XESS Corporation

Introduction to WebPACK 8.1

Using Xilinx WebPACK Software to Create FPGA Designs for the XSA Board

Release date: 5/16/2005

Table of Contents

0

What This Is and *Is Not*

There are numerous requests on newgroups that go something like this:

"I am new to using programmable logic like FPGAs and CPLDs. How do I start? Is there a tutorial and some free tools I can use to learn more?"

XILINX has released their WebPACK on the web so that anyone can download a free set of tools for CPLD and FPGA-based logic designs. And XESS Corp. has written this tutorial that attempts to give you a gentle introduction to using the WebPACK tools. (Other programmable logic manufacturers have also released free toolsets. Someone else will have to write a tutorial for them.)

This tutorial shows the use of the WebPACK tools on two simple design examples: 1) an LED decoder and 2) a counter, which displays its current value on a seven-segment LED. Along the way, you will see:

- How to start an FPGA project.
- How to target a design to a particular type of FPGA.
- How to describe a logic circuit using VHDL and/or schematics.
- How to detect and fix VHDL syntactical errors.
- How to synthesize a netlist from a circuit description.
- How to fit the netlist into an FPGA.
- How to check device utilization and timing for an FPGA.
- How to generate a bitstream for an FPGA.
- How to download a bitstream into an FPGA.
- How to test the programmed FPGA.

That said, it is important to say what this tutorial will not teach you:

- It will not teach you how to design logic with VHDL.
- It will not teach you how to choose the best type of FPGA or CPLD for your design.

- It will not teach you how to arrange your logic for the most efficient use of the resources in an FPGA.
- It will not teach you what to do if your design doesn't fit in a particular FPGA.
- It will not show you every feature of the WebPACK software and discuss how to set every option and property.
- It will not show you how to use the variety of peripheral devices available on the XSA Boards.

In short, this is just a tutorial to get you started using the XILINX WebPACK FPGA tools. After you go through this tutorial you should be able to move on to more advanced topics.

1

FPGA Programming

Implementing a logic design with an FPGA usually consists of the following steps (depicted in the figure which follows):

1. You enter a description of your logic circuit using a *hardware description language* (HDL) such as VHDL or Verilog. You can also draw your design using a schematic editor.

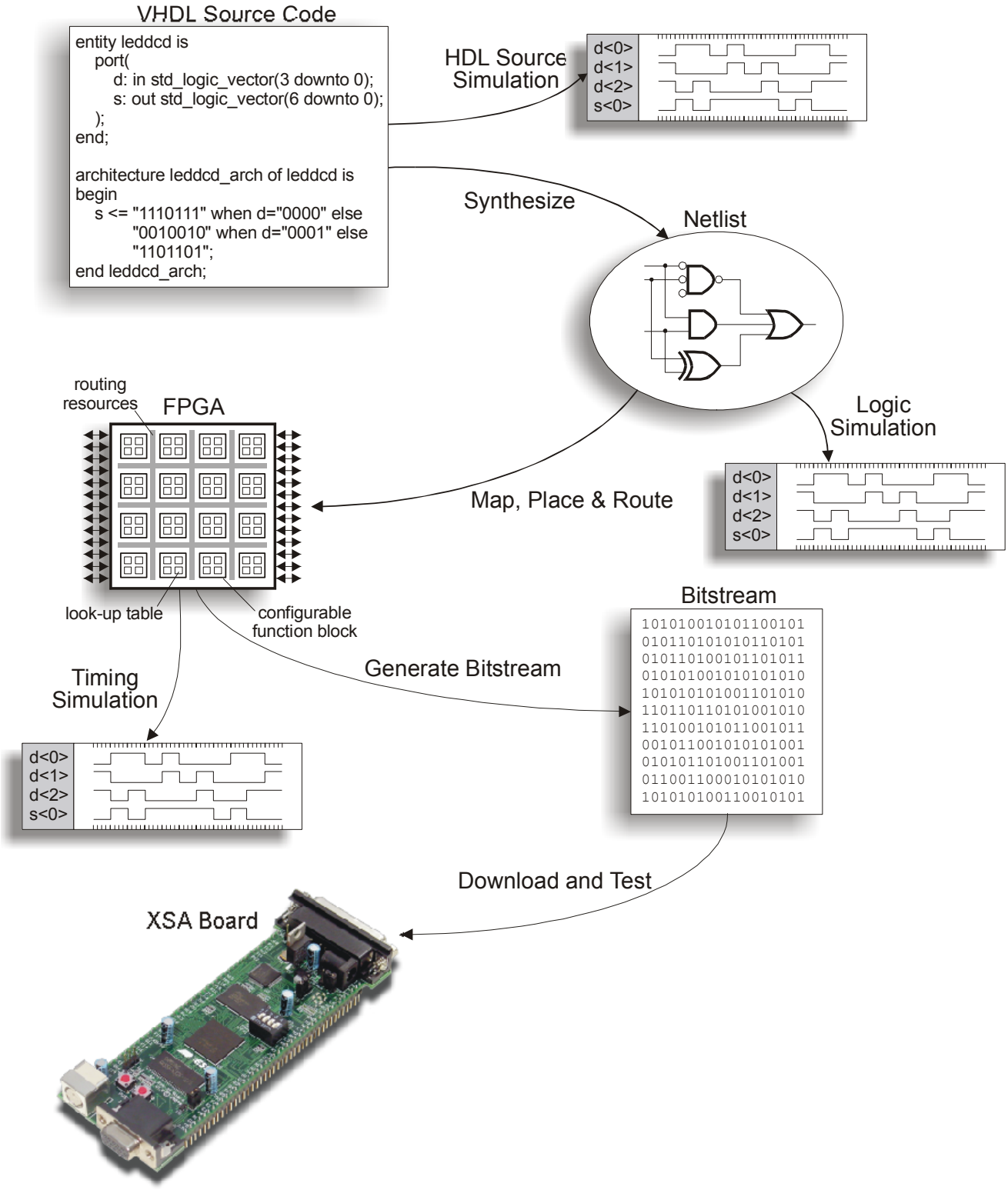
2. You use a *logic synthesizer* program to transform the HDL or schematic into a *netlist*. The netlist is just a description of the various logic gates in your design and how they are interconnected.

3. You use the *implementation tools* to map the logic gates and interconnections into the FPGA. The FPGA consists of many *configurable logic blocks*, which can be further decomposed into *look-up tables* that perform logic operations. The CLBs and LUTs are interwoven with various *routing resources*. The mapping tool collects your netlist gates into groups that fit into the LUTs and then the place & route tool assigns the groups to specific CLBs while opening or closing the switches in the routing matrices to connect them together.

4. Once the implementation phase is complete, a program extracts the state of the switches in the routing matrices and generates a *bitstream* where the ones and zeroes correspond to open or closed switches. (This is a bit of a simplification, but it will serve for the purposes of this tutorial.)

5. The bitstream is *downloaded* into a physical FPGA chip (usually embedded in some larger system). The electronic switches in the FPGA open or close in response to the binary bits in the bitstream. Upon completion of the downloading, the FPGA will perform the operations specified by your HDL code or schematic.

That's really all there is to it. XILINX WebPACK provides the HDL and schematic editors, logic synthesizer, fitter, and bitstream generator software. The XSTOOLs from XESS provide utilities for downloading the bitstream into the FPGA on the XSA Board.

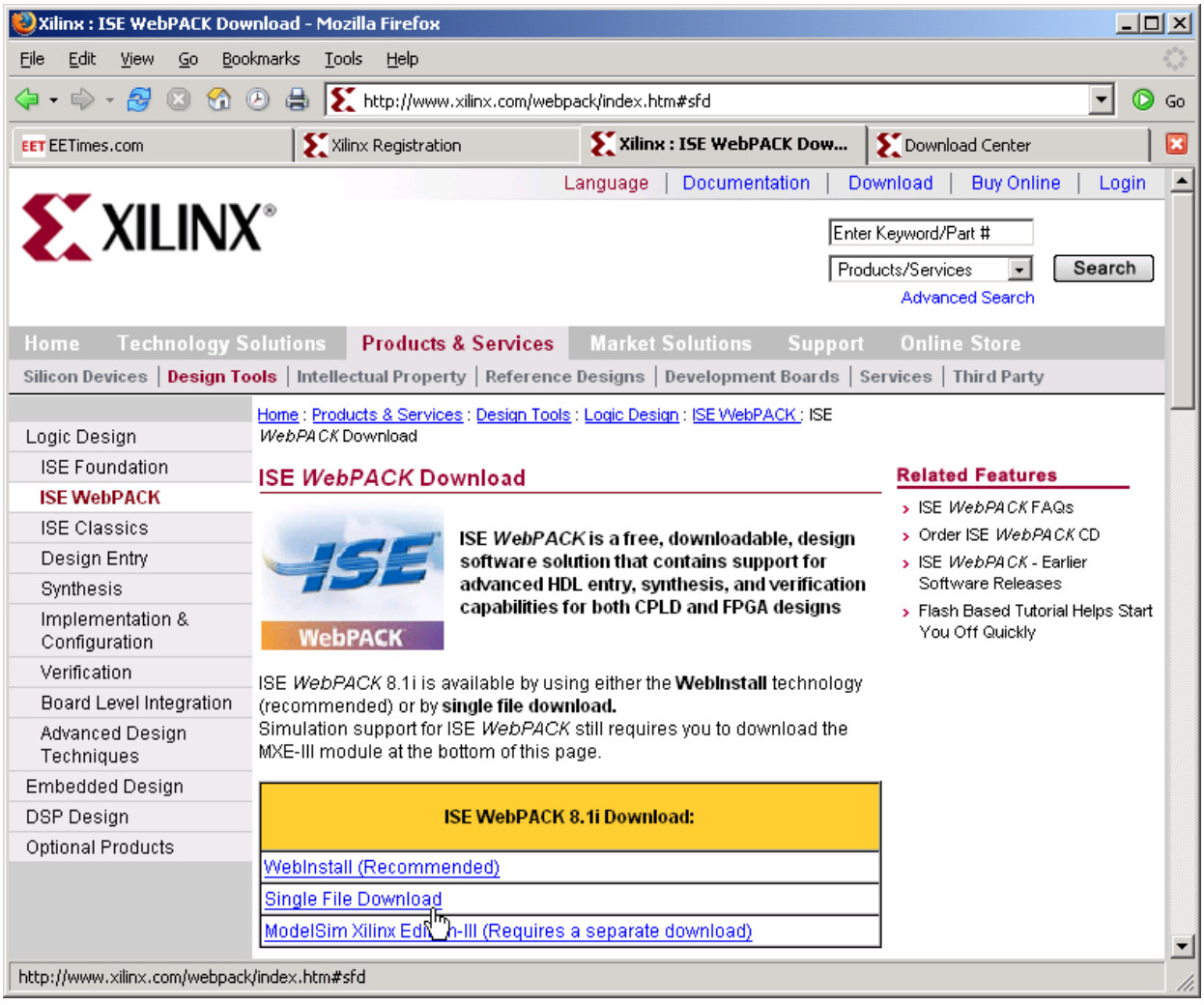

VHDL Source Code
entity leddcd is
port(
d: in std_logic_vector(3 downto 0);
s: out std_logic_vector(6 downto 0);
);
end;
architecture leddcd_arch of leddcd is
begin
s <= "1110111" when d="0000" else
"0010010" when d="0001" else
"1101101";
end leddcd_arch;
HDL Source Simulation
d<0>
d<1>
d<2>
s<0>
Synthesize
Netlist
routing resources
FPGA
look-up table
configurable function block
Map, Place & Route
Logic Simulation
d<0>
d<1>
d<2>
s<0>
Bitstream
101010010101100101
010110101010110101
010110100101101011
010101001010101010
101010101001101010
110110110101001010
110100101011001011
001011001010101001
010101101001101001
011001100010101010
101010100110010101
Generate Bitstream
Timing Simulation
d<0>
d<1>
d<2>
s<0>
Download and Test
XSA Board

2

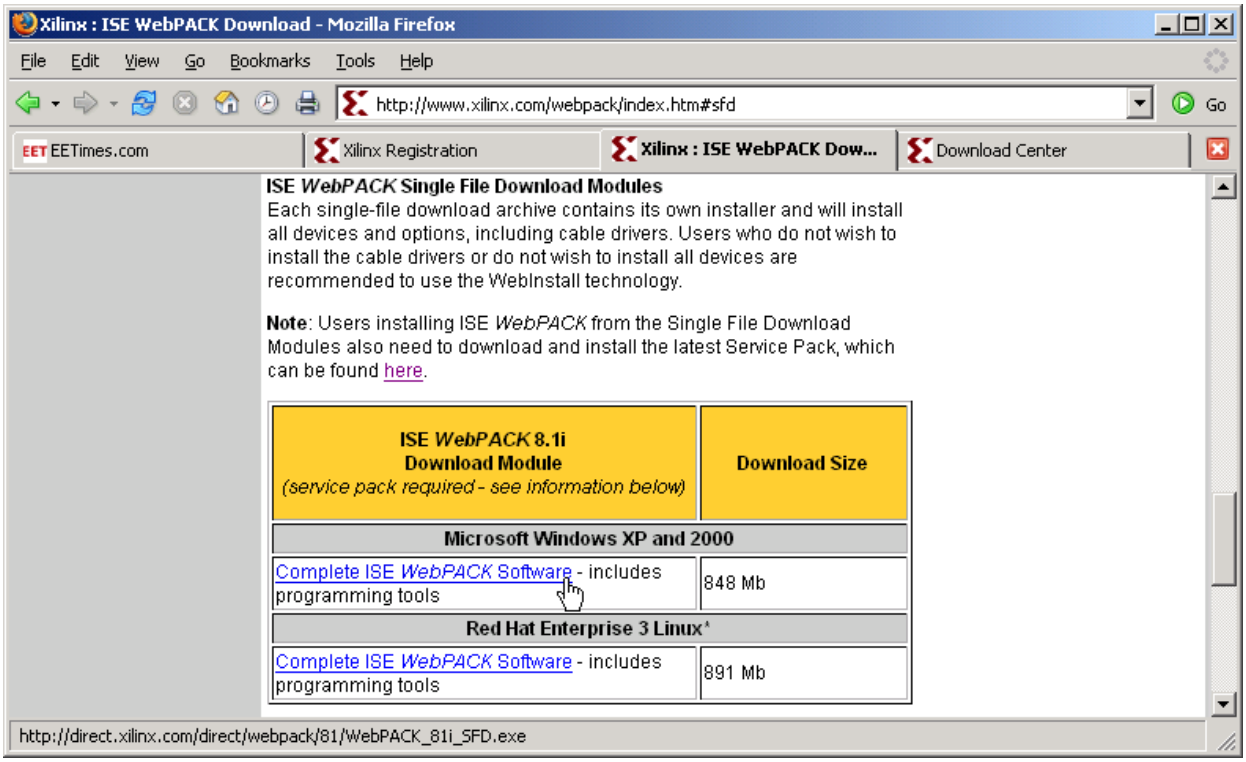
Installing WebPACK

Getting WebPACK

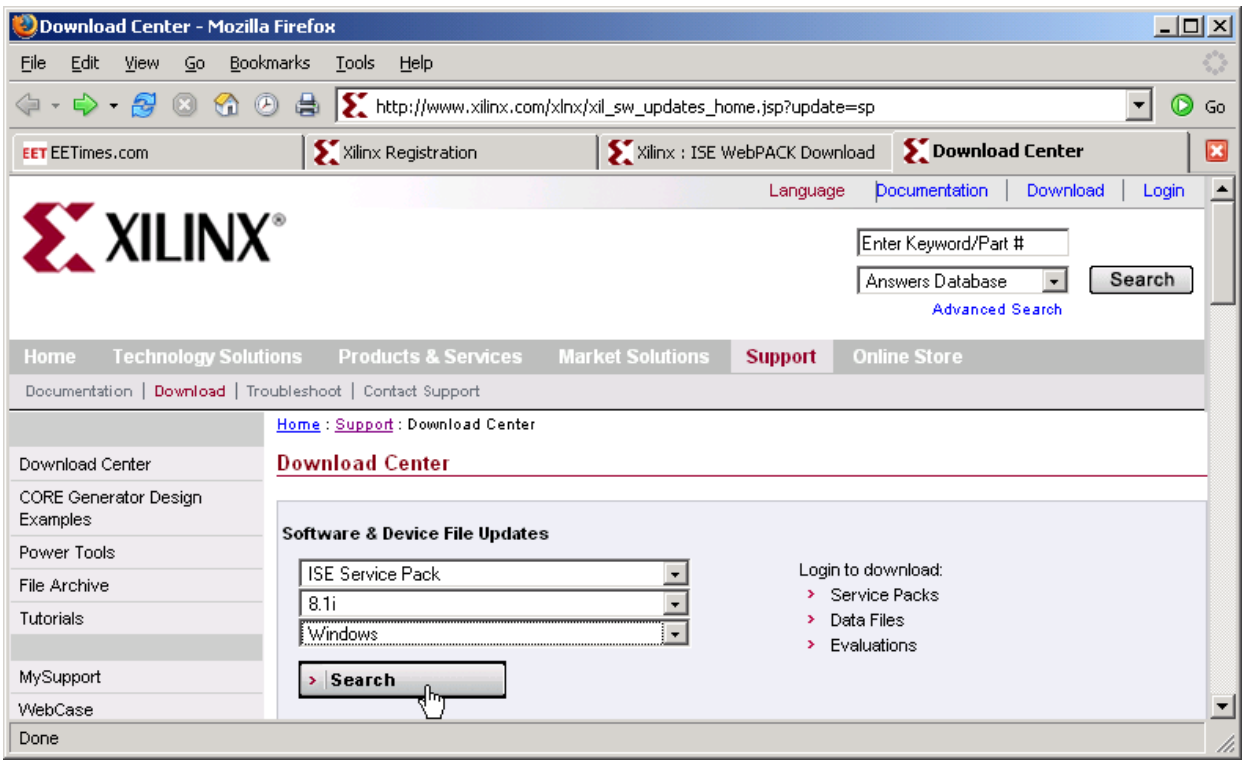
Before downloading the WebPACK software, you will have to create an account. You will choose a user ID and password and then you will be allowed to enter the site. Then you can go to http://www.xilinx.com/webpack/index.htm to begin downloading the WebPACK software. After entering the WebPACK homepage, click on the Single File Download link as shown below.



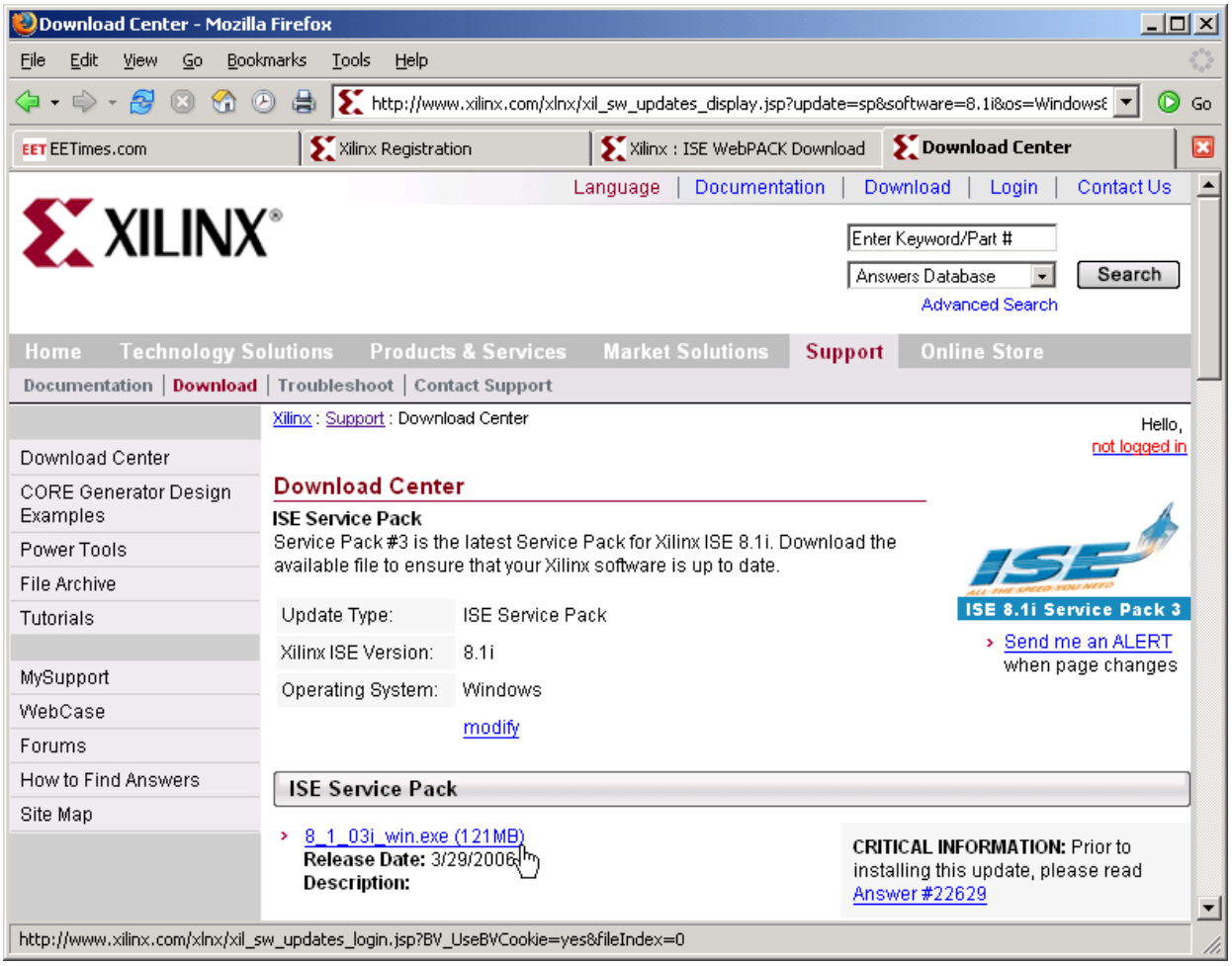
Next, click on the Complete ISE WebPACK Software link. This will initiate the download of all the WebPACK software modules that cover both FPGA and CPLD designs.



After the WebPACK download completes, you also need to download the latest ISE service pack to get the most current WebPACK updates. Set the search parameters as shown below and click Search.



Then select the ISE service pack for download. (There are service packs for other Xilinx software as well, but you don't need them right now.)



Installing WebPACK

After the WebPACK software download completes, double-click the WebPACK_81i_SFD.exe file. The installation script will run and install the software. Accept the default settings for everything and you shouldn't have any problems. Then update the WebPACK software by running the service pack install file 8_1_03i_win.exe.

Getting XSTOOLs

If you are going to download your FPGA bitstreams into an XSA Board, then you will need to get the XSTOOLS software from http://www.xess.com/ho07000.html. Just download the setup-XSTOOLs-4_0_6.exe file.

Installing XSTOOLs

Double-click the setup-XSTOOLs-4_0_6.exe file. The installation script will run and install the software. Accept the default settings for everything.

Getting the Design Examples

You can download the project files for the designs shown in this tutorial from http://www.xess.com/projects/webpack-8_1-xsa.zip. The ZIP file contains versions of the project files for each XSA Board model.

3

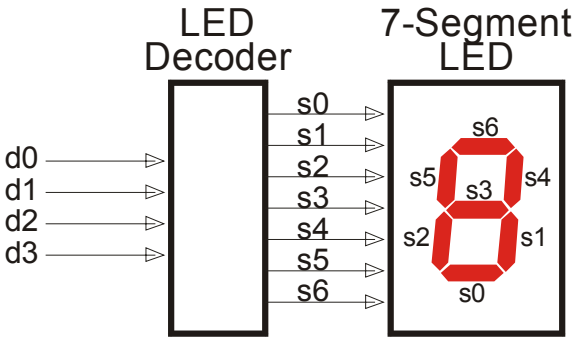
Our First Design

An LED Decoder

The first FPGA design you will try is an LED decoder. An LED decoder takes a four-bit input and outputs seven signals, which drive the segments of an LED digit. The LED segments will be driven to display the digit corresponding to the hexadecimal value of the four input bits as follows:

Four-bit Input	Hex Digit	LED Display
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	A
1011	B	b
1100	C	c
1101	D	d
1110	E	E
1111	F	F

A high-level diagram of the LED decoder looks like this:



Starting WebPACK Project Navigator

You start WebPACK by double-clicking the 

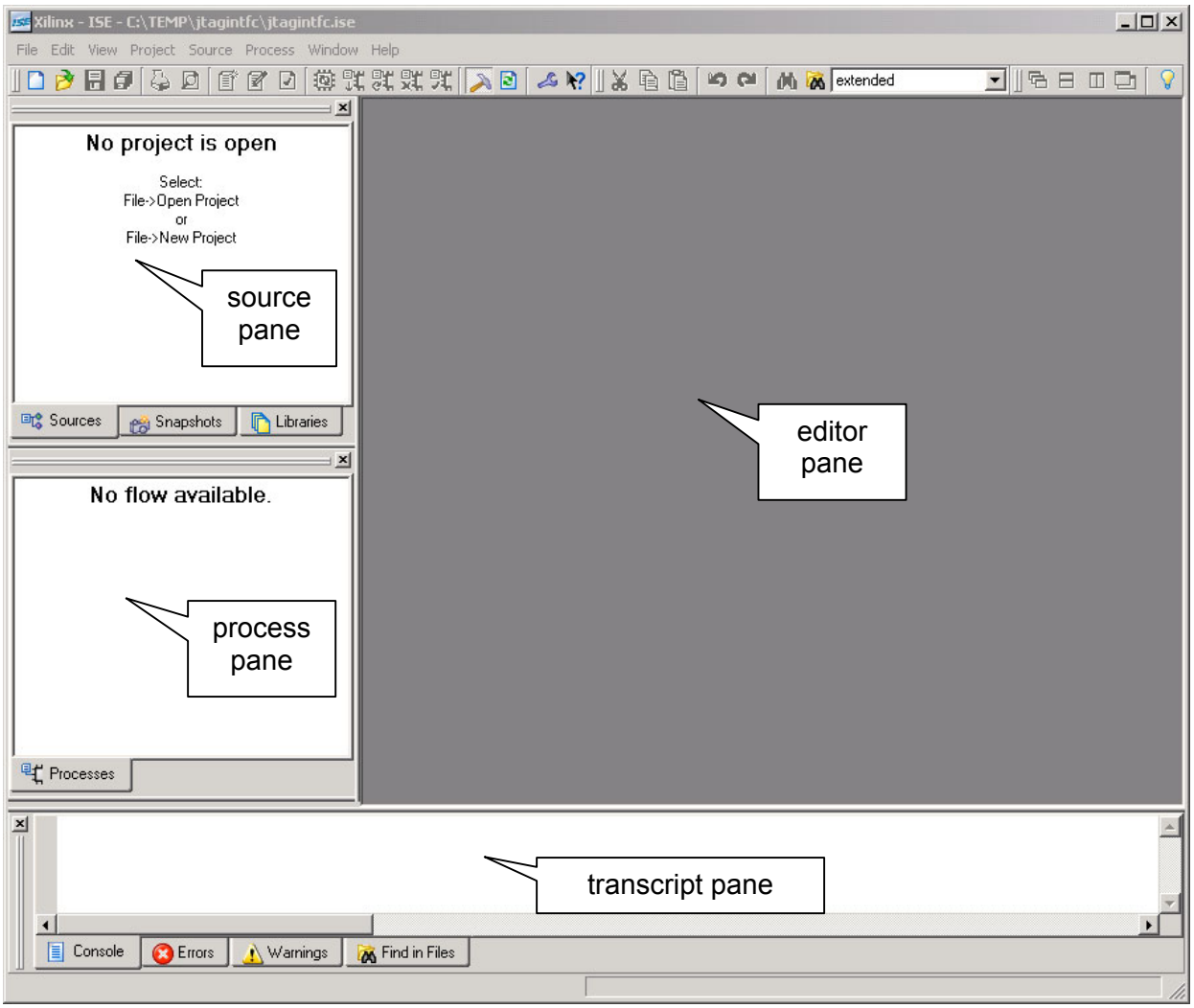
icon on the desktop. This will bring up an empty project window as shown below. The window has four panes:

1. A **source pane** that shows the organization of the source files that make up your design. There are three tabs so you can view the functional modules, or HDL libraries for your project, or look at various snapshots of the project.

2. A **process pane** that lists the various operations you can perform on a given object in the source pane.

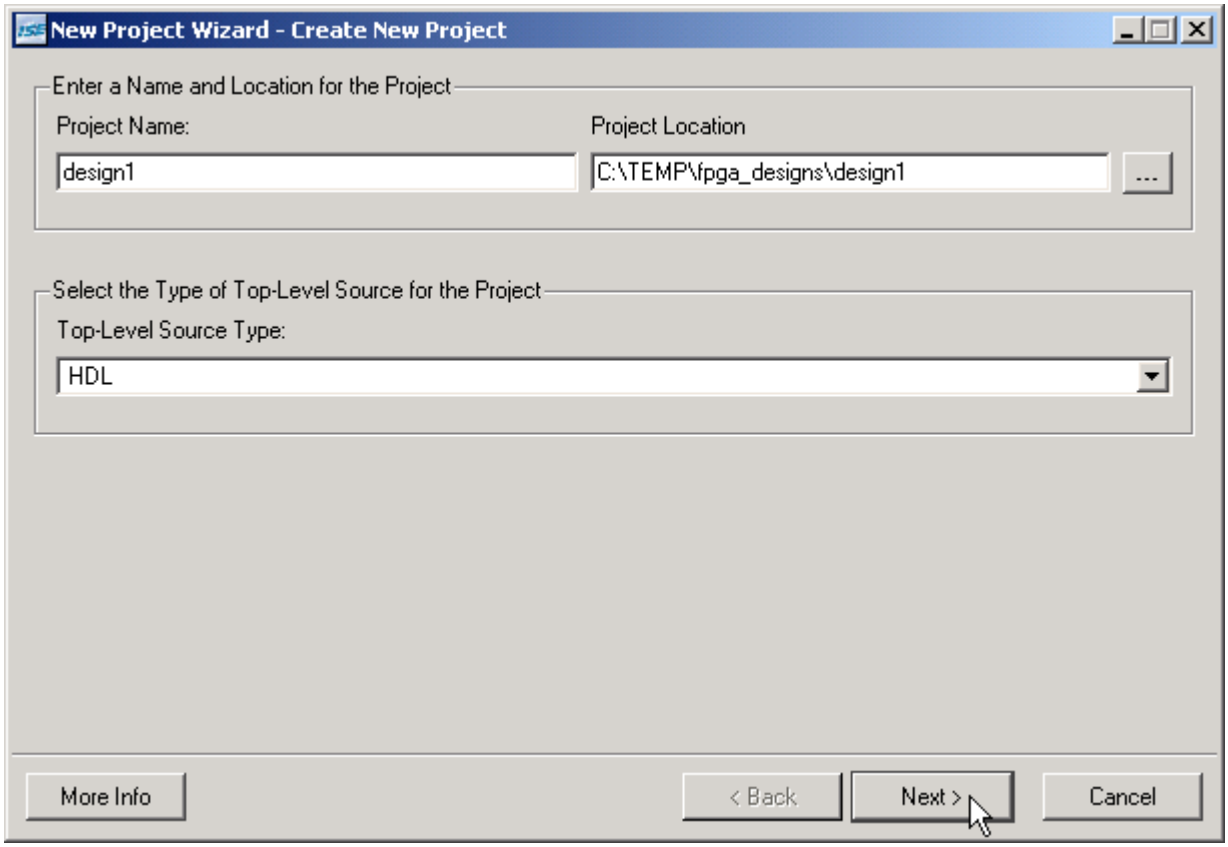
3. A **transcript pane** that displays the various messages from the currently running process.

4. An **editor pane** where you can enter HDL code, schematics, state diagrams, etc.



To start your design, create a new project by selecting the File➔New Project item from the menu bar. This brings up the **New Project Wizard** window where you can enter the name of your

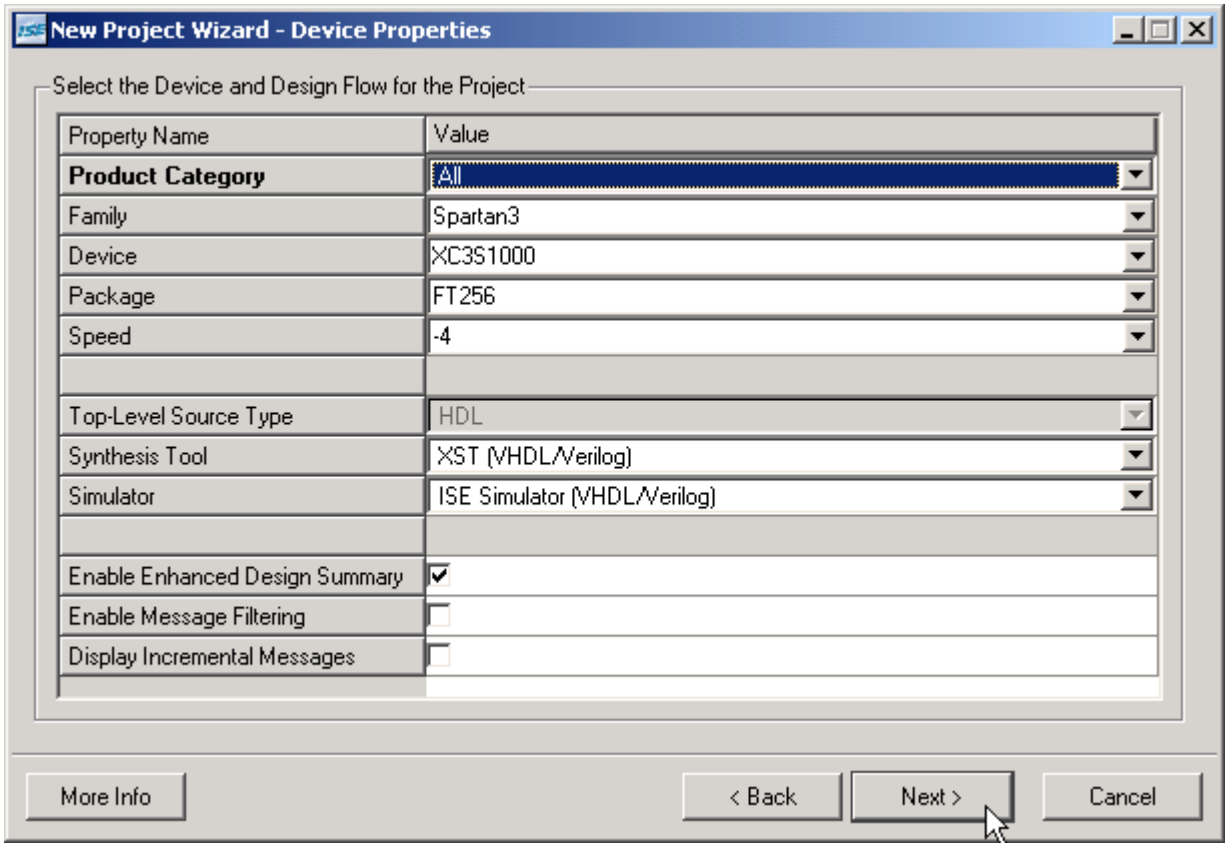
project, the location of your project files, and the style in which you will describe your design at the top level. I have given my project the descriptive name of ***design1*** and will place the files in the C:\TEMP\fpga_designs\design1 folder. (You may choose differently.) I am going to describe the LED decoder using VHDL, so I have set the top-level type (there will only be one level) to HDL (which would also be used if the design was done with Verilog). Click Next to continue creating your project.



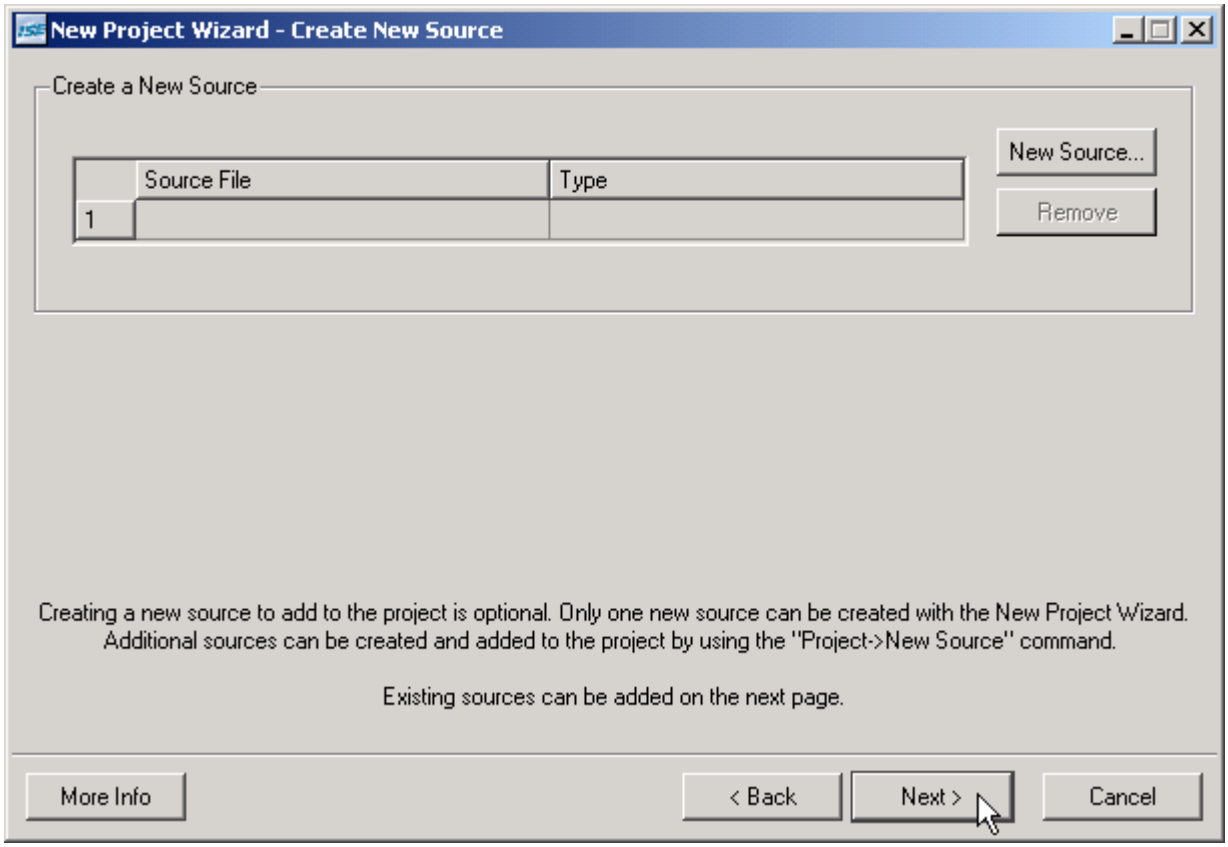
Now you need to tell WebPACK what FPGA you are going to use for your design. The device family, family member, package and speed grade for the FPGA on each model of XSA Board are shown below.

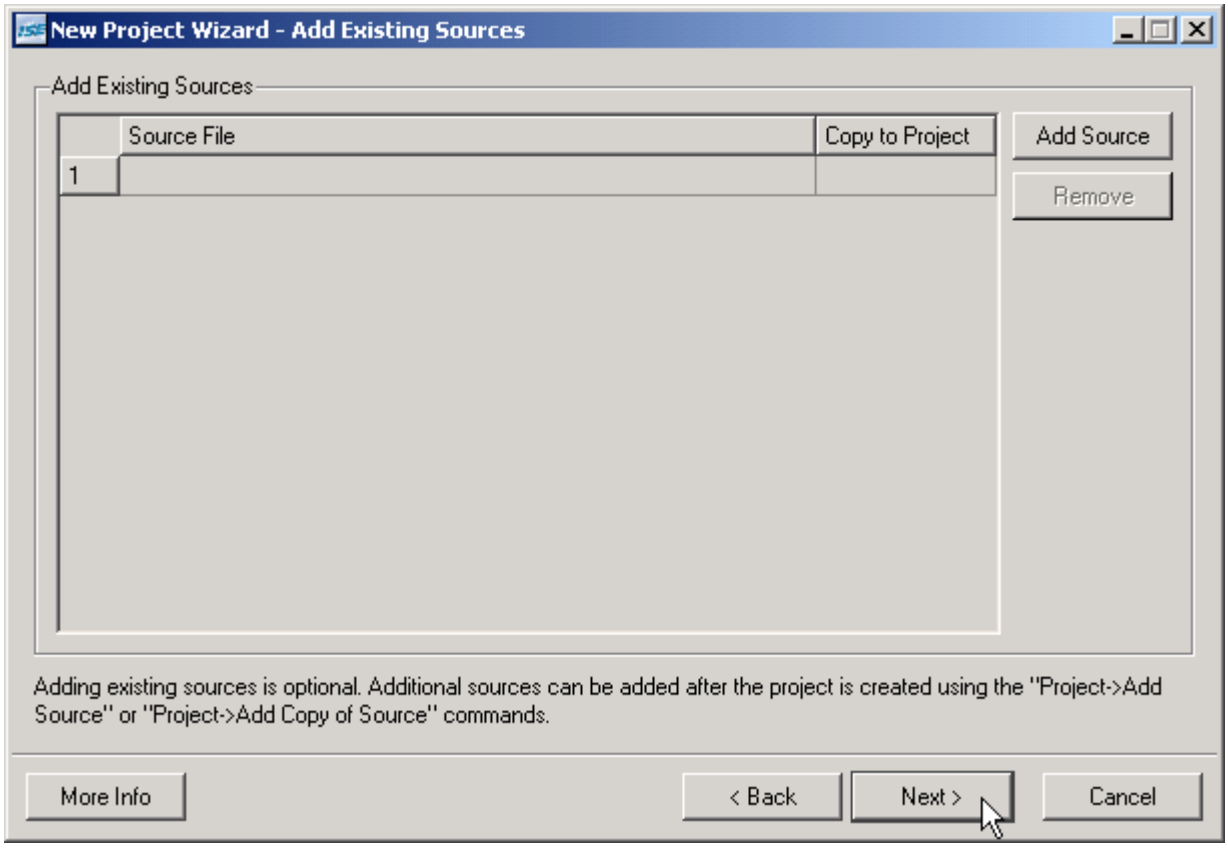
XSA Board	Device Family	Device	Package	Speed Grade
XSA-50	Spartan2	XC2S50	TQ144	-5
XSA-100	Spartan2	XC2S100	TQ144	-5
XSA-200	Spartan2	XC2S200	FG256	-5
XSA-3S1000	Spartan3	XC3S1000	FT256	-4

For this tutorial, I will target my design to the XSA-3S1000 Board so I have set the Value field of the Family, Device, Package and Speed properties as shown below. (Set these fields to whatever values are appropriate for your particular board using the table shown above.) The Top-Level Source Type, Synthesis Tool, and Simulator can all be left at their default values, so you can just click on the Next button to continue creating the project.

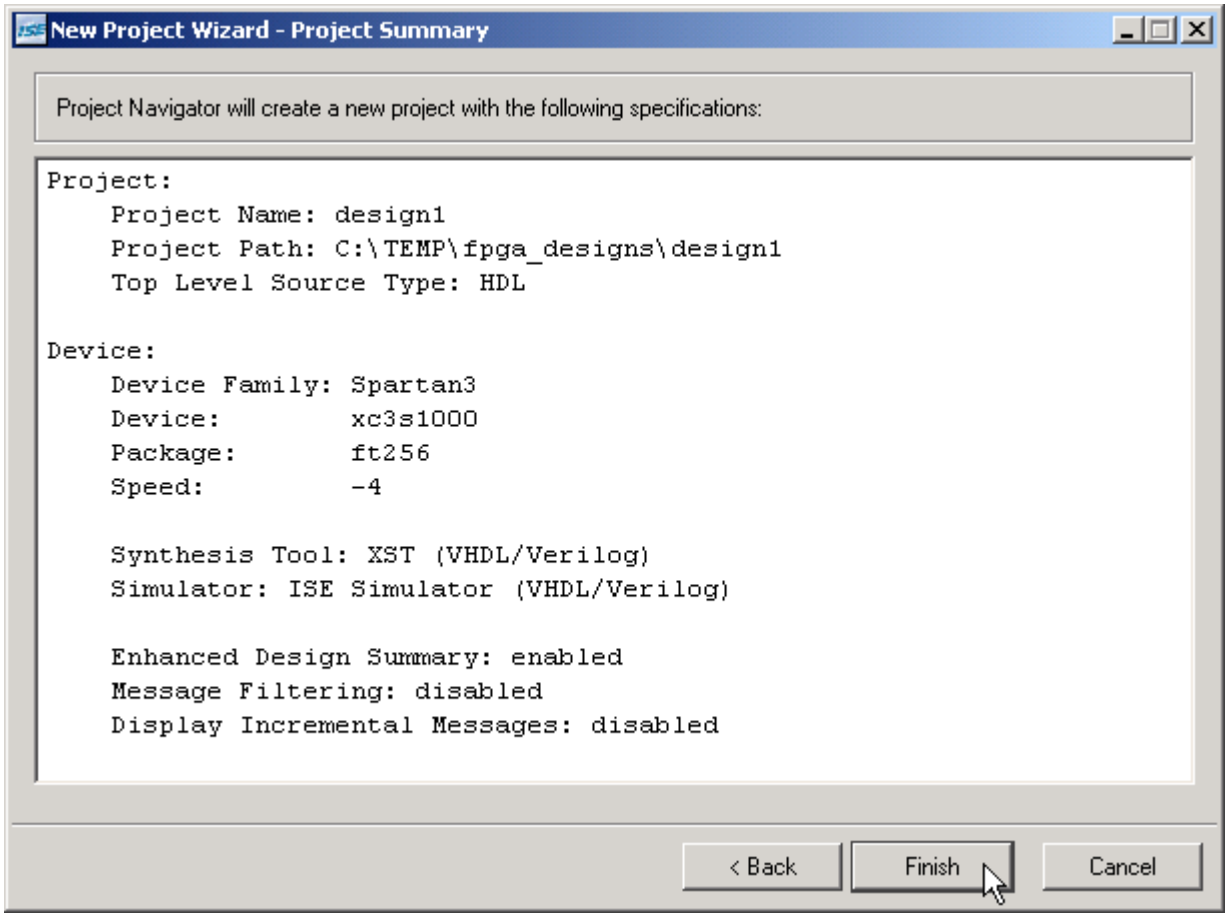


Click the Next button in the following two windows for creating or adding source files. (You will create the VHDL source code for the LED decoder at a later step.)



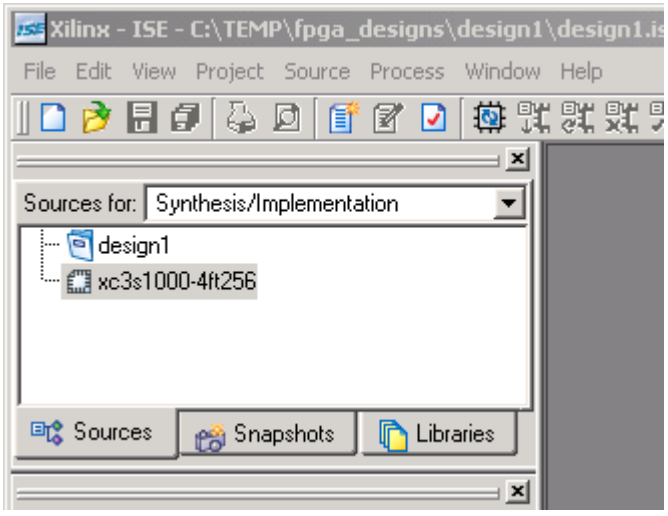


The final screen shows the pertinent information for the new project. Click on the Finish button to complete the creation of the project.



