Verilog HDL – I

黃元豪
Yuan-Hao Huang

國立清華大學電機工程學系
Department of Electrical Engineering
National Tsing-Hua University

Outline

- Introduction
- Sample Design
- Structural Modeling
- RTL Modeling
- Logic Modeling and Simulation Using Xilinx ISE
- An Example of Combinational Circuits

Introduction

Verilog HDL

- Verilog 硬體描述語言(Verilog Hardware Description Language)
  - 在積體電路設計（特別是超大型積體電路的計算機輔助設計）的電子設計自動化領域中，**Verilog HDL**是一種用於描述、設計電子系統（特別是數位電路的硬體描述語言）。
  - Verilog是電力電子工程師學會（IEEE）的1364號標準。

- Verilog 硬體描述語言在邏輯設計上的用途
  - 用途一: 邏輯電路設計的模擬與驗證
  - 用途二: FPGA邏輯電路(**IC電路)的設計與實作

    ** IC電路的設計與實作在【EE4292積體電路設計實驗】教授。

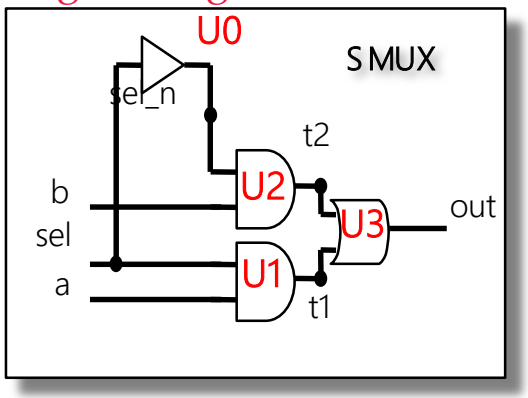
Verilog HDL Utilization Scenario I

- Simulation and verification of logic circuits on PC.

Specification

2-to-1 Multiplexer

Logic Design

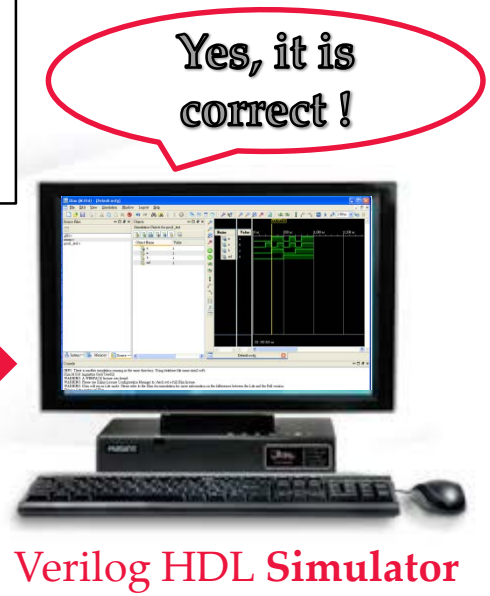


Verilog HDL Coding

```
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
```

Test Pattern

a, b, sel
000
001
010
…
111



Verilog HDL **Simulator**

- Imaging what happens when the circuit is as complex as a CPU or MP3 player processor.

Verilog HDL Utilization Scenario II

- Design and Implementation of logic circuits in FPGA (IC)

Specification

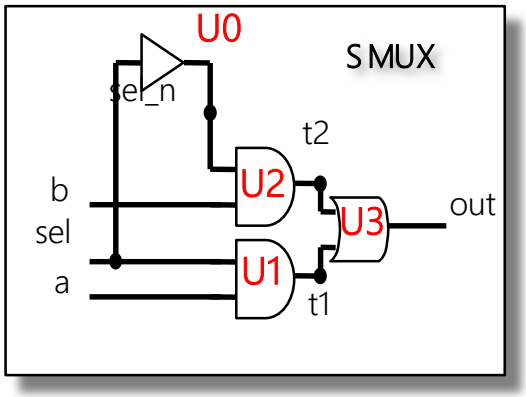
2-to-1 Multiplexer

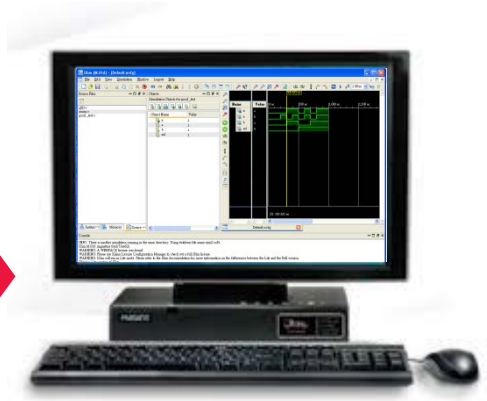
Verilog HDL Coding

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

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

 assign out = sel ? a:b;

endmodule
```

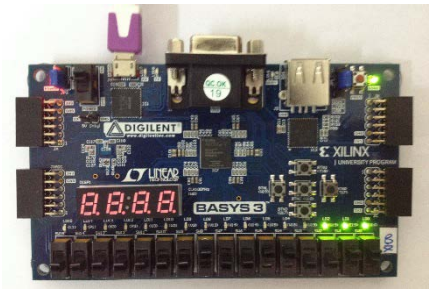
Verilog HDL Logic **Synthesizer**

Synthesized Logic Circuits





FPGA **Implement Design** and **Programming**

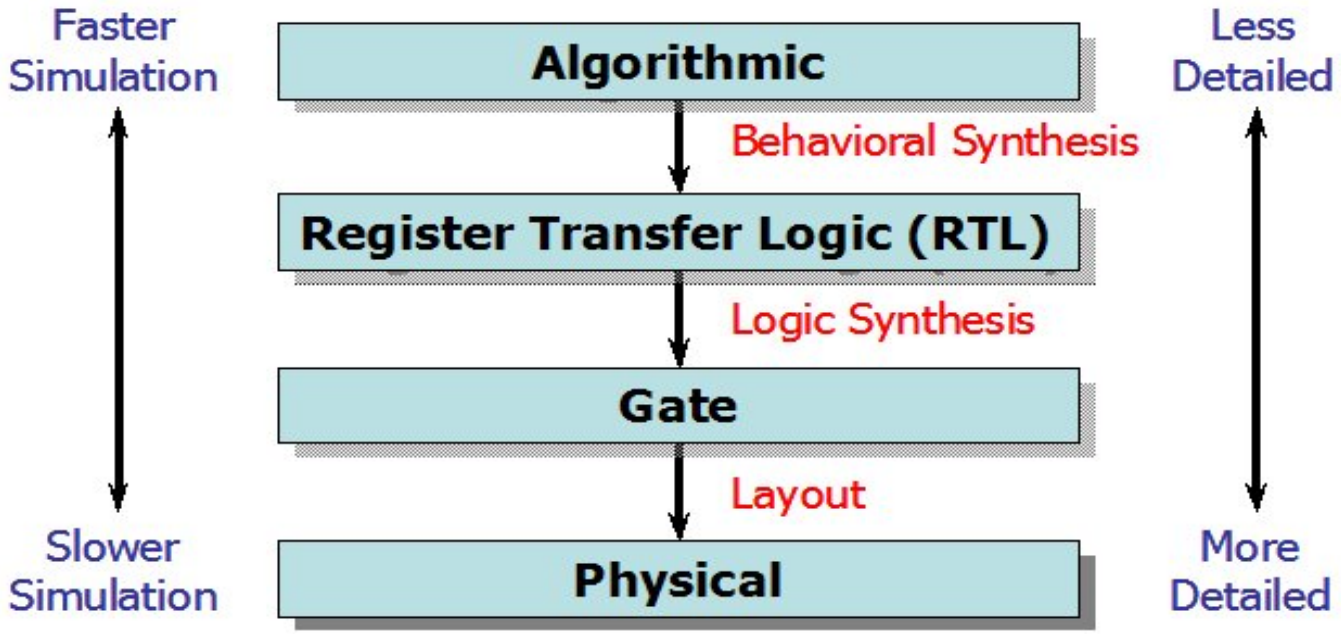


Now, it can work !

Verilog HDL – 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)
  - 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 logic gates and the interconnections between them
- Transistor/Switch/Physical Level
  - A model that describes the transistors and the interconnections between them

Verilog HDL – Levels of Abstraction (1/2)



Level of Abstraction Example

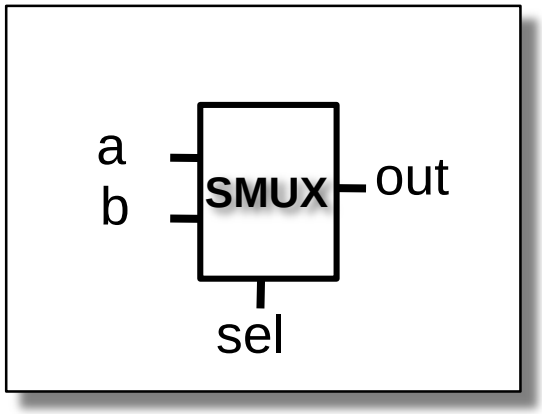
- Describe the operation of a circuit at various levels of abstraction (MP3 Player Decoder as example)
  - Behavior (MP3 Decoding, C/C++, HDL)
  - Function (Filtering, Fourier Transformation, C/C++, HDL)
  - Structure (Adder/Subtractor, Multiplier, Divider, HDL)

- Compared with C/C++ Program, verilog HDL
  - describes the timing (delay) of a circuit
  - expresses the concurrency (parallelism) of circuit operation

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 test-bench
  - Synthesis tools accept only a limited subset of behavior level description
    - 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
```

