

Introduction to WebPACK 6.1

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

Release date: 10/30/2003

All XS-prefix product designations are trademarks of XESS Corp.

All XC-prefix product designations are trademarks of Xilinx.

Table of Contents

What This Is and <i>Is Not</i> 1
FPGA Programming3
Installing WebPACK5
Getting WebPACK5
Installing WebPACK6
Getting XSTOOLs6
Installing XSTOOLs7
Our First Design8
An LED Decoder8
Starting WebPACK Project Navigator10
Describing Your Design With VHDL21
Checking the VHDL Syntax28
Fixing VHDL Errors29
Synthesizing the Logic circuitry for Your Design
Synthesizing the Logic circuitry for Your Design
Implementing the Logic Circuitry in the FPGA
Implementing the Logic Circuitry in the FPGA
Implementing the Logic Circuitry in the FPGA
Implementing the Logic Circuitry in the FPGA 33 Checking the Implementation 35 Assigning Pins with Constraints 36 Viewing the Chip 45
Implementing the Logic Circuitry in the FPGA 33 Checking the Implementation 35 Assigning Pins with Constraints 36 Viewing the Chip 45 Generating the Bitstream 53
Implementing the Logic Circuitry in the FPGA33Checking the Implementation35Assigning Pins with Constraints36Viewing the Chip45Generating the Bitstream53Downloading the Bitstream55
Implementing the Logic Circuitry in the FPGA33Checking the Implementation35Assigning Pins with Constraints36Viewing the Chip45Generating the Bitstream53Downloading the Bitstream55Testing the Circuit57
Implementing the Logic Circuitry in the FPGA33Checking the Implementation35Assigning Pins with Constraints36Viewing the Chip45Generating the Bitstream53Downloading the Bitstream55Testing the Circuit57Hierarchical Design59

Adding a Counter66
Tying Them Together72
Constraining the Design97
Synthesizing the Logic Circuitry for the Design103
Implementing the Logic Circuitry in the FPGA104
Checking the Implementation104
Checking the Timing106
Generating the Bitstream107
Downloading the Bitstream112
Testing the Circuit
Going Further



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 to program 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-50 Board.

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.

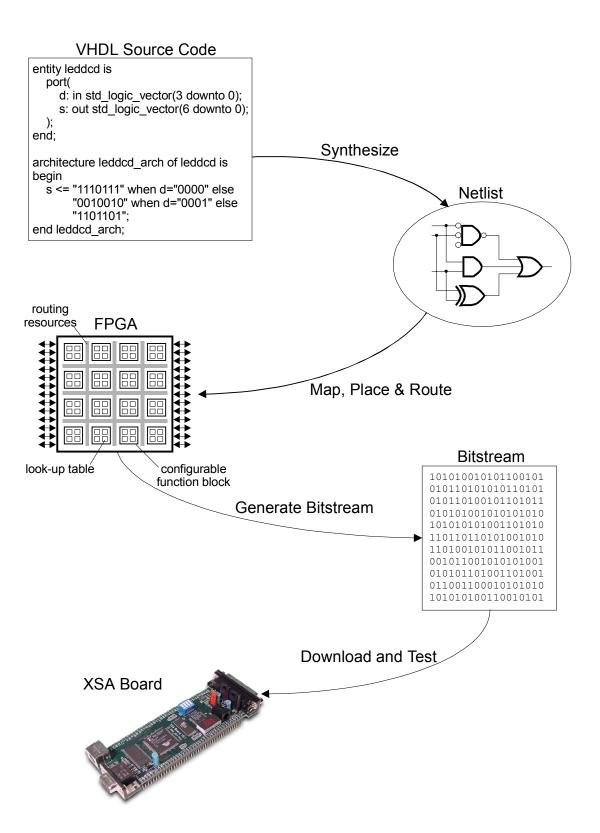
1

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 gate collections to specific CLBs while opening or closing the switches in the routing matrices to connect the gates 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 an <u>XSA-50 Board</u> containing a Xilinx XC2S50 SpartanIIE FPGA.



2

Installing WebPACK

Getting WebPACK

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

Home : ISE Logic Design T	ools : Free ISE WebPACK - Microsoft Internet Explorer	
		1
Address 🙆 http://www.xilinx.co		- 👸 - 🔗 🗗
	Products and Services	
Silic	con Solutions <u>Design Resources</u> System Resources Services Docume	entation
Entire Site	Home: Products & Services : ISE Logic Design Tools : ISE WebPACK ISE WebPACKTM ISE WebPACK is a free, downloadable, design software solution that contains support for advanced HDL entry, synthesis, and verification capabilities for both CPLD and FPGA designs The 6.11 ISE WebPACK release is available by using either the WebInstall	Related Features Revision History Help ISE WebPACK FAQs ISE WebPACK Overview Order ISE WebPACK CD ISE WebPACK Update Information (latest 09/25/03) Earlier ISE WebPACK releases
Module Description Revision History Buy Samples Help ISE WebPACK Home	technology (recommended) or the conventional single file download. Please note - For the 6.1 release the only supported platforms are Windows XP and Windows 2000 with Service Pack2 or greater.	 Flash based tutorial helps start you off quickly
WebInstall Single File Download	Xilinx WebInstall (1.5 Mb) WebInstall is a Xilinx developed program that will quickly download and install only the files needed by the families and tools you desire. Just download and run the WebInstall program and select your desired installation. The current Service Pack will be installed automatically at the	▼ ■ Internet

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

download completes, you also need to download <u>Service Pack 1</u> to get the most current WebPACK updates.

🚳 Home : ISE Logic Design T	ools : Free ISE WebPACK - Microsof	t Internet Explor	er				٦×
] ← - → - 🔕 😰 🖓 📾	🛞 🍏 🖨 👘 📋 <u>F</u> ile <u>E</u> dit	⊻iew F <u>a</u> vorites	<u>T</u> ools	: <u>H</u> elp)		
Address 🙆 http://www.xilinx.co	m/webpack/index.htm 🔄 (∂Go <mark>Goog</mark>	le -			- 💏 - 🔗	₽
	ISE WebPACK Single File Download Below are the single-file download m WebPACK releases. These software feature set that best suits your needs its own installer. For more details, ple page.	iodules similar to packages allow . Each self-extrac	you to : ting zip	select t i file co	he ntains		×
	ISE WebPACK Download Module (service pack required - see information below)	Download Size	PLD Design Environment	CPLD Device Support	FPGA Device Support		
	Complete ISE WebPACK Software - includes programming tools (뉴)	187 Mb*	-	~	-		
	Complete CPLD Tool Set - includes programming tools	121 Mb*	-	-			
	CPLD Programming Tools	20 Mb*		1			
	Complete Programming Tools	47 Mb*		4	~		
	* Customers installing the above <i>We</i> , download and install the latest Servic available. Service pack installation ins located in the <u>README</u> .	e Pack. At this tir	ne, <u>Ser</u>	vice Pa		-	•
ittp://direct.xilinx.com/dir	ect/webpack/61/WebPACK_61_fcfu	II_i.exe				🔹 Internet	

Installing WebPACK

After the WebPACK software download completes, double-click the WebPACK_61_fcfull_i.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 repeat this procedure with the Service Pack 1 install file 6_1_01i_pc.exe.

Getting XSTOOLs

If you are going to download your FPGA bitstreams into an XSA-50 Board, then you will need to get the XSTOOLS software from <u>http://www.xess.com/ho07000.html</u>. Just download the <u>xstools4_0_3.exe</u> file.

Installing XSTOOLs

Double-click the xstools4_0_3.exe file. The installation script will run and install the software. Accept the default settings for everything.



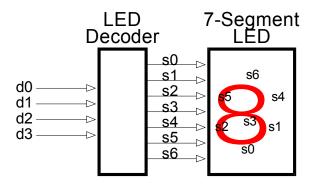
Our First Design

An LED Decoder

The first FPGA design we 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	5
0011	3	Э
0100	4	ч
0101	5	5
0110	6	6
0111	7	٦
1000	8	8
1001	9	9
1010	A	8
1011	В	8
1100	С	C
1101	D	D
1110	E	E
1111	F	F

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

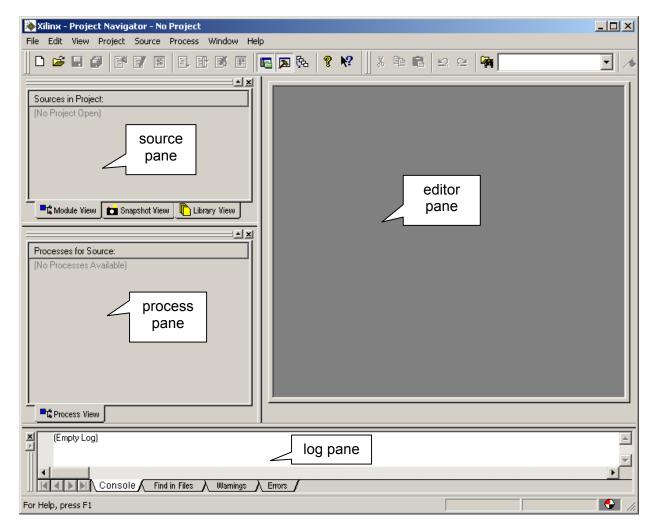


Starting WebPACK Project Navigator



We start WebPACK by double-clicking the Navigator 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 our design. There are four tabs so we can view the source files, functional modules, or HDL libraries for our project or look at various snapshots of the project.
- 2. A **process pane** that lists the various operations we can perform on a given object in the source pane.
- 3. A log pane that displays the various messages from the currently running process.
- 4. An **editor pane** where we can enter HDL code. Schematics are entered in a separate window.



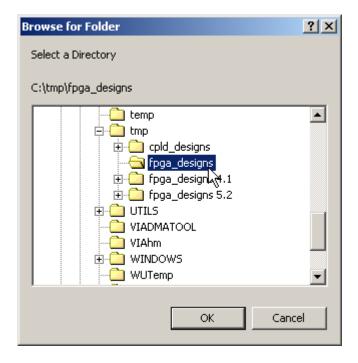
To start our design, we must create a new project by selecting the File→New Project item from the menu bar.

ò Xilinx - Project Navigator ·	No Project	
File Edit View Project Sour	e Process Window Help	
New Project Open Project	i i i i i i i i i i i i i i i i i i i	≌ 🙀 🔽 🖌
Open Example Close Project Save Project As		
New Ctrl+N		
Open Ctrl+O		
Close		
Save Ctrl+S Save As		
Print Ctrl+P	Tiew Library View	
Save All		
Recent Projects Recent Files		
Exit		
	·	
Process View		
(Empty Log)	Find in Files λ Warmings λ Errors /	× •
Create a new project		

This brings up the **New Project** window where we can enter the location of our project files, project name, the target device for this design, and the tools used to synthesize logic from our source files.

Enter a Name and Loca	ation for the Projec	:t	 	
Project Name:		Project Location: C:\ise\ISEexamp		
Select the tupe of Top I	.evel module for ti	he Project		
Select the type of 1 op-t				
Top-Level Module Ty		-		
Top-Level Module Ty				
Top-Level Module Ty				
Top-Level Module Ty				

Click on the ... button next to the Project Location field and use the **Browse for Folder** window to select a folder where our project files will be stored. For our design examples, we will store everything in the C:\tmp\fpga_designs folder. Click on the OK button ater highlighting this folder.



Next we will give our LED decoder design the descriptive title of design1 by typing it into the Project name field. Then we click on the Next button to continue creating this project.

New Project	×
Enter a Name and Location for the Project	1
Project Name: Project Location: design1 C:\tmp\fpga_designs\design1	
Select the type of Top-Level module for the Project Top-Level Module Type: HDL	
]
< Back Next > Cancel Help	

To set the family of FPGA devices we will target with this design, click in the Value field of the Device Family property. Select the Spartan2 entry in the drop-down menu that appears.

Property Name	Value
Device Family	Spartan2 📃 💌
Device Realized	CoolRunner XPLA3 CPLDs
Package Speed Grade	CoolRunner2 CPLDs Spartan2
	Spartan2E
Top-Level Module Type	Spartan3 Č
Synthesis Tool	Virtex2
Simulator	Virtex2P VirtexE
Generated Simulation Language	XC9500 CPLDs
	XC9500XL CPLDs
	XC9500XV CPLDs

Then click in the Value field of the Device property to select a particular device within the device family. For our designs, we will select the XC2S50 since this is the device used in the XSA-50 Board where we will test our design.

Property Name Device Family	Spartan2
Device	xc2s50 💌
Package Speed Grade	xc2s15 xc2s30
	xc2s50
Top-Level Module Type Synthesis Tool	xc2s100 kč xc2s150
Simulator	xc2s200
Generated Simulation Language	VHDL

Now click in the Value field of the Package property and choose the TQ144 package style for the FPGA on the XSA-50 Board.

	Name	Value
Device Family		Spartan2
Device		xc2s50
Package		tq144
Speed Grade		tq144
		fg256 √ζ
Top-Level Module Type		pq208
Synthesis Tool		XST (VHDL/Verilog)
Simulator		Other
Generated Simulation Languag	ge	VHDL

Then set the speed grade property for the FPGA to -5.

Device Family		Spartan2
Device Rockage		xc2s50
Package Speed Grade		tq144 -5 ▼
Speed Glade		
Top-Level Mod	dule Type	
Synthesis Tool		-50
Simulator		Other
Generated Sim	ulation Language	VHDL

The Top-Level Module Type, Synthesis Tool, Simulator and Generated Simulation Language can all be left at their default values, so we can just click on the Next button to continue creating the project.

Device Fa Device	mily	Spartan2 xc2s50
Package		tg144
Speed Gra	de	-5
Top-Level	Module Type	HDL
Synthesis		XST (VHDL/Verilog)
Simulator		Other
Generated	Simulation Language	VHDL

Click the Next button on the next window that appears. (We will create the VHDL source code at a later step.)

Create	a New Source					
Create	a New Source					
Lreate	a New Source					
	C		T		New Source	
	Source File		Туре		New Source	
1					Remove	
					Tomovo	
Create	a new source to add to	the project (optio	nal). Only one new	source ca	in be specified not	N.
Addition	hal new sources can be	e added after proje	ect creation using t	he ''Projec	t->New Source''	
comma	nd					
		< Back	Next>	Cano	el Hel	

We have no existing source files to add to this project, so once again click the Next button.