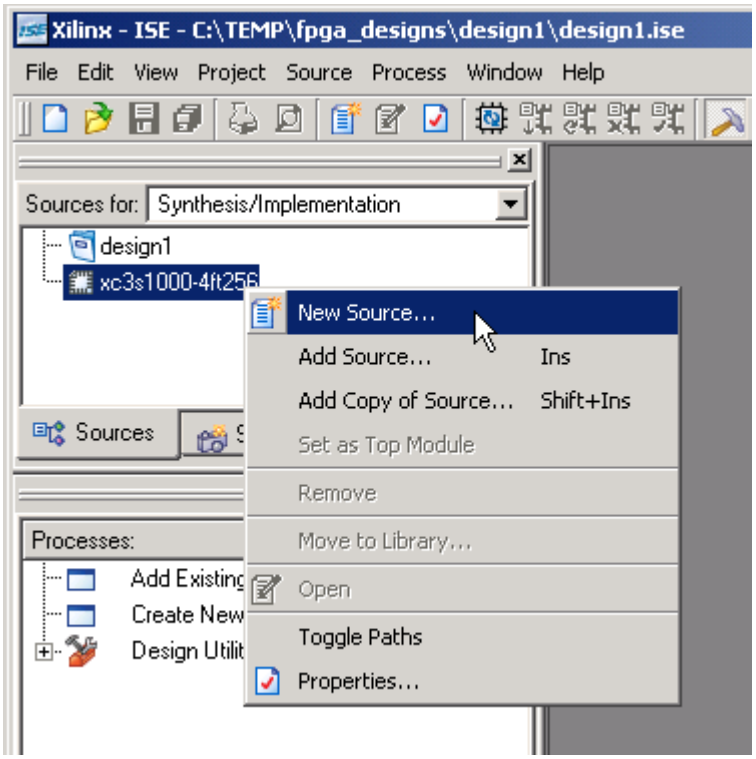
Now the Sources pane contains two items:

1. A project object called design1.
2. A chip object called xc3s1000-4ft256.

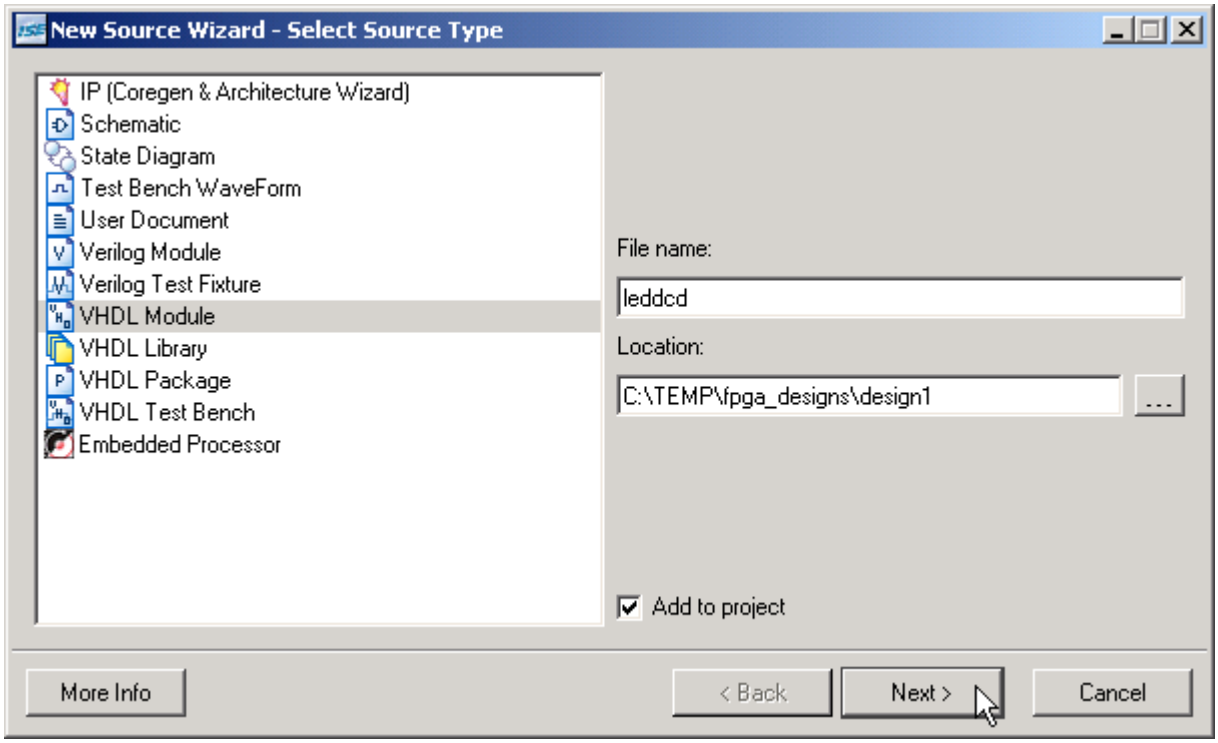


Describing Your Design With VHDL

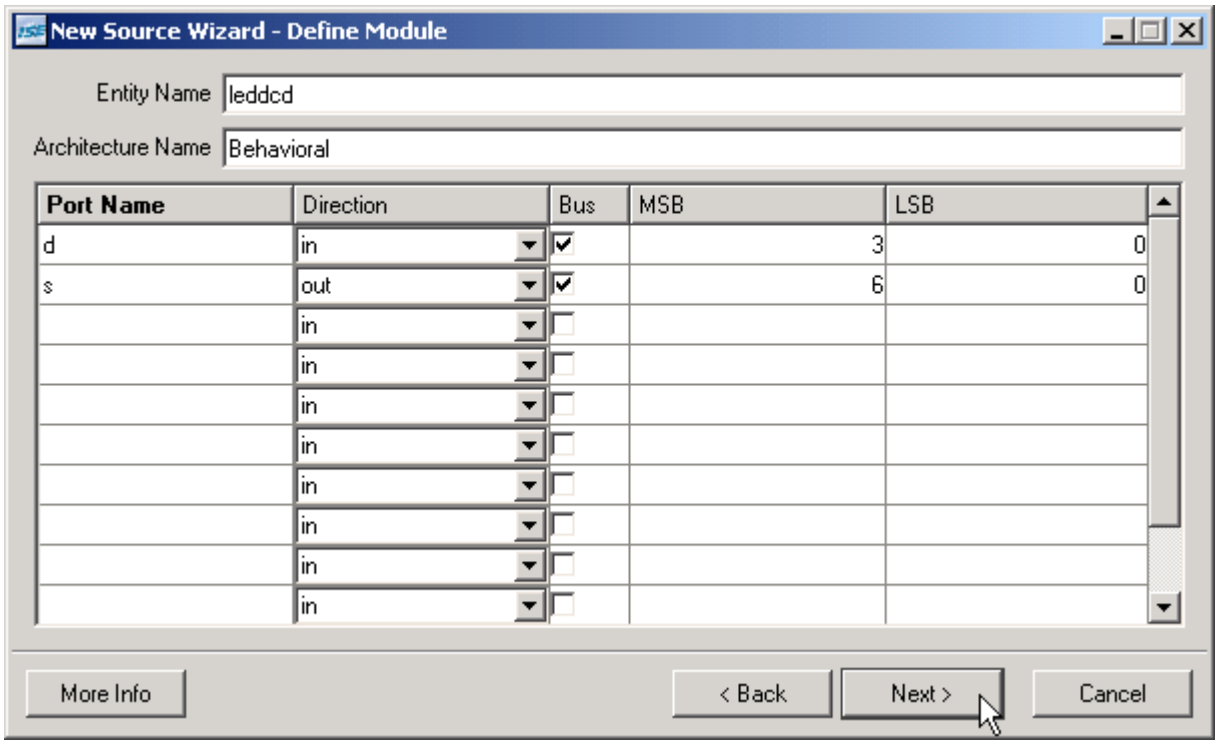
Once all the project set-up is complete, you can begin to actually design your LED decoder circuit. Start by adding a VHDL file to the ***design1*** project. Right-click on the xc3s1000-4ft256 object in the Sources pane and select New Source ... from the pop-up menu as shown below.



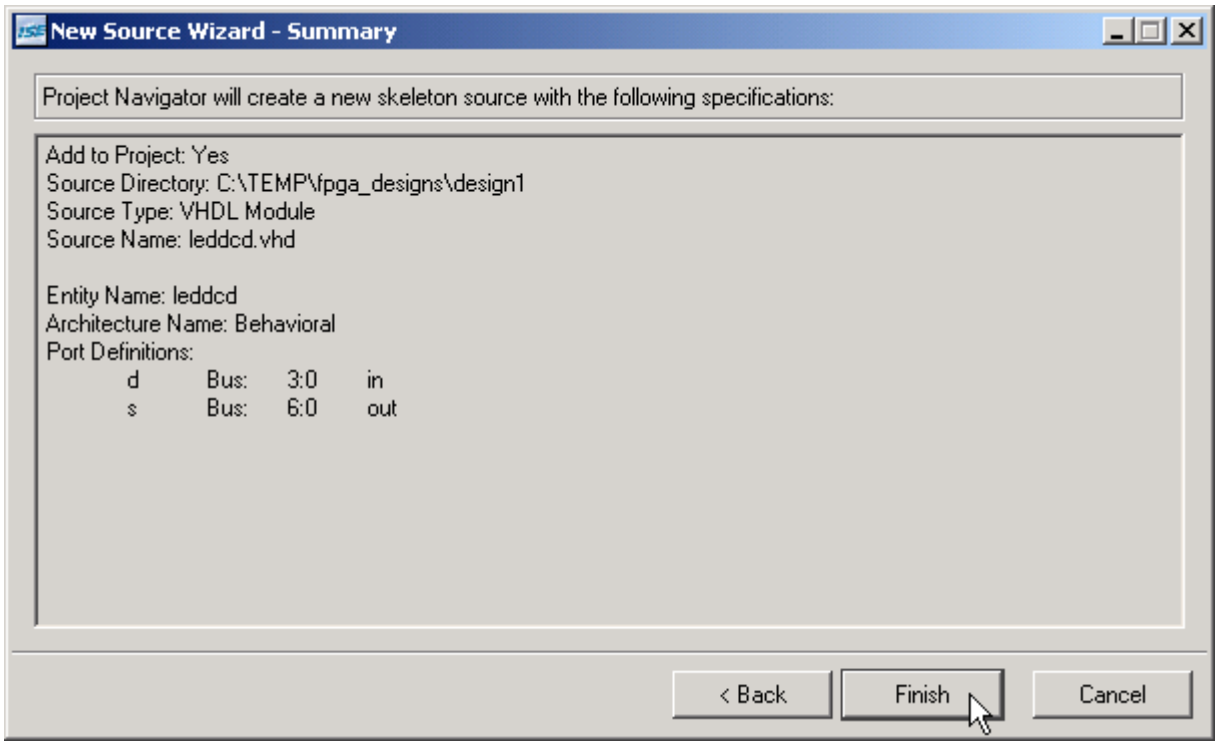
A window appears where you must select the type of source file you want to add. Since you are describing the LED decoder with VHDL, highlight the VHDL Module item. Then type the name of the module, **leddcd**, into the File name field and click on Next.



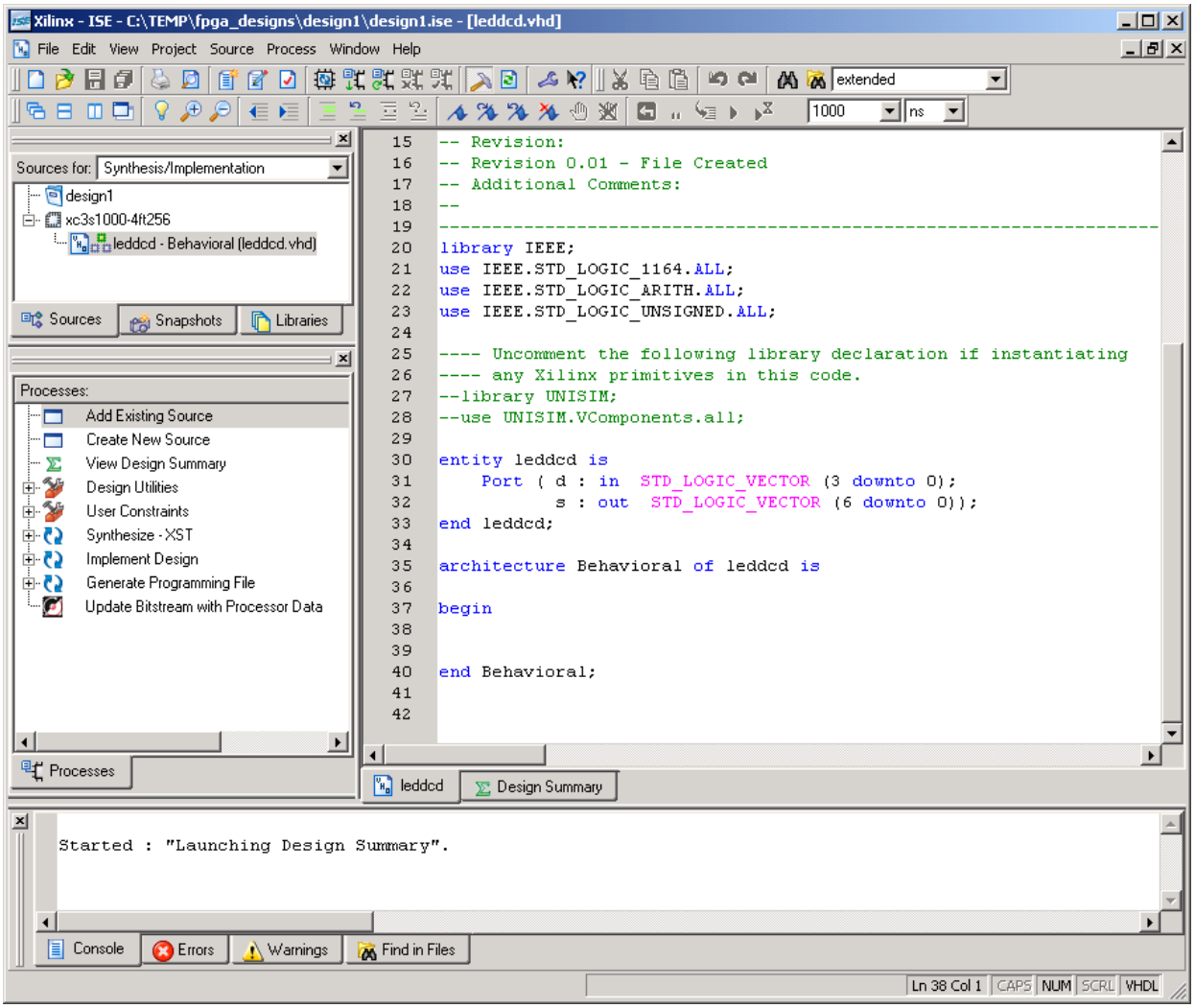
The **Define VHDL Source** window now appears where you can declare the inputs and outputs to the LED decoder circuit. In the first row, click in the Port Name field and type in **d** (the name of the inputs to the LED decoder). The **d** input bus has a width of four, so check the Bus box and type 3 in the MSB field while leaving 0 in the LSB field. Perform the same operations to create the seven-bit wide **s** output bus that drives the LEDs.



Click on Next in the **Define VHDL Source** window to get a summary of the information you just typed in:



After clicking on Finish, the editor pane displays a design summary and a VHDL skeleton for the LED decoder. (You can also see the leddcd.vhd file has been added to the Sources pane.) Click on the leddcd tab at the bottom of the editor pane and then scroll to the bottom of the VHDL skeleton. Lines 20-23 create links to the IEEE library and packages that contain various useful definitions for describing a design. The LED decoder inputs and outputs are declared in the VHDL entity on lines 30-33. You will describe the logic operations of the decoder in the architecture section between lines 37 and 40.



The completed VHDL file for the LED decoder is shown below. The architecture section contains a single statement which assigns a particular seven-bit pattern to the **s** output bus for any given four-bit input on the **d** bus (lines 39-54).

leddcd.vhd

File Edit View Window

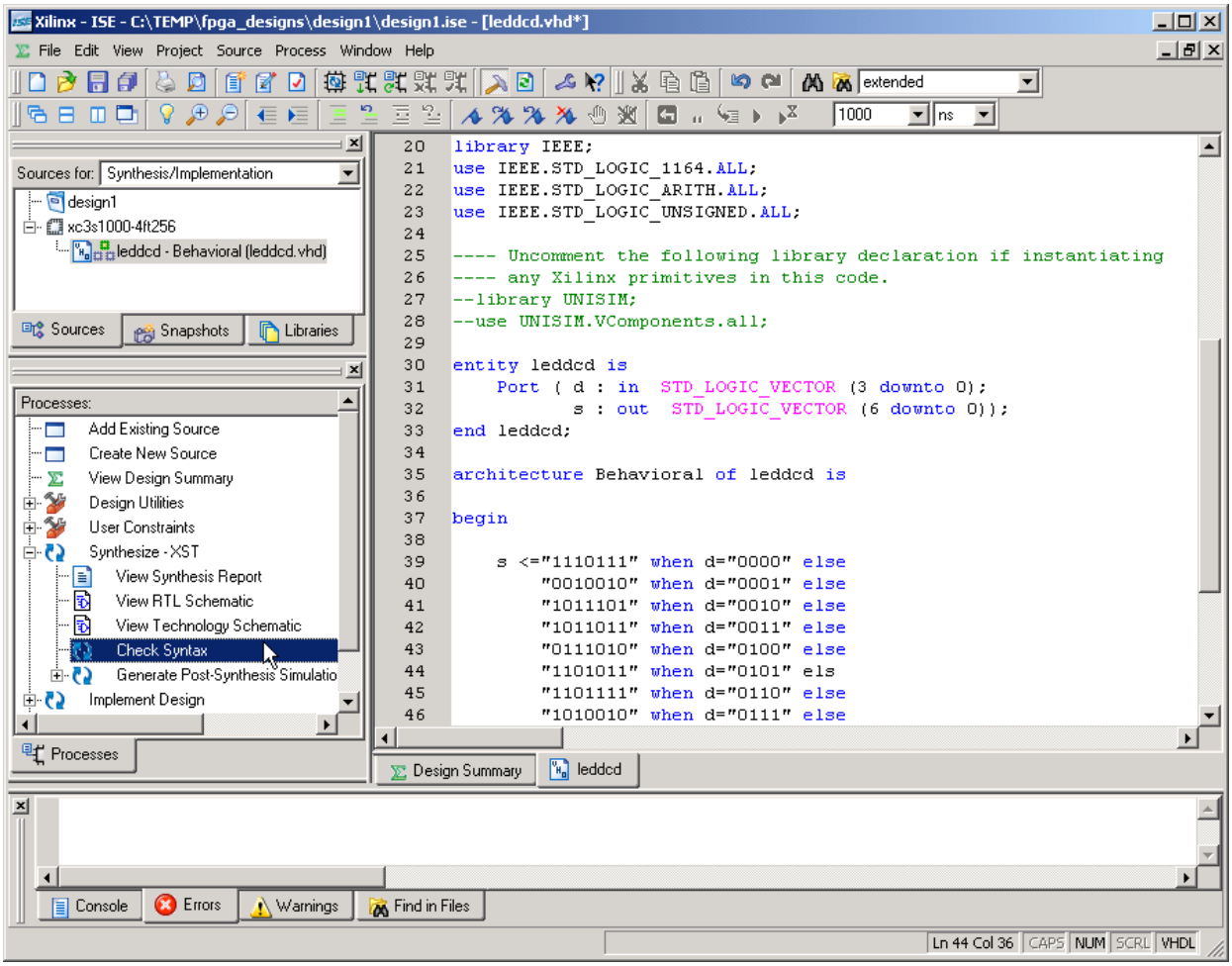
```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  ---- Uncomment the following library declaration if instantiating
26  ---- any Xilinx primitives in this code.
27  --library UNISIM;
28  --use UNISIM.VComponents.all;
29
30  entity leddcd is
31      Port ( d : in  STD_LOGIC_VECTOR (3 downto 0);
32             s : out  STD_LOGIC_VECTOR (6 downto 0));
33  end leddcd;
34
35  architecture Behavioral of leddcd is
36
37  begin
38
39      s <="1110111" when d="0000" else
40          "0010010" when d="0001" else
41          "1011101" when d="0010" else
42          "1011011" when d="0011" else
43          "0111010" when d="0100" else
44          "1101011" when d="0101" els
45          "1101111" when d="0110" else
46          "1010010" when d="0111" else
47          "1111111" when d="1000" else
48          "1111011" when d="1001" else
49          "1111110" when d="1010" else
50          "0101111" when d="1011" else
51          "0001101" when d="1100" else
52          "0011111" when d="1101" else
53          "1101101" when d="1110" else
54          "1101100"
55
56  end Behavioral;
```

CAPS NUM SCRL Ln 54 Col 18 VHDL

Once the VHDL source is entered, click on the button to save it in the leddcd.vhd file.

Checking the VHDL Syntax

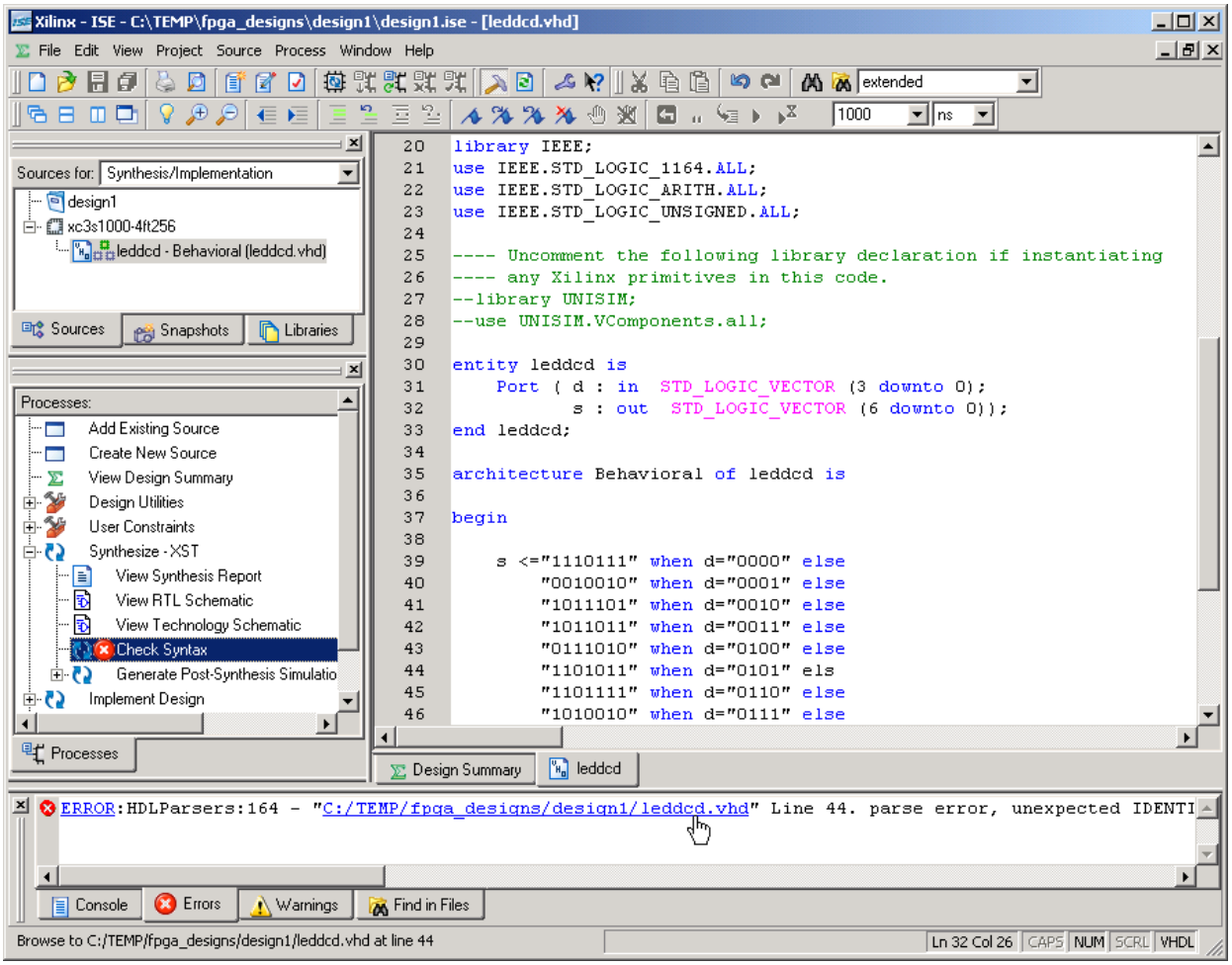
You can check for errors in our VHDL by highlighting the leddcd object in the Sources pane and then double-clicking on Check Syntax in the Process pane as shown below.



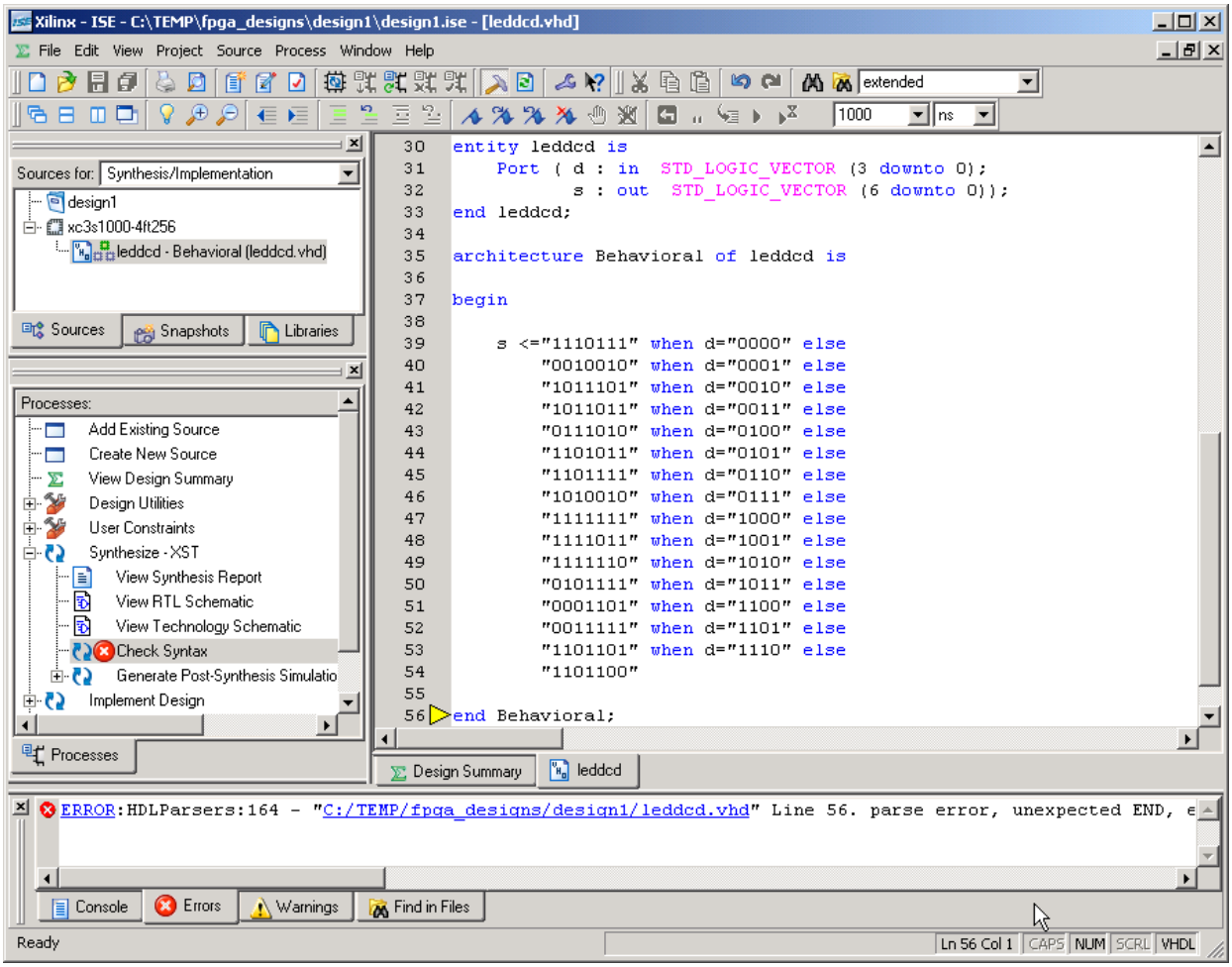
The syntax checking tool grinds away and then displays the result in the process pane. In this case, an error was found as indicated by the  next to the Check Syntax process. But what is the error and where is it?

Fixing VHDL Errors

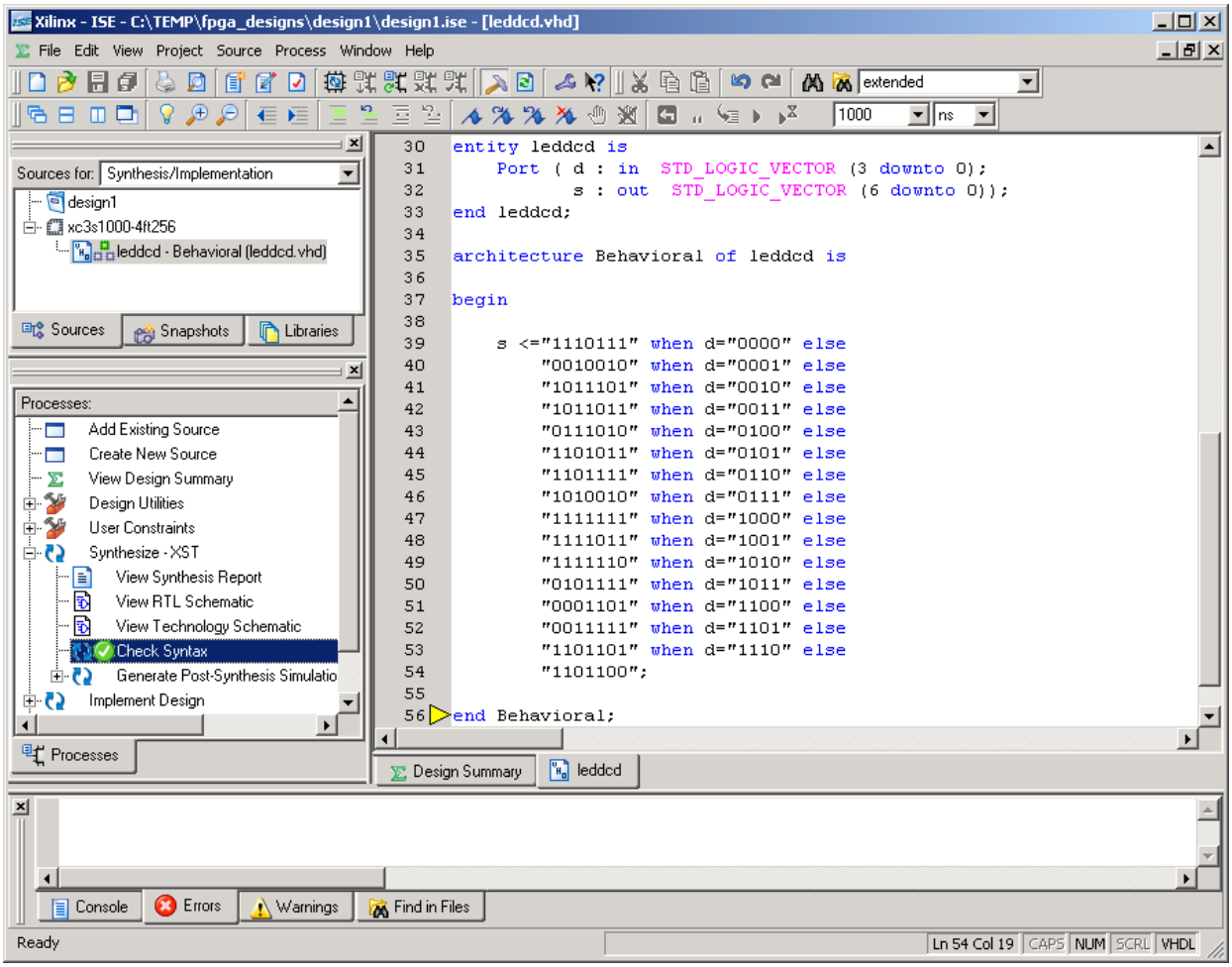
You can find the location of the error by clicking on the Errors tab at the bottom of the transcript pane. In this case, the error is located on line 44 and you can manually scroll there. You can also click on the error message in the log pane to go directly to the erroneous source. (This is most useful in more complicated projects consisting of multiple source files.) You can also get a more detailed explanation of the error by clicking on the ERROR hyperlink at the beginning of the message.



On line 44, you can see that the 'e' was left off the end of the `else` keyword. After correcting this error and saving the file, double-click the on Check Syntax in the Process pane to re-check the VHDL code. The syntax checker now finds another error on line 56 of the VHDL code.

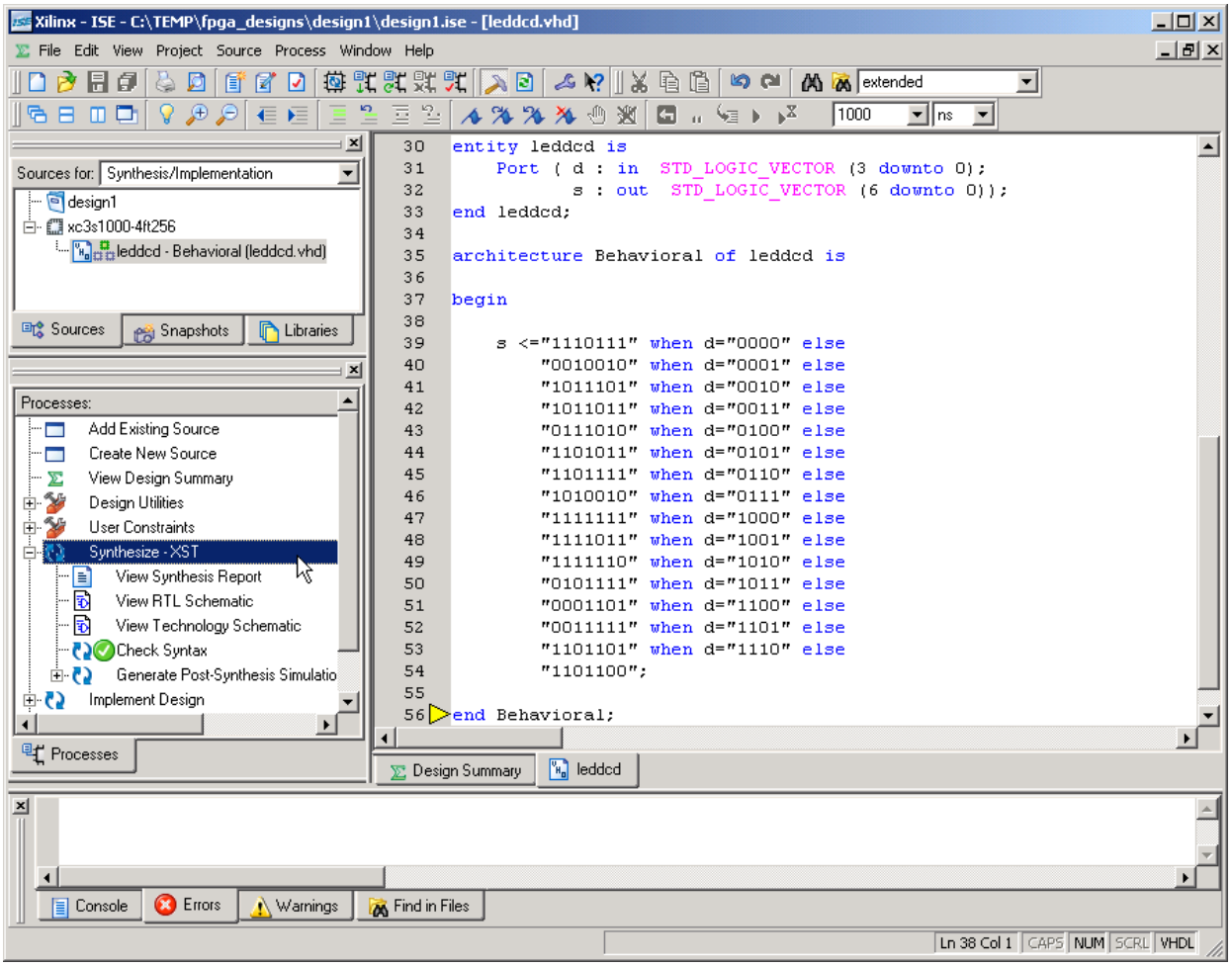


Examining line 56, you can see it is just the end statement for the architecture section. The actual error occurred on line 54. The VHDL syntax checker was expecting to find a ';' but it is missing. After adding the semicolon and saving the file, the Check Syntax process runs without errors and displays a .



Synthesizing the Logic circuitry for Your Design

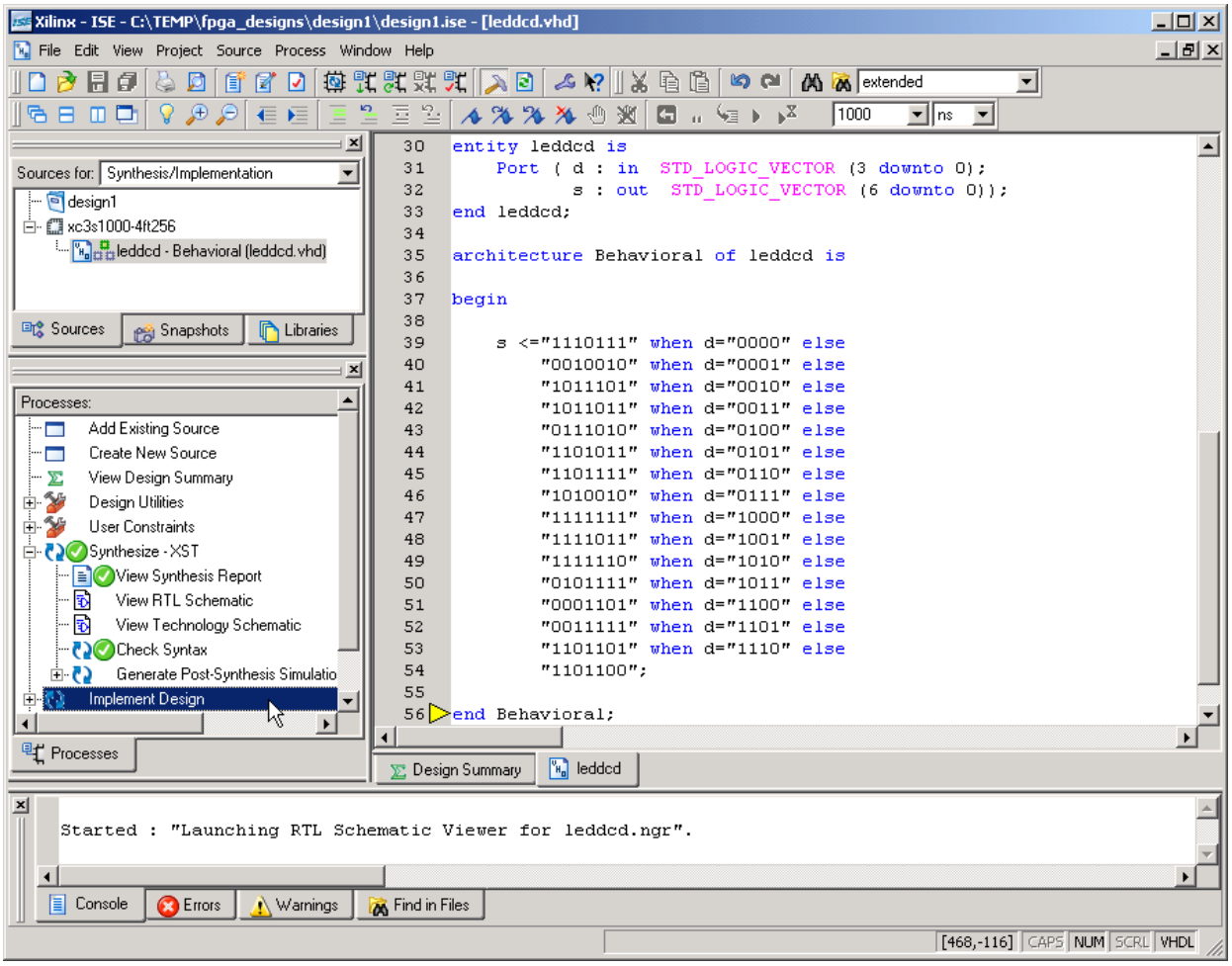
Now that you have valid VHDL for your design, you need to convert it into a logic circuit. This is done by highlighting the leddcd object in the Sources pane and then double-clicking on the Synthesize-XST process as shown below.



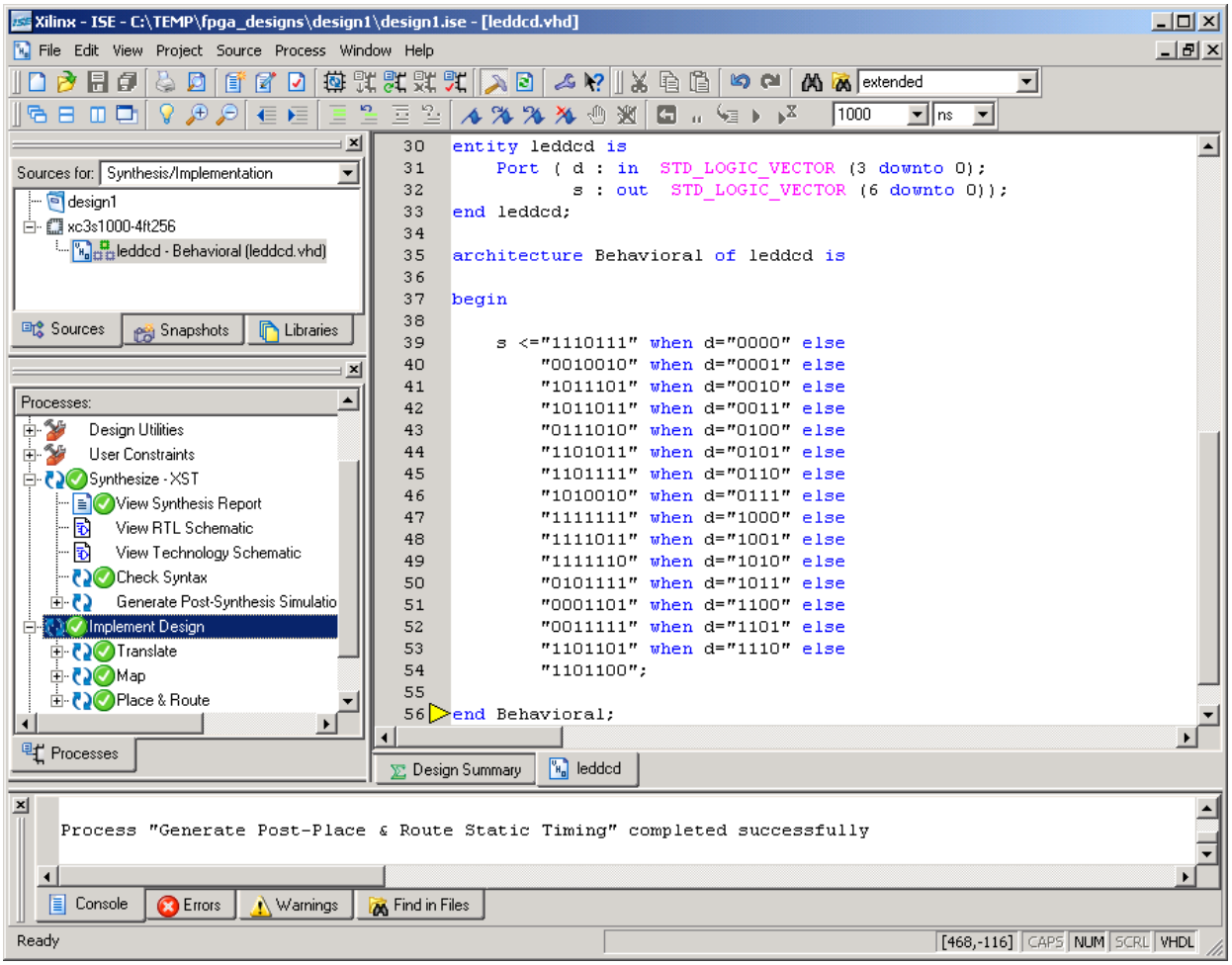
The synthesizer will read the VHDL code and transform it into a netlist of gates. This will take less than a minute. If no problems are detected, a ✅ will appear next to the Synthesize process. You can double-click on the View Synthesis Report to see the various synthesizer options that were enabled and some device utilization and timing statistics for the synthesized design (the device utilization is also viewable by clicking on the Design Summary tab in the editor pane). You can also double-click on View RTL Schematic to see the schematic that was derived from the VHDL source code, but it's not very interesting in this case.

Implementing the Logic Circuitry in the FPGA

You now have a synthesized logic circuit for the LED decoder, but you need to translate, map and place & route it into the logic resources of the FPGA in order to actually use it. Start this process by highlighting the leddcd object in the Sources pane and then double-click on the Implement Design process.

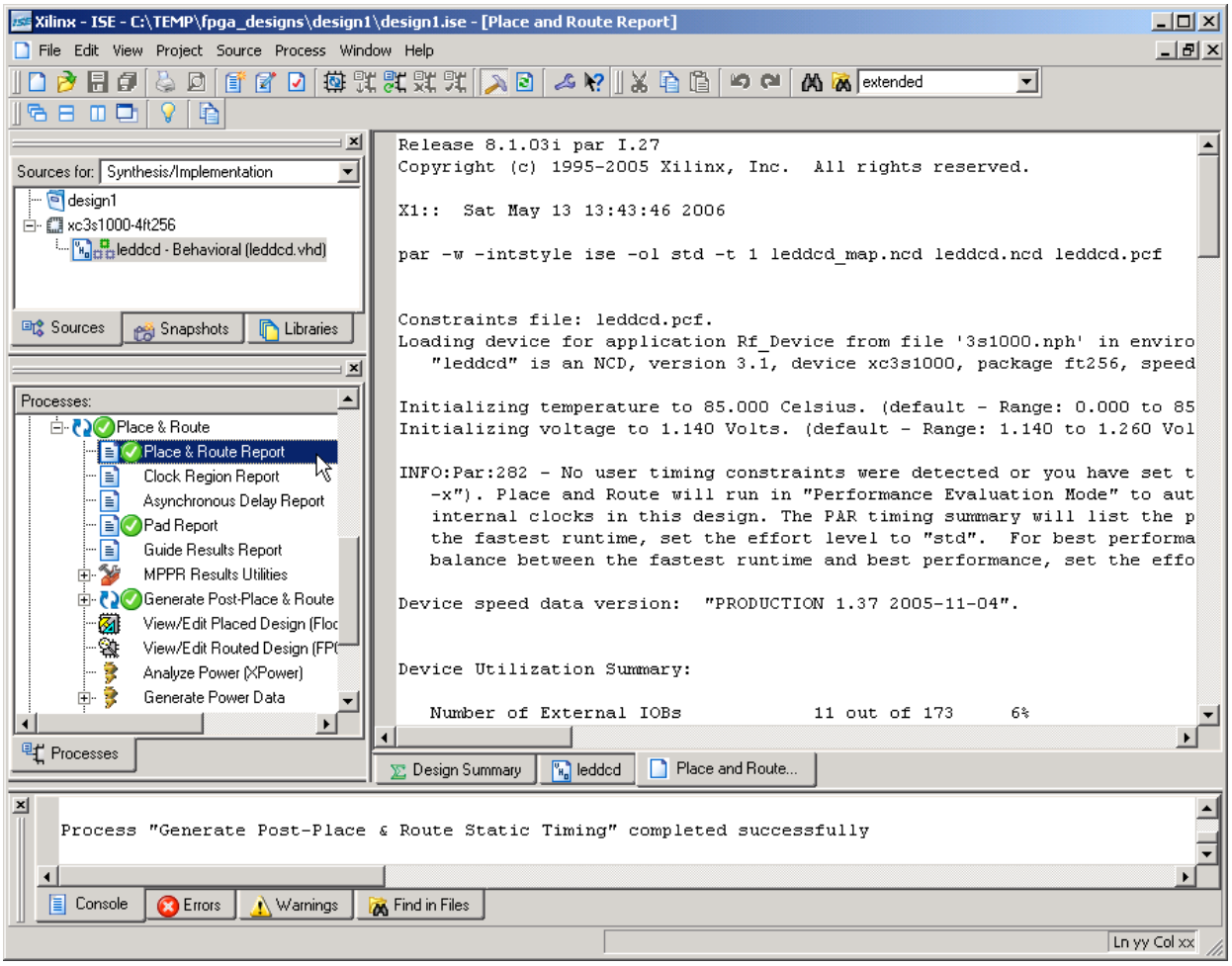


You can watch the progress of the implementation process in the console tab of the transcript pane. For a simple design this, the implementation is completed in less than 30 seconds (on a 1.8 GHz Athlon PC with 1 Gbyte of RAM). A successful implementation is indicated by the ✅ next to the Implement Design process. You can expand the Implement Design process to see the subprocesses within it. The Translate process converts the netlist output by the synthesizer into a Xilinx-specific format and annotates it with any design constraints you may specify (more on that later). The Map process decomposes the netlist and rearranges it so it fits nicely into the circuitry elements contained in the specified FPGA device. Then the Place & Route process assigns the mapped elements to specific locations in the FPGA and sets the switches to route the logic signals between them. If the Implement Design provcess had failed, a ❌ would appear next to the subprocess where the error occured. You may also see a ⚠ to indicate a successful completion but some warnings were issued or not all the subprocesses were enabled.



Checking the Implementation

You have your design fitted into the FPGA, but how much of the chip does it use? Which pins are the inputs and outputs assigned to? You can find answers to these questions by double-clicking on the Place & Route Report and the Pad Report in the Process pane.



The device utilization of the LED decoder circuit can be found near the top of the place & route report (or in the Design Summary tab). The circuit only uses four of the 7680 available slices in the XC3S1000 FPGA. Each slice contains two CLBs and each CLB can compute the logic function for one LED segment output.

```
Device utilization summary:

   Number of External IOBs           11 out of 173     6%
      Number of LOCed External IOBs   0 out of 11      0%

   Number of Slices                   4 out of 7680    1%
      Number of SLICEMs               0 out of 3840    0%
```

The pad report shows what pins the LED decoder inputs and outputs use to enter and exit the FPGA. (The pad report was edited to remove unused pins and fields so it would fit into this document.)

```
----------|------------|----------|----------|------------|
Pin Number|Signal Name |Pin Usage |Direction |IO Standard |
----------|------------|----------|----------|------------|
M6        |d<1>        |IOB       |INPUT     |LVCMOS25    |
M7        |d<3>        |IOB       |INPUT     |LVCMOS25    |
N5        |s<4>        |IOB       |OUTPUT    |LVCMOS25    |
N6        |s<1>        |IOB       |OUTPUT    |LVCMOS25    |
N7        |s<6>        |IOB       |OUTPUT    |LVCMOS25    |
P5        |s<5>        |IOB       |OUTPUT    |LVCMOS25    |
P6        |s<2>        |IOB       |OUTPUT    |LVCMOS25    |
P7        |d<0>        |IOB       |INPUT     |LVCMOS25    |
R5        |d<2>        |IOB       |INPUT     |LVCMOS25    |
R6        |s<3>        |IOB       |OUTPUT    |LVCMOS25    |
R7        |s<0>        |IOB       |OUTPUT    |LVCMOS25    |
```

Assigning Pins with Constraints

The problem now is that the inputs and outputs for the LED decoder were assigned to pins picked by the implementation process, but these are not the pins you actually want to use on the FPGA. You want the inputs assigned to pins on the FPGA that you can force high and low so as to test the LED decoder operation for each possible input pattern. In addition, the outputs should be attached to a seven-segment LED to make it easy to verify the correct operation of the design.