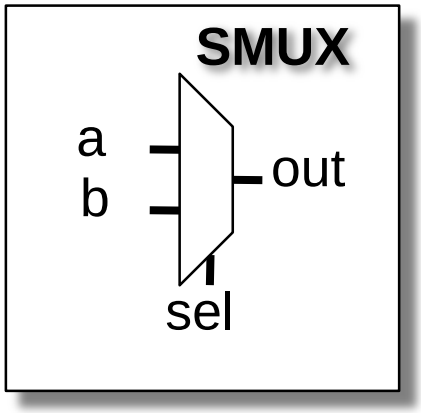


Behavior Level Abstraction

- Case 2: always @ (input1 or input2 or …)

```
begin
    out1=
end
```

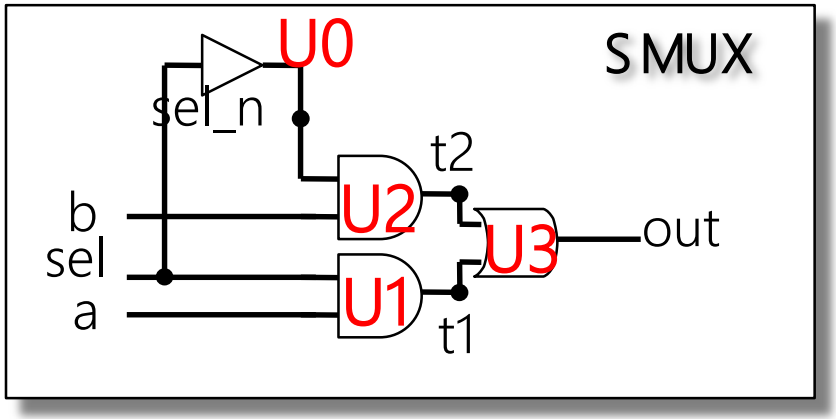
```
module SMUX(out, a, b, sel);

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

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



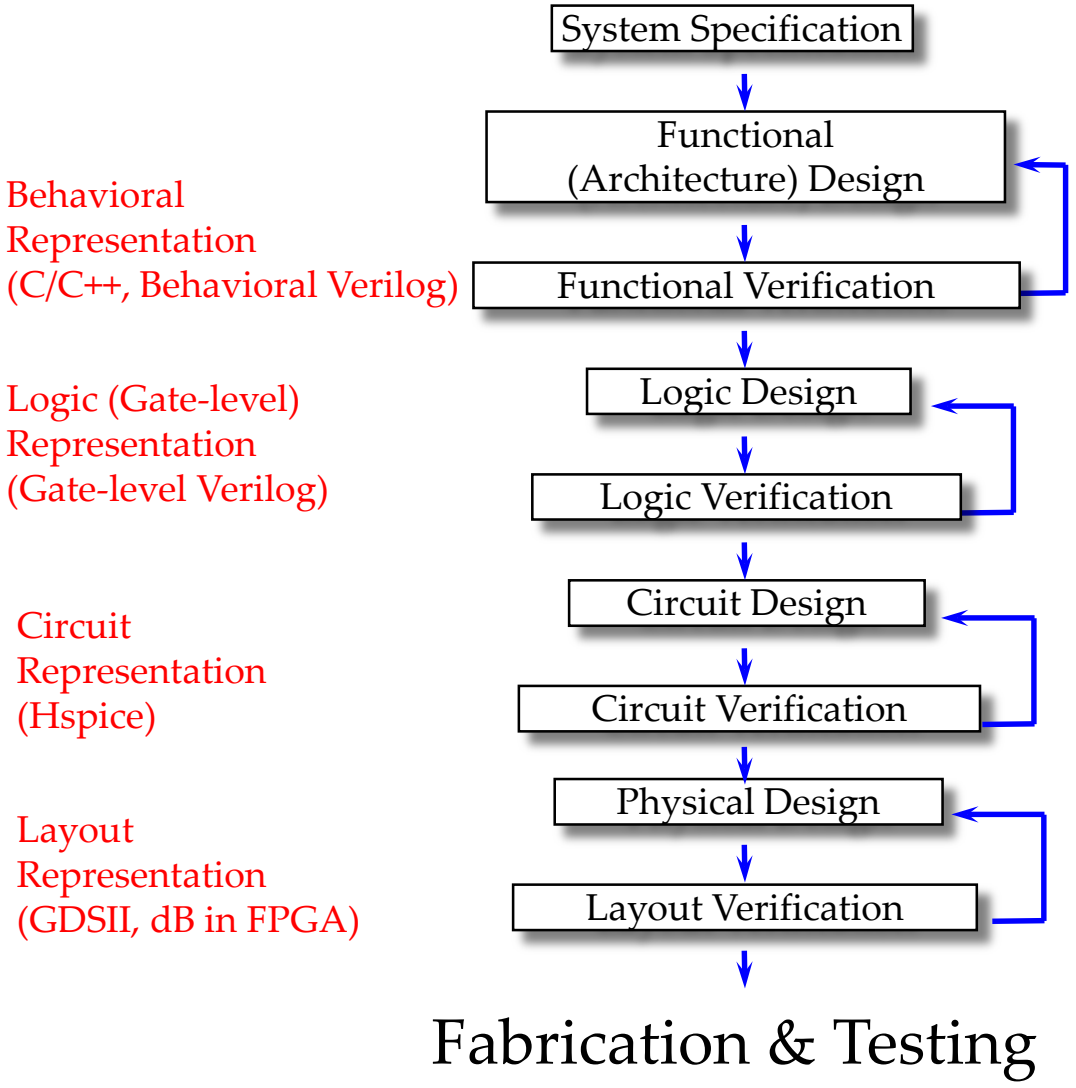
Gate Level Abstraction

- Gate-level abstraction describes a pure structural design of circuits.
  - You must derive and draw the circuit schematics first before writing Verilog code.

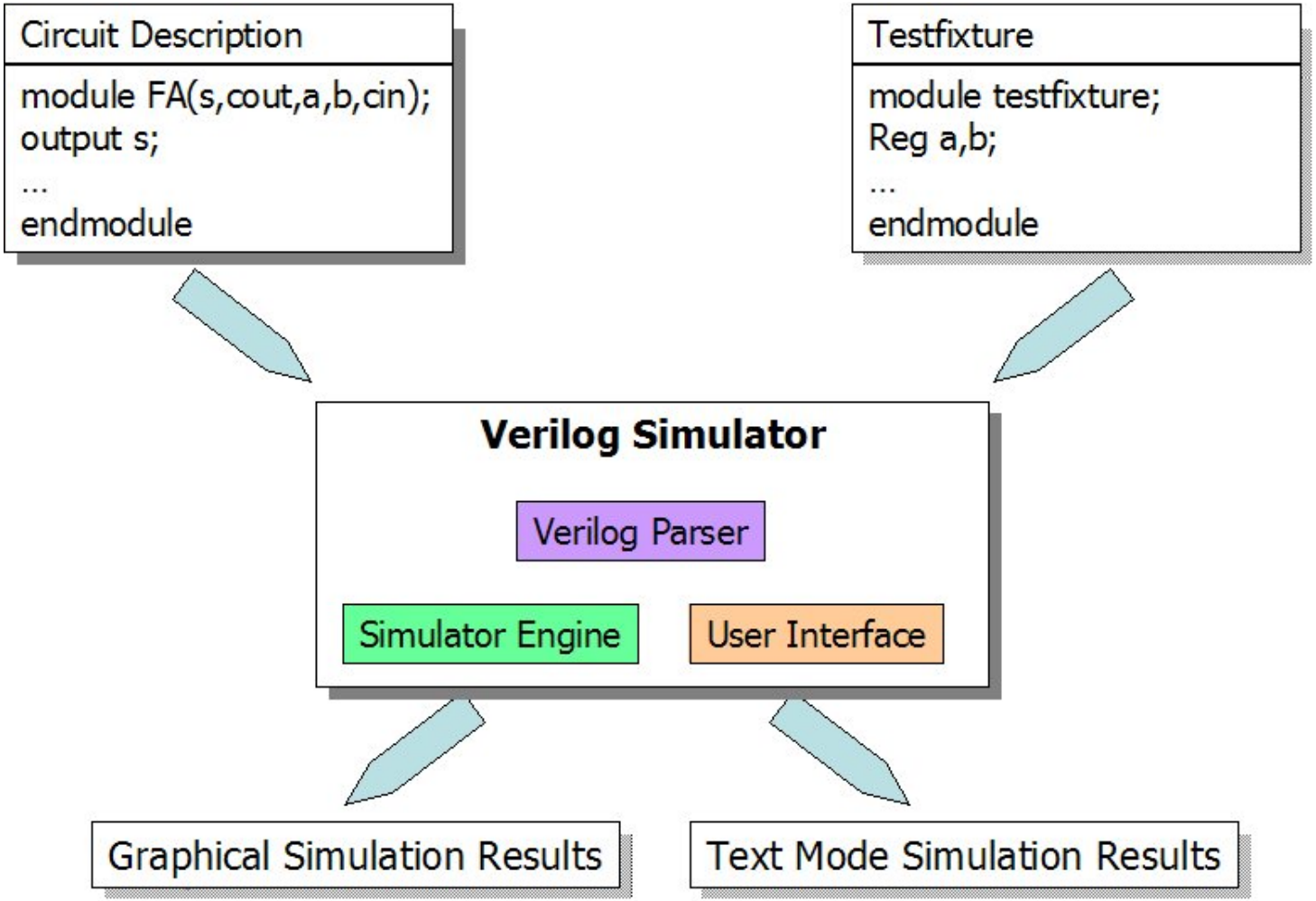
```
module SMUX(out, a, b, sel);

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

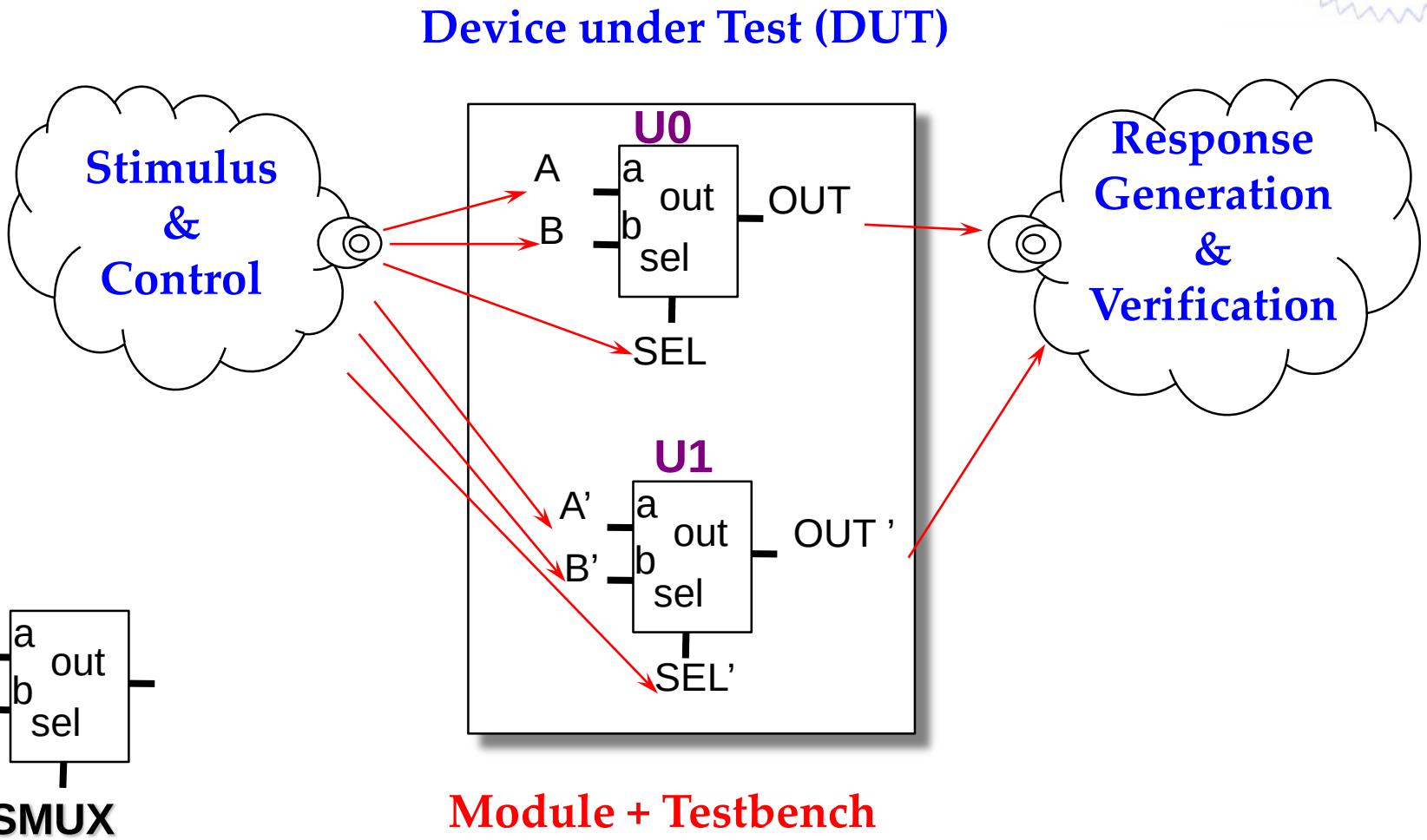


Verilog Simulation



Sample Design

Scenario


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

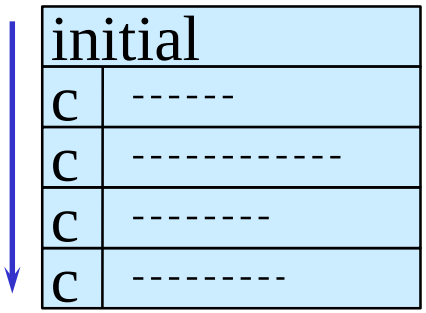
- module testfixture;
  - ➢ Declare signals
  - ➢ Instantiate modules
  - ➢ Applying stimulus
  - ➢ Monitor signals
- endmodule

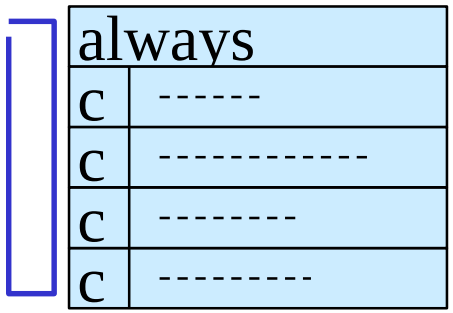
Testbench (2/4)

