Lab1: Introduction to the EDA Tools

National Chiao Tung University
Chun-Jen Tsai
9/15/2017

Lab 1 Goal: Design an 8-bit Multiplier

- In this lab, you must design an 8-bit sequential binary multiplier and use the Vivado Simulator to verify your design
  - You should review your old textbook on Digital Circuit Design by Mano. Some design guideline of the sequential binary multiplier is in section 8.7 of Mano’s book.
  - You must design your multiplier using only adder, shifter, multiplexor, and gate-level operators. You can not use the multiplication operator of Verilog.
- You must demo to your TA during the Lab hours on 9/19 that your circuit works

The Target Technology of Digital Labs

- Digital circuits can be implemented in different ways
  - Circuits Boards
    - Circuit board design using standard IC parts (e.g., 74SLxx)
  - Application Specific ICs
    - Full-custom and Cell-based IC designs
  - Programmable logics
    - Field Programmable Gate Array (FPGA) design
- Here, we use Xilinx FPGAs for circuit implementation
  - Xilinx is the largest FPGA manufacturing company in the world
  - Ross Freeman, the co-founder of Xilinx, invented the very first FPGA in 1986†

† US Patent 4,870,302

Xilinx Vivado Design Suite

- Xilinx has two different EDA tools for FPGA-based digital system designs
  - Vivado Design Suite
    - Only for 7th-generation FPGAs and above
    - Unified IDE for both “SoC” and “digital circuit” designs
  - ISE Design Suite
    - For 7th- and older generations of FPGAs
    - ISE EDK for SoC designs
    - ISE Project Navigator for digital circuit designs
- In this course, we use the Vivado Design Suite for digital circuit design

Vivado Circuit Implementation Flow

- Step 1: Design Entry
  - Input your circuit design using Hardware Description Language (HDL), such as Verilog or VHDL
- Step 2: Synthesis
  - Convert from the HDL programs or schematics to a netlist file that define a list of circuit blocks and how they are connected
- Step 3: Mapping
  - Determine what FPGA resource will be used to implement which part of the netlist
- Step 4: Place-and-Route
  - Determine physical location and routing of the circuit resource
  - A “*.bit” file will be generated for the FPGA device

Vivado Circuit Debug Flow

- Your design may not be perfect in the first try!
  - Circuit debugging is done via “simulation” or “signal probing”
- Vivado supports several simulation types. In particular:
  - Behavioral simulation
    - Functional simulation before synthesis; assumes zero delay
  - Post implementation functional simulation
    - Functional simulation after synthesis; assumes zero delay
  - Post implementation timing simulation
    - Simulate signal switching of your circuit with exact signal delays on the target devices
    - Also called “post-sim”
- Vivado Logic Analyzer can analyze runtime signals

Install Your Own Vivado Design Suite

- Install a copy of Vivado Design Suite – HLx Editions 2017.2 onto your computer.
  - You can download it from:

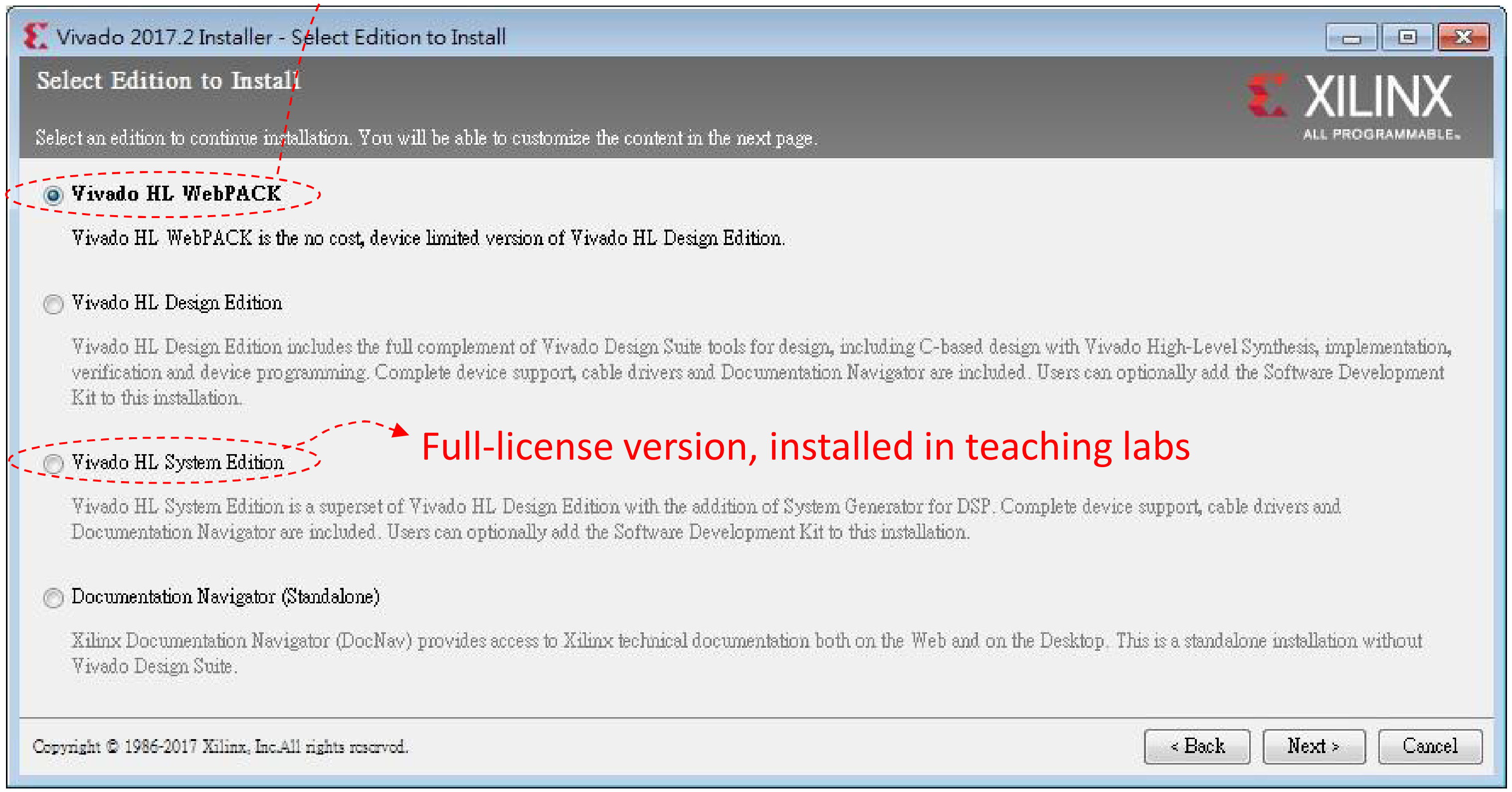
    https://www.xilinx.com/support/download.html

- The installation requires at least 25.3 GiB of disk space
  - Please install the “WebPACK” version and register online for a free license

ISE Installation Note

- You must select the right version upon installation:

Free license version, for personal use, no Vivado Logic Analyzer



Vivado 2017.2 Installer - Select Edition to Install

Select Edition to Install

Select an edition to continue installation. You will be able to customize the content in the next page.

[✓ **Vivado HL WebPACK**]

Vivado HL WebPACK is the no cost, device limited version of Vivado HL Design Edition.

Vivado HL Design Edition

Vivado HL Design Edition includes the full complement of Vivado Design Suite tools for design, including C-based design with Vivado High-Level Synthesis, implementation, verification and device programming. Complete device support, cable drivers and Documentation Navigator are included. Users can optionally add the Software Development Kit to this installation.

Vivado HL System Edition

Full-license version, installed in teaching labs

Vivado HL System Edition is a superset of Vivado HL Design Edition with the addition of System Generator for DSP. Complete device support, cable drivers and Documentation Navigator are included. Users can optionally add the Software Development Kit to this installation.

Documentation Navigator (Standalone)

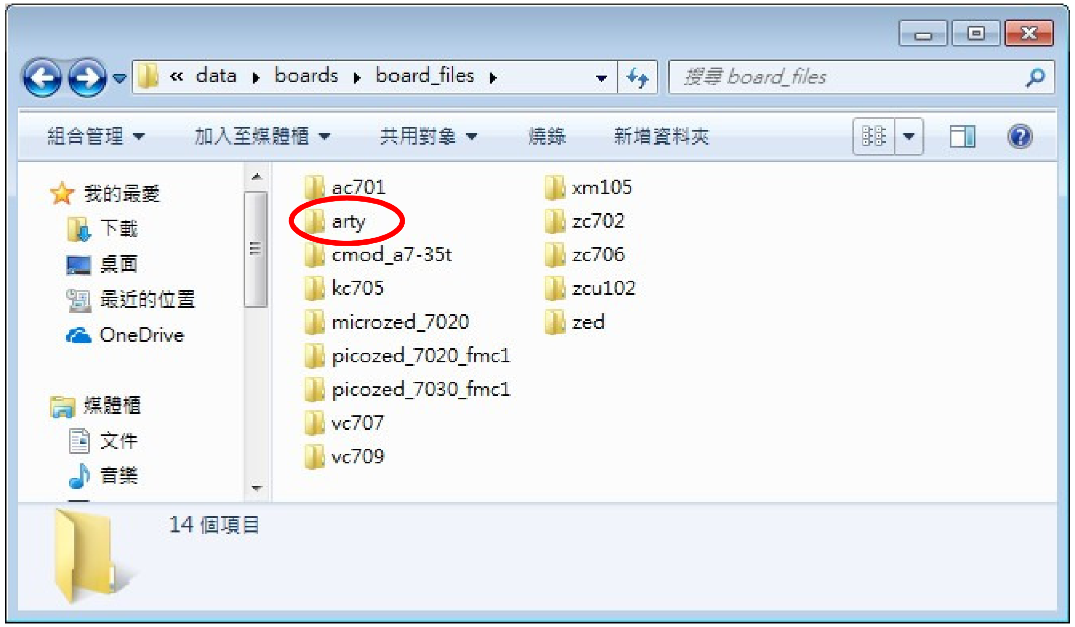
Xilinx Documentation Navigator (DocNav) provides access to Xilinx technical documentation both on the Web and on the Desktop. This is a standalone installation without Vivado Design Suite.

Copyright © 1986-2017 Xilinx, Inc.All rights reserved.

Installation of Arty Board Definitions

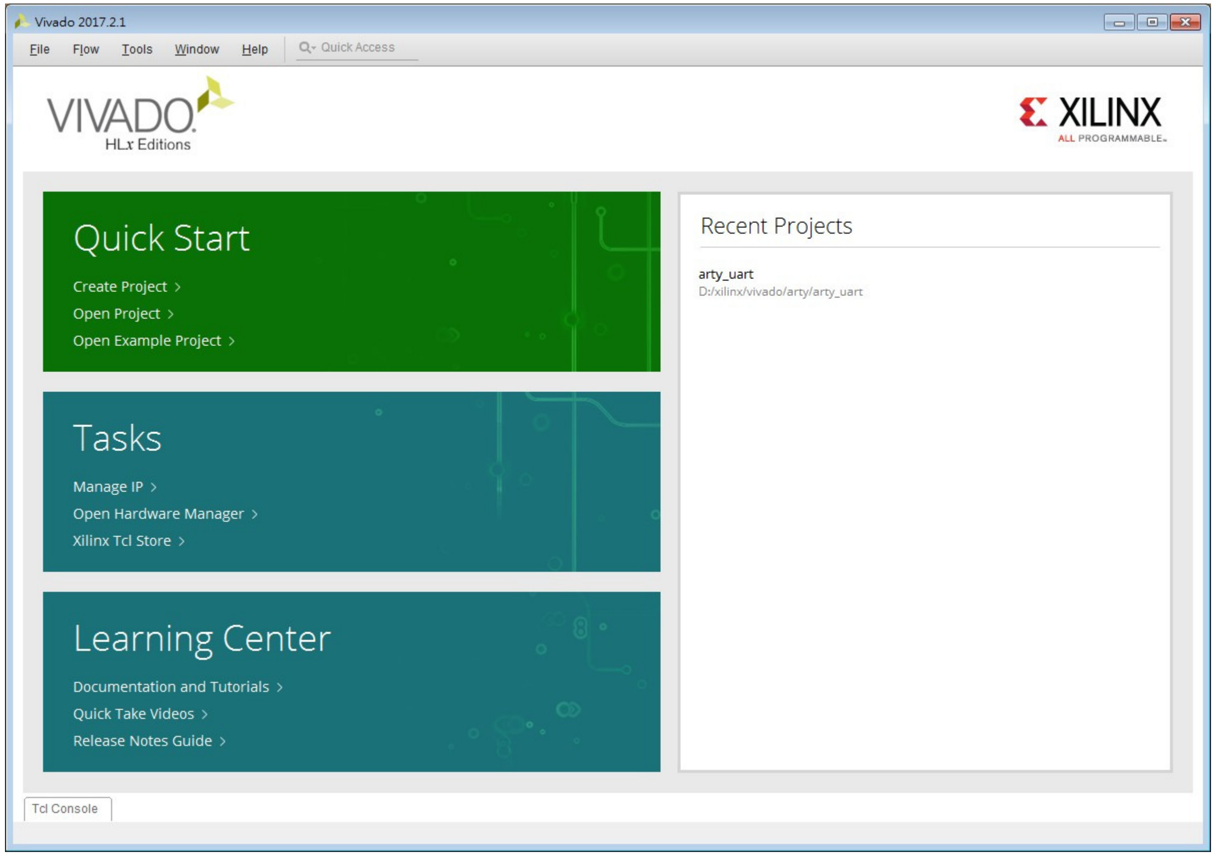
- After the installation of Vivado, you must install the board definition file of Arty:
  - Download arty.zip from E3.
  - Unzip arty.zip to the following directory:

C:\<INST_DIR>\Xilinx\Vivado\2017.2\data\boards\board_files

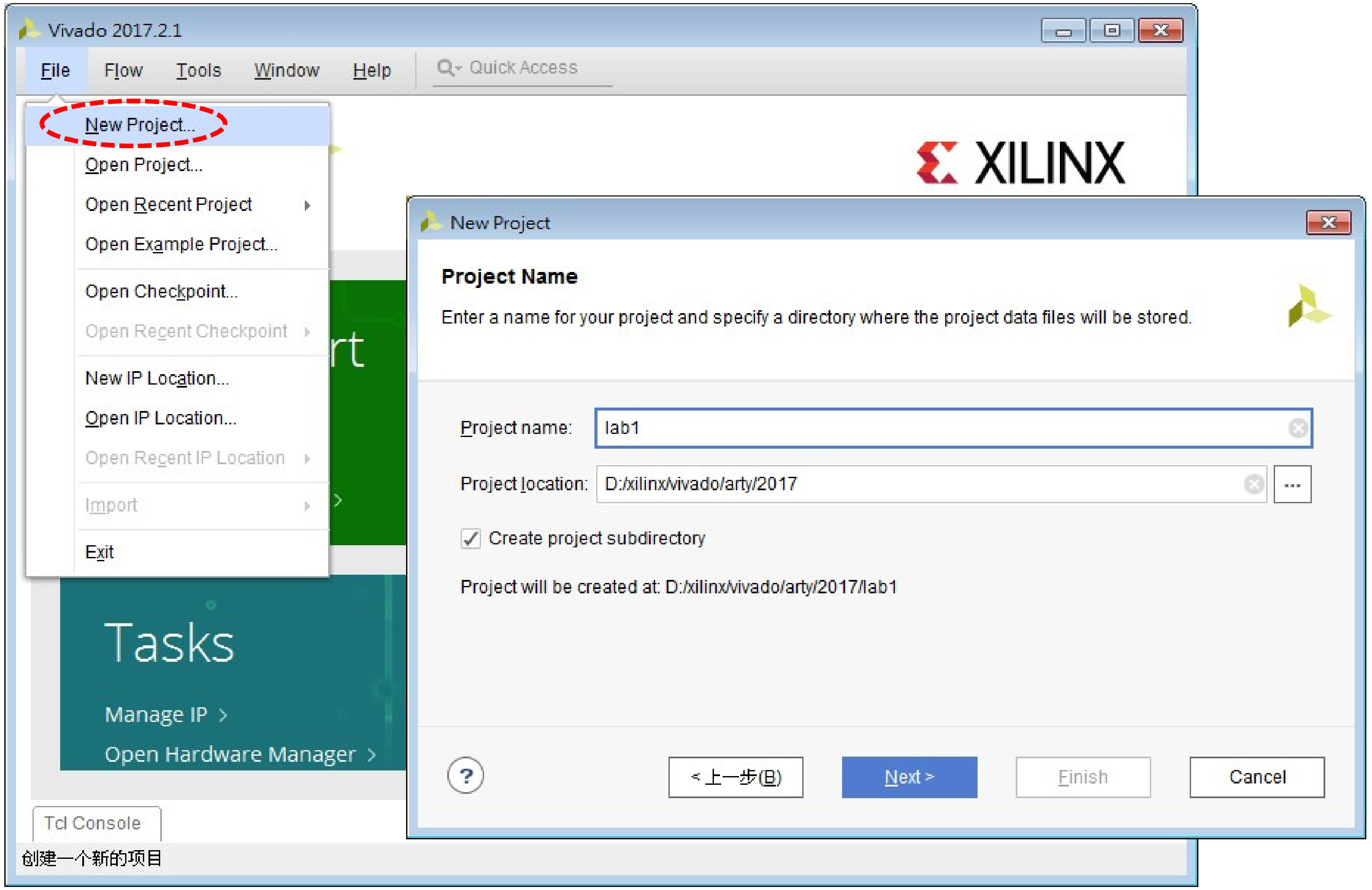


Invoke Vivado 2017.2

- ❑ Double-clicking the Vivado 2017.2 icon on desktop:



Create a New Project in Vivado



Project Name

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

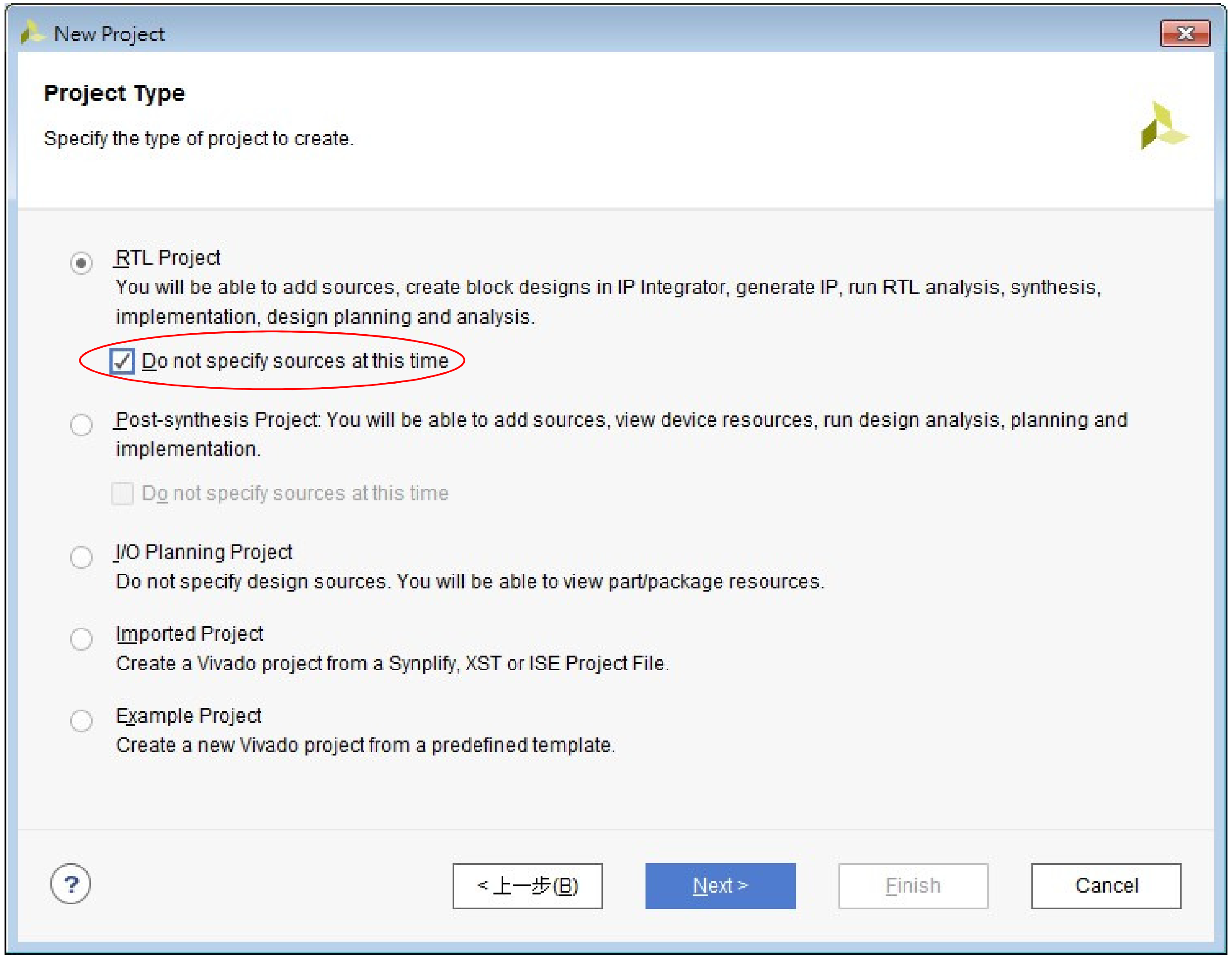
Project name: lab1

Project location: D:/xilinx/vivado/arty/2017

✓ Create project subdirectory

Project will be created at: D:/xilinx/vivado/arty/2017/lab1

Select HDL or Schematic Design



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 Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.

[✓ Do not specify sources at this time]

○ Post-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation.

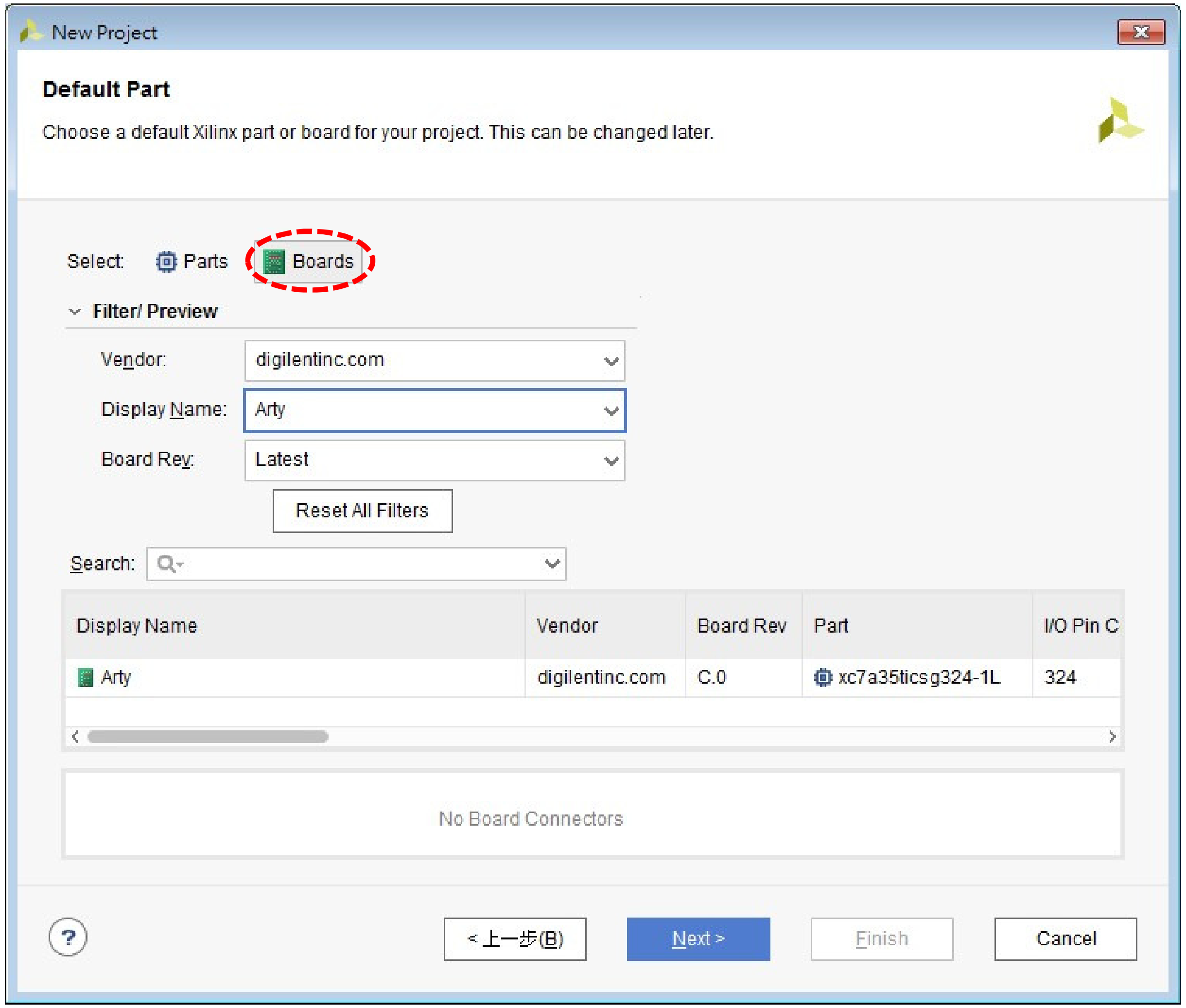
☐ Do not specify sources at this time

○ I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

○ 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.

Select the Target Device



New Project

Default Part

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

Select: Parts [✓ Boards]

Filter/ Preview

Vendor: digilentinc.com

Display Name: Arty

Board Rev: Latest

Reset All Filters

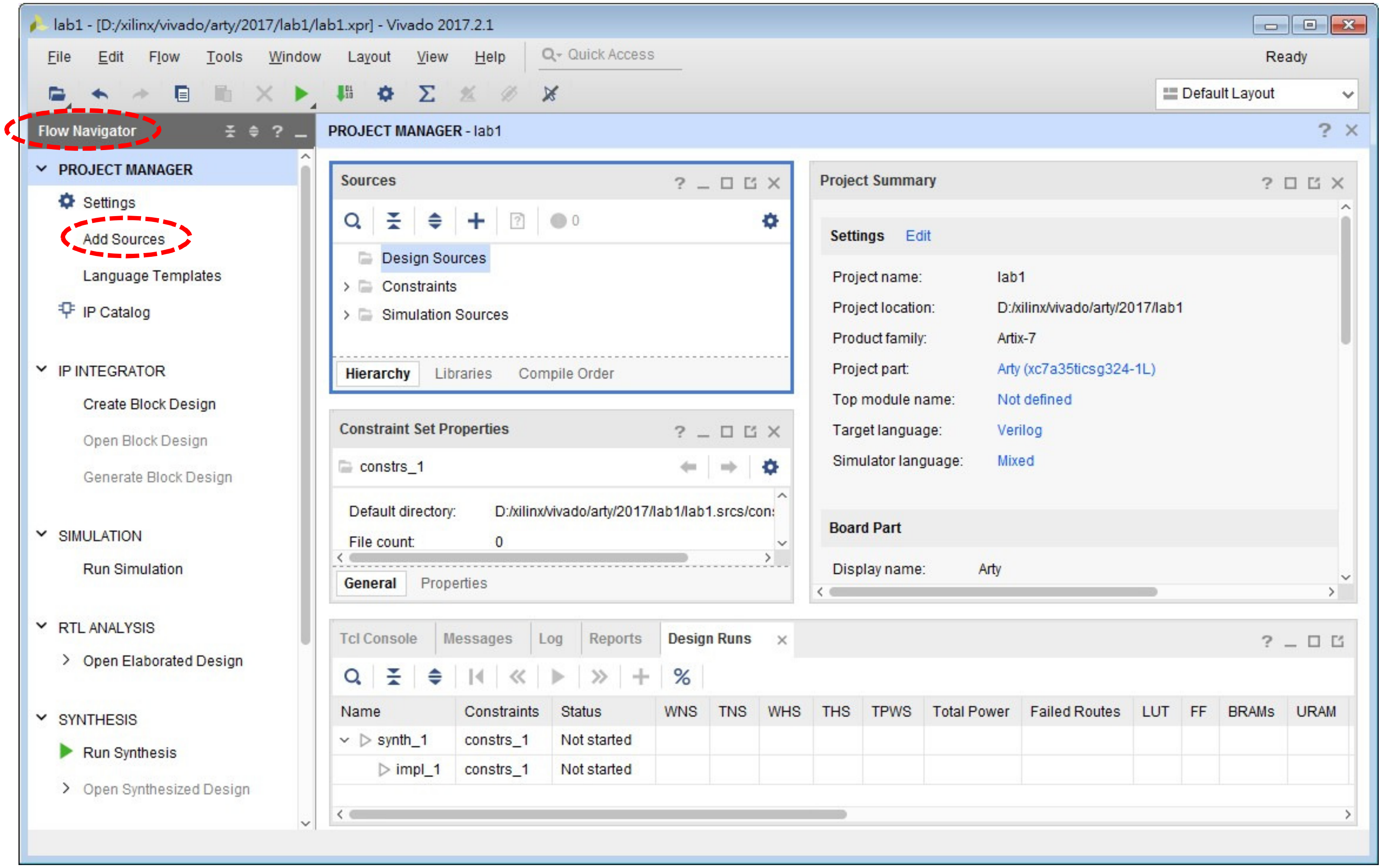
Search:

Display Name	Vendor	Board Rev	Part	I/O Pin C
Arty	digilentinc.com	C.0	xc7a35ticsg324-1L	324

No Board Connectors

< 上一步(B) | Next > | Finish | Cancel

Add a New HDL Source Code



Default directory: D:/xilinx/vivado/arty/2017/lab1/lab1.srcs/con:
File count: 0

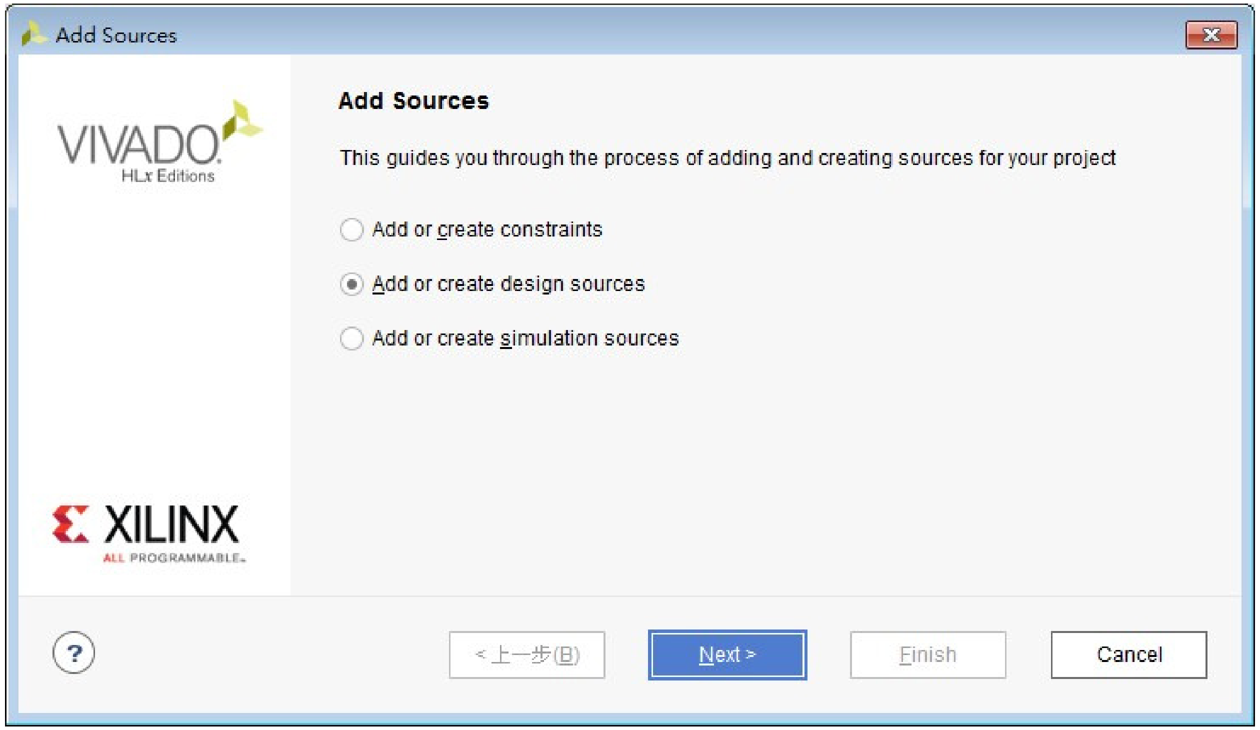
Project name:	lab1
Project location:	D:/xilinx/vivado/arty/2017/lab1
Product family:	Artix-7
Project part:	Arty (xc7a35ticsg324-1L)
Top module name:	Not defined
Target language:	Verilog
Simulator language:	Mixed

Display name: Arty

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

Specifying the Source Type to Create

- There are several types of source files in a circuit design project: design sources, constraint sources, simulation sources, and memory sources, etc.



Add Sources

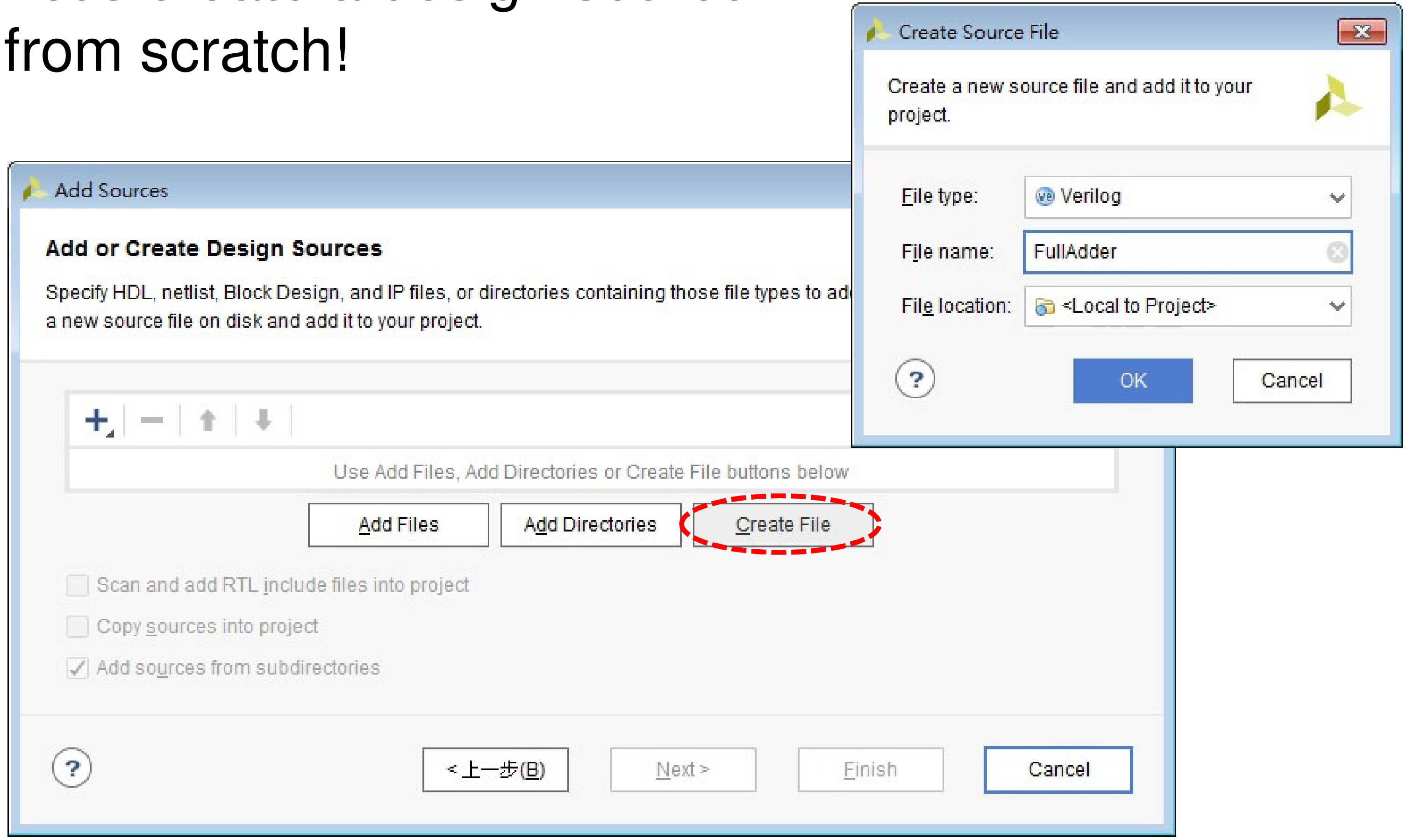
Add Sources

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

Create a 4-bit Full Adder Design

- Let's create a design source from scratch!



Add Sources

Add or Create Design Sources

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

Use Add Files, Add Directories or Create File buttons below

Add Files | Add Directories | Create File

☐ Scan and add RTL include files into project
☐ Copy sources into project
☑ Add sources from subdirectories

< 上一步(B) | Next > | Finish | Cancel

Create Source File

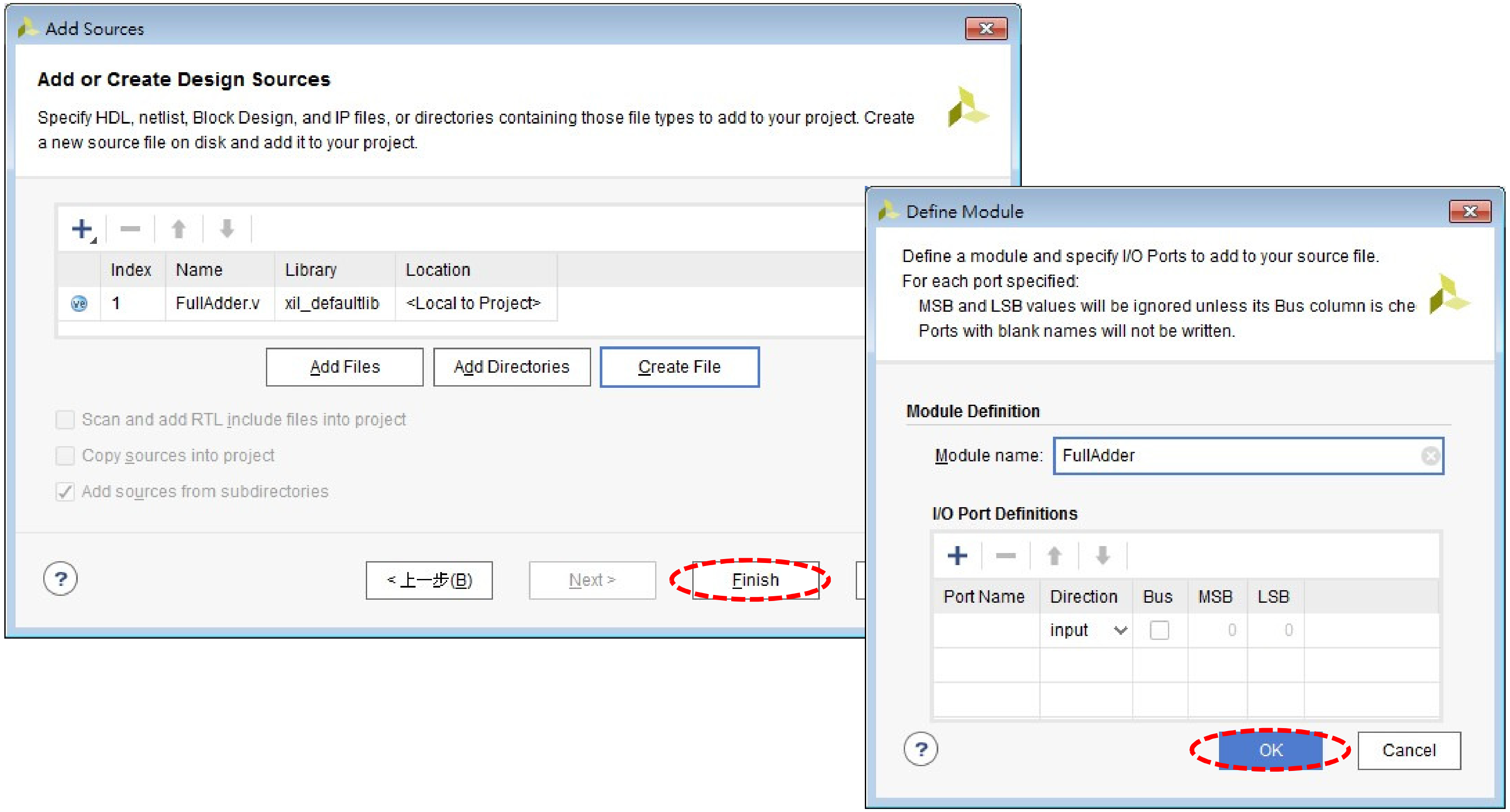
Create a new source file and add it to your project.

File type: Verilog
File name: FullAdder
File location: <Local to Project>

OK | Cancel

Confirm to Create the Verilog Module

- You can define your ports here or do it in the HDL code:



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.

Index	Name	Library	Location
1	FullAdder.v	xil_defaultlib	<Local to Project>

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

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 che
Ports with blank names will not be written.

Module Definition

Module name: FullAdder

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
	input	☐	0	0

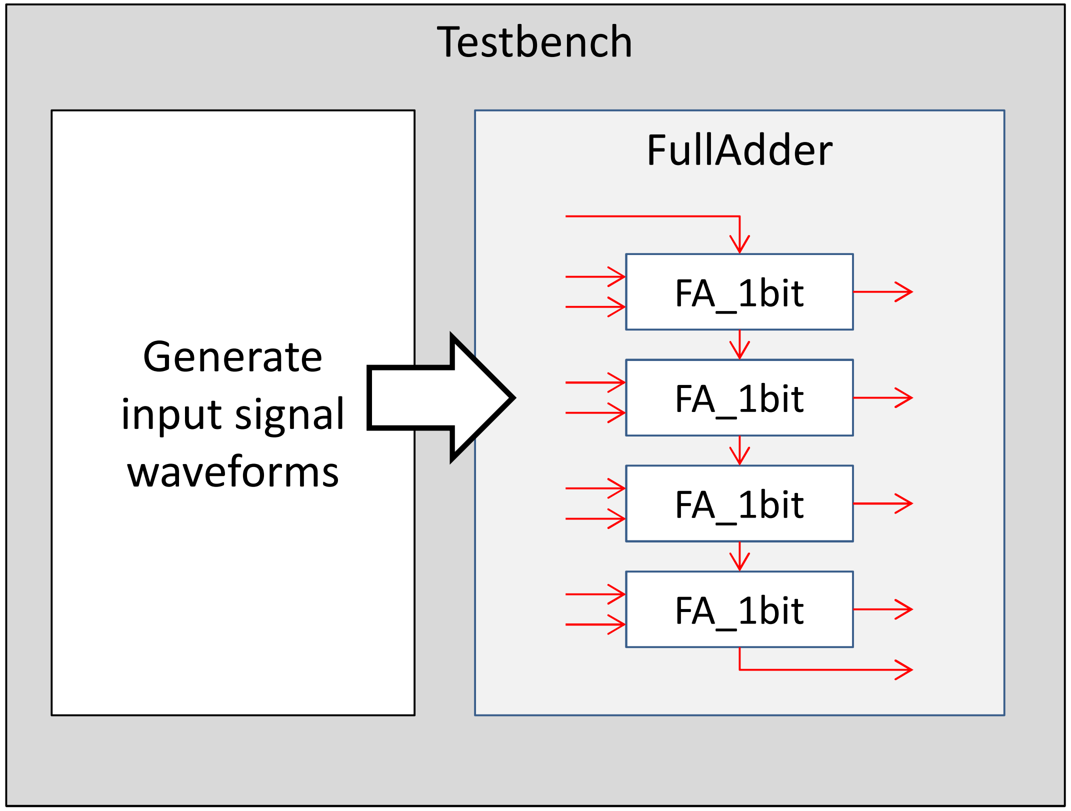
Type in the HDL Source Code

- You can now enter the following source code:

```
// ------------- A four-bit full adder -------------------------------
module FullAdder(A, B, Cin, S, Cout);
  input [3:0] A, B;
  input Cin;
  output [3:0] S;
  output Cout;
  wire [2:0] t;

  FA_1bit FA0(.A(A[0]), .B(B[0]), .Cin(Cin), .S(S[0]), .Cout(t[0]));
  FA_1bit FA1(.A(A[1]), .B(B[1]), .Cin(t[0]), .S(S[1]), .Cout(t[1]));
  FA_1bit FA2(.A(A[2]), .B(B[2]), .Cin(t[1]), .S(S[2]), .Cout(t[2]));
  FA_1bit FA3(.A(A[3]), .B(B[3]), .Cin(t[2]), .S(S[3]), .Cout(Cout));
endmodule

// ---------------- A 1-bit full adder -----------------------------
module FA_1bit(A, B, Cin, S, Cout);
  input A, B, Cin;
  output S, Cout;

  assign S = Cin ^ A ^ B;
  assign Cout = (A & B) | (Cin & B) | (Cin & A);
endmodule
```

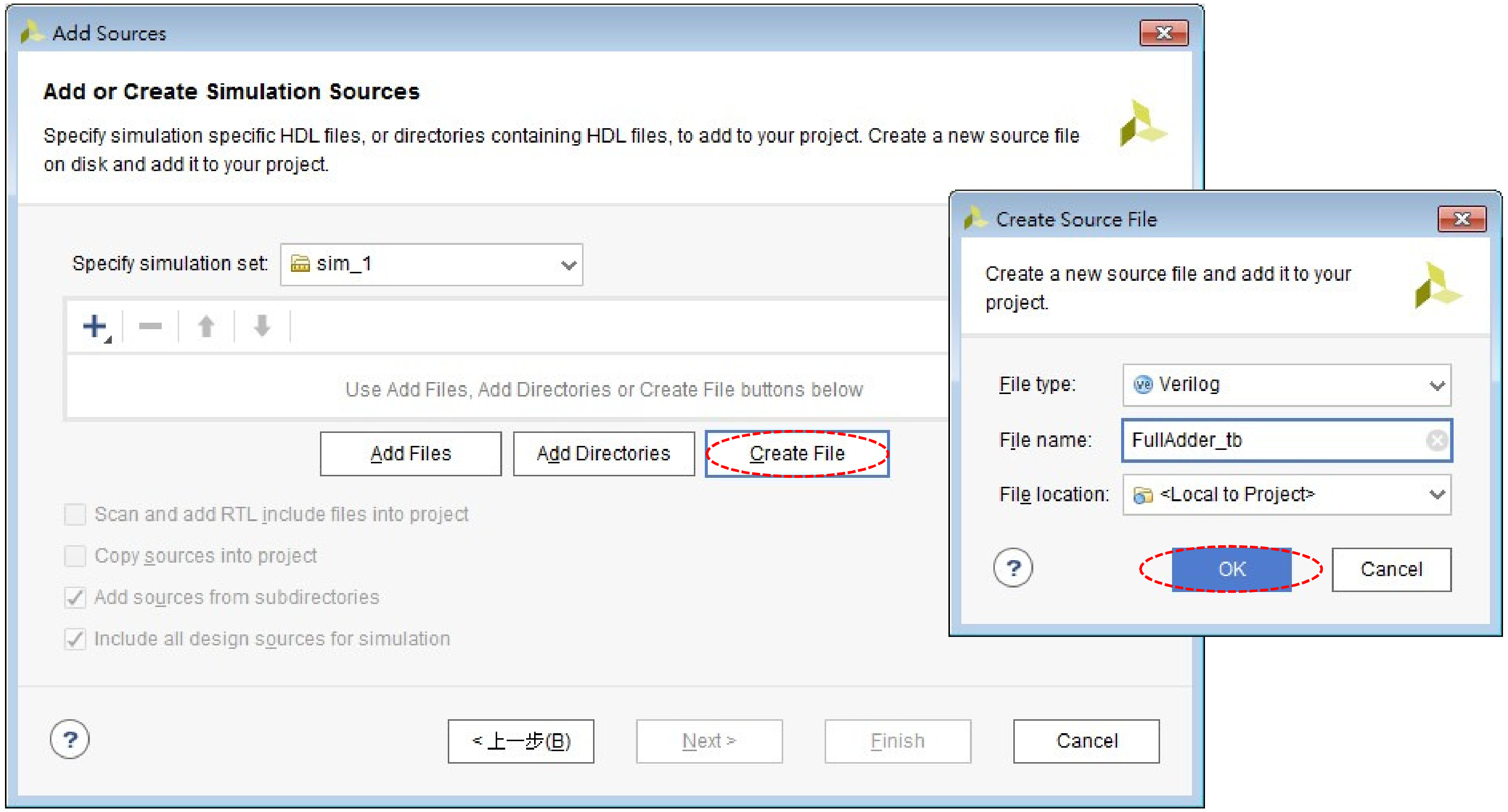
TestBench Design

- You must create a testbench to generate input signals that can feed into your circuit module, such that you can analyze the output to verify its correctness



Create the Testbench Source Code

- ❑ Click “Add Sources” button again, and this time, select “Add or create simulation sources”



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, Add Directories or Create File buttons below

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

Create Source File

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

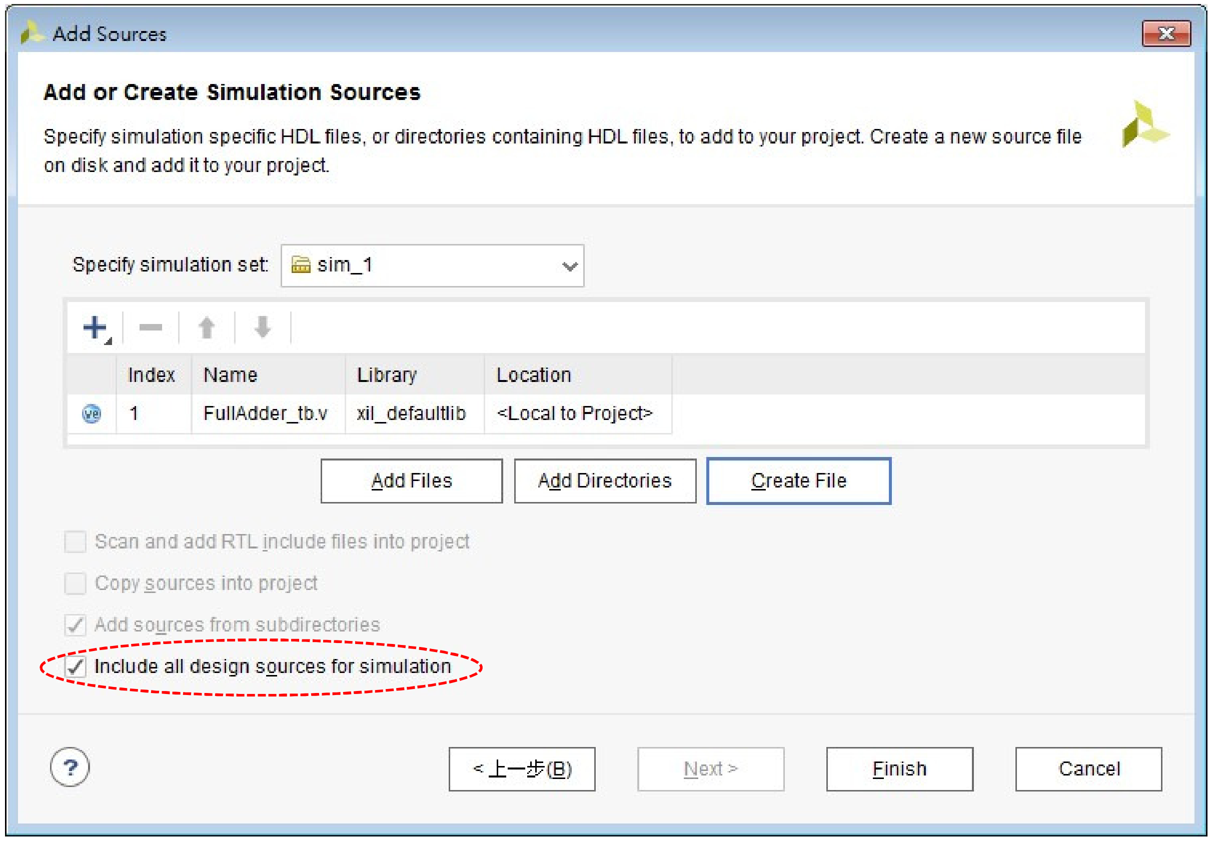
File type: Verilog

File name: FullAdder_tb

File location: <Local to Project>

Confirm the Creation of the Testbench

- Here, we include the design sources into the simulation set so that we can test the modules under development



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.

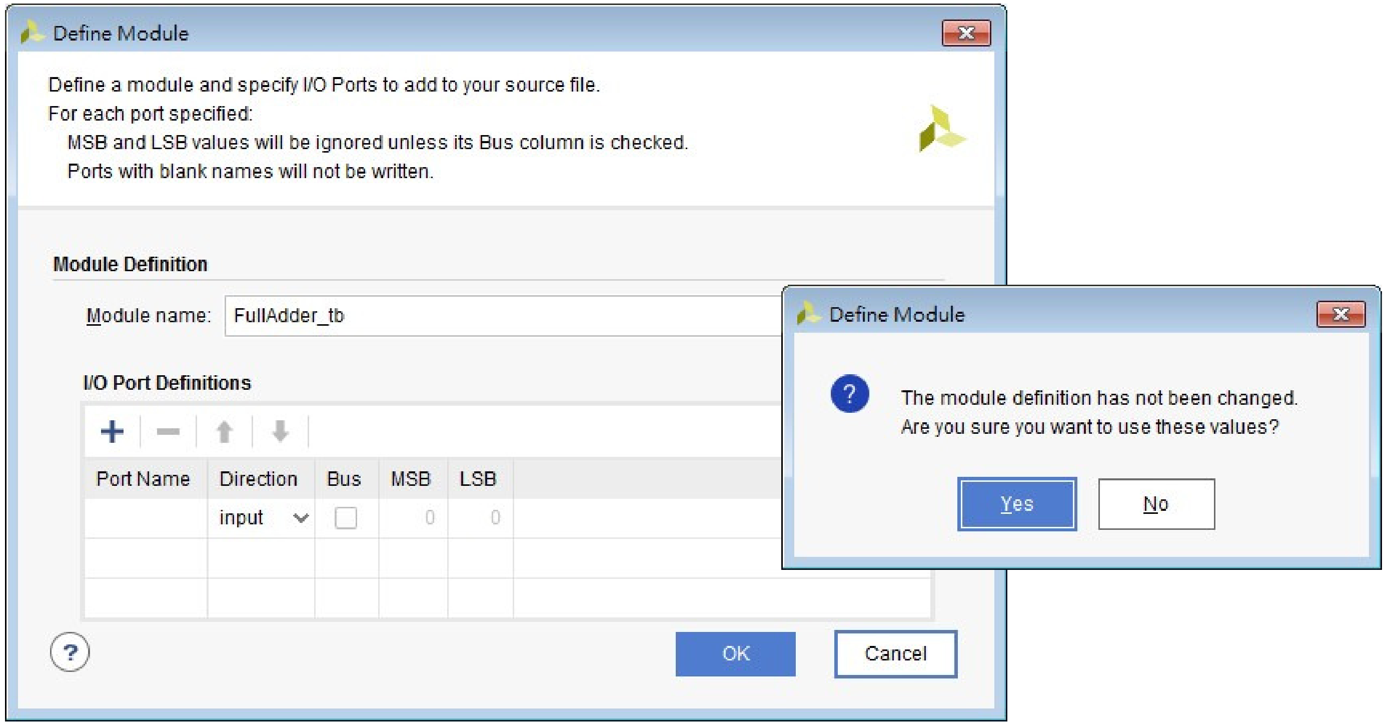
Specify simulation set: sim_1

	Index	Name	Library	Location
	1	FullAdder_tb.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

Create the Testbench Template

- Hit “OK” then “Yes” to create an empty testbench template → top-level of the testbench template usually has no I/O ports



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: FullAdder_tb

I/O Port Definitions

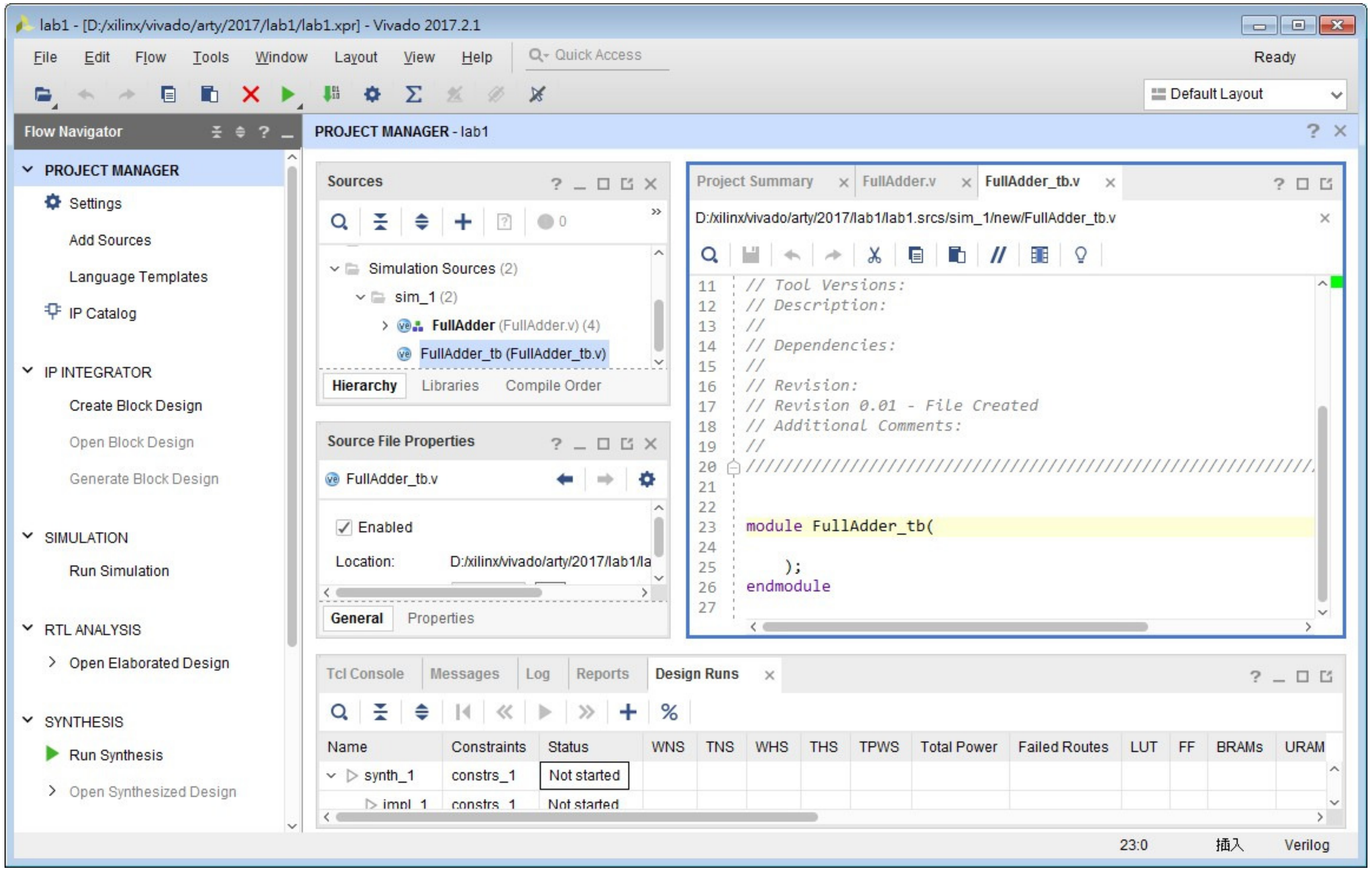
Port Name	Direction	Bus	MSB	LSB
	input	☐	0	0

Define Module

The module definition has not been changed.
Are you sure you want to use these values?

Type in the Testbench Source Code

- We can now type in (copy-paste) the testbench code:



```
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////


module FullAdder_tb(

    );
endmodule
```

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

The Sample Testbench Code

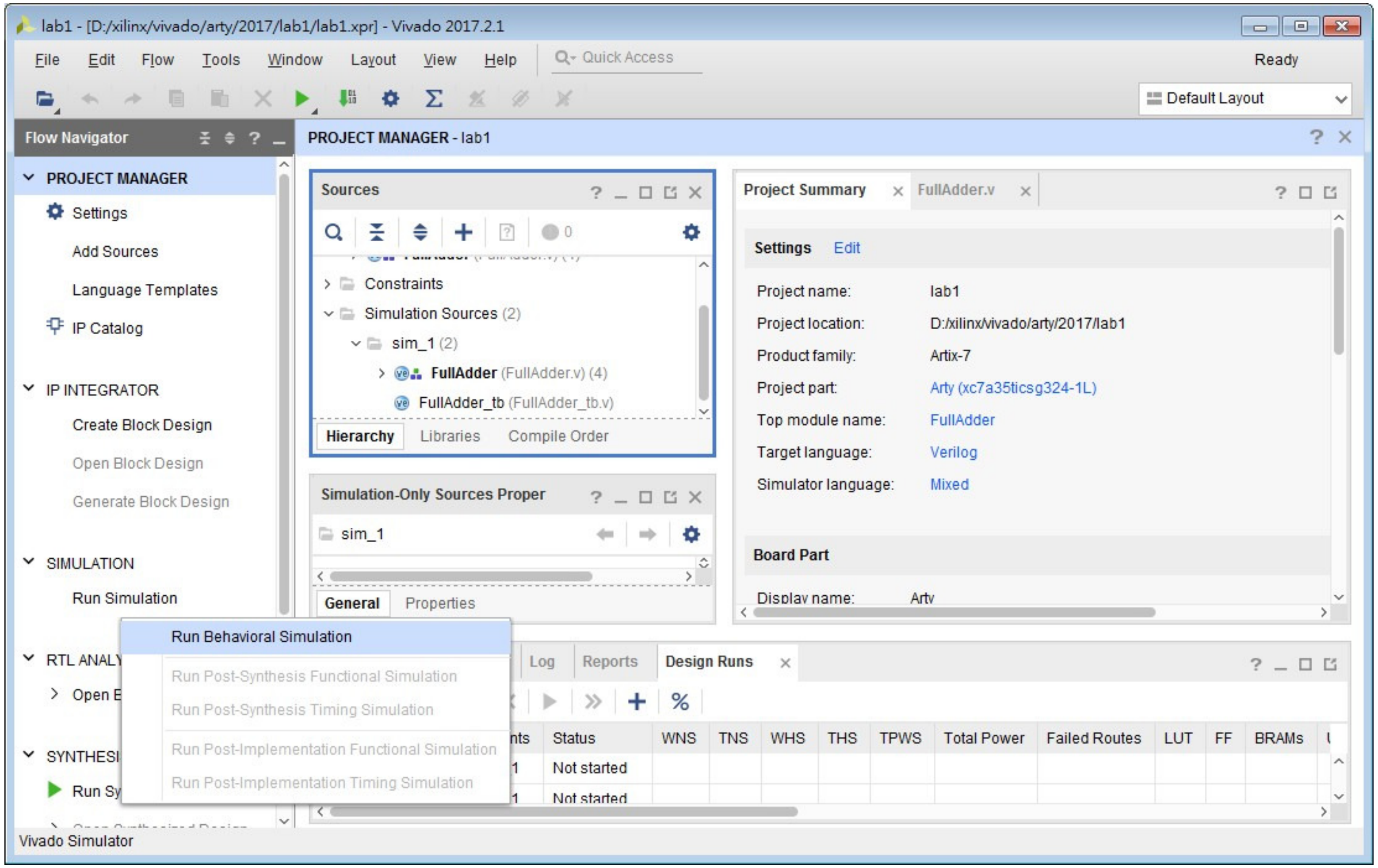
- The template created by Vivado is an empty module; we must add your own test pattern generator to it

```
module FullAdder_tb;

 // inputs
  reg clk = 1;
  reg [3:0] A, B;
  reg Cin;

  // outputs
  wire [3:0] S;
  wire Cout;

  // Instantiate the Unit
  // Under Test (UUT)

  FullAdder uut(
    .A(A),
    .B(B),
    .Cin(Cin),
    .S(S),
    .Cout(Cout)
  );
```

```
  // 100MHz clock generator
  always
    #5 clk = !clk;

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

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

    // Add stimulus here
    A = 4'b0101; B = 4'b1010;
    #50;
    A = 4'b1111; B = 4'b0001;
    #50;
    A = 4'b0000; B = 4'b1111;
    Cin = 1'b1;
    #50;
    A = 4'b0110; B = 4'b0001;
  end
endmodule
```

Compile the Design for Simulation

- Now, you can run the simulation:

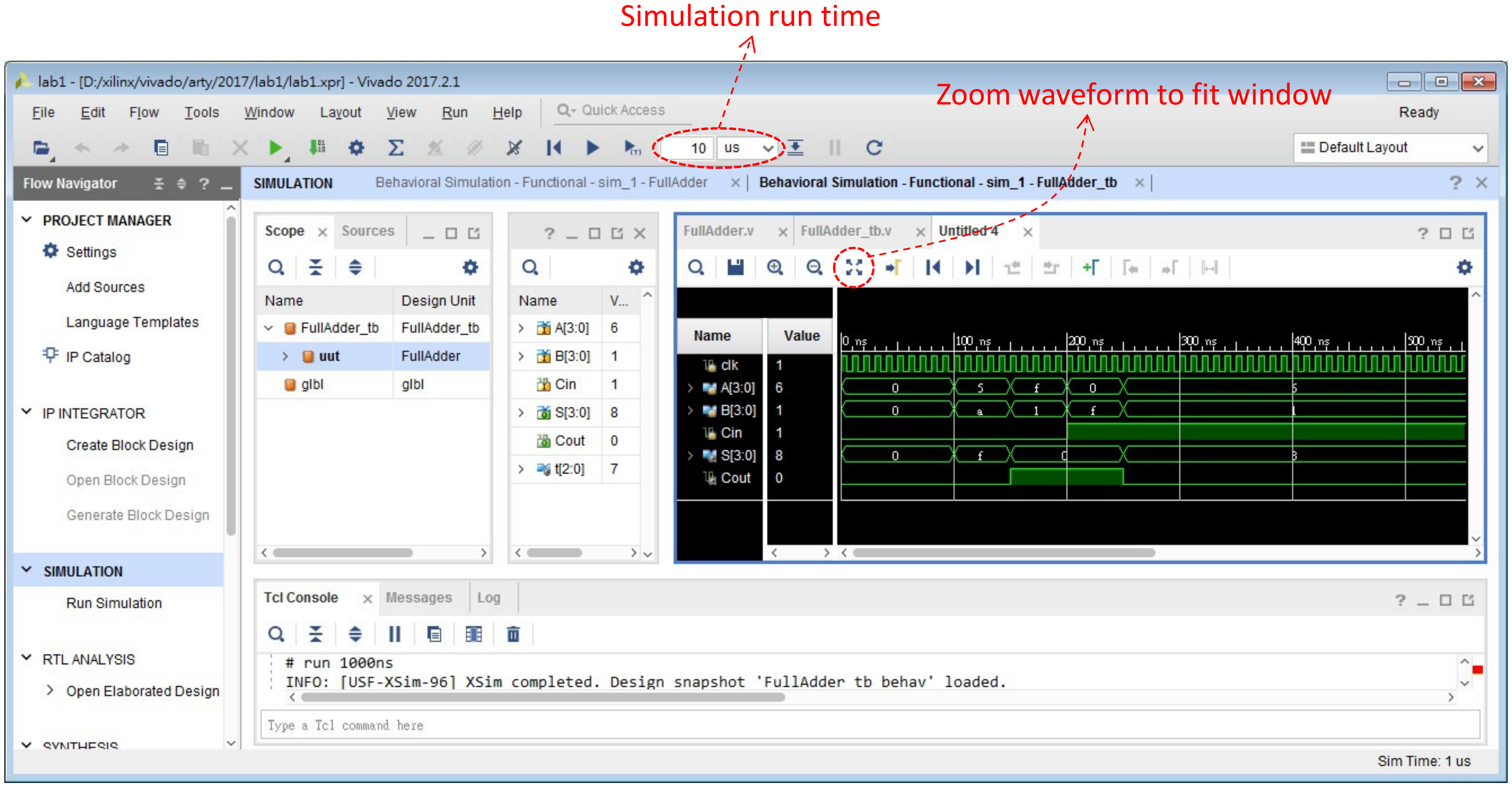


Project name:	lab1
Project location:	D:/xilinx/vivado/arty/2017/lab1
Product family:	Artix-7
Project part:	Arty (xc7a35ticsg324-1L)
Top module name:	FullAdder
Target language:	Verilog
Simulator language:	Mixed

nts	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs
1	Not started										
1	Not started										

Vivado Simulator Window

- When the simulation is done, you get the waveforms:



Name	Design Unit
FullAdder_tb	FullAdder_tb
uut	FullAdder
glbl	glbl

Name	V...
A[3:0]	6
B[3:0]	1
Cin	1
S[3:0]	8
Cout	0
t[2:0]	7

Name	Value
clk	1
A[3:0]	6
B[3:0]	1
Cin	1
S[3:0]	8
Cout	0

```
# run 1000ns
INFO: [USF-XSim-96] XSim completed. Design snapshot 'FullAdder_tb_behav' loaded.
```

Type a Tcl command here

Register Transfer-Level (RTL) Coding

- ❑ For our labs, you do not have to design the circuits purely at gate-level; you can write RTL codes
- ❑ For example, the four-bit adder module can be implemented using the RTL code simply as follows:

```
// ------------- A four-bit full adder ------------------------------
module FullAdder(A, B, Cin, S, Cout);
  input [3:0] A, B;
  input Cin;
  output [3:0] S;
  output Cout;

  assign { Cout, S } = A + B + Cin;
endmodule
```

Module Specification for Lab 1

- The input/output ports of the multiplier is as follows:

```
module SeqMultiplier(
    input wire clk,
    input wire enable,
    input wire [7:0] A,
    input wire [7:0] B,
    output wire [15:0] C
    );
```

‘`clk`’ is the system clock,
‘`enable`’ activates the multiplication operation,
‘`A`’ is the 8-bit unsigned multiplicand input,
‘`B`’ is the 8-bit unsigned multiplier input, and
‘`C`’ is the 16-bit unsigned product output.

Sequential Binary Multiplier Behavior

- ❑ The behavior of sequential binary multiplication of 10111 and 10011 is as follows:

```
         00010111   → multiplicand
  ×      00010011   → multiplier
  ----------------
         00010111
        00010111
       00000000
      00000000
+    00010111
  ----------------
     000110110101   → product
```

  - ■ At each clock cycle, reads one bit from the multiplier and shift-and-adds the multiplicand to the product
- ❑ For digital circuit design, we can perform the process reversely

C Model of the Multiplier

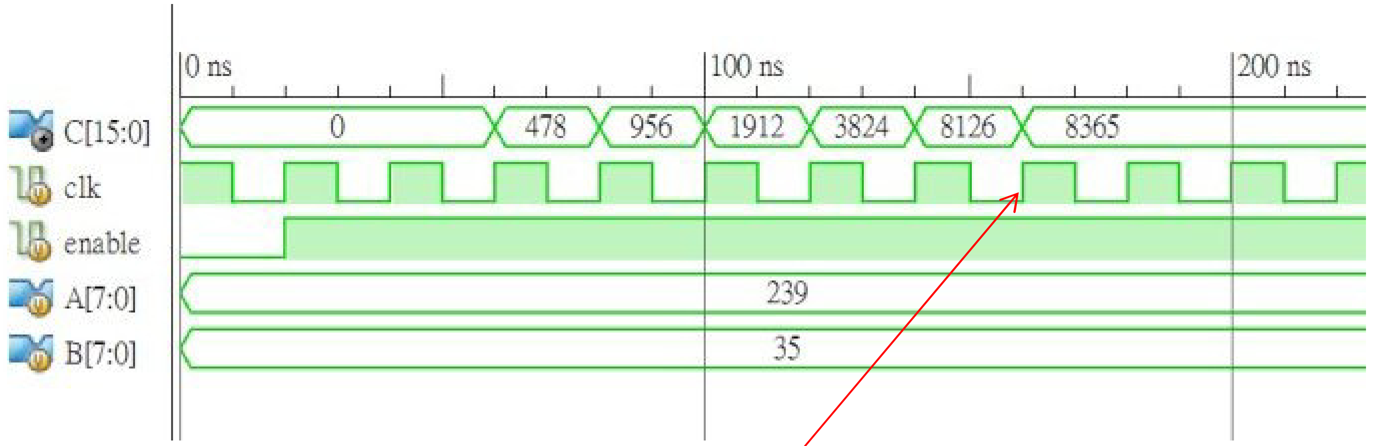
- The C code that performs the sequential multiplication:

```
typedef unsigned char Byte;
Typedef unsigned short Word;

SeqMultiplier(Byte A, Byte B, Word *C)
{
    int idx;

    *C = 0;
    for (idx = 8; idx != 0; idx--)
    {
        *C = *C << 1;
        if ((B & 0x80) == 0x80)
        {
            *C += A;
        }
        B = B << 1;
    }
}
```

Your task for lab1 is to rewrite this C code using Verilog!

Example of Simulated Waveforms

- An example of the timing diagram of 239×35 = 8365



Stable output 8365 happens
8 cycles after enable == 1