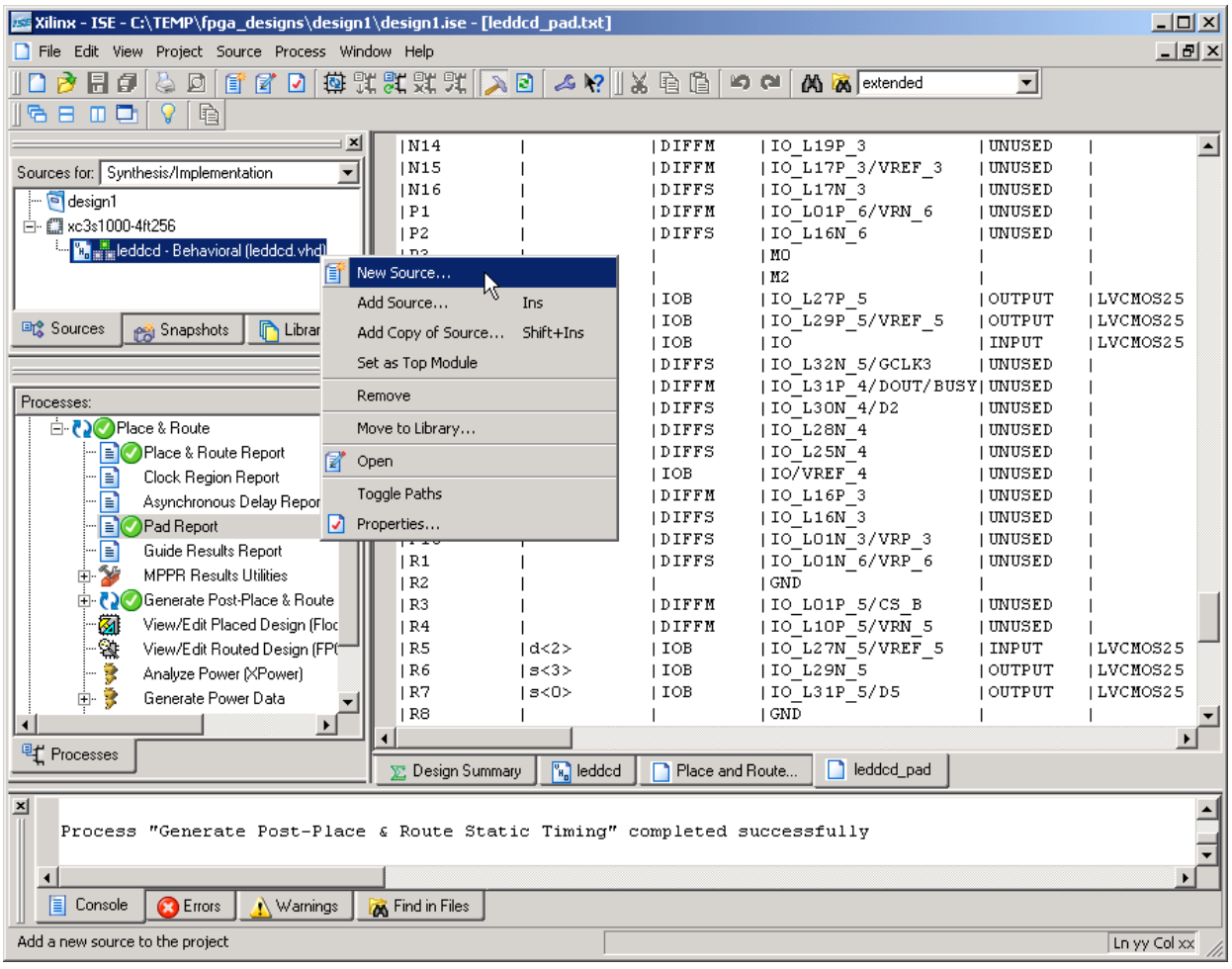
The GXSPORT utility lets you set the levels on the eight data outputs of the PC parallel port. The parallel port data pins attach to a group of eight specific pins on the FPGA of each model of XSA Board. You should assign the LED decoder inputs to four of these so that you can control the inputs using GXSPORT. The four pins I selected from the group of eight on each XSA Board are shown below:

LED Decoder Input	XSA-50	XSA-100	XSA-200	XSA-3S1000
d0	P50	P50	E13	N14
d1	P48	P48	C16	P15
d2	P42	P42	E14	R16
d3	P47	P47	D16	P14

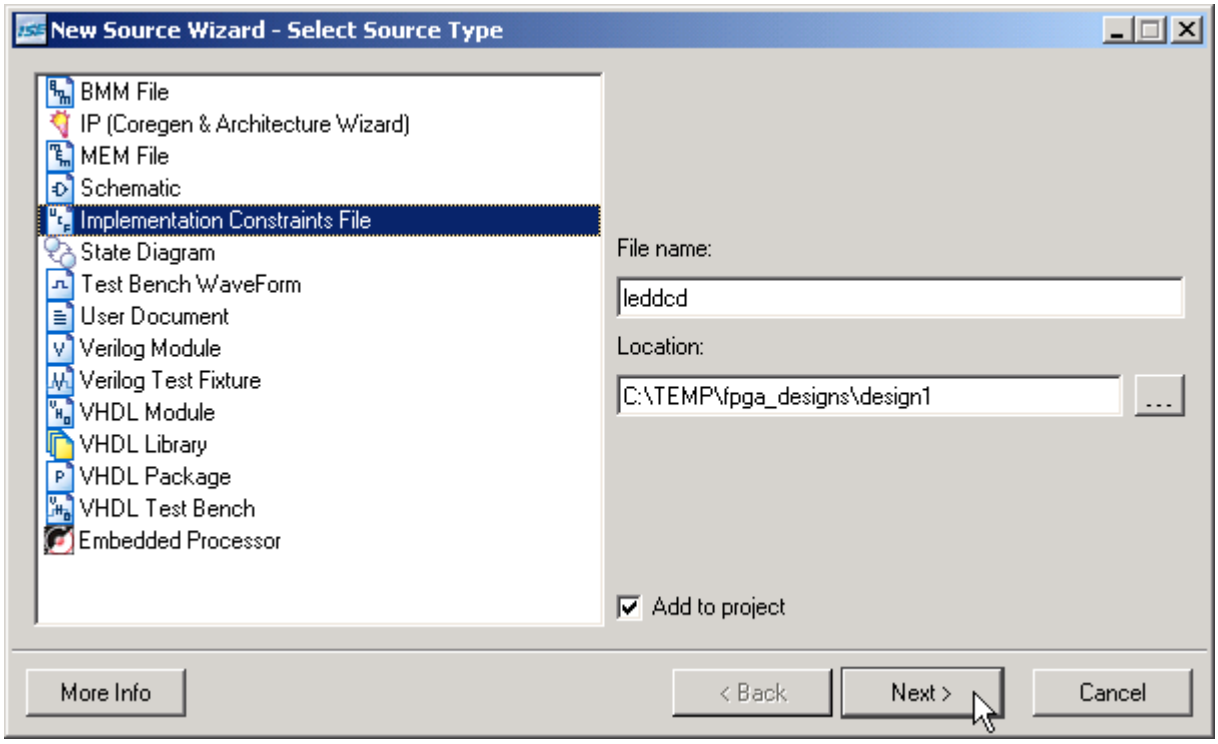
Likewise, each XSA Board has a seven-segment LED attached to the following pins of the FPGA:

LED Decoder Output	XSA-50	XSA-100	XSA-200	XSA-3S1000
s0	P67	P67	N14	M6
s1	P39	P39	D14	M11
s2	P62	P62	N16	N6
s3	P60	P60	M16	R7
s4	P46	P46	F15	P10
s5	P57	P57	J16	T7
s6	P49	P49	G16	R10

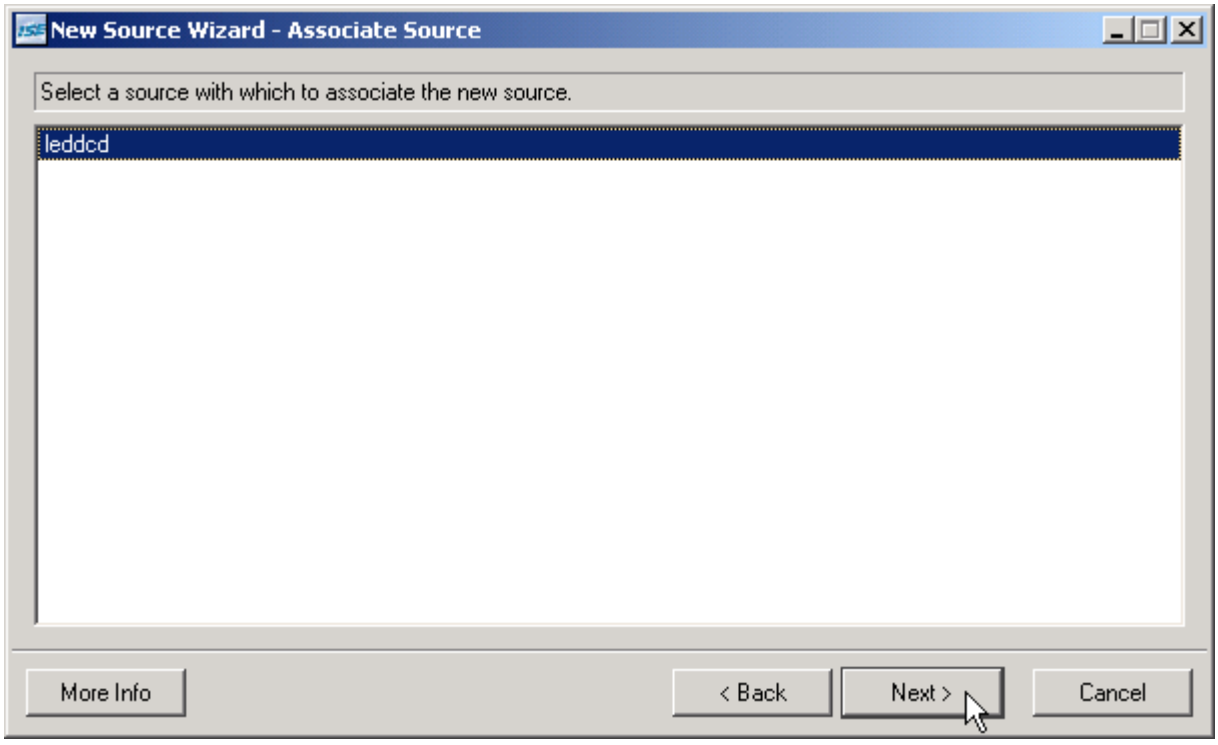
How do you direct the implementation process so it assigns the inputs and outputs to the pins you want to use? This is done by using *constraints*. In this case, you are constraining the implementation process so it assigns the inputs and outputs only to the pins shown in the previous tables. Start creating these constraints by right-clicking the leddcd object in the Sources pane and selecting New Source... from the pop-up menu.



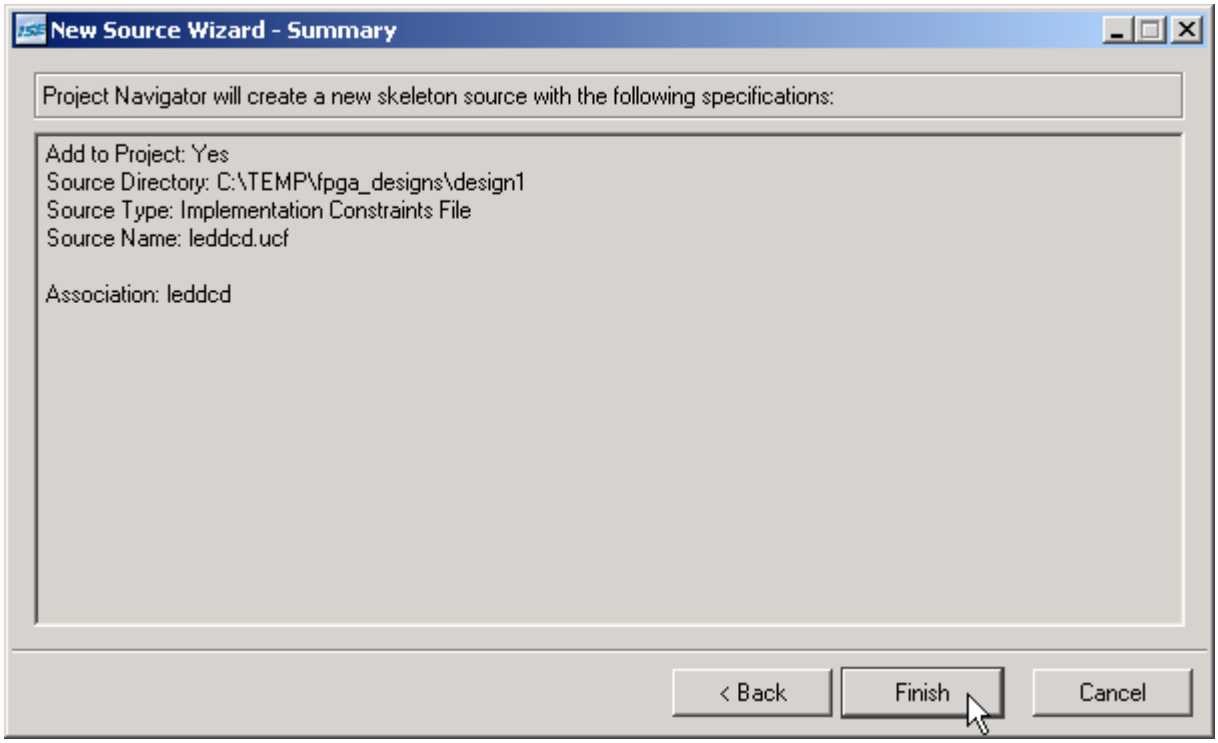
Select Implementation Constraints File as the type of source file you want to add and type `leddcd` in the File Name field. Then click on the Next button.



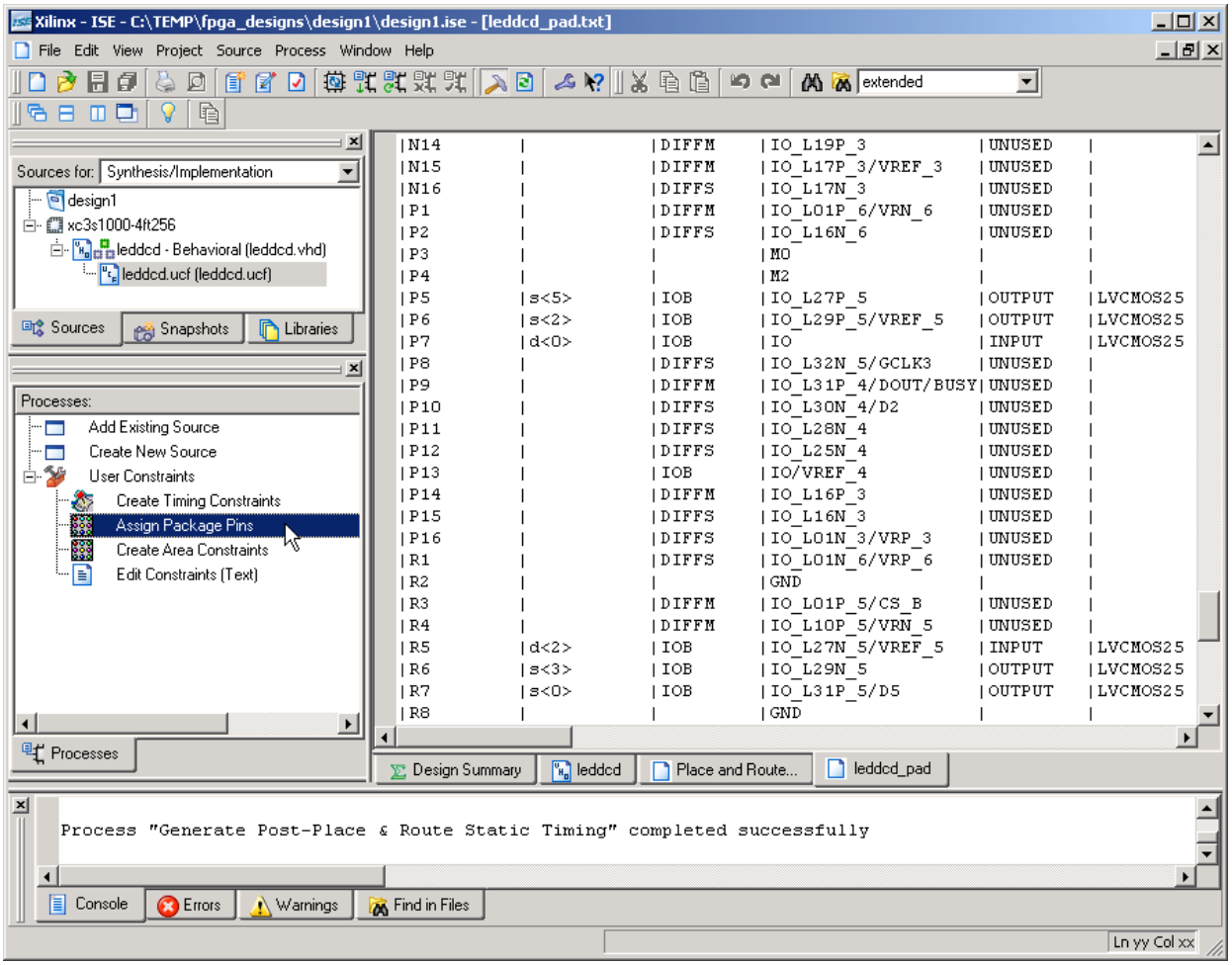
Then you are asked to pick the file that the constraints apply to. For this design there is only one choice, so click on the Next button and proceed.



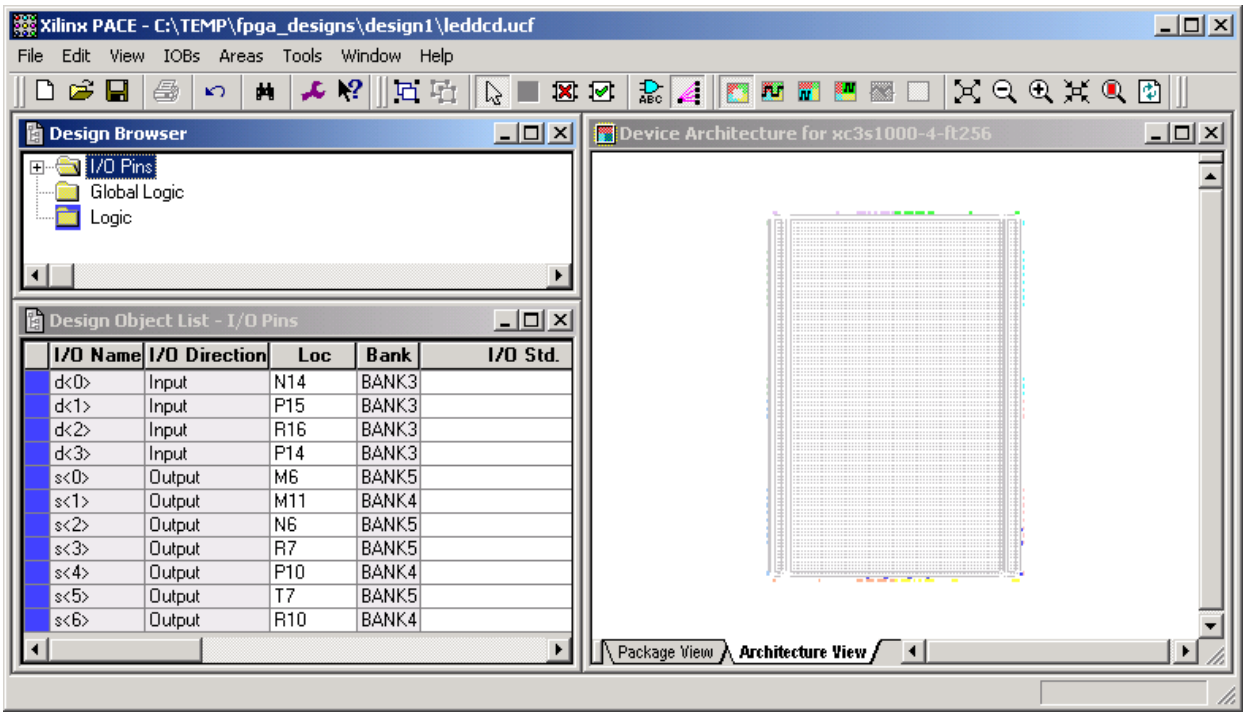
You will receive a feedback window that shows the name and type of the file you created and the file to which it is associated. Click on the Finish button to complete the addition of the leddcd.ucf constraint file to this project.



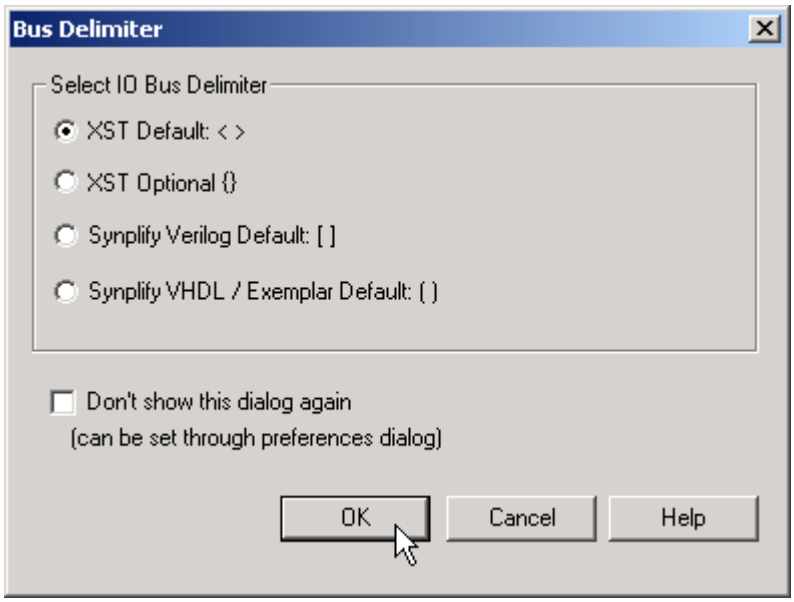
Now highlight the leddcd.ucf object in the Sources pane and double-click the Assign Package Pins in the Process pane to begin adding pin assignment constraints to the design.



The **Xilinx PACE** window now appears. Click on the I/O Pins item in the Design Browser pane. A list of the current inputs and outputs for the LED decoder will appear in the Design Object List – I/O Pins pane. You can change your pin assigments here. You start by clicking in the Loc field for the **d<0>** input. Then just type in the pin assignment for this input: `N14`. Do this for all of the inputs and outputs using the pin assignments from the previous table.



After the pin assignments are entered, save the file. A dialog window will appear requesting that you select the delimiter for the I/O buses. Select <> since the XST synthesizer is being used for this project, and then click on OK. Then close the **Xilinx PACE** window.



Now you can re-implement your design by highlighting the leddcd object in the Sources pane and double-clicking on the Implement Design process. After the implementation process completes,

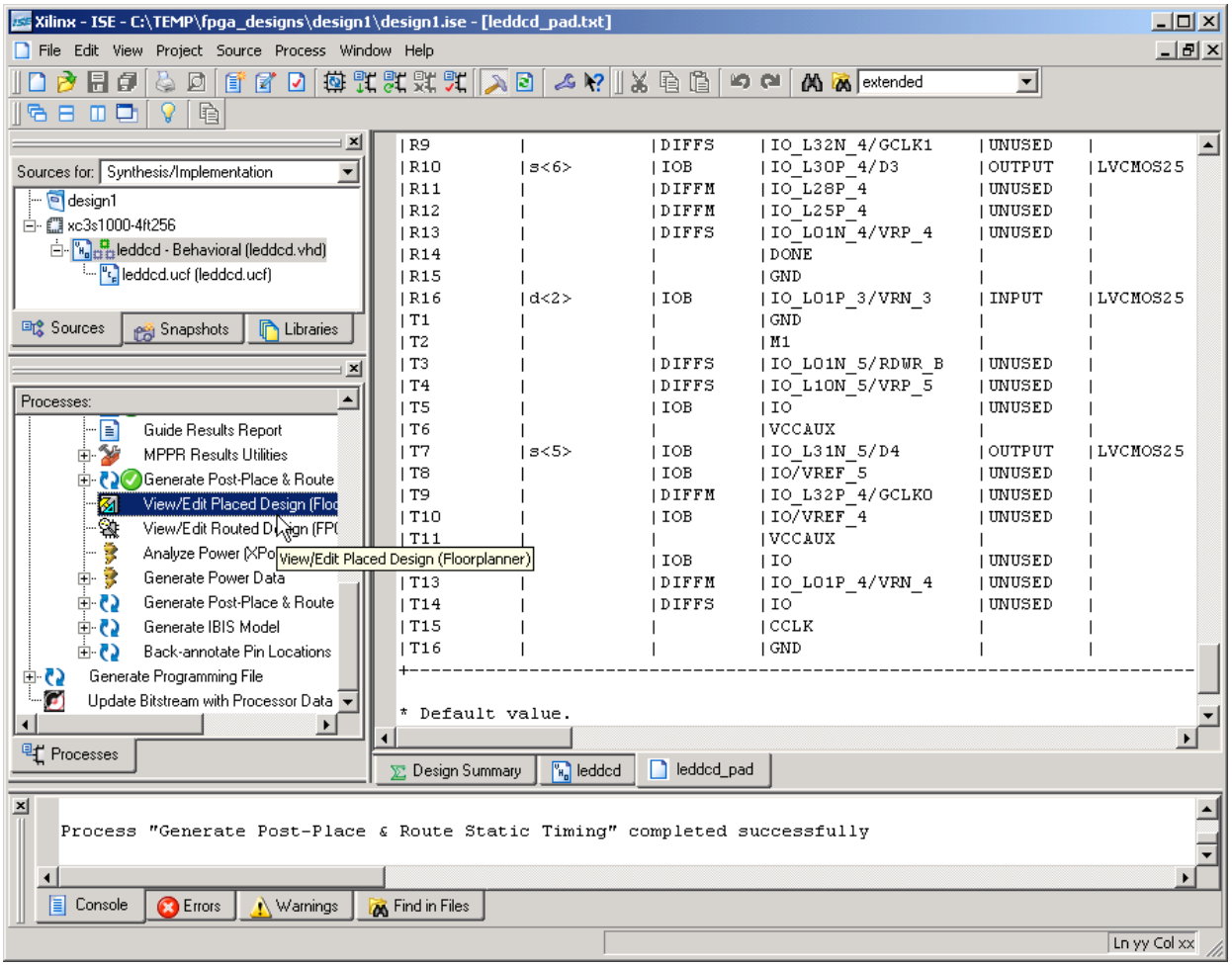
double-click on Pad Report to view the pin assignments. Now the pad report shows the following pin assignments:

Pin Number	Signal Name	Pin Usage	Direction	IO Standard
M6	s<0>	IOB	OUTPUT	LVCMOS25
M11	s<1>	IOB	OUTPUT	LVCMOS25
N6	s<2>	IOB	OUTPUT	LVCMOS25
N14	d<0>	IOB	INPUT	LVCMOS25
P10	s<4>	IOB	OUTPUT	LVCMOS25
P14	d<3>	IOB	INPUT	LVCMOS25
P15	d<1>	IOB	INPUT	LVCMOS25
R7	s<3>	IOB	OUTPUT	LVCMOS25
R10	s<6>	IOB	OUTPUT	LVCMOS25
R16	d<2>	IOB	INPUT	LVCMOS25
T7	s<5>	IOB	OUTPUT	LVCMOS25

The reported pin assignments match the assignments made in the **Xilinx Pace** window, so it appears the I/O have been constrained to the appropriate pins.

Viewing the Chip

After the implementation process completes, you can get a graphical depiction of how the logic circuitry and I/O are assigned to the FPGA CLBs and pins. Just highlight the leddcd object in the Sources pane and then double-click the View/Edit Placed Design (FloorPlanner) process.

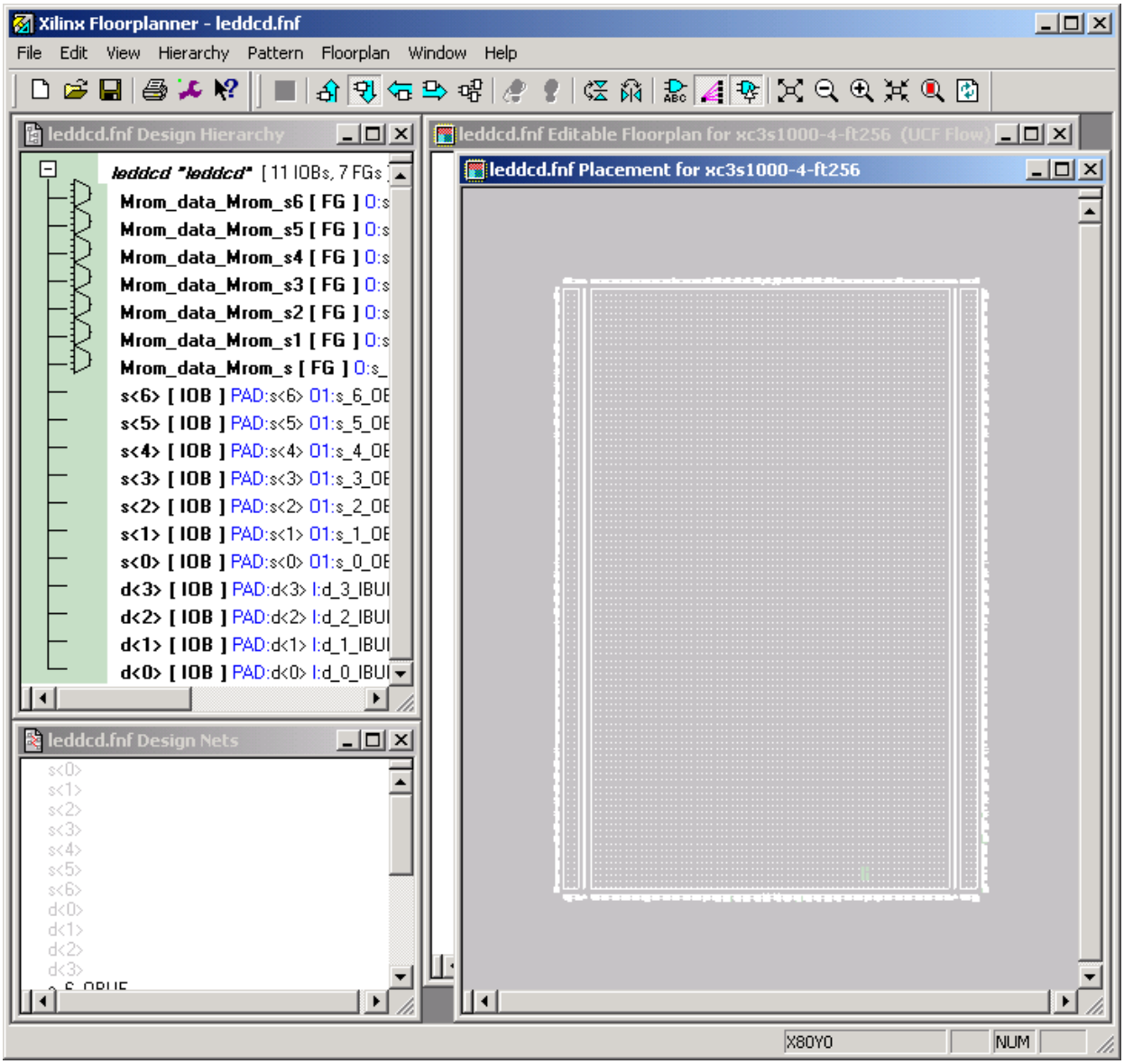


The **FloorPlanner** window will appear containing three panes:

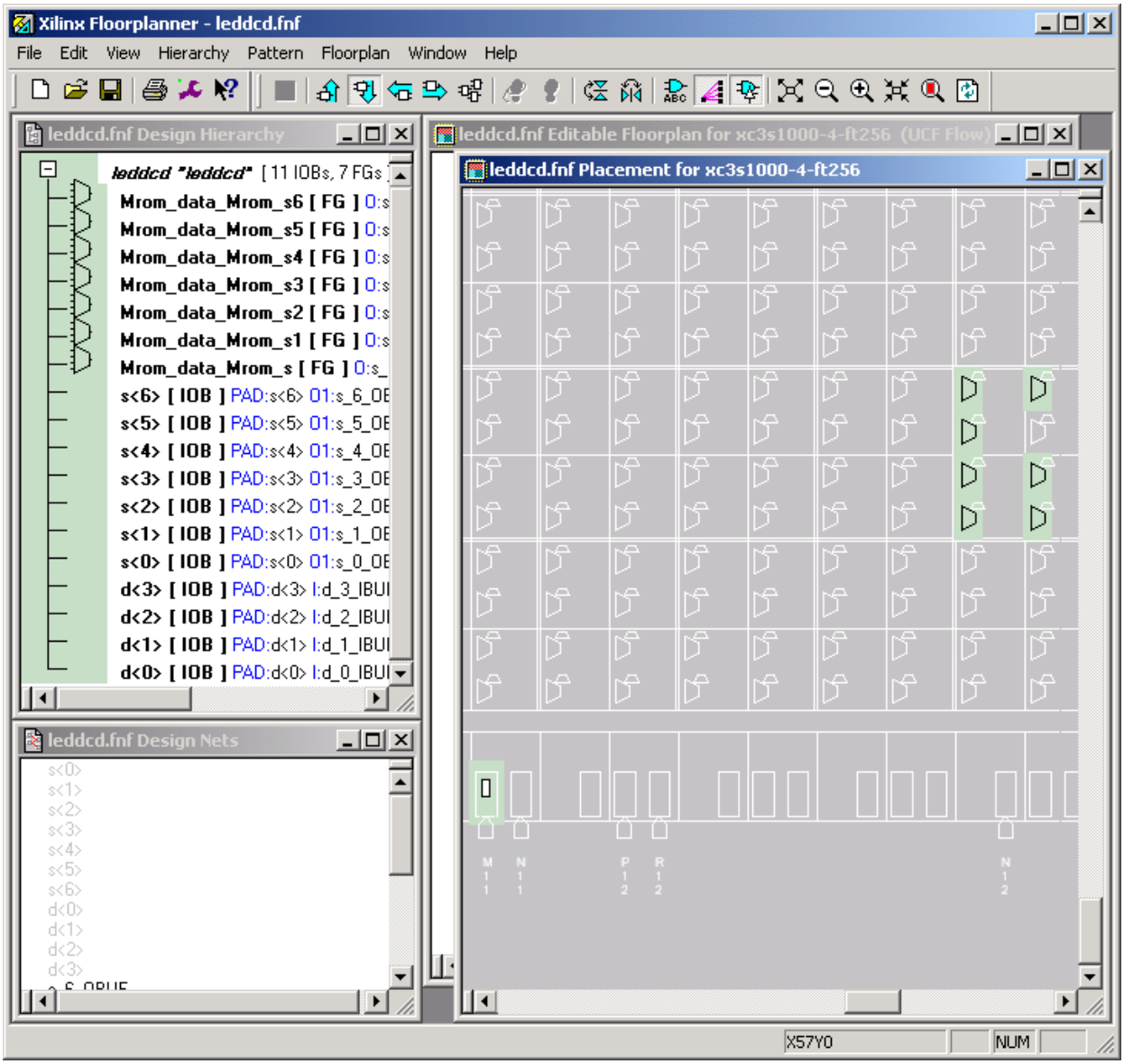
1. The Design Hierarchy pane lists the LED decoder inputs, outputs and LUTs assigned to the various CLBs in the FPGA.

2. The Design Nets pane lists the various signal nets in the LED decoder.

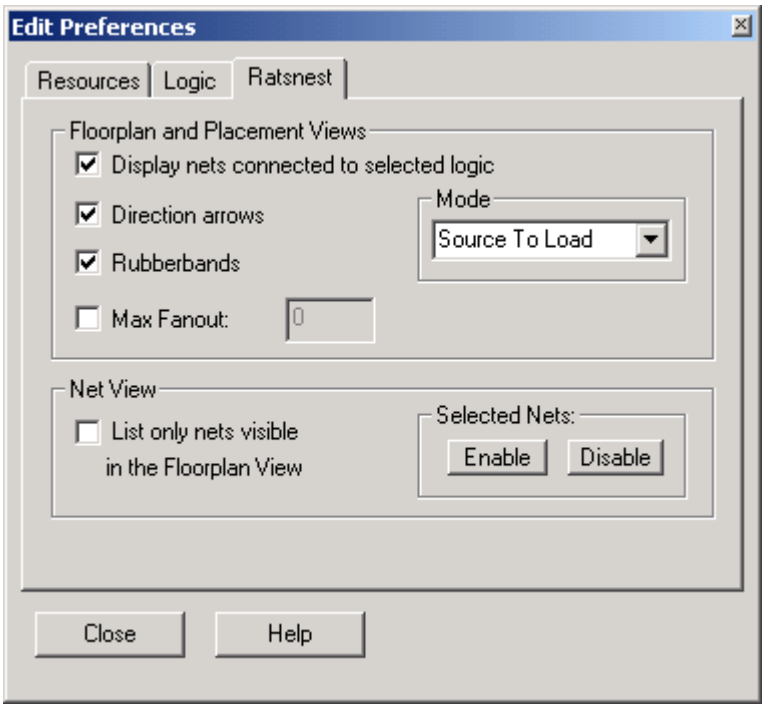
3. The Placement pane shows the array of CLBs in the FPGA. The I/O pins are also shown around the periphery. (The pins used for Vcc, GND, and programming are not shown.)



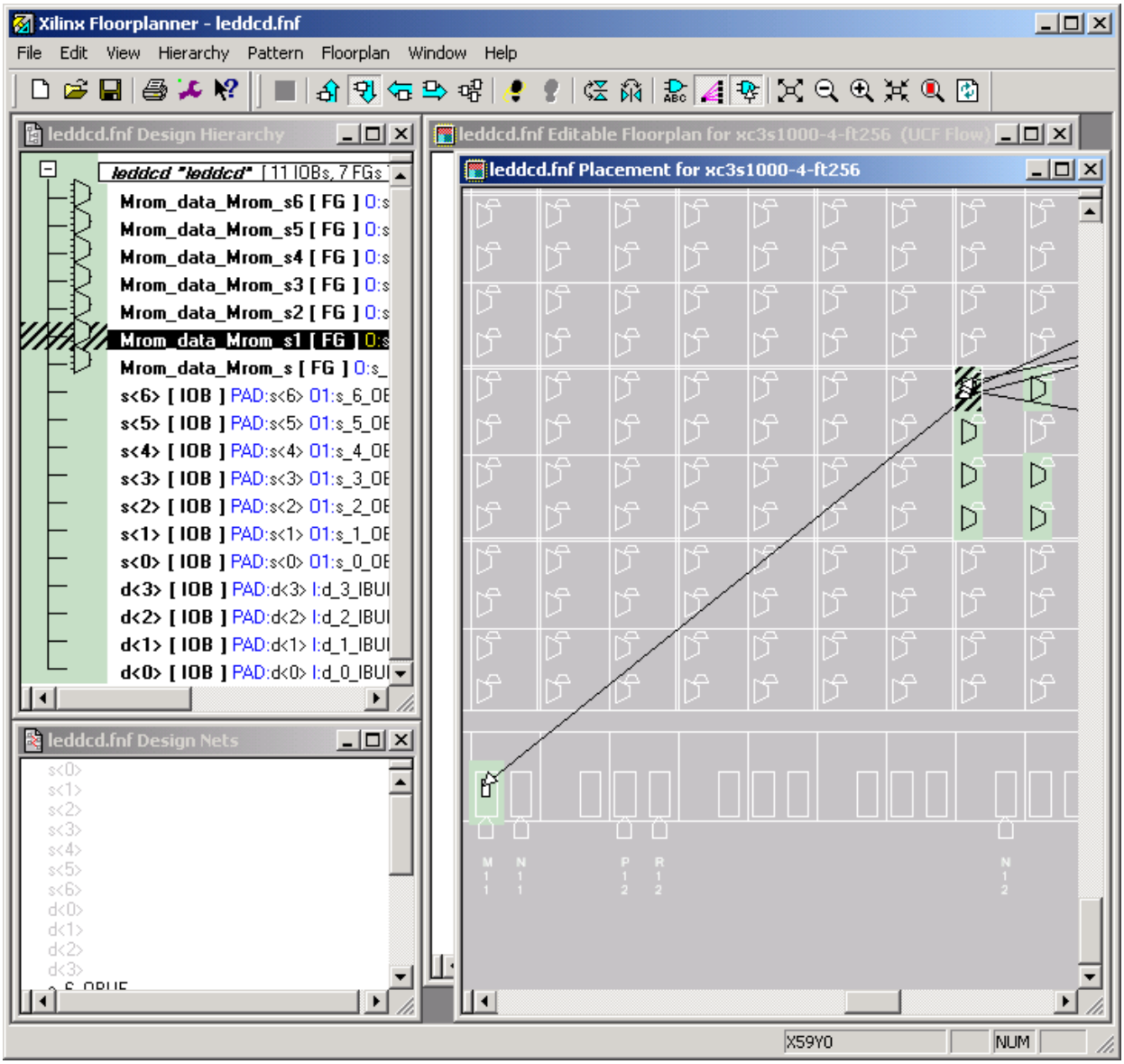
The CLBs used by the LED decoder circuit are highlighted in light-green and are clustered near the lower right-hand edge of the CLB array. To enlarge this region of the array, click on the  button and then draw a rectangle around the highlighted CLBs in the Placement pane. The enlarged view of the CLBs used by the LED decoder appear as shown below.



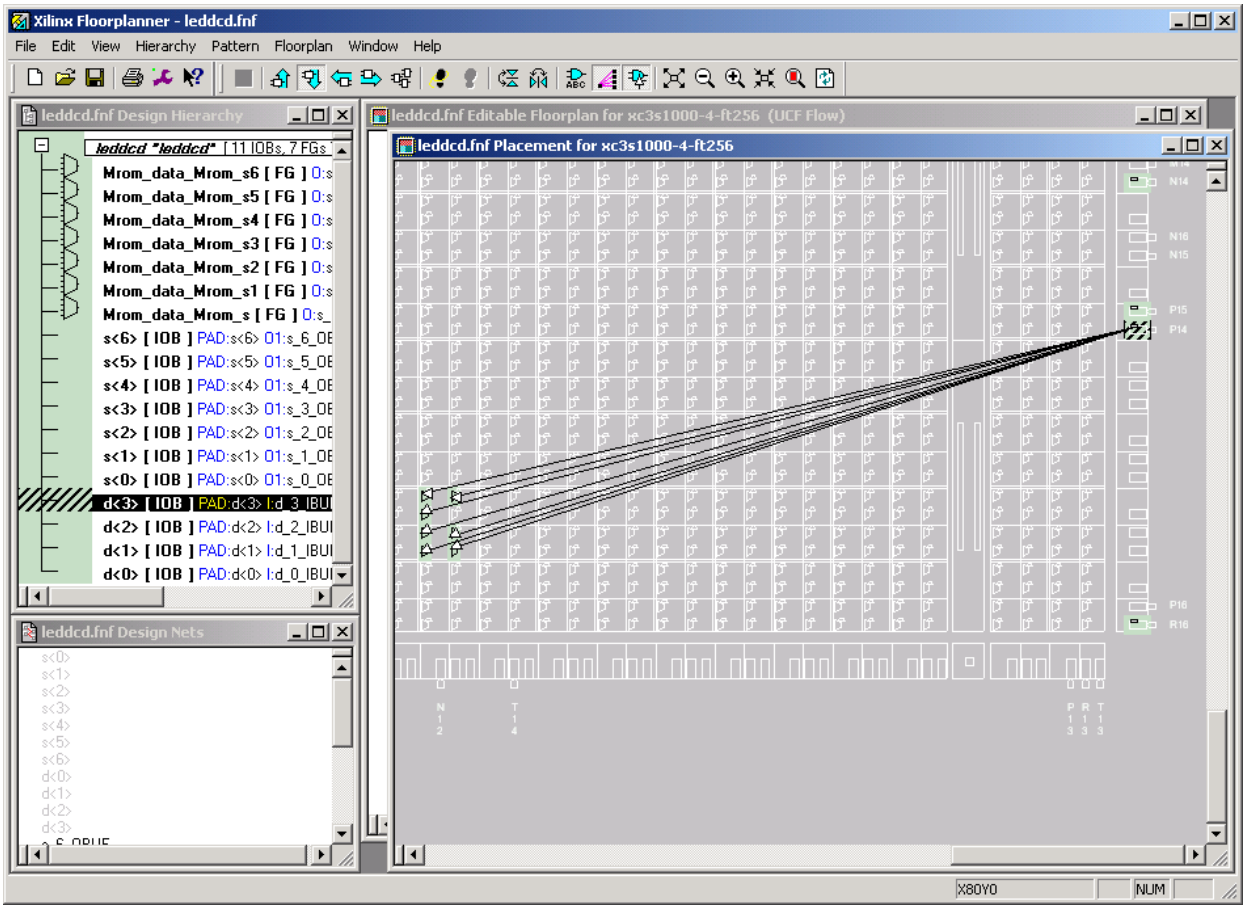
You can enable the display of the connections between I/O pins and CLBs by selecting the Edit➔Preferences menu item and then checking the boxes in the Ratsnest tab of the **Edit Preferences** window as shown below.



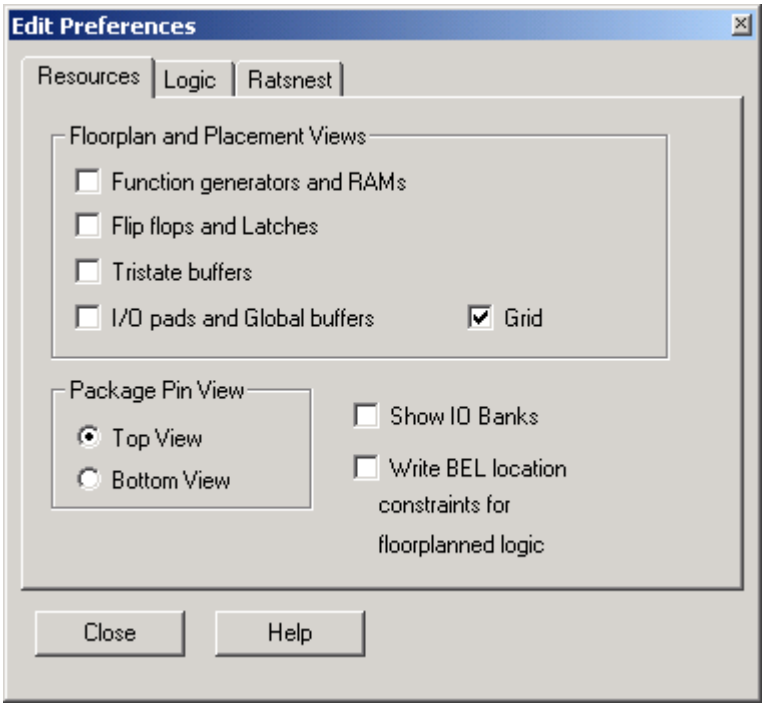
Now clicking on a CLB will highlight the nets connecting the inputs and output to the CLB.



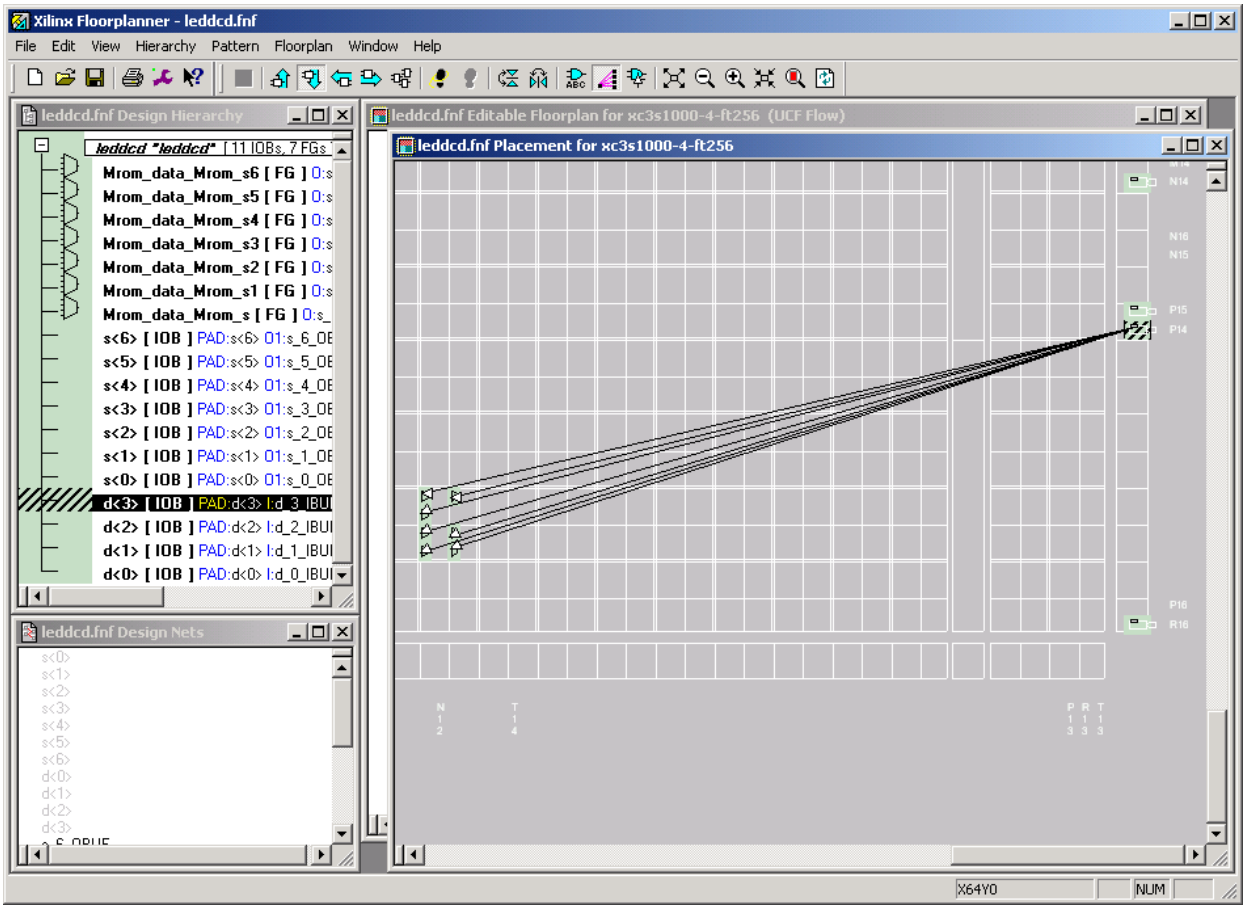
In an analogous manner, you can click on an input pin to highlight which CLBs are dependent on that input. (For this design, each input affects every CLB.)



The Placement pane may be showing too many details of the FPGA internal structure. To view only the FPGA resources that are used by the design, select Edit➔Preferences and uncheck all the boxes in the Resources tab as shown below.



Now the Placement pane shows only the LUTs and I/O pins that are used in this design.

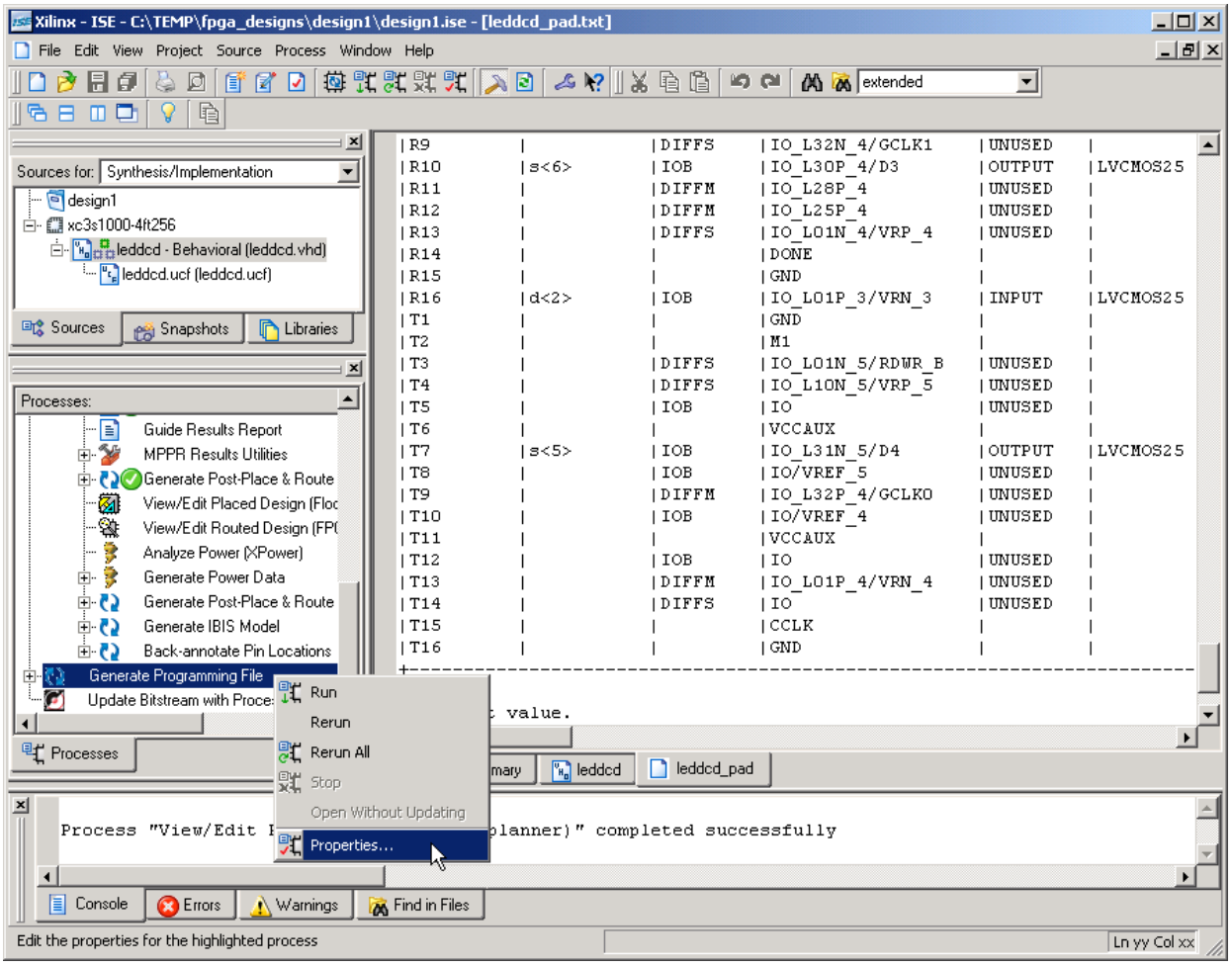


Viewing the placement of circuit elements after the place & route process can give you insights into the resource usage of certain VHDL language constructs. In addition to viewing the placement of the design, the Floorplanner can be used to re-arrange and optimize the placement. This is akin to the software technique of hand-optimizing assembly code output by a compiler. You won't do this here, but it is an option for designs which push at the limits of the capabilities of FPGAs.

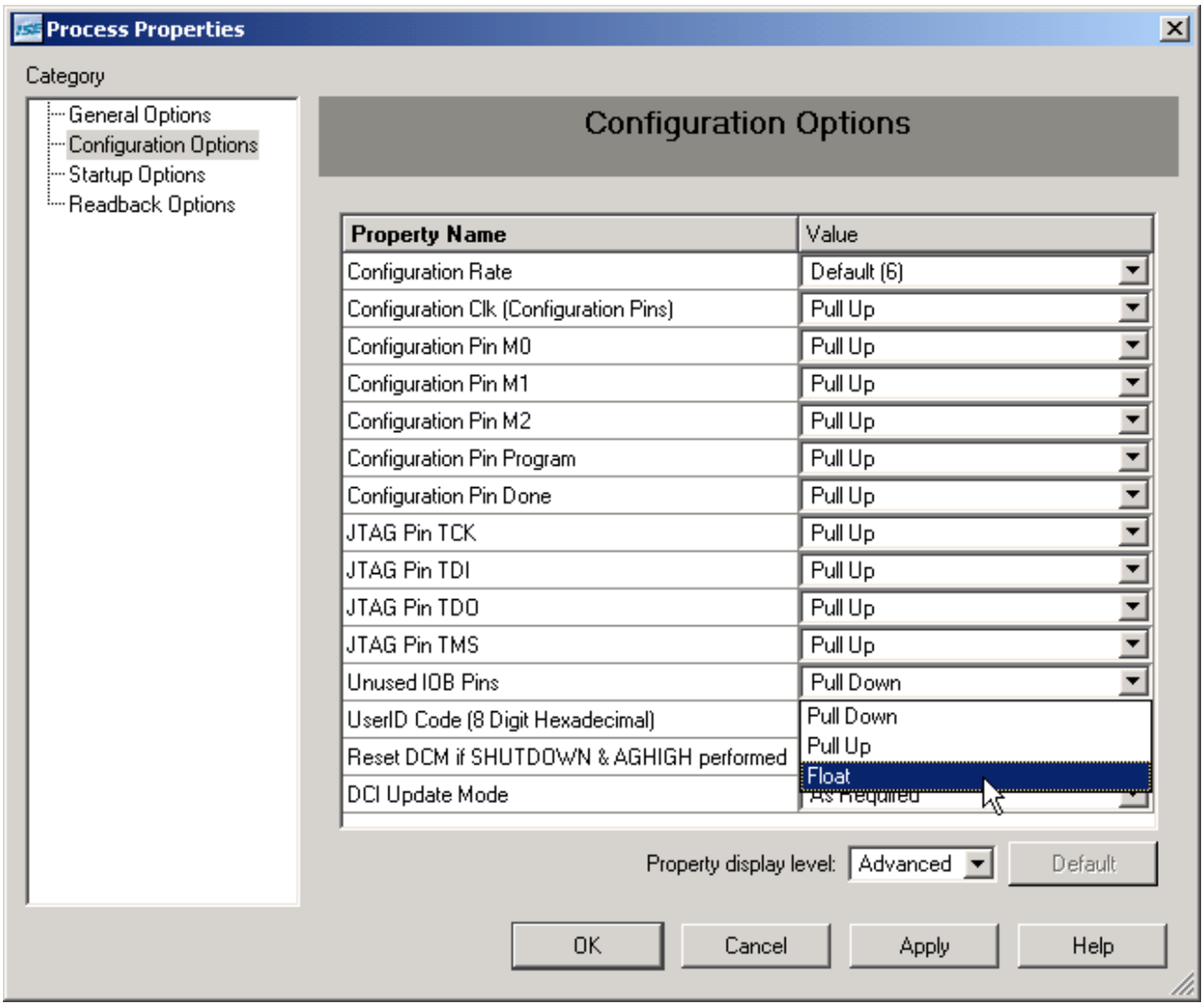
Generating the Bitstream

Now that you have synthesized our design and mapped it to the FPGA with the correct pin assignments, you are ready to generate the bitstream that is used to program the actual chip.

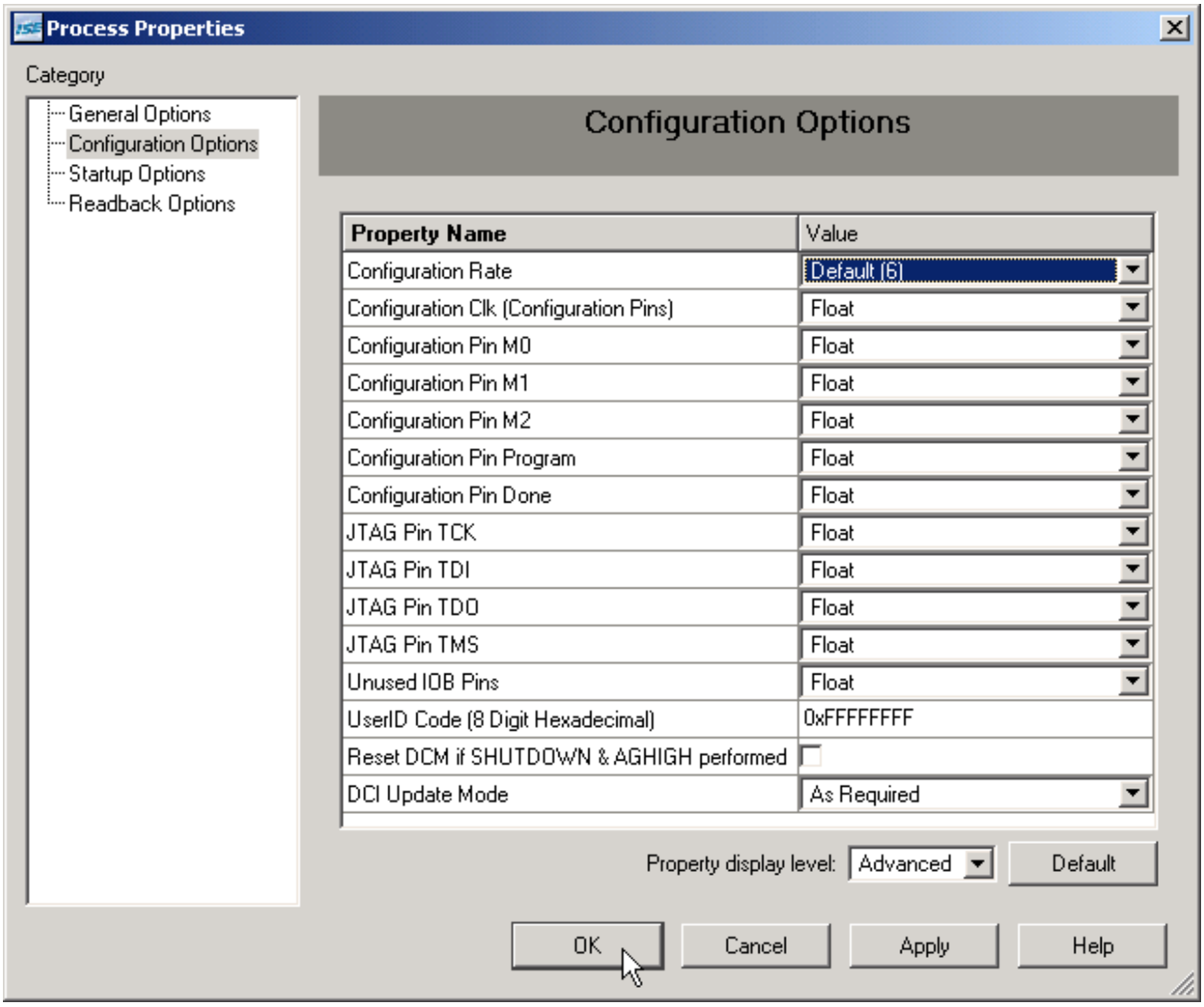
When using the XSA-3S1000 Board, you need to set some options before generating the bitstream. (This is not needed if you are using the XSA-50, XSA-100 or XSA-200 Board, but it wouldn't hurt, either.) Right-click on the Generate Programming File process and select Properties... from the pop-up menu.



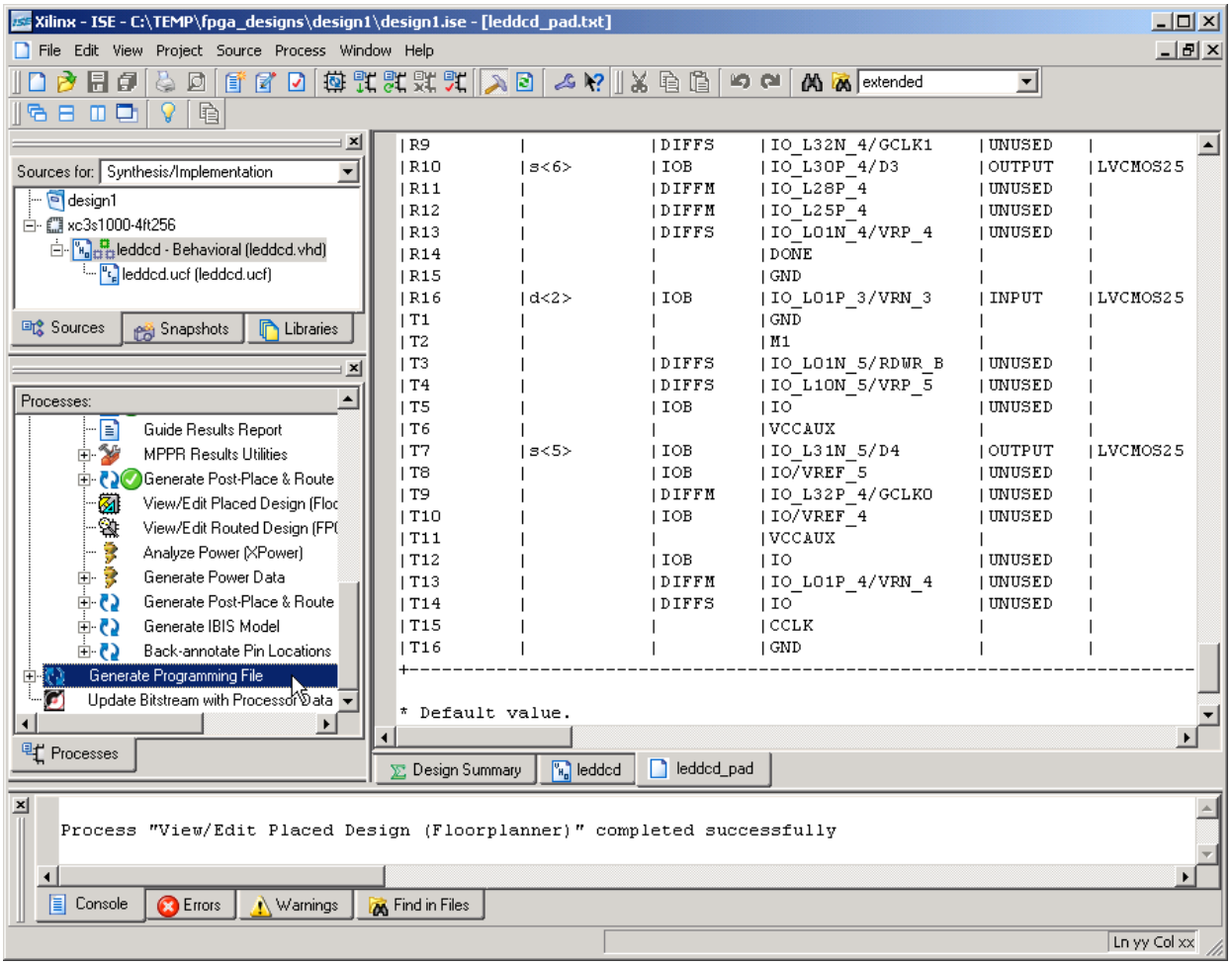
Now select the Configuration Options tab in the **Process Properties** window and set all the Pull Up and Pull Down values to Float to disable the internal resistors of the FPGA. (At the minimum, the Unused IOB Pins value must be set to Float, but it is best to set them all to Float.) If they are not disabled, the strong internal pull-up and pull-down resistors in the Spartan3 FPGA will overpower the external resistors on the XSA Board.



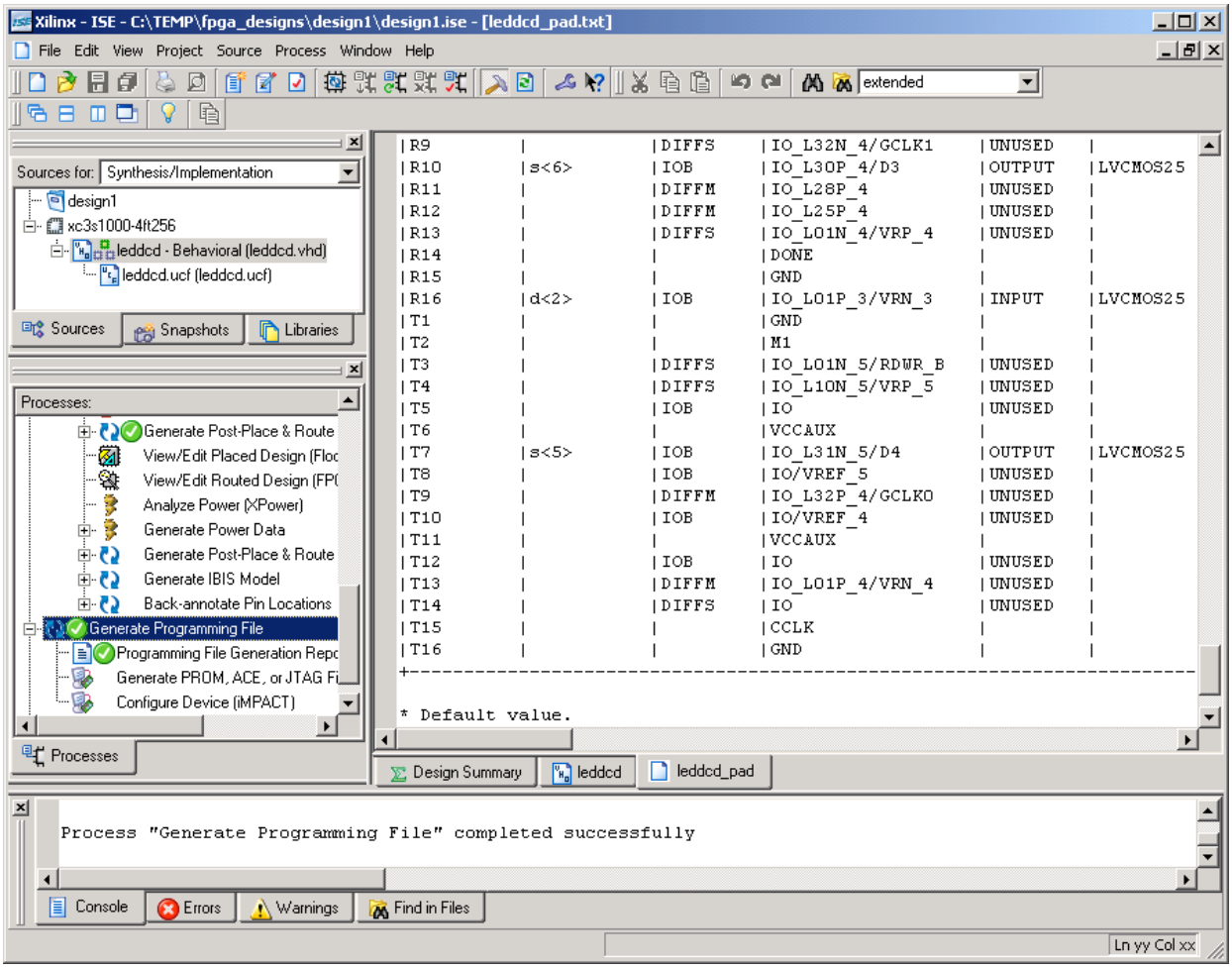
After setting all the values as shown below, click on OK.



Once you have set the bitstream generation options, highlight the leddcd object in the Sources pane and double-click on the Generate Programming File process.

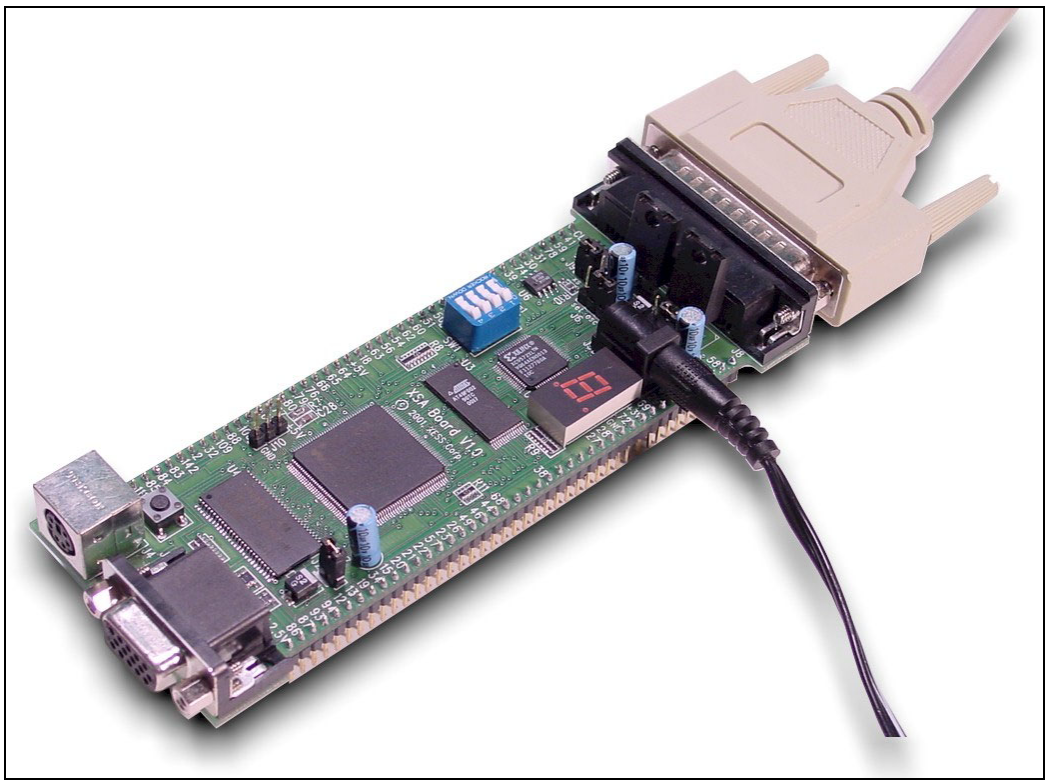


Within a few seconds, a ✓ will appear next to the Generate Programming File process and a file detailing the bitstream generation process will be created. A bitstream file named leddcd.bit can now be found in the design1 folder.



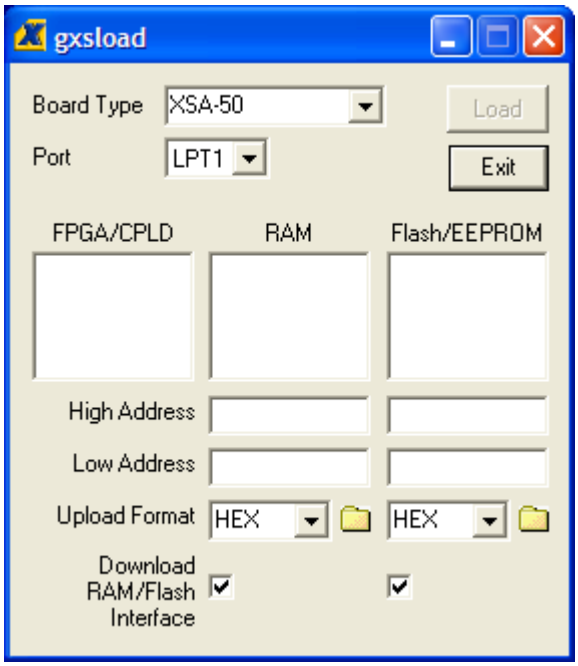
Downloading the Bitstream

Now you have to get the bitstream file programmed into the FPGA on the XSA Board. The XSA Board is powered with a DC power supply and is attached to the PC parallel port with a standard 25-wire cable as shown below.

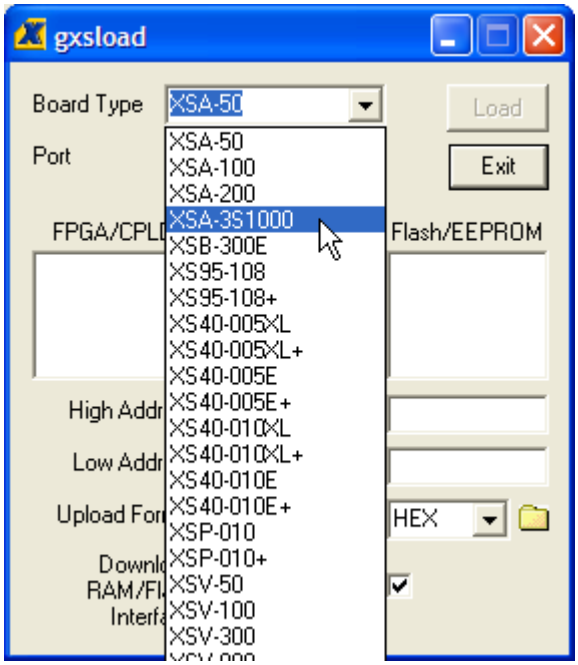


The XSA Boards are programmed using the gxsload utility. Double click the 

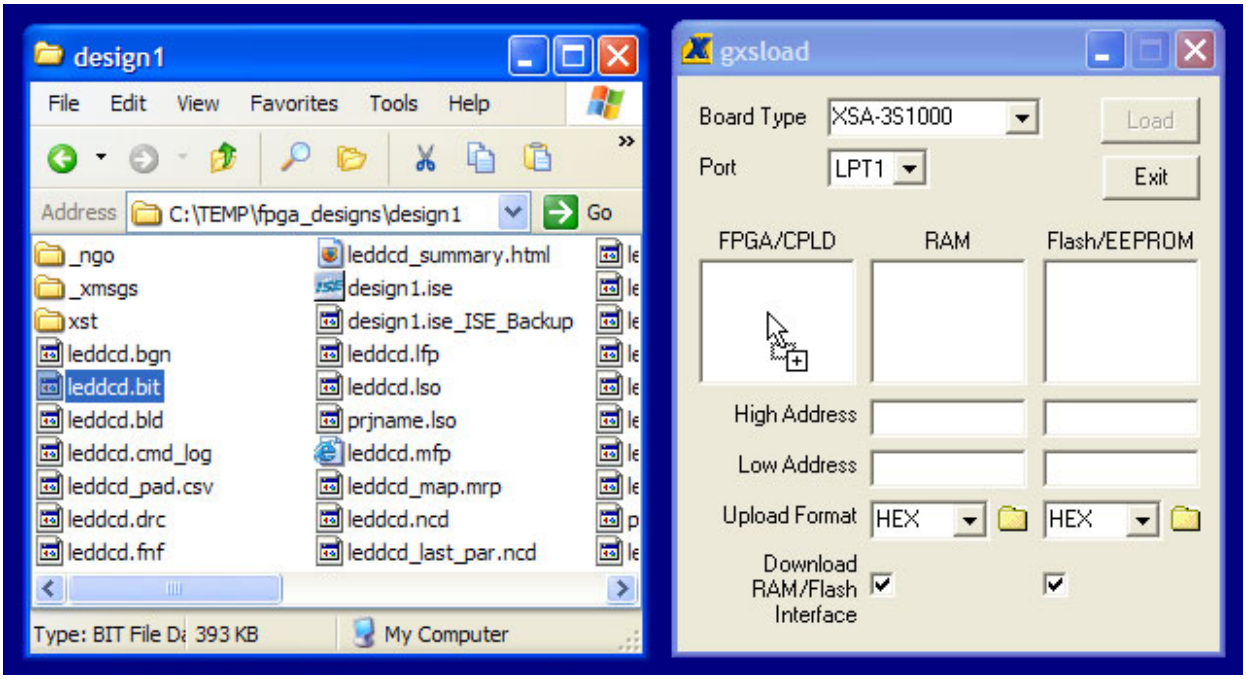
icon to bring up the **gxsload** window:



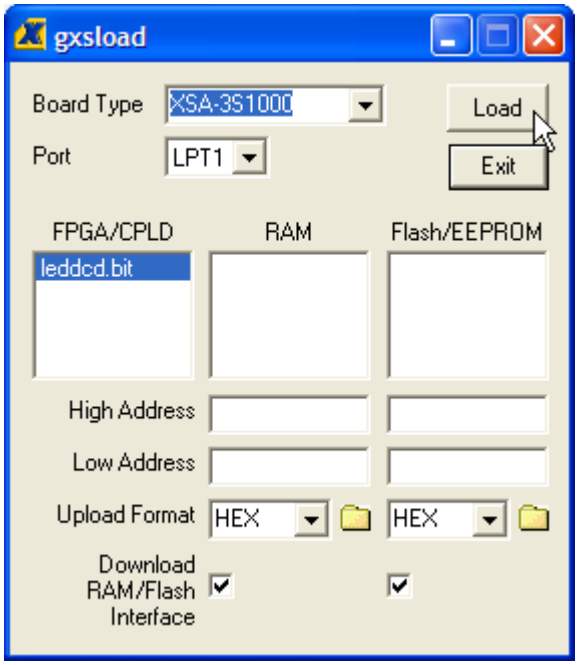
Then click in the Board Type field and select XSA-3S1000 from the drop-down menu since this is the board you are going to load with the bitstream.



Then open a window that shows the contents of the folder where you stored our LED decoder design (C:\tmp\fpga_designs\design1 in this case). You just drag-and-drop the leddcd.bit file from the **design1** window into the FPGA/CPLD pane of the **gxsload** window.



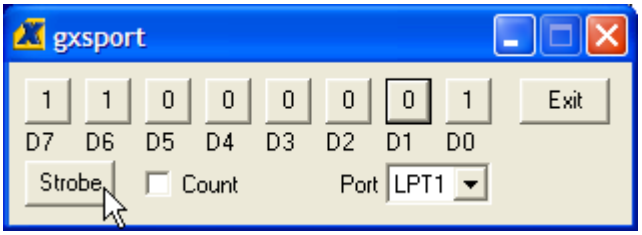
Then you click on the Load button to initiate the programming of the FPGA. Downloading the leddcd.bit file to the XSA Board takes only a few seconds.



Testing the Circuit

Once the FPGA on the XSA Board is programmed, you can begin testing the LED decoder. The eight data pins of the PC parallel port connect to the FPGA through the downloading cable. You have assigned the inputs of the LED decoder to pins, which are connected to the parallel port data pins. The gxsport utility lets you control the logic values on these pins. By placing different bit patterns on the pins, you can observe the outputs of the LED decoder through the seven-segment LED on the XSA Board.

Double-clicking the icon initiates the gxsport utility. The **d0**, **d1**, **d2**, and **d3** inputs of the LED decoder are assigned to the pins controlled by the D0, D1, D2, and D3 buttons of the **gxsport** window. To apply a given input bit pattern to the LED decoder, click on the D buttons to toggle their values. Then click on the Strobe button to send the new bit pattern to the pins of the parallel port and on to the FPGA. For example, setting (D3,D2,D1,D0) = (1,1,1,0) will cause E to appear on the seven-segment LED of the XSA Board.



If you check the Count box in the **gxsport** window, then each click on the Strobe button increments the eight-bit value represented by D7-D0. This makes it easy to check all sixteen input combinations.

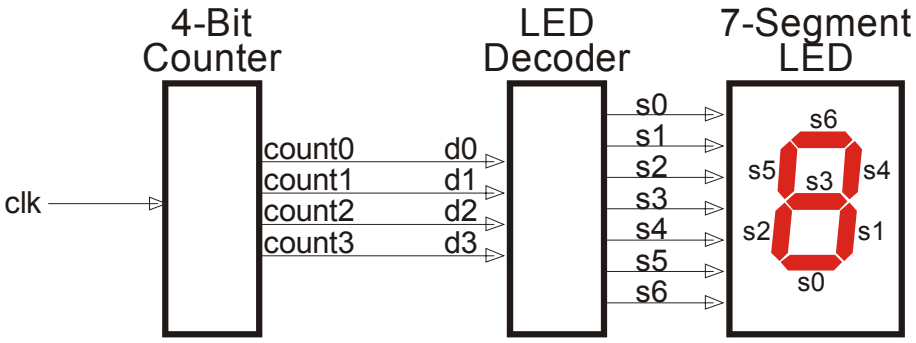
NOTE: Bit D7 of the parallel port controls the /PROGRAM pin of the FPGA. Do not set D7 to 0 or you will erase the configuration of the FPGA. Then you will have to download the bitstream again to continue testing your design.

4

Hierarchical Design

A Displayable Counter

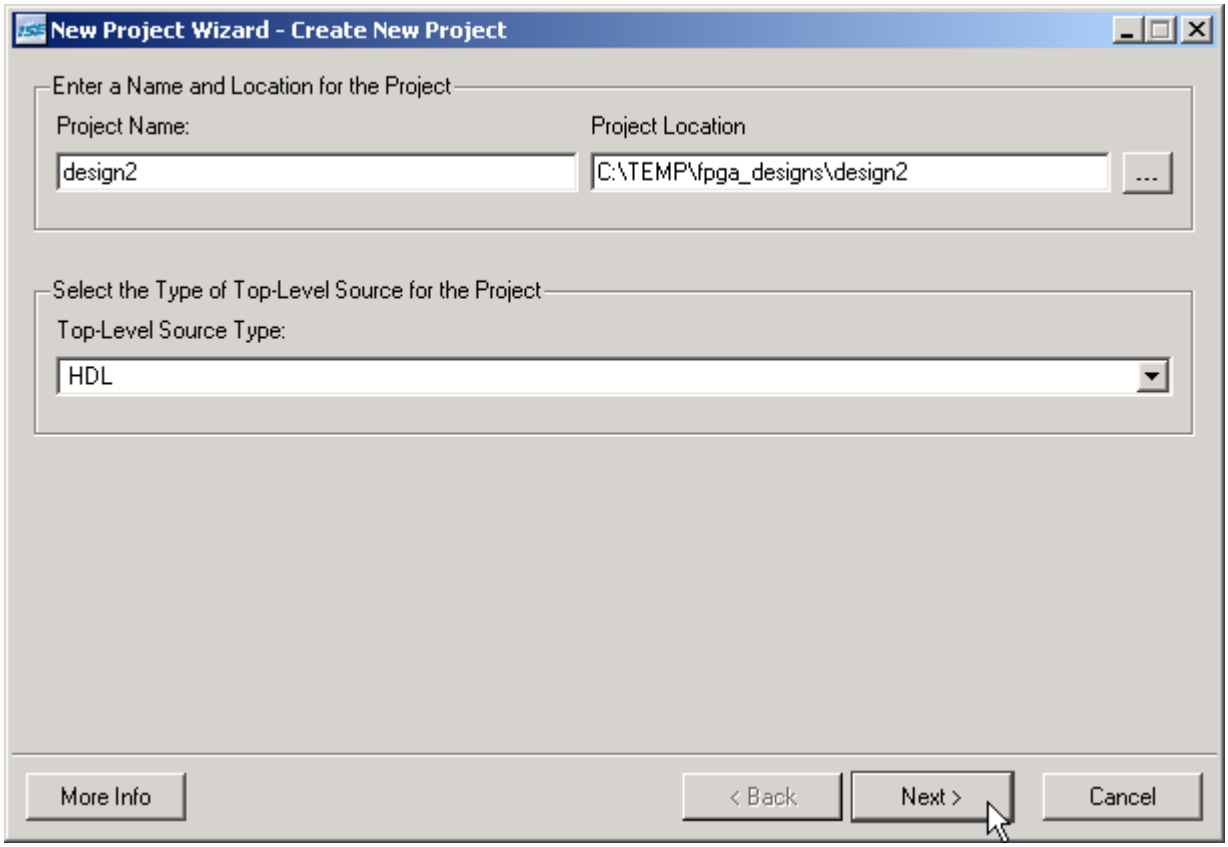
You went through a lot of work for your first FPGA design, so you will reuse it in this design: a four-bit counter whose value is displayed on a seven-segment display. The counter will increment on the falling edge of the clock. The four-bit output from the counter enters the LED decoder whereupon the counter value is displayed on the seven-segment LED. A high-level diagram of the displayable counter looks like this:



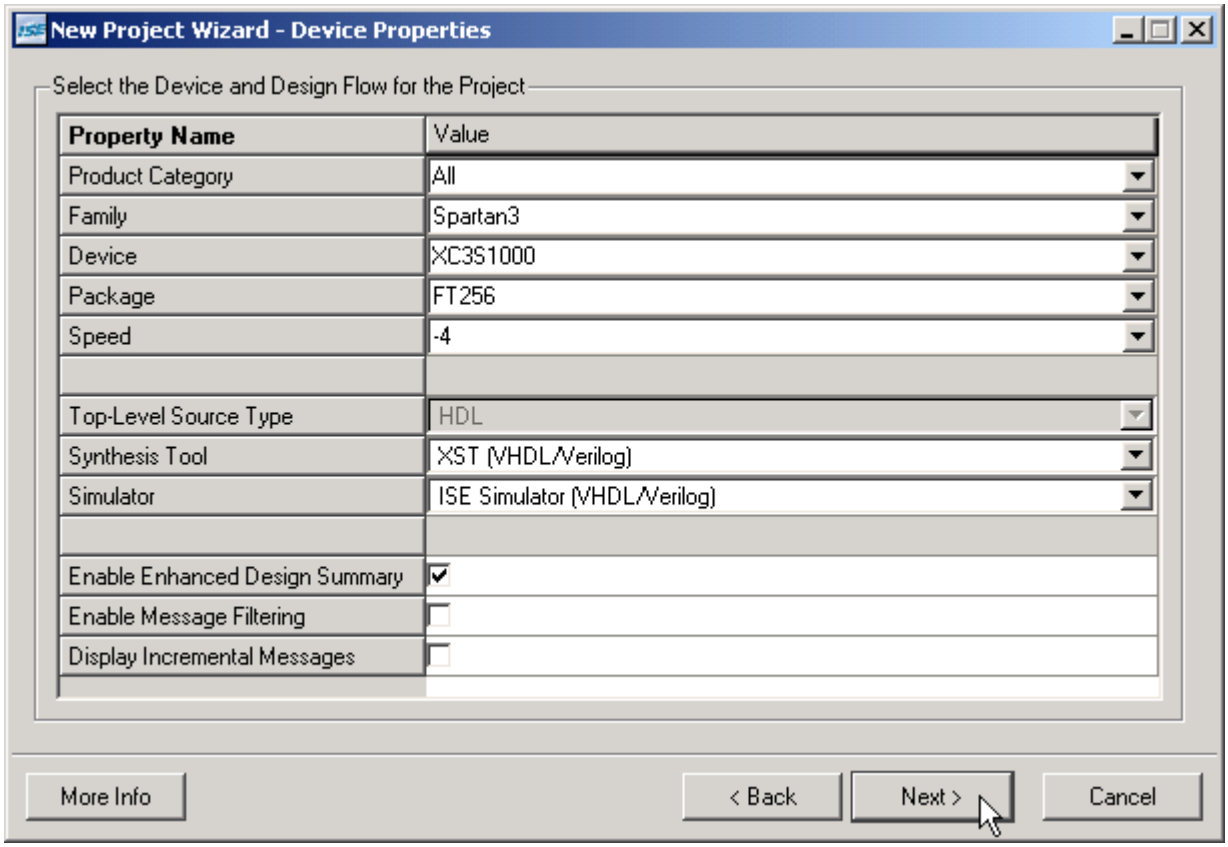
This design is hierarchical in nature. The LED decoder and counter are modules, which are interconnected within a top-level module.

Starting a New Design

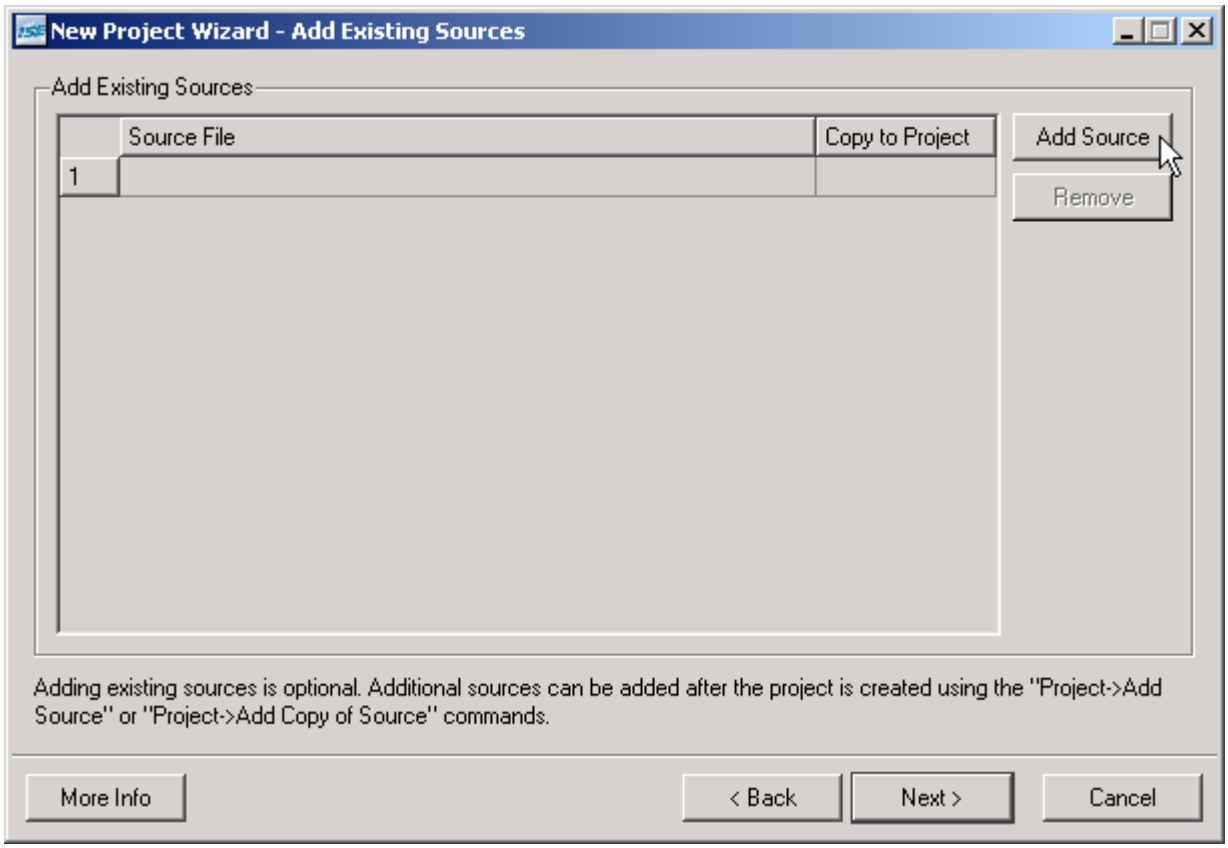
You can start a new project using the File➔New Project... menu item. Name the project ***design2*** and store it in the same folder as the previous design: C:\tmp\fpga_designs. Then click on the Next button.



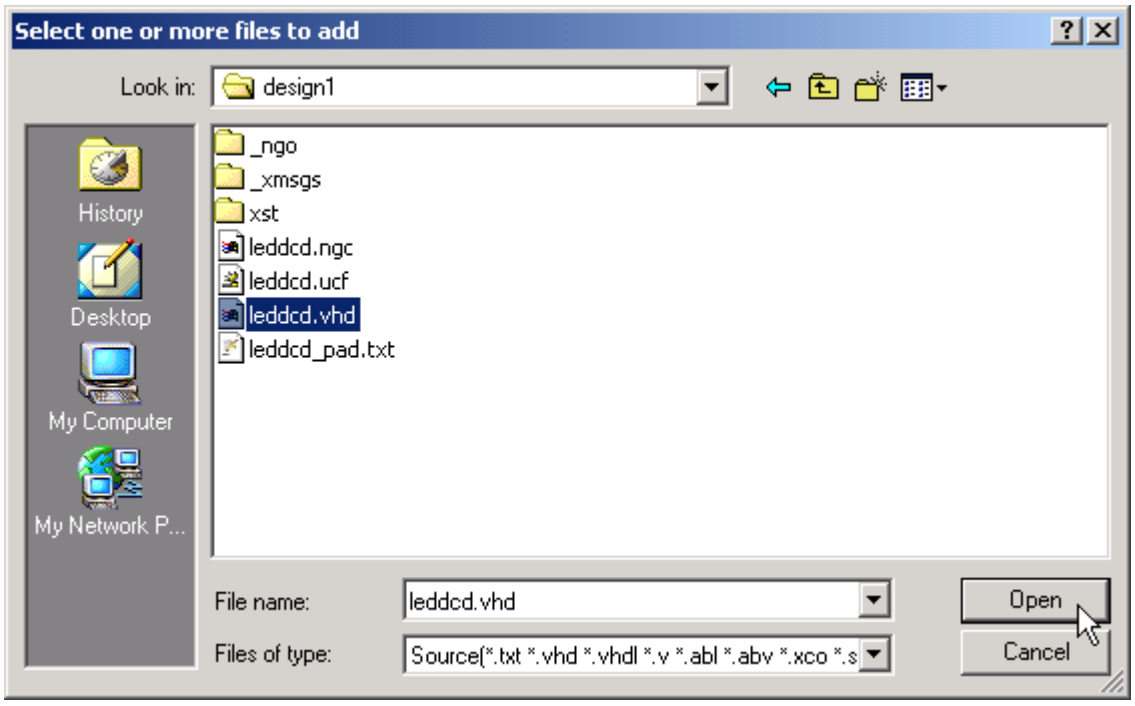
You will target the same FPGA on the XSA Board, so the other properties in the **New Project** window retain the same values you set in the previous project.



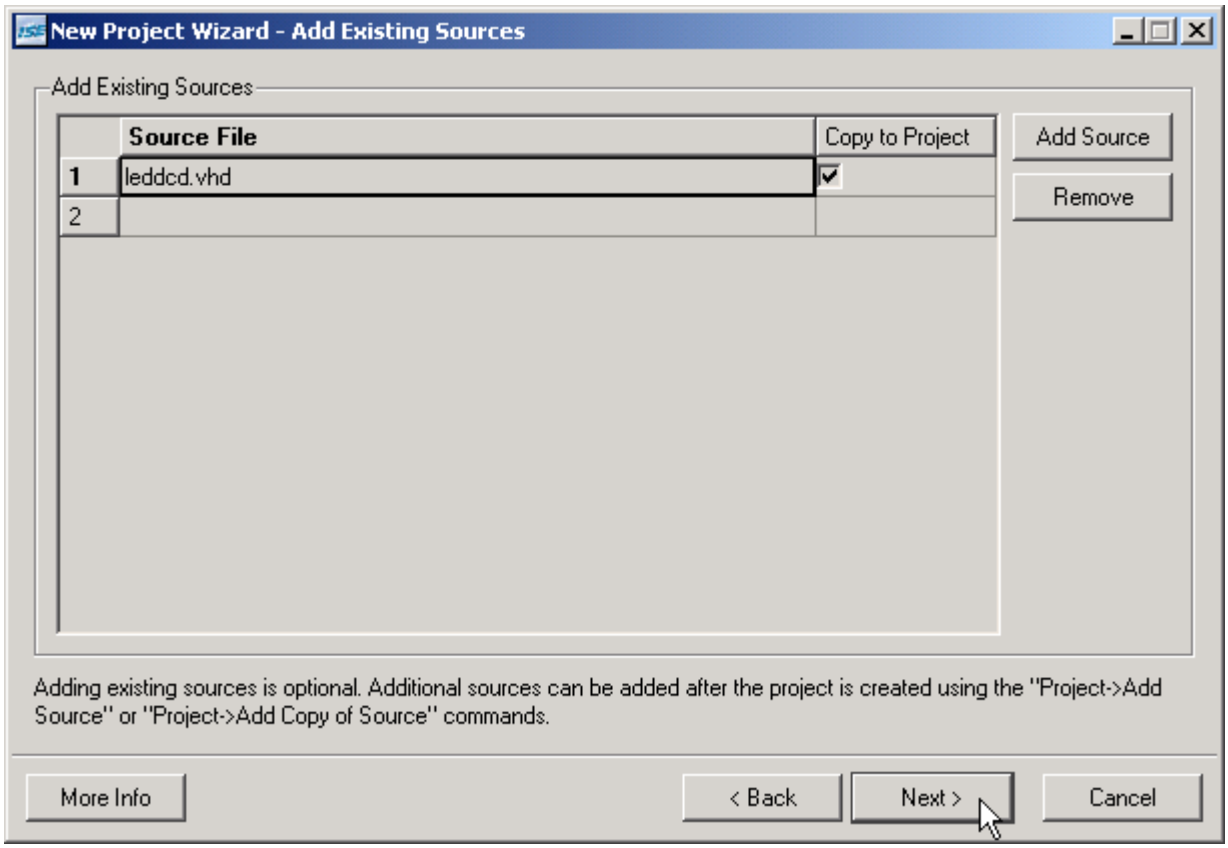
You won't add any new source files at the moment, so just click on the Next button on the **Create New Source** window. This brings up the **Add Source** window shown below that lets you add existing source files to your new project. It will save time if you re-use the LED decoder from the previous design, so click on the Add Source... button.



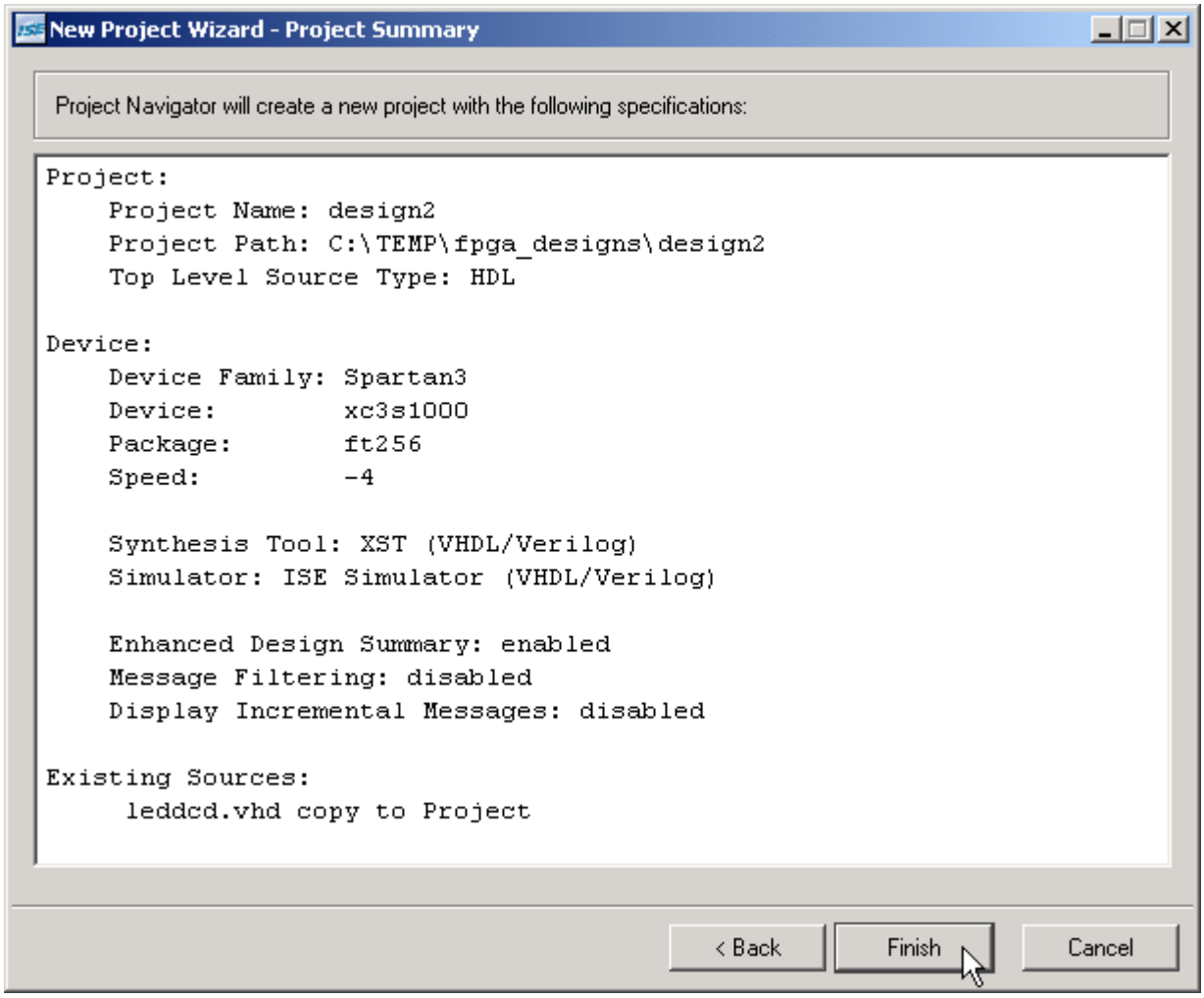
The **Add Existing Sources** window appears. Move to the C:\TEMP\fpga_designs\design1 folder and highlight the leddcd.vhd file that contains the VHDL source code for the LED decoder.



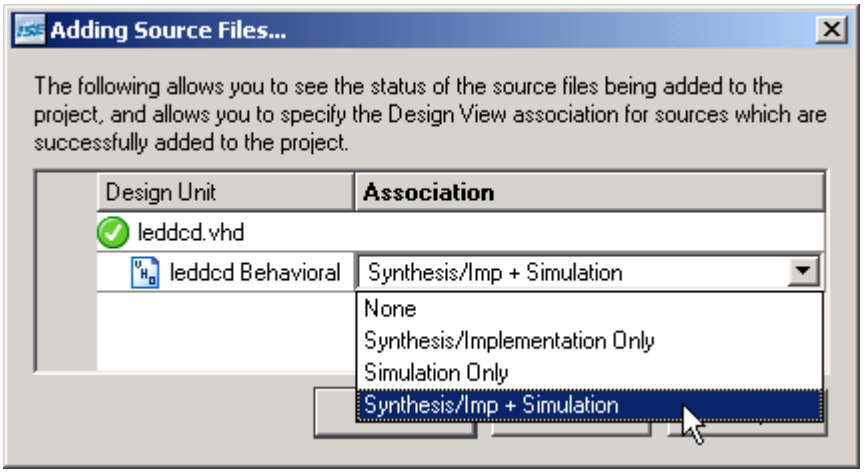
The **New Project** window now shows that a copy of the leddcd.vhd file will be added to the project. The Copy to Project box is checked, so the leddcd.vhd file will be copied from the design1 folder to the design2 folder. Unchecking this box will cause the ***design2*** project to link directly to the leddcd.vhd file in the design1 folder, so any change in this file will affect both projects. For this example, I have chosen to make a separate copy so the box remains checked. This is the only existing file you need to add, so click on the Next button to move on to the next window.



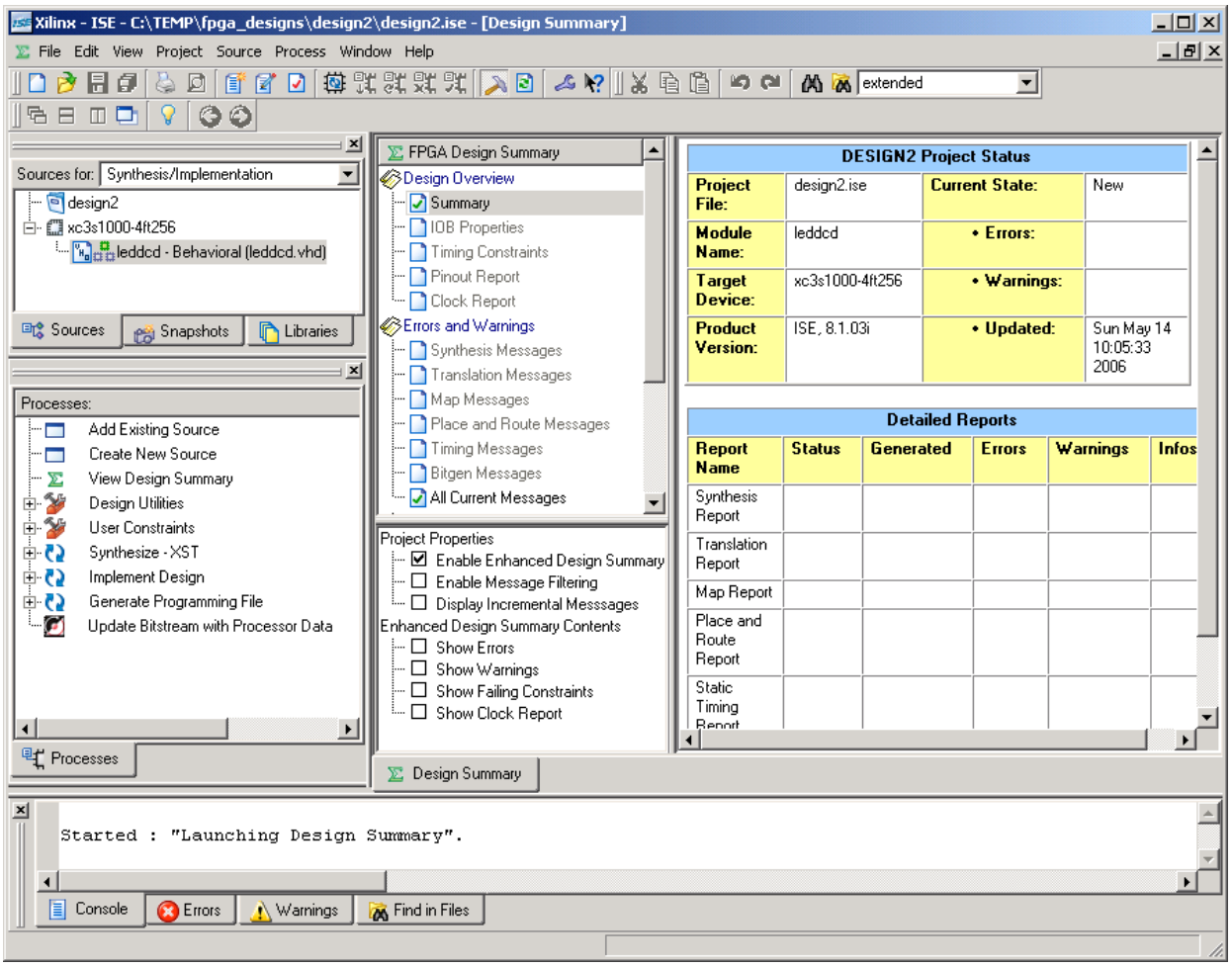
The final screen shows the pertinent information for the new project. Click on the Finish button to complete the creation of the project.



An **Adding Source Files…** window will appear that lets you specify the intended use for the files you are adding. Files can be used during synthesis and implementation of an FPGA bitstream as we did in the previous design. They can also be used only during simulation, such as testbench files that specify stimulus patterns that are driven onto the inputs of a design. Or they can be for both synthesis/implementation and simulation (as most are) because they describe the function and/or structure of the design in both these domains. There are also files, such as text files containing design documentation that are added to a project but are not used in either domain. I selected Synthesis/Imp + Simulation for the leddcd.vhd file even though it will only be used for synthesis and implementation in this example.

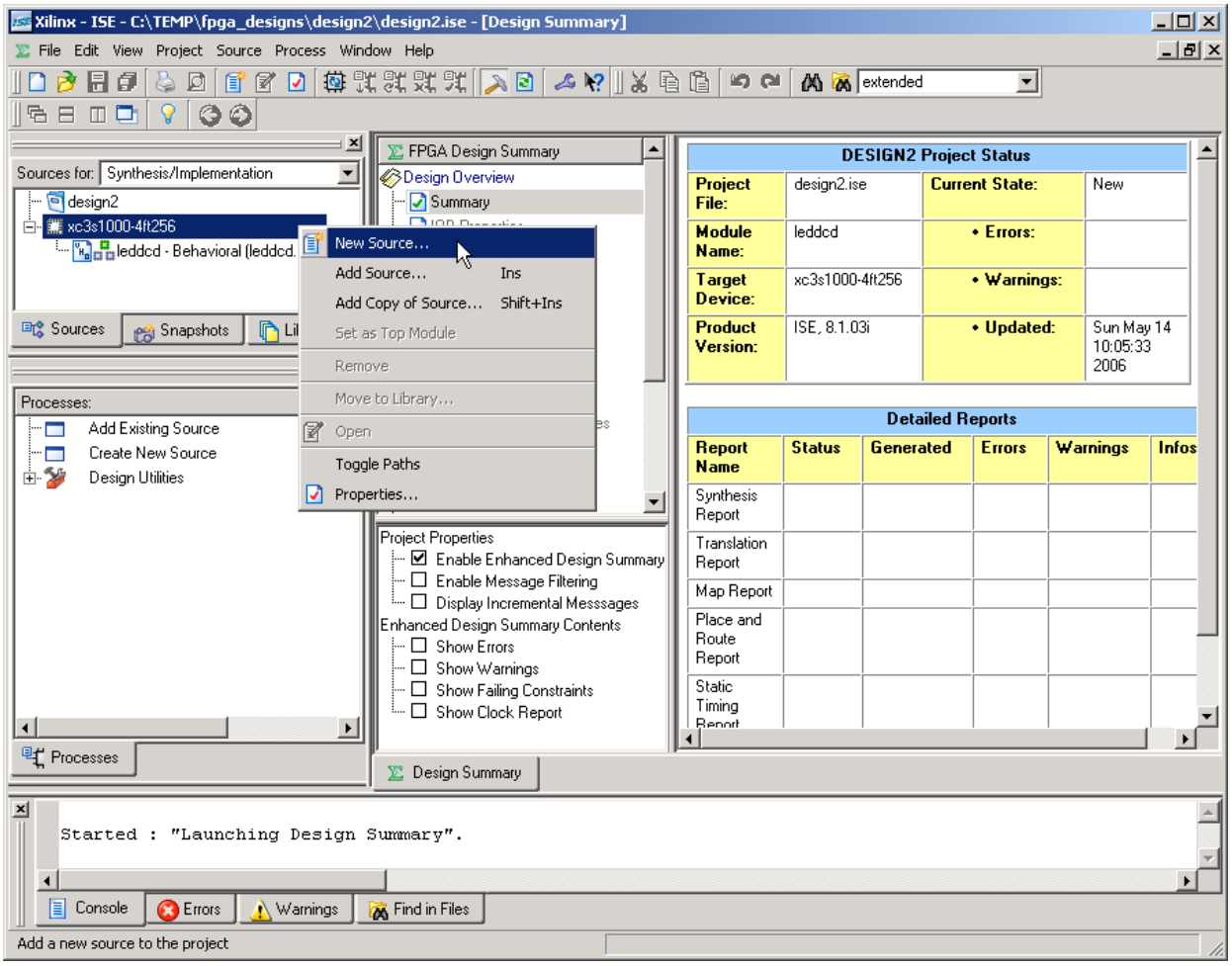


After you click on Finish in the **Adding Source Files…** window, the Sources pane contains only the single leddcd.vhd source file.

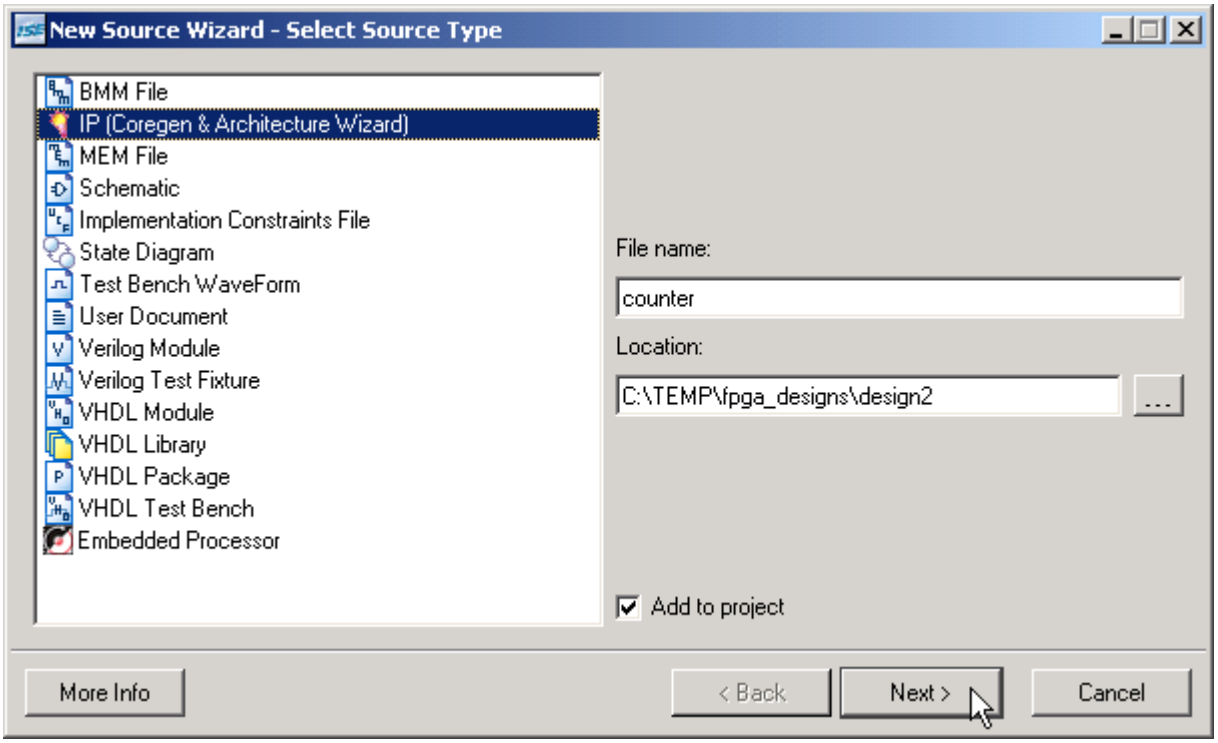


Adding a Counter

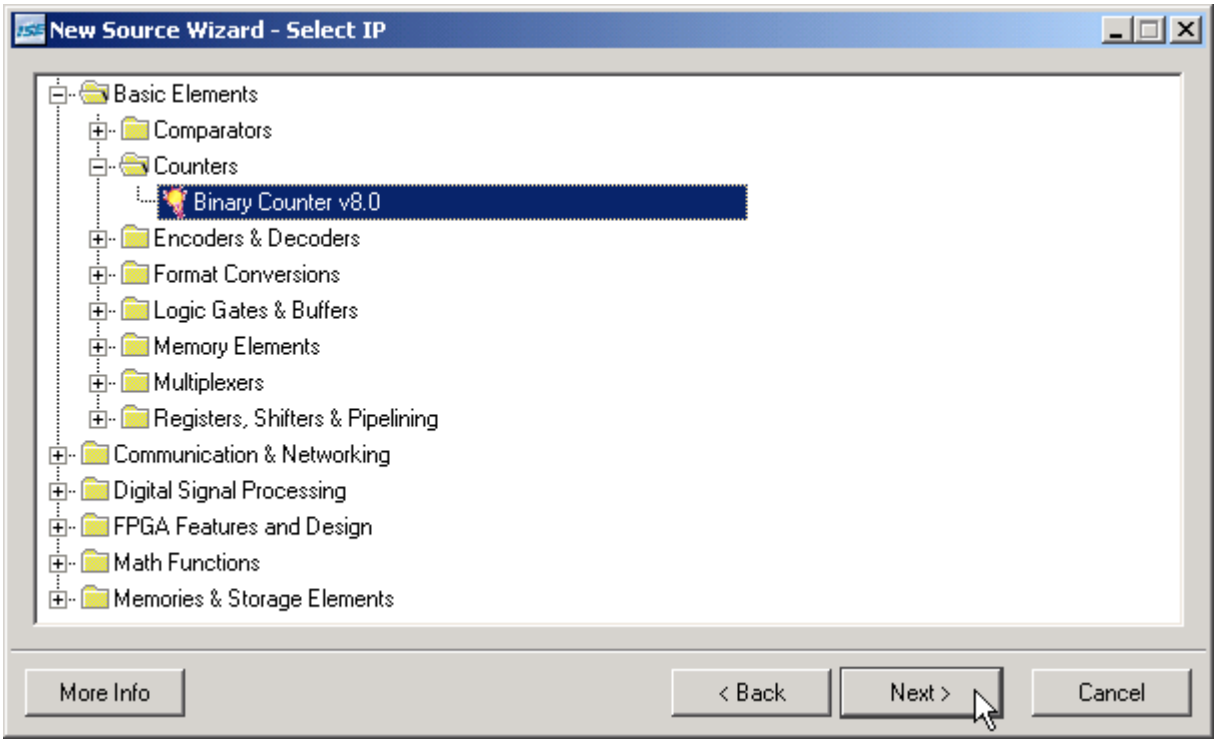
Now you have to add the counter to the design. There is no counter module yet, so you have to build one. Right-click on the xc3s1000-4ft256 object and select New Source... from the pop-up menu.



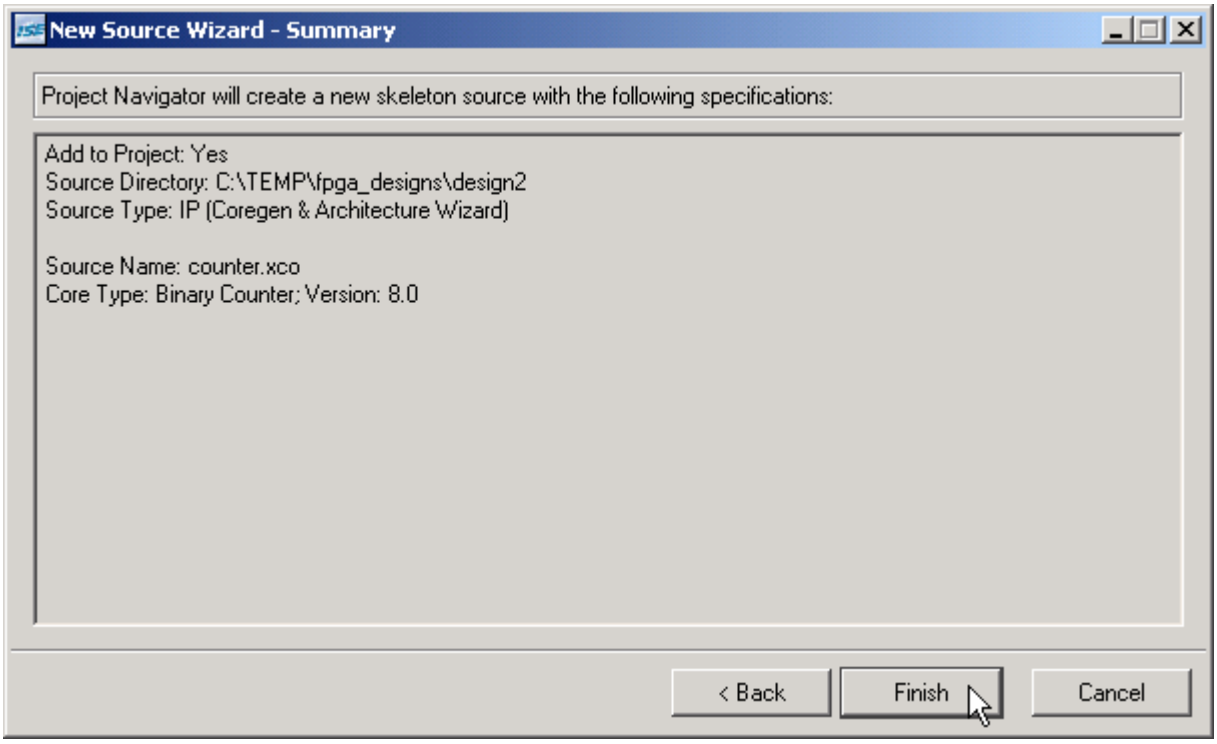
As in the previous example, you are prompted for the type of file to add to the project. This time, choose the IP (Coregen & Architecture Wizard) menu item. This will allow you to select a counter from a library of pre-built, configurable components. Then type `counter` into the File Name field and click on the Next button.



In the next window, expand the library tree until you find the binary counter component. Then click on Next.



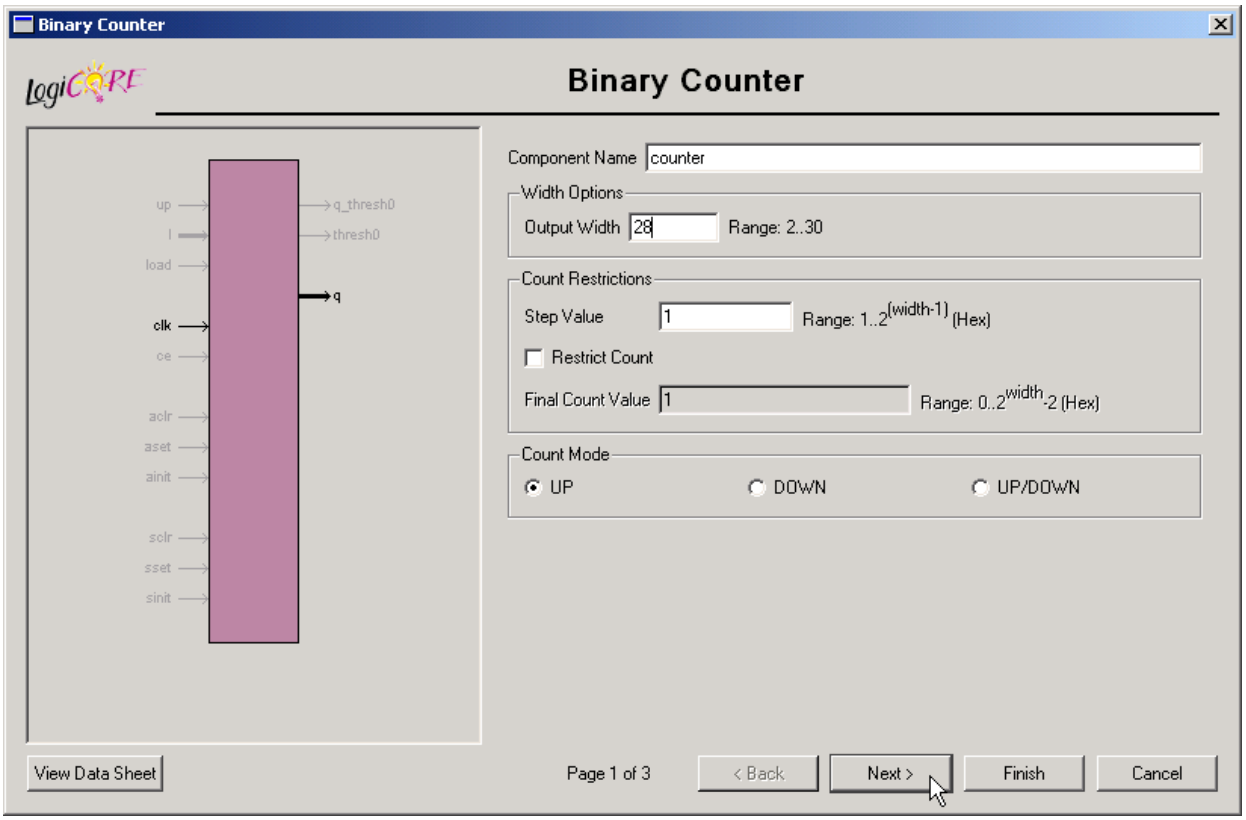
A very uninteresting summary of your choice appears. Click Finish to move on to the screens that will let you configure the counter component for this design.



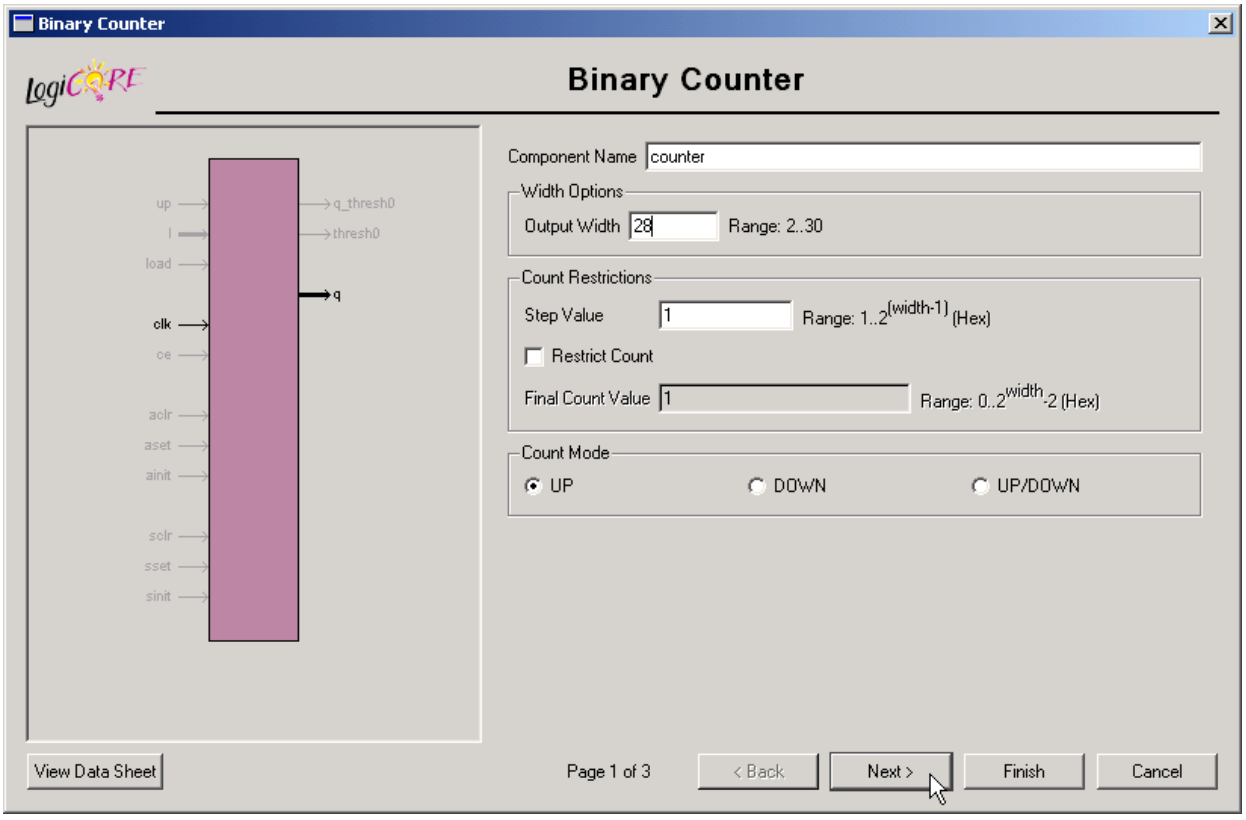
The initial configuration window specifies a sixteen-bit counter that increments its value upward by one on each clock cycle until it reaches 65,535 after which it rolls-over back to 0. You can change many of the features of this counter on the following screens. (Click on the View Data Sheet button to see all the details on how this counter can be configured.)

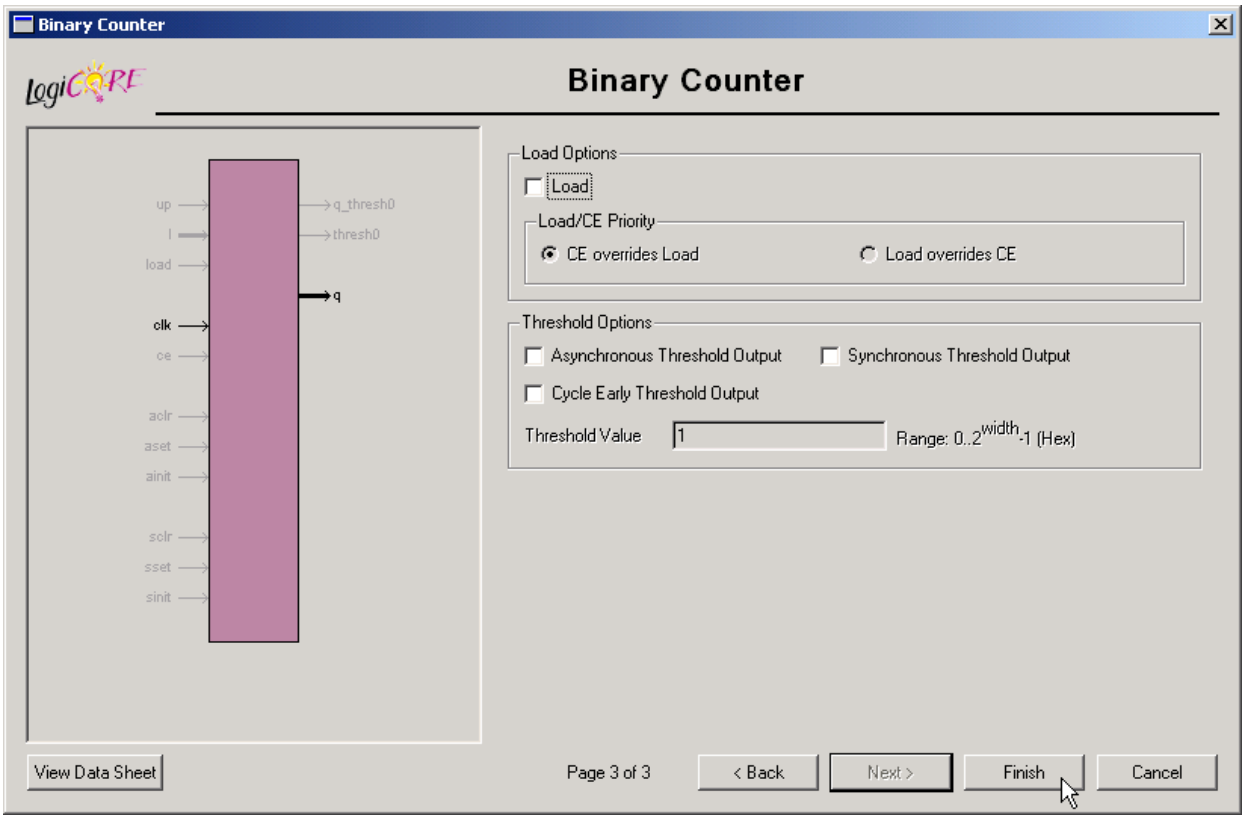
The only change that needs to be made for this design is to increase the counter width to 28 bits. Why build a 28-bit counter and when only the upper four bits are used? The counter will be driven by a clock signal on the XSA Board that has a frequency of 50 MHz. The LED display would be changing much too quickly to see at this frequency. By connecting the LED decoder to the upper four bits of the 28-bit counter, the display will only change once in every 2^{24} clock cycles. So the LED display will change every 2^{24} / (50 x 10^6) = 0.336 seconds which is slow enough to read.

After changing the counter width, click on Next to continue configuring the counter.

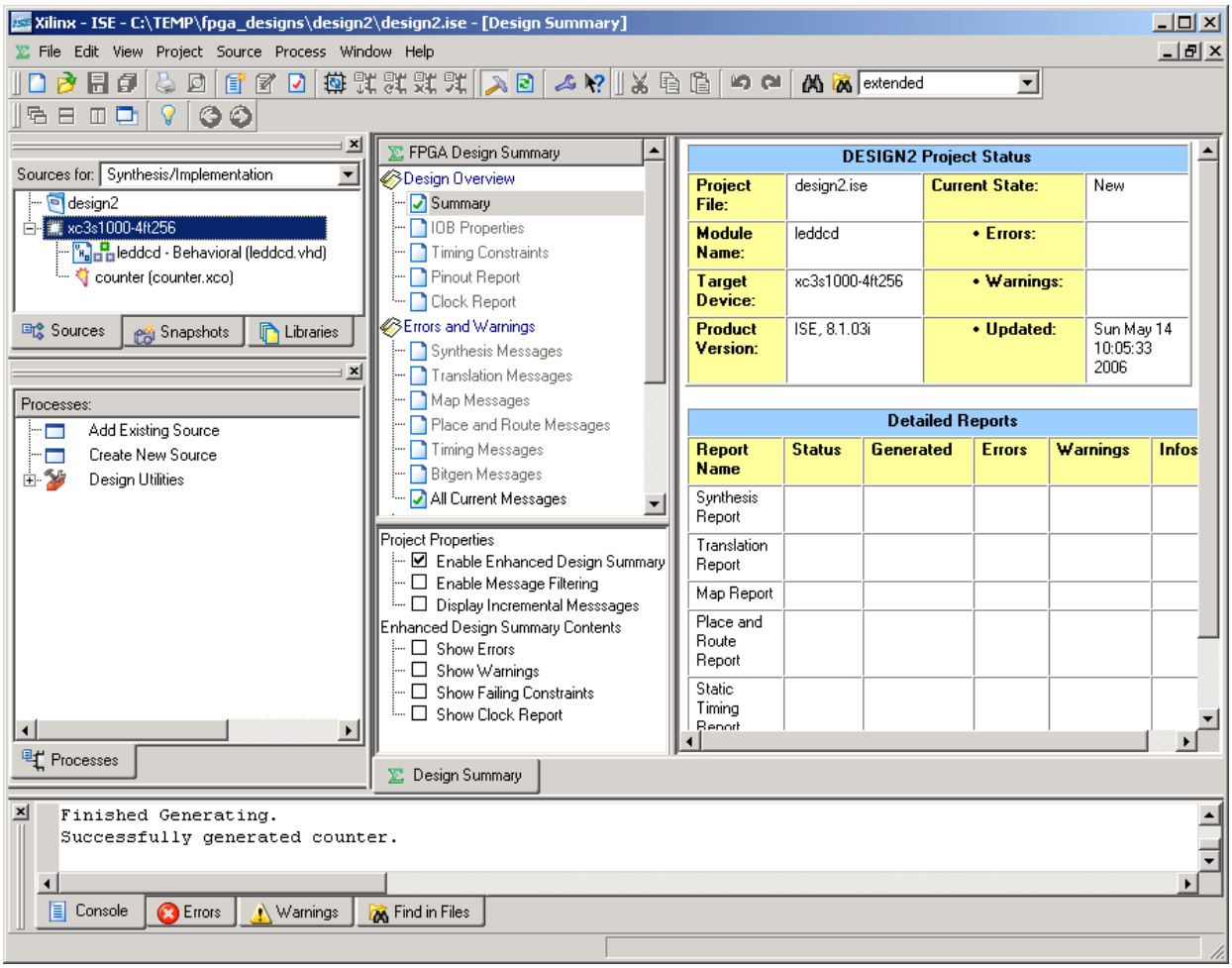


None of the advanced configuration options for the counter are needed in this design, so click Next and Finish in the next two windows that appear.



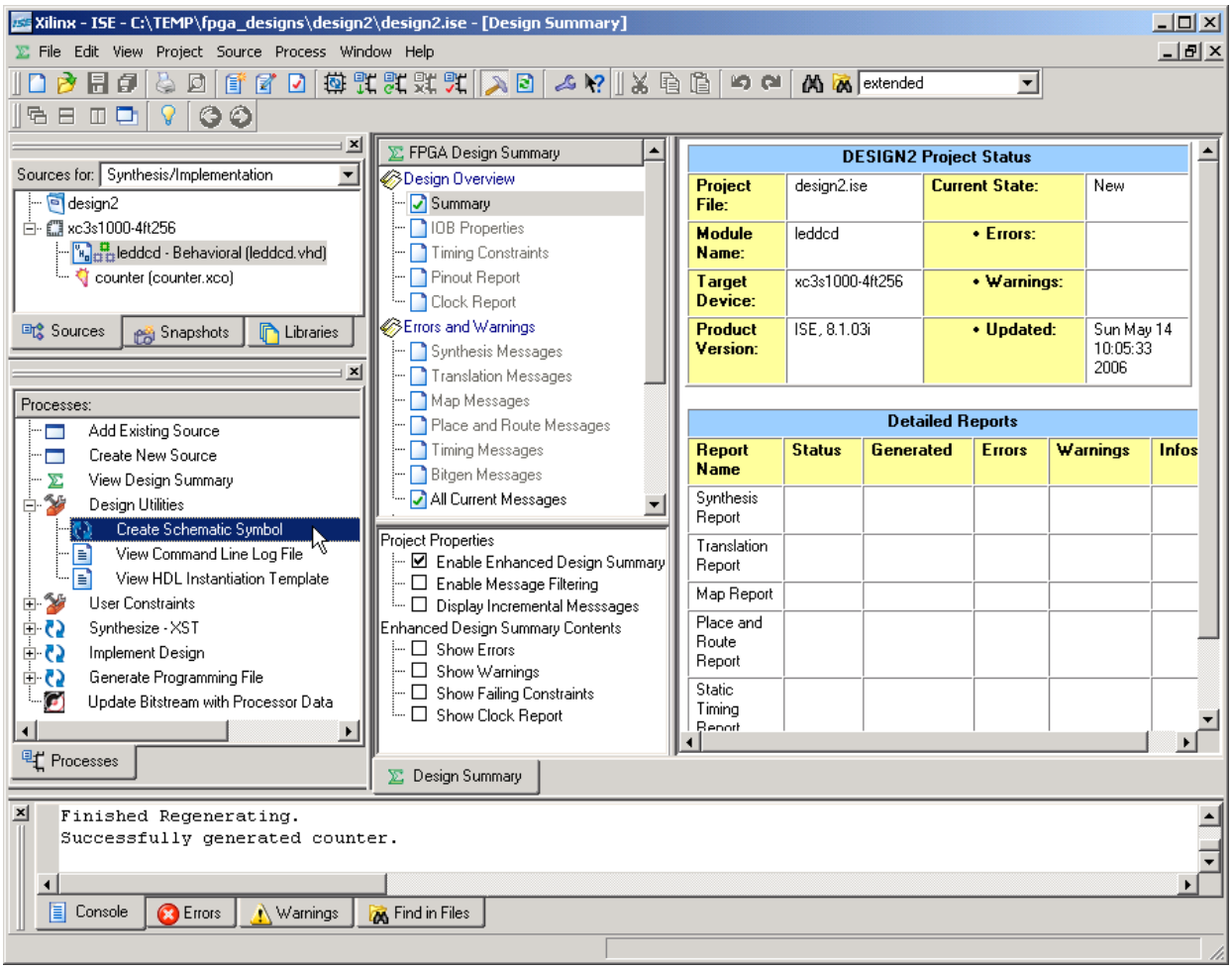


After clicking Finish in the final configuration window, the **counter** module appears in the Sources pane. The counter is stored as a Xilinx Coregen object (.xco file suffix). If you need to change the operation of the counter, just double-click the counter module and make your changes in the Coregen wizard windows that appear.

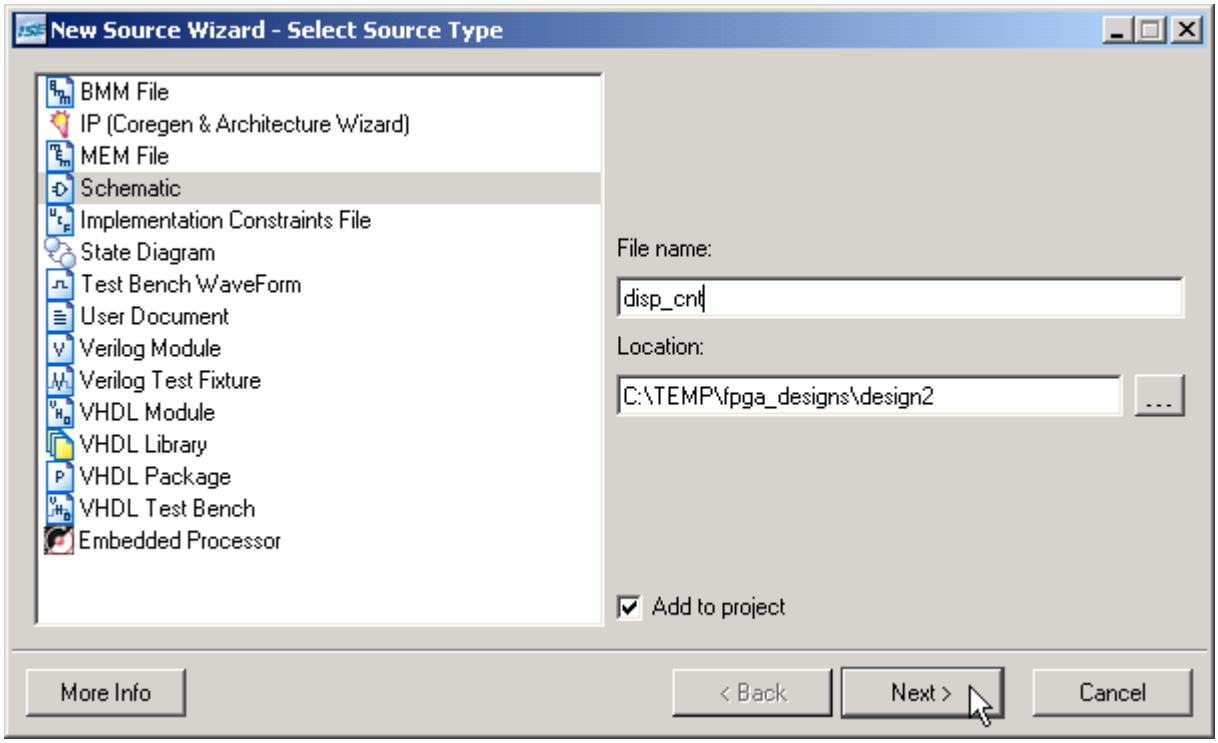


Tying Them Together

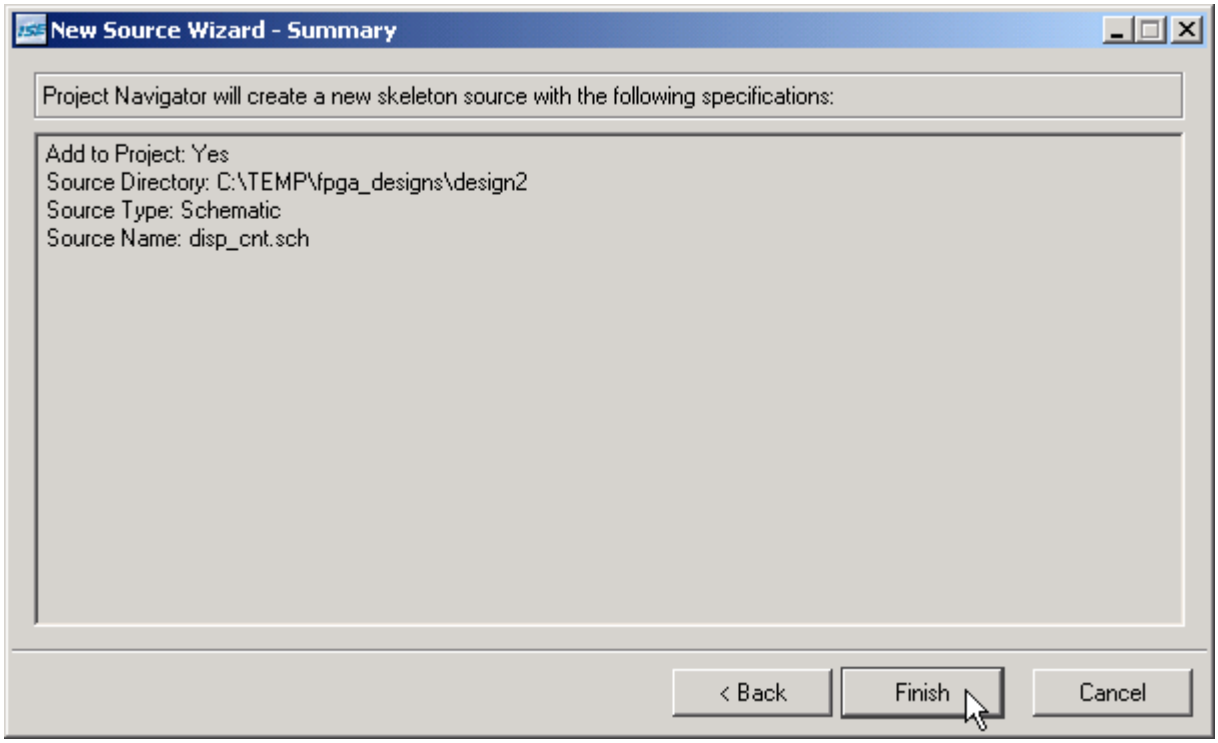
You have the LED decoder and the counter, but now you need to tie them together to build the displayable counter. You will do this by connecting the counter to the LED decoder in a top-level schematic. Before you can do this, you have to create a schematic symbol for the LED decoder module from its VHDL source code. (The counter module already has a schematic symbol since it was created using Coregen.) To create the LED decoder schematic symbol, highlight the leddcd object in the Sources pane and then double-click the Create Schematic Symbol process. A message indicating the schematic symbol has been created will appear in the Console tab of the Transcript pane.



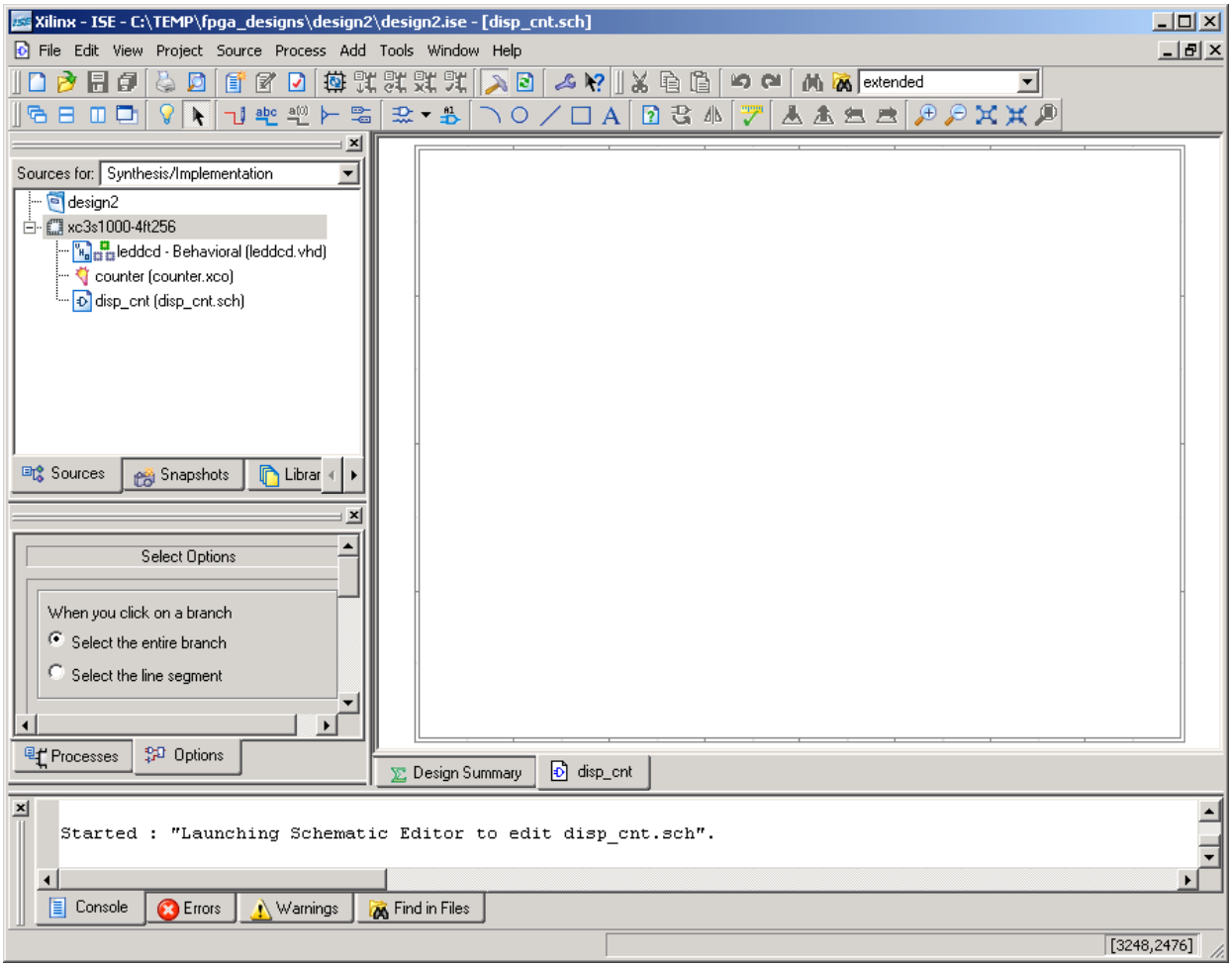
Now it is time to create the top-level schematic that will hold the counter and LED decoder symbols. Right-click on the xc3s1000-4ft256 object and select New Source... from the pop-up menu. Then highlight the Schematic entry in the **New Source** window and name the schematic **disp_cnt**. Then click on Next.



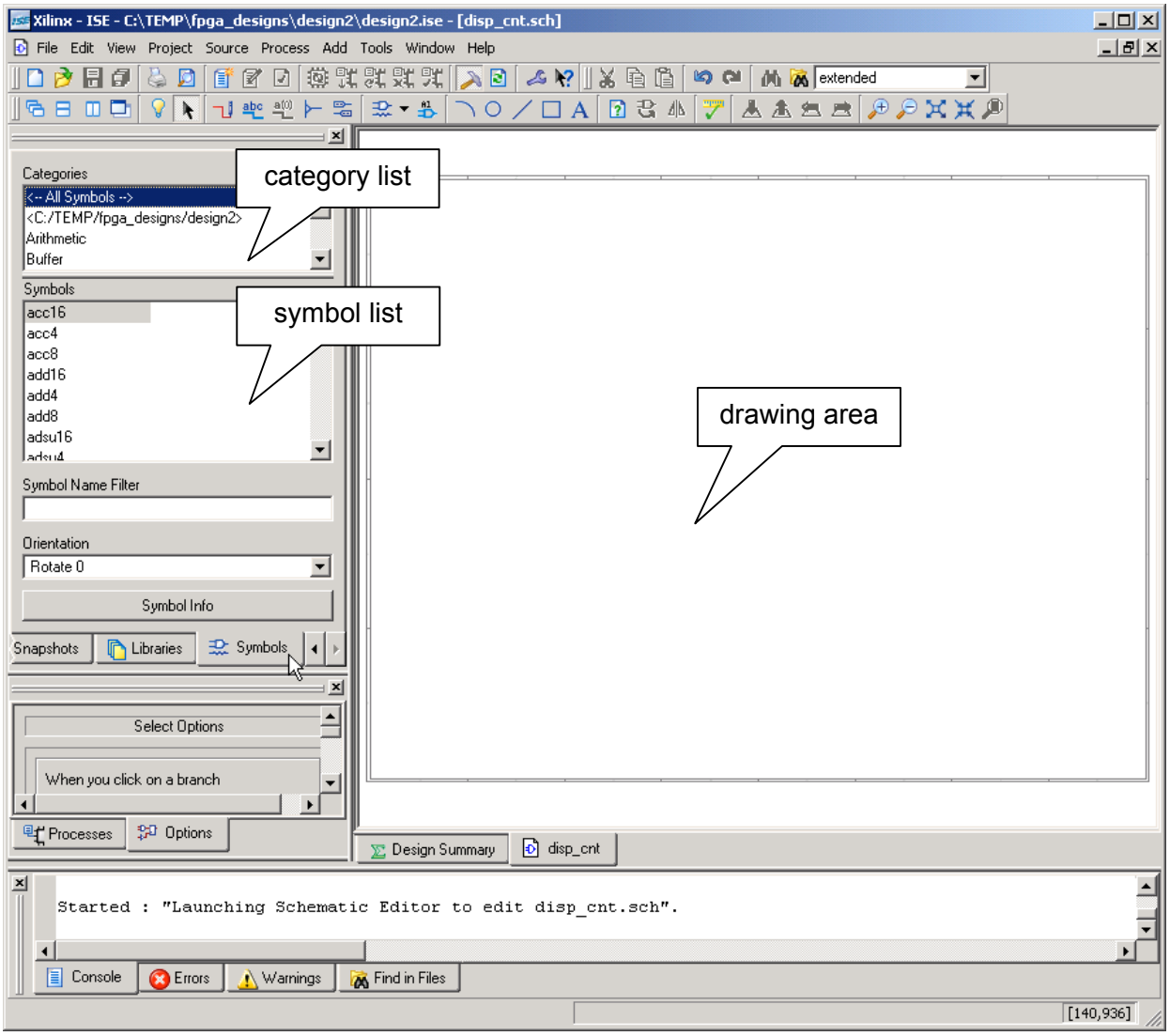
There is very little to do when initializing a schematic, so just click on the Finish button in the **Summary** window that appears.



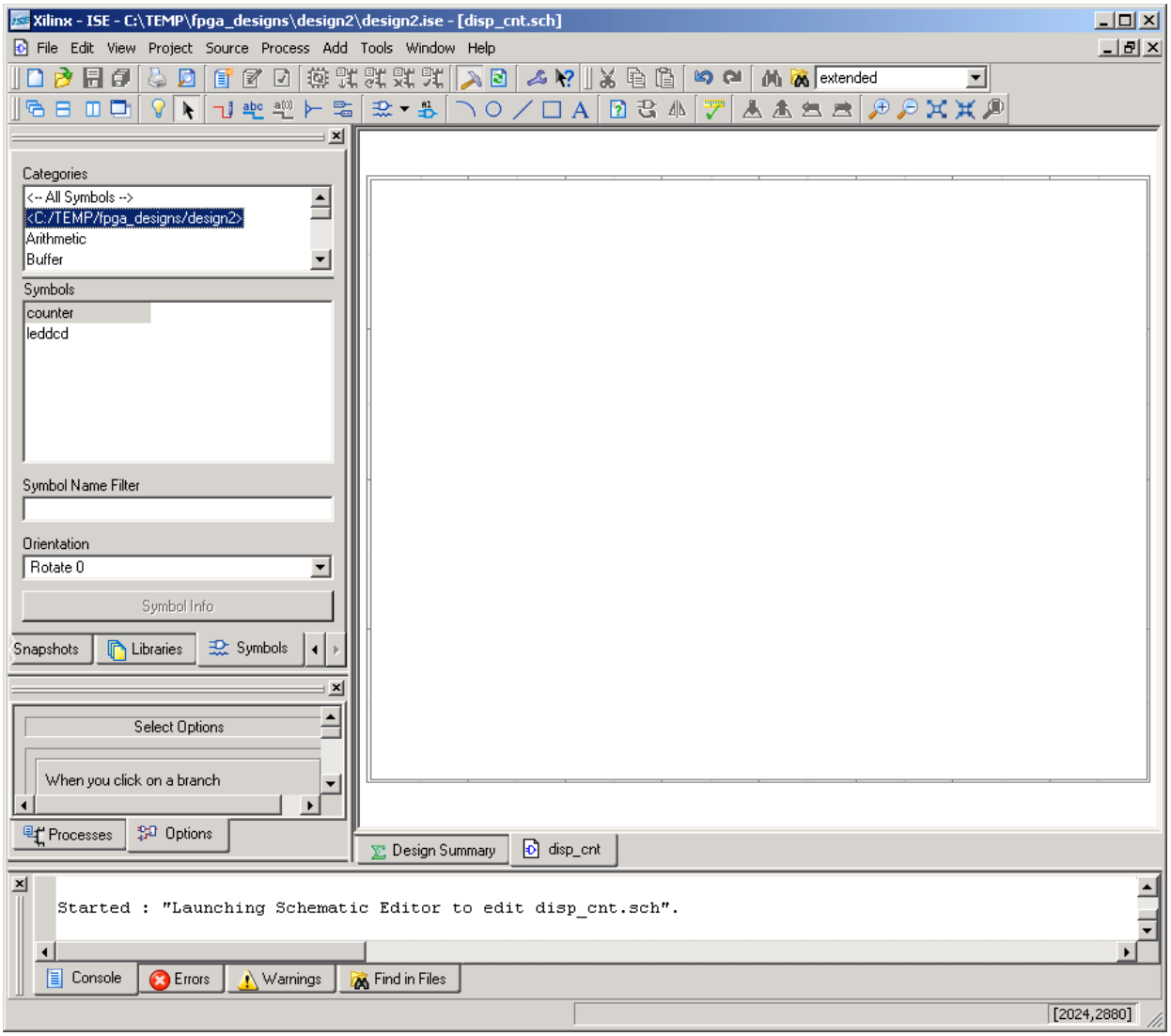
The disp_cnt schematic object has now been added to the Sources pane. You can double-click it to begin creating the schematic, but a schematic editor window should open automatically once the file is created.



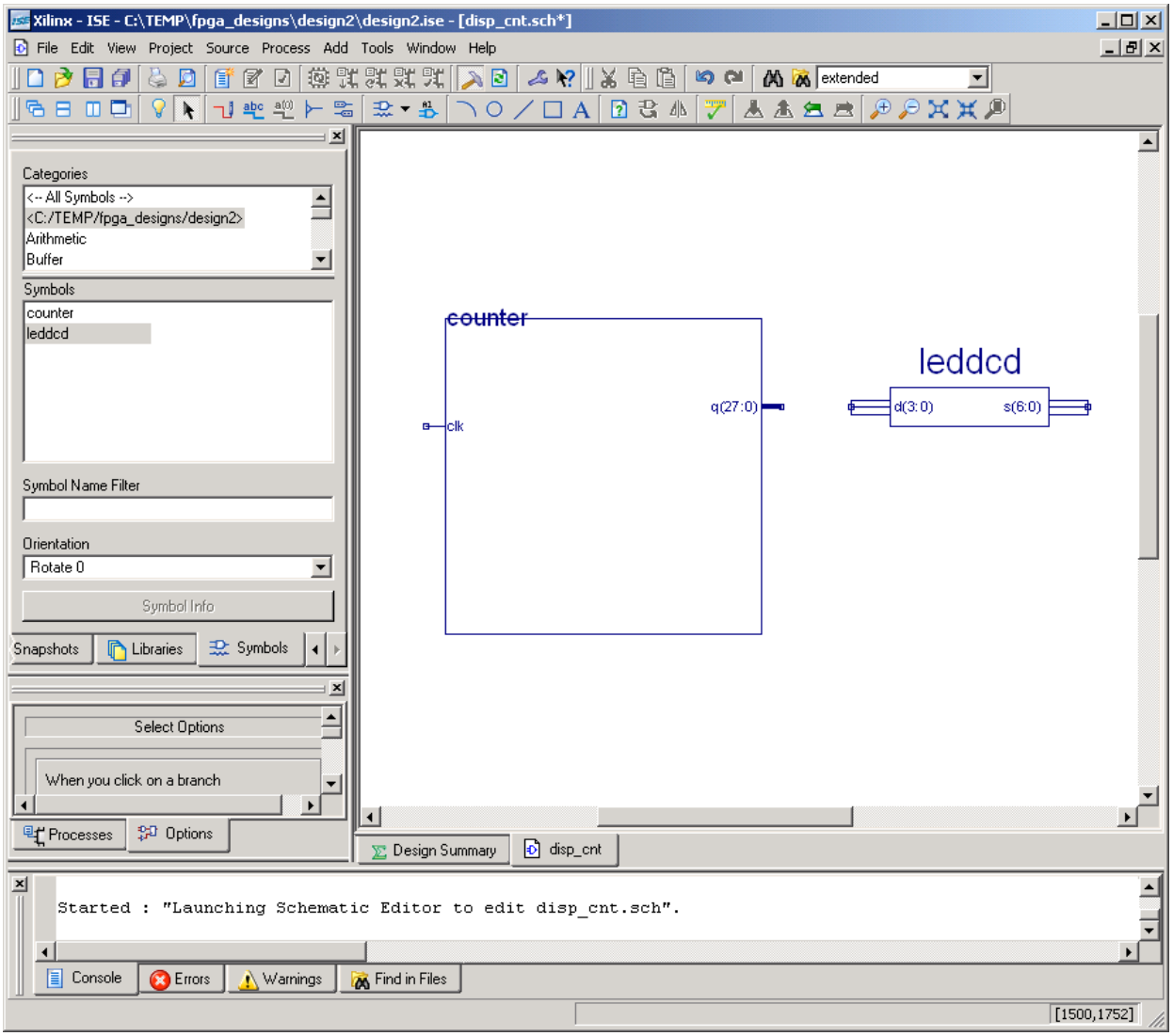
The schematic editor window has a drawing area. Click on the Symbols tab at the bottom of the Sources pane. The Symbols tab contains a list of categories for various logic circuit elements that can be used in a schematic. Below that is the list of circuit element symbols in the highlighted category.



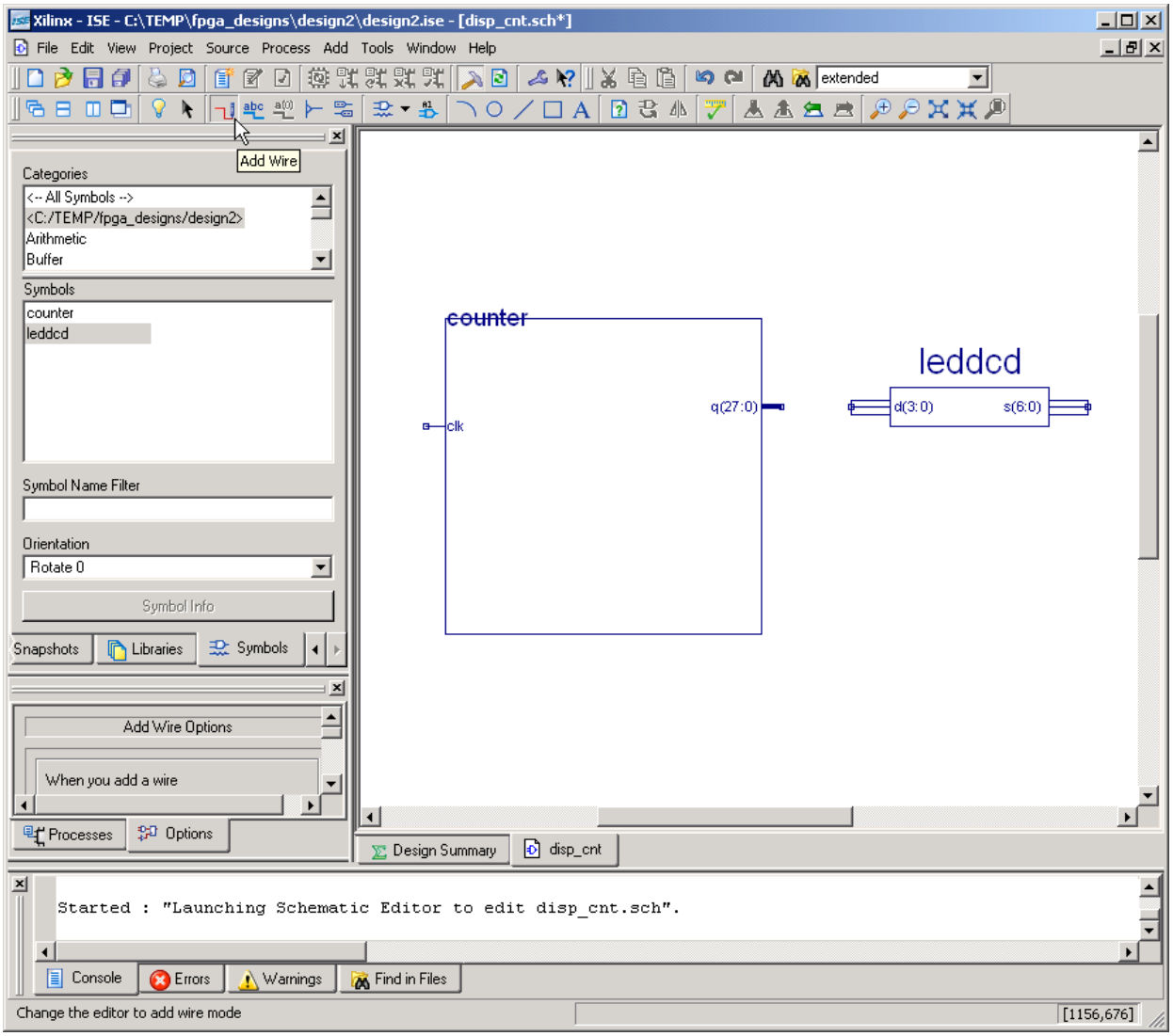
To start creating the top-level schematic, highlight the second entry in the category list. The c:/TEMP/fpga_designs/design2 category contains the schematic symbols for the ***design2*** project's counter and LED decoder modules. You can see the names of these modules in the symbol list.



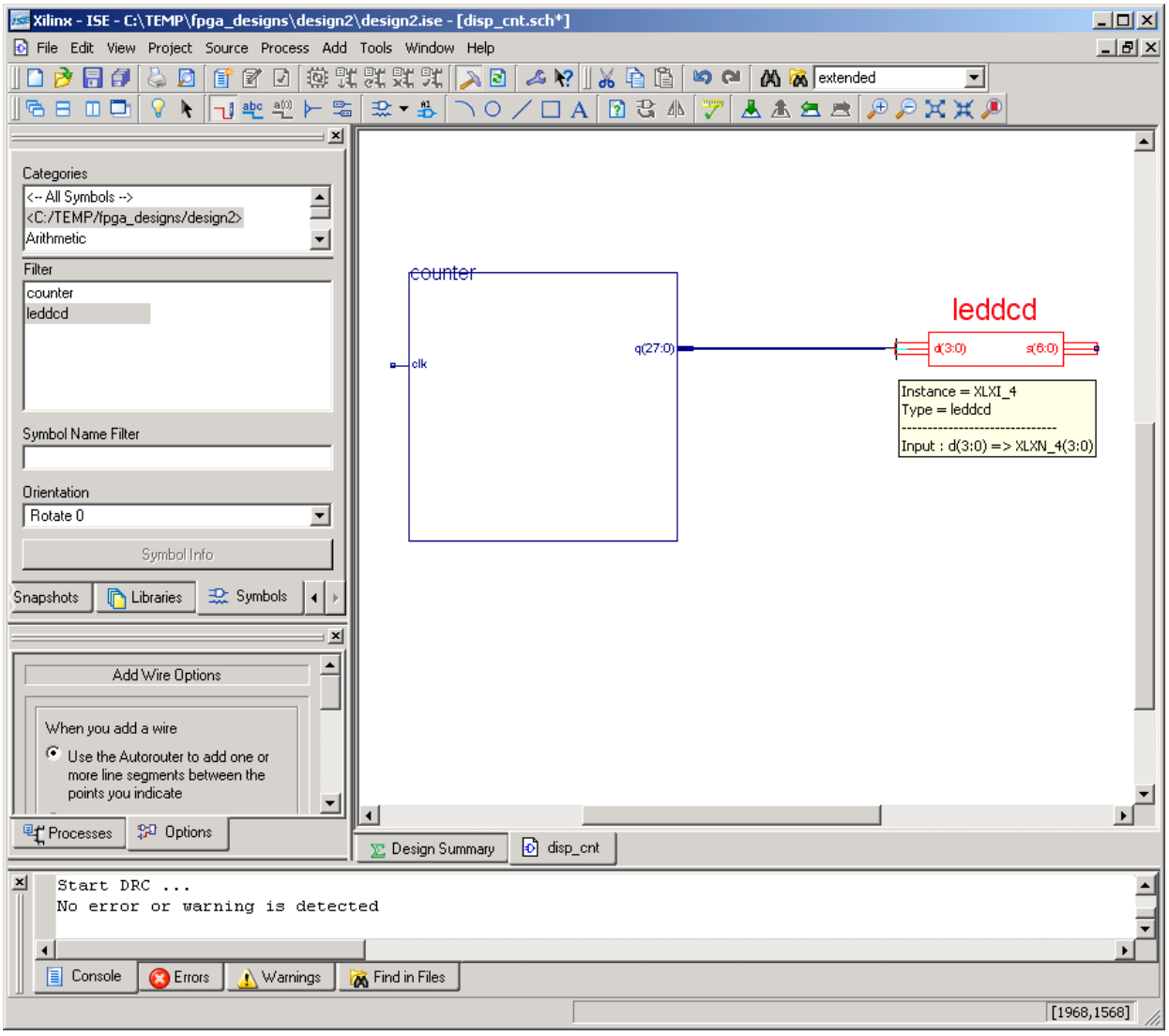
Click on the counter entry in the Symbols list. Then move the mouse cursor into the drawing area and left-click to place an instance of the counter into the schematic. Repeat this process with the leddcd module to arrive at the arrangement shown below.



Next, click on the  button to begin adding wires to the schematic.



Left-click the mouse on the **q(27:0)** bus on the right-hand edge of the **counter** module. Then left-click on the **d(3:0)** bus on the left-hand edge of the **leddcd** module. This creates a four-bit bus between the output of the counter module and the input of the LED decoder module.



However, the four-bit bus causes a problem. Click on the  button to run a design-rule check. An error is reported in the Console tab:

```
Error:DesignEntry - disp_cnt.sch: Pin 'q(27:0)' is connected to a bus
of a different width.
```

The error is caused by the mismatch between the 28-bit counter output and the four-bit bus. You can fix this problem through the following steps:

1. Click on the selection tool button:  .
2. Right-click on the existing bus and select Delete from the pop-up menu to remove this bus.

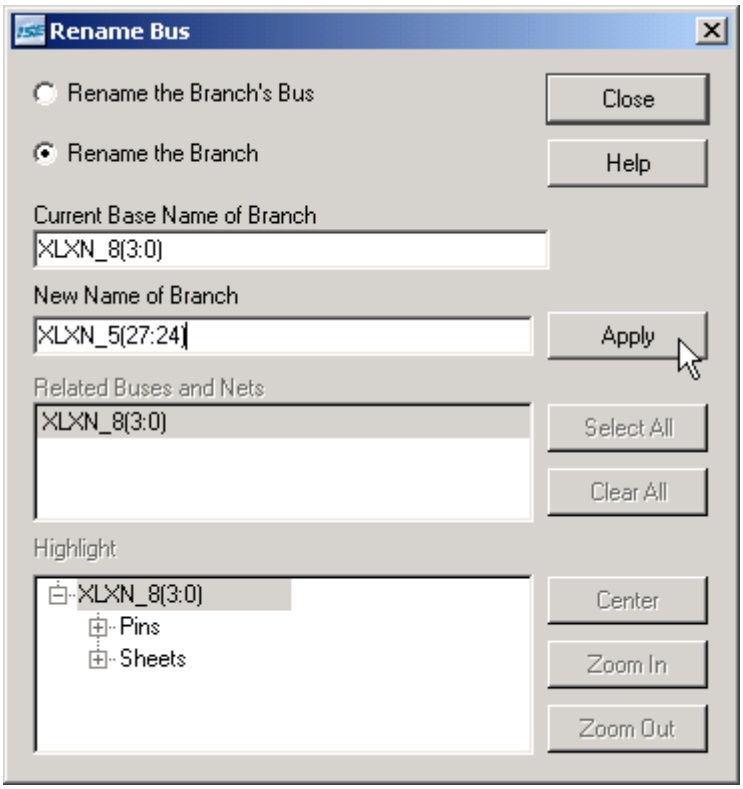
3. Click on the wiring tool,  , click on the counter output, and draw a small bus stub outward. Terminate the bus with a double-click. This will create a 28-bit bus connected to the counter outputs.

4. Repeat the previous step to attach a four-bit bus to the LED decoder input.

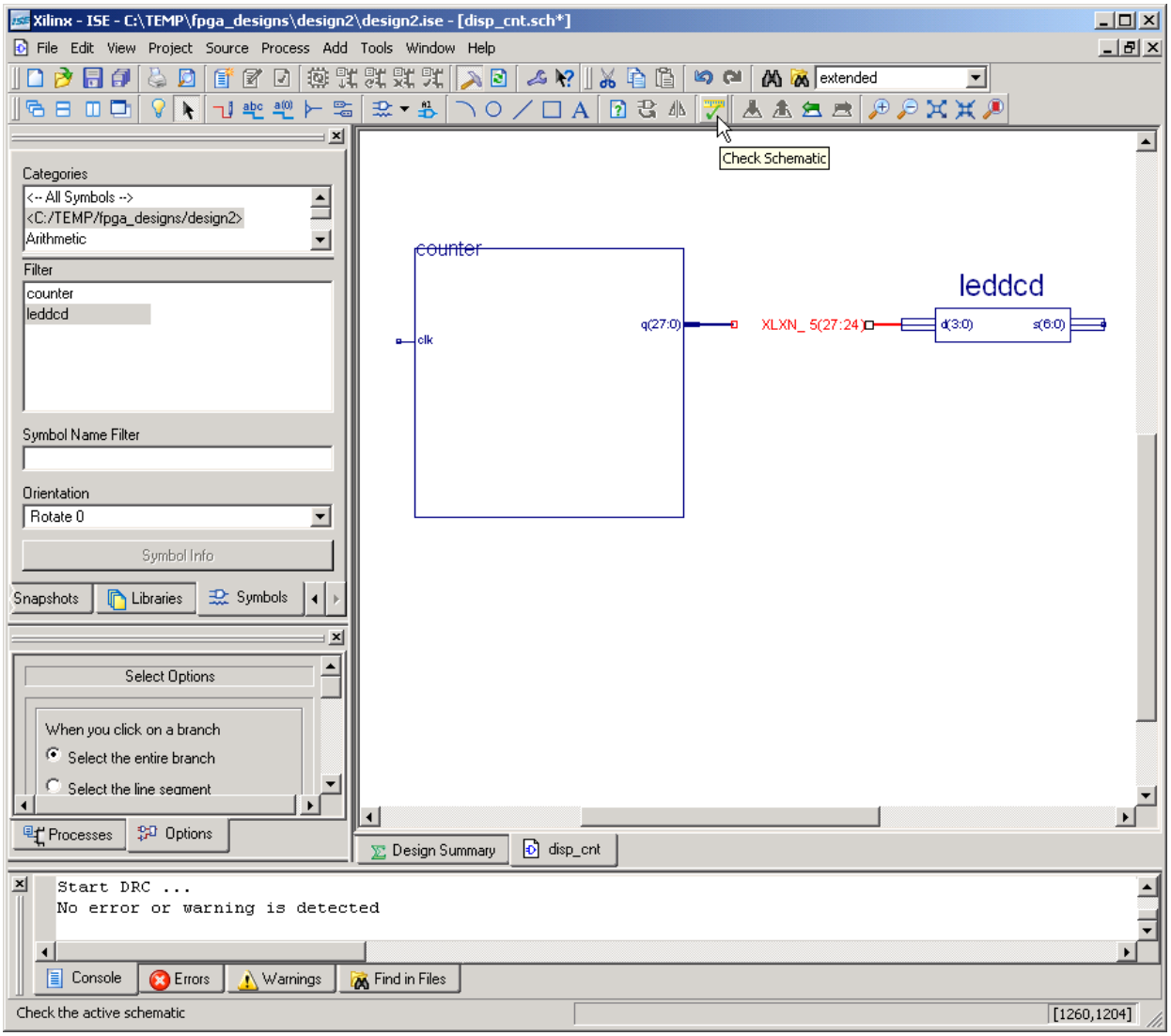
5. Click on the selection tool,  , and then hover the mouse pointer over the bus connected to the counter. The name of the bus will be shown. In my project, the bus is named **XLXN_5(27:0)**.

6. Right-click on the bus connected to the LED decoder and select Rename Selected Bus... in the pop-up menu.

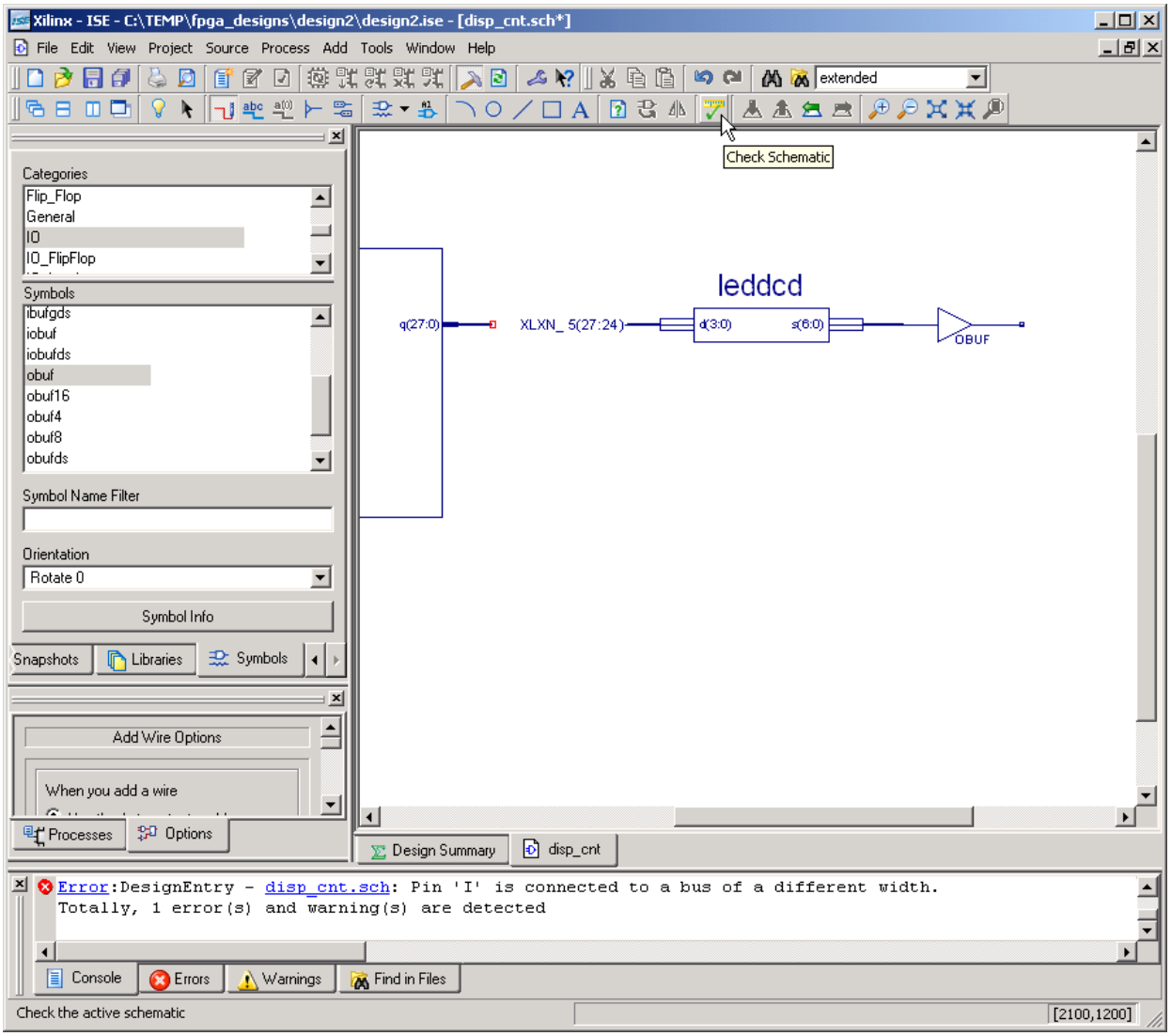
7. Rename the LED decoder input bus from **XLXN_8(3:0)** to **XLXN_5(27:24)** as shown below and click on the Apply button. This will connect the LED decoder inputs to the four most-significant bits of the counter output bus.



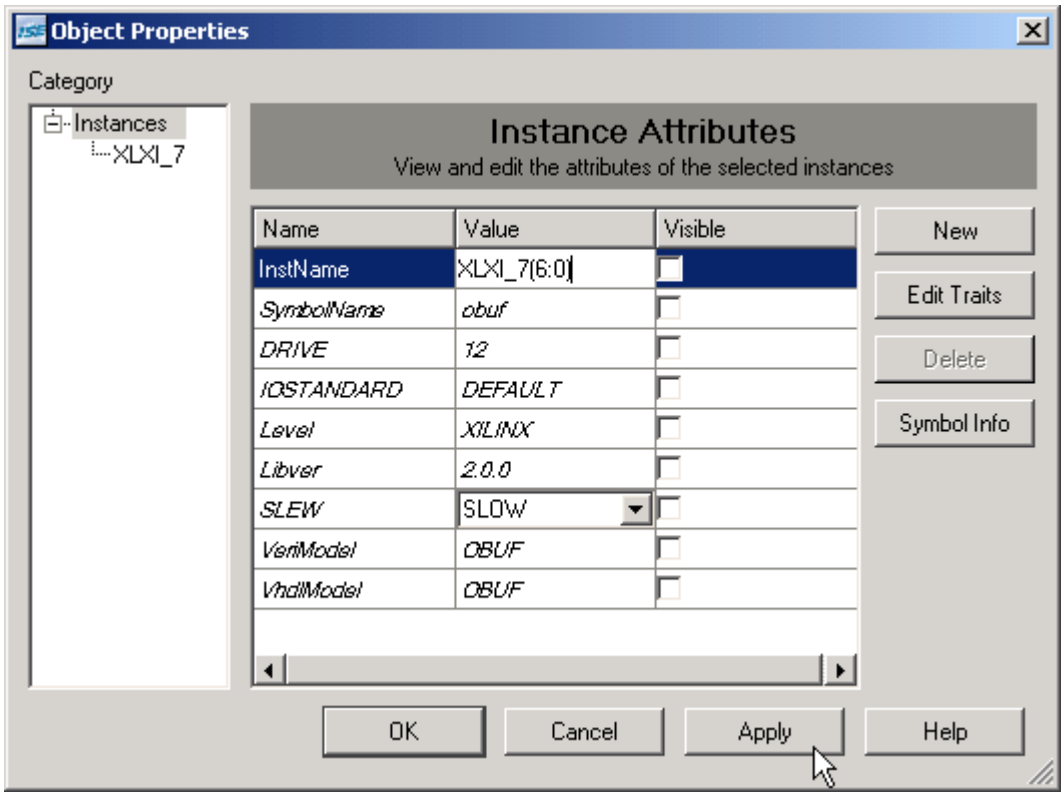
At this point, the schematic should appear as shown below. The design-rule checker should no longer detect any problems.



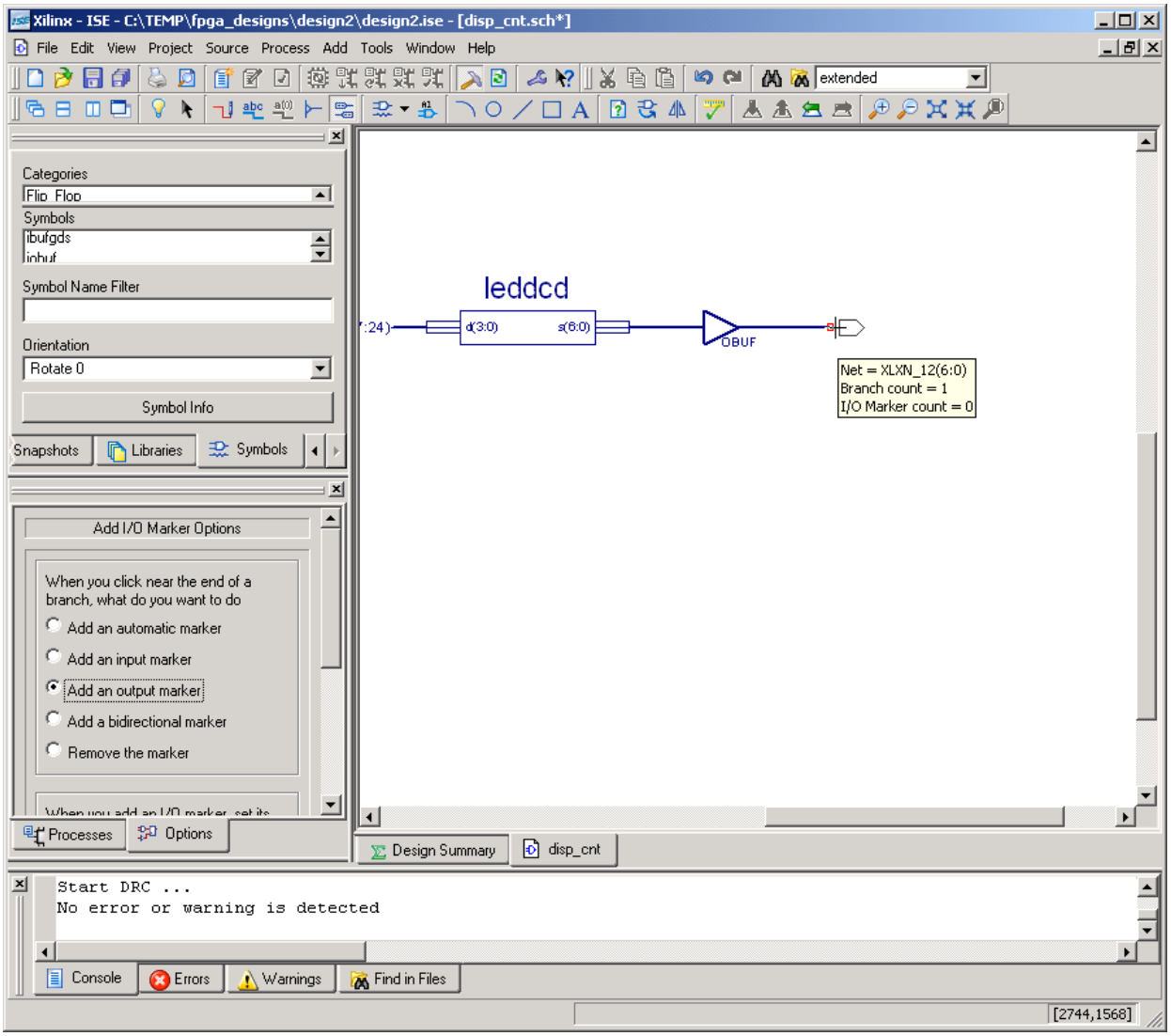
Now highlight the IO category and select a single-bit output buffer (obuf) from the list of symbols. Attach the output buffer to the output of the LED decoder as shown below.



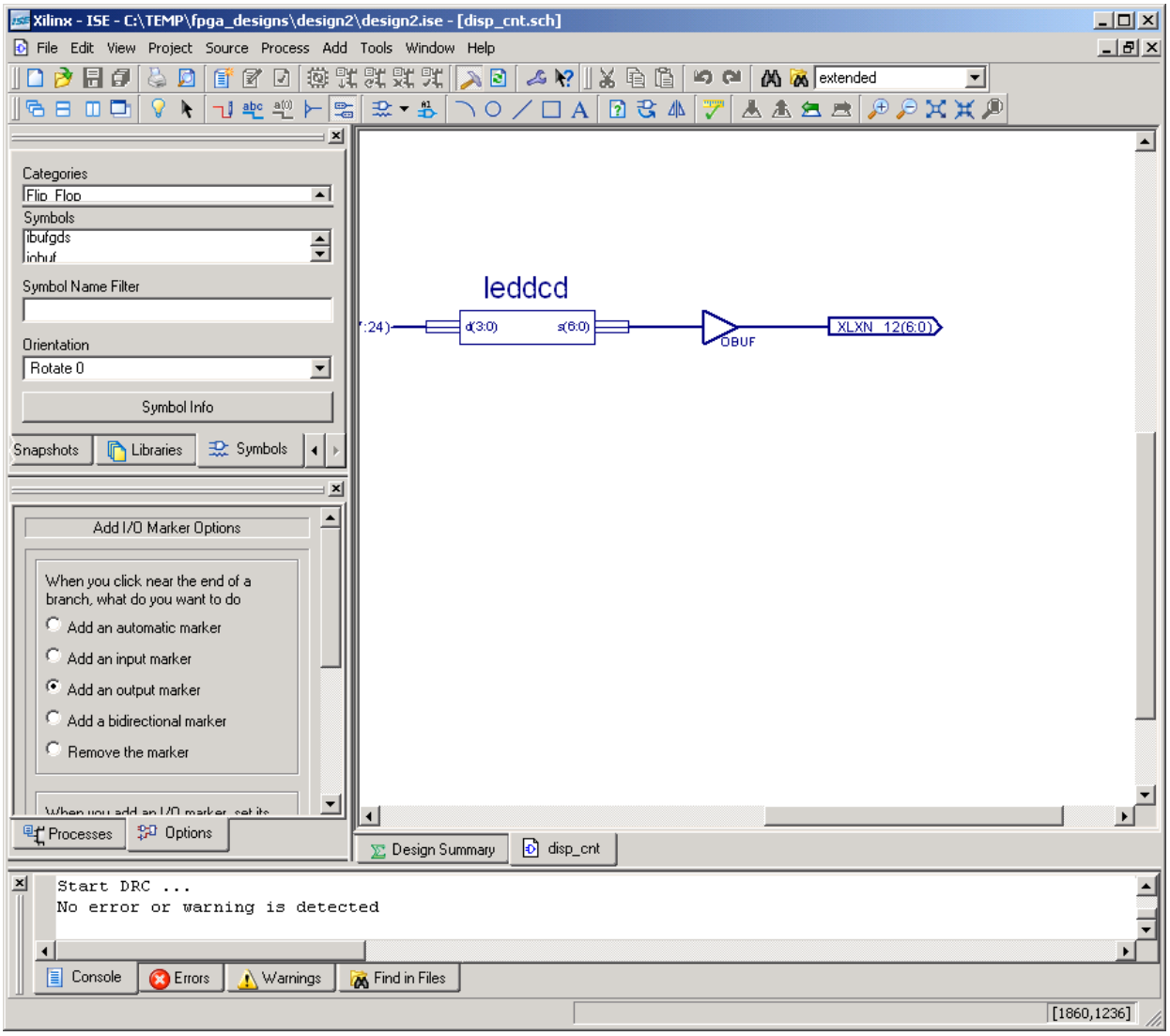
The design-rule checker will detect a width-mismatch error in the above schematic since a single output buffer has been attached to the seven-bit LED decoder output bus. This error can be fixed by right-clicking on the OBUF symbol and selecting Object Properties from the pop-up menu. Then append `(6:0)` to the instance name of the output buffer as shown below and click on Apply. This will expand the single buffer into an array of seven buffers with each one connected to a bit of the LED decoder output bus.



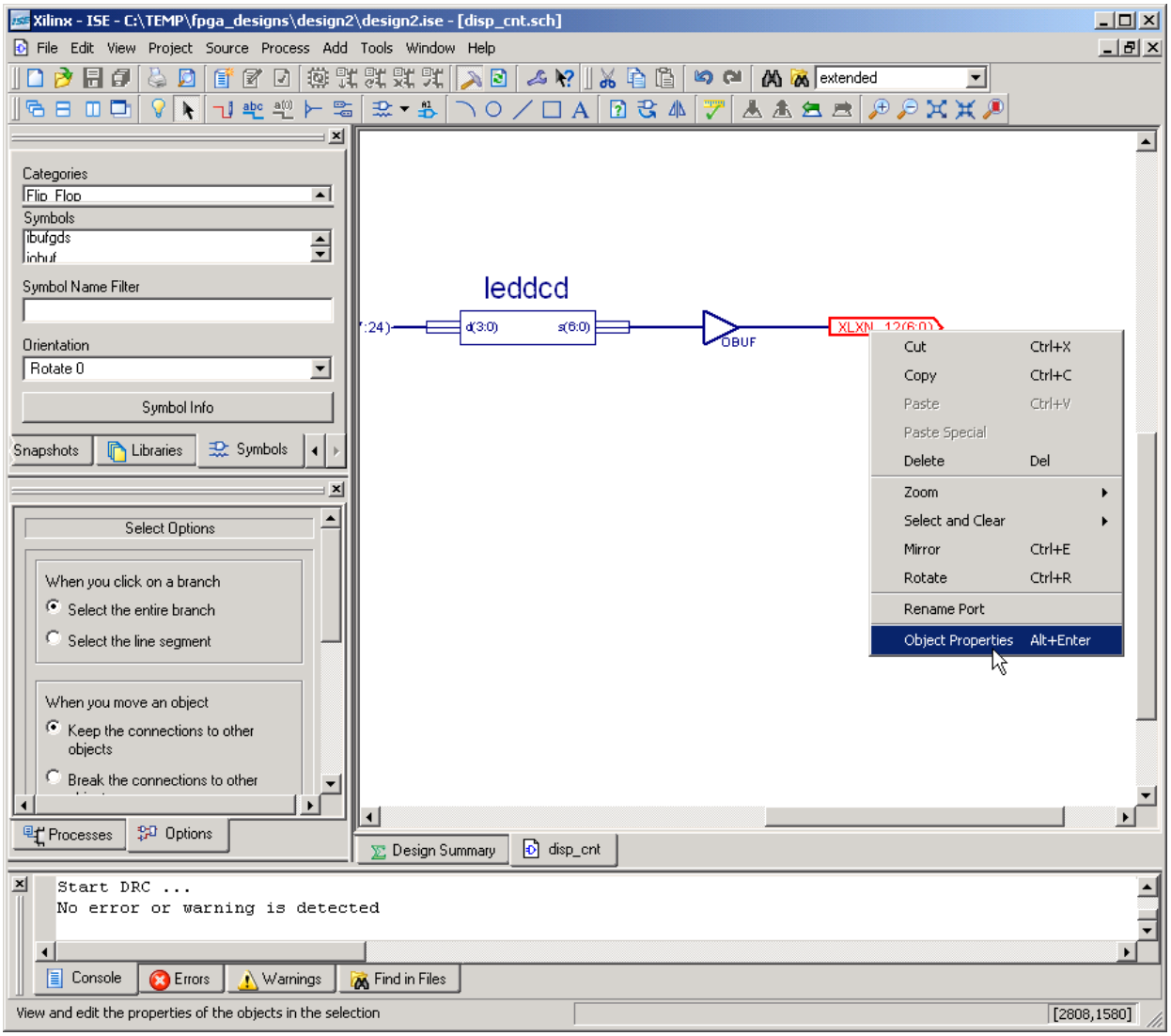
Next, attach a short bus segment to the output of the OBUF. Then click on the  button for adding I/O markers. Click on the Add an output marker button in the Options tab and then click on the free end of the wire segment that you just added.



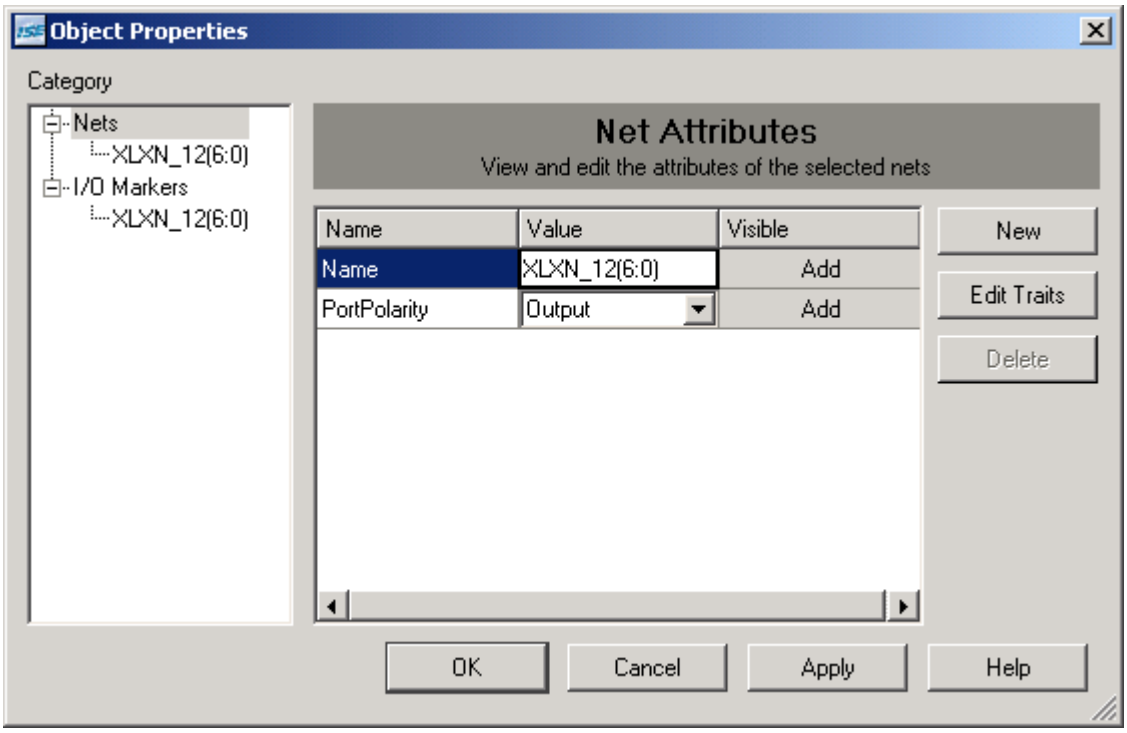
Clicking on the end of the wire creates a seven bit-wide set of output pins.



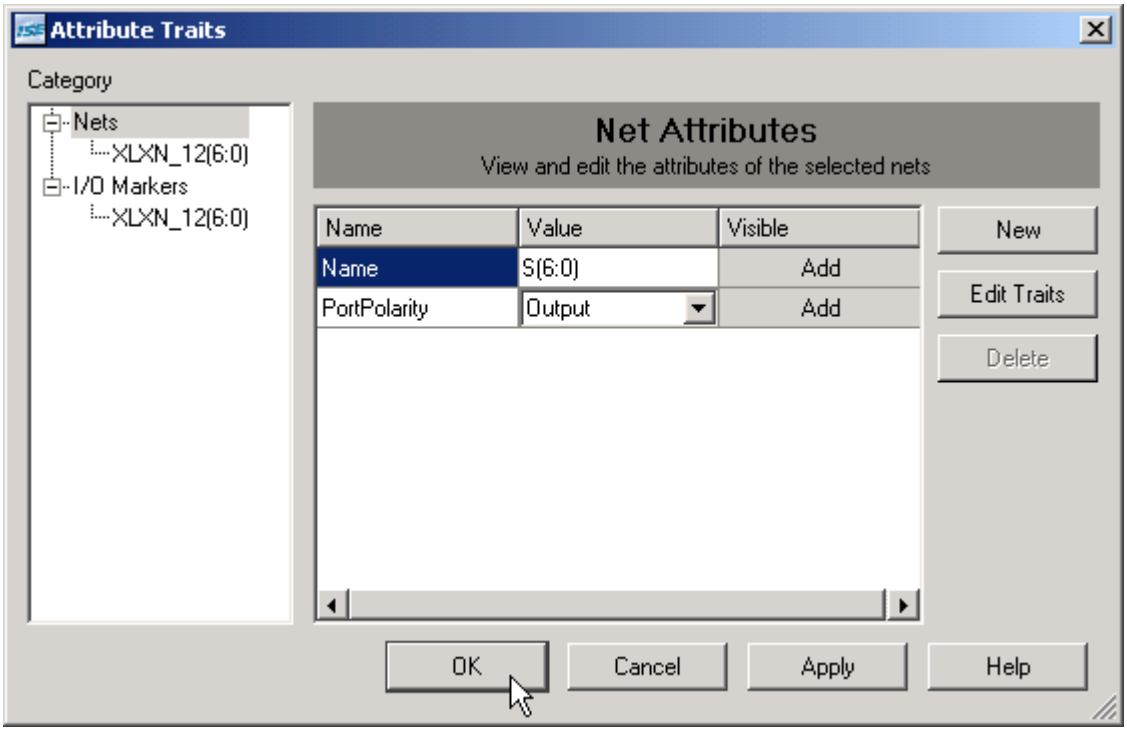
The output pins automatically assume the same name as the bus to which they are attached, but this name was automatically generated and doesn't carry a lot of meaning. To change the name of the outputs (and the associated bus), right-click on the I/O marker and select Object Properties... from the pop-up menu.



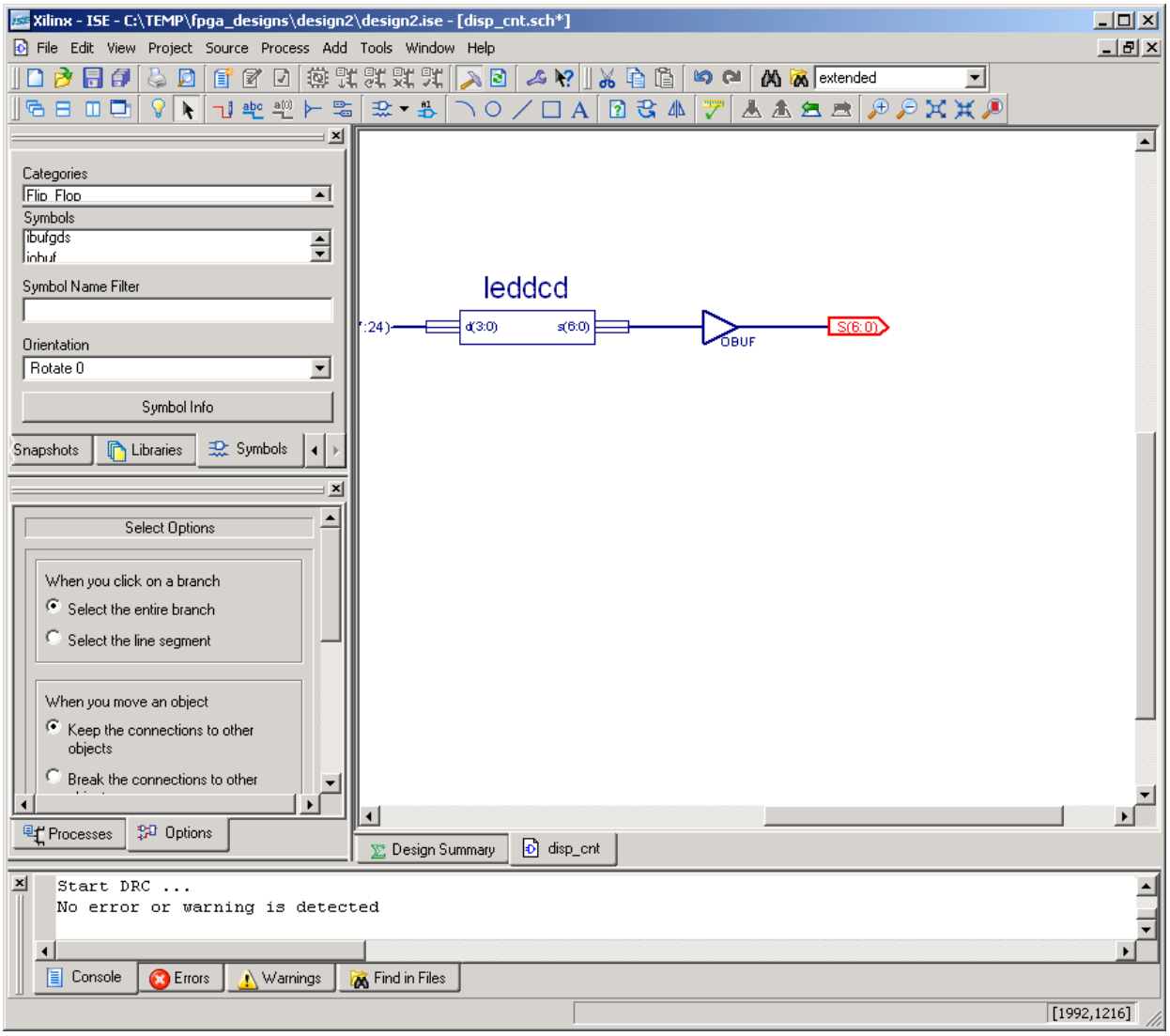
The **Object Properties** window allows you to set the name and direction of the pins.



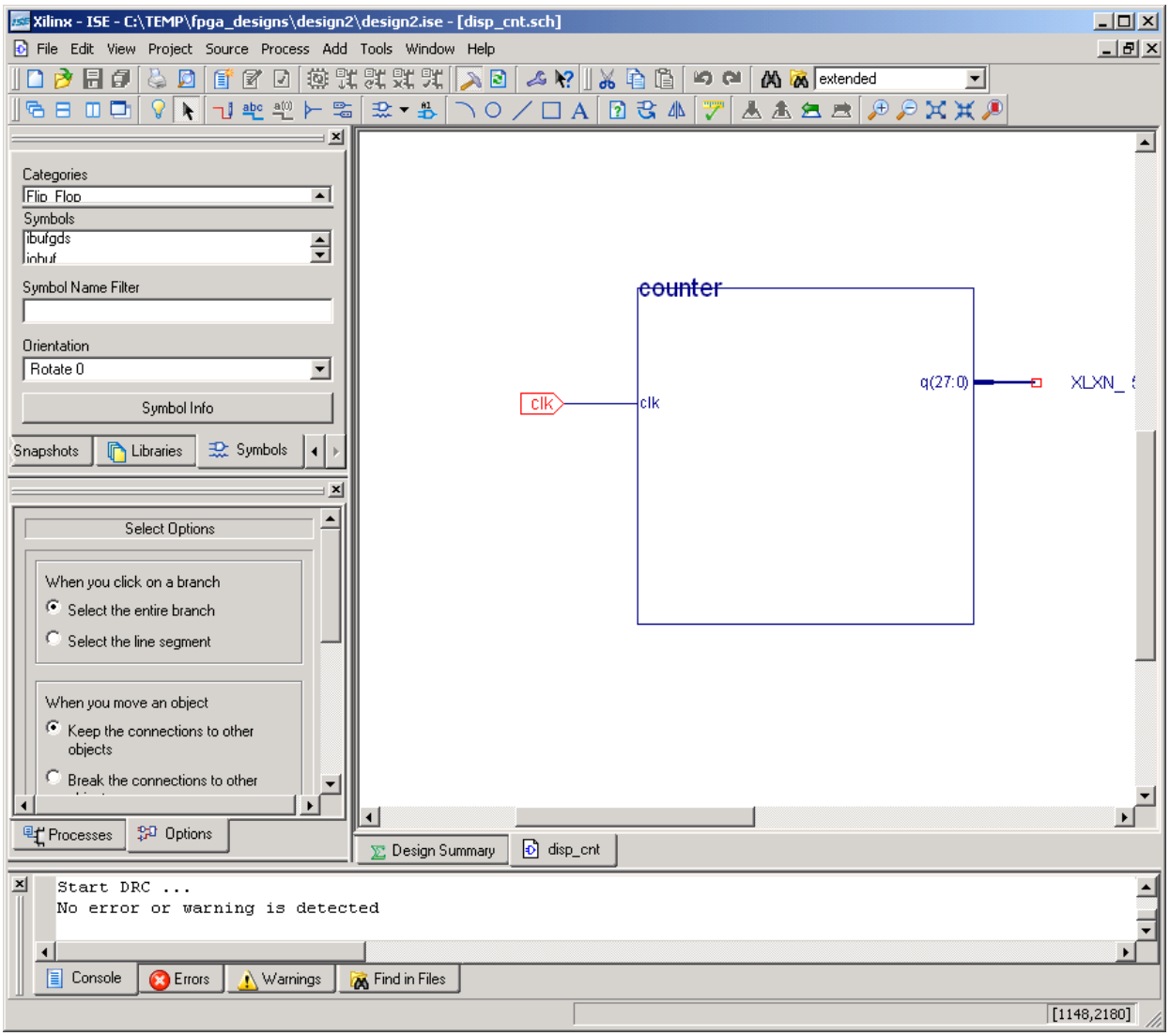
Replace the existing bus name with a seven-bit bus for driving the LED segments: **S(6:0)**. The direction of the bus pins is already set to Output so you can finish by clicking on the OK button.



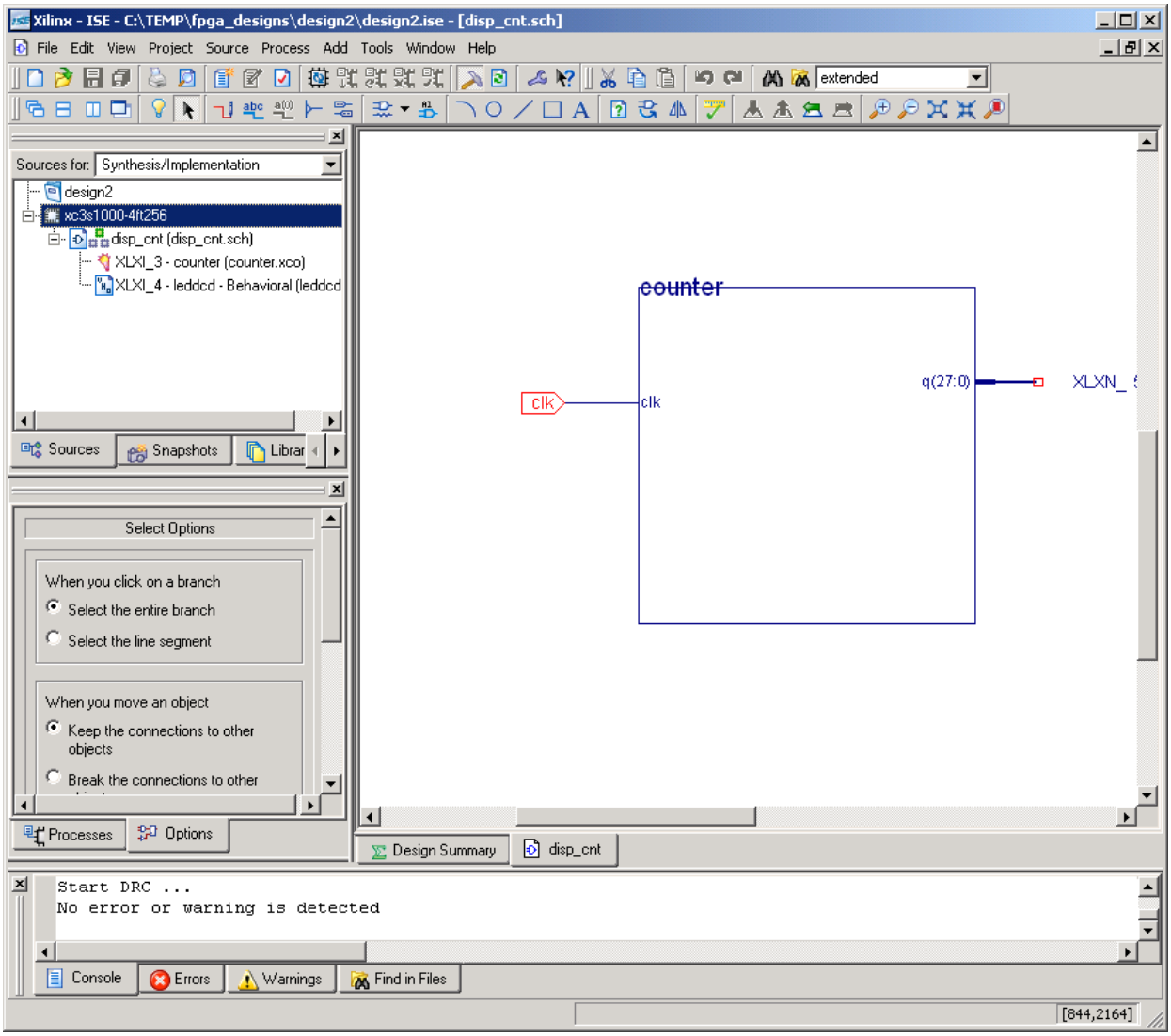
The output pins now appear with their new name.



Once the outputs from the circuit are in place, you can connect a single input I/O marker to the clock input of the counter. (No input buffer is needed because the clock signal will enter the FPGA through a dedicated global clock input.) Right-click on the I/O marker and rename it to `clk`. After this, perform another schematic check to detect any errors and save the schematic using the File➔Save command.

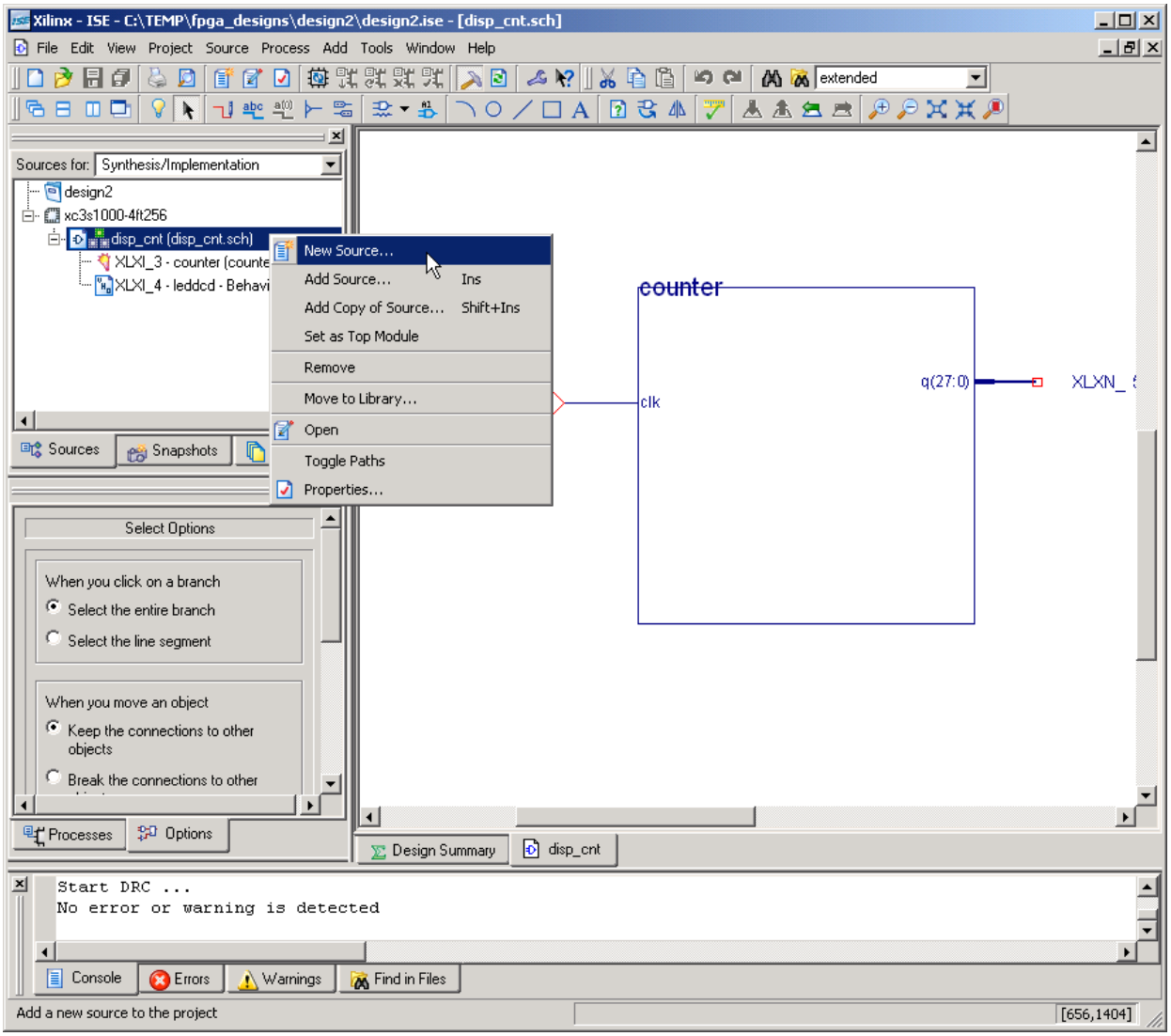


Once you save the schematic for the top-level module, the hierarchy in the Sources pane gets updated. Now the **counter** and **leddcd** modules are shown as lower-level modules that are included within the top-level **disp_cnt** module.

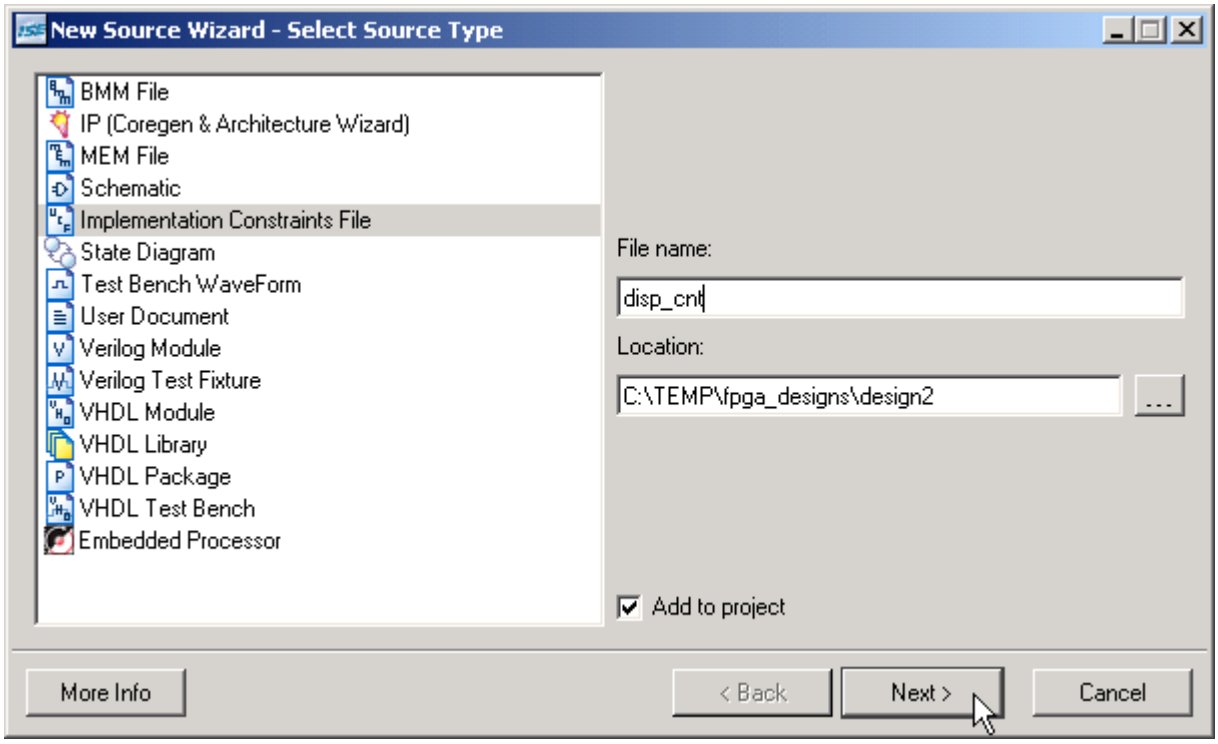


Constraining the Design

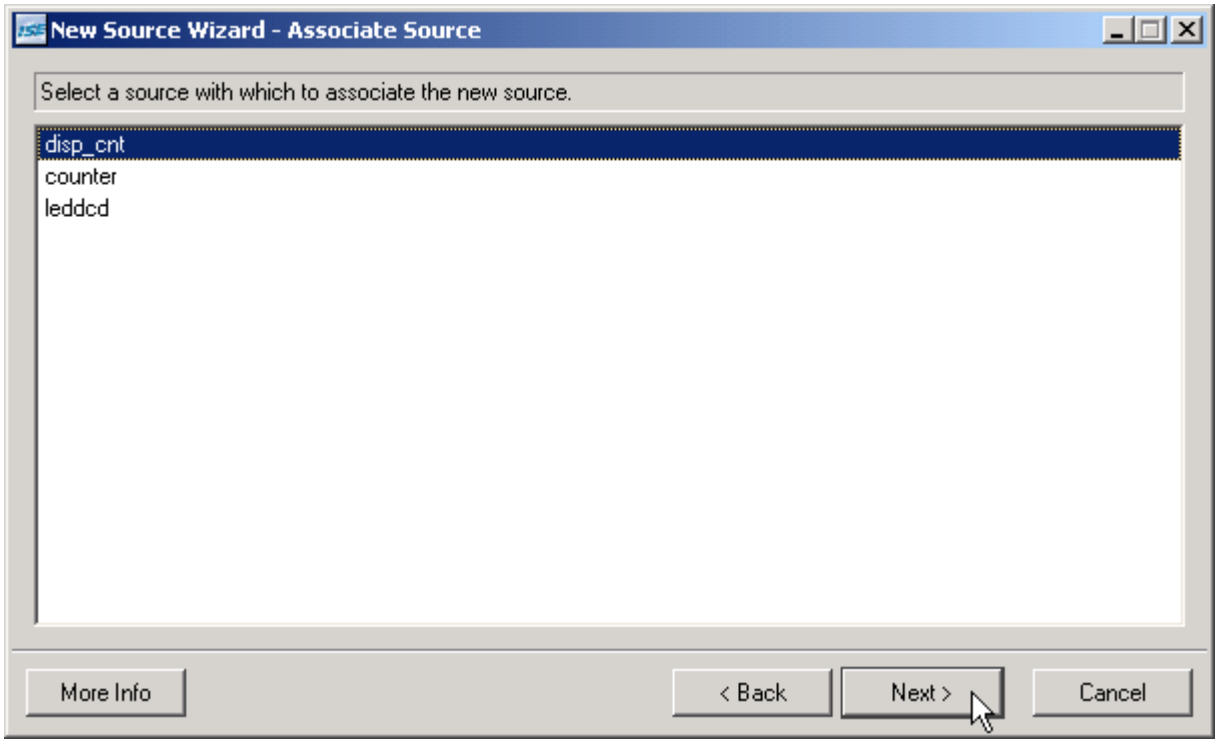
Before synthesizing the displayable counter, you need to assign pins to the inputs and outputs. Start by right-clicking the disp_cnt object in the Sources pane and selecting New Source... from the pop-up menu.



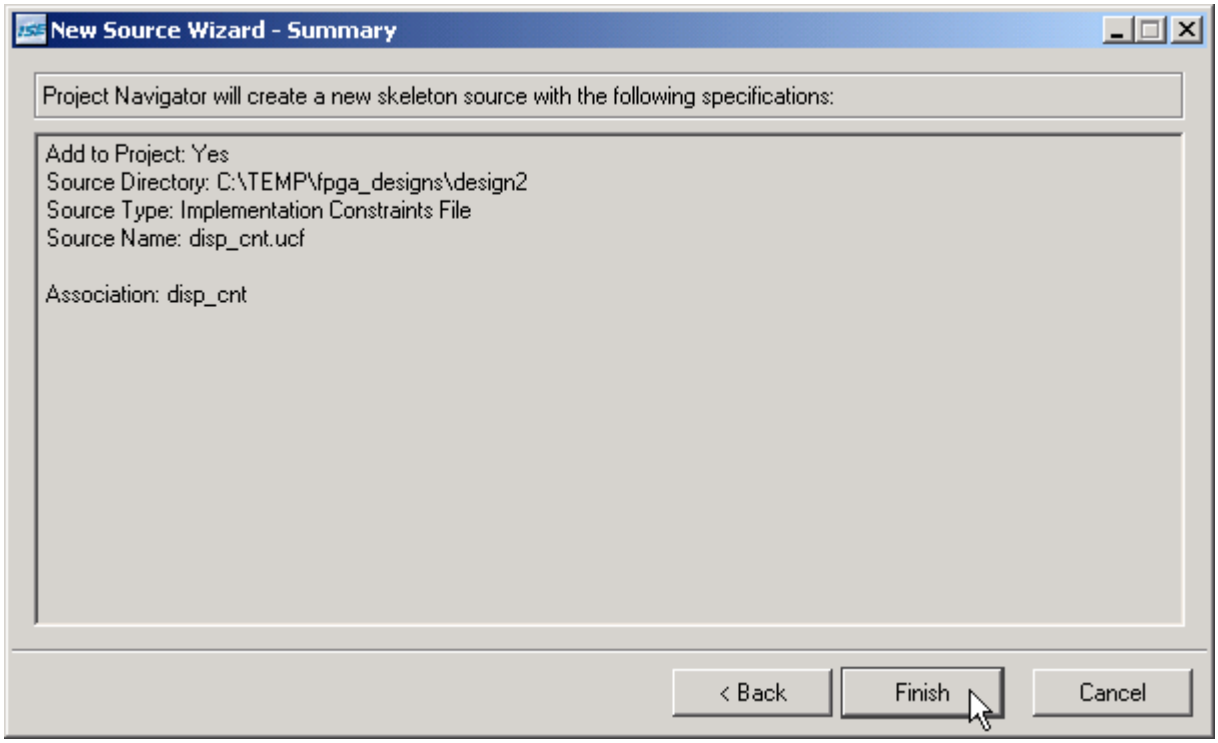
Select Implementation Constraints File as the type of source file to add and type `disp_cnt` in the File Name field. Then click on the Next button.



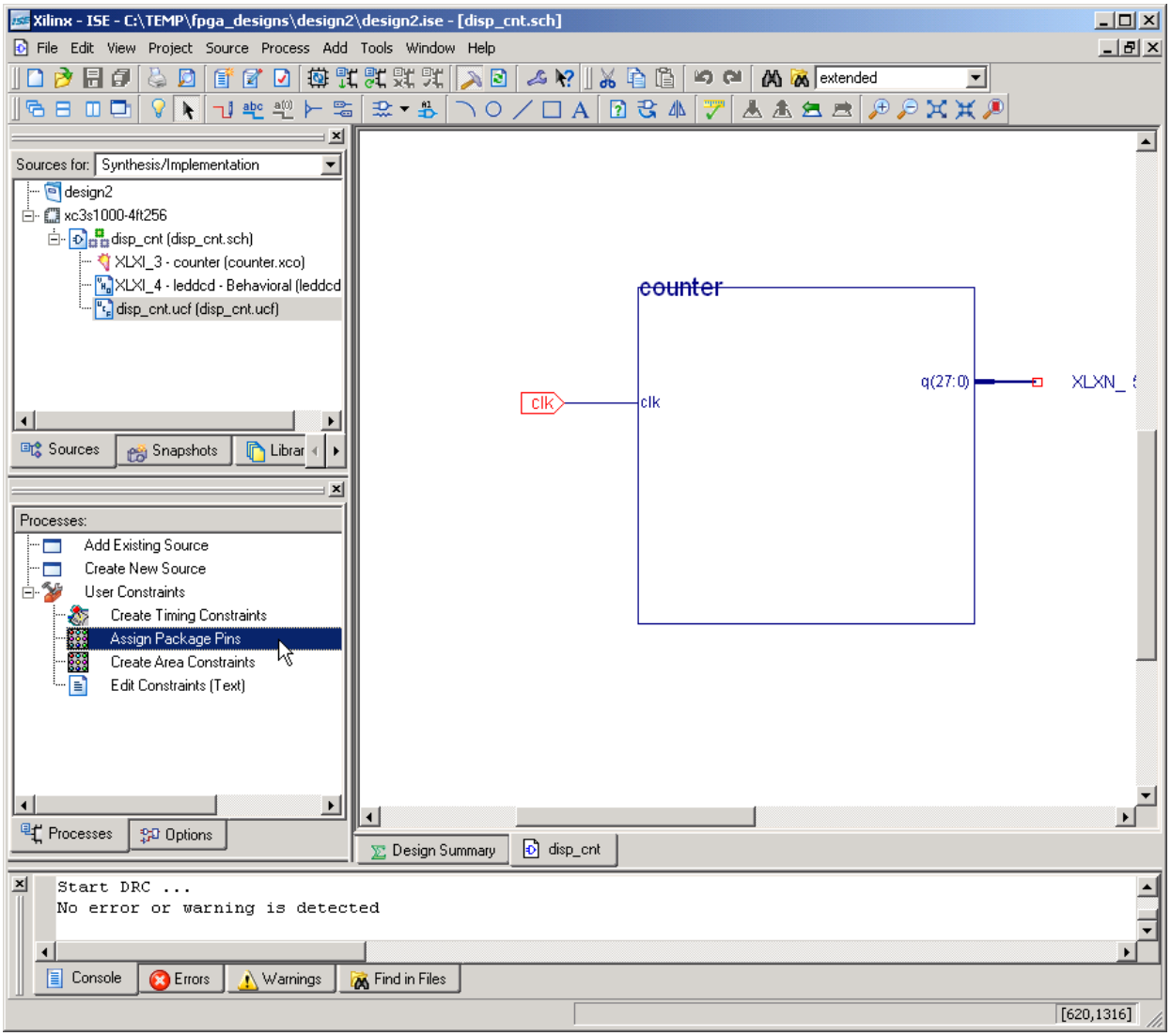
Then you are asked to pick the file the constraints will apply to. The pin assignments will be made for the top-level module in the design hierarchy, so highlight the disp_cnt item in the list. Then click on the Next button and proceed.



You will receive a feedback window that shows the name and type of the file you created and the file to which it is associated. Click on the Finish button to complete the addition of the disp_cnt.ucf file to this project.



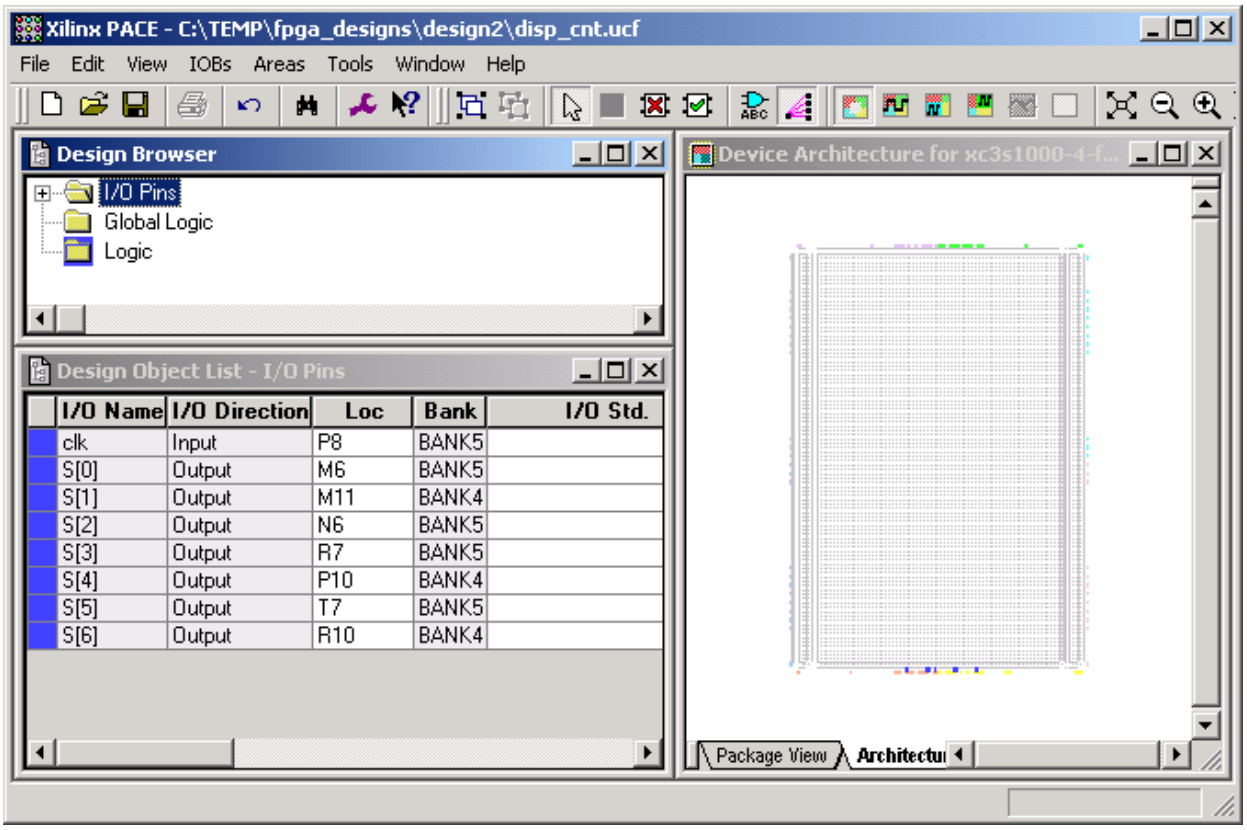
Now highlight the disp_cnt.ucf object that was added to the Sources pane and double-click the Assign Package Pins process to begin adding pin assignment constraints to the design.



The appropriate pin assignments for each model of XSA Board are shown below. The **clk** input is assigned to a dedicated clock input on each FPGA to which a 50 MHz clock signal is applied. The seven-segment LED pin assignments are the same as in the previous design.

I/O Signal	XSA-50	XSA-100	XSA-200	XSA-3S1000
clk	P88	P88	B8	P8
s0	P67	P67	N14	M6
s1	P39	P39	D14	M11
s2	P62	P62	N16	N6
s3	P60	P60	M16	R7
s4	P46	P46	F15	P10
s5	P57	P57	J16	T7
s6	P49	P49	G16	R10

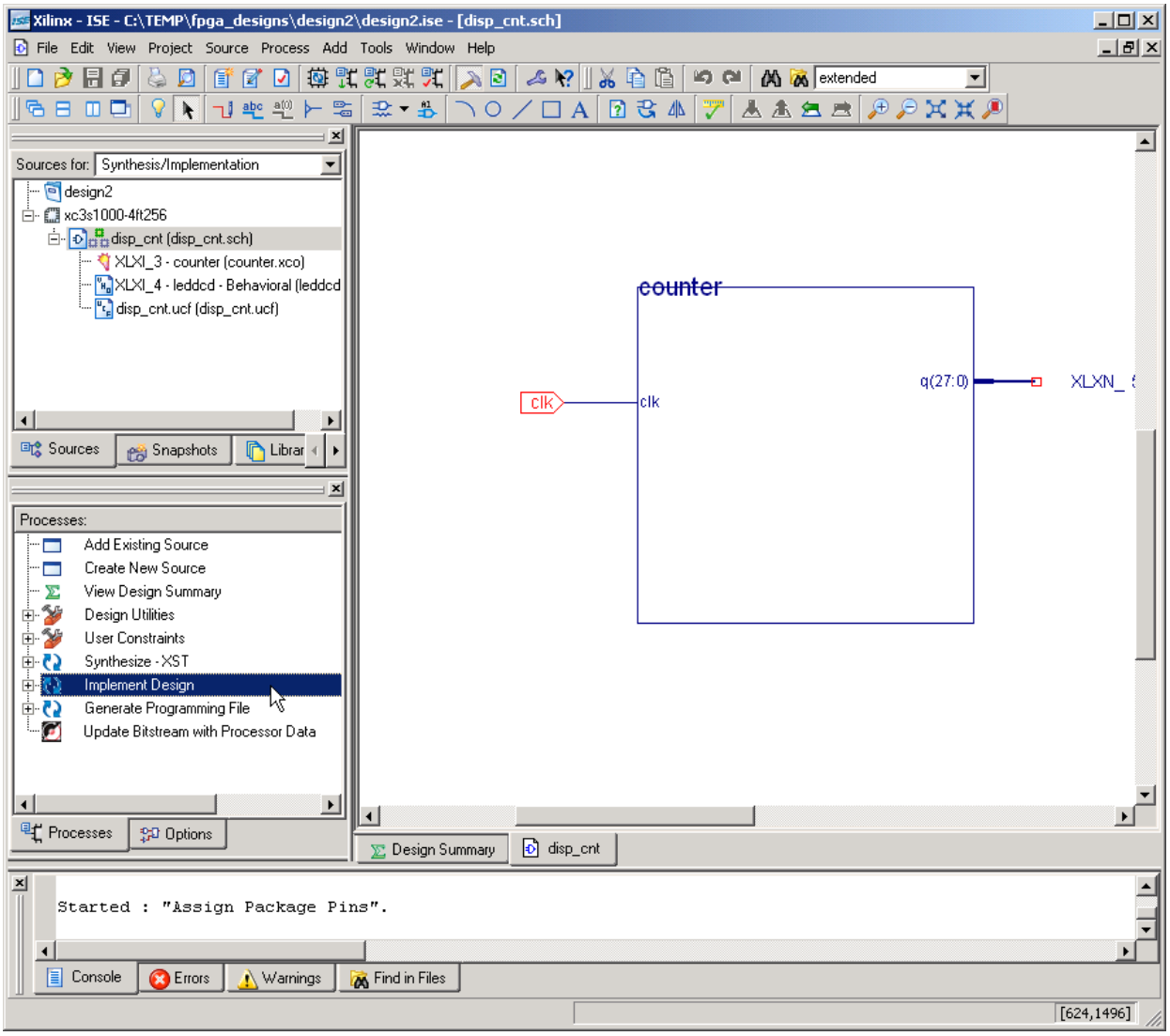
In the **Design Object List – I/O Pins** pane of the **Xilinx PACE** window that appears, set the pin assignments for the clock input and LED segment drivers as shown below. Then save the pin assignments and close the **PACE** window.



	I/O Name	I/O Direction	Loc	Bank	I/O Std.
	clk	Input	P8	BANK5	
	S[0]	Output	M6	BANK5	
	S[1]	Output	M11	BANK4	
	S[2]	Output	N6	BANK5	
	S[3]	Output	R7	BANK5	
	S[4]	Output	P10	BANK4	
	S[5]	Output	T7	BANK5	
	S[6]	Output	R10	BANK4	

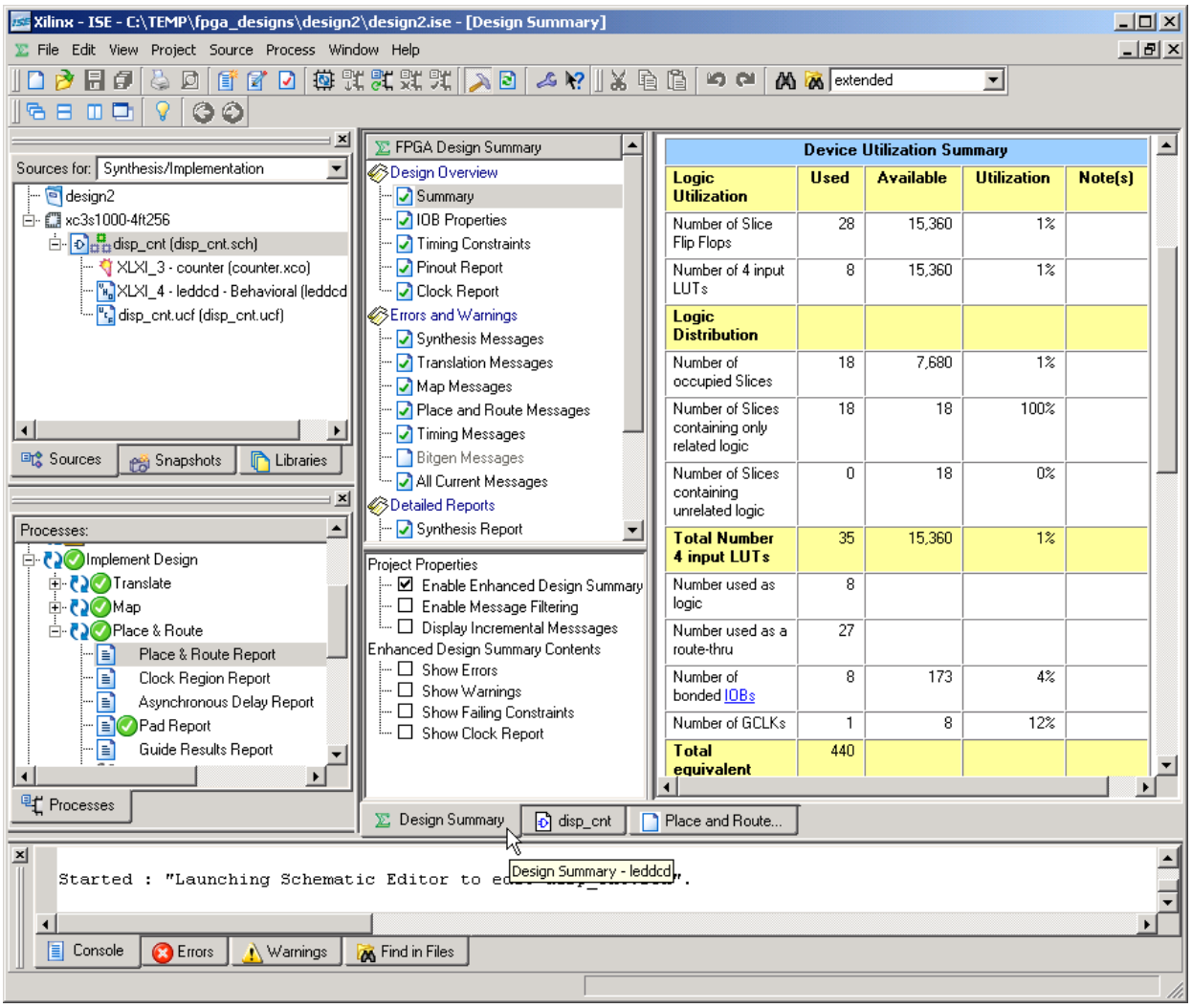
Synthesizing and Implementing the Design

Now you can synthesize the logic circuit netlist and translate, map and place & route it into the FPGA by highlighting the top-level disp_cnt module in the Sources pane and double-clicking the Implement Design process. The software will automatically invoke the synthesizer and then pass the synthesized netlist to the implementation tools. There should be no problems synthesizing and implementing the VHDL, Coregen and schematic files in the design.



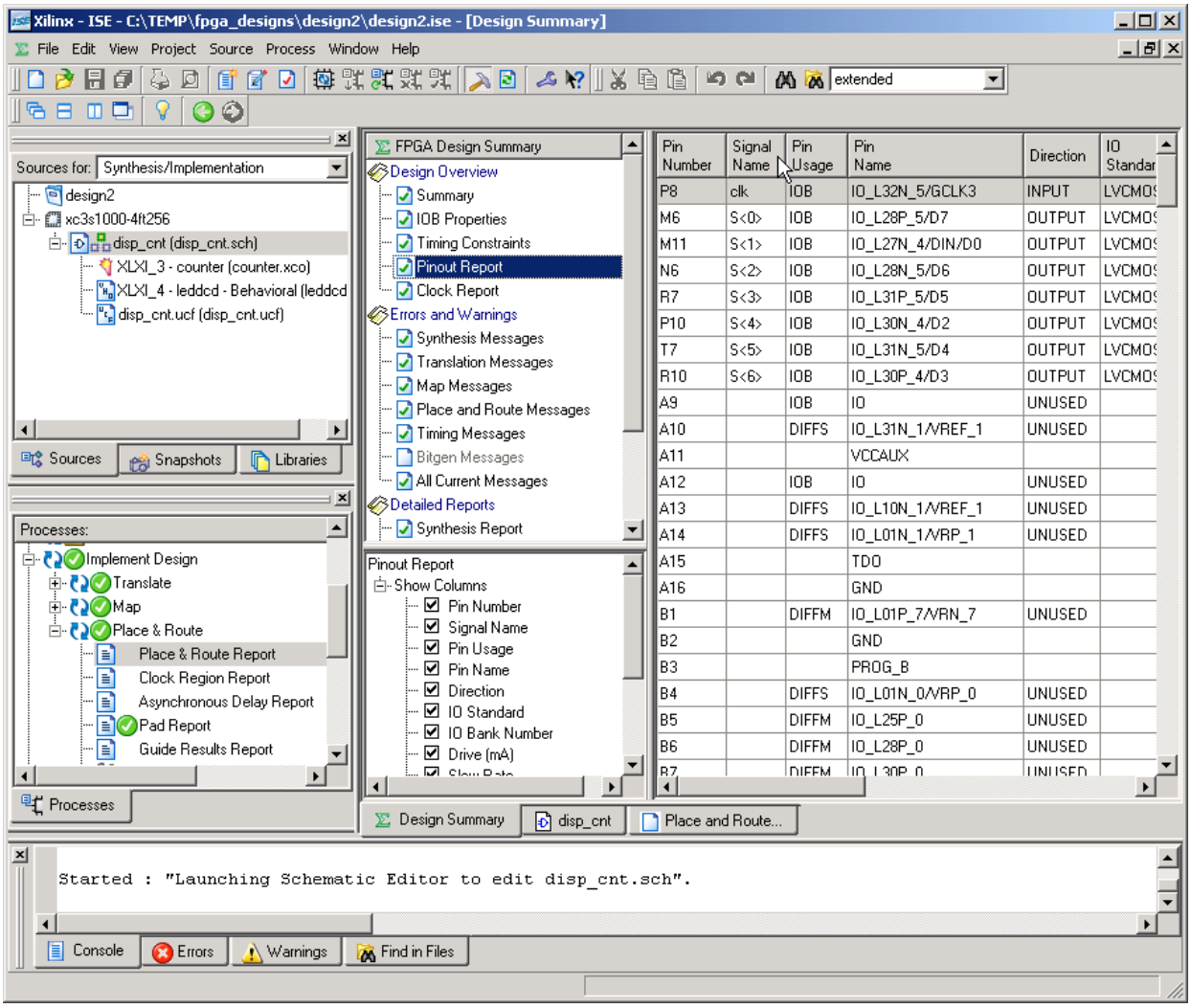
Checking the Implementation

After the implementation process is done, you can check the logic utilization by clicking on the Design Summary tab (or by double-clicking on the Place & Route Report).



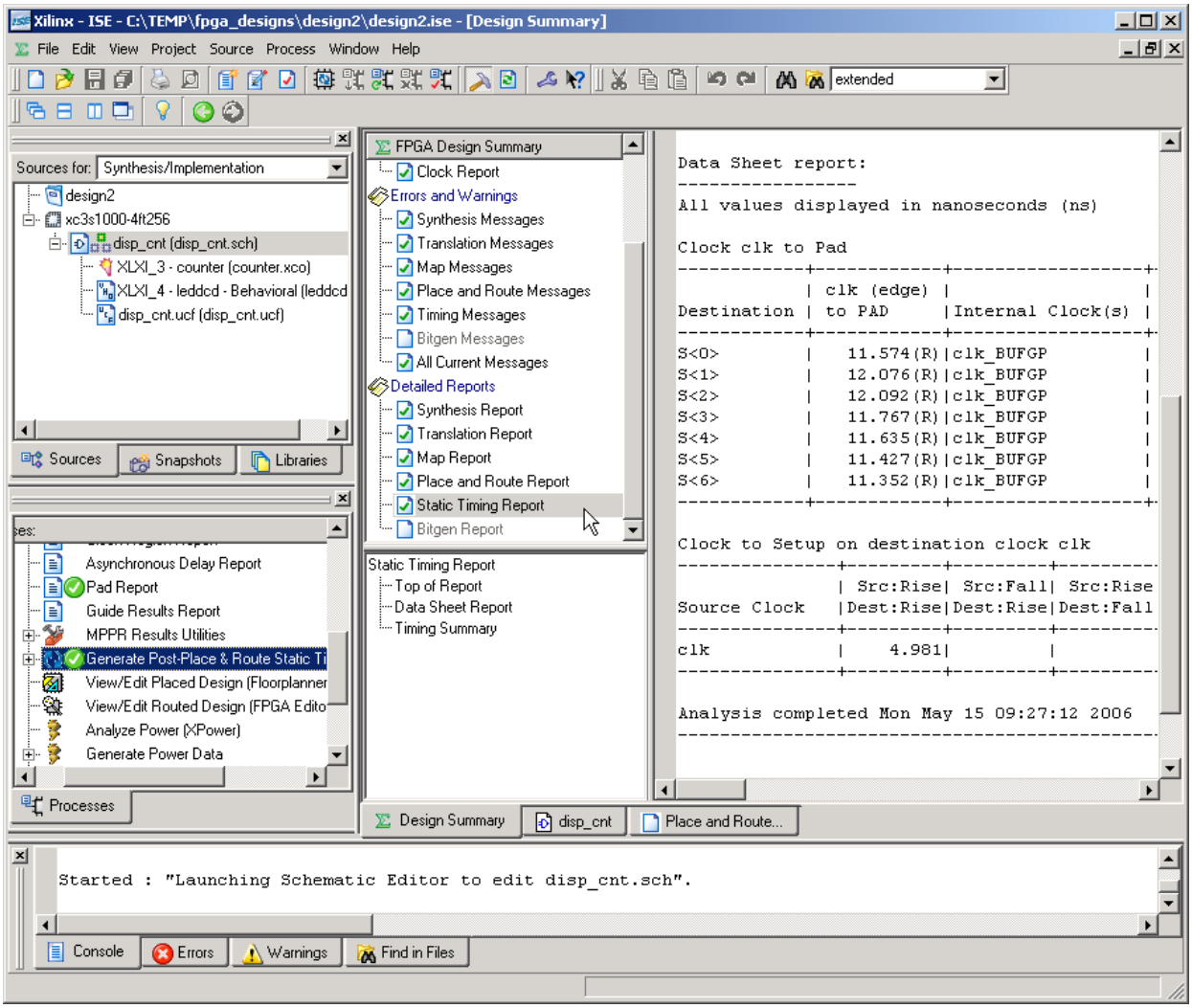
The displayable counter consumes 18 of the 7680 slices in the FPGA. Each slice contains two CLBs, so the displayable counter uses a maximum of 36 CLBs. The 28-bit counter requires at least 28 CLBs and the LED decoder requires 7 CLBs so this totals to 35 CLBs. So the device utilization statistics make sense.

As a precaution, you should also click on the Pinout Report in the Design Summary pane and check that the pin assignments for the clock input and LED decoder outputs match the assignments made with PACE. (You can make the comparison easier by clicking on the Signal Name column header to bring all the signal-pin assignments to the top.)



Checking the Timing

You have the displayable counter synthesized and implemented in the FPGA with the correct pin assignments. But how fast can the counter run? To find out, double-click on the Generate Post-Place & Route Static Timing process. This will determine the maximum delays between logic elements in the design taking into account logic and wiring delays for the routed circuit. (If a ✅ is already visible, then the static timing analysis has already been done.)

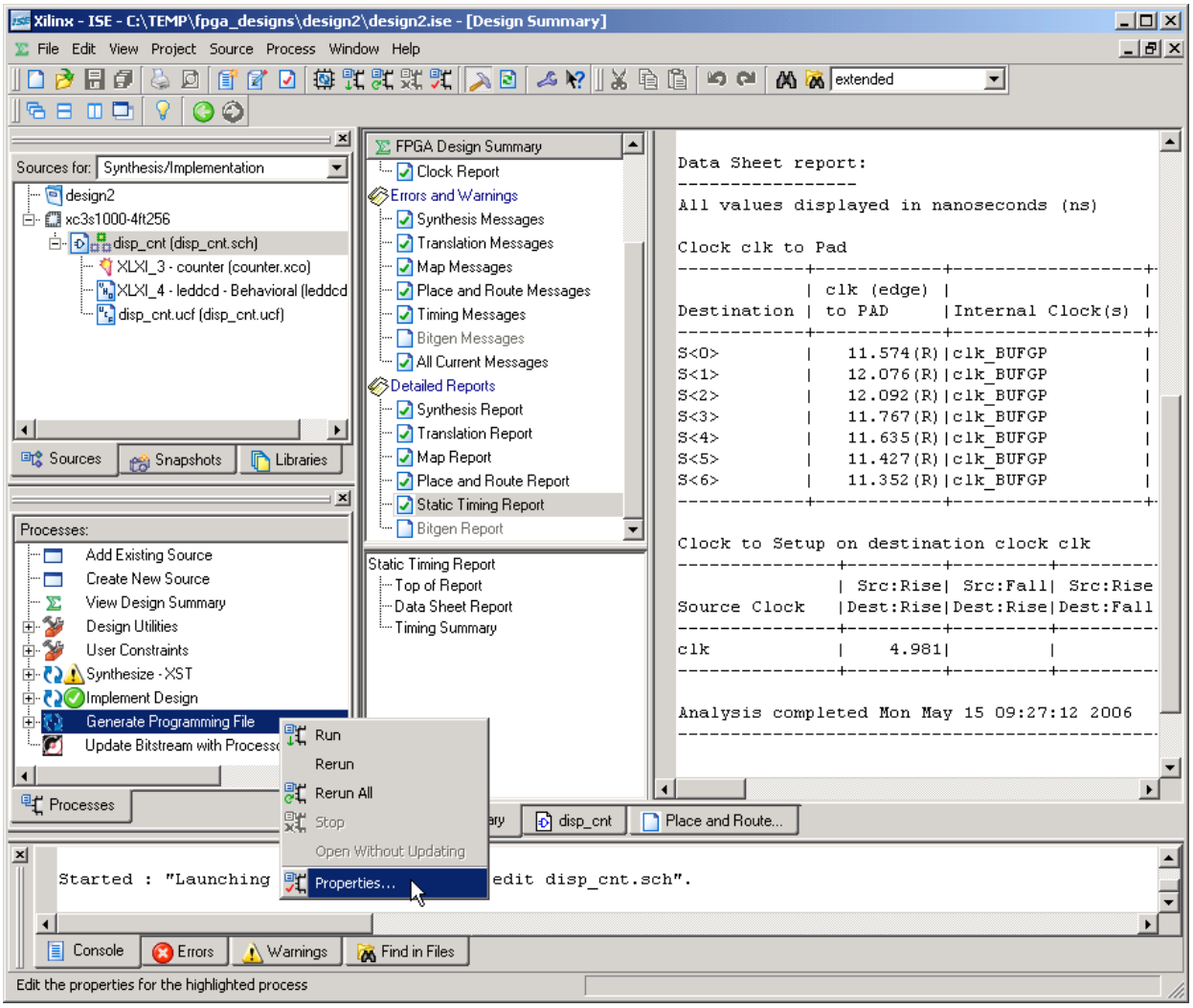


After the static timing delays are calculated, click on the Static Timing Report item in the Design Summary pane to view the results of the analysis. From the information shown at the bottom of the timing report, the minimum clock period for this design is 4.981 ns which means the maximum clock frequency is 200.8 MHz. The clock frequency on the XSA Board is 50 MHz which is well below the maximum allowable frequency for this design.

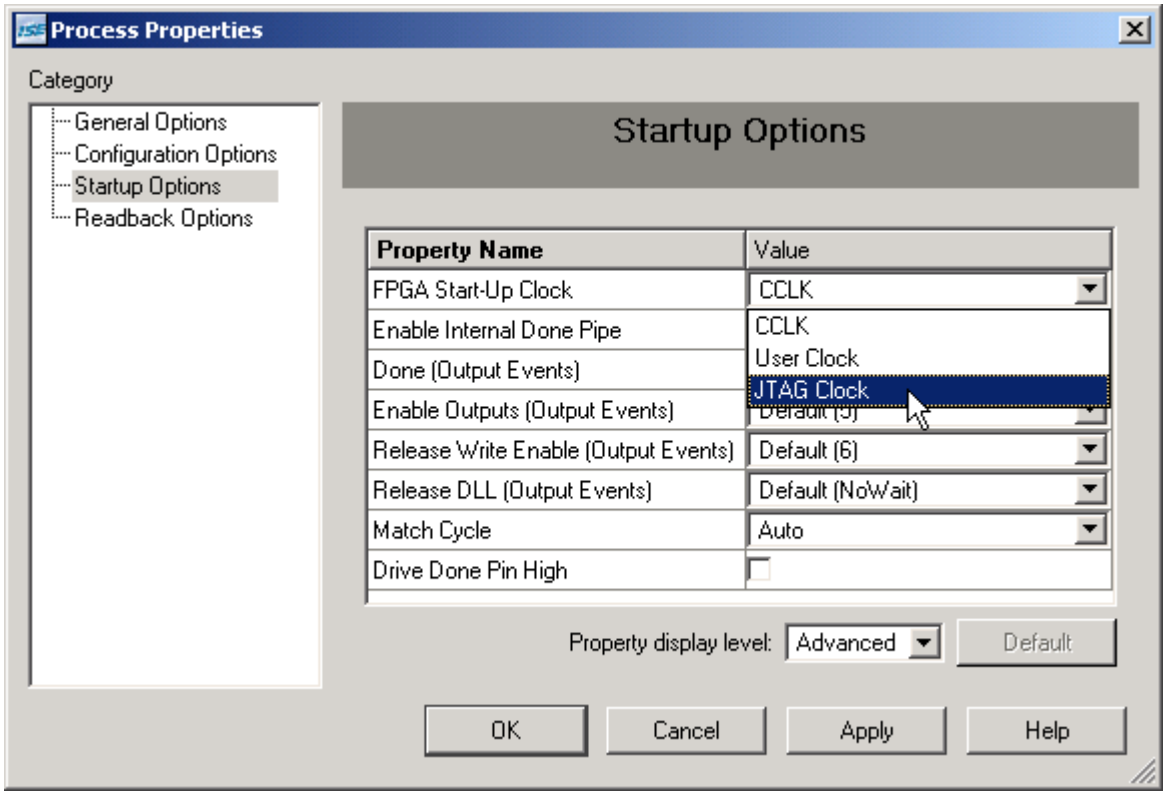
Generating the Bitstream

Now that you have synthesized our design and mapped it to the FPGA with the correct pin assignments, you are ready to generate the bitstream that is used to program the actual chip. In this example, rather than use the gxsload utility you will employ the downloading utilities built

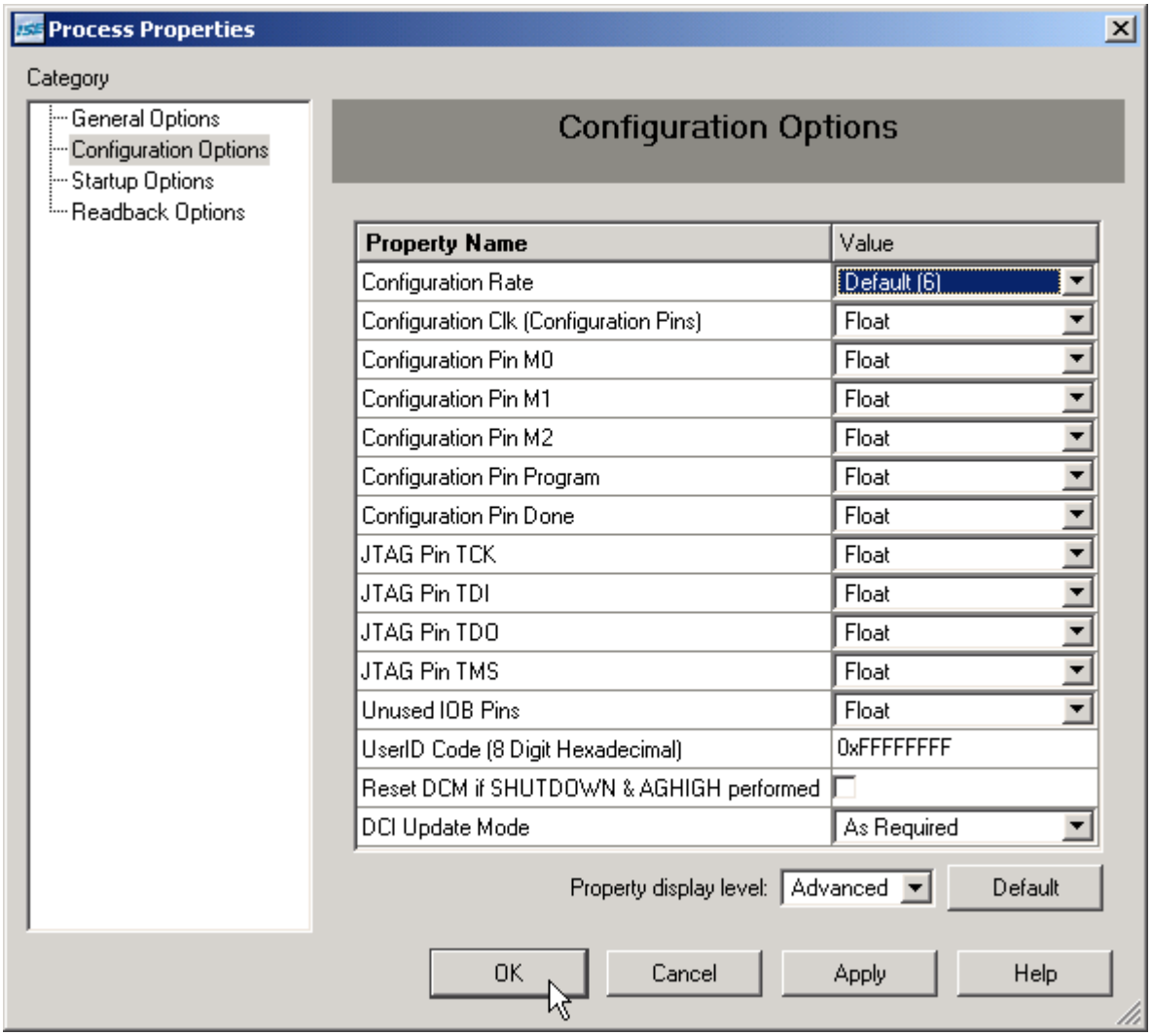
into WebPACK. The iMPACT programming tool downloads the bitstream through the JTAG interface of the FPGA, so you need to adjust the way the bitstream is generated to account for this. Right click on the Generate Programming File process and select the Properties... entry from the pop-up menu.



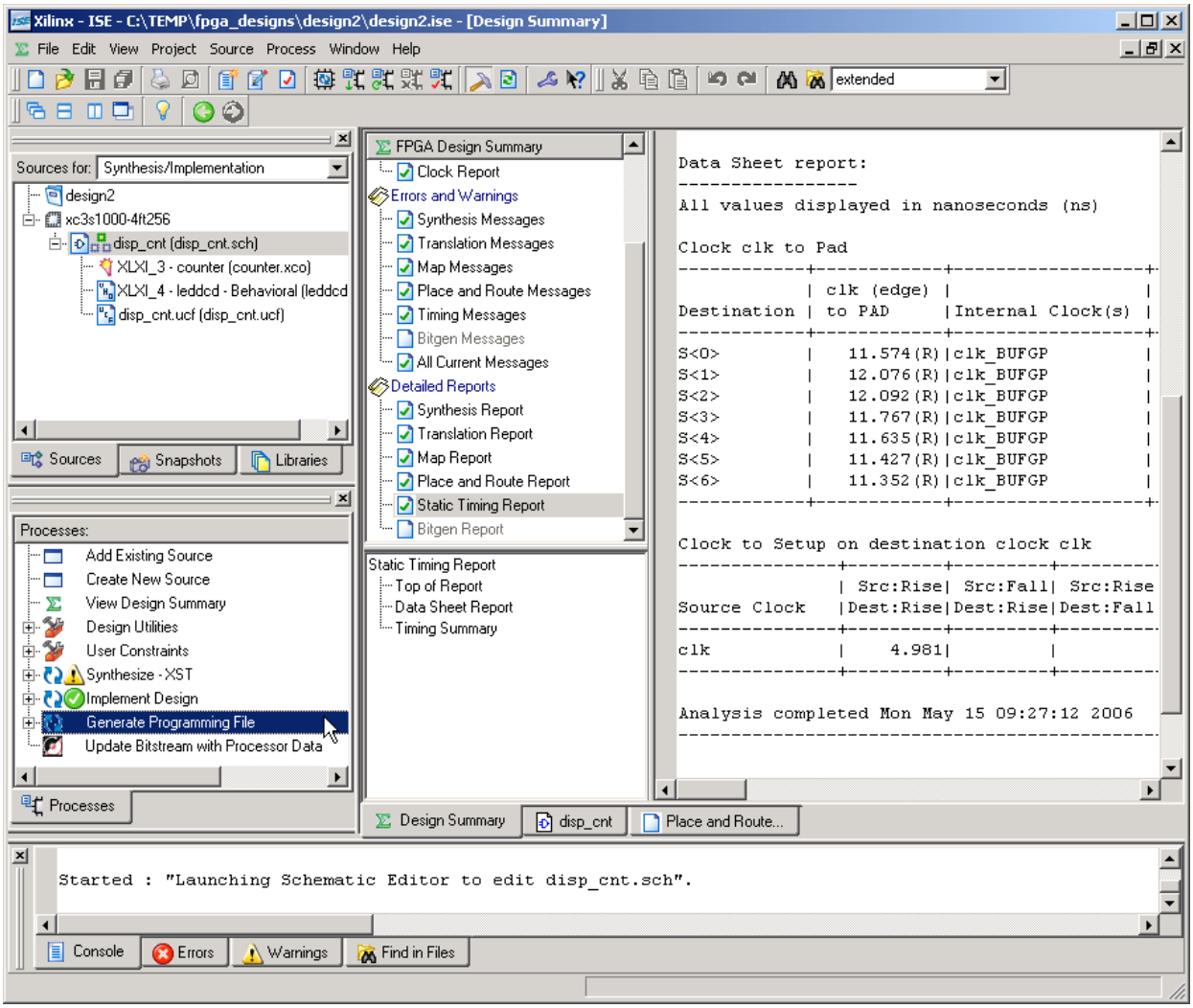
Select the Startup Options tab of the **Process Properties** window. Change the FPGA Start-Up Clock property to JTAG Clock so the FPGA will react to the clock pulses put out by the iMPACT tool during the final phase of the downloading process. If this option is not selected, the FPGA will not finish its configuration process and it will fail to operate after the downloading completes. Note that the startup clock is only used to complete the configuration process; it has no affect on the clock that is used to drive the actual circuit after the FPGA is configured.



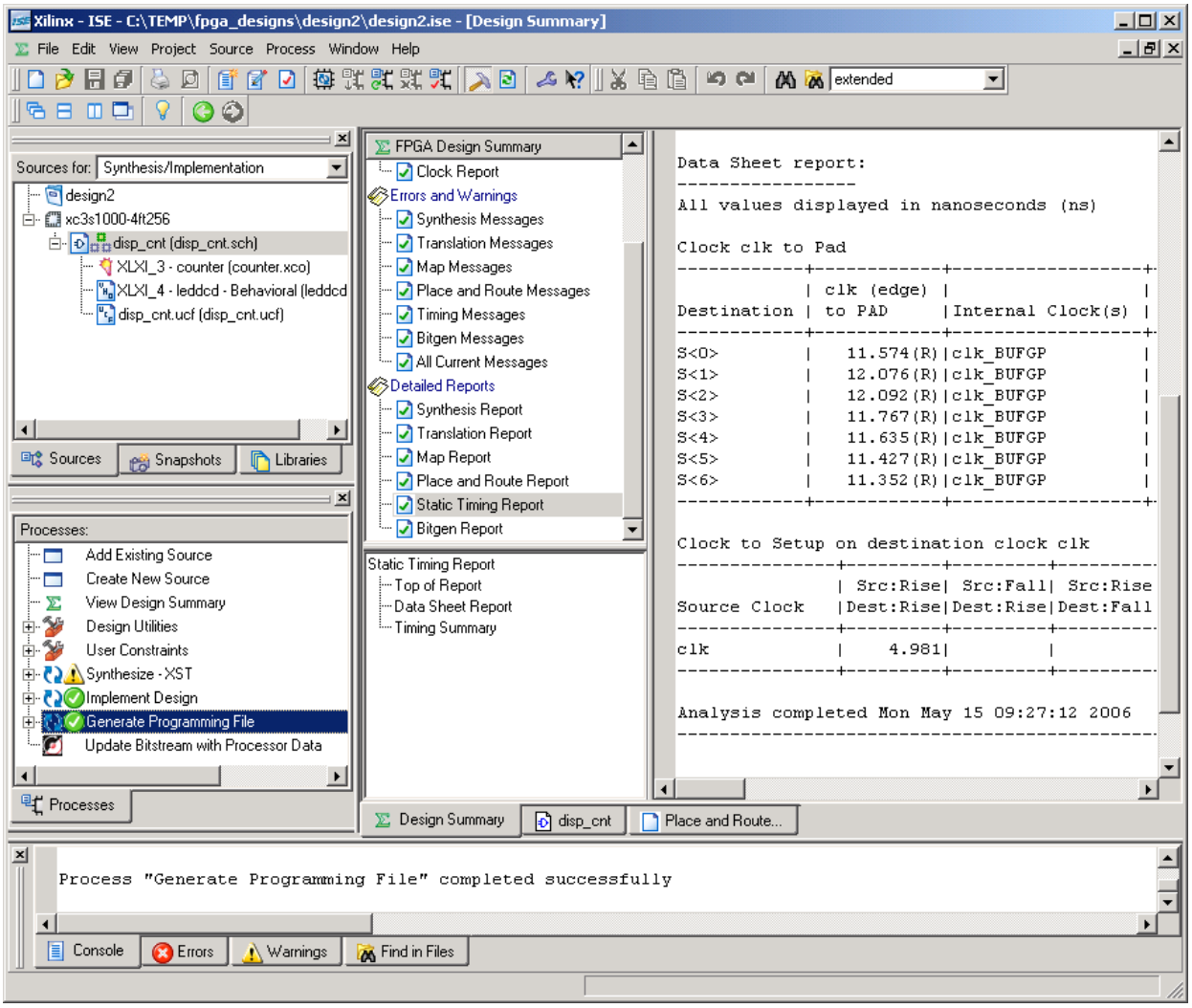
Next, click on the Configuration Options tab and disable all the internal pull-up and pull-down resistors in the FPGA as we did in the previous design. Then click OK.



Now that the bitstream generation options are set, highlight the disp_cnt object in the Sources pane and double-click on the Generate Programming File process to create the bitstream file.



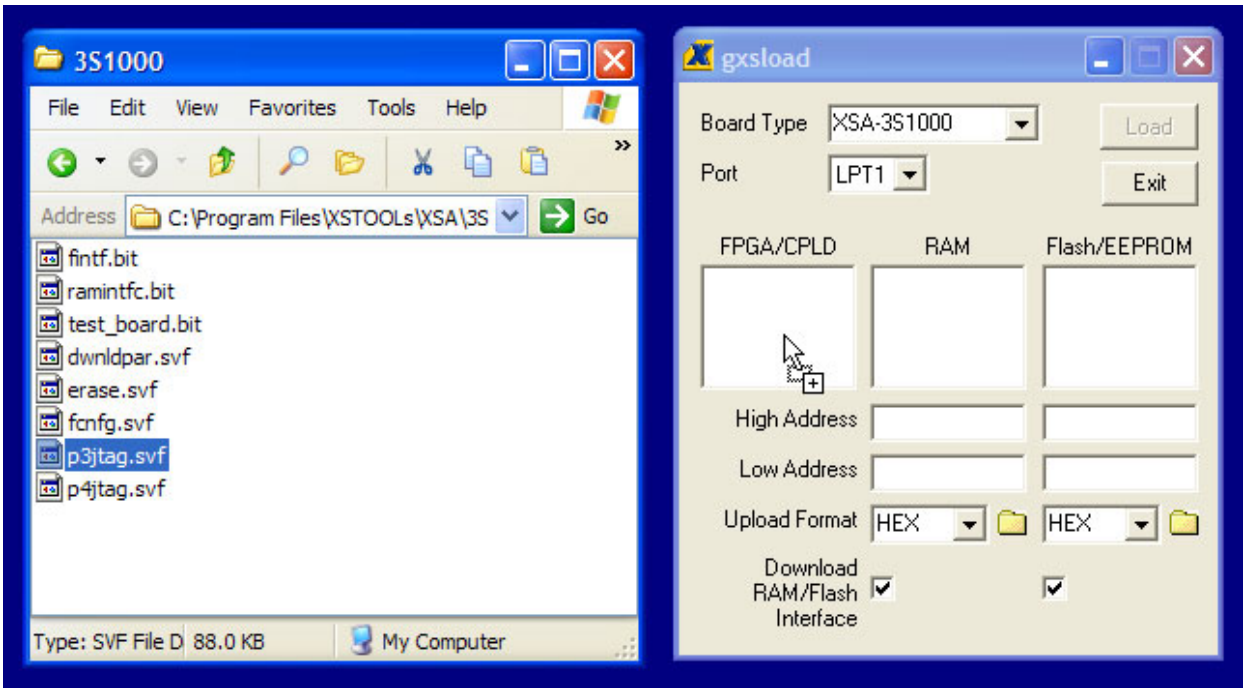
Within a few seconds, a ✅ will appear next to the Generate Programming File process and a file detailing the bitstream generation process will be created. A bitstream file named disp_cnt.bit is placed in the design2 folder.



Downloading the Bitstream

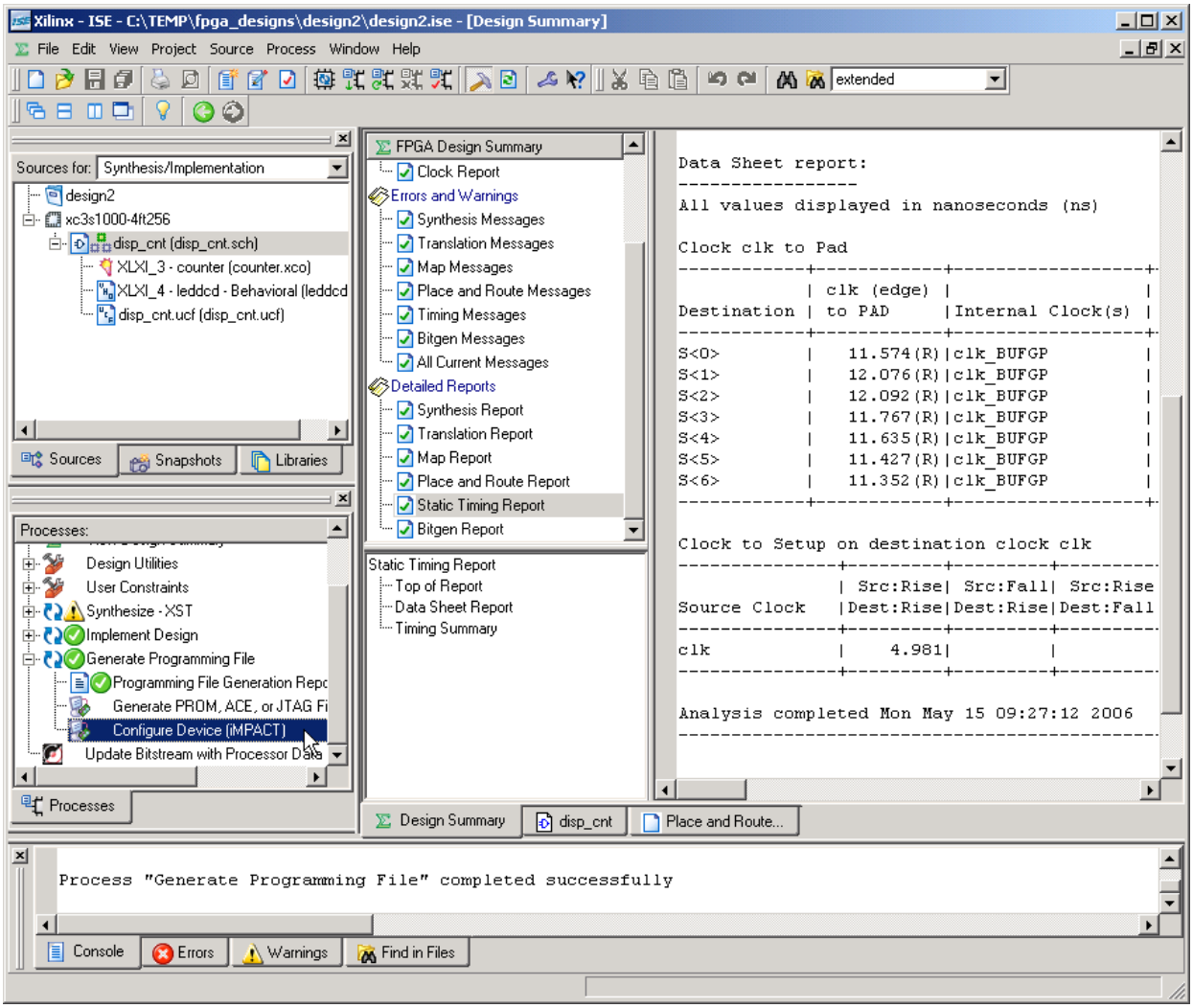
Before downloading the disp_cnt.bit file, you must configure the interface CPLD on the XSA-3S1000 board so it will work with the iMPACT programming tool. Double click the 

icon and then drag & drop the p3jtag.svf file from the C:\Program Files\XSTOOLS\XSA\3S1000 folder into the FPGA/CPLD pane of the **gxsload** window. Then click on the Load button and the CPLD will be reprogrammed in less than a minute.

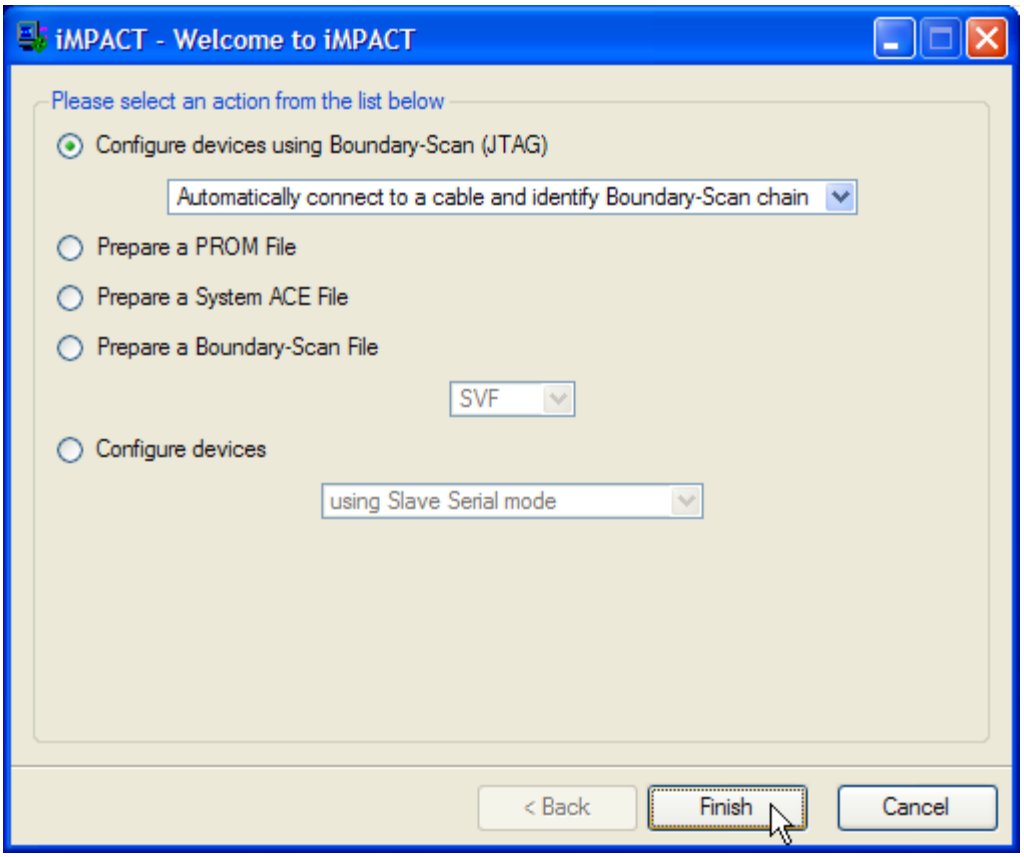


After the p3jtag.svf file is loaded into the XSA Board, move the shunt on jumper J9 from the **xs** to the **xi** position. The XSA Board is now setup so the FPGA can be configured through its JTAG boundary-scan pins with the iMPACT programming tool. Note that this process only needs to be done once because the CPLD on the XSA Board will retain its configuration even when power is removed from the board. (If you want to go back to using the gxsload programming utility, you must move the shunt on J9 back to the **xs** position and download the dwnldpar.svf file into the CPLD.)

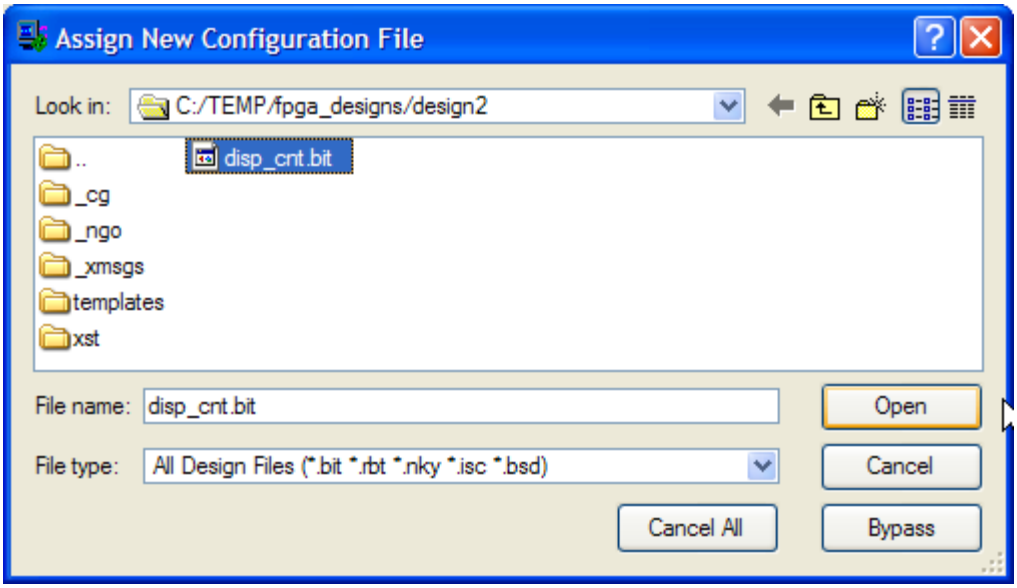
Now double-click on the Configure Device (iMPACT) process.



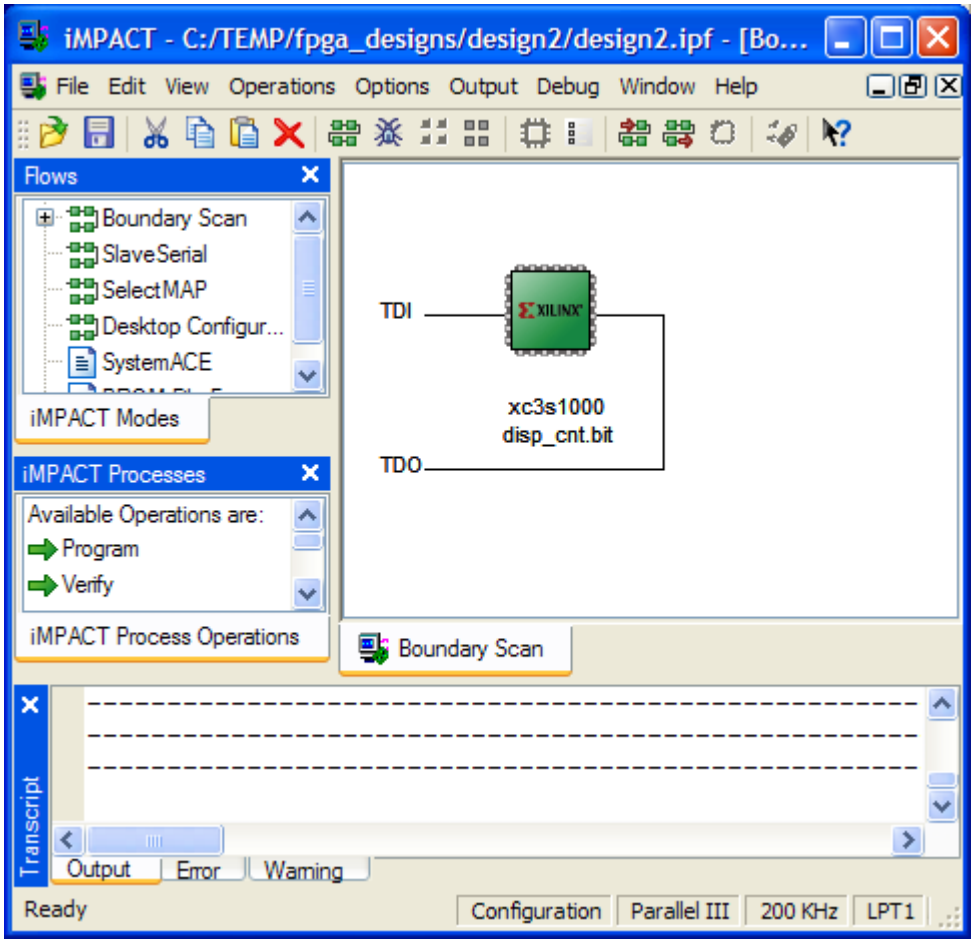
The **Configure Devices** window now appears. You just programmed the CPLD on the XSA Board so it would support programming of the FPGA through its boundary-scan pins. Boundary-scan mode allows the configuration of multiple FPGAs connected together in a chain. To accomplish this, the iMPACT software needs to know the types of the FPGAs in the chain. There is just a single FPGA on the XSA Board and you could easily describe this to iMPACT. But iMPACT can also probe the boundary-scan chain and automatically identify the types of the FPGAs. This is even easier, so select the Automatically connect to a cable and identify Boundary-Scan chain option and click on the Next button.



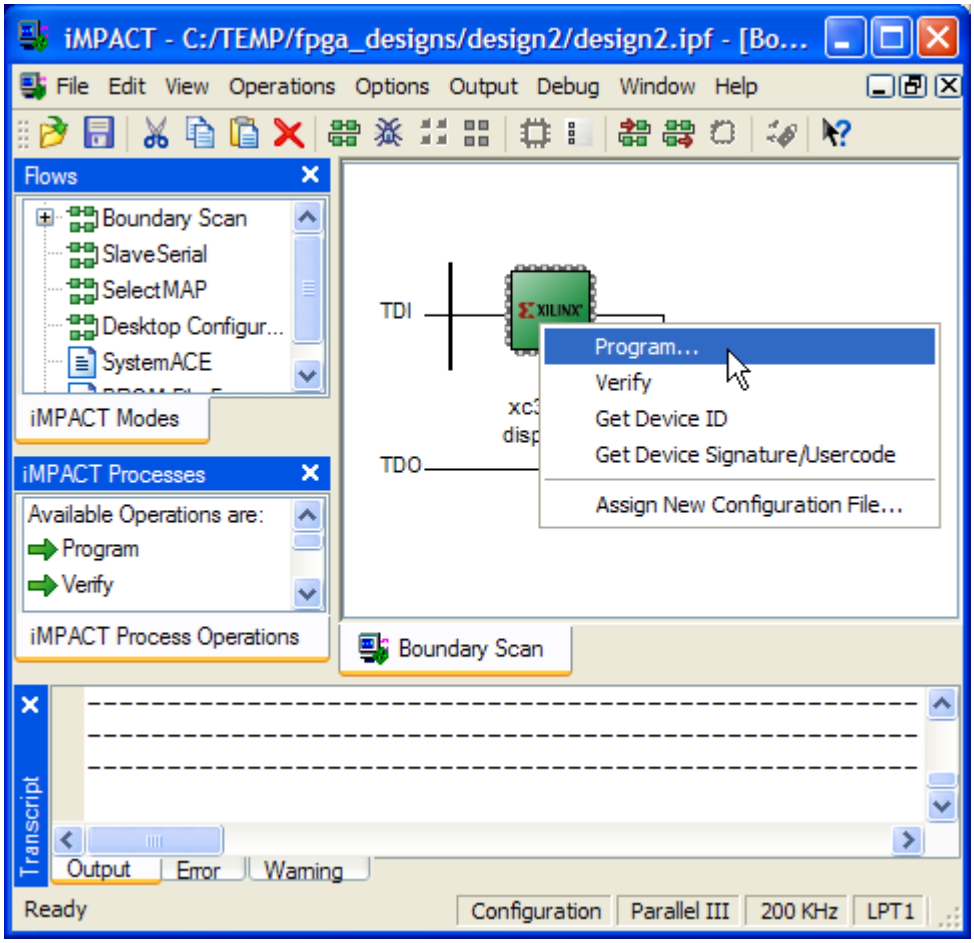
The **Assign New Configuration File** window now appears. Now you need to tell iMPACT what bitstream file to download into the FPGA. Go to the tmp\fpga_designs\design2 folder and highlight the disp_cnt.bit file. Then click on the Open button.



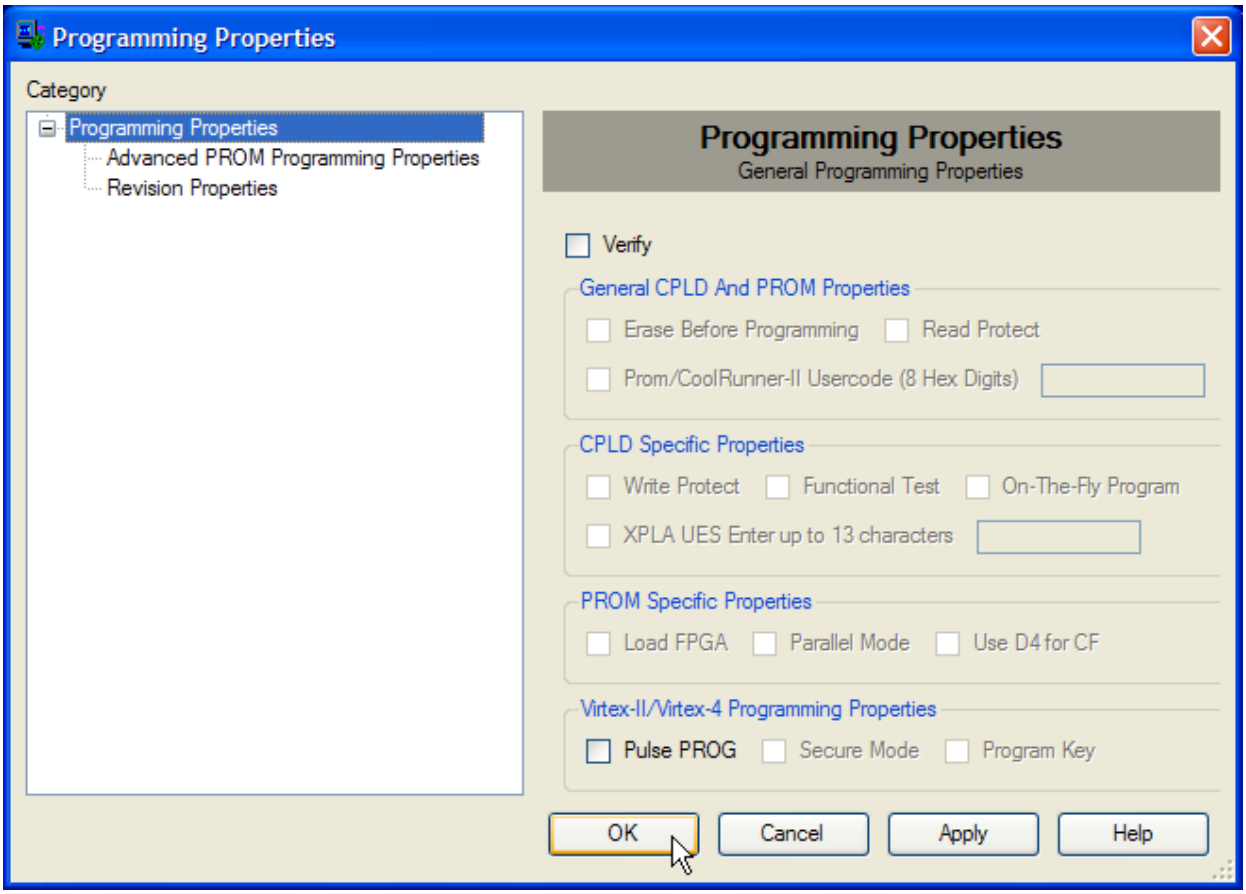
The iMPACT software will probe the boundary-scan chain and the main **iMPACT** window will appear showing a boundary-scan chain consisting of a single XC3S1000 FPGA.



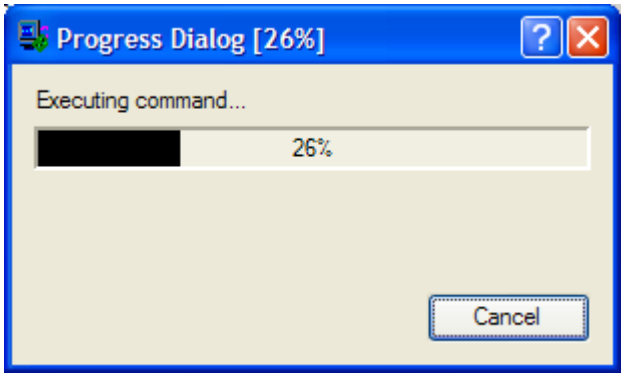
Now right-click on the xc3s1000 icon and select the Program... item on the pop-up menu.



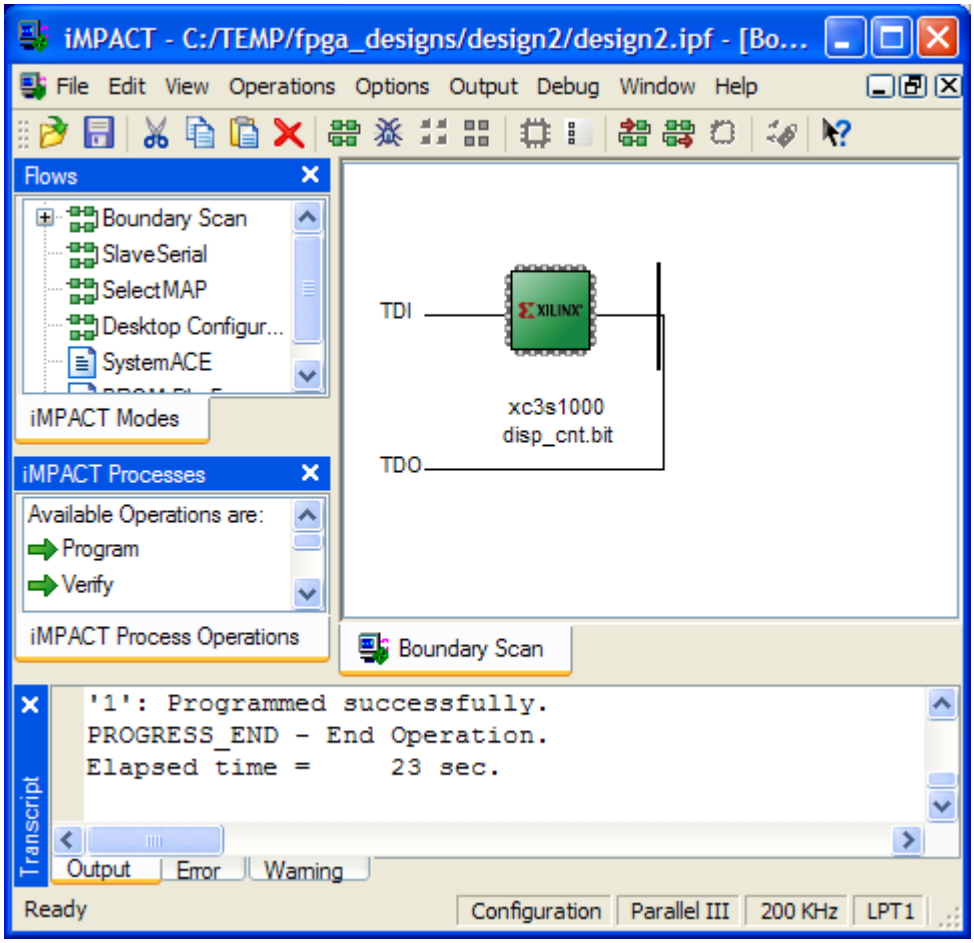
The **Program Options** window will appear. All you need to do at this point is click on the OK button to begin loading the disp_cnt.bit file into the FPGA.



The progress of the bitstream download will be displayed. The download operation should take less than a minute.



After the download operation completes, you can check the status messages in the bottom pane of the **iMPACT** window to see if the FPGA was configured successfully. The large message block in the main window also helps.



Testing the Circuit

Once the FPGA on the XSA Board is programmed, the circuit will begin operating without any further action from you. The LED display should repeatedly count through the sequence 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, c, d, E, F with a complete cycle taking 5.4 seconds.

5

Going Further…

OK! You made it to the end! You have scratched the surface of programmable logic design, but how do you learn even more? Here are a few easy things to do:

- Select Help➔Software Manuals. You will be presented with an Adobe Acrobat document that lists all the manuals for the ISE/WebPACK software. This includes a 300-page set of guidelines on synthesis and simulation techniques for FPGA designs.
- Select Help➔Xilinx on the Web➔Xilinx Application Notes. This will take you to a large set of interesting designs that have been done using Xilinx FPGAs.
- Get *Essential VHDL* (ISBN:0-9669590-0-0) or *The Designer's Guide to VHDL* (ISBN:1-55860-270-4) to learn more about VHDL for logic design.
- Read the *comp.arch.fpga* newsgroup for helpful questions and answers about programmable logic design.