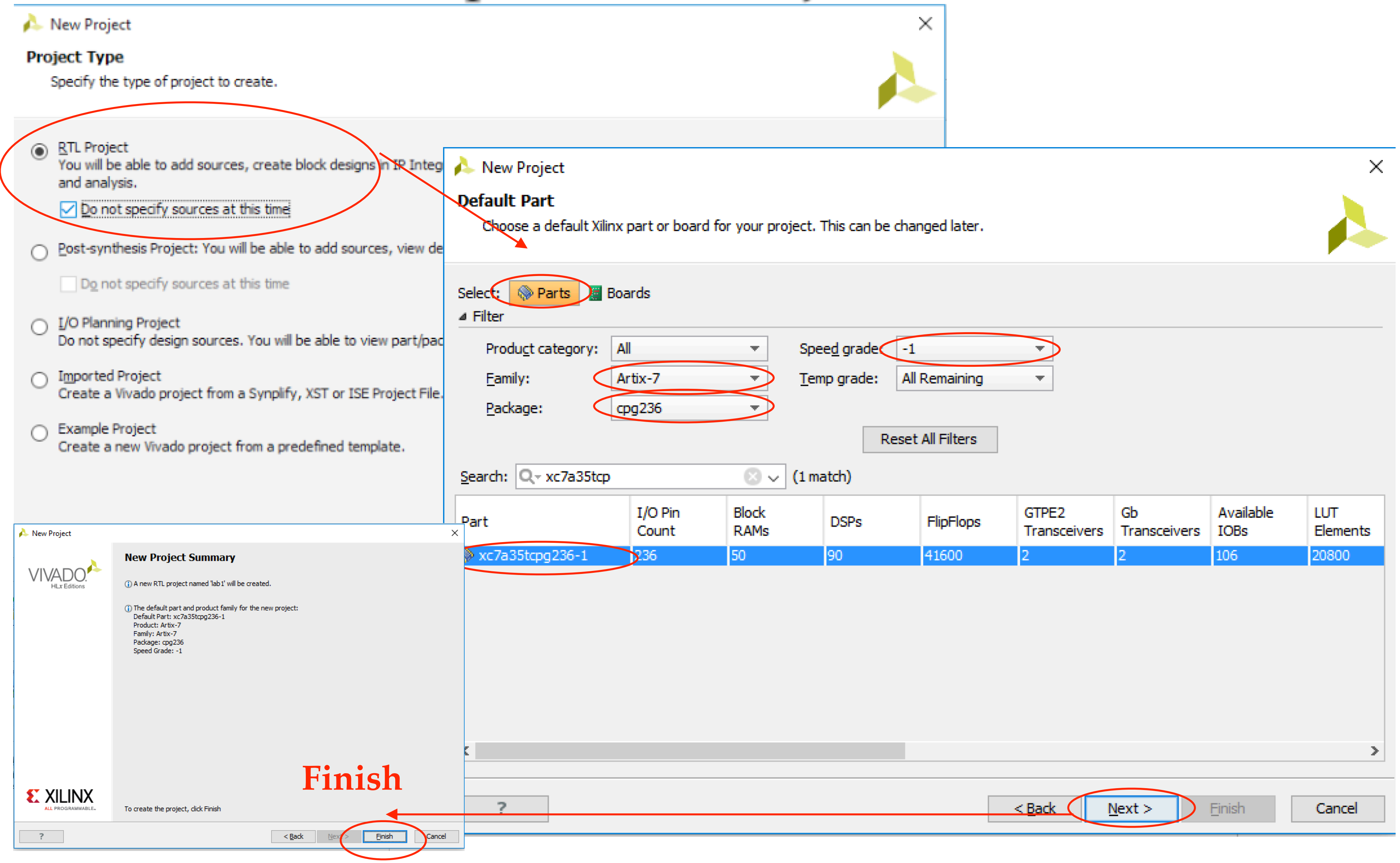
Project location: C:/Users/hp/LD

[✓] Create project subdirectory

Project will be created at: C:/Users/hp/LD/smux

Hsi-Pin Ma

Open New Project (2/3)



New Project

Project Type

Specify the type of project to create.

- [✓ RTL Project]
  You will be able to add sources, create block designs in IP Integ
  and analysis.
  - [✓ Do not specify sources at this time]
- Post-synthesis Project: You will be able to add sources, view de
  - Do not specify sources at this time
- I/O Planning Project
  Do not specify design sources. You will be able to view part/pac
- Imported Project
  Create a Vivado project from a Synplify, XST or ISE Project File.
- Example Project
  Create a new Vivado project from a predefined template.

New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Select: [✓ Parts] Boards

Filter

Product category:	All	Speed grade:	-1
Family:	Artix-7	Temp grade:	All Remaining
Package:	cpg236		

Reset All Filters

Search: xc7a35tcp (1 match)

Part	I/O Pin Count	Block RAMs	DSPs	FlipFlops	GTPE2 Transceivers	Gb Transceivers	Available IOBs	LUT Elements
xc7a35tcpg236-1	236	50	90	41600	2	2	106	20800

New Project

New Project Summary

ⓘ A new RTL project named 'lab1' will be created.

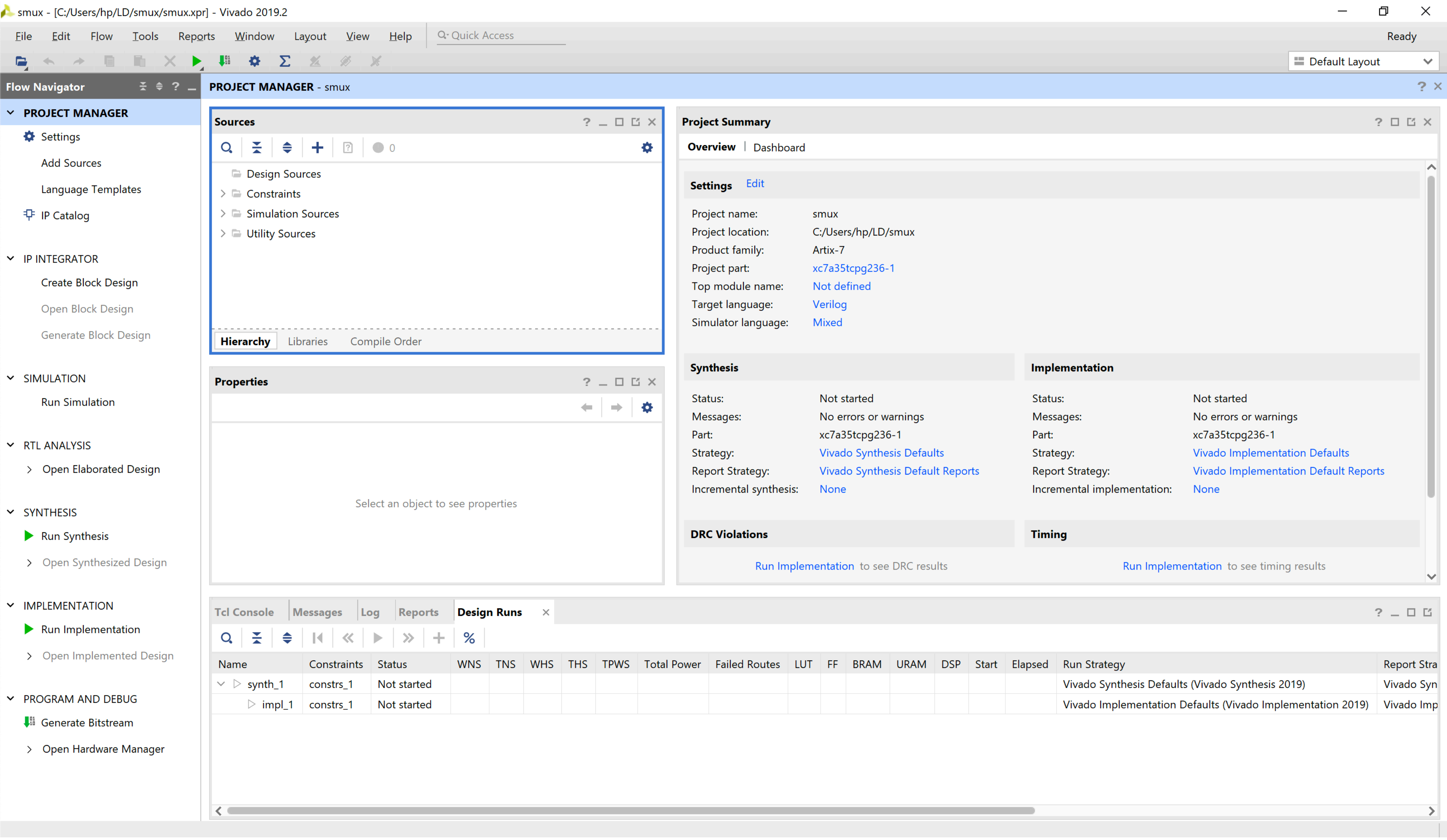
ⓘ The default part and product family for the new project:
Default Part: xc7a35tcpg236-1
Product: Artix-7
Family: Artix-7
Package: cpg236
Speed Grade: -1

To create the project, click Finish

Finish

Laboratory for Reliable Computing

Open New Project (3/3)


smux - [C:/Users/hp/LD/smux/smux.xpr] - Vivado 2019.2

File Edit Flow Tools Reports Window Layout View Help | Quick Access | Ready

Default Layout

Flow Navigator

PROJECT MANAGER
- Settings
- Add Sources
- Language Templates
- IP Catalog

IP INTEGRATOR
- Create Block Design
- Open Block Design
- Generate Block Design

SIMULATION
- Run Simulation

RTL ANALYSIS
- Open Elaborated Design

SYNTHESIS
- Run Synthesis
- Open Synthesized Design

IMPLEMENTATION
- Run Implementation
- Open Implemented Design

PROGRAM AND DEBUG
- Generate Bitstream
- Open Hardware Manager

PROJECT MANAGER - smux

Sources
- Design Sources
- Constraints
- Simulation Sources
- Utility Sources

Hierarchy | Libraries | Compile Order

Properties

Select an object to see properties

Project Summary

Overview | Dashboard

Settings Edit

Project name:	smux
Project location:	C:/Users/hp/LD/smux
Product family:	Artix-7
Project part:	xc7a35tcpg236-1
Top module name:	Not defined
Target language:	Verilog
Simulator language:	Mixed

Synthesis

Status:	Not started
Messages:	No errors or warnings
Part:	xc7a35tcpg236-1
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports
Incremental synthesis:	None

Implementation

Status:	Not started
Messages:	No errors or warnings
Part:	xc7a35tcpg236-1
Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Implementation Default Reports
Incremental implementation:	None

DRC Violations

Run Implementation to see DRC results

Timing

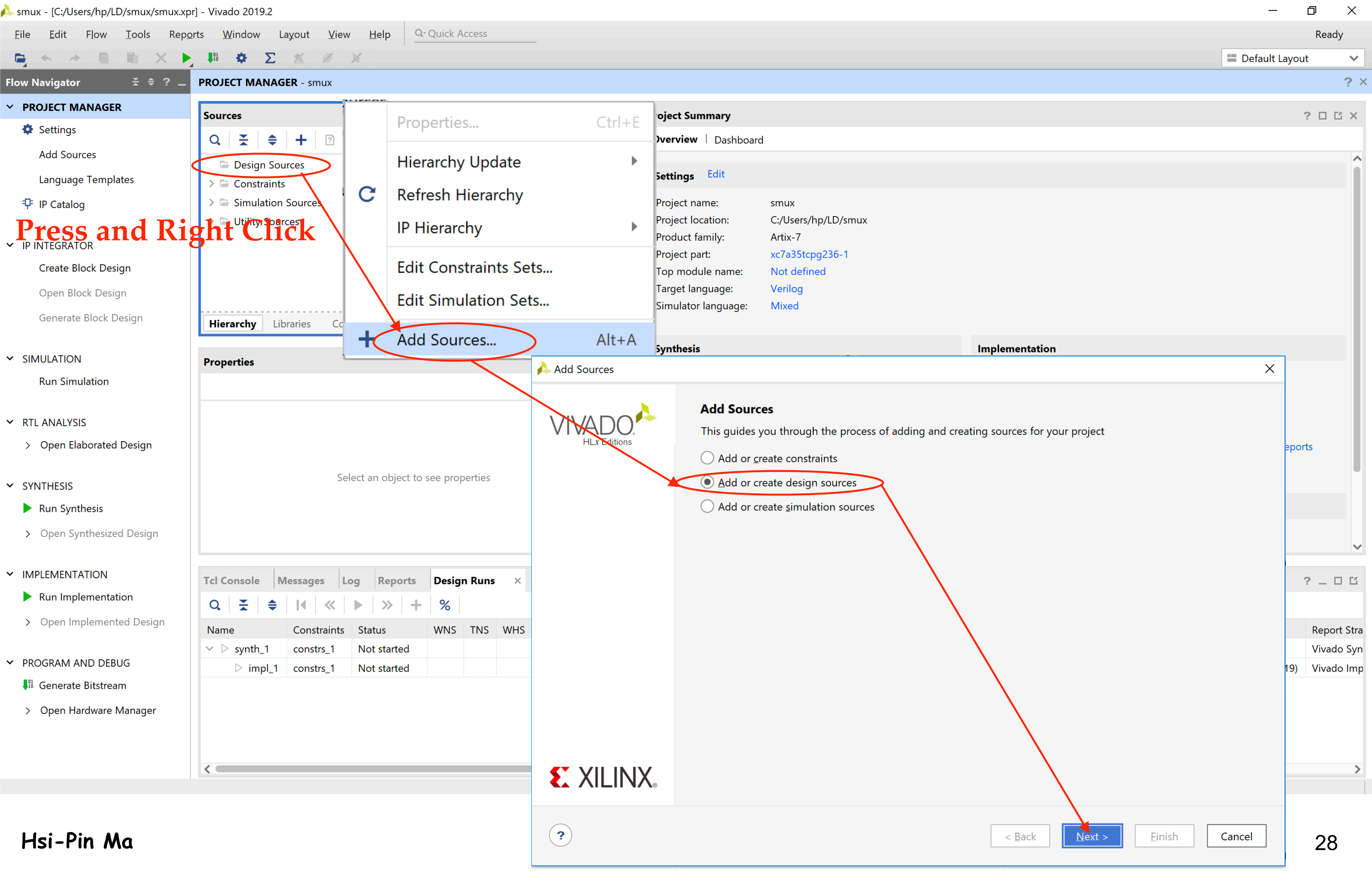
Run Implementation to see timing results

Tcl Console | Messages | Log | Reports | Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy	Report Stra
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado Synthesis 2019)	Vivado Syn
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Vivado Implementation 2019)	Vivado Imp

Laboratory for Reliable Computing

New Source (1/5)



smux - [C:/Users/hp/LD/smux/smux.xpr] - Vivado 2019.2

PROJECT MANAGER - smux

Project name:	smux
Project location:	C:/Users/hp/LD/smux
Product family:	Artix-7
Project part:	xc7a35tcpg236-1
Top module name:	Not defined
Target language:	Verilog
Simulator language:	Mixed

Name	Constraints	Status	WNS	TNS	WHS
synth_1	constrs_1	Not started			
impl_1	constrs_1	Not started			

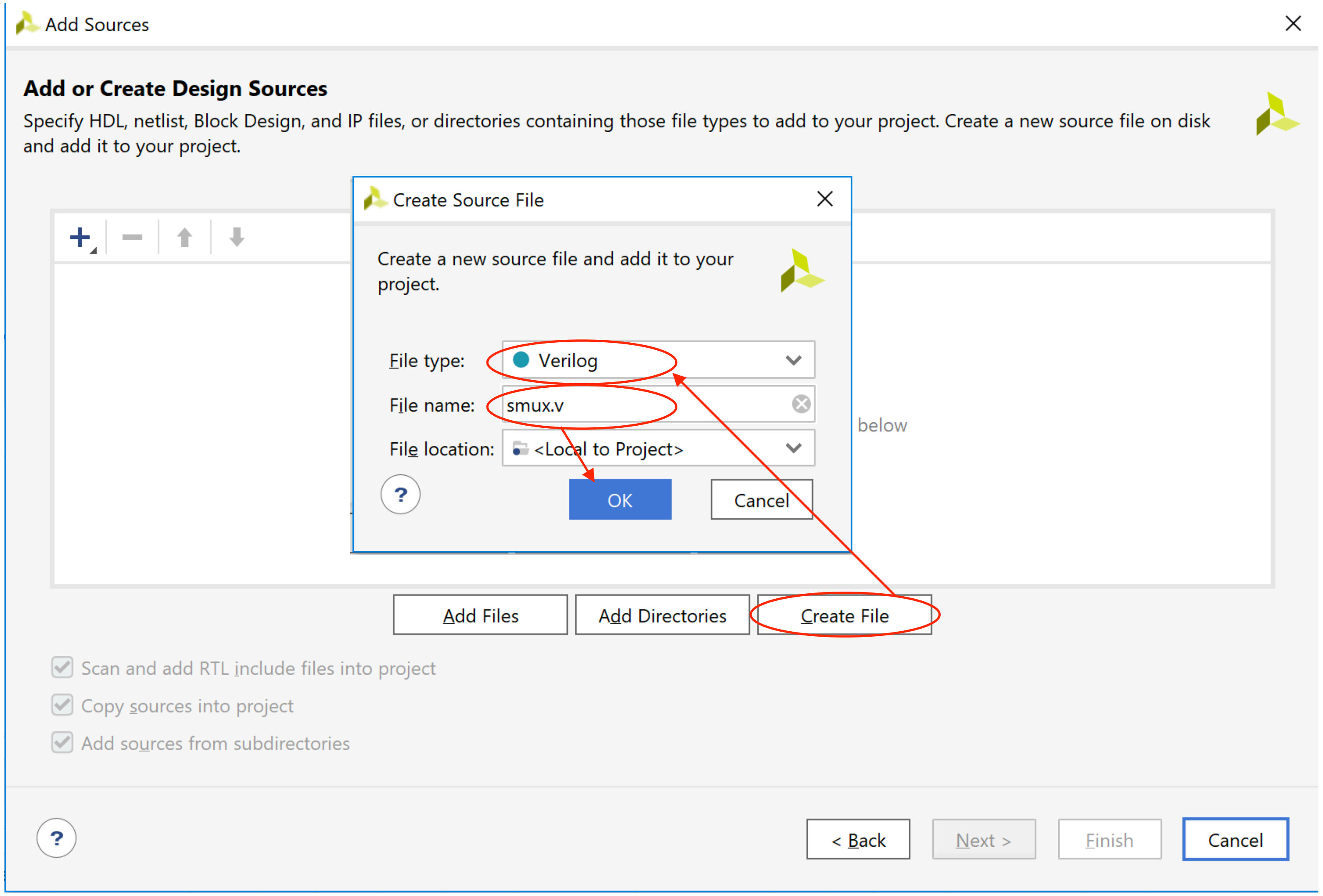
Add Sources

This guides you through the process of adding and creating sources for your project

( ) Add or create constraints
(●) Add or create design sources
( ) Add or create simulation sources

Hsi-Pin Ma

New Source (2/5)


Add Sources

Add or Create Design Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

Create Source File

Create a new source file and add it to your project.

File type: Verilog

File name: smux.v

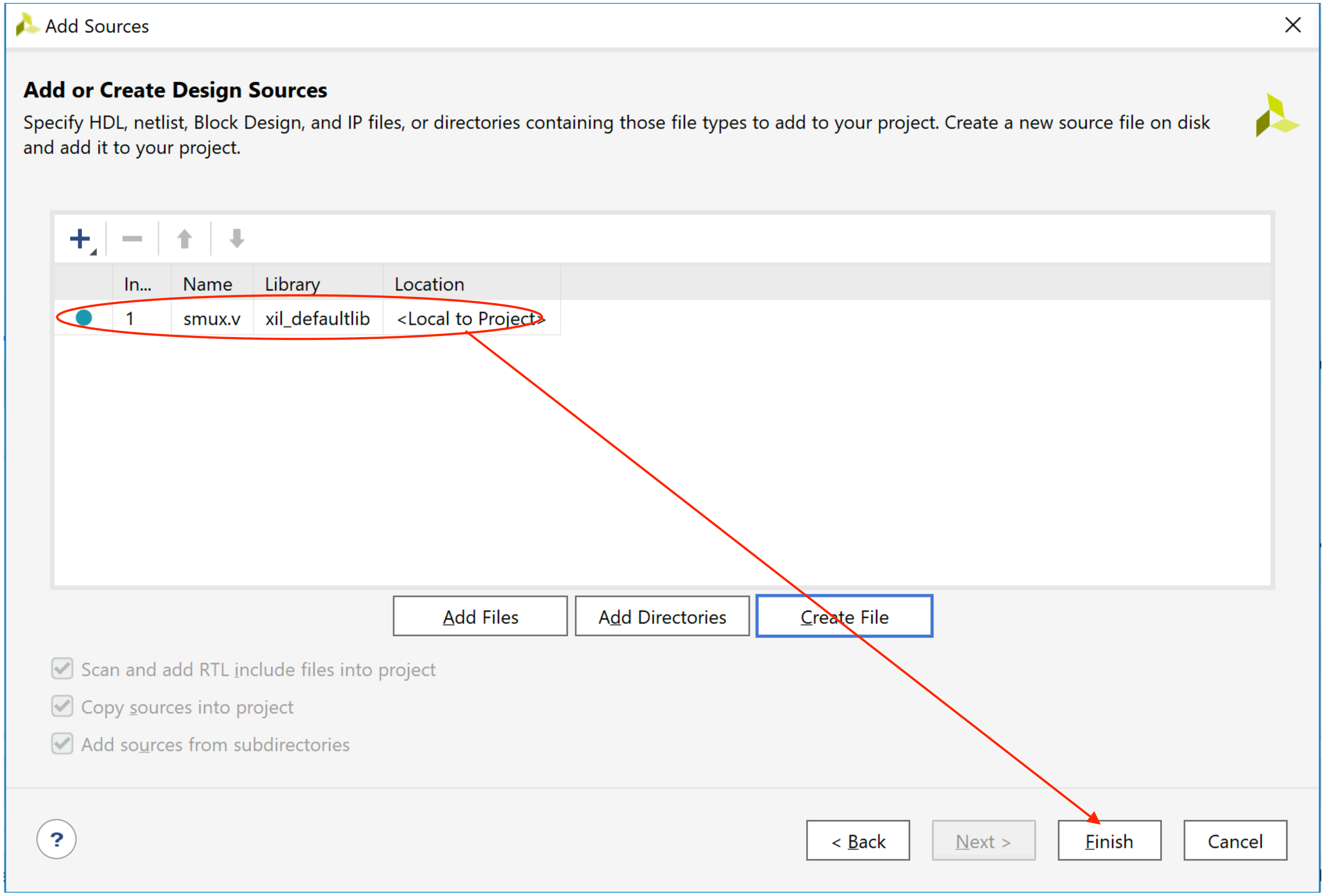
File location: <Local to Project>

Scan and add RTL include files into project ✓

Copy sources into project ✓

Add sources from subdirectories ✓

New Source (3/5)



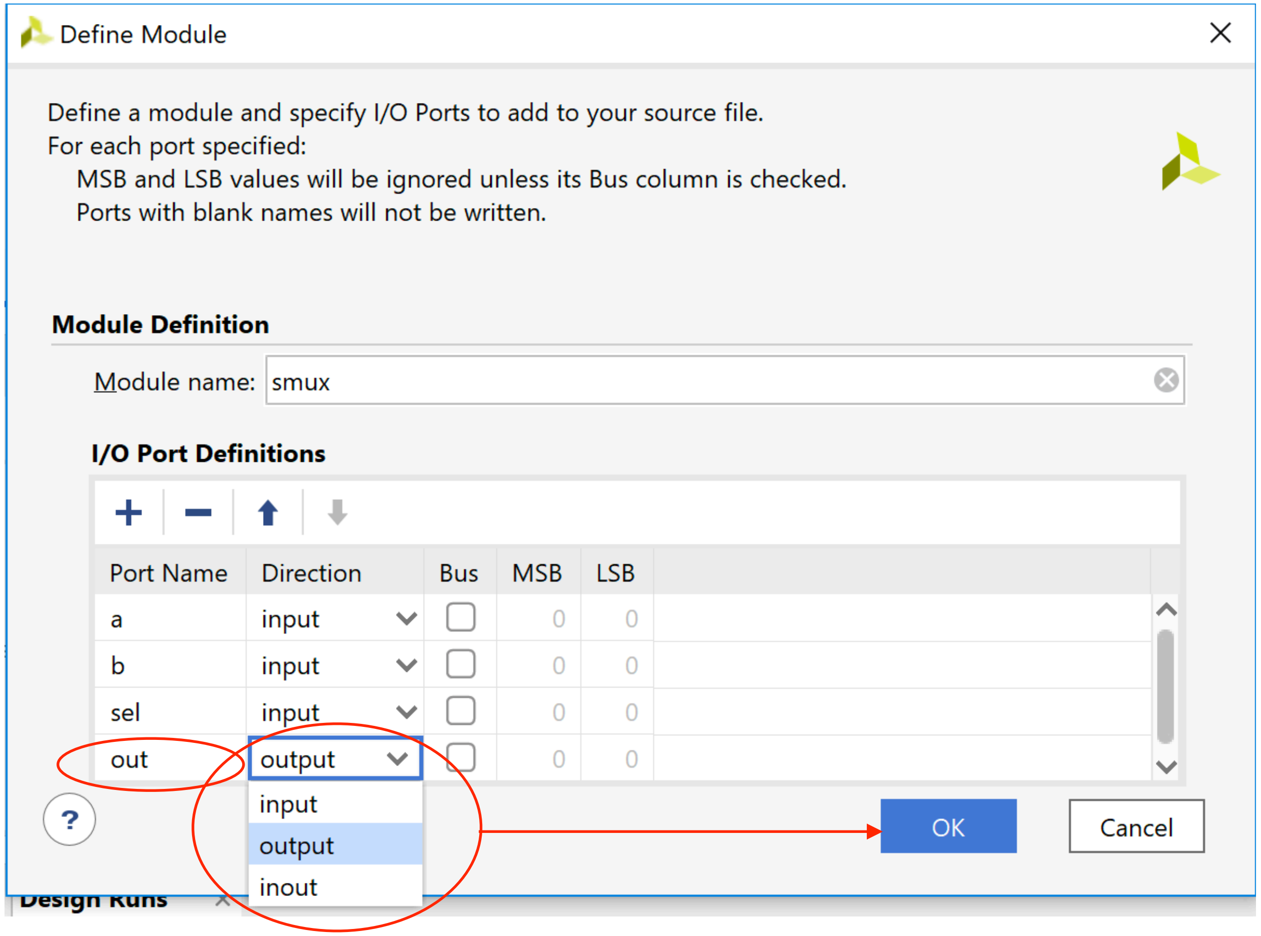
Add or Create Design Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

	In...	Name	Library	Location
●	1	smux.v	xil_defaultlib	<Local to Project>

- [✓] Scan and add RTL include files into project
- [✓] Copy sources into project
- [✓] Add sources from subdirectories

New Source (4/5)



Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

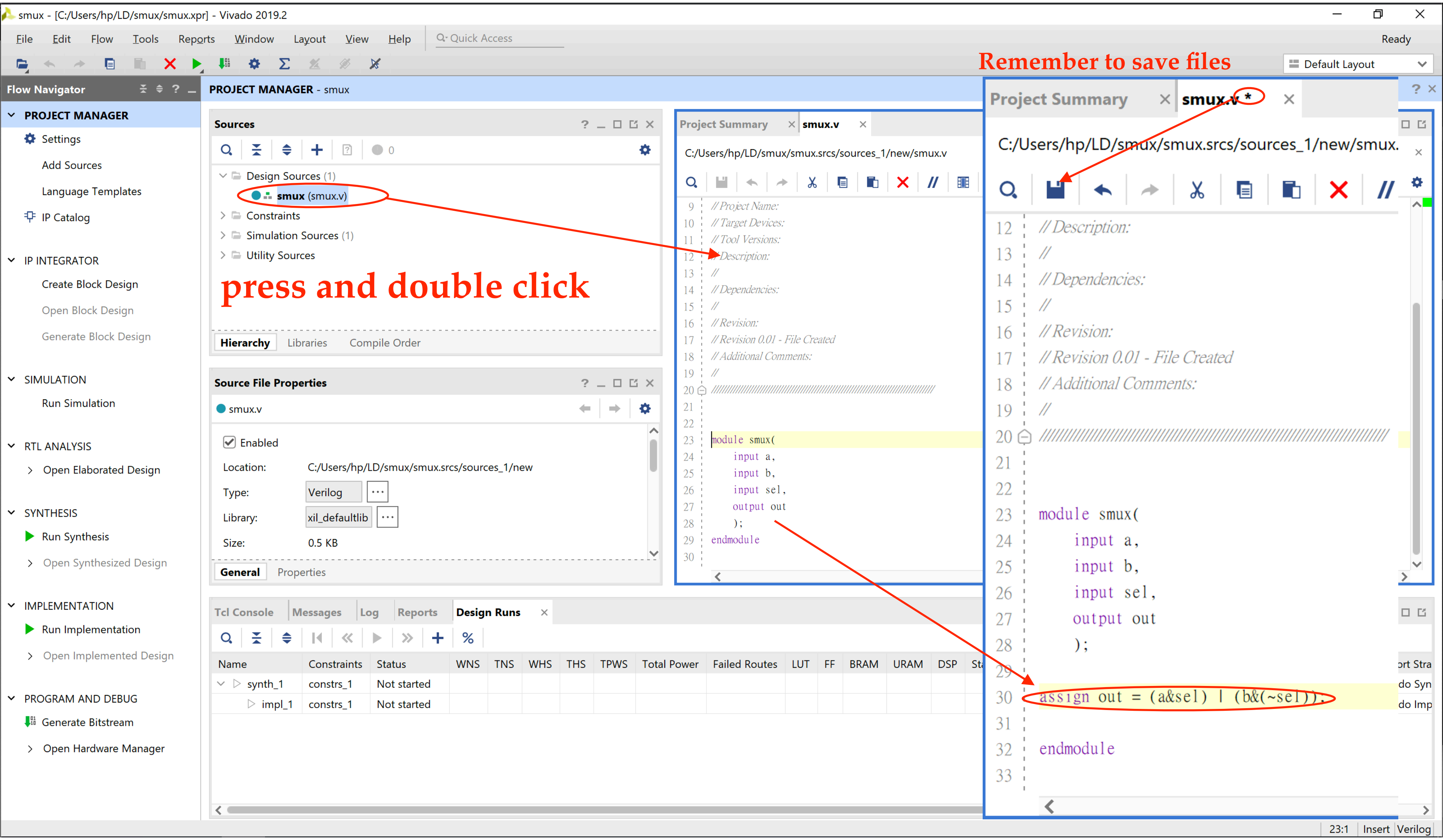
Module name: smux

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
a	input	☐	0	0
b	input	☐	0	0
sel	input	☐	0	0
out	output	☐	0	0

Laboratory for Reliable Computing

New Source (5/5)



smux - [C:/Users/hp/LD/smux/smux.xpr] - Vivado 2019.2

File Edit Flow Tools Reports Window Layout View Help Quick Access Ready

Remember to save files

Default Layout

Flow Navigator

PROJECT MANAGER
- Settings
- Add Sources
- Language Templates
- IP Catalog

IP INTEGRATOR
- Create Block Design
- Open Block Design
- Generate Block Design

SIMULATION
- Run Simulation

RTL ANALYSIS
- Open Elaborated Design

SYNTHESIS
- Run Synthesis
- Open Synthesized Design

IMPLEMENTATION
- Run Implementation
- Open Implemented Design

PROGRAM AND DEBUG
- Generate Bitstream
- Open Hardware Manager

PROJECT MANAGER - smux

Sources

- Design Sources (1)
  - smux (smux.v)
- Constraints
- Simulation Sources (1)
- Utility Sources

press and double click

Hierarchy Libraries Compile Order

Source File Properties

smux.v

Enabled	✓
Location:	C:/Users/hp/LD/smux/smux.srcs/sources_1/new
Type:	Verilog
Library:	xil_defaultlib
Size:	0.5 KB

General Properties

Tcl Console Messages Log Reports Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Sta
synth_1	constrs_1	Not started													
impl_1	constrs_1	Not started													

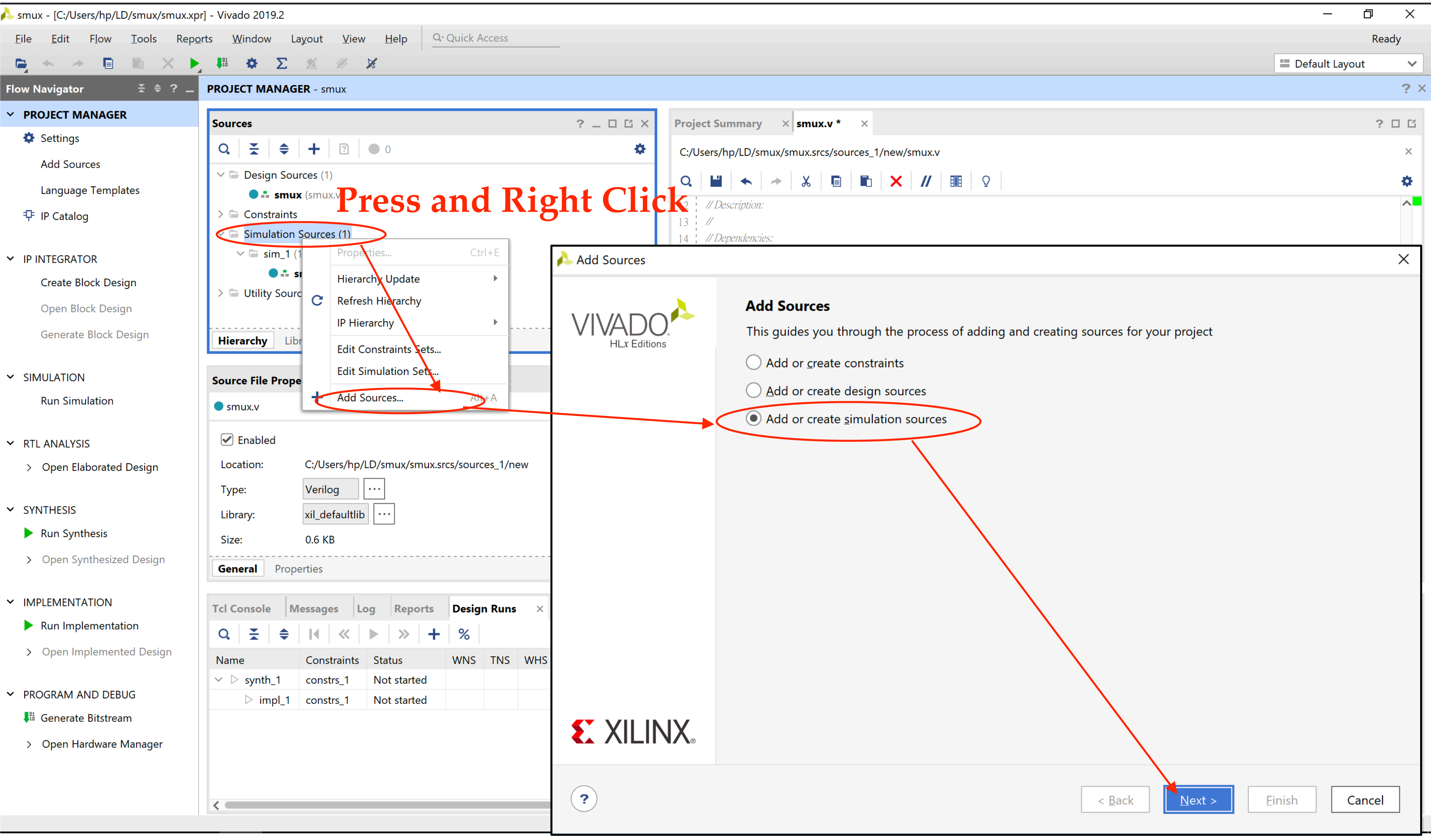
Project Summary smux.v *

C:/Users/hp/LD/smux/smux.srcs/sources_1/new/smux.v

```
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module smux(
    input a,
    input b,
    input sel,
    output out
    );

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

endmodule
```

23:1 Insert Verilog

Add Testbench (1/5)



Name	Constraints	Status	WNS	TNS	WHS
synth_1	constrs_1	Not started			
impl_1	constrs_1	Not started			

Add Sources

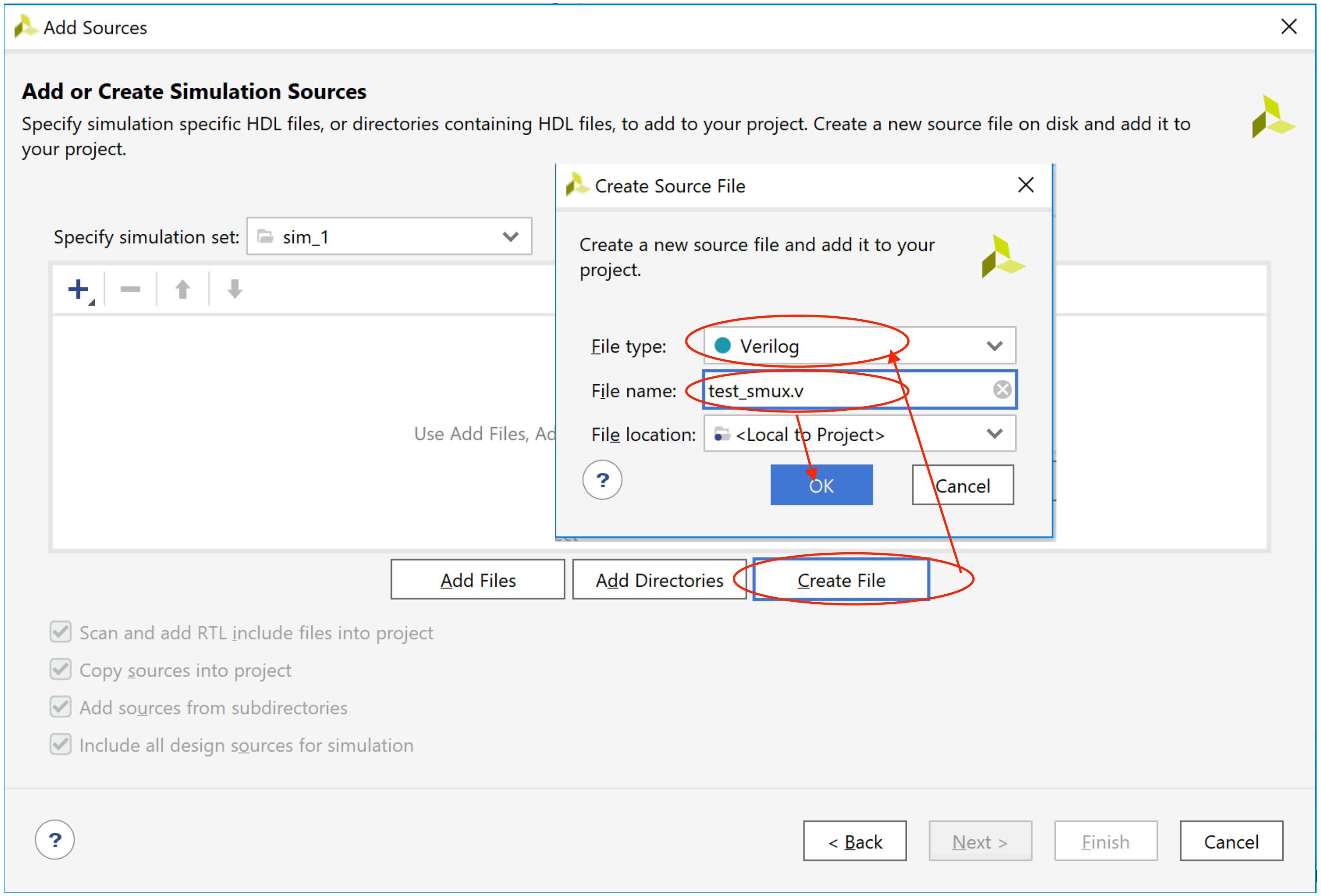
This guides you through the process of adding and creating sources for your project

( ) Add or create constraints
( ) Add or create design sources
[✓ Add or create simulation sources]

Hsi-Pin Ma

NTHU EE Laboratory for Reliable Computing

Add Testbench (2/5)


Add Sources

Add or Create Simulation Sources

Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.

Specify simulation set: sim_1

Use Add Files, Ad

Create Source File

Create a new source file and add it to your project.

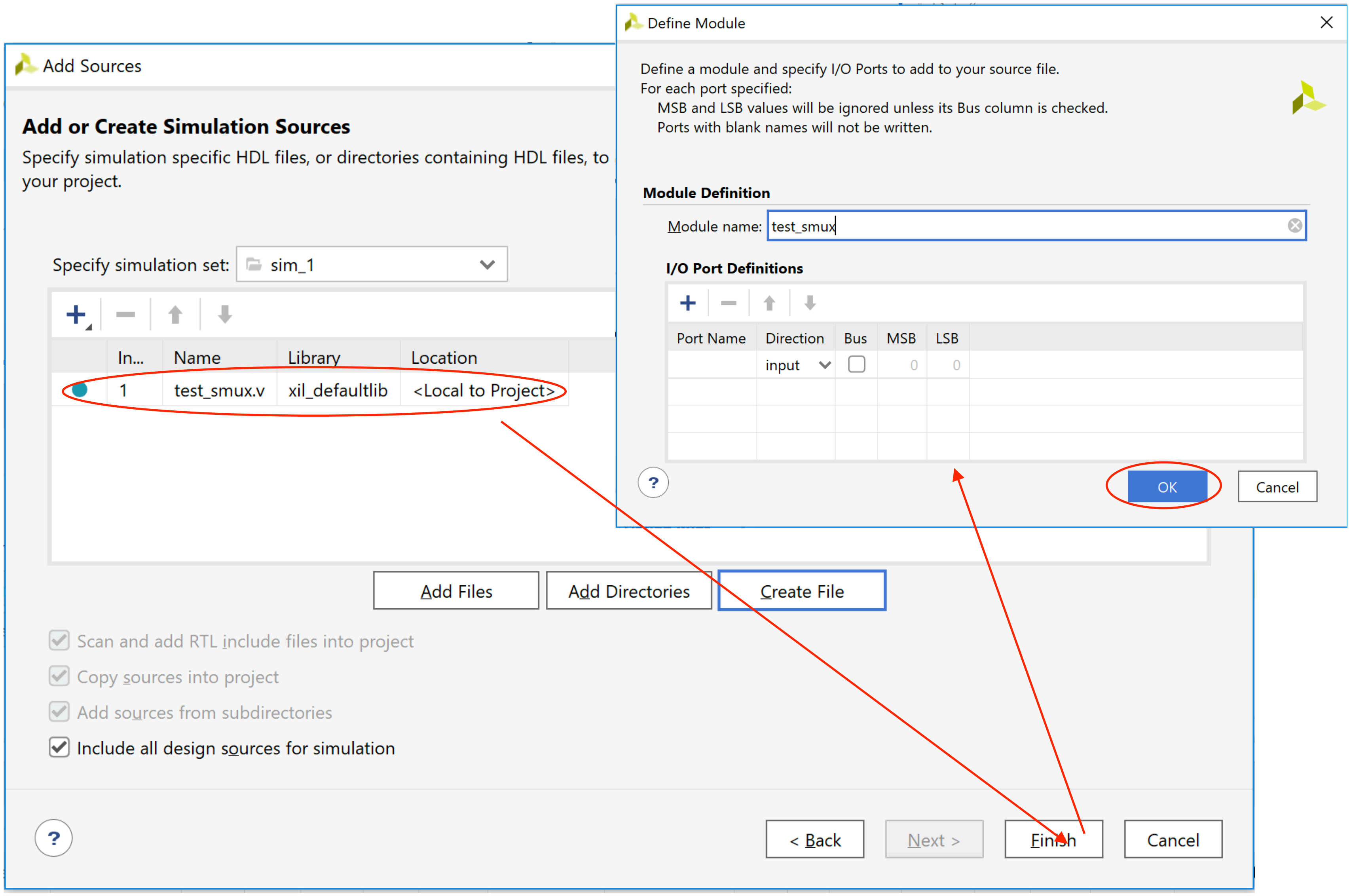
File type: Verilog

File name: test_smux.v

File location: <Local to Project>

- ✓ Scan and add RTL include files into project
- ✓ Copy sources into project
- ✓ Add sources from subdirectories
- ✓ Include all design sources for simulation

Add Testbench (3/5)



Add Sources

Add or Create Simulation Sources

Specify simulation specific HDL files, or directories containing HDL files, to your project.

Specify simulation set: sim_1

	In...	Name	Library	Location
●	1	test_smux.v	xil_defaultlib	<Local to Project>

Scan and add RTL include files into project
Copy sources into project
Add sources from subdirectories
[✓] Include all design sources for simulation

Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

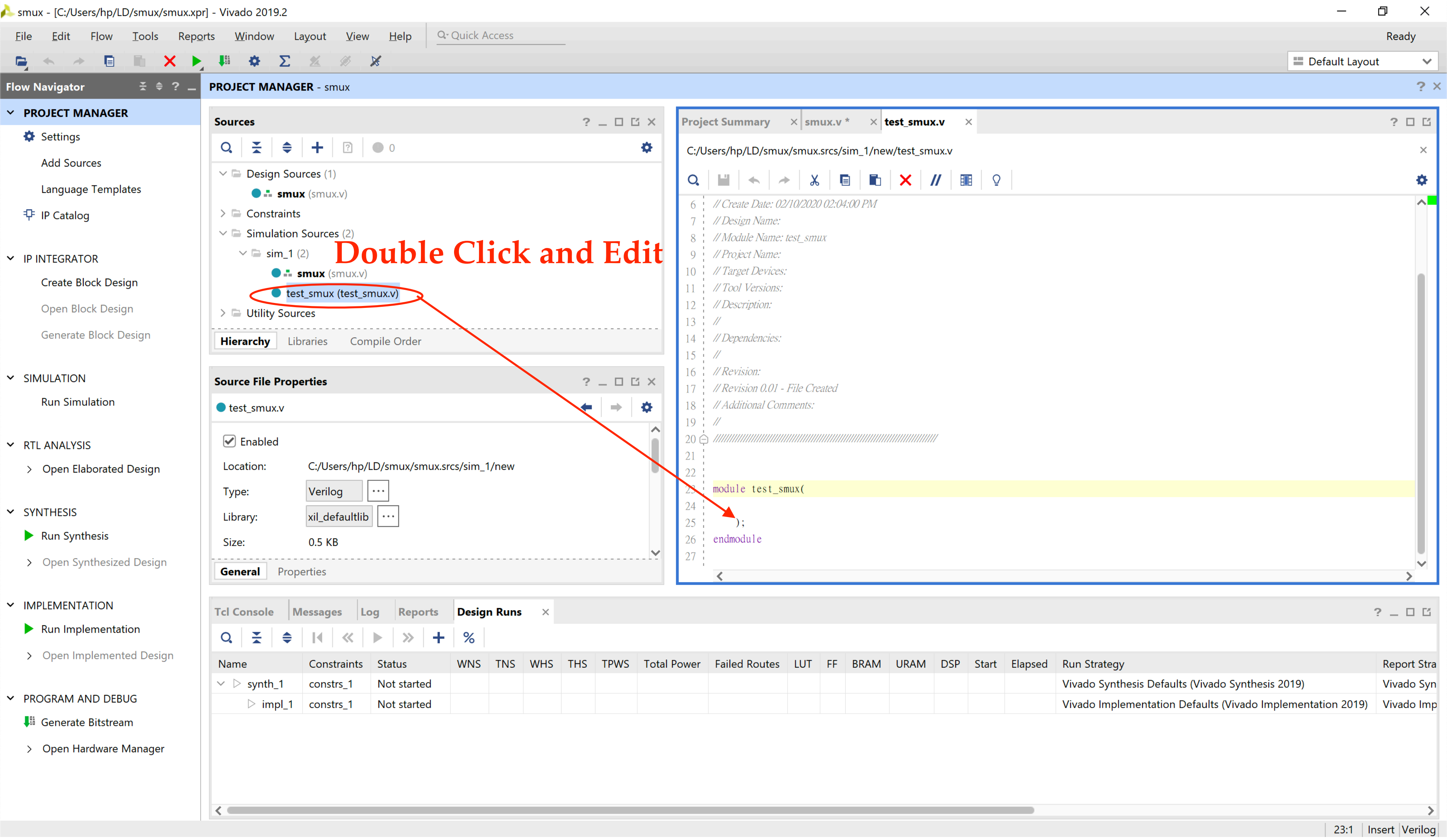
Module Definition

Module name: test_smux

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
	input	☐	0	0

Add Testbench (4/5)



smux - [C:/Users/hp/LD/smux/smux.xpr] - Vivado 2019.2

Design Sources (1)
- smux (smux.v)

Constraints

Simulation Sources (2)
- sim_1 (2)
  - smux (smux.v)
  - test_smux (test_smux.v)

Utility Sources

Double Click and Edit

test_smux.v

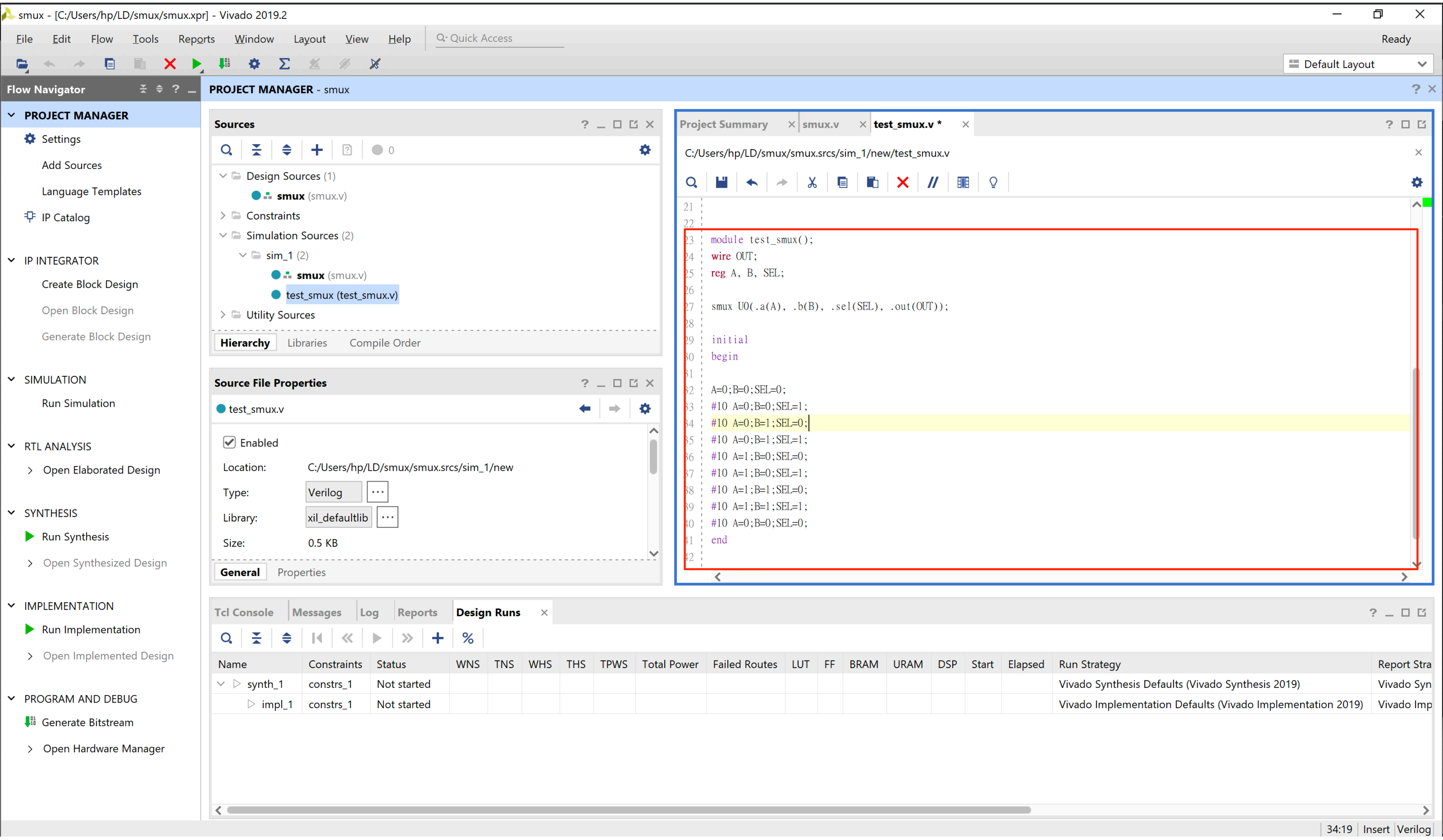
- [✓] Enabled
- Location: C:/Users/hp/LD/smux/smux.srcs/sim_1/new
- Type: Verilog
- Library: xil_defaultlib
- Size: 0.5 KB

C:/Users/hp/LD/smux/smux.srcs/sim_1/new/test_smux.v

```verilog
// Create Date: 02/10/2020 02:04:00 PM
// Design Name:
// Module Name: test_smux
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module test_smux(

    );
endmodule
```

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy	Report Stra
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado Synthesis 2019)	Vivado Syn
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Vivado Implementation 2019)	Vivado Imp

Add Testbench (5/5)



smux - [C:/Users/hp/LD/smux/smux.xpr] - Vivado 2019.2

PROJECT MANAGER - smux

Sources
- Design Sources (1)
  - smux (smux.v)
- Constraints
- Simulation Sources (2)
  - sim_1 (2)
    - smux (smux.v)
    - test_smux (test_smux.v)
- Utility Sources

Source File Properties

test_smux.v

- Enabled: ✓
- Location: C:/Users/hp/LD/smux/smux.srcs/sim_1/new
- Type: Verilog
- Library: xil_defaultlib
- Size: 0.5 KB

C:/Users/hp/LD/smux/smux.srcs/sim_1/new/test_smux.v

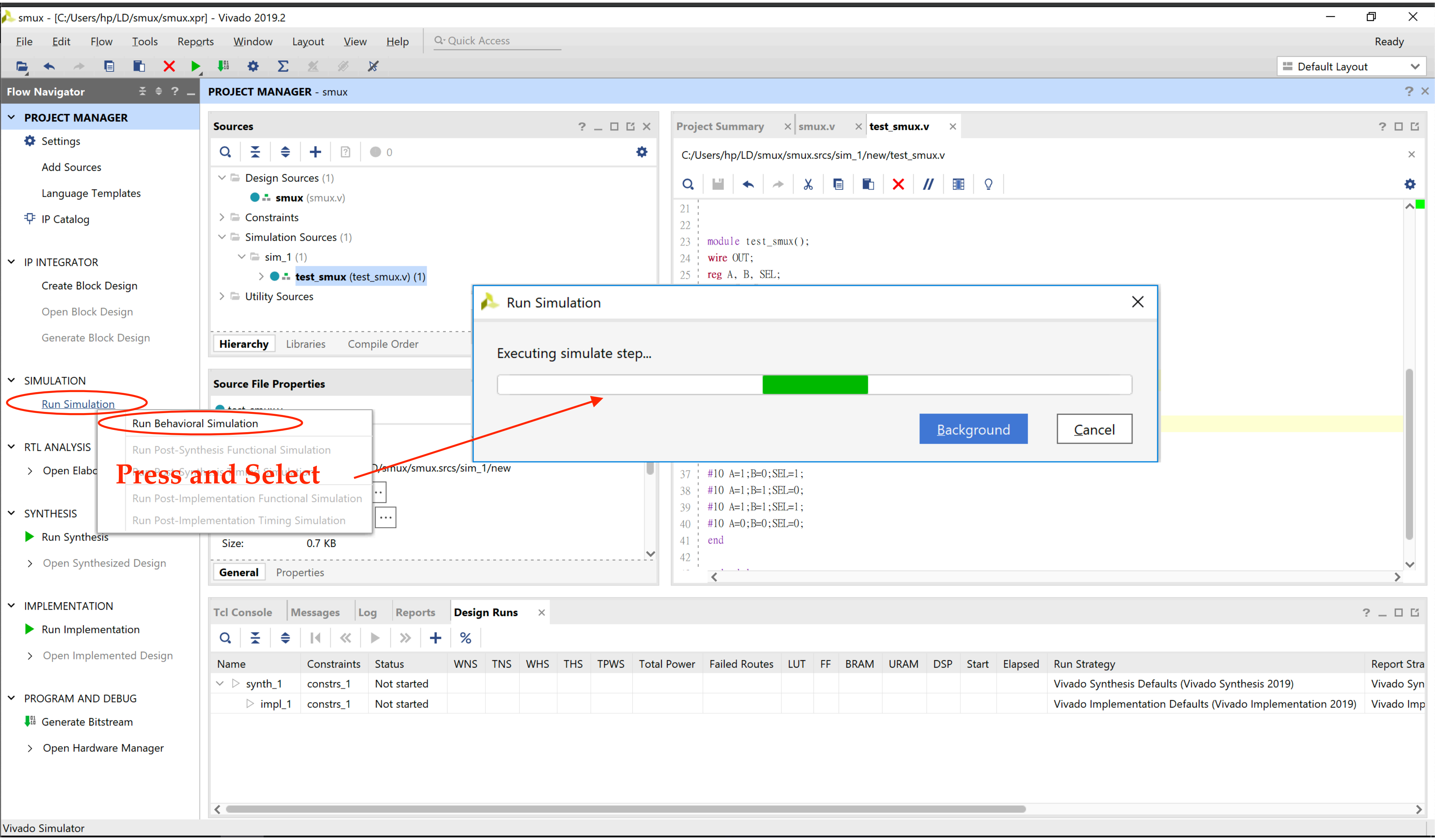
```verilog


module test_smux();
wire OUT;
reg A, B, SEL;

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

initial
begin

A=0;B=0;SEL=0;
#10 A=0;B=0;SEL=1;
#10 A=0;B=1;SEL=0;
#10 A=0;B=1;SEL=1;
#10 A=1;B=0;SEL=0;
#10 A=1;B=0;SEL=1;
#10 A=1;B=1;SEL=0;
#10 A=1;B=1;SEL=1;
#10 A=0;B=0;SEL=0;
end

```

Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy	Report Stra
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado Synthesis 2019)	Vivado Syn
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Vivado Implementation 2019)	Vivado Imp

34:19 Insert Verilog

Simulation (1/4)



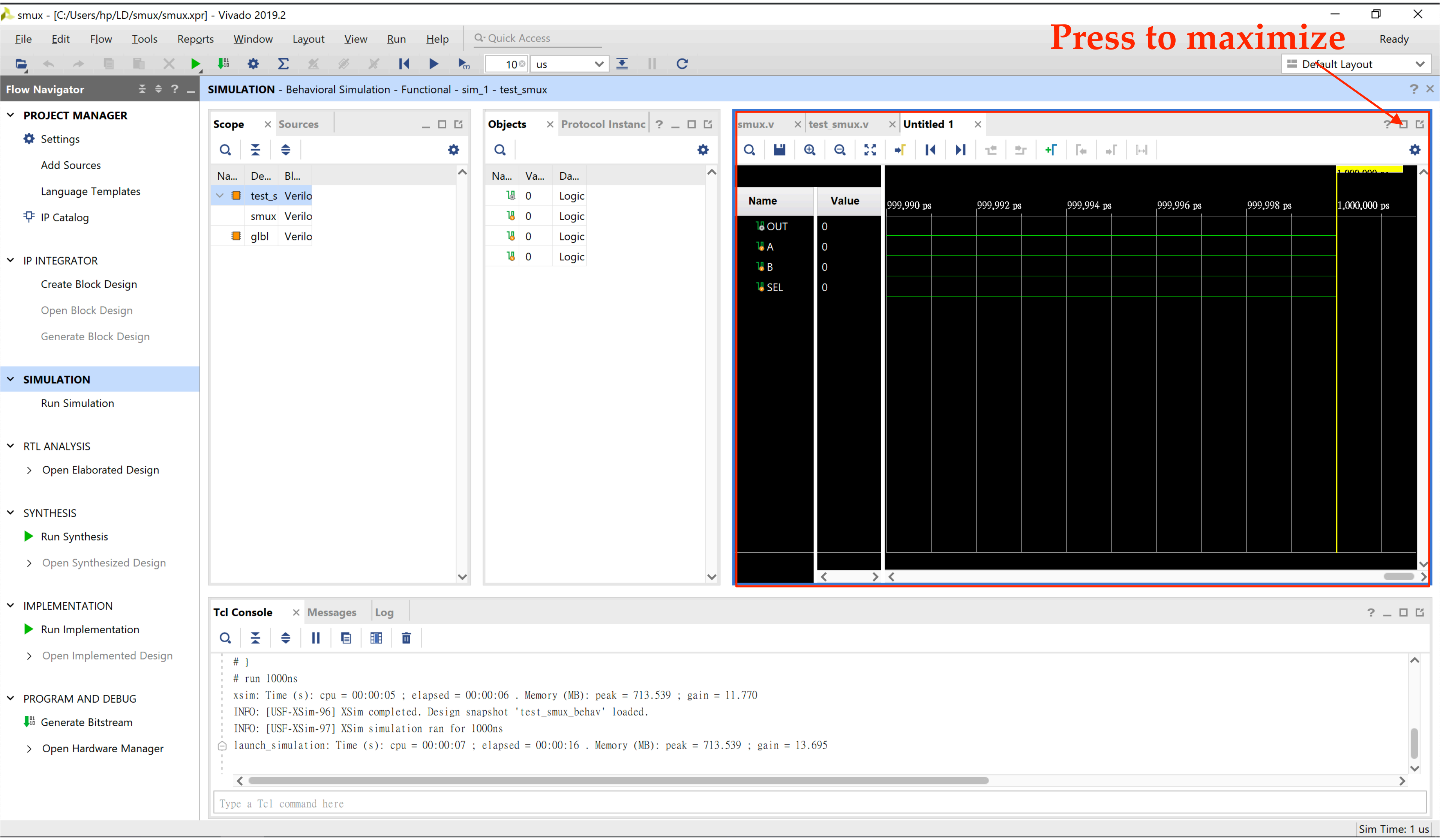
smux - [C:/Users/hp/LD/smux/smux.xpr] - Vivado 2019.2

C:/Users/hp/LD/smux/smux.srcs/sim_1/new/test_smux.v

```
module test_smux();
wire OUT;
reg A, B, SEL;
...
#10 A=1;B=0;SEL=1;
#10 A=1;B=1;SEL=0;
#10 A=1;B=1;SEL=1;
#10 A=0;B=0;SEL=0;
end
```

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy	Report Stra
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado Synthesis 2019)	Vivado Syn
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Vivado Implementation 2019)	Vivado Imp

Simulation (2/4)



Na...	De...	Bl...
test_s	Verilo	
smux	Verilo	
glbl	Verilo	

Na...	Va...	Da...
	0	Logic
	0	Logic
	0	Logic
	0	Logic

Name	Value
OUT	0
A	0
B	0
SEL	0

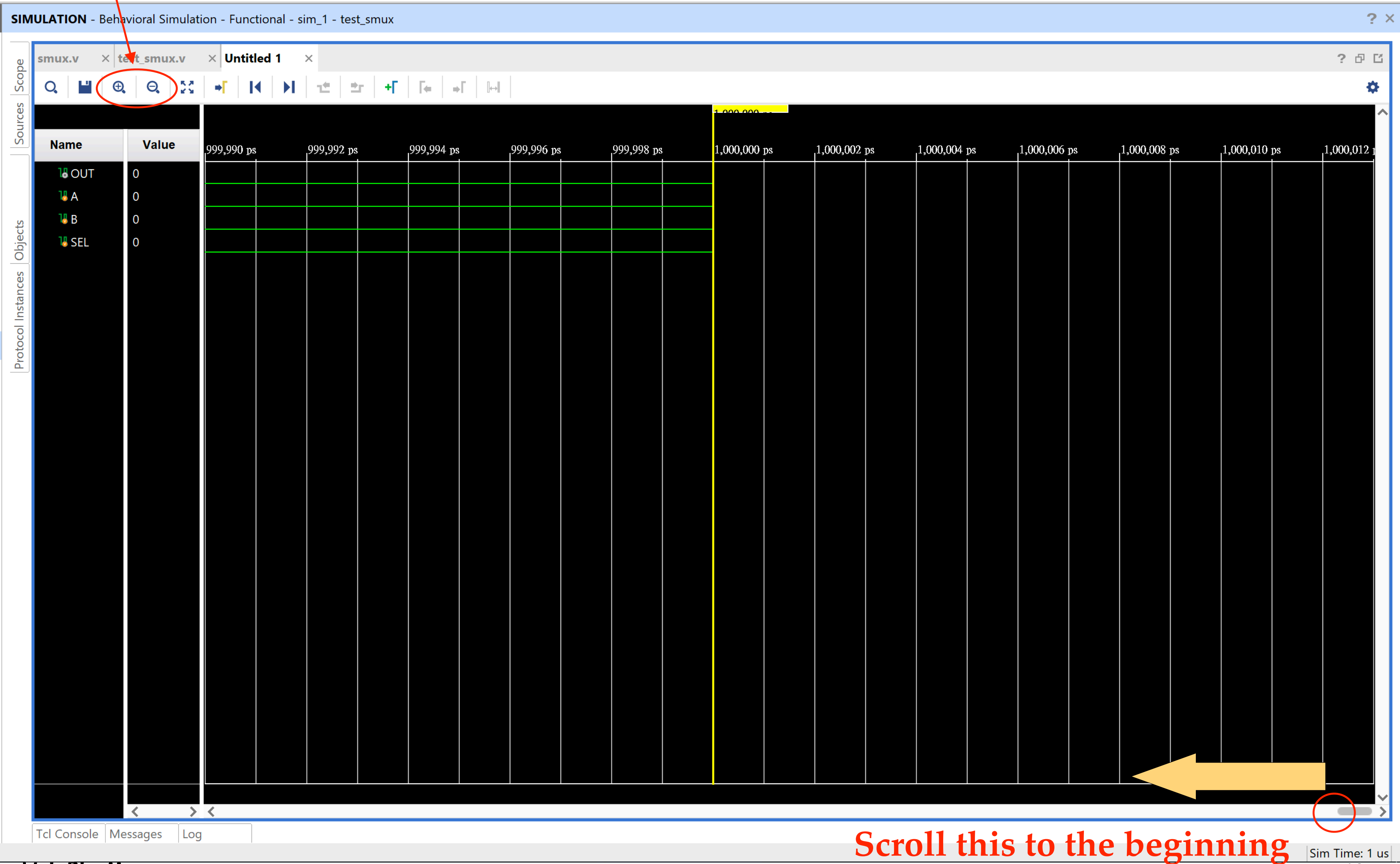
999,990 ps 999,992 ps 999,994 ps 999,996 ps 999,998 ps 1,000,000 ps

```
# }
# run 1000ns
xsim: Time (s): cpu = 00:00:05 ; elapsed = 00:00:06 . Memory (MB): peak = 713.539 ; gain = 11.770
INFO: [USF-XSim-96] XSim completed. Design snapshot 'test_smux_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:07 ; elapsed = 00:00:16 . Memory (MB): peak = 713.539 ; gain = 13.695
```

Type a Tcl command here

Simulation (3/4)

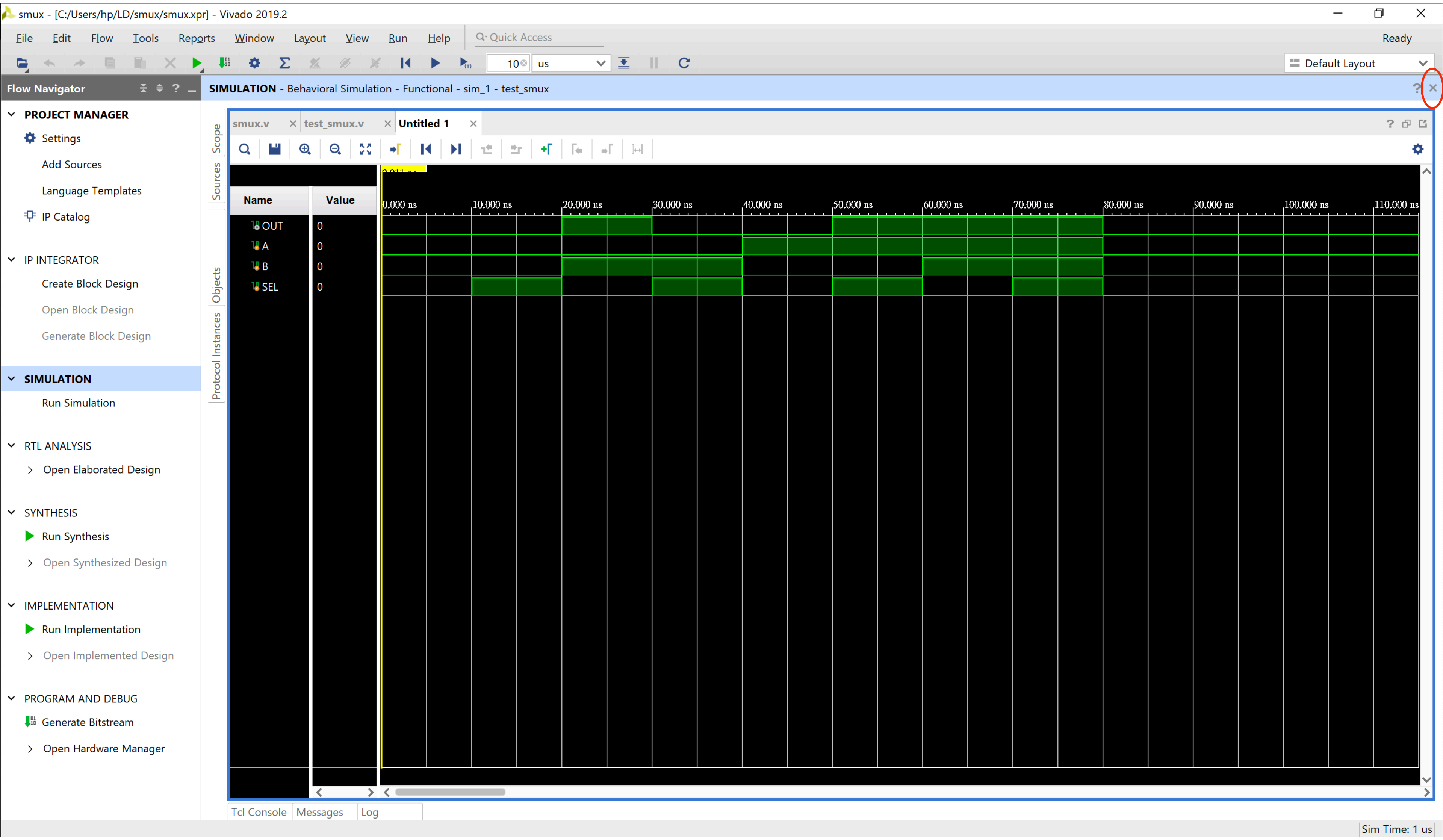
Use to zoom in or zoom out


SIMULATION - Behavioral Simulation - Functional - sim_1 - test_smux

Scroll this to the beginning

Simulation (4/4)

Press to close simulation



smux - [C:/Users/hp/LD/smux/smux.xpr] - Vivado 2019.2

SIMULATION - Behavioral Simulation - Functional - sim_1 - test_smux

Run time: 10 us

Name	Value
OUT	0
A	0
B	0
SEL	0

Sim Time: 1 us

Types of Verilog Construction

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 positive-edge-triggered D-type flip-flop with asynchronous reset

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

NTHU EE

Laboratory for Reliable Computing

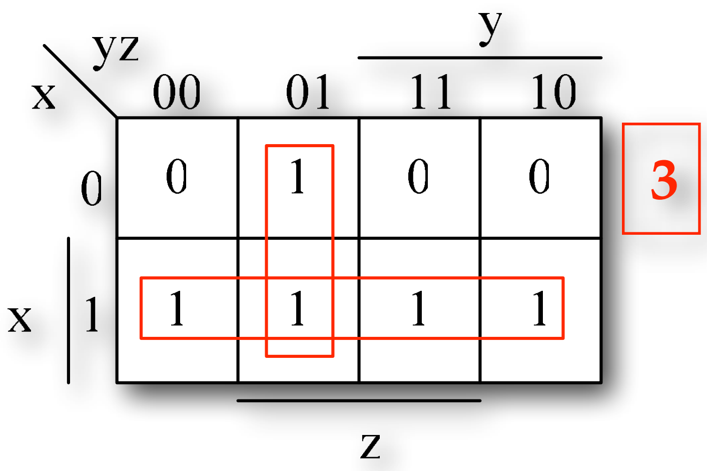
Some Combinational Logic Examples

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

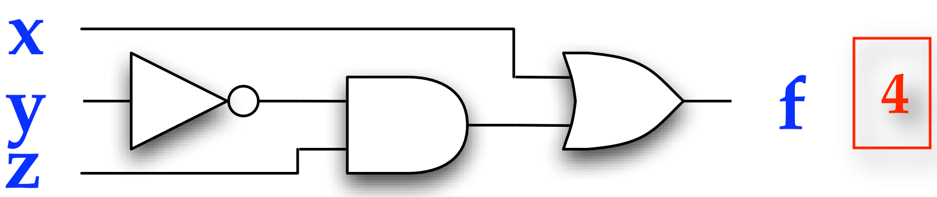
1 input: x,y,z output: f

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



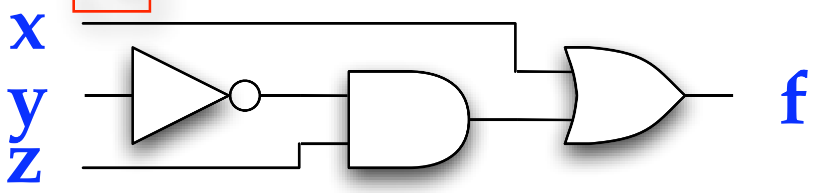
f=F(x,y,z)=x+y'z



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

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

4



5

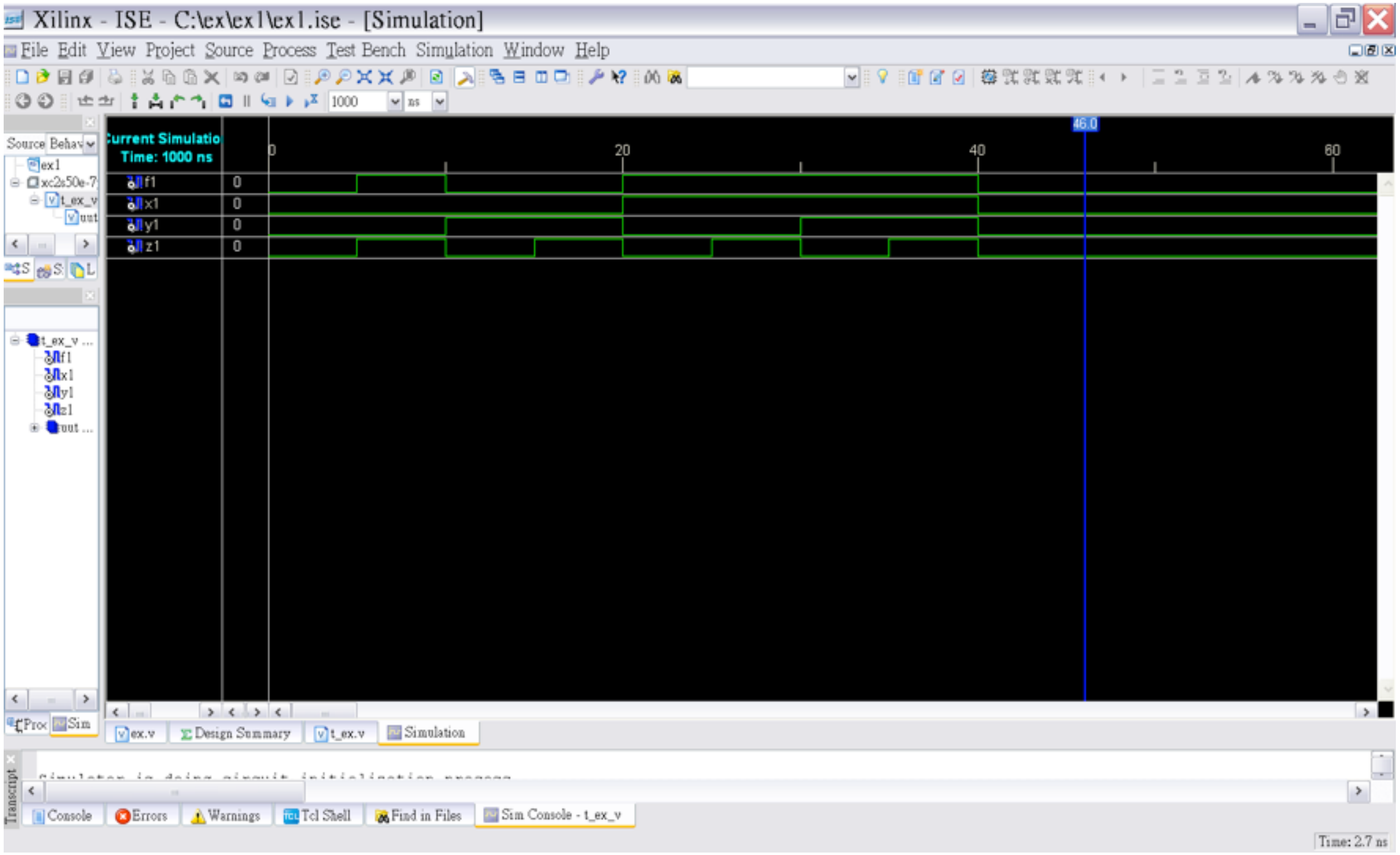
```
module ex(f,x,y,z);
output f;
input x,y,z;

  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
```

Laboratory for Reliable Computing

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



Xilinx - ISE - C:\ex\ex1\ex1.ise - [Simulation]
File Edit View Project Source Process Test Bench Simulation Window Help
Current Simulation Time: 1000 ns
f1 0
x1 0
y1 0
z1 0
0
20
40
60
46.0
ex.v
Design Summary
t_ex.v
Simulation
Console
Errors
Warnings
Tcl Shell
Find in Files
Sim Console - t_ex_v
Time: 2.7 ns

Simulator is doing circuit initialization process.

Example 2 (1/3)

$$\begin{aligned} f_1 &= x'y'z + xy'z' + xyz \\ &= m_1 + m_4 + m_7 = \sum(1,4,7) \end{aligned}$$

- Module definition

```
module exp1(f1, x, y, z);
output f1;
input x,y,z;

assign f1 = ((~x)&(~y)&z) | (x&(~y)&(~z) | x&y&z);

endmodule
```

x	y	z	f_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Example 2 (2/3)

- Testbench

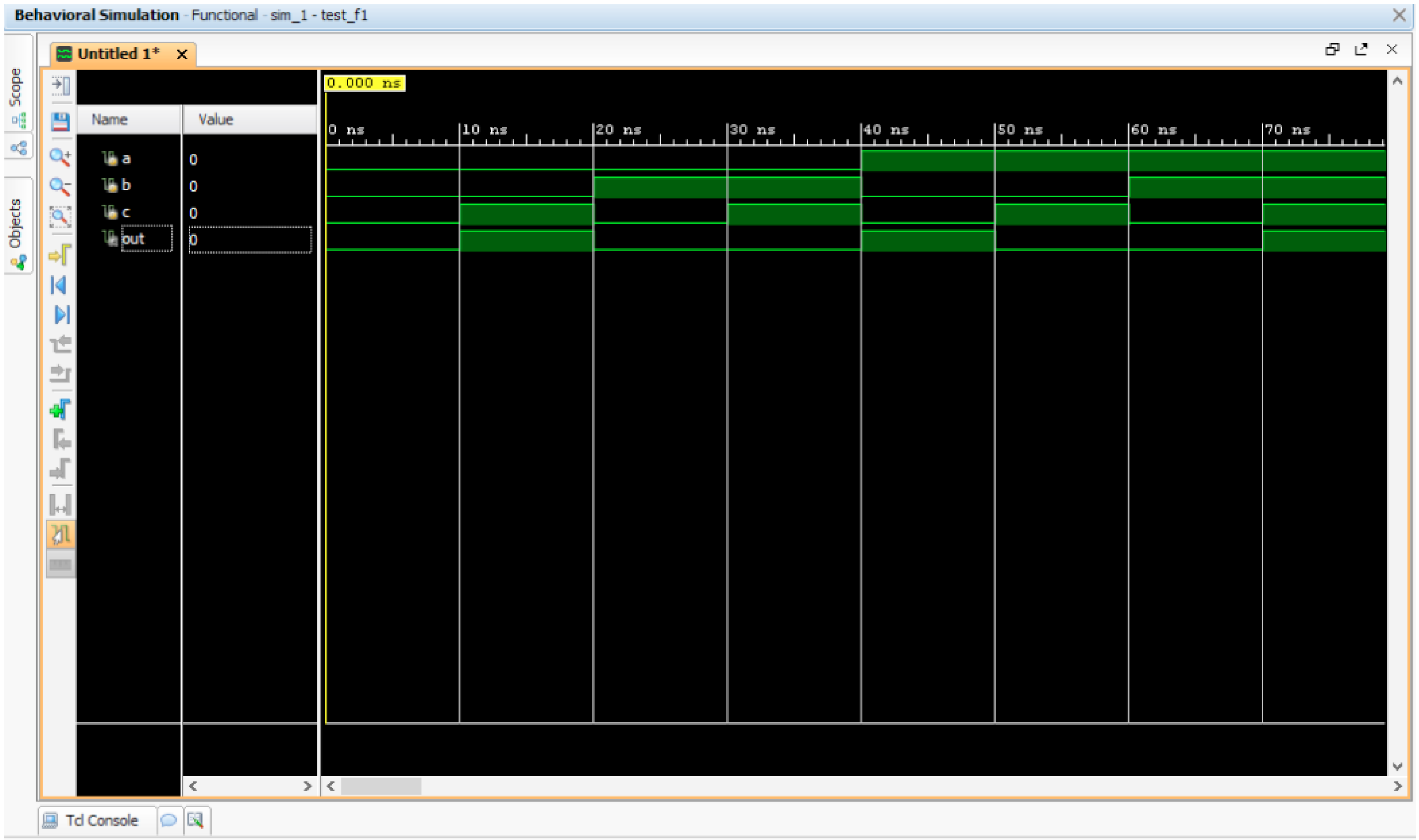
```
module test_f1;
wire out;
reg a,b,c;

exp1 U0(.f1(out),.x(a),.y(b),.z(c));

initial
begin
  a=0;b=0;c=0;
  #10 a=0;b=0;c=1;
  #10 a=0;b=1;c=0;
  #10 a=0;b=1;c=1;
  #10 a=1;b=0;c=0;
  #10 a=1;b=0;c=1;
  #10 a=1;b=1;c=0;
  #10 a=1;b=1;c=1;
end

endmodule
```

x	y	z	f_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Example 2 (3/3)

- Simulation Results

x	y	z	f_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



Example 3 (1/3)

$$\begin{aligned} f_2 &= x'yz + xy'z + xyz' + xyz \\ &= m_3 + m_5 + m_6 + m_7 = \sum(3,5,6,7) \end{aligned}$$

- Module definition

```
module exp2(f2, x, y, z);
output f2;
input x,y,z;

assign f2 = ((~x)&y&z) | (x&(~y)&z) |( x&y&(~z))| (x&y&z);

endmodule
```

x	y	z	f_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Example 3 (2/3)

- Testbench

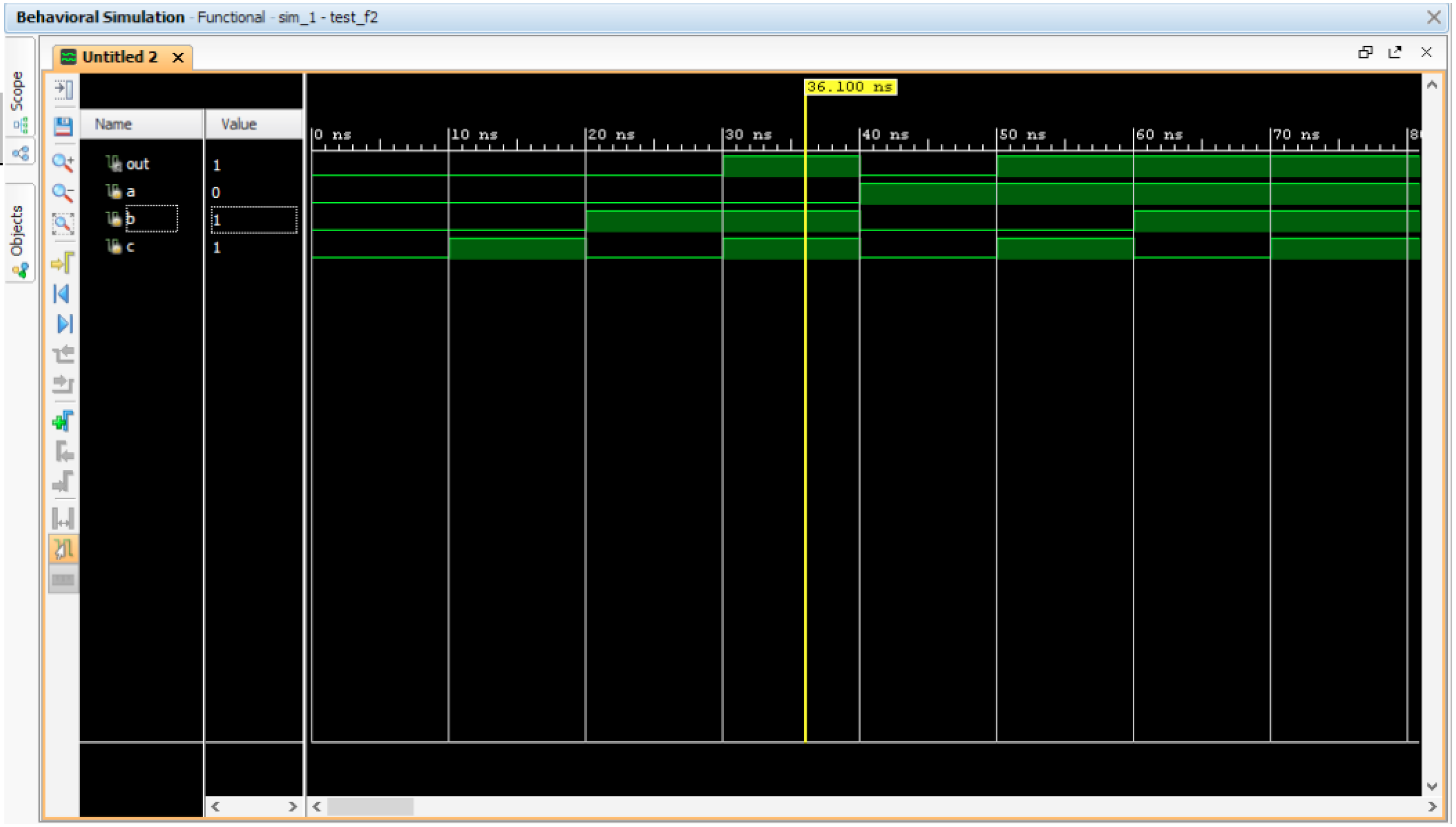
```
module test_f2;
wire out;
reg a,b,c;

exp2 U0(.f2(out),.x(a),.y(b),.z(c));

initial
begin
  a=0;b=0;c=0;
  #10 a=0;b=0;c=1;
  #10 a=0;b=1;c=0;
  #10 a=0;b=1;c=1;
  #10 a=1;b=0;c=0;
  #10 a=1;b=0;c=1;
  #10 a=1;b=1;c=0;
  #10 a=1;b=1;c=1;
end

endmodule
```

x	y	z	f_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Example 3 (3/3)

- Simulation Results

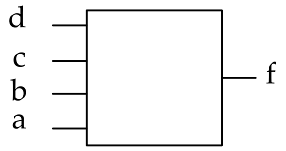
x	y	z	f_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Name	Value
out	1
a	0
b	1
c	1

Example 4: Prime Detector (1/2)

- Spec 1
  - A circuit outputs a 1 when its four-bit input represents a prime number in binary
  - f(d,c,b,a)=1 if $dcba_2$ is a prime



- Truth table

2

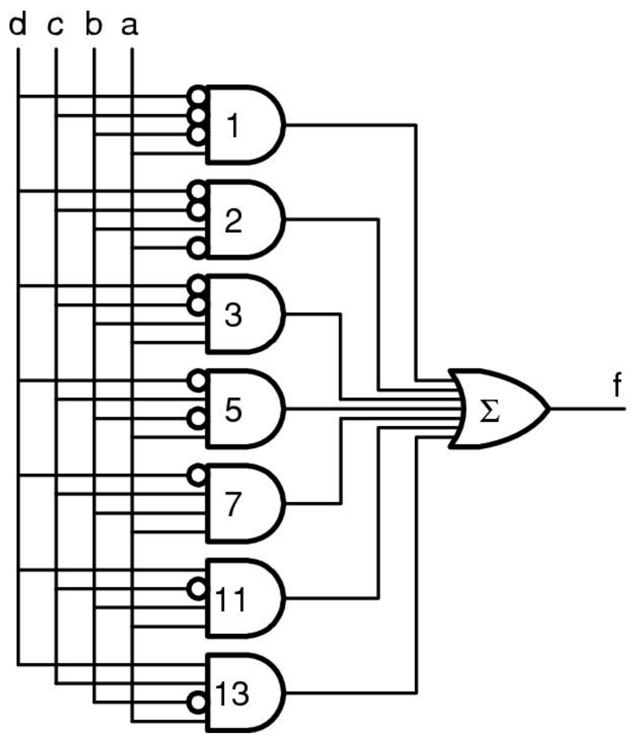
No	dcba	f
0	0000	0
1	0001	1
2	0010	1
3	0011	1
4	0100	0
5	0101	1
6	0110	0
7	0111	1
8	1000	0
9	1001	0
10	1010	0
11	1011	1
12	1100	0
13	1101	1
14	1110	0
15	1111	0

abbreviated truth table

No	dcba	f
1	0001	1
2	0010	1
3	0011	1
5	0101	1
7	0111	1
11	1011	1
13	1101	1
Otherwise		0

Example 4: Prime Detector (2/2)

- Normal form 2
  - $f(d,c,b,a) = d'c'b'a + d'c'ba' + d'c'ba + d'cb'a + d'cba + dc'ba + dcb'a$
- Sum of minterms
  - $f(d,c,b,a) = \sum m(1,2,3,5,7,11,13)$



Verilog Module v1 (assign)

$$f(d,c,b,a) = d'a + d'c'b + cb'a + c'ba$$

```
module prime(isprime, a, b, c, d);
output isprime;
input a,b,c,d;

assign isprime= ((~d)&a) | ((~d)&(~c)&b) | (c&(~b)&a) | ((~c)&b&a));

endmodule
```

Verilog Module v2 (case)

let in[3:0]=dcba (bus)

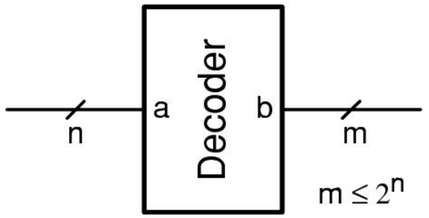
- Use "case"

```
module prime(isprime, in);
output isprime;  //true if input is prime
input [3:0] in;  //4-bit input
reg isprime;  //for signal to be assigned in always block

always @*
  case (in)
    1,2,3,5,7,11,13: isprime = 1'b1;
    default: isprime = 1'b0;
  endcase

endmodule
```

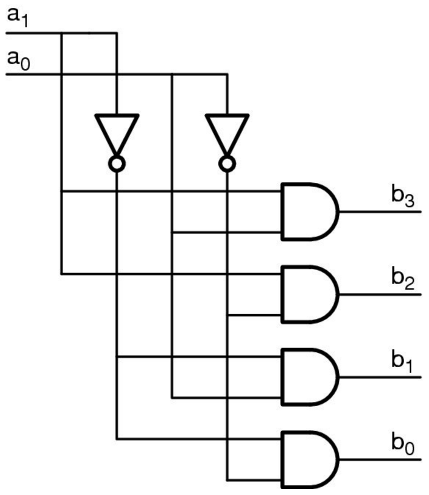
Testbench

```
module test_prime;
wire isprime;
reg [3:0] in;

/ / instantiate module to test
prime U0(.isprime(isprime),.in(in));

initial
begin
  in=0;  // set to initial value
  repeat (16)  // loop 16 times
  begin
    #100    // delay for 100 time unit
    $display(“in = %2d isprime = %1b”, in, isprime);
    in = in +1;  // increment input
  end
 end
endmodule
```

A 2-to-4 Line Decoder



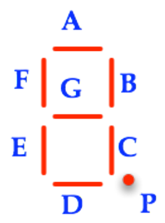
a_1	a_0	b_3	b_2	b_1	b_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



```
module Dec24(a, b);
input [1:0] a; //binary input (2 bits wide)
output [3:0] b;  // one-hot output (4 bits wide)

assign b[3]=a[1]&a[0];
assign b[2]=a[1]&(~a[0]);
assign b[1]=(~a[1])&a[0];
assign b[0]=(~a[1])&(~a[0]);

endmodule
```

$$
\begin{aligned}
b_3 &= a_1 a_0 \\
b_2 &= a_1 a_0' \\
b_1 &= a_1' a_0 \\
b_0 &= a_1' a_0'
\end{aligned}
$$

A BCD to Seven-Segment Display Decoder



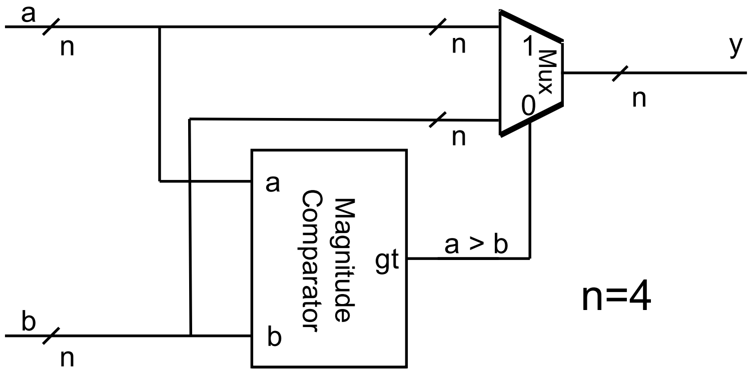
```
// define segment codes
`define SS_0 7'b1111110
`define SS_1 7'b0110000
`define SS_2 7'b1101101
`define SS_3 7'b1111001
`define SS_4 7'b0110011
`define SS_5 7'b1011011
`define SS_6 7'b1011111
`define SS_7 7'b1110000
`define SS_8 7'b1111111
`define SS_9 7'b1111011
```

```
module sseg(segs, bin);
output [6:0] segs;
input [3:0] bin;
reg [6:0] segs;

always @*
 case (bin)
   4'd0: segs = `SS_0;
   4'd1: segs = `SS_1;
   4'd2: segs = `SS_2;
   4'd3: segs = `SS_3;
   4'd4: segs = `SS_4;
   4'd5: segs = `SS_5;
   4'd6: segs = `SS_6;
   4'd7: segs = `SS_7;
   4'd8: segs = `SS_8;
   4'd9: segs = `SS_9;
   default: segs = 7'b0000000;
 endcase
endmodule
```

NTHU EE Laboratory for Reliable Computing

Maximun Unit

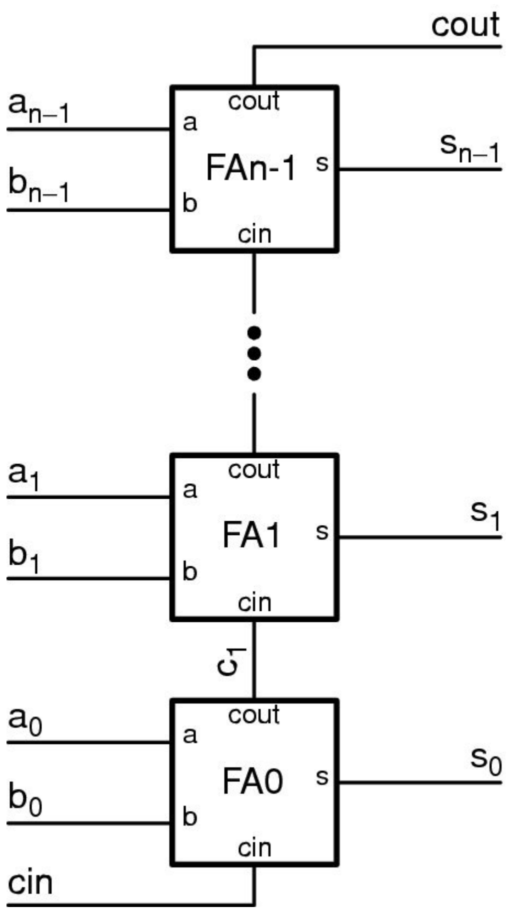
$$y = \max\{a, b\}$$



```
module Max(a, b, y);
input [3:0] a, b; //two input variables (4 bits wide)
output reg [3:0] y;  // maximum output (4 bits wide)

assign gt=(a>b) ? 1 : 0;
always @*
 if (gt)
   y = a;
 else
   y = b;

endmodule
```

```
always @*
 if (a > b)
   y = a;
 else
   y = b;
```

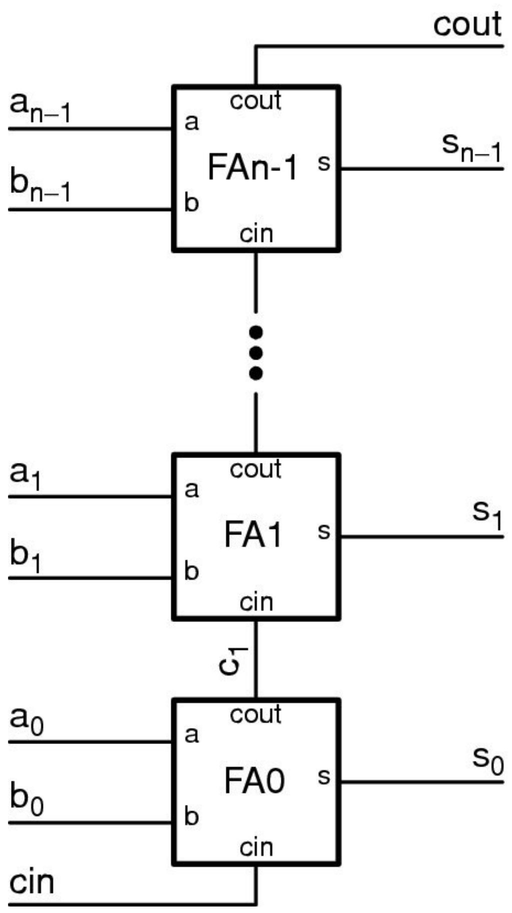
Ripple-Carry Adder (1/2)



```
module adder(a, b, cin, cout, s);
parameter n=4;
input [n-1:0] a, b;
input cin;
output reg [n-1:0] s;
output cout;
reg [n-1:0] c;
integer i;

assign cout = c[n];
always @*
  c[0] = cin;
always @*
  for (i=0;i<n;i=i+1)
    {c[i+1],s[i]} = a[i] + b[i] + c[i];

endmodule
```

```
assign {c[1],s[0]} = a[0] + b[0] + c[0];
assign {c[2],s[1]} = a[1] + b[1] + c[1];
assign {c[3],s[2]} = a[2] + b[2] + c[2];
assign {c[4],s[3]} = a[3] + b[3] + c[3];
```

Multi-bit Binary Adder (2/2)



```
module Adder1(a, b, cin, cout, s);
parameter n=8;
input [n-1:0] a, b;
input cin;
output [n-1:0] s;
output cout;

assign {cout, s} = a + b + cin;

endmodule
```