- 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

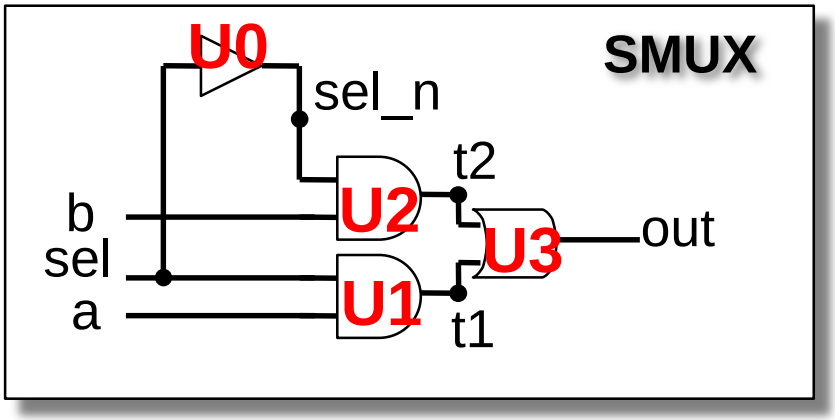
Structural Modeling

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

*** Synthesis not supported : The divisor for divide operator may be restricted to constants and a power of 2**

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)

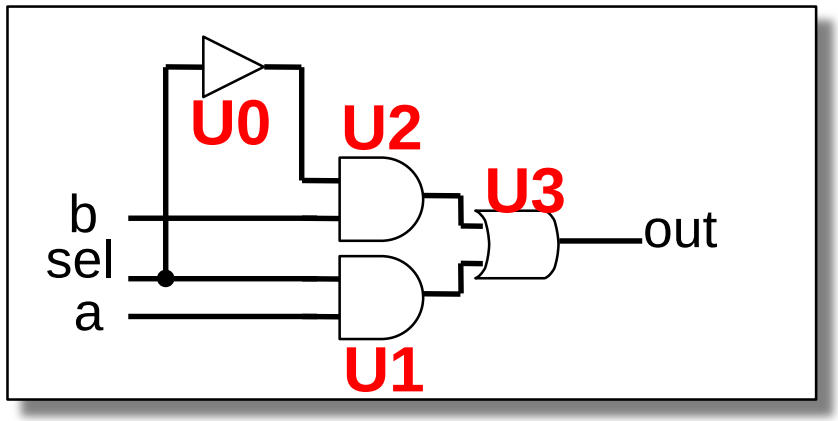
Logical Shift Operators		
OP	Usage	Description
<<	m << n	Shift m left n-times
>>	m >> n	Shift m right n-times

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
```



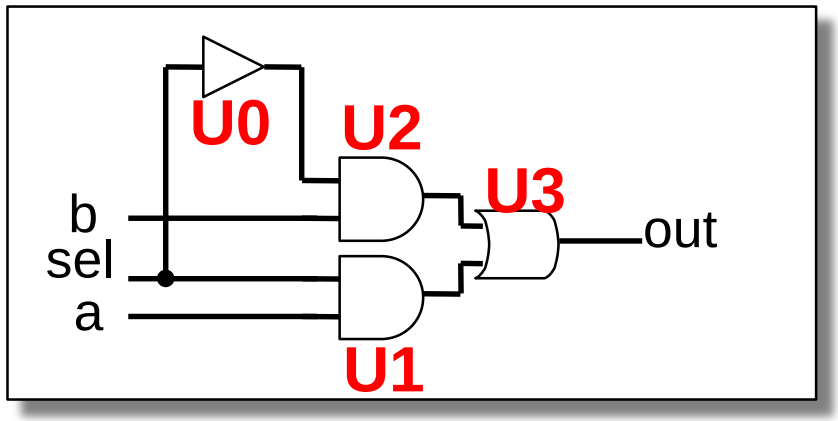
This out has to be declared as “wire” or “output” data type.
This expression can not be inside always @().

always

- **always** statements

```
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
```

sensitivity list

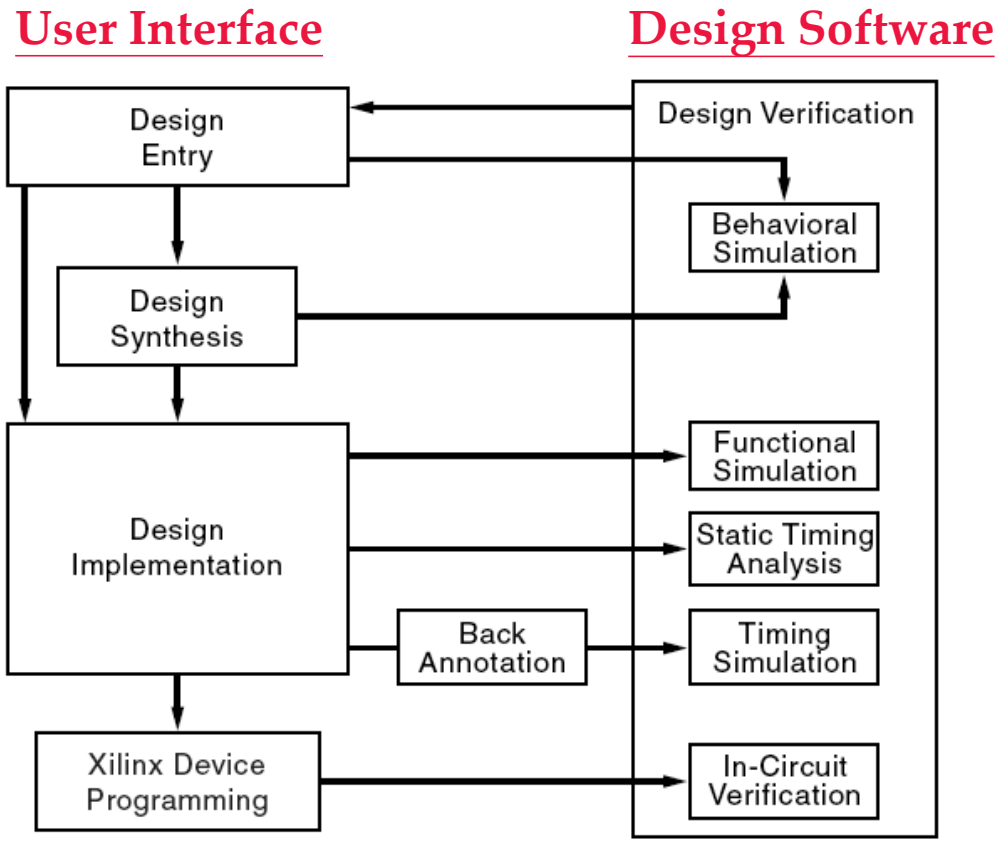


This out has to be declared as "reg" data type.

Logic Modeling and Simulation Using Xilinx Vivado

Design Flow

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

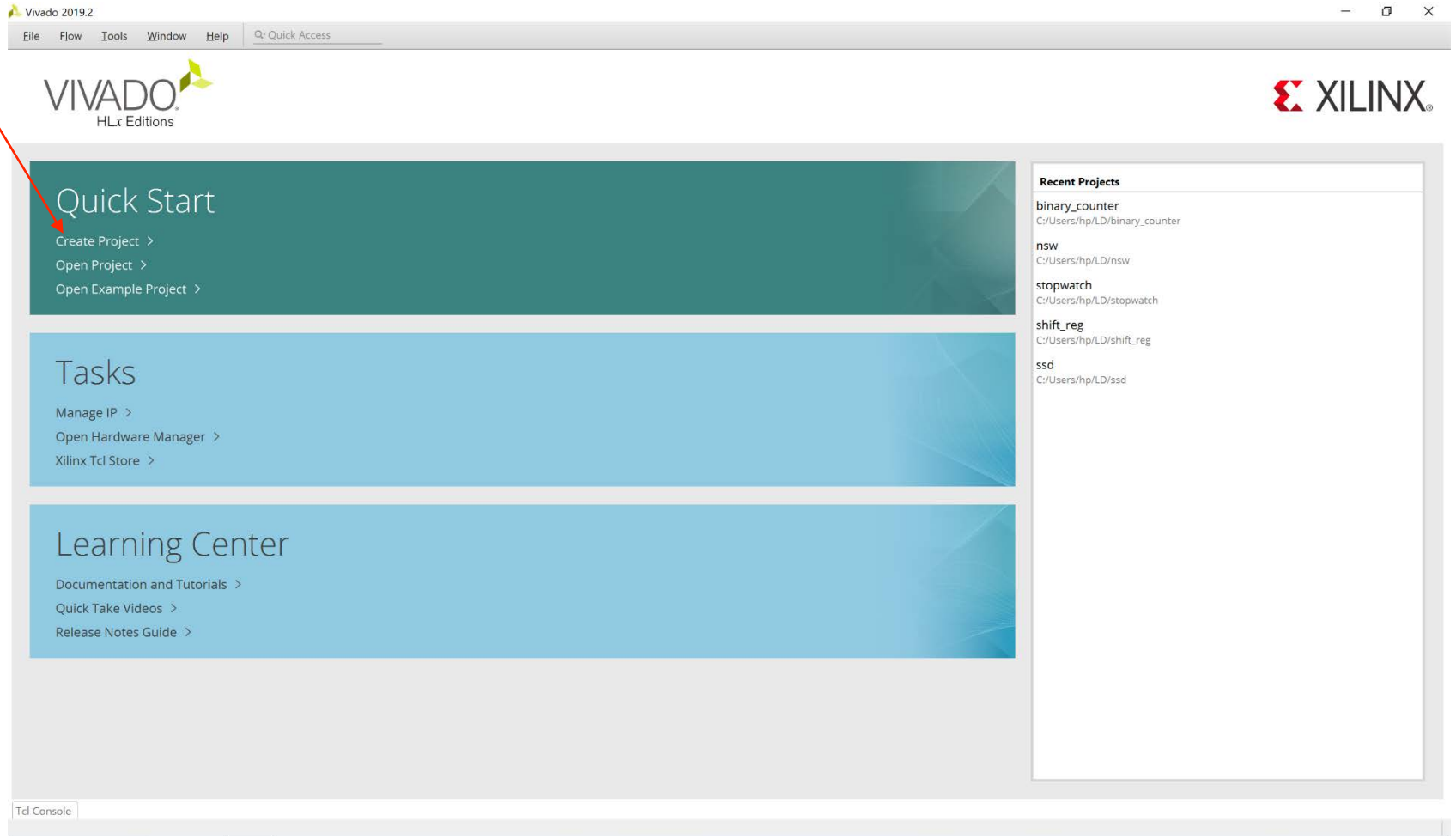


Important Notes

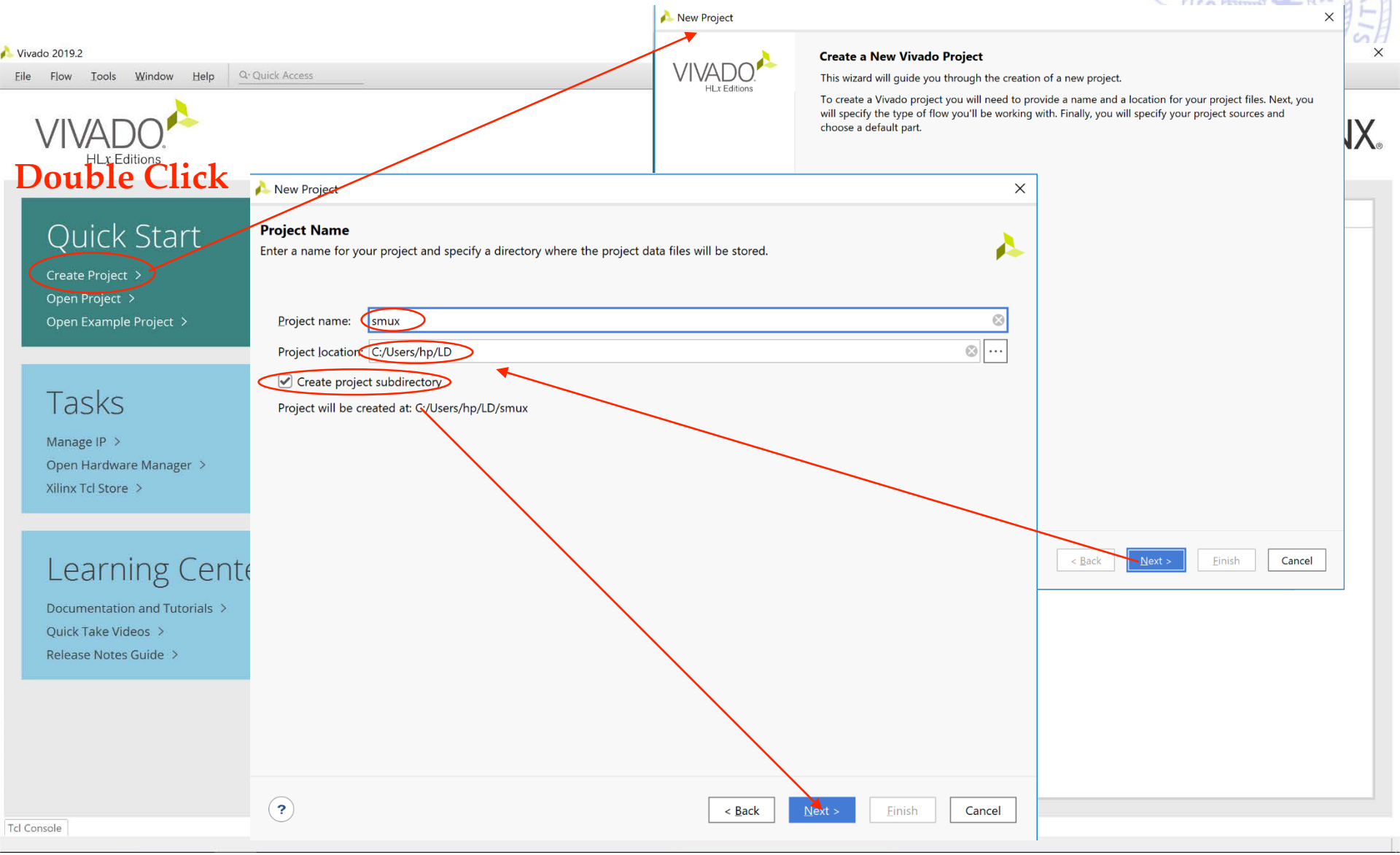
- **Draw schematic first** and then construct Verilog codes.
- Verilog RTL coding philosophy is not the same as C programming
  - Every Verilog RTL construct has its own logic mapping (for synthesis)
  - You should have the logics (draw schematic) first and then the RTL codes
  - You have to write **synthesizable**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

Open Vivado

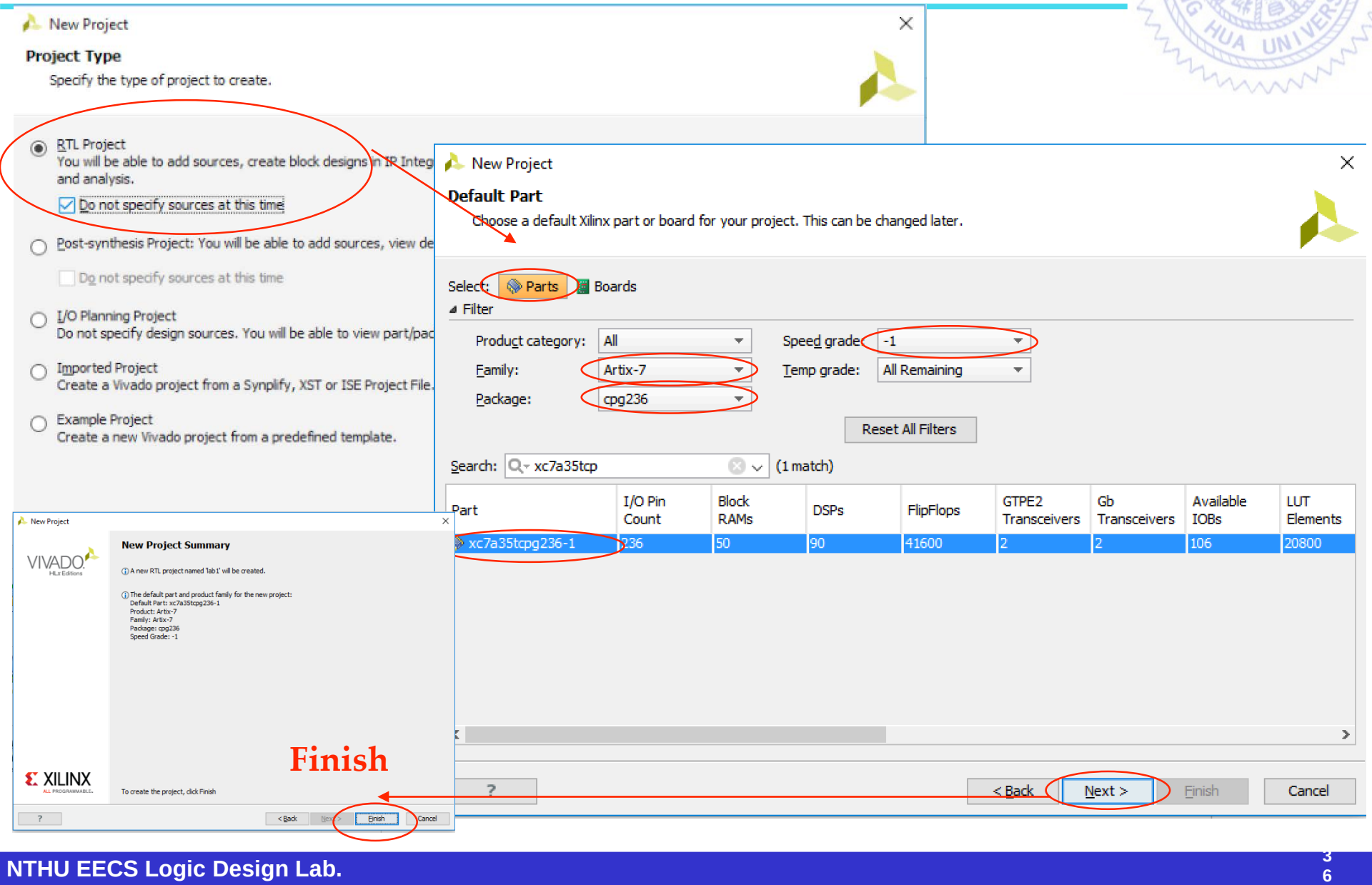

Vivado


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

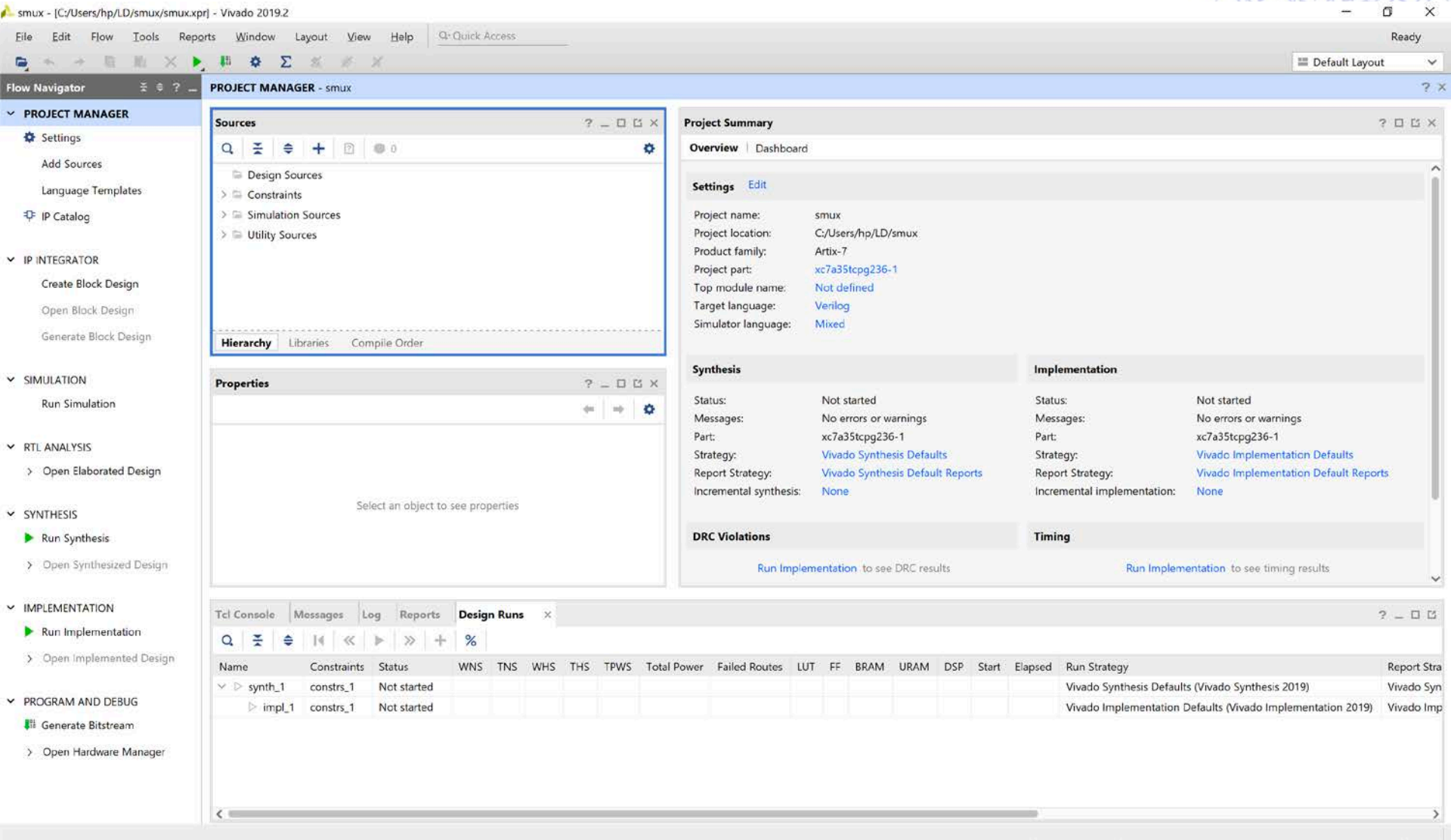
Open New Project (1/3)



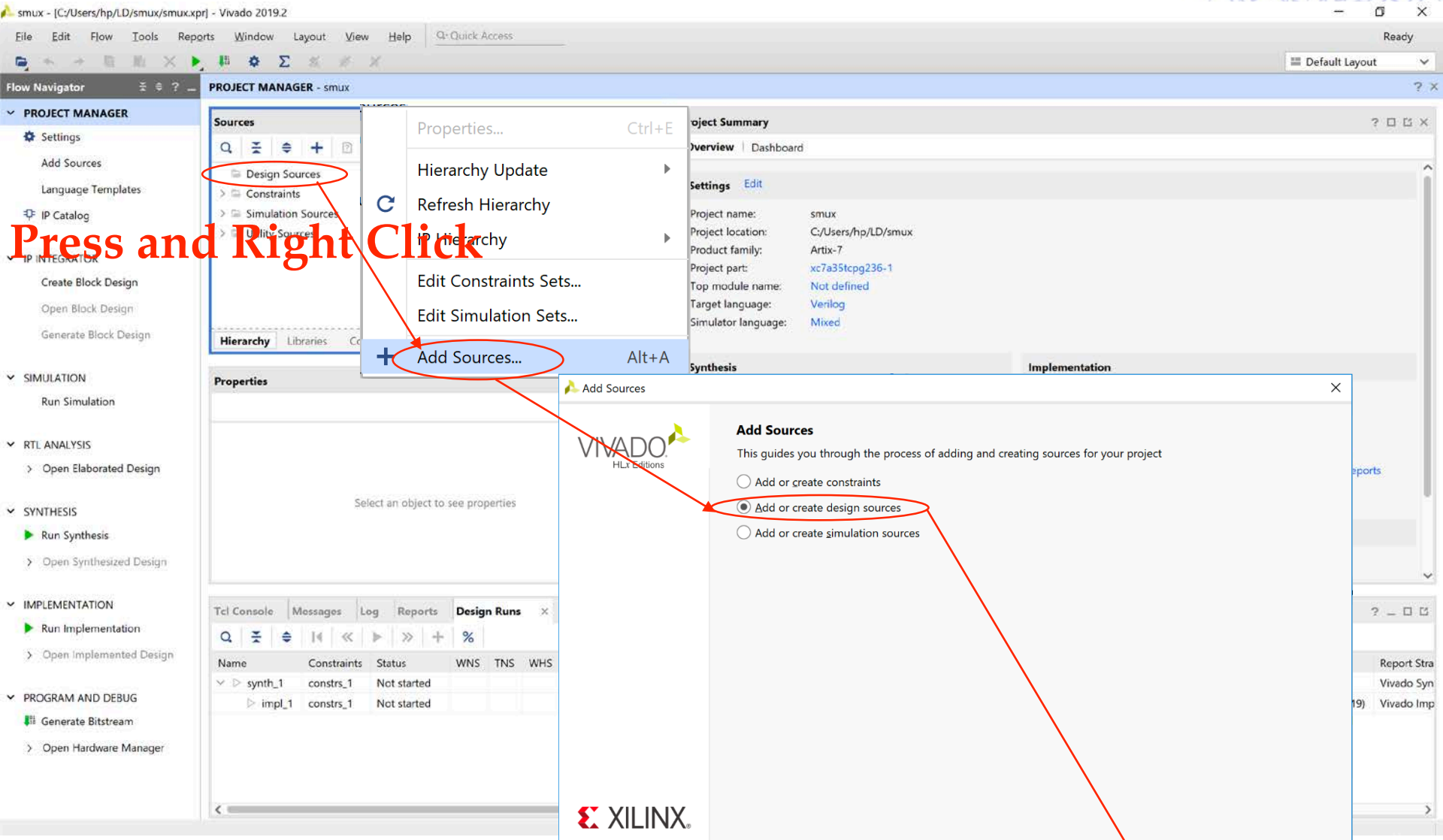
Open New Project (2/3)



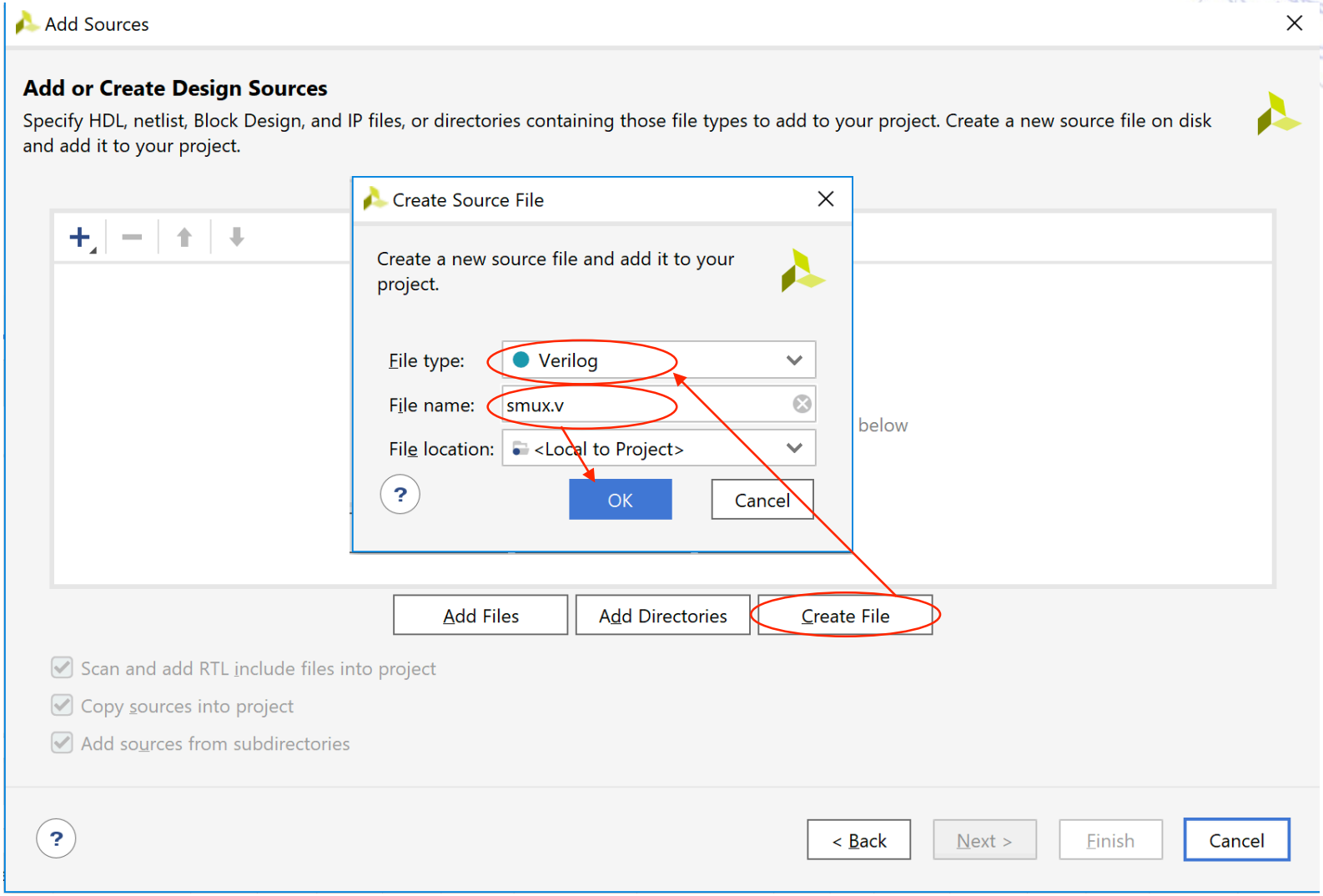
Open New Project (3/3)



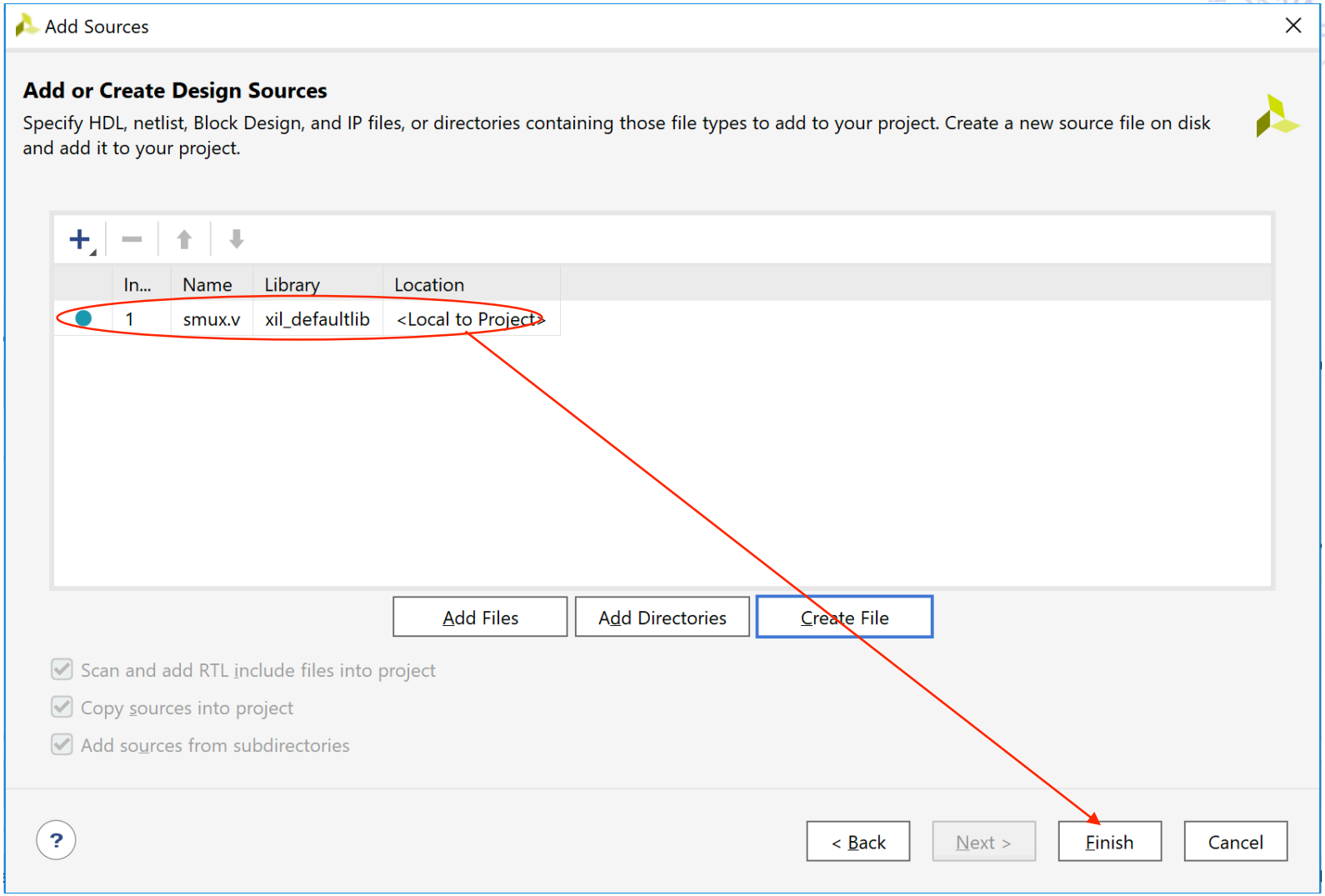
New Source (1/5)



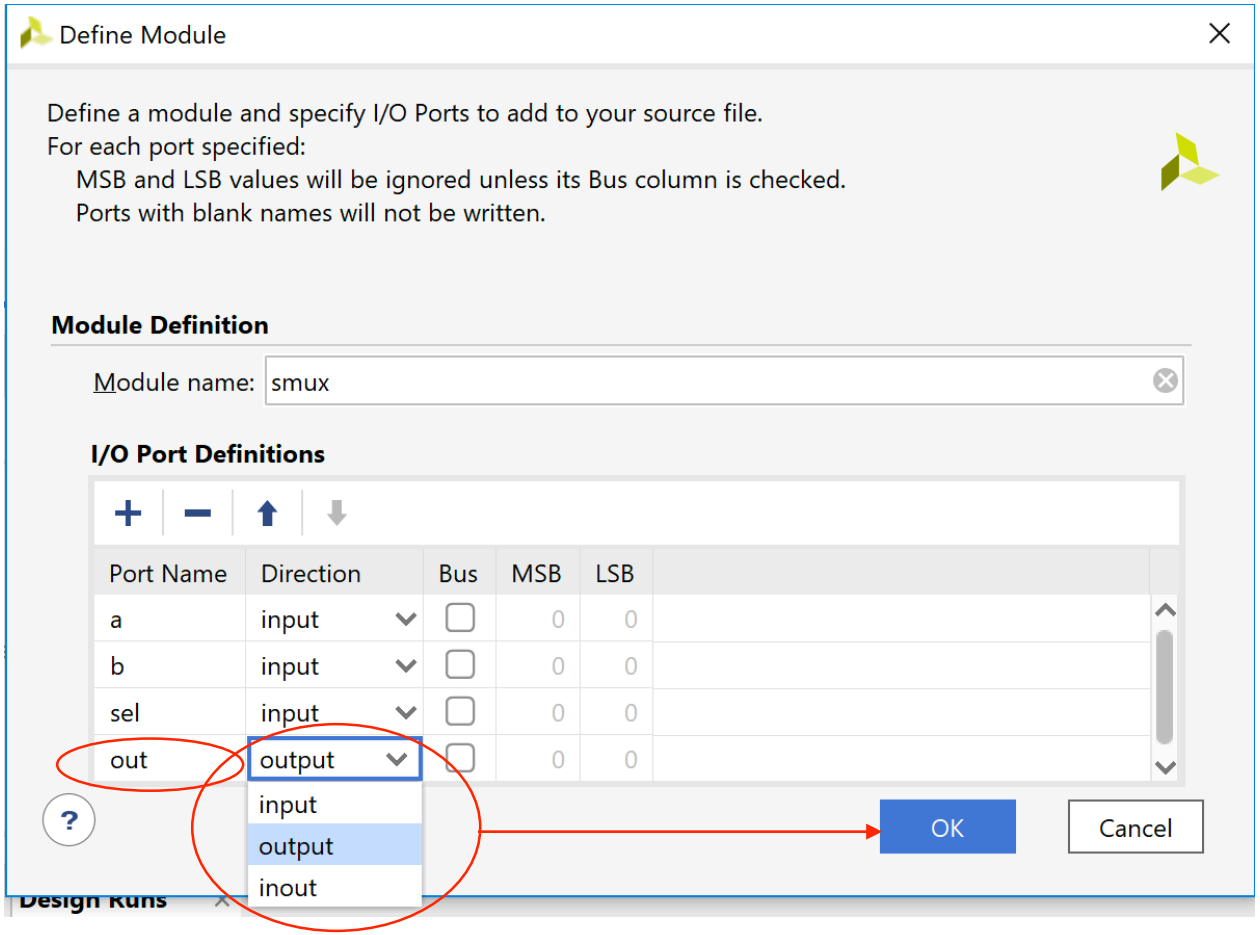
New Source (2/5)



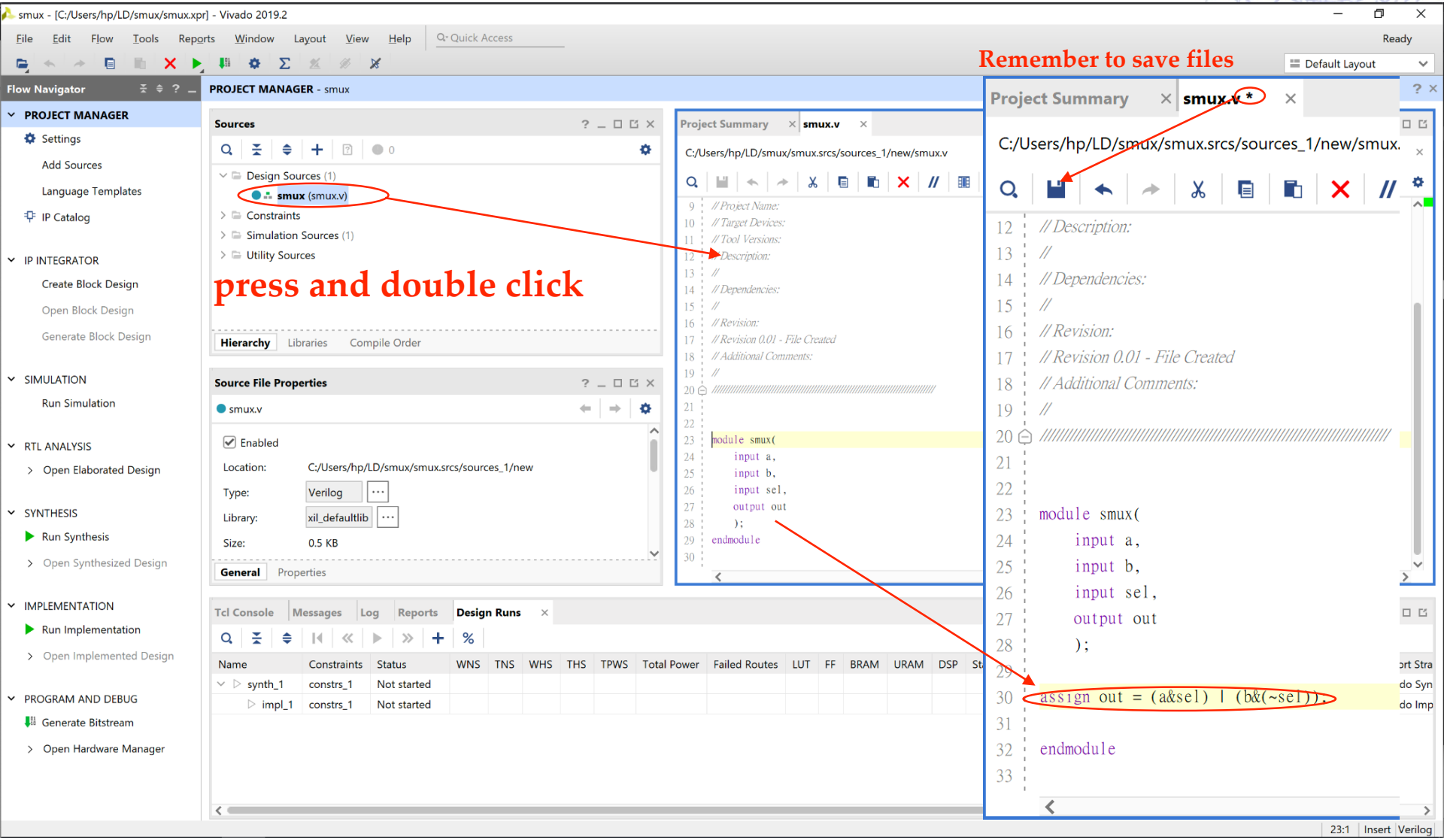
New Source (3/5)



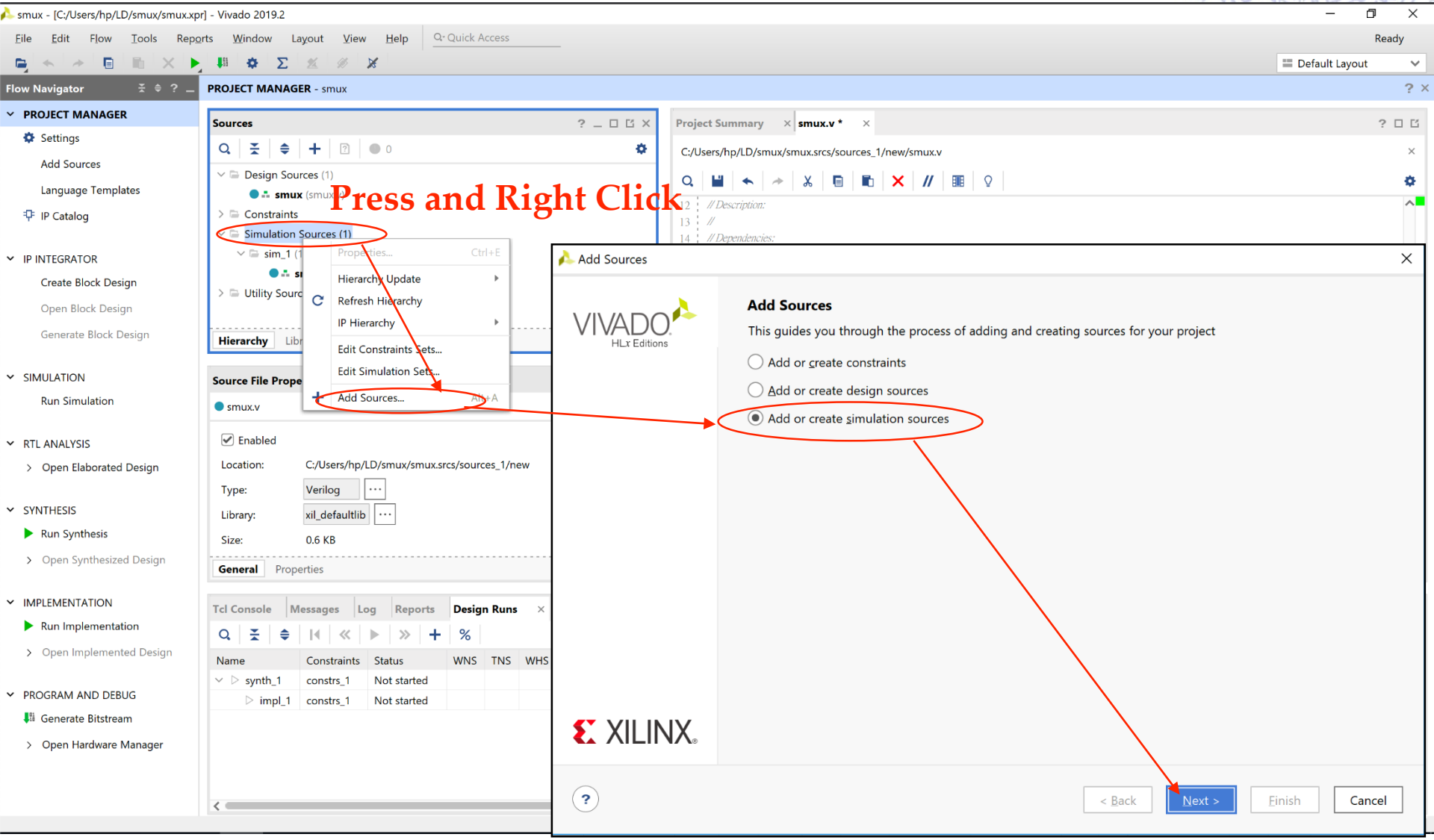
New Source (4/5)



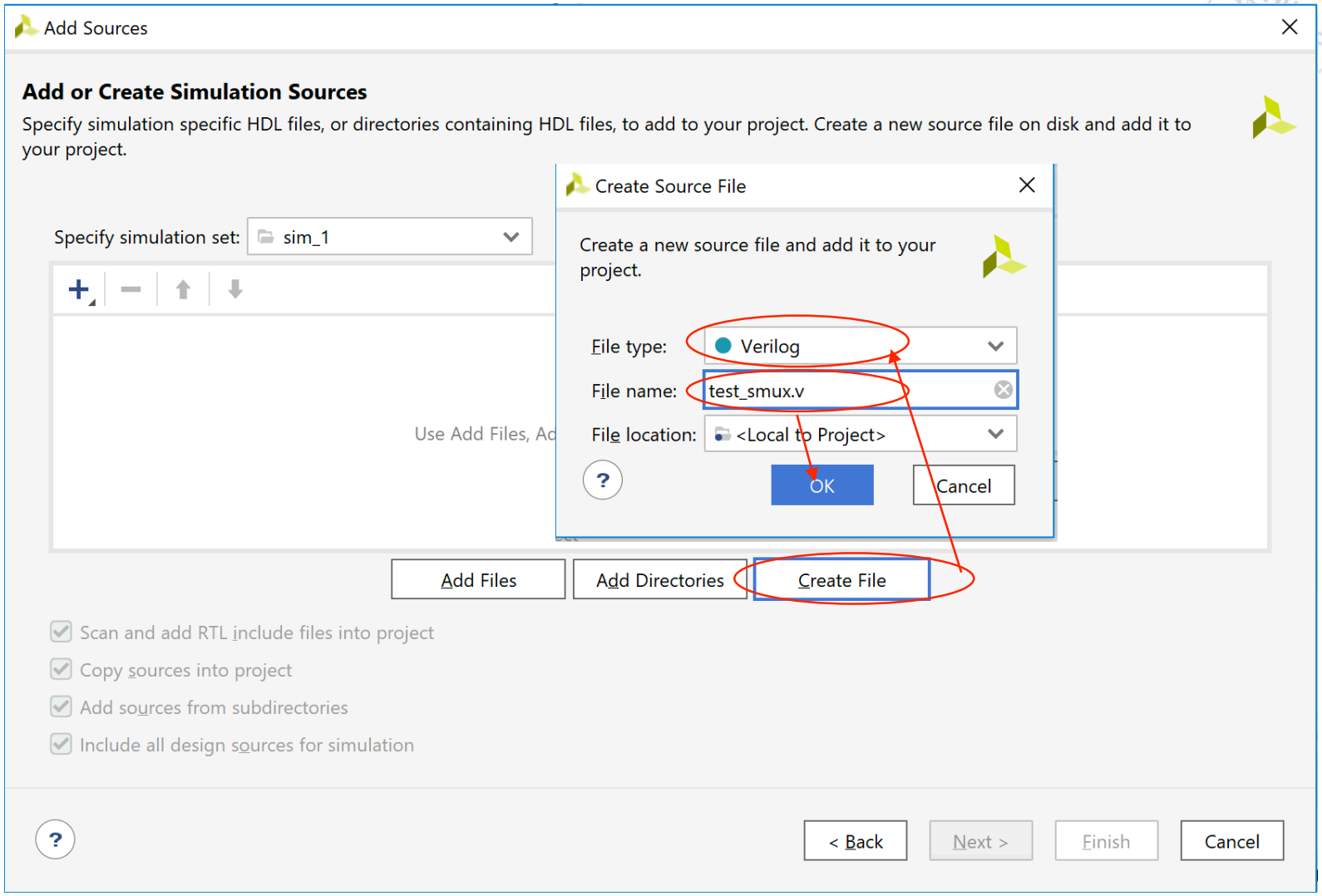
New Source (5/5)



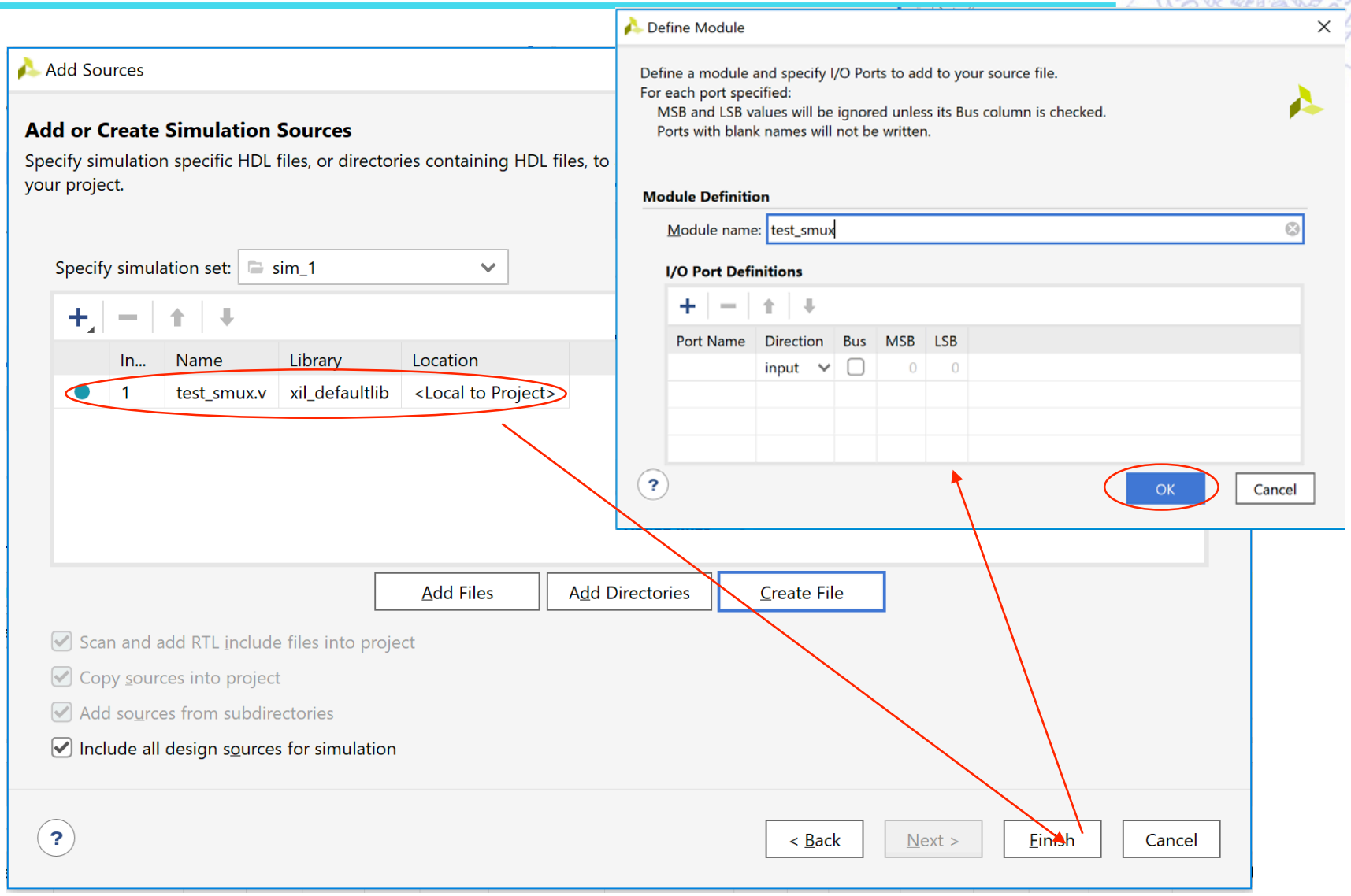
Add Testbench (1/5)



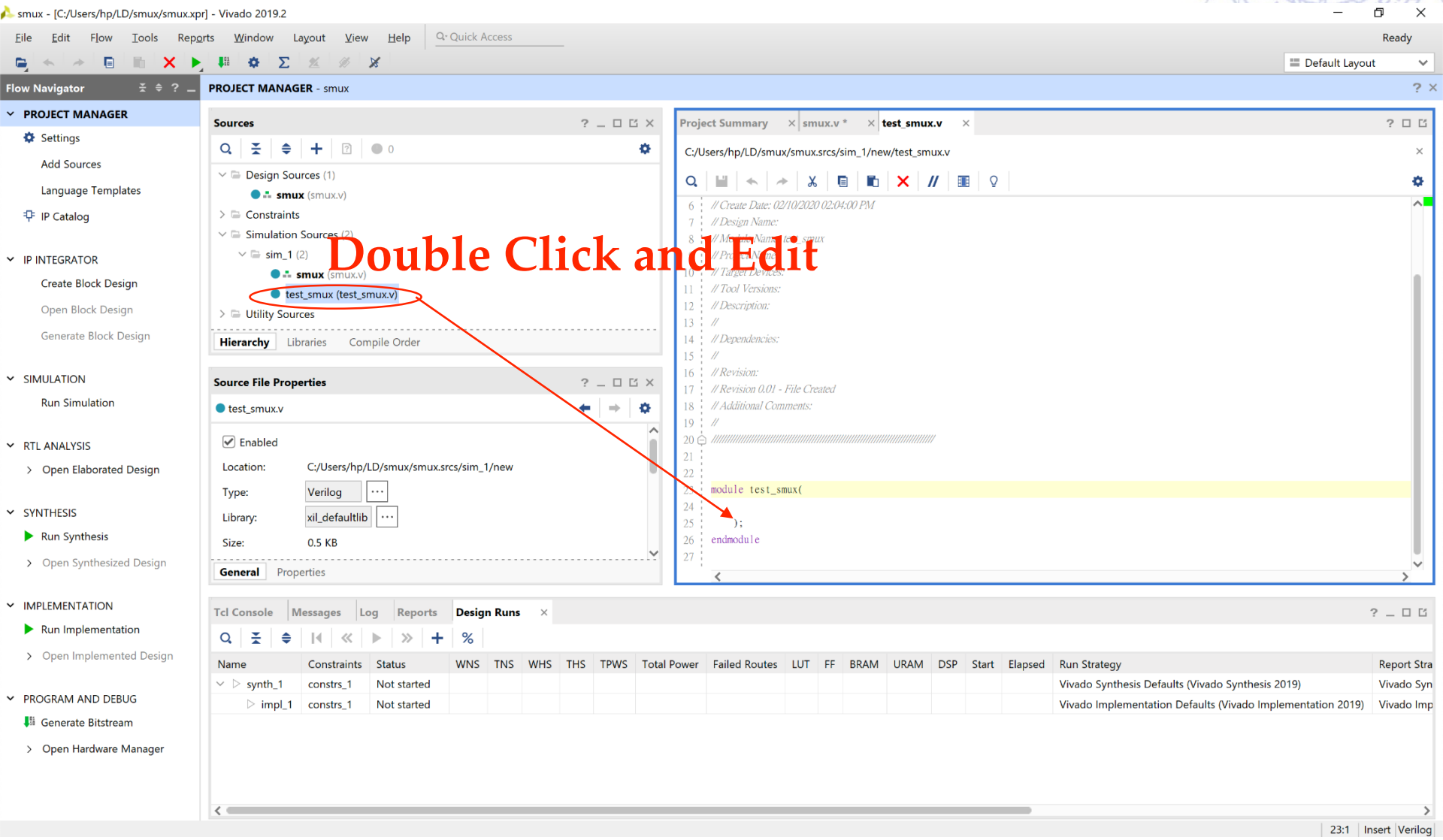
Add Testbench (2/5)



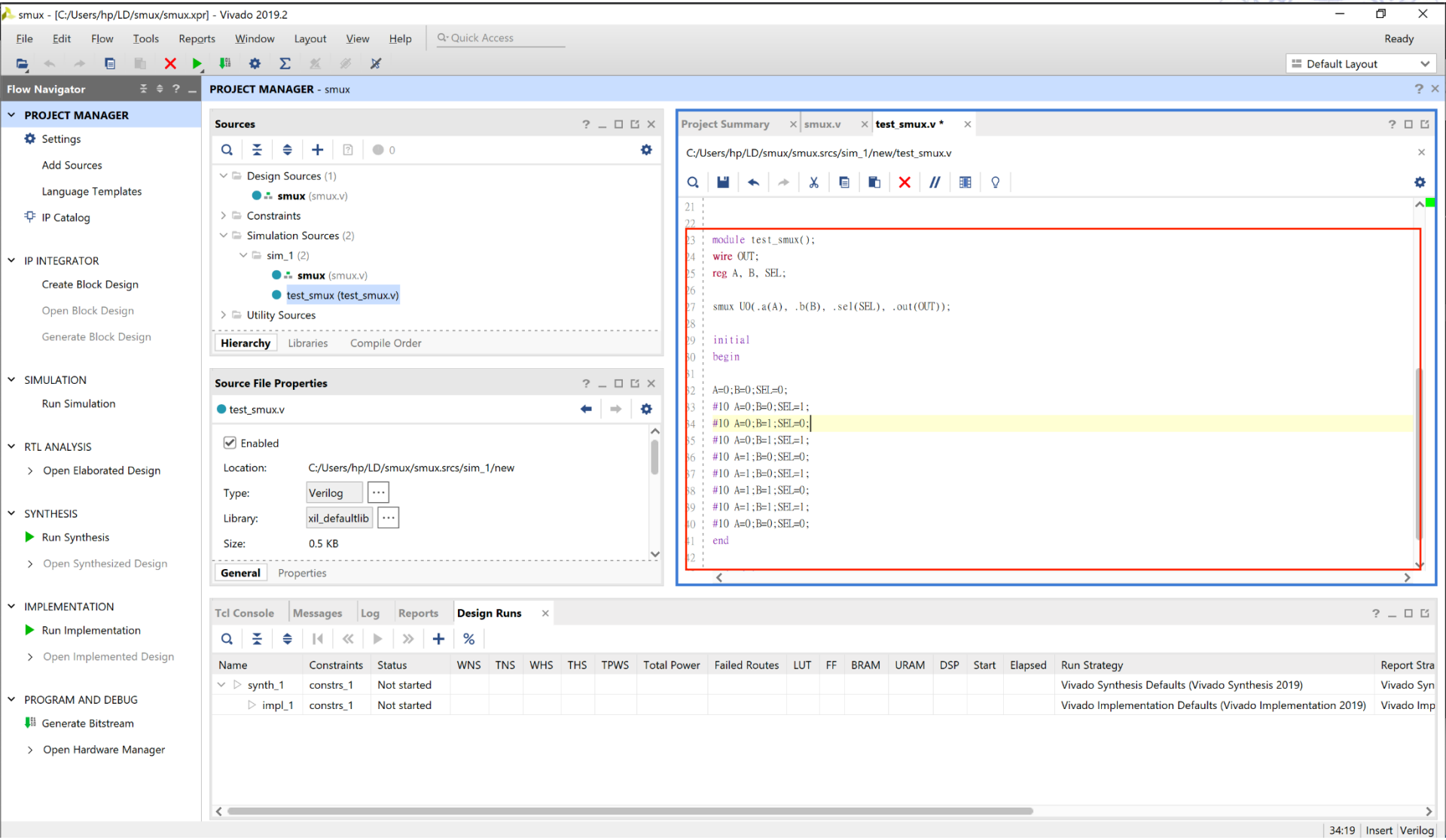
Add Testbench (3/5)



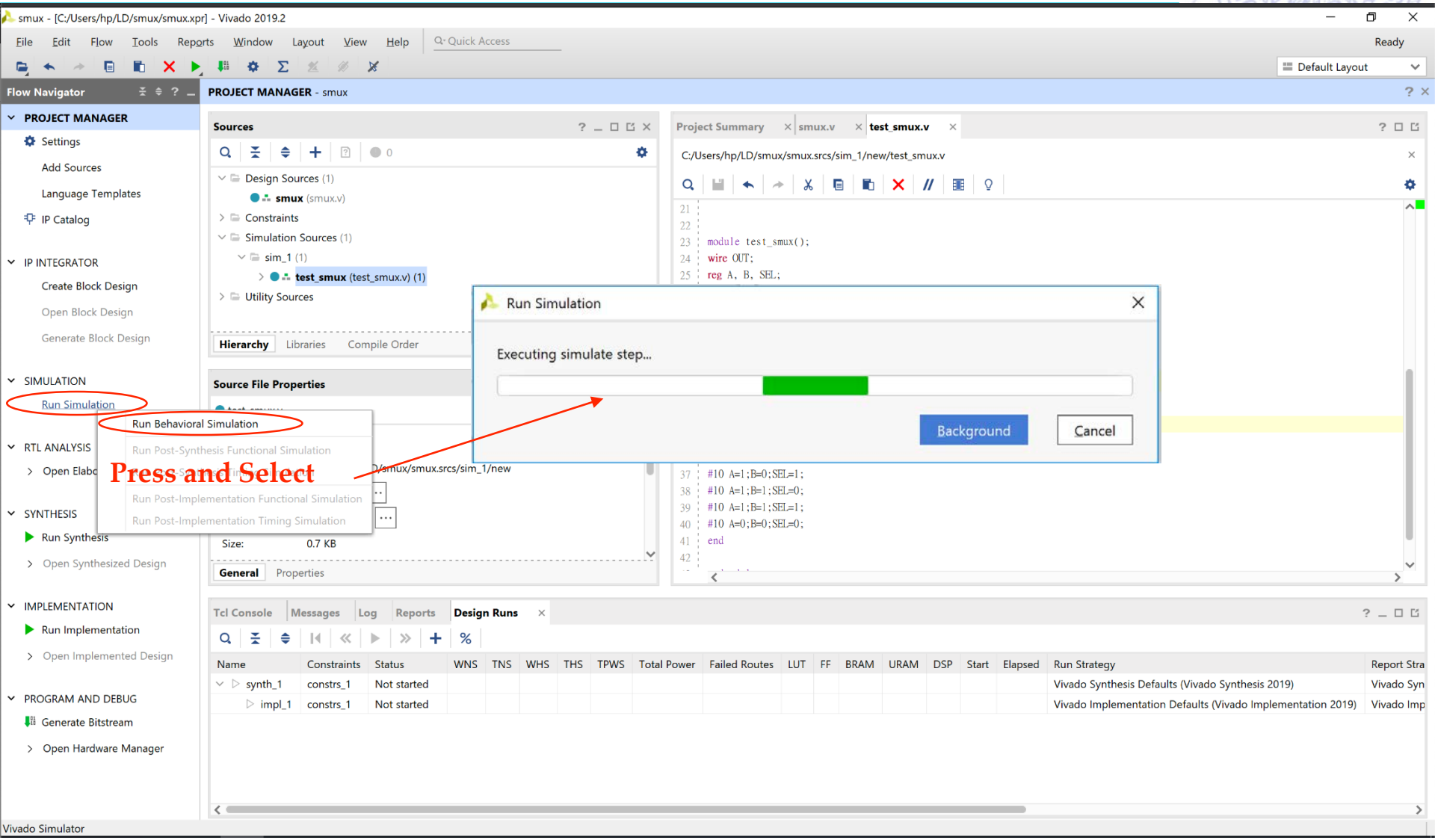
Add Testbench (4/5)



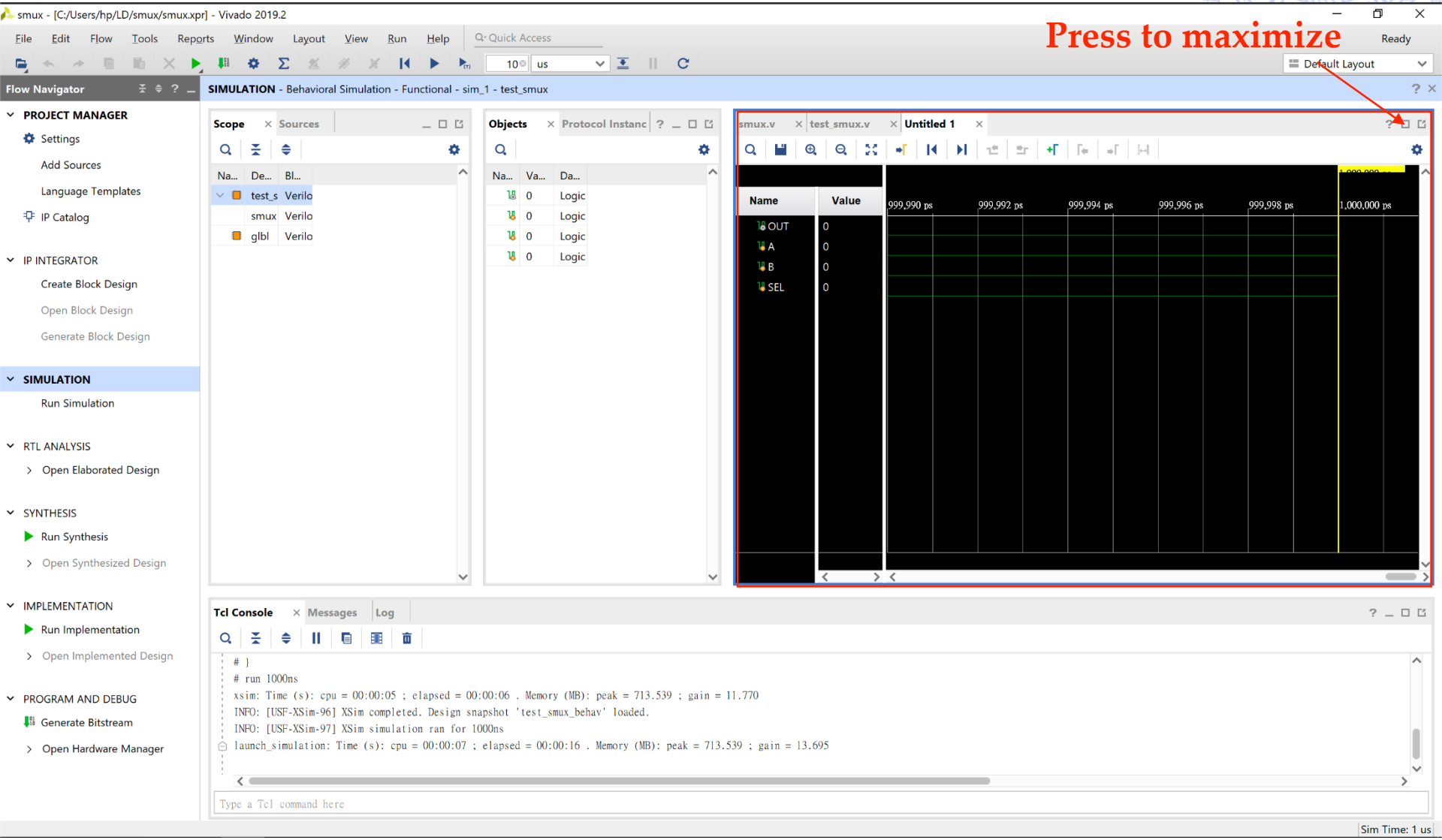
Add Testbench (5/5)



Simulation (1/4)

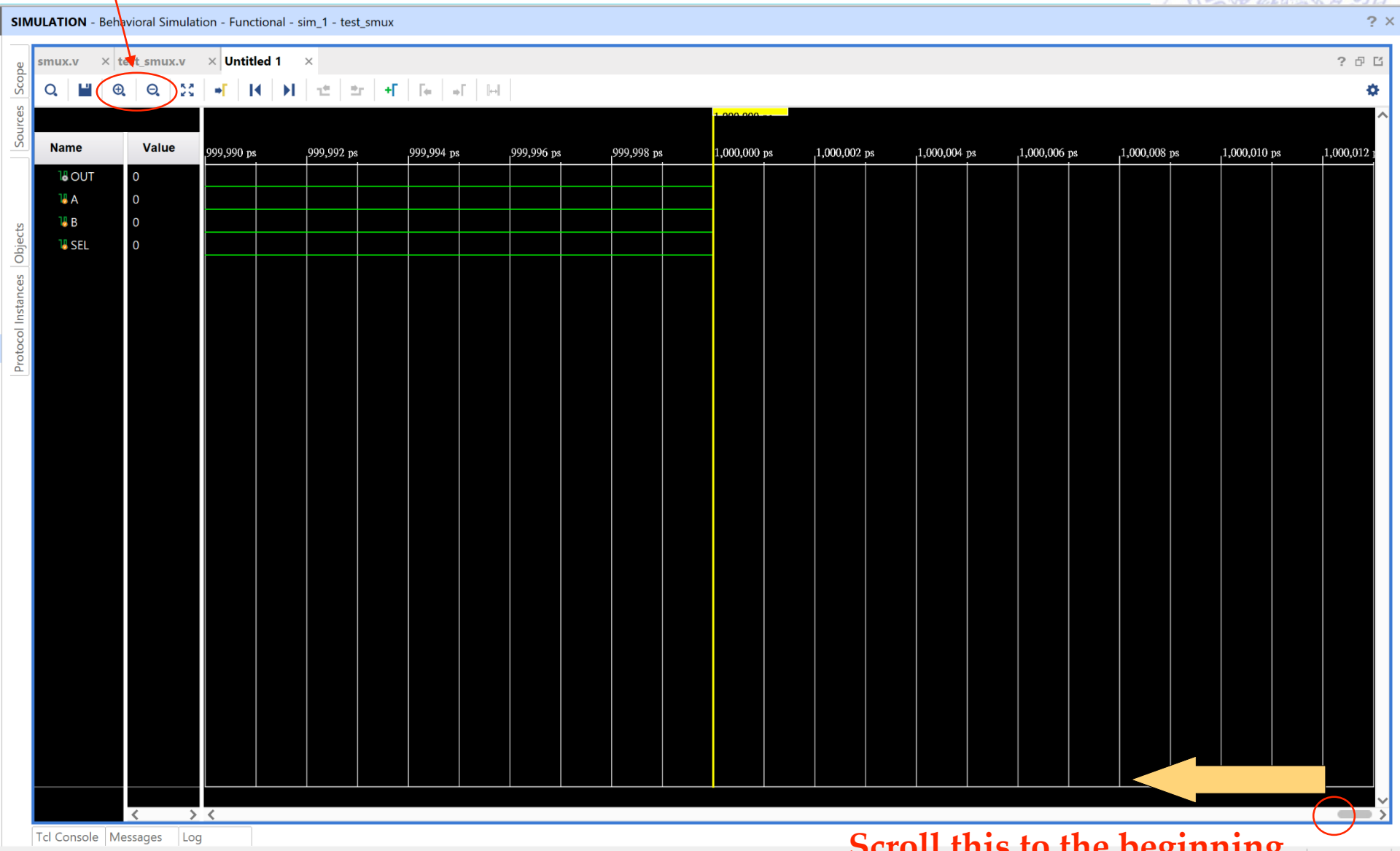


Simulation (2/4)



Simulation (3/4)

Use to zoom in or zoom out



Scroll this to the beginning

Simulation (4/4)

Press to close simulation