- Add E	xisting Sources			
1 2 3 4	Source File	Туре	Copy to Projec	Add Source Remove
			onal sources can be added act->Add Copy of Source'' c	
		< Back	Next > Cano	cel Help

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

N	ew Project Information	×
	Project Navigator will create a new Project with the following specifications:	
	Project: Project Name: design1 Project Location: C:\tmp\fpga_designs\design1 Project Type: HDL Device: Device: xc2s50 Package: tq144 Speed Grade: -5 : Top-Level Module Type: HDL Synthesis Tool: XST (VHDL/Verilog) Simulator: Other Generated Simulation Language: VHDL	
	Kack Finish Cancel Help	_

Now the Sources pane in the **Project Navigator** window contains two items:

- 1. A project object called design1.
- 2. A chip object called XC2S50-5TQ144.

Xilinx - Project Navigator - C:\tmp\fpga_designs\design1\design1.npl	
File Edit View Project Source Process Window Help	
	• /*
Sources in Project: design1 xc2s50-5tq144	
Module View Snapshot View Library View	
Processes for Source: "xc2s50-5tq144" Add Existing Source Create New Source Design Entry Utilities	
C Process View	
(Empty Log)	×
Hierarchy is up to date.	- 🚱 //.

Describing Your Design With VHDL

Once all the project set-up is complete, we can begin to actually design our LED decoder circuit. We start by adding a VHDL file to the *design1* project. Right-click on the XC2S50-5TQ144 object in the Sources pane and select New Source ... from the pop-up menu as shown below.

🔌 Xilinx - Project Navigator - C:\tmp\fpga_designs	sign1\design1.npl
File Edit View Project Source Process Window I	
	⊇ 🔉 R: 💡 K? 🛛 X 🛍 🖻 🗠 🗠 🖓 🦳 💽 🗡
· · · · · · · · · · · · · · · · · · ·	
Sources in Project:	
cesson cesso ceso	
New Source	
Add Source ^M Insert Add Copy of Source Shift+I	
Remove Delete	
Move to Library	
Open	
Processes for Source: Toggle Paths	
Add Exis Properties	
Create New Source	
Process View	
Empty Log)	
(Empty Log)	<u> </u>
	Y
Console Find in Files Warmings	Errors /
Add a new source to the project	

This causes a window to appear where we must select the type of source file we want to add. Since we are describing the LED decoder with VHDL, highlight the VHDL Module item. Then we type the name of the module, <code>leddcd</code>, into the File Name field and click on Next.

New Source	×
 Schematic State Diagram Test Bench Waveform User Document Verilog Module Verilog Test Fixture VHDL Library VHDL Library VHDL Package VHDL Test Bench 	File Name: leddcd Location: c:\tmp\fpga_designs\design1
	Add to project
< Back Next >	Cancel Help

Then the **Define VHDL Source** window appears where we 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 click in the MSB field and increment the upper range of the input field to 3 while leaving 0 in the LSB field.

efine VHDL Source				>
Entity Name led	ded			
Architecture Name Bel	navioral			
Port Name	Direction	MSB	LSB	^
d	in	3 3	Q	
	in	4	>	
	in			-
	Back Next>	Cancel	I Help	
<				

Perform the same operations to create the seven-bit wide **s** bus that drives the LEDs.

Define VHDL Source				X
Entity Name leddo	d			
Architecture Name Beha	vioral			
Port Name	Direction	MSB	LSB	
d	in	3	0	
S	in	6 8	0	
	in	~ ~	2	
	in			_
	in			
	in ·			_
	in			_
	in			
	in			
< B	ack Next>	Cance	el Help	

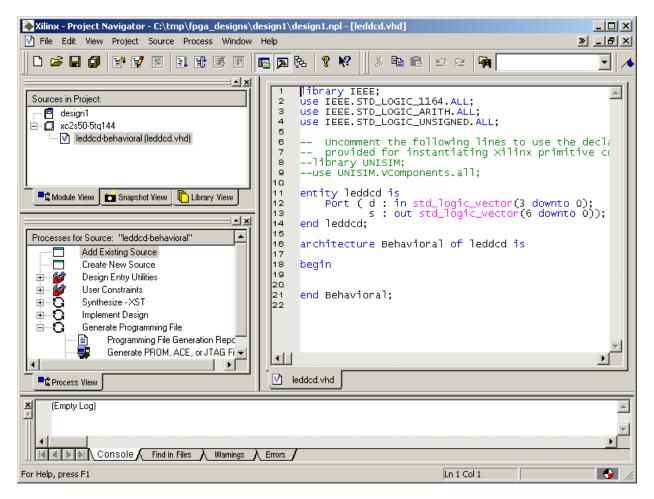
We must also click in the Direction field for the \mathbf{s} bus and select out from the drop-down menu in order to make the \mathbf{s} bus signals into outputs.

efine VHDL Source				2
Entity Name led	aca			
Architecture Name Be	havioral			
Port Name	Direction	MSB	LSB	
d	in	3	0	
s	in 💌	6	0	
	in]		
	out			
	inout kz	<u> </u>		
	in ·			_
	in			_
	in in			_
	in			_
	in			
	in			-
	in			-
	1		1	
		_		
<	Back Next>	Canc	el Hel	р

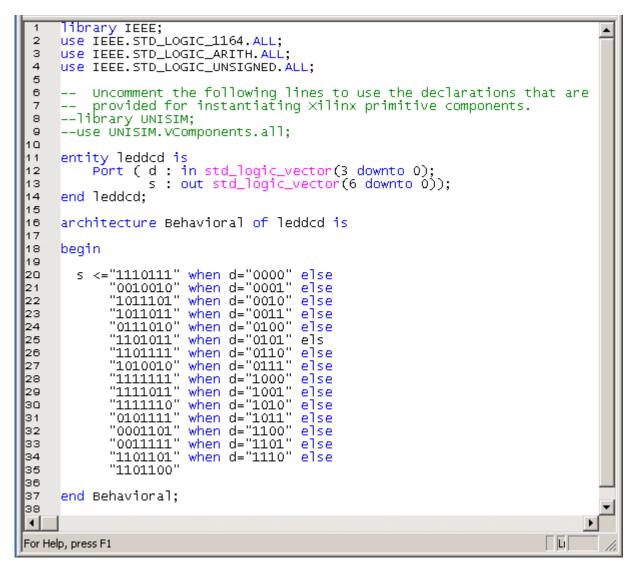
Click on Next in the **Define VHDL Source** window and we will get a summary of the information we just typed in:

New Source Informatio	n			×
Project Navigator will cr following specifications:		eleton source	e with the	
Source Type: VHDL Mo Source Name: leddcd.vi Entity Name: leddcd Architecture Name: Beh Port Definitions:	hd			<u> </u>
d s	vector: vector:	3:0 6:0	in out	
✓ Source Directory: c:\	tmp\fpga_des	signs\design1		¥ ¥
	Back	Finish	Cancel	Help

After clicking on Finish, the editor pane of the **Project Navigator** window displays a VHDL skeleton for our LED decoder. (We also see the leddcd.vhd file has been added to the Sources pane.) Lines 1-4 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 11-14. We will describe the logic operations of the decoder in the architecture section between lines 18 and 21.



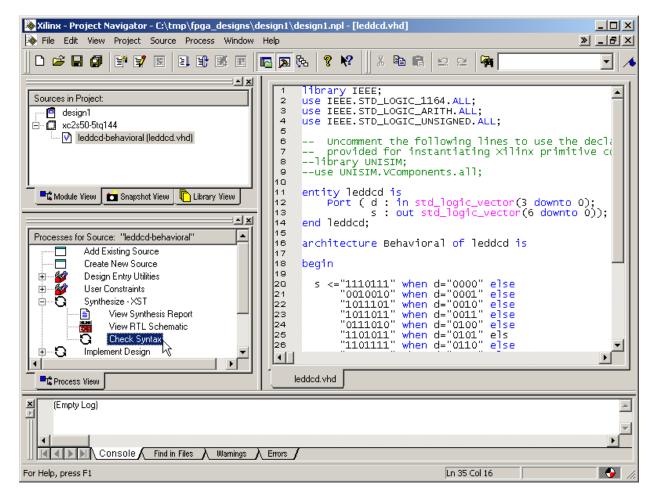
The completed VHDL file for the LED decoder is shown below. The architecture section contains a single statement which assigns a particular seven-bit pattern to the **s** output bus for any given four-bit input on the **d** bus (lines 20-35).



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

Checking the VHDL Syntax

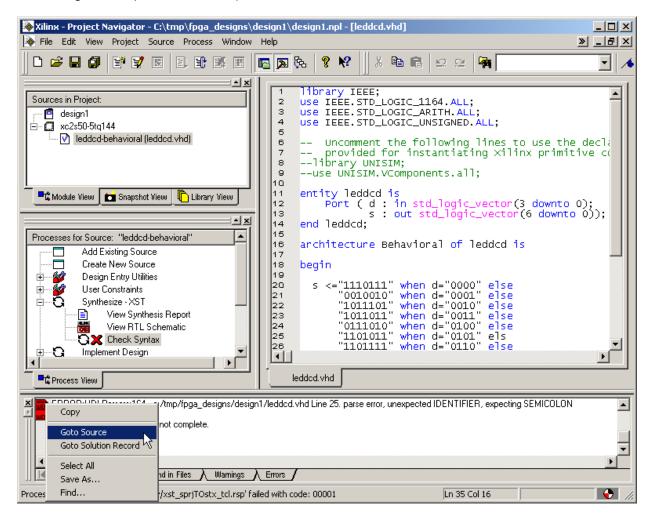
We can check for errors in our VHDL by highlighting the leddcd object in the Sources pane and then double-clicking on Check Syntax in the Process pane as shown below.



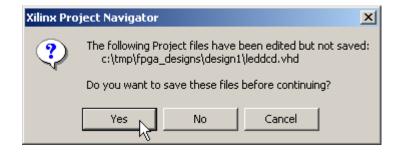
The syntax checking tool grinds away and then displays the result in the process window. In our case, an error was found as indicated by the X next to the Check Syntax process. But what is the error and where is it?

Fixing VHDL Errors

We can find the location of the error by scrolling the log pane at the bottom of the **Project Navigator** window until we find an error message. In this case, the error is located on line 25 and we can manually scroll there. You can also right-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.)



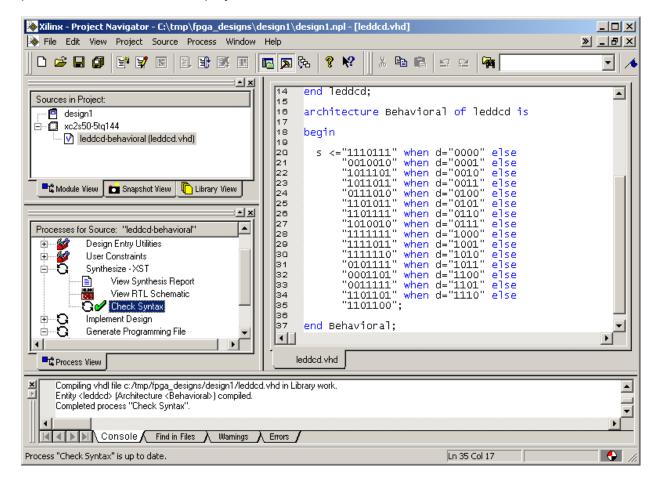
On line 25, we see that we have left the 'e' off the end of the else keyword. After correcting this error, we can double-click the on Check Syntax in the Process pane to re-check the VHDL code. We will be asked to save the file before the syntax check proceeds. Click on Yes.



Xilinx - Project Navigator - C:\tmp\fpga_designs\d		- I × » - I ×
	🖪 🗖 🗞 💡 🛠 📗 X 🛍 🖻 🗠 🗠 🖓	•
Sources in Project: design1 xc2s50-5tq144 Module View Snapshot View Library View Processes for Source: "leddcd-behavioral" Add Existing Source Create New Source Create New Source Synthesize - XST View Synthesis Report View Synthesis Report Synthesis Report Synthe	<pre>14 end leddcd; 15 16 architecture Behavioral of leddcd is 17 18 begin 20 s <="1110111" when d="0000" else 21 "0010010" when d="0001" else 22 "1011101" when d="0011" else 23 "101101" when d="0100" else 24 "0111010" when d="0101" else 25 "1101011" when d="0110" else 26 "1101111" when d="0110" else 27 "1010010" when d="1010" else 28 "1111111" when d="1010" else 29 "1111101" when d="1010" else 30 "1111101" when d="1010" else 31 "0001101" when d="1101" else 32 "0001101" when d="1101" else 33 "0001101" when d="1100" else 34 "110110" when d="1110" else 35 "1101100"</pre>	
ERROR:HDLParsers:164 - c:/tmp/fpga_designs/design ERROR: XST failed Process "Check Syntax" did not complete.	1/leddcd.vhd Line 37. parse error, unexpected END, expecting SEMICOLON	×
Process 'xst_sprjTOstx.tclprojnav/xst_sprjTOstx_tcl.rsp' fail	led with code: 00001 Ln 37 Col 1	

The syntax checker now finds another error on line 37 of the VHDL code.

When we look at line 37 we see it is just the end statement for the architecture section. The VHDL syntax checker was expecting to find a ';' and we can see it is missing from the end of line 35. Add the semicolon and save the file. Now when we double-click the Check Syntax process, it runs and then displays a \checkmark to indicate there are no more errors.



Synthesizing the Logic circuitry for Your Design

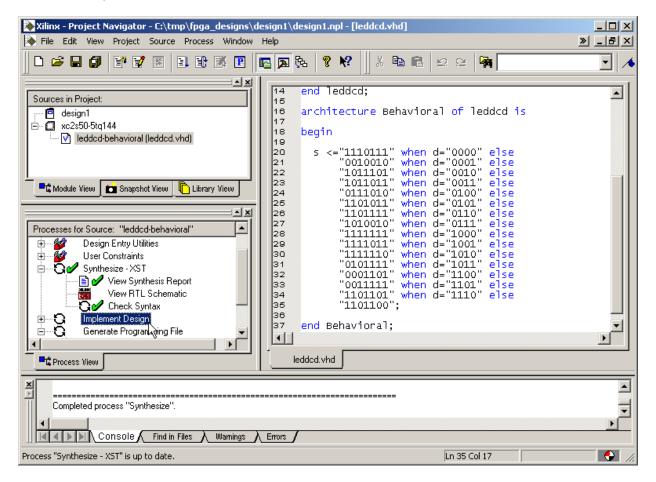
Now that we have valid VHDL for our design, we need to convert it into a logic circuit. This is done by highlighting the leddcd object in the Sources pane and then double-clicking on the Synthesize process as shown below.

📚 Xilinx - Project Navigator - C:\tmp\fpga_designs\de	esign1\design1.npl - [leddcd.vhd]	
File Edit View Project Source Process Window	Help	<u> - 8 ×</u>
	E 🔉 🗞 💡 🛠 🗍 X 🛍 🖻 🗠 🗠 🖓	•
Sources in Project: design1 xc2s50-5tq144 Module View Snapshot View Library View Processes for Source: "leddcd-behavioral" Design Entry Utilities User Constraints View Syntassis Report View RTL Schematic View RTL Schematic Check Syntas Implement Design Generate Programming File Process View Compiling vhdl file c:/tmp/fpga_designs/design1/leddcd. Entity <leddcd> (Architecture <behaviorab) compiled.<="" td=""><td><pre>if4 end leddcd; is architecture Behavioral of leddcd is is begin is constant state is constant st</pre></td><td></td></behaviorab)></leddcd>	<pre>if4 end leddcd; is architecture Behavioral of leddcd is is begin is constant state is constant st</pre>	
Completed process "Check Syntax".	Errors /	•
Process "Check Syntax" is up to date.	Ln 35 Col 17	

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 vill 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. 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.

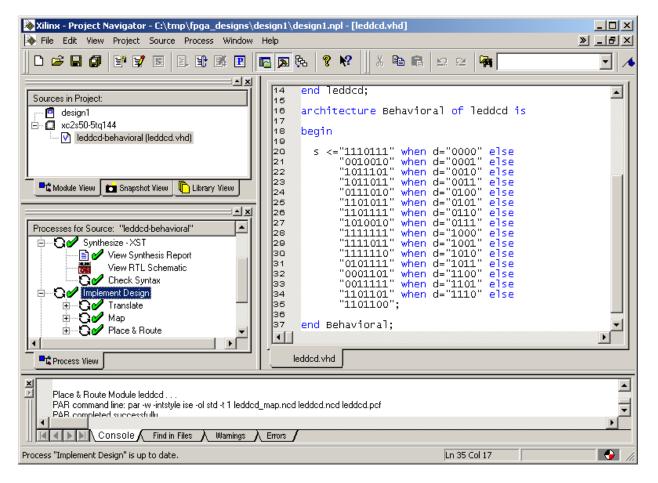
Implementing the Logic Circuitry in the FPGA

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



You can watch the progress of the implementation process in the status bar at the bottom of the **Project Navigator** window. For a simple design like the LED decoder, the fitting is completed in seconds (on a 850 MHz Athlon PC with 768 Mbytes or 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 we 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 FPGA device we selected. 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 ¹/₂ to indicate a successful completion but some warnings were issued or not all the subprocesses were enabled.



Checking the Implementation

We have our design fitted into the XC2S50 FPGA, but how much of the chip does it use? Which pins are the inputs and outputs assigned to? We can find answers to these questions by double-clicking on the Place & Route Report and the Pad Report in the Process pane.

📚 Xilinx - Project Navigator - C:\tmp\fpga_designs\de	esign1\design1.npl - [leddcd.vhd]	
File Edit View Project Source Process Window	Help	» _ ð ×
	▣ ▶ %	•
Sources in Project: 	<pre>14 end leddcd; 16 17 18 begin 19 20 s <="1110111" when d="0000" else 21 "0010010" when d="0001" else 22 "1011101" when d="0010" else 23 "101101" when d="0100" else 24 "0111010" when d="0101" else 25 "1101011" when d="0101" else 26 "1101111" when d="0110" else 27 "1010010" when d="0111" else 28 "1111111" when d="1000" else 29 "1111011" when d="1001" else 30 "1111110" when d="1001" else 31 "0001101" when d="1101" else 33 "001111" when d="1101" else 34 "1101101" when d="1101" else 35 "1101100"; 36 37 end Behavioral; 4</pre>	
Place & Route Module ledded PAR command line: par -w -intstyle ise -ol std -t 1 ledded PAR completed successfullu Console Find in Files Warnings	_map.ncd leddcd.ncd leddcd.pcf	
Process "Implement Design" is up to date.	Ln 35 Col 17	

The device utilization of the LED decoder circuit can be found near the top of the place & route report. The circuit only uses 4 of the 768 available slices in the XC2S50 FPGA. Each slice contains two CLBs and each CLB can compute the logic function for one LED segment output. Device utilization summary:

Number of Number	External of LOCed	 IOBs		out out			11% 0%
Number of	SLICES		4	out	of	768	1%

The pad report shows what pins the inputs and outputs use. The **d** inputs have been assigned to pins 124, 121, 130 and 118. The **s** outputs which drive the LED segments have been routed through pins 126, 116, 129, 122, 120, 123 and 117. (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
P116	s<1>	IOB	OUTPUT	LVTTL
P117	s<6>	IOB	OUTPUT	LVTTL
P118	d<3>	IOB	INPUT	LVTTL
P120	s<4>	IOB	OUTPUT	LVTTL
P121	d<1>	IOB	INPUT	LVTTL
P122	s<3>	IOB	OUTPUT	LVTTL
P123	s<5>	IOB	OUTPUT	LVTTL
P124	d<0>	IOB	INPUT	LVTTL
P126	s<0>	IOB	OUTPUT	LVTTL
P129	s<2>	IOB	OUTPUT	LVTTL
P130	d<2>	IOB	INPUT	LVTTL

Assigning Pins with Constraints

The problem we have 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 we actually want to use on the XSA-50 Board. The FPGA on the XSA-50 Board has eight inputs which are driven by the PC parallel port and we would like to assign the LED decoder inputs to four of these as follows:

LED Decoder Input	XSA-50 XC2S50 FPGA Pin
d0	P50
d1	P48
d2	P42
d3	P47

Likewise, the XSA-50 Board has a seven-segment LED attached to the following pins of the FPGA:

LED Decoder Output	XSA-50 XC2S50 FPGA Pin
s0	P67
s1	P39
s2	P62
s3	P60
s4	P46
s5	P57
s6	P49

How do we constrain the implementation process so it assigns the inputs and outputs to the pins we want to use? We start by right-clicking the leddcd object in the Sources pane and selecting New Source... from the pop-up menu.

Xilinx - Project Navigator - C:\tmp\fpga_desig File Edit View Project Source Process Wind		eddcd.pad_txt (R	EAD ONLY)]		LDL NBL «
	E 🖪 🏊 🗞 💡 📢	X 🖻 🖻	2 2 4		• 🔺
Sources in Project: design1 C. xc2s50-5tq144 New Source Add Source	128 P112 127 P113 128 P114 129 P115 130 P116 130 P116 130 P116 145 P131 146 P132	s<1> s<6> d<3> s<4> d<1> s<3> s<5> d<0> s<0> s<2> d<2>	IOB IOB	IO_VREF_6 GND IO_VREF_6 VCCINT IO_TRDY VCCO GND IO_IRDY IO_VREF_7	
Pad Report Guide Results Report	 Id8 P134 P135 Id8 P135 Id8 P135 Id8 P135 	eddcd.pad_t	Тов 	GND	
Place & Route Module leddcd PAR command line: par -w -intstyle ise -ol std -t 1 leddcd_map.ncd leddcd.ncd leddcd.pcf PAR completed successfully Console Find in Files Warmings Errors					
Add a new source to the project			Ln 124 Col 1		- 📀 /i.

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

New Source	×
 BMM File Implementation Constraints File MEM File Schematic State Diagram Test Bench Waveform User Document Verilog Module Verilog Test Fixture VHDL Library VHDL Module VHDL Package VHDL Test Bench 	File Name: leddcd Location: c:\tmp\fpga_designs\design1
< Back Next >	Cancel Help

Then we are asked to pick the file with which to associate the constraints. For this design there is only one choice, so click on the Next button and proceed.

Select	×
Source File	
leddcd	
	_
< Back Next > Cancel Help	

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

New Source Information	×
Project Navigator will create a new skeleton source with the following specifications:	
Source Type: Implementation Constraints File Source Name: leddcd.ucf Association: leddcd	<u> </u>
र	▼
Source Directory: c:\tmp\fpga_designs\design1	
< Back Finish Cancel	Help

Now double-click the leddcd.ucf object in the Sources pane to begin adding pin assignments to the design.

📚 Xilinx - Project Navigator - C:\tmp\fpga_designs\da	esign1\d	lesign1.n	pl - [leo	idcd.pad_txt	(READ ONLY)]	
File Edit View Project Source Process Window	Help						» _ B ×
	na 🔊 (£ 💡	₩?	X 🖻 🖬	1 2 2	A	•
Sources in Project: Carded Sign1 Carded Content of the second s	128 127 128 129 130 131 132 134 135 136 137 138 138 139 140 141 142 143 144 145 146 147 148 149	P112 P113 P114 P115 P116 P117 P120 P121 P122 P123 P124 P125 P126 P126 P127 P128 P129 P130 P131 P132 P134 P135		s<1> s<6> d<3> s<4> d<1> s<3> s<5> d<0> s<5> d<0> s<2> d<2>	IOB IOB IOB IOB IOB IOB IOB IOB IOB IOB	IO_VREF_6 GND IO_VREF_6 VCCINT IO_TRDY VCCO GND IO_IRDY IO_IRDY IO_VREF_7 GND	
Place & Route Module leddcd PAR command line: par -w -intstyle ise -ol std -t 1 leddcd_map.ncd leddcd.ncd leddcd.pcf							
Console Find in Files Warnings	Errors	,					
Hierarchy is up to date.					Ln 124 Co	ol 1	

The **Xilinx PACE** window now appears. Click on the Ports tab in the upper pane. A list of the current inputs and outputs for the LED decoder will appear in the **Design Object List – I/O Pins** pane. We can change our pin assignments here.

ЖXilinx PACE - c:\tmp\fpga_designs\design1\leddcd.ucf	
File Edit View IOBs Areas Tools Window Help	
🗅 🖆 🖨 🥌 🖍 🗰 🦊 👯 🔁 理: 🔯 🐼 😥 🗶 🛃 🛅 📰 📰 🔤	ା X Q Q X Q ◙ [
Design Browser	Package Pin Legend 🛛 🛛
	Symbol Pin Type
Global Logic	User IO
	User Prohibit
	GND GND
Design Object List - I/O Pins	🔘 GCLK / GCK
1/O Name 1/O Direction Loc Bank 1/O Std. Vr	Temperature Diode
s<6> Output	Not Connected
S S Output	Bank0
Image: s<4> Output Image: s<3> Output	Bank1
	Bank2
	Bank3
	Bank4
	Bank5
# Group I/O Direction Loc I/O Std. Vref Vcco D	Bank6
Part S Output Part 4 d Input	Bank7
名 4 d Input	
Package View Architecture View	
For Help, press F1	

We start by clicking in the Location field for the **d<0>** input. Then just type in the pin assignment for this input: P50. Do this for each of the inputs and outputs using the pin assignments from the tables shown previously. After doing this, the **Xilinx PACE** window appears as follows.

Xilinx PACE	- [Design Obj View IOBs A												
		H 🕹		F		: 🗹	ABC 🔏					Q. Q.)	∢ ℚ 😰 📗
1/O Name	1/O Direction	Loc	Bank	170 Std.	Vref V	cco Di	rive Str.	Termin	ation	Slew	Delay		
	Output	P49	BANK]	
	Output	P57	BANK										
	Output	P46	BANK										
	Output	P60	BANK										
	Output	P62	BANK										
	Output	P39 P67	BANK BANK									-	
	Output Input	P67 P47	BANK									-	
	Input	P42	BANK									-	
	Input	P48	BANK										
	Input	P50	BANK										
# Group I.	/O Direction	Loc	1/0 Sta	l. Vref V	cco Dr	ive Str.	Termir	nation	Slew	Delay	Swap		
	lutput										Yes		
4 d Ir	nput										Yes		
						_			_				

After the pin assignments are entered, click on the \square button to save the pin assignment constraints. Then select File \rightarrow Exit to close the **Xilinx PACE** window.

Now we can re-implement our design by highlighting the leddcd object in the Sources pane and double-clicking on the Implement Design process.

Xilinx - Project Navigator - C:\tmp\fpga_designs\	design1\design1.npl - [leddcd.vhd]	
File Edit View Project Source Process Window	Help	» _ B ×
		• 14
Sources in Project: Carlot design1 Carlot Carlot Carlot (leddcd.vhd) Carlot Carlot	<pre>11 entity leddcd is 12 Port (d : in std_logic_vector(3 dow 13 s : out std_logic_vector(6 do 14 end leddcd; 16 16 architecture Behavioral of leddcd is 17 18 begin 19</pre>	vnto 0);
Module View Snapshot View Library View Processes for Source: "leddod-behavioral" View RTL Schematic C C Check Syntax C C Check Syntax Place & Route Place & Route Place & Route Place & Route Report Asynchronous Delay Report	20 s <="1110111" when d="0000" else	
Process View	leddcd.vhd	
Place & Route Module leddcd . PAR command line: par -w -intsty PAR completed successfully Console Find in Files Warnings	le ise -ol std -t 1 leddcd_map.ncd leddcd.ncd l λ Emons /	eddcd.pcf
Hierarchy is up to date.	Ln 35 Col 17	

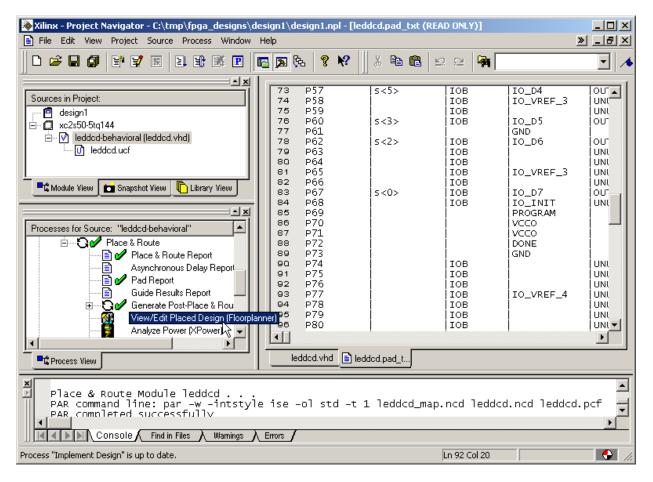
Next double-click on Pad Report to view the pin assignments made during the implementation process. Now the pad report contains the following pin assignments:

Pin Number	Signal Name	Pin Usage	Direction	IO Standard
P39	s<1>	IOB	OUTPUT	LVTTL
P42	d<2>	IOB	INPUT	LVTTL
P46	s<4>	IOB	OUTPUT	LVTTL
P47	d<3>	IOB	INPUT	LVTTL
P48	d<1>	IOB	INPUT	LVTTL
P49	s<6>	IOB	OUTPUT	LVTTL
P50	d<0>	IOB	INPUT	LVTTL
P57	s<5>	IOB	OUTPUT	LVTTL
P60	s<3>	IOB	OUTPUT	LVTTL
P62	s<2>	IOB	OUTPUT	LVTTL
P67	s<0>	IOB	OUTPUT	LVTTL

The reported pin assignments match the assignments we made in the **Constraints Editor** window so it appears we accomplished what we wanted.

Viewing the Chip

After the implementation process completes, we 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 32 × 48 array of slices in the FPGA. The I/O pins are also shown around the periphery. (The pins used for Vcc, GND, and programming are not shown.)

🔀 Xilinx Floorplanner - leddcd.fnf File Edit View Hierarchy Pattern Floorplan W	odow. Help	<u>_ </u>
· · · · · · · · · · · · · · · · · · ·	\$	
leddcd.fnf Design Hierarchy	leddcd.fnf Editable Floorplan for XC2550-5-TQ144 (UCF Flow)	_ 🗆 🗙
Image: Second	Ieddcd.fnf Placement for XC2550-5-TQ144	
	R16C25	

The CLBs used by the LED decoder circuit are highlighted in light-green and are clustered near

the 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 will appear as shown below.

🐼 Xilinx F	loorplanner - leddcd.fnf								
File Edit	View Hierarchy Pattern Floorplan W	ndow Help							
🗍 🗅 🗃	🗅 🖆 🖬 🚭 🖊 😢 🖶 😚 🗇 약을 🔮 🗶 🕅 🏦 🚂 🔀 Q. Q. X. Q. 🔯								
🖹 leddco	d.fnf Design Hierarchy	📕 leddcd.fnf Editable Floorplan for XC2550-5-TQ144 (UCF Flow)							
٦.	leddcd "leddcd" (11 IOBs, 7 FGs 🗖	leddcd.fnf Placement for XC2550-5-TQ144							
∥⊢₽	Mrom_s_inst_lut4_61 [FG] 0:s								
FK	Mrom_s_inst_lut4_51 [FG] 0:s		P47						
IIFK	Mrom_s_inst_lut4_41 [FG] O:s								
IIFK	Mrom_s_inst_lut4_31 [FG] 0:s								
II EK	Mrom_s_inst_lut4_21 [FG] 0:s Mrom_s_inst_lut4_11 [FG] 0:s								
II LÐ	Mrom_s_inst_lut4_01 [FG] 0:s								
II F	s<6> [10B] 0:s_6_0BUF								
	s<5> [IOB] 0:s_5_0BUF		P49						
	s<4> [IOB] 0:s_4_OBUF		P943						
	s<3> [IOB] 0:s_3_OBUF		P49						
	s<2> [IOB] 0:s_2_OBUF								
	s<1> [IOB] 0:s_1_OBUF		P50						
	s<0> [10B] 0:s_0_0BUF d<3> [10B] 1:d_3_IBUF								
			P51						
s<2>	d.fnf Design Nets		P54						
s<3>	_								
s<4> s<5>									
s<6>			P56						
d<0> d<1>] 4 D D D A D D D A 4 🗔							
d<2>			P57						
d<3>			العي						
		R16C0							

We 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.

Edit Preferences	\ge
Resources Logic Ratsnest	
Floorplan and Placement Views ✓ Display nets connected to selected logic ✓ Direction arrows ✓ Rubberbands ✓ Max Fanout:	
Net View List only nets visible in the Floorplan View Enable Disable	
Close Help	

	oorplanner - leddcd.fnf View Hierarchy Pattern Floorplan W	indow Help	
0 🗃		· · · · · · · · · · · · · · · · · · ·	
🖹 leddcd	l.fnf Design Hierarchy 📃 🔲 🗙	leddcd.fnf Editable Floorplan for XC2550-5-TQ144 (UCF Flow)	_ 🗆 🗵
E.	leddcd "leddcd" (11 IOBs, 7 FGs 🔺	leddcd.fnf Placement for XC2550-5-TQ144	
I KAL	Mrom_s_inst_lut4_61 [FG] 0:s		
	Mrom_s_inst_lut4_51 [FG] 0:s		P42
	Mrom_s_inst_lut4_41 [FG] 0:s	40000400004 4 🚬 🖓	1.44
IIFK	Mrom_s_inst_lut4_31 [FG] 0:s		P43
II EK	Mrom_s_inst_lut4_21 [FG] 0:s		P44
II Es	Mrom_s_inst_lut4_11 [FG] 0:s Mrom_s_inst_lut4_01 [FG] 0:s		P46
$\parallel \perp^r$	s<6> [10B] 0:s_6_0BUF		
	s<5> [10B] 0:s_5_0BUF		P47
	s<4> [IOB] 0:s_4_OBUF		
	s<3> [IOB] 0:s_3_OBUF		
	\$<2> [IOB] 0:s_2_OBUF		
	s<1> [IOB] 0:s_1_OBUF		
	s<0> [IOB] 0:s_0_0BUF		
	d<3>[IOB]1:d_3_IBUF		P49
🖹 leddcd	l.fnf Design Nets		P49
s<2>			
s<3> s<4>			P50
s<5>			
s<6> d<0>			
d<1>			P51
d<2>	-		P54
		K10C24.0	

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

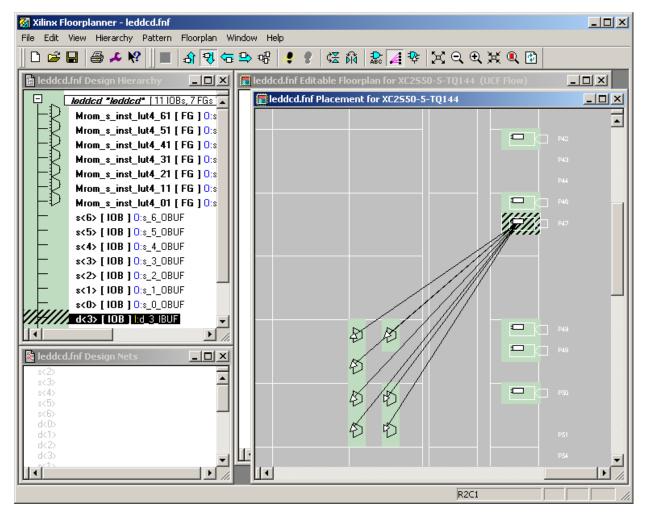
In an analogous manner, we can click on an input pin to highlight which CLBs are dependent on that input.

🐼 Xilinx Floorplanner - leddcd.fnf		- O ×
File Edit View Hierarchy Pattern Floorplan	Window Help	
📙 🗅 😅 🔚 🚭 🥕 😢 🗍 🔳 😂 🔇	5♀\$\$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$	
leddcd.fnf Design Hierarchy	leddcd.fnf Editable Floorplan for XC2550-5-TQ144 (UCF Flow)	
📙 🖻 🚬 🗆 Ieddicd " leddicd" [11 IOBs, 7 FGs 🗖	leddcd.fnf Placement for XC2550-5-TQ144	
Mrom_s_inst_lut4_61 [FG] 0:s		
Mrom_s_inst_lut4_51 [FG] 0:s		P42
Mrom_s_inst_lut4_41 [FG] 0:s	4 50 50 50 4 💻 🖓	
Mrom_s_inst_lut4_31 [FG] 0:s		P43
Mrom_s_inst_lut4_21 [FG] 0:s		P44
Mrom_s_inst_lut4_11 [FG] 0:s		P46
Mrom_s_inst_lut4_01 [FG] 0:s s<6> [IOB] 0:s_6_0BUF		
s(5) [10B] 0:s 5 OBUF		P47
s<4>[10B]]0:s_4_0BUF		
s<3> [10B] 0:s_3_0BUF		
s<2> [IOB] 0:s_2_OBUF		
s<1> [IOB] 0:s_1_OBUF		
s<0> [10B] 0:s_0_0BUF		
d<3> [10B] [:d_3 [BUF		P48
🖹 leddcd.fnf Design Nets		P49
\$<2>		
s<3> s<4>		P50
\$<5>		
s<6> d<0>		
d(1)		P51
d<2> d<3>		P54
p		
	pro section and a section of the sec	

-

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.

Edit Preferences	×
Resources Logic Ratsnest	
Floorplan and Placement Views Function generators and RAMs Flip flops and Latches Tristate buffers I/O pads and Global buffers IGrid	
Package Pin View Top View O Bottom View G Bottom View floorplanned logic	
Close Help	

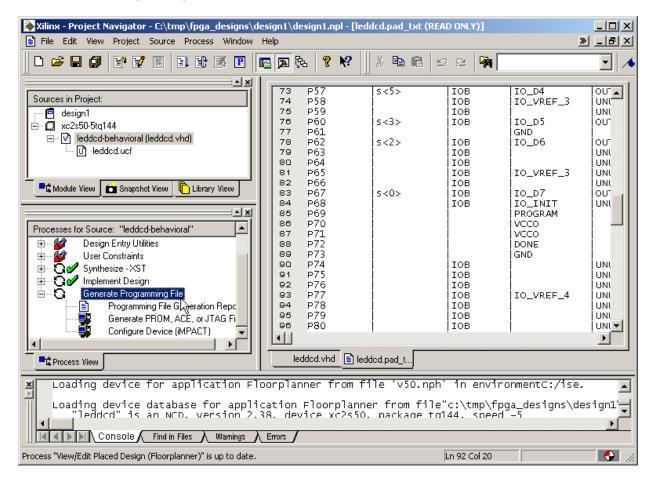


Now the **Placement** pane shows only the LUTs and I/O pins that are used in this design.

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. We won't do this here, but it is an option for designs which push at the extremes of the capabilities of FPGAs.

Generating the Bitstream

Now that we have synthesized our design and mapped it to the FPGA with the correct pin assignments, we are ready to generate the bitstream that is used to program the actual chip. To initiate the programmer, we highlight the leddcd object in the Sources pane and double-click on the Generate Programming File process.

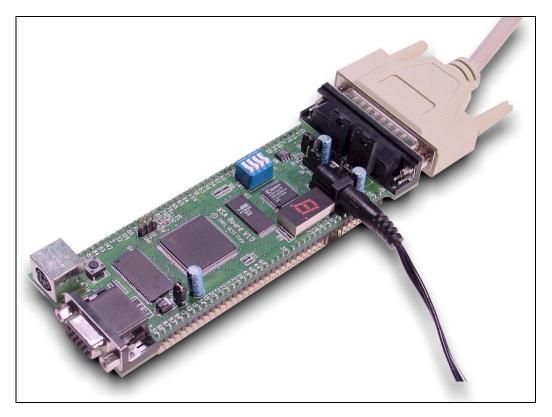


Within a few seconds, a 🖋 will appear next to the Generate Programming File process and a file detailing the bitstream generation process will be created. A bitstream file named leddcd.bit can now be found in the design1 folder.

Xilinx - Project Navigator - C:\tmp\fpga_designs\delta		gn1.npl - [le	ddcd.pad_txt (Rf	AD ONLY)]	1	
	- 	१ №	X 🖻 💼	n n 🙀 🗌		•
Sources in Project: design1 	74 F 75 F 76 F 77 F 78 F 79 F 80 F 81 F 81 F	257 258 259 260 261 262 263 264 263 264 265 266	\$<5> \$<3> \$<2>	IOB IOB IOB IOB IOB IOB IOB IOB IOB	IO_D4 IO_VREF_3 IO_D5 GND IO_D6 IO_VREF_3	
	84 F 86 F 87 F 88 F 90 F 91 F 92 F 93 F 94 F 95 F 96 F	967 968 970 971 972 973 974 975 977 977 977 977 977 977 977 978 979 980	s<0>	IOB IOB IOB IOB IOB IOB IOB IOB IOB	IO_D7 IO_INIT PROGRAM VCCO VCCO DONE GND IO_VREF_4	
Process View		cd.vhd 📄 le	ddcd.pad_t			
Started process "Generate Program Completed process "Generate Progra Console Find in Files Warmings	-					×
Process "Generate Programming File" is up to date.				Ln 92 Col 20		- 🔶 /i.

Downloading the Bitstream

Now we have to get the bitstream file programmed into the FPGA on the XSA Board. The XSA Board is powered with a 9 VDC power supply and is attached to the PC parallel port with a standard 25-wire cable as shown below.





The XSA-50 Board is programmed using the gxsload utility. We double click the GX5LOAD icon to bring up the **gxsload** window:

<mark>X</mark> gxsload		_ 🗆 🗙
	8-300E 💌	Load Exit
FPGA/CPLD	RAM	Flash/EEPROM
High Address		
Low Address		
Upload Format	HEX 💌 🗋	HEX 💌 🗋

Then we click in the Board Type field and select XSA-50 from the drop-down menu since this is the board we are going to load with the bitstream.

<mark>X</mark> gxsload			<u> </u>
Board Type	XSA-50	•	Load
Port	XSA-50 XSA-100	×	Exit
FPGA/CPLI	XSB-300E XS95-108 XS95-108+ XS40-005XL XS40-005XL+ XS40-005E XS40-005E+ XS40-010XL		Flash/EEPROM
	XS40-010XL+ XS40-010E		
Low Addr	XS40-010E+ XSP-010		
Upload For	XSP-010+ XSV-50 XSV-100		HEX 💌 🗀

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

C:\tmp\fpga_designs\design1	🔀 gxsload
File Edit View Favorites Tools H Tools $\leftrightarrow \rightarrow \rightarrow \rightarrow \bullet$ \Box_{\bullet}	Board Type XSA-50 Coad Port LPT1 Exit
projnav design1.npl _ngo leddcd.bgn xst leddcd.bit .untf leddcd.bid _projnav.log leddcd.cmd_log leddcd.drc leddcd.drc leddcd.lfp design1.dhp leddcd.lso Type: BIT 68.3 KB My Computer	FPGA/CPLD RAM Flash/EEPROM Image: Construction of the state of the

Then we click on the Load button to initiate the programming of the FPGA. Downloading the leddcd.bit file to the XSA-50 takes only a few seconds.

🔀 gxsload		
	450 ▼ T1 ▼	Load Exit
FPGA/CPLD	BAM	Flash/EEPROM
leddcd.bit		
High Address		
Low Address		
Upload Format	XESS-3 🔻 🗀	HEX 💌 🗀

Testing the Circuit

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



Double-clicking the GXSPORT 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.

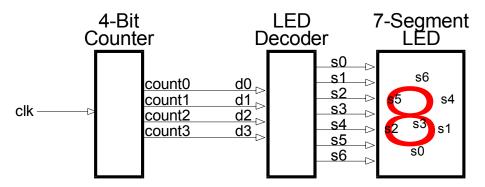
NOTE: Bit D7 of the parallel port controls the /PROGRAM pin of the FPGA. Do not set D7 to 0 or you will erase the configuration of the FPGA. Then you will have to download the bitstream again to continue testing your design.



Hierarchical Design

A Displayable Counter

We went through a lot of work for our first FPGA design, so we will reuse it in this design: a four-bit counter whose value is displayed on a seven-segment display. The counter will increment on a rising 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

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

New Project		×
Enter a Name and Location for the Pr	oject	
Project Name: design2	Project Location: C:\tmp\fpga_designs\design2	
Select the type of Top-Level module f	for the Project]
Top-Level Module Type: HDL	_	
		J
		_
	< Back Next > Cancel Help	

We are still targeting the same FPGA on the XSA-50 Board, so the other properties in the **New Project** window retain the same values we set in the previous project.

Device Family		Spartan2
Device		xc2s50
Package		tg144
Speed Grade		-5
Top-Level Module Type		HDL
Synthesis Tool		XST (VHDL/Verilog)
Simulator		Other
Generated Simulation Lang	guage	VHDL

We won't add any new source files at the moment, so just click on the Next button.

Create	e a New Source		
1	Source File	Туре	New Source Remove
Create	e a new source to add t onal new sources can b	to the project (optional). Only one new source be added after project creation using the "Proj	can be specified now. ect->New Source''
Additi			
Additi comm			

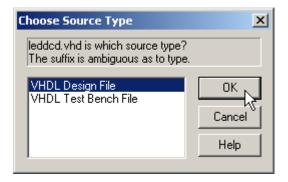
The next window allows us to add existing source files to our new project. We want to re-use the LED decoder from our previous design, so click on the Add Source... button.

	Source File	Туре	Copy to Projec Add Source	_ 1
1				5
3			Remove	e'
4			_	
exist	ing sources to the proje	ect (optional). Addit d Source'' or ''Proir	ional sources can be added after project	
exist ation	ing sources to the proje using the "Project->Ad	ect (optional). Addit d Source'' or ''Proje	ional sources can be added after project act->Add Copy of Source'' commands.	
exist ation	ing sources to the proje using the "Project->Ad	ect (optional). Addit d Source'' or ''Proje	ional sources can be added after project ect->Add Copy of Source'' commands.	

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

Add Existing	Sources	? ×
Look in: 🔁) design1 💽 🗢 🛍 📺 -	
projnav	🗏 leddcd_pad.txt	
ngo		
kst leddcd.ngo	c.	
leddcd.ucf		
🖹 leddcd.vho	d	
, File name:	leddcd.vhd Open	
Files of type:	Sources (*.txt;*.vhd;*.vhd;*.v;*.abl;*.xco;*.sc Cance	Ľ)

After clicking on Open, a window appears that asks us the type of file we are adding to the project. Select VHDL Design File and click the OK button.



The **New Project** window now shows that a copy of the leddcd.vhd file will be added to the project. This is the only existing file we want to add, so click on the Next button to move on to the next window.

ew Proje	ect Existing Sources			
1 2 3 4	Source File leddcd.vhd	Type VHDL Design File	Copy to Projec	Add Source Remove
		roject (optional). Additiona Add Source'' or ''Project-)		

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

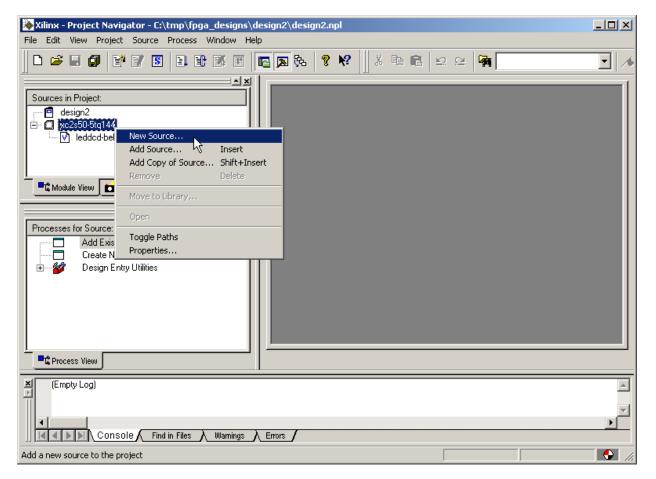
New Project Information
Project Navigator will create a new Project with the following specifications:
Project: Project Name: design2 Project Location: C:\tmp\fpga_designs\design2 Project Type: HDL Device: Device Family: Spartan2 Device: xc2s50 Package: tq144 Speed Grade: -5 : Top-Level Module Type: HDL Synthesis Tool: XST (VHDL/Verilog) Simulator: Other Generated Simulation Language: VHDL Sources: VHDL Design File leddcd.vhd copied to Project
< Back Finish Cancel Help

Once we click on Finish in the **New Project** window, the **Project Navigator** window appears as shown below. The project currently contains only the single leddcd.vhd source file.

Xilinx - Project Navigator - C:\tmp\fpga_designs\design2\design2.npl	<u>- I X</u>
File Edit View Project Source Process Window Help	
	2 2 🛉 🔽 🔹 🖈
Sources in Project: design2 xc2s50-5tq144 Module View Snapshot View Library View	
(Empty Log)	× ×
Hierarchy is up to date.	

Adding a Counter

Now we have to add the counter to our design. We don't have a counter module yet, so we have to build one with VHDL. Right-click on the XC2S50-5TQ144 object and select New Source... from the pop-up menu.



As in the previous example, we are prompted for the type of file we want to add to the project. Once again, we select the VHDL Module menu item. Then we type <code>counter</code> into the File Name field and click on the Next button.

×
File Name: counter Location: c:\tmp\fpga_designs\design2
Cancel Help

Then we declare the inputs and outputs for the counter in the **Define VHDL Source** window as shown below. The **counter** module receives a single input, **clk**, and has a four-bit output bus, **count**, which outputs the current counter value.

Define ¥HDL Source				×
Entity Name coun	ter			
Architecture Name Beha	vioral			_
	inordi			
Port Name	Direction	MSB	LSB	
clk	in			
count	out	3	0	
	in			
	in	ļ	ļ	-
< B	ack Next >	Canc	el Help	

Click on Next and check the information about the module.

Nev	• Source Information	×
	Project Navigator will create a new skeleton source with the following specifications:	
) E A	iource Type: VHDL Module iource Name: counter.vhd intity Name: counter wrchitecture Name: Behavioral ont Definitions:	Ă
	clk scalar in count vector: 3:0 out	
	C Source Directory: c:\tmp\fpga_designs\design2	▼ ▶
_	< Back Finish Cancel	Help

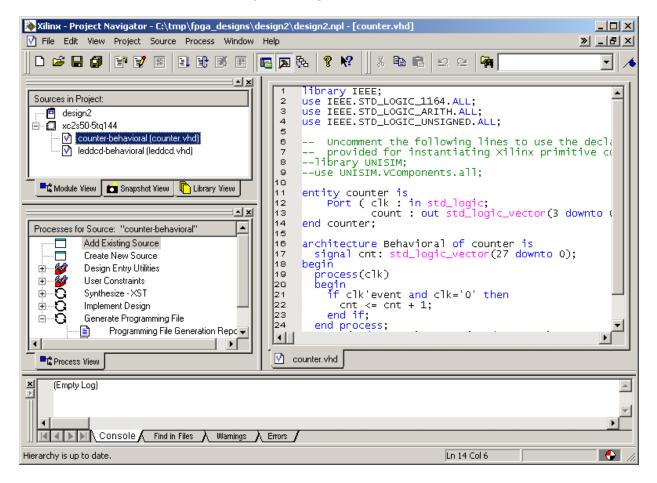
After clicking Finish in the **New Source Information** window, we are presented with a VHDL skeleton for the counter. We flesh-out the skeleton as follows:

🔚 counter.vhd	<u>»</u>	-D×
File Edit		
🔒 🏅 🖻 🛍 🗠 🗠 🖊	1 × % % %	
<pre>7 provided for in: 8library UNISIM; 9use UNISIM.VCompoin 10 11⇒ Entity counter is 12 Port (clk : in 13 count : in 14 end counter; 15 16 architecture Behavio 17 signal cnt: std_1</pre>	ARITH.ALL; UNSIGNED.ALL; ollowing lines to use the declarations that stantiating Xilinx primitive components. nents.all; std_logic; out std_logic_vector(3 downto 0));	are
<pre>18 begin 19 process(clk) 20 begin 21 if clk'event and 22 cnt <= cnt + 1 23 end if; 24 end process; 25 count(3 downto 0) 26 end Behavioral; 27</pre>	d clk='0' then l; <= cnt(27 downto 24);	×
For Help, press F1	∏ ī	. //.

Line 17 declares a 28-bit signal, **cnt**, that is the current value of the counter. The process on lines 19-24 controls when the counter increments. The condition clause of line 21 is only true when the value on the **clk** input goes from 1 to 0. Then the statement on line 22 replaces the value in **cnt** with its incremented value. (We can use the high-level addition operator instead of having to describe a 28-bit adder because on line 4 we have linked into the ieee.std_logic_unsigned.all package that supports unsigned arithmetic.) Finally, line 25 places the upper four bits of the current counter value onto the outputs of the module.

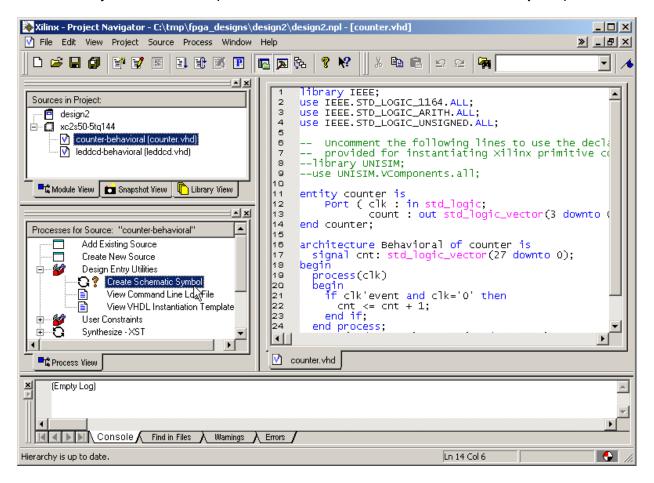
Why are we building a 28-bit counter and using only the upper four bits? The counter will be driven by the oscillator on the XSA-50 Board which has a default frequency of 50 MHz. The LED display would be changing much too quickly to see at this frequency. By connecting the LED decoder to the upper four 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 10⁶) = 0.336 seconds which is slow enough to be read.

After entering the VHDL shown above and saving it, we see that the counter module has been added to the Sources pane of the **Project Navigator** window.

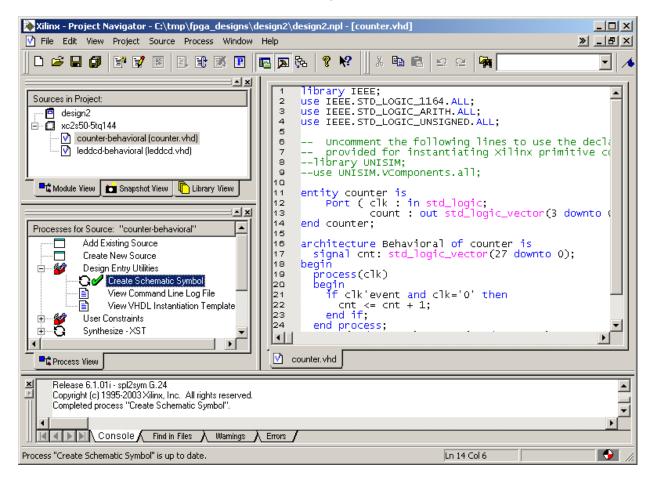


Tying Them Together

We have the LED decoder and the counter, but now we need to tie them together to build the displayable counter. We will do this by connecting the counter to the LED decoder in a top-level schematic. Before we can do this, we have to create schematic symbols for both the counter and LED decoder VHDL modules. To create the counter schematic symbol, highlight the counter object in the Sources pane and then double-click the Create Schematic Symbol process.



A 🖋 will appear next to the Create Schematic Symbol process after the counter symbol is created. Repeat this procedure to create the schematic symbol for the LED decoder.



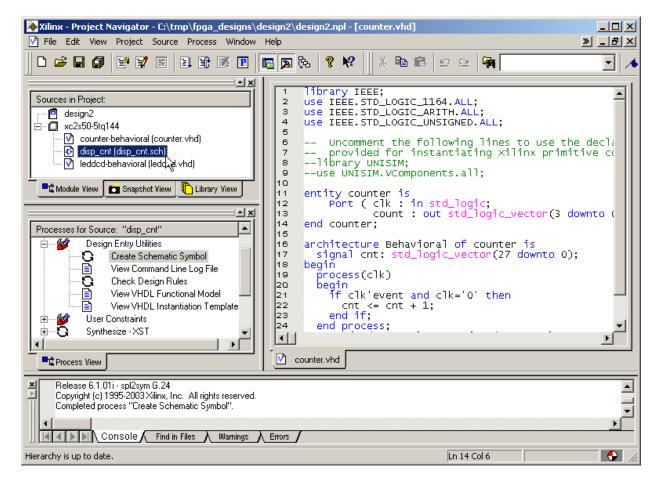
Once the schematic symbols for the lower-level modules are built, we can add the top-level schematic to the project. Right-click on the XC2S50-5TQ144 object and select New Source... from the pop-up menu. Then highlight the Schematic entry in the **New Source** window and name the schematic **disp_cnt**. Then click on Next.

New Source	X
 BMM File Implementation Constraints File MEM File Schematic State Diagram Test Bench Waveform User Document Verilog Module Verilog Test Fixture VHDL Library VHDL Library VHDL Package VHDL Test Bench 	File Name: disp_cnt Location: c:\tmp\fpga_designs\design2
< Back Next >	Cancel Help

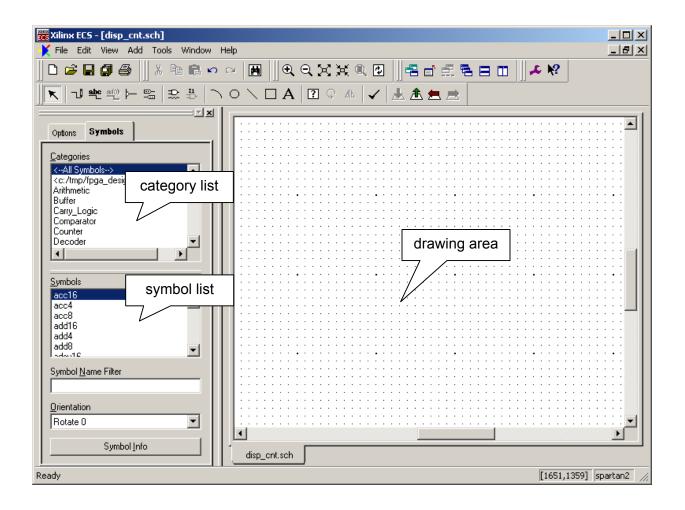
There is very little to do when setting-up a schematic, so just click on the Finish button in the **New Source Information** window that appears.

New Sour	ce Inform	nation				×
	Navigator g specifica	will create a r tions:	iew skeleto	in source w	ith the	
	lype: Sche Name: disp					4
						•
•						►
Sourc	e Directory	: c:\tmp\fpg	a_designs\	design2		
		< Back	Fini	ish 💦 🗌	Cancel	Help

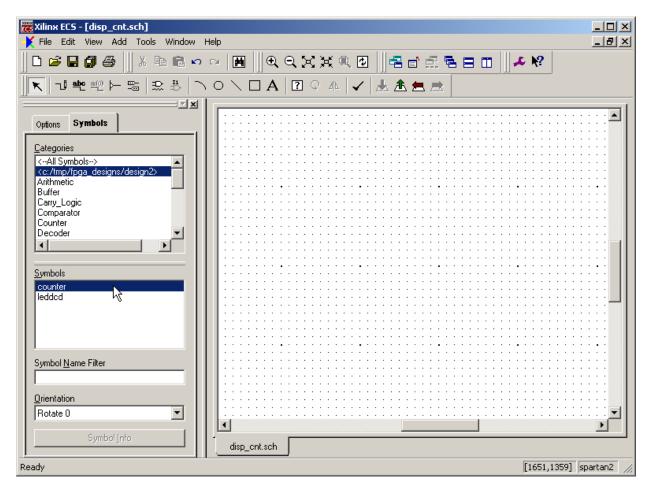
Now the disp_cnt schematic object has been added to the Sources pane. We can double-click it to begin creating the schematic, but a schematic editor window should open automatically once the file is created.



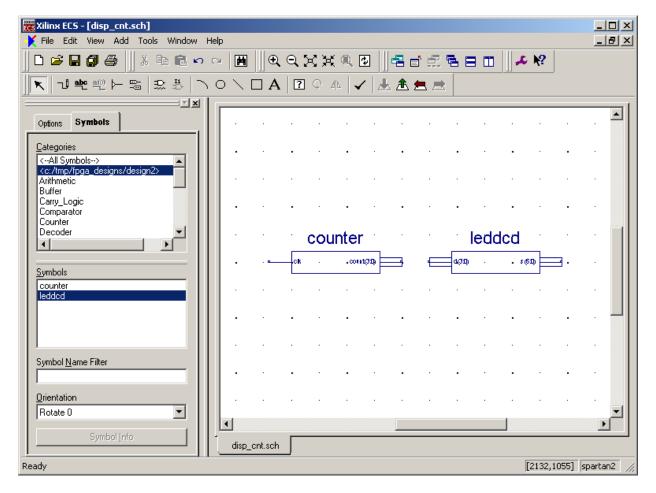
The schematic editor window has a drawing area and a list of categories for various logic circuit elements that can be used in a schematic. Below that is the list of symbols for circuit elements in a highlighted category.

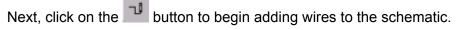


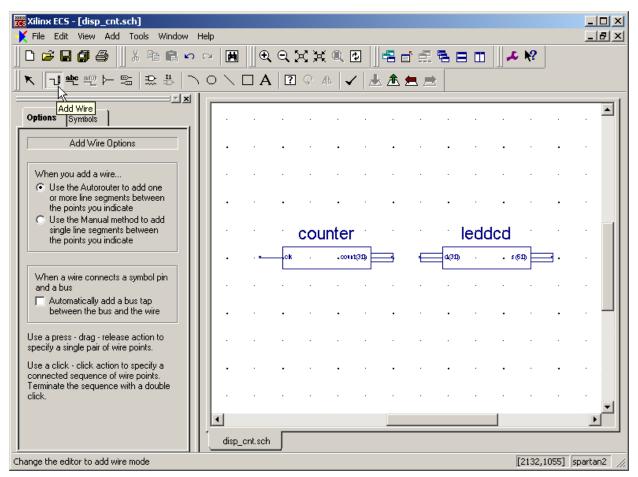
To start creating the top-level schematic, highlight the second entry in the category list. The c:/tmp/fpga_designs/design2 category contains the schematic symbols for the *design2* project's counter and LED decoder modules. We can see the names of these modules in the symbol list.



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.



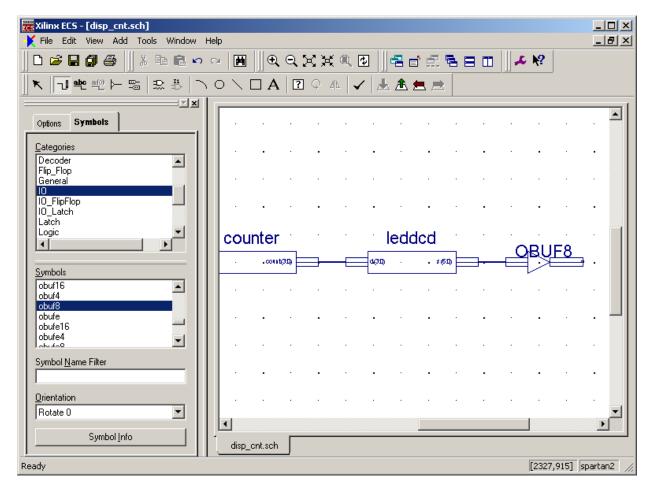




Left-click the mouse on the **count(3:0)** bus on the right-hand edge of the **counter** module. Then left-click on the **d(3:0)** bus on the left-hand edge of the **leddcd** module. As a result of this procedure, a four-bit bus is created between the output of the counter module and the input of the LED decoder module.

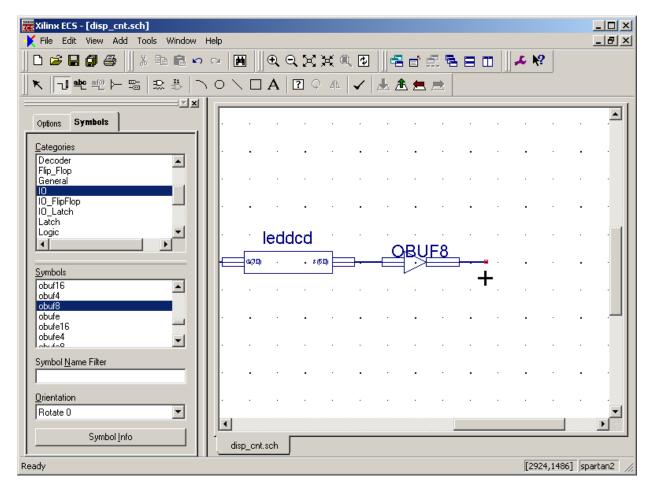
Xilinx ECS - [disp_cnt.sch] File Edit View Add Tools Window H	elo						_ D ×
	• 	9.XX	(-2 of E.	580	≁ №	
11	$\circ \setminus \Box A$. ? Ç 4	▲ 🖌 🛃	k 🏝 💻 🖻	2		
Options Symbols							
Add Wire Options	· · ·	•	•	•	•	•	
When you add a wire							
 Use the Autorouter to add one or more line segments between the points you indicate 	• •	•	•	•	•	•	•
 Use the Manual method to add single line segments between the points you indicate 		cou	inter		ledo	lcd	· · ·
	• • •	ck .	.0011t3D		43D) -	. \$6D	а
When a wire connects a symbol pin and a bus				. +			
Automatically add a bus tap between the bus and the wire		•	•	•	•	•	–
Use a press - drag - release action to specify a single pair of wire points.	· · ·						
Use a click - click action to specify a connected sequence of wire points.	• · ·	•	•	•	•	•	•
Terminate the sequence with a double click.	· ·						
	•						
	disp_cnt.sch						
Ready						[2180,15	514] spartan2 //

Now highlight the IO category and select a byte-wide output buffer (OBUF8) from the list of symbols. Attach the output buffer to the output of the LED decoder as shown below.

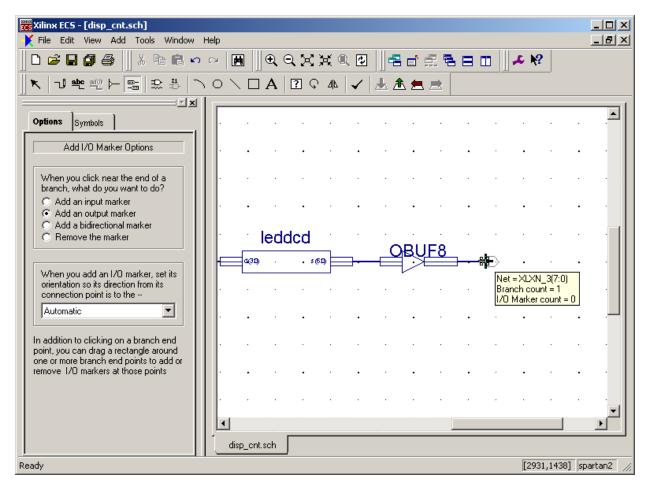


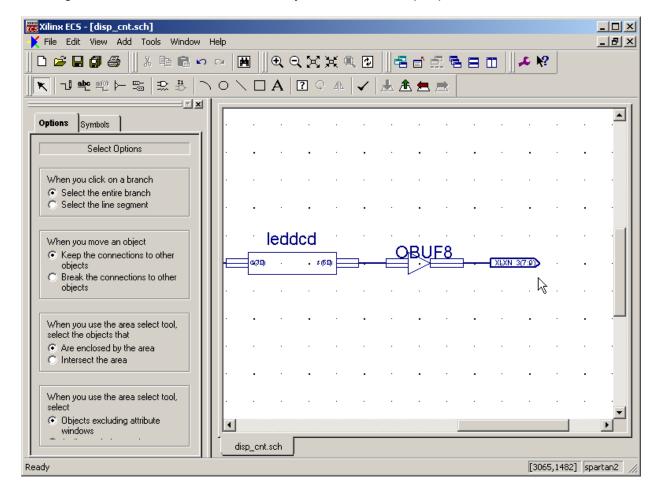
Next attach a short bus segment to the output of the byte-wide buffer. We do this by selecting

the wiring tool, clicking on the output of the byte-wide buffer, and then moving the cursor slightly to the right and double-clicking to create a stub.



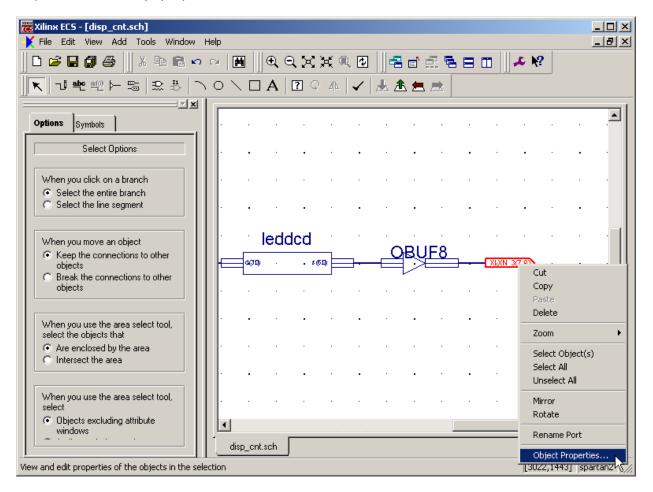
Now click on the button for adding I/O markers. Click on the Add an output marker button in the Options tab and then click on the free end of the wire segment that we just added.





Clicking on the end of the wire creates a byte-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.

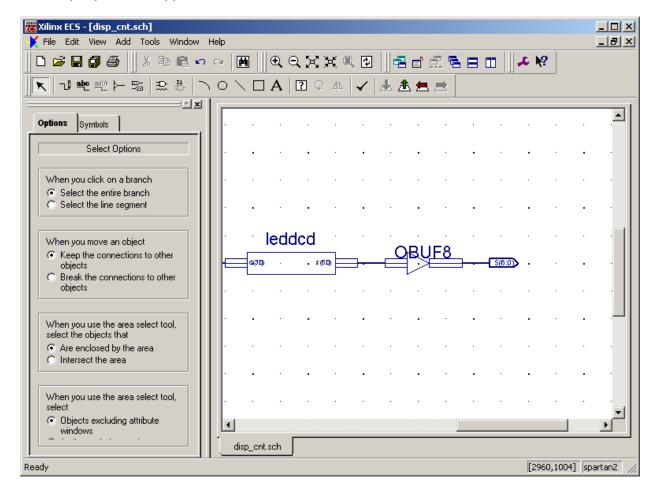


The **Object Properties** window allows us to set the name and direction of the pins.

📲 Object Properties				×
Category	Viev	Net Attri w and edit the attribut		ed nets
XLXN_3(7:0)	Name	Value	Visible	New
	Name	XLXN_3(7:0)	Add	
	PortPolarity	Output	Add	Edit Traits
				Delete
	ОК	Cancel	Apply	Help

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 we can finish by clicking on the OK button.

Content in the second s				×
Category				
	Vieu	Net Attri w and edit the attribut		ad nate
⊡ I/O Markers	viev	wand edit the attribut	es of the select	
LXN_3(7:0)	Name	Value	Visible	New
	Name	S(6:0)	Add	
	PortPolarity	Output	Add	Edit Traits
				Delete
1				
	ОК	Cancel	Apply	Help



The output pins now appear with their new name, width, and direction.

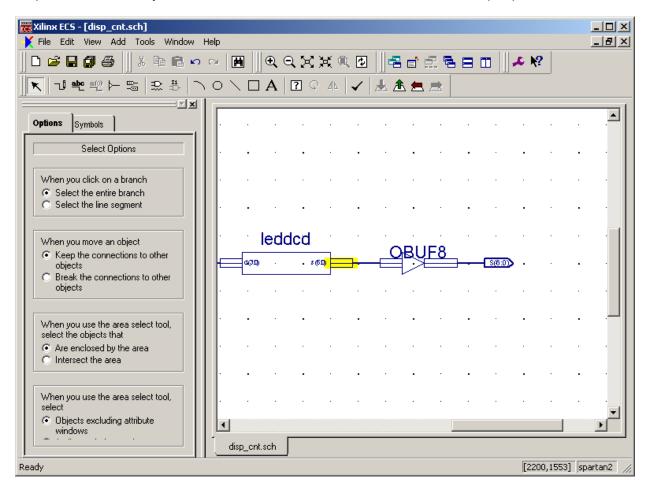
At this point it makes sense to check the schematic to see if there are any errors such as unterminated wire stubs or mismatched bus widths. Click on the button to perform a schematic check.

Xilinx ECS - [disp_cnt.sch] Image: State of the state of	telp	_ D ×
□ ☞ 묘 @ ●		
There is a second secon		
Options Symbols	Check Schematic	
Select Options		
When you click on a branch Select the entire branch		
C Select the line segment		
When you move an object	OBUE8	· · ·
 Keep the connections to other objects Break the connections to other objects 		•
When you use the area select tool, select the objects that		
Are enclosed by the area Intersect the area		· ·
When you use the area select tool, select		
Objects excluding attribute windows	disp_cnt.sch	
Check the active schematic		004] spartan2 //

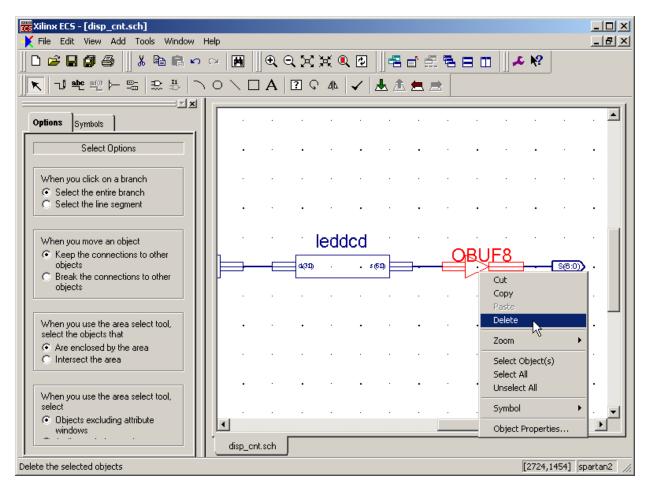
The **Schematic Check Errors** window will appear showing two errors. We can find the place in the schematic where the error occurs by clicking on the associated error message. Then click on the Zoom In button to see an enlarged view of the area where the error lies.

Schematic	×	
Error No.	Error Msg	Center
1	Error: Pin 's(6:0)' is connected to a bus of a different width	Zoom In 🕟
2	Error: Pin '0(7:0)' is connected to a bus of a different width	
		Zoom Out
		Close
		Help

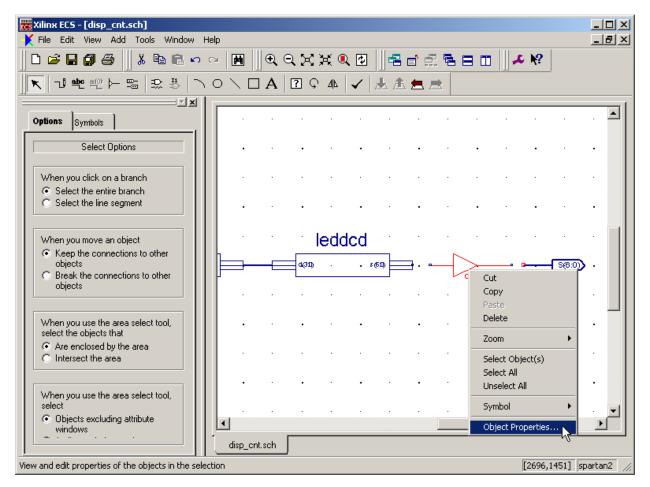
The first error indicates that the seven-bit output of the LED decoder does not match with the byte-wide input of the output buffer symbol. Note how the output of the leddcd symbol is highlighted to indicate the error. The second error is similar to the first in that the byte-wide output of the OBUF8 symbol does not match the width of the seven-bit output pin marker.



The simplest way remove these errors is to replace the byte-wide output buffer with a seven-bit wide version. To remove the byte-wide buffer, right-click on the OBUF8 symbol and select delete from the pop-up menu. Do the same for the bus that connected to the input of the byte-wide buffer.



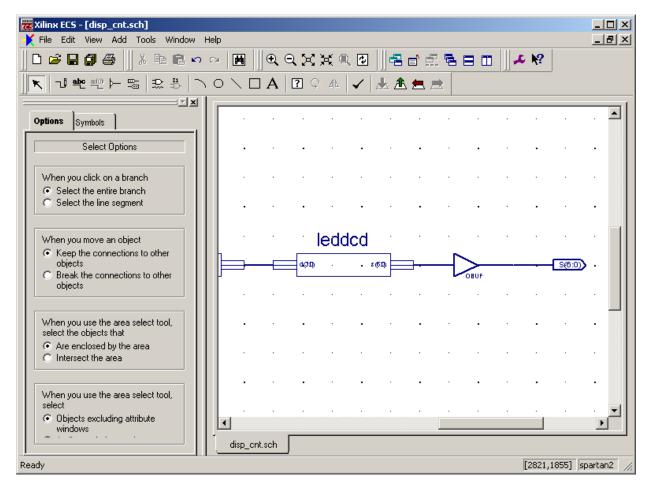
Now add a single OBUF symbol to the schematic. Then right-click on it and select Object Properties... from the pop-up menu.



In the **Object Properties** window, highlight the instance name for the buffer and change it to mybuf(6:0) and then click on the OK button. This will change the single-bit output buffer to an array of seven output buffers.

👷 Object Properties				×
Category				
Instances	View ar	Instance A nd edit the attributes		instances
	Name	Value	Visible	New
	InstName	my_buf(6:0)		
	SymbollName	obuf		Edit Traits
	Level	XILINX		Delete
	Libver	2.0.0		
	VeriModel	OBUF		Symbol Info
	VhdiModel	OBUF		
	ок	Cancel	Apply	Help

After changing the width of the output buffer, reconnect it to the LED decoder and the output terminals as shown below.

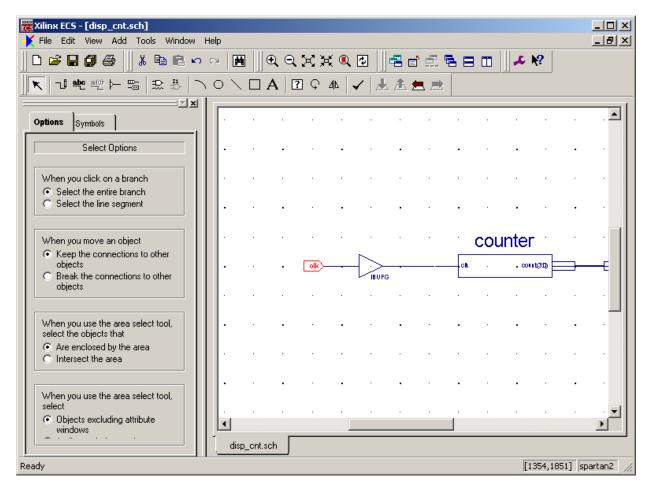


Now when we click on the schematic check button, \checkmark , we see the errors have been corrected.

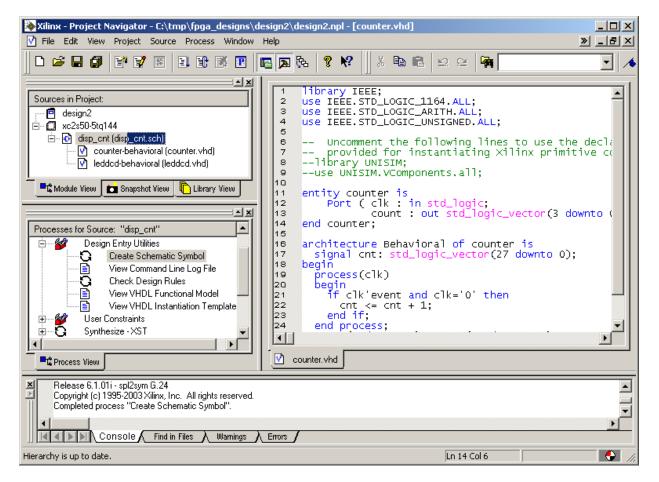
3	Schematic	Check Errors	×
	Error No.	Error Msg	Center
	1	No errors detected	
			Zoom In
			Zoom Out
			Close
			Help
1			

Once the outputs from the circuit are in place, we can create the analogous circuitry for the input. We connect a single, low-skew input buffer module to the clock input of the counter and then we connect a single input I/O marker to the IBUFG symbol. Right-click on the I/O marker

and rename it to clk. After this, perform another schematic check to detect any errors, save the schematic using the File→Save command and then close the schematic editor.

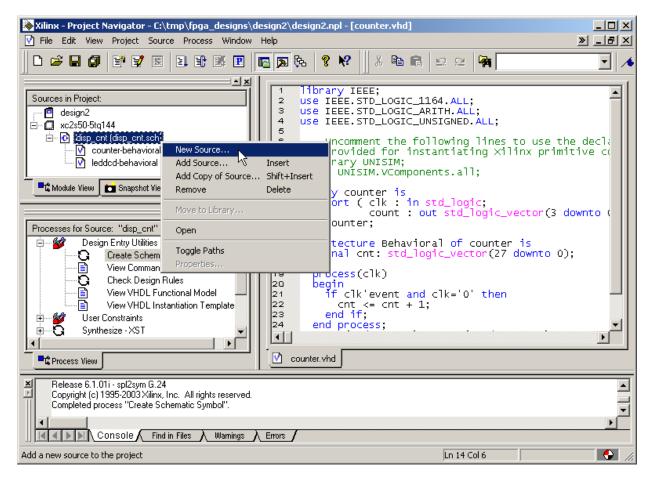


Once we save the schematic for the top-level module, we see the updated hierarchy in the Sources pane of the **Project Navigator** window. Now the **counter** and **leddcd** modules are shown as lower-level modules that are included within the top-level **disp_cnt** module.



Constraining the Design

Before synthesizing the displayable counter, we need to assign the pins which the inputs and outputs will use. We 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 we want to add and type $disp_cnt$ in the File Name field. Then click on the Next button.

New Source	×			
 BMM File Implementation Constraints File MEM File Schematic State Diagram Test Bench Waveform User Document Verilog Module Verilog Test Fixture VHDL Library VHDL Module VHDL Package VHDL Test Bench 	File Name: disp_cnt Location: c:\tmp\fpga_designs\design2			
< Back Next > Cancel Help				

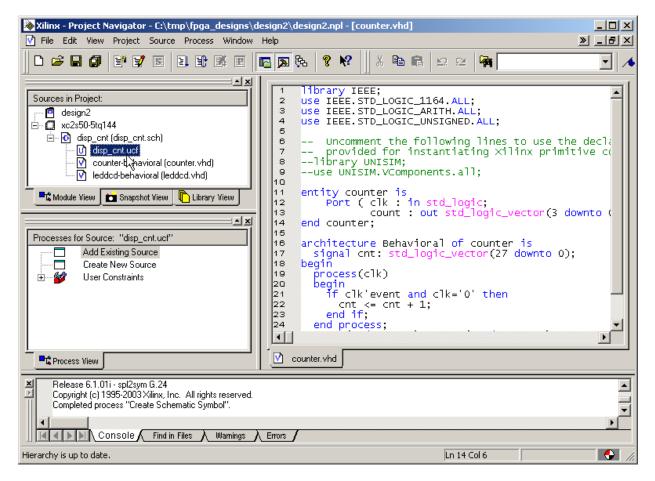
Then we are asked to pick the file with which to associate the constraints. The pin assignments will be made for the top-level module in the design hierarchy, so highlight the disp_cnt item in the list. Then click on the Next button and proceed.

Select	×
Source File	
leddcd	
counter disp_cnt	
· · · · · · · · · · · · · · · · · · ·	
< Back Next > Cancel	Help
<i>vv</i>	

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.

New Source Information	×
Project Navigator will create a new sł following specifications:	eleton source with the
Source Type: Implementation Constra Source Name: disp_cnt.ucf Association: disp_cnt	ints File
	▼ ▼
Source Directory: c:\tmp\fpga_des	igns\design2
< Back	Finish Cancel Help

Now double-click the disp_cnt.ucf object in the Sources pane to begin adding pin assignments to the design.



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 follows:

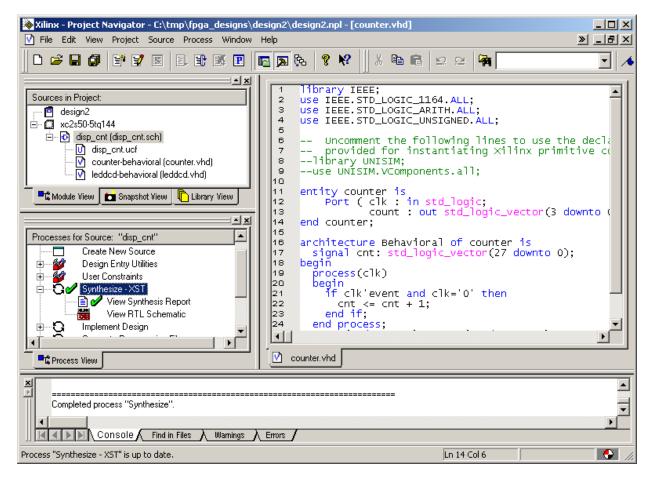
🗱 Xilinx PACE - c:\tmp\fpga_designs	\design2\disp_cnt.ucf	_ 🗆 🗵
File Edit View IOBs Areas Tools V	Window Help	
] 🗅 😂 🔜 🎒 🗠 🗰 🦊 🕨	《]]] [] [] [] [] [] [] [] [] [] [] [] []	\ 🕑 □
😫 Design Browser	Device Architecture for XC2550-5-TQ144	
Global Logic		-
Logic		
🖹 Design Object List - I/O Pins		
1/O Name 1/O Direction Loc	Bank I/O Std. Vref	
clk Input P88	BANK	
S<0> Output P67	BANK	
S<1> Output P39	BANK	
S<2> Output P62	BANK	
S<3> Output P60	BANK	
S<4> Output P46	BANK	
S<5> Output P57	BANK	
S<6> Output P49	BANK	
# Group 1/O Direction Loc	I/O Std. Vref Vcco D	
居 7 S Output		
	Package View Architecture View	
For Help, press F1		

Assigning the **clk** input to pin P88 lets us use the onboard oscillator of the XSA-50 Board to drive the counter. The output assignments connect the displayable counter to the seven-segment LED on the XSA-50 Board as in the previous design example.

After the pin assignments are entered, click on the button to save the pin assignment constraints. Then select File > Exit to close the **Xilinx PACE** window.

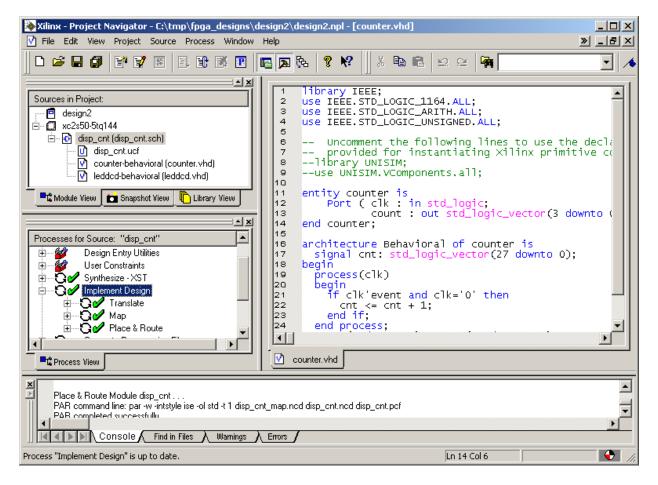
Synthesizing the Logic Circuitry for the Design

Now we can synthesize the logic circuit netlist by highlighting the top-level **disp_cnt** module in the Sources pane and double-clicking the Synthesize process. There should be no problems synthesizing the netlist from the combined VHDL and schematic files.



Implementing the Logic Circuitry in the FPGA

Once the netlist is synthesized, we can begin the process of translating, mapping and placing & routing it into the FPGA. Highlight the disp_cnt object in the Sources pane and then double-click the Implement Design process. There should be no problems implementing the design in the FPGA.



Checking the Implementation

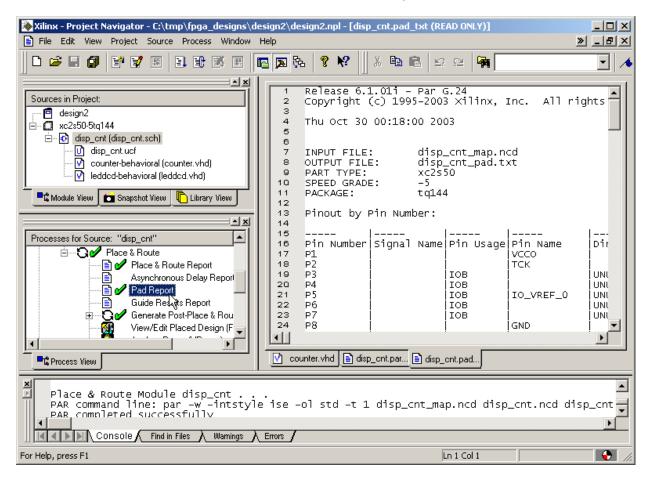
After the implementation process is done, we can check the logic utilization by double-clicking on the Place & Route Report process. Near the top of the file we find:

Device utilization summary:

Number of Number of Number	External	 IOBs	7	out out out	of	92	25% 7% 100%
Number of	SLICEs		18	out	of	768	2%

The displayable counter consumes 18 of the 3072 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.

As a precaution, we should also double-click the Pads Report and check that the pin assignments for the clock input and LED decoder outputs match the assignments we made with PACE:

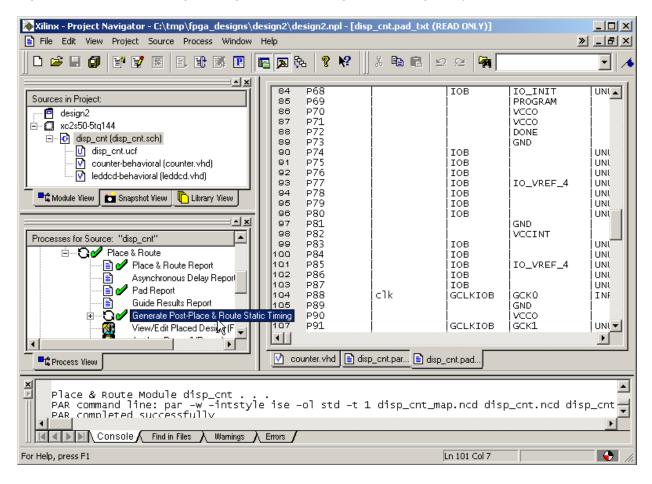


We can see that the pin assignments for the clock input and the LED decoder outputs agree with the constraints we placed in the UCF file.

Pin Number	Signal Name	Pin Usage	Direction	IO Standard
			-	
P39	S<1>	IOB	OUTPUT	LVTTL
P46	S<4>	IOB	OUTPUT	LVTTL
P49	S<6>	IOB	OUTPUT	LVTTL
P57	S<5>	IOB	OUTPUT	LVTTL
P60	S<3>	IOB	OUTPUT	LVTTL
P62	S<2>	IOB	OUTPUT	LVTTL
P67	S<0>	IOB	OUTPUT	LVTTL
P88	clk	GCLKIOB	INPUT	LVTTL

Checking the Timing

We have the displayable counter synthesized and implemented in the XC2S50 FPGA with the correct pin assignments. But how fast can we run the counter? 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.



After the static timing delays are calculated, double-click the Text-based Post-Place & Route Static Timing Report to view the results of the analysis.

Xilinx - Project Navigator - C:\tmp\fpga_designs\de		×□_ ×B_
	🔤 🗛 😵 🕺 🐰 🛍 🛍 🗠 🗠 🙀	•
Sources in Project: design2 xc2s50-5tq144 Counter-behavioral (counter.vhd) leddcd-behavioral (leddcd.vhd) Module View Snapshot View Library View Processes for Source: "disp_ont" Generate Post-Place & Rou Post-Place & Routes Post-Place & Routes Process View	<pre>1 2 Release 6.1.01i Trace G.23 3 Copyright (c) 1995-2003 Xilinx, Inc. All rig 4 5 C:/ise/bin/nt/trce.exe -intstyle ise -e 3 -1 6 disp_cnt.twr disp_cnt.pcf 7 8 9 Design file: disp_cnt.pcf 11 Device, speed: xc2s50, -5 (PRODUCT) 12 Report level: error report 13 14 Environment Variable Effect 15 16 NONE No environment vari 17 18 19 INF0:Timing:2698 - No timing constraints four 20 INF0:Timing:2752 - To get complete path cover 21 ontion. All paths that are not constrained 23 24 1 Counter.vhd disp_cnt.par disp_cnt.pa disp_cnt.twr</pre>	3 — хг сом 1 iable:
PAR completed successfully	le ise -ol std -t 1 disp_cnt_map.ncd disp_cnt.ncd di	sp_cnt
For Help, press F1	Ln 1 Col 1	<i>[i</i> ,

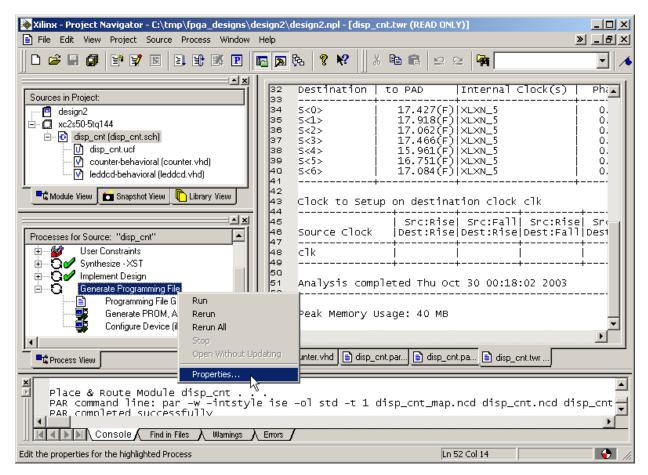
From the information shown in the timing report, we see the minimum clock period for this design is 6.188 ns which means the maximum clock frequency is 161.6 MHz. The default clock frequency on the XSA-50 Board is 50 MHz which is well below the maximum allowable frequency for this design.

Clock to Setup o				LL
Source Clock	Src:Rise Dest:Rise	Src:Fall Dest:Rise	Src:Rise Dest:Fall	Src:Fall Dest:Fall
	-	-	-	6.188

Generating the Bitstream

Now that we have synthesized our design and mapped it to the FPGA with the correct pin assignments, we are ready to generate the bitstream that is used to program the actual chip. In this example, rather than use the gxsload utility we will employ the downloading utilities built into WebPACK. The iMPACT programming tool downloads the bitstream through the JTAG interface of the FPGA so we 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 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.

rocess Properties			>
General Options Configuration Options	Startup Options	Readback Options	
Property Name		Value	
FPGA Start-Up Clock		CCLK	•
Enable Internal Done Pipe		CCLK	
Done (Output Events)		User Clock	
Enable Outputs (Output Events)		JTAG Clock	,
Release Set/Reset (Output Events)		Default (6) h	ζ
Release Write Enable (Output Events)		Default (6)	
Release DLL (Output Events)		Default (NoWait)	
Drive Done Pin High			
OK	Cancel	Default He	elp

After setting the Start-Up Clock option, click on the OK button. Then highlight the disp_cnt object in the Sources pane and double-click on the Generate Programming File process to create the bitstream file.

Xilinx - Project Navigator - C:\tmp\fpga_designs\delta		design2.npl - [disp_	cnt.twr (READ ONL	Y)]	ž	
	R	₿a 🤋 📢 📗 J	K 🖻 🖻 🗠 🤅	2 🙀 📃		•
Sources in Project: design2 	32 33 34 35 36 37 38 39 40 41	Destination S<0> S<1> S<2> S<3> S<4> S<5> S<5> S<6>	to PAD 17.427(F) 17.918(F) 17.062(F) 17.466(F) 15.961(F) 16.751(F) 17.084(F)	XLXN_5 XLXN_5 XLXN_5 XLXN_5 XLXN_5 XLXN_5	=1ock(s)	Ph:▲ 0. 0. 0. 0. 0. 0. 0. 0.
	41 42 43 44 45 46 47	Clock to Setu	Src:Rise	+ Src:Fall	+ Src:Rise	+ Sr Des1
User Constraints User Constraints Synthesize - XST Implement Design Generate Programming File Generate PROM, ACE, or JTAG Fi Configure Device (iMPACT)	48 49 50 51 52 53 54 55	clk Analysis comp Peak Memory (+ t 30 00:18	 + :02 2003	 +
Process View Image: Counter.vhd isp_cnt.par is disp_cnt.par						
Place & Route Module disp_cnt PAR command line: par -w -intstyle ise -ol std -t 1 disp_cnt_map.ncd disp_cnt.ncd disp_cnt PAR completed successfully Console Find in Files Warnings Errors						
For Help, press F1			Ln S	52 Col 14		- •

Within a few seconds, a vill 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 can now be found in the design2 folder.

Xilinx - Project Navigator - C:\tmp\fpga_designs\ File Edit View Project Source Process Window			esign2.npl - [disp_	cnt.twr (READ ONL	Y)]	X	
			ծ 🛛 😵 🕅 🤅	(B C <u>S</u> S	2		•
Sources in Project: design2 		32 33 34 35 36 37 38 39	 S<0> S<1> S<2> S<3> S<4> S<4> S<5>	to PAD 17.427(F) 17.918(F) 17.062(F) 17.466(F) 15.961(F) 16.751(F)	XLXN_5 XLXN_5 XLXN_5 XLXN_5 XLXN_5 XLXN_5 XLXN_5	clock(s)	Phi A 0. 0. 0. 0. 0. 0. 0. 0.
	-	40 41 43 44 45 46 47	S<6> Clock to Setu Source Clock	l Src:Rise	ion clock	+ Src:Rise	0, + Sr(Des1 +
User Constraints Synthesize - XST Generate Programming File Generate PROM, ACE, or JTAG Fi Configure Device (iMPACT)		48 49 50 51 52 53 54 55	clk Analysis comp Peak Memory L		30 00:18	:02 2003	 + •
Console Find in Files Warmings Errors							
Process "Generate Programming File" is up to date.				Ln S	i2 Col 14		- 💽 //;

Downloading the Bitstream

Before downloading the disp_cnt.bit file, we must configure the interface CPLD on the XSA-50



board so it will work with the iMPACT programming tool. Double click the GXSLOAD icon and then drag & drop the piijitag.svf file from the C:\XSTOOLS4\XSA folder into the **gxsload** window. Then click on the Load button and the CