

Overview

This tutorial provides simple instruction for using the Xilinx ISE WebPACK toolset for basic development on Digilent system boards. This tutorial will go through the following steps:

- Creating a Xilinx ISE project
- Writing VHDL to create logic circuits and structural logic components
- Creating a User Constraints File (UCF)
- Synthesizing, implementing, and generating a Programming file

More detailed tutorials on the Xilinx ISE tools can be found at
<http://www.xilinx.com/support/techsup/tutorials/>.

Getting Started

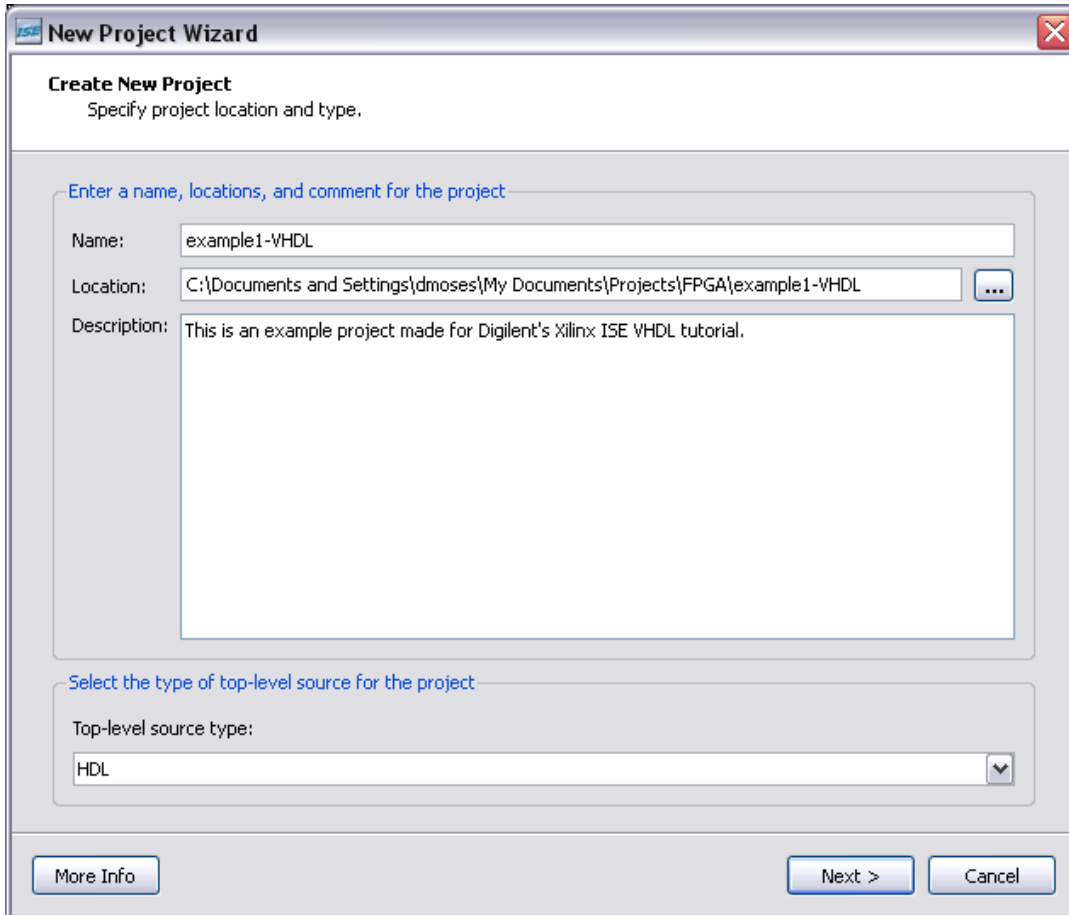
First, install Xilinx ISE WebPACK on your PC or laptop. This tutorial is based on version 11.1. It is available as a free download from www.xilinx.com.

This tutorial uses settings for the Nexys2 500k board, which can be purchased from www.digilentinc.com. The settings for other Digilent system boards can be found there as well.

Starting a New Project

To create a new project, open Project Navigator either from the Desktop shortcut icon or by selecting Start > Programs > Xilinx ISE Design Suite 11 > ISE > Project Navigator. In Project Navigator, select the New Project option from the Getting Started menu (or by selecting Select File > New Project).

This brings up a Dialog box where you can enter the desired project name and project location. You should choose a meaningful name for easy reference. In this tutorial, we call this project “example1-VHDL” and save it in a local directory. You can place comments for your project in the Description text box. We use HDL for our top-level source type in this tutorial.



New Project Wizard

Create New Project
Specify project location and type.

Enter a name, locations, and comment for the project

Name:

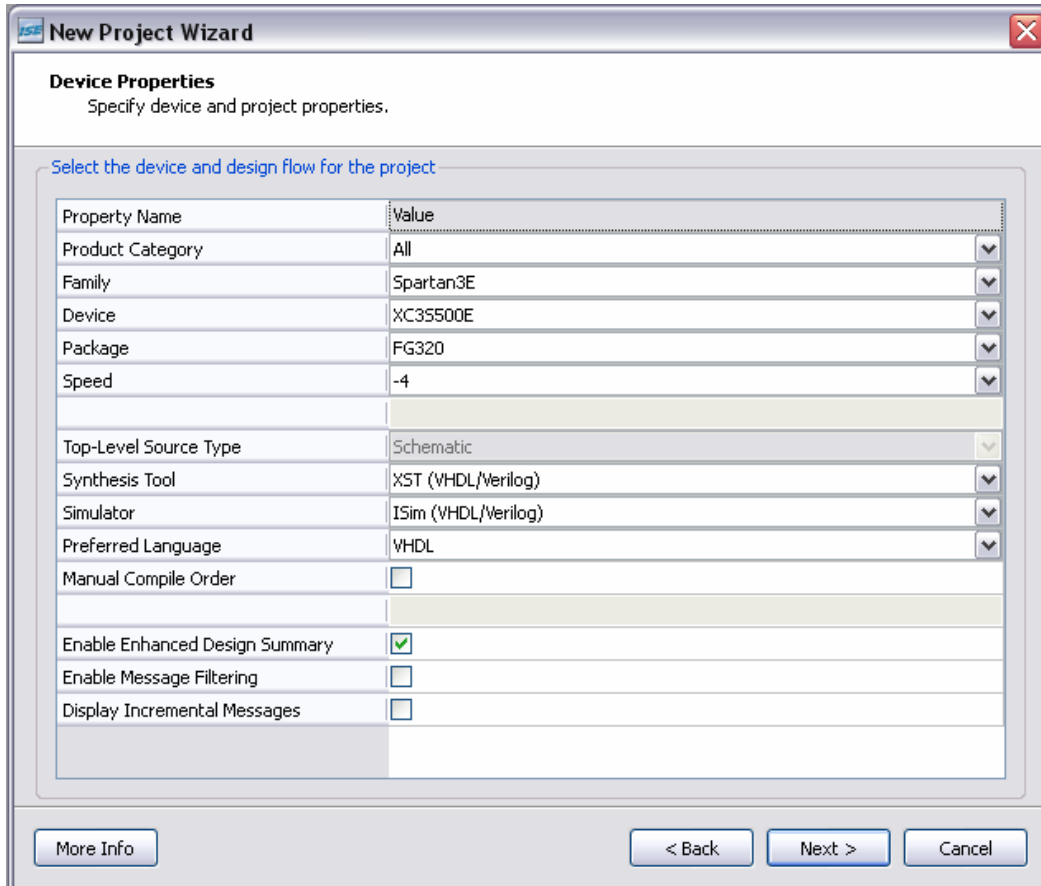
Location: ...

Description:

Select the type of top-level source for the project

Top-level source type:

The next step is to select the proper Family, Device, and Package for your project. This depends on the chip you are targeting for this project. The appropriate settings for a project suited for the Nexys2 500k board are as follows:



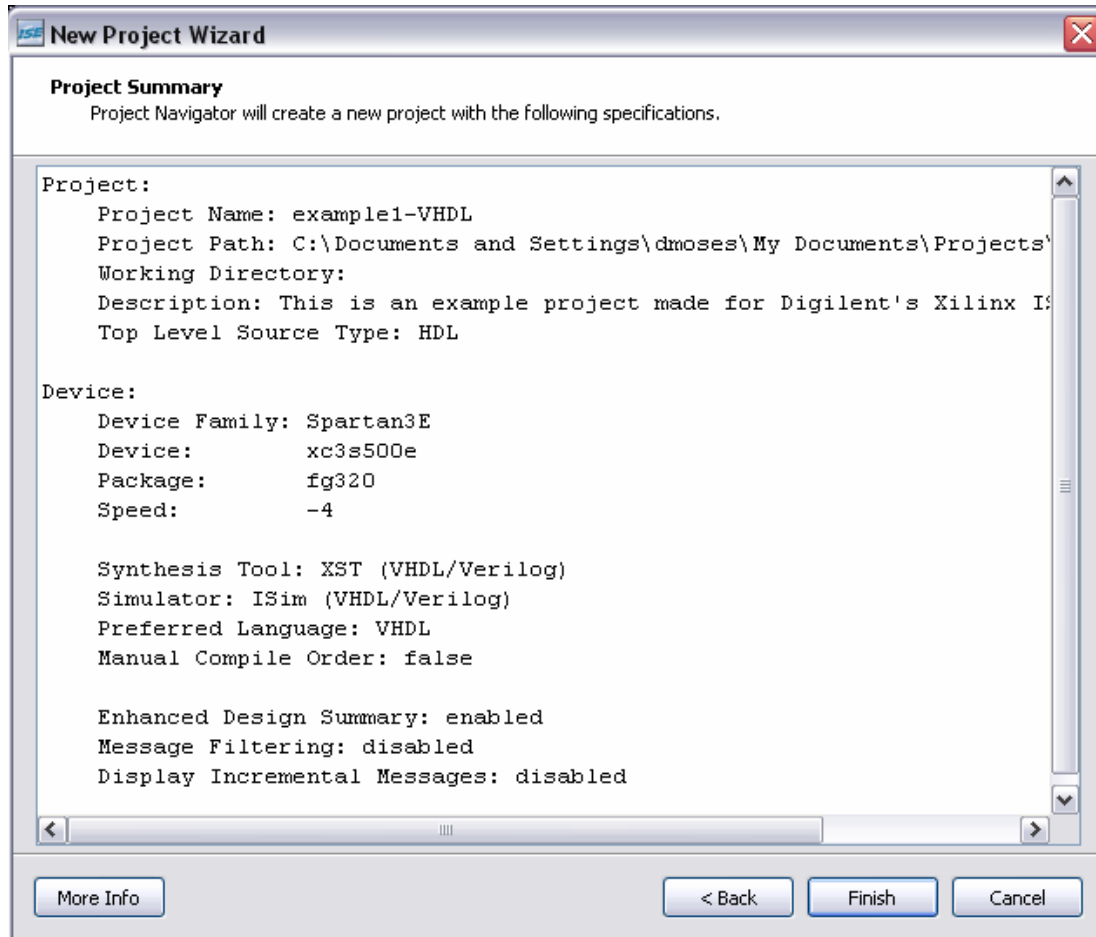
The image shows the 'New Project Wizard' dialog box, specifically the 'Device Properties' step. The title bar says 'New Project Wizard' with a close button. Below the title bar, it says 'Device Properties' and 'Specify device and project properties.' The main area is titled 'Select the device and design flow for the project'. It contains a table with two columns: 'Property Name' and 'Value'. The properties are: Product Category (All), Family (Spartan3E), Device (XC3S500E), Package (FG320), Speed (-4), Top-Level Source Type (Schematic), Synthesis Tool (XST (VHDL/Verilog)), Simulator (ISim (VHDL/Verilog)), Preferred Language (VHDL), Manual Compile Order (checkbox), Enable Enhanced Design Summary (checkbox checked), Enable Message Filtering (checkbox), and Display Incremental Messages (checkbox). At the bottom, there are three buttons: 'More Info', '< Back', 'Next >', and 'Cancel'.

Property Name	Value
Product Category	All
Family	Spartan3E
Device	XC3S500E
Package	FG320
Speed	-4
Top-Level Source Type	Schematic
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Manual Compile Order	<input type="checkbox"/>
Enable Enhanced Design Summary	<input checked="" type="checkbox"/>
Enable Message Filtering	<input type="checkbox"/>
Display Incremental Messages	<input type="checkbox"/>

Once the appropriate settings have been entered, click Next.

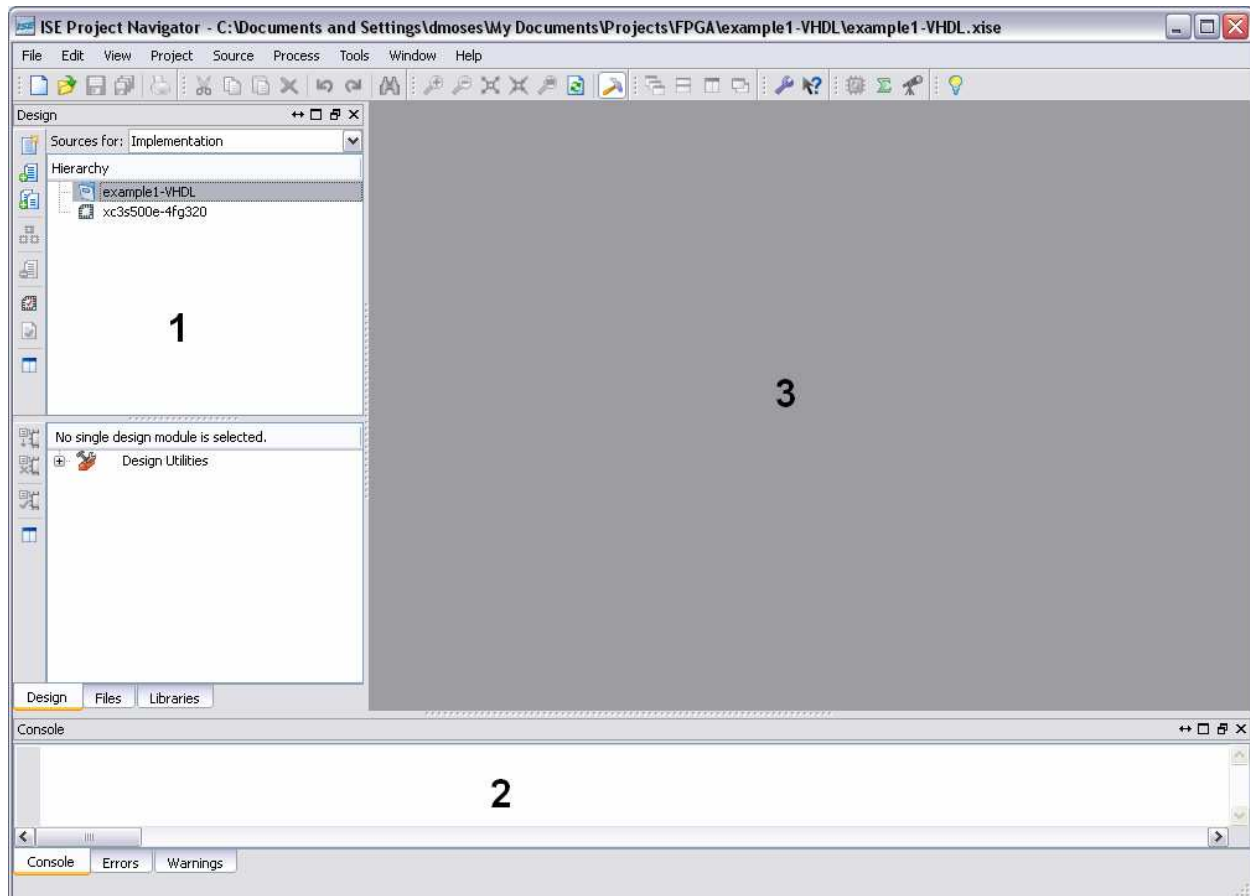
The next two dialog boxes give you the option of adding new or existing source files to your project. Since we will fulfill these steps later, click Next without adding any source files.

Before the new project is created, the New Project Wizard gives you a project summary consisting of the selected specifications you have chosen for the project. Make sure all settings are correct before clicking Finish to end the New Project Wizard. Any modifications to these settings can be made by clicking the Back button.



Project Navigator Overview

Once the new project has been created, ISE opens the project in Project Navigator. Click the Design tab to show the Design panel and click the Console tab to show the Console panel.



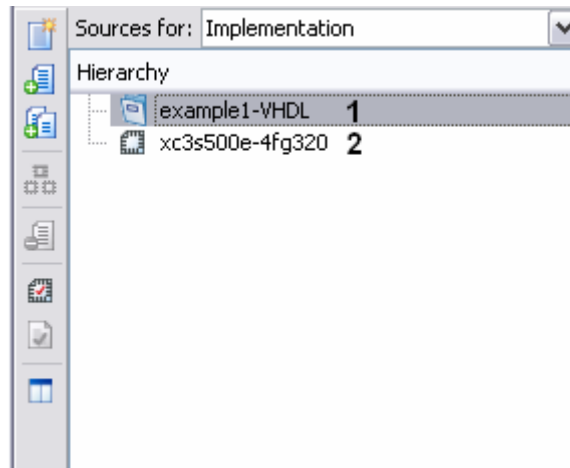
The Design panel (1) contains two windows: a Sources window that displays all source files associated with the current design and a Process window that displays all available processes that can be run on a selected source file.

The Console panel (2) displays status messages including error and warning messages.

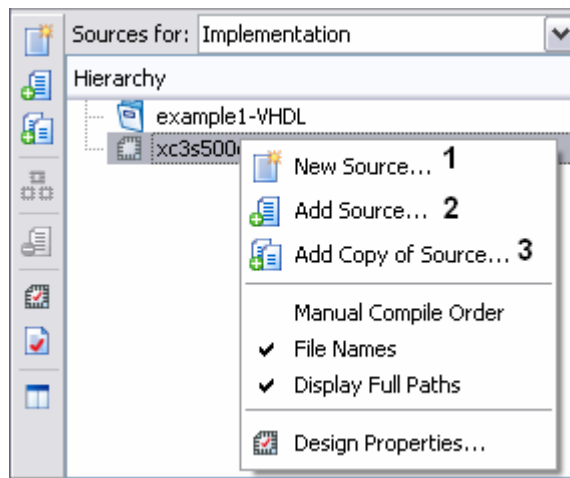
The HDL editor window (3) displays source code from files selected in the Design panel.

Adding New Source Files

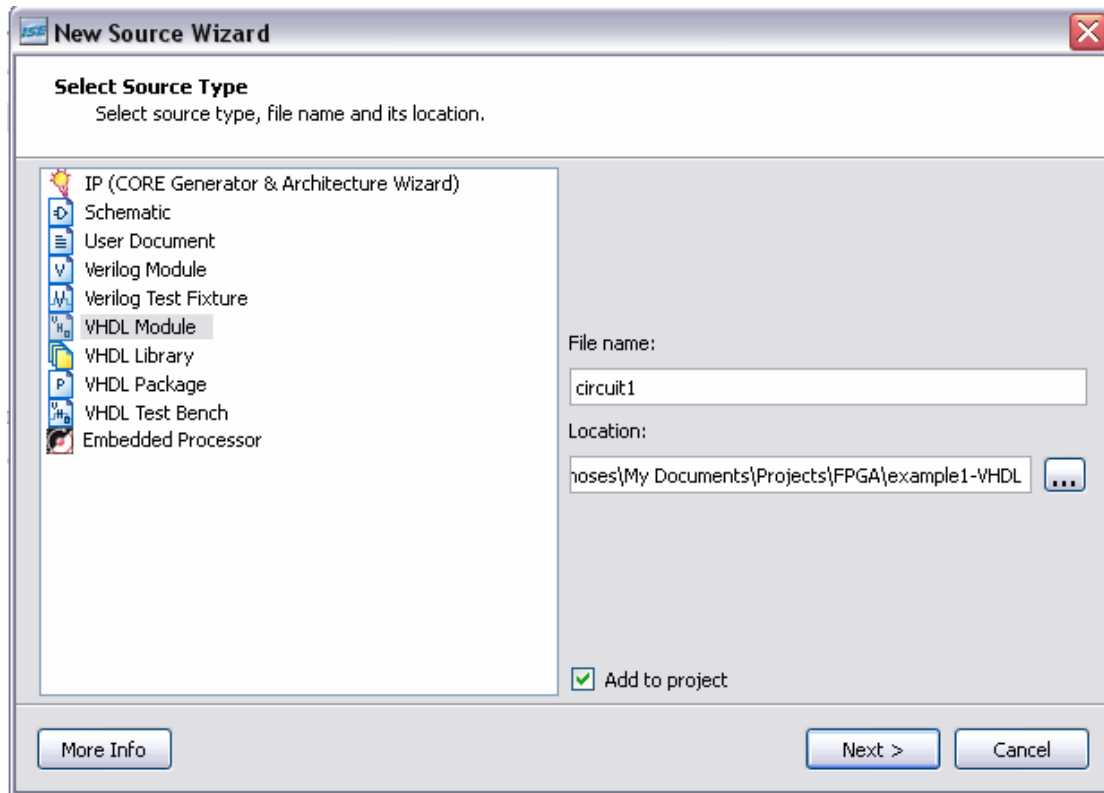
Once the new project is created, two sources are listed under sources in the Design panel: the Project file name and the Device targeted for design.



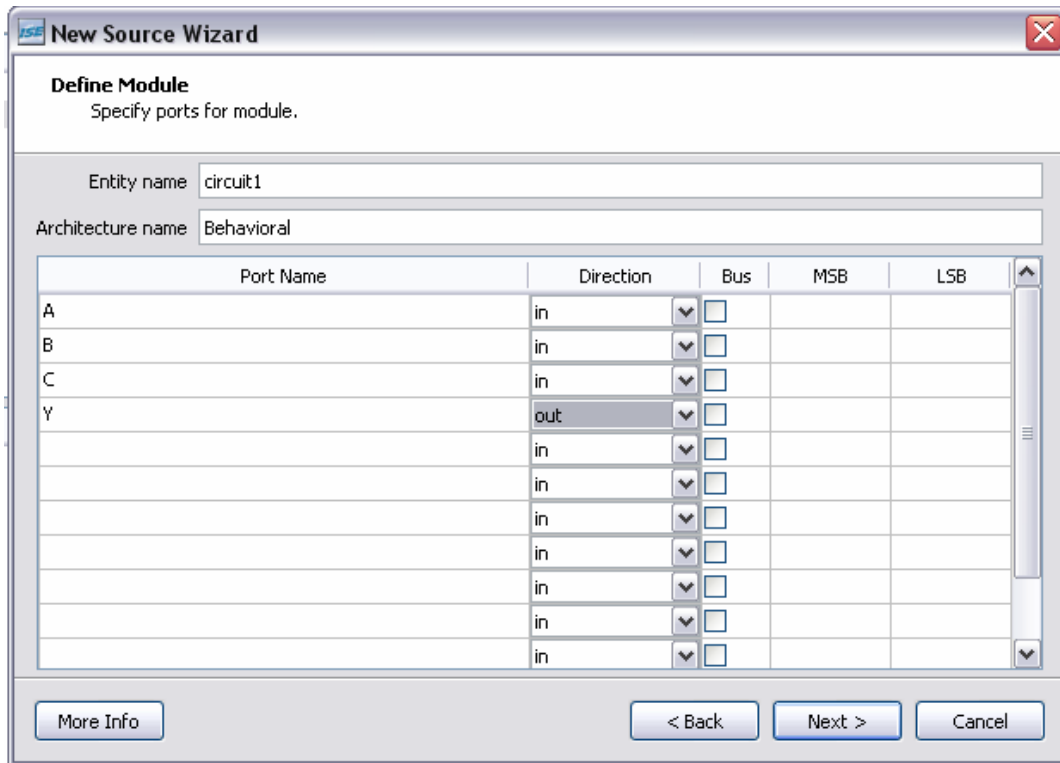
You can add a new or existing source file to the project. To do this, right-click on the target device and select one of the three options for adding source files.



In this tutorial, we will create a new source file, so select New Source from the list. This starts the New Source Wizard, which prompts you for the Source type and file name. Select VHDL module and give it a meaningful name (we name it circuit1).



When you click Next you have the option of defining top-level ports for the new VHDL module. We chose A, B, and C as input ports and Y as an output port.



New Source Wizard

Define Module
Specify ports for module.

Entity name: circuit1

Architecture name: Behavioral

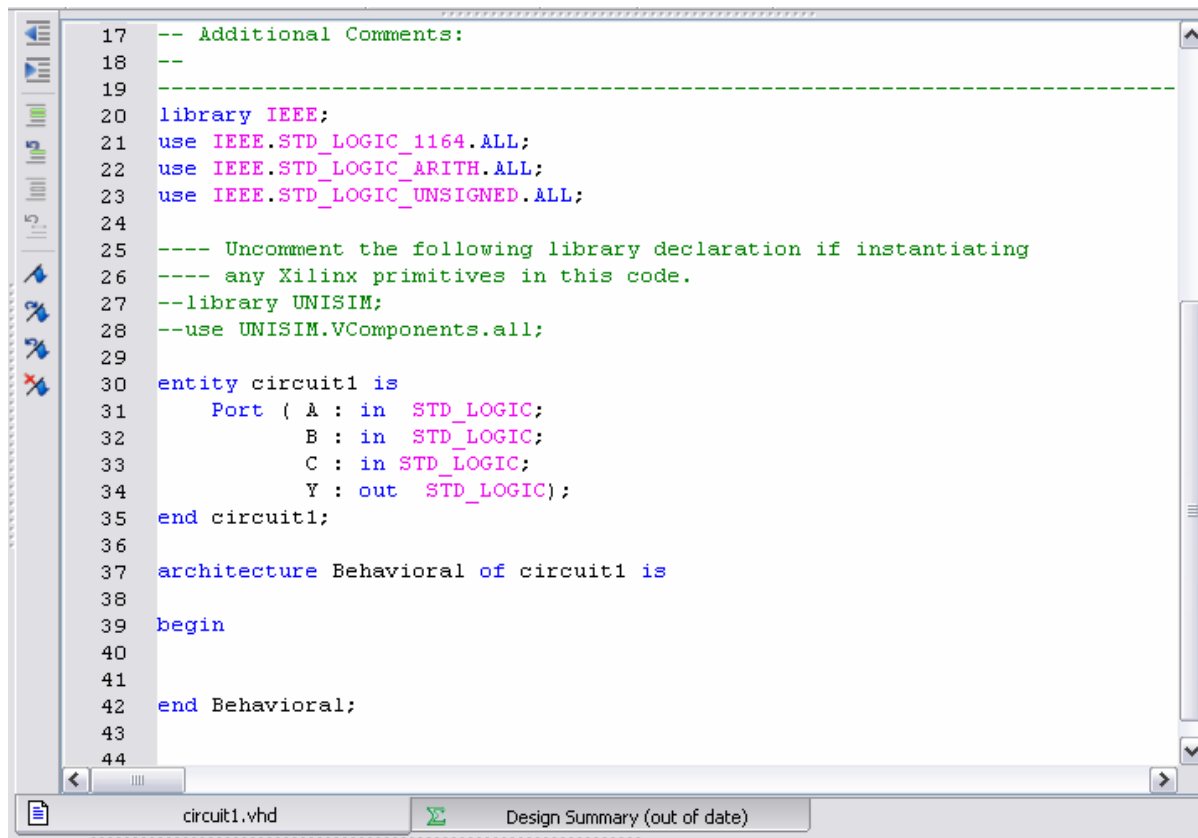
Port Name	Direction	Bus	MSB	LSB
A	in	<input type="checkbox"/>		
B	in	<input type="checkbox"/>		
C	in	<input type="checkbox"/>		
Y	out	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		

More Info < Back Next > Cancel

Click Next and then Finish to complete the VHDL source file creation.

HDL Editor Window

Once you have created the new VHDL file, the HDL Editor Window displays the circuit1.vhd source code.



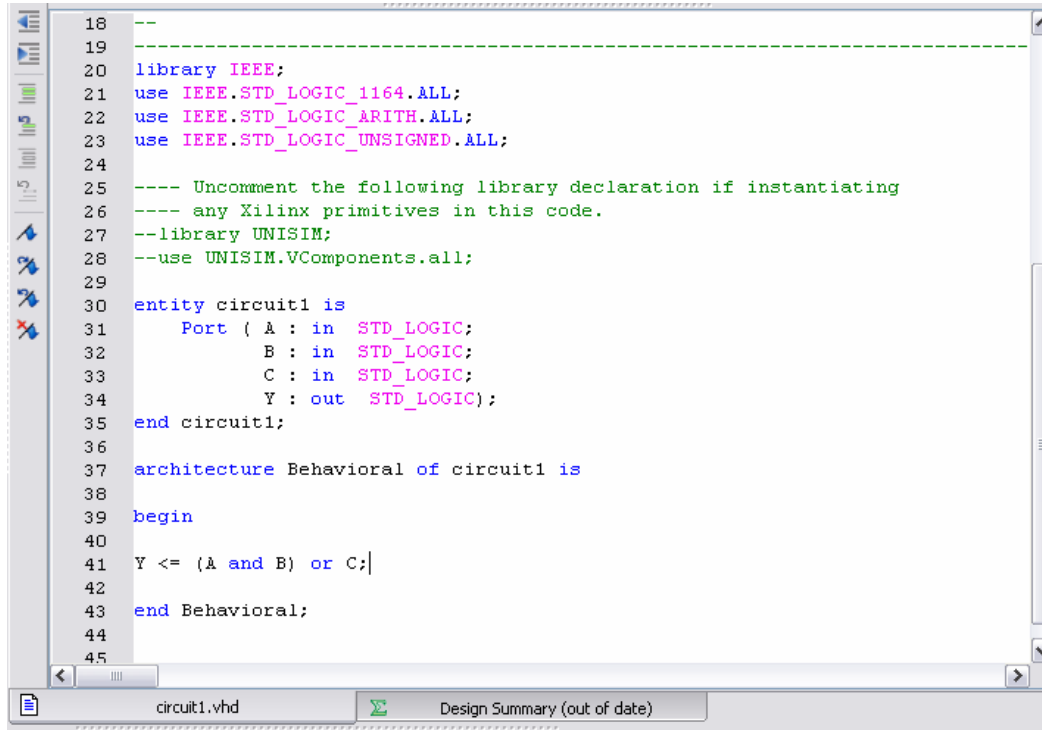
```
17  -- Additional Comments:
18  --
19  -----
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  ---- Uncomment the following library declaration if instantiating
26  ---- any Xilinx primitives in this code.
27  --library UNISIM;
28  --use UNISIM.VComponents.all;
29
30  entity circuit1 is
31      Port ( A : in  STD_LOGIC;
32            B : in  STD_LOGIC;
33            C : in  STD_LOGIC;
34            Y : out STD_LOGIC);
35  end circuit1;
36
37  architecture Behavioral of circuit1 is
38
39  begin
40
41
42  end Behavioral;
43
44
```

ISE automatically generate lines of code in the file to get you started with circuit development. This generated code includes:

- library definitions
- an entity statement
- an architecture statement with begin and end statements included
- a comment block template for documentation.

The actual behavioral or structural description of the given circuit is to be placed between the “begin” and “end Behavioral” statements in the file. Between these statements, you can define any VHDL circuit you wish. In this tutorial we use a simple combinational logic example, and then show how it can be used as a structural component in another VHDL module. We start with the basic logic equation: $Y \leq (A \cdot B) + C$.

The following figure shows the implementation:



```
18  --
19  -----
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  ---- Uncomment the following library declaration if instantiating
26  ---- any Xilinx primitives in this code.
27  --library UNISIM;
28  --use UNISIM.VComponents.all;
29
30  entity circuit1 is
31      Port ( A : in  STD_LOGIC;
32            B : in  STD_LOGIC;
33            C : in  STD_LOGIC;
34            Y : out STD_LOGIC);
35  end circuit1;
36
37  architecture Behavioral of circuit1 is
38
39  begin
40
41  Y <= (A and B) or C;
42
43  end Behavioral;
44
45
```

If there are any syntax errors in the source file, ISE can help you find them. For example, the parentheses around the A and B inputs are crucial to this implementation; without them, ISE would return an error during synthesis.

This VHDL module can also be used as a structural component in a separate project module. To illustrate this, we add another source file called “circuit2.vhd” and give it four input ports and one output port as follows:

```
entity circuit2 is
    Port ( AT : in  STD_LOGIC;
          BT : in  STD_LOGIC;
          CT : in  STD_LOGIC;
          DT : in  STD_LOGIC;
          YT : out STD_LOGIC);
end circuit2;
```

A structural component definition in VHDL must follow this format:

```
component <entity-architecture name>  
    Port(<input/output port definitions>);  
end component;
```

In the new circuit2.vhd file, we define the previously created entity, circuit1, as a structural component:

```
component circuit1  
    Port ( A : in   STD_LOGIC;  
          B : in   STD_LOGIC;  
          C : in   STD_LOGIC;  
          Y : out  STD_LOGIC);  
end component;
```

Structural VHDL often requires the use of signals. Signals are a specific class of object in a VHDL model that can be specified to a given type. A signal definition in VHDL must follow this format:

```
signal <signal name> : <signal type>:= <initialization value, usually '0'>;
```

In this tutorial, we define a signal called “wiresig”. The example definition is as follows:

```
signal wiresig : std_logic:= '0';
```

The instantiation of a structural component in VHDL must follow this format:

```
<label>: <component entity-architecture name> port map(<component I/O>=> <top level I/O or signals>);
```

Instantiation of the structural component “circuit1” is as follows:

```
ex1: circuit1 port map (A=>AT,  
                        B=>BT,  
                        C=>wiresig,  
                        Y=>YT);
```

To finish defining the simple circuit we add this statement:

```
wiresig<= CT and DT;
```

The finished definition of the circuit $Y \leq (A \cdot B) + (C \cdot D)$ is as follows:

```
entity circuit2 is
    Port ( AT : in  STD_LOGIC;
          BT : in  STD_LOGIC;
          CT : in  STD_LOGIC;
          DT : in  STD_LOGIC;
          YT : out STD_LOGIC);
end circuit2;

architecture Behavioral of circuit2 is

    component circuit1
        Port ( A : in  STD_LOGIC;
              B : in  STD_LOGIC;
              C : in  STD_LOGIC;
              Y : out STD_LOGIC);
    end component;

    signal wiresig : std_logic := '0';

begin
    ex1: circuit1 port map (A=>AT,
                           B=>BT,
                           C=>wiresig,
                           Y=>YT);

    wiresig<=CT and DT;

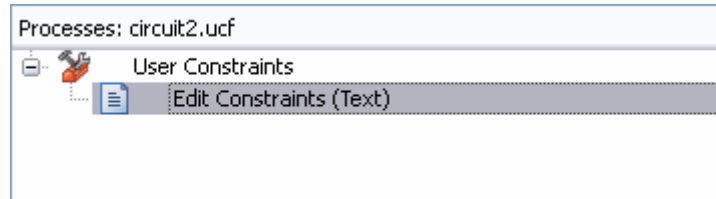
end Behavioral;
```

UCF File Creation

The Xilinx tools use a User Constraints File (.ucf file) to define user constraints like physical pin to circuit net mappings. This is sometimes referred to as an Implementation Constraints File. The .ucf file can be modified inside ISE using a text editor.

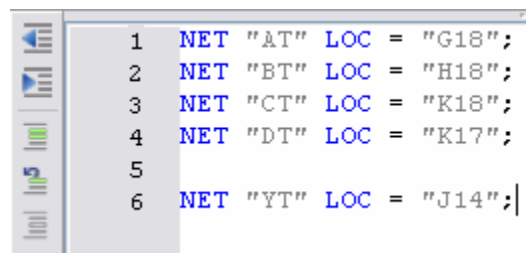
To add a .ucf file to your design, go to the Sources window and right-click the source file that requires user constraints. Select the Add New Source option in the drop-down menu. The New Source Wizard prompts you for the Source type and file name. Select Implementation Constraints File and give it a meaningful name (we'll name it circuit2).

To edit the .ucf file, select it in the sources window, expand the User Constraints option in the Processes window below, and double-click the Edit Constraints (Text) option. A blank text editor appears.



To associate a physical pin with a given net name, type: `NET "netname" LOC = "XXX";` on a line in the .ucf file. In the statement, "netname" (quotes included) is the name of the net to attach to pin number XXX (quotes included).

For our example project, the four inputs are assigned to switches 0 through 3 and the output is assigned to LED0 on the Nexys2 board. The finished .ucf file is as follows:

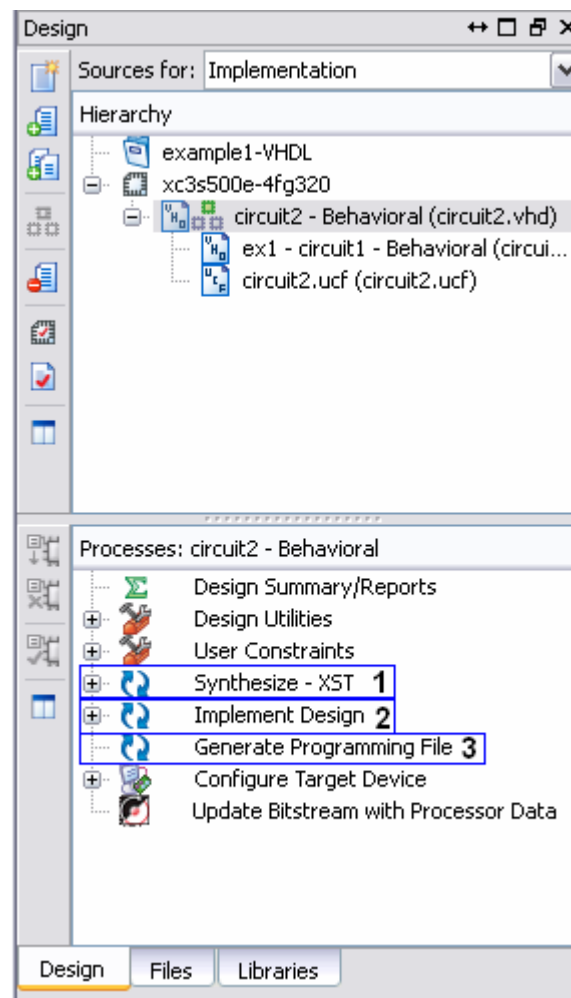


```
1 NET "AT" LOC = "G18";
2 NET "BT" LOC = "H18";
3 NET "CT" LOC = "K18";
4 NET "DT" LOC = "K17";
5
6 NET "YT" LOC = "J14";
```

Programming File Generation

Now we are ready to create a programming file (.bit) for the Nexys2 FPGA.

Go to the Sources window and select the top-level module (indicated by the three blocks shown with the source name.)



Now go to the Processes window where there are three particular processes in a row:

1. Synthesize – XST
2. Implement Design
3. Generate Programming file

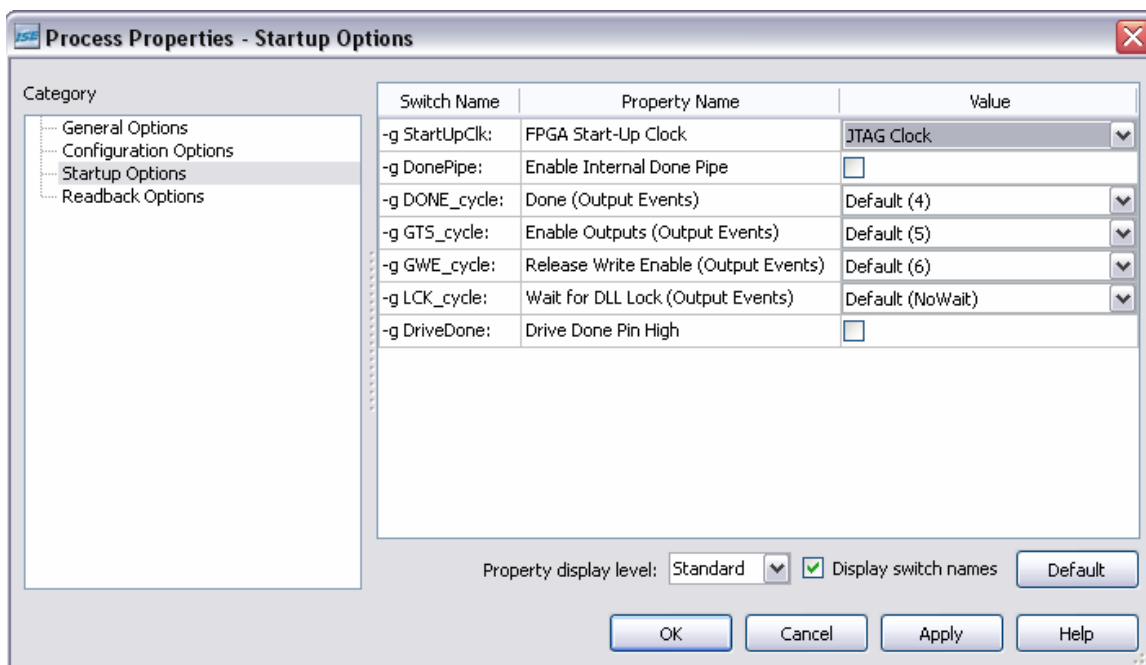
Run the synthesis process by either double-clicking on Synthesize or left clicking and selecting the run option. This process analyzes the circuit you have created, checking for valid connections, syntax, and structure, to verify that the circuit is valid and synthesizable.

If the Synthesize process does not return any errors, you can move on and run the Implement Design process. This process uses various algorithms to map out the digital circuit and then creates place and route information so that it can be placed on the physical FPGA.

Startup Clock Options

If the Implement Design process does not return any errors, you can run the Generate Programming File process. Before we do this, right-click on the Generate Programming File process and select Process Properties. In the category pane of the Process Properties window, select Startup Options.

The first item in the right panel is the FPGA Start-Up Clock property. This option allows a configuration file to either configure a board straight from the PC, or load a configuration from platform flash memory on the board. To configure the board from the PC, the start-up clock value should be JTAG Clock. To configure the board from platform flash, the start-up clock value should be CCLK.



After selecting the appropriate start-up clock value, click OK and run the Generate Programming File process. After this process completes, a configuration .bit file should appear in the directory where your project is located.

Board Configuration

The configuration .bit file that has been generated can now be used by Digilent's Adept software to configure a Digilent system board. See *Adept Software Basic Tutorial* for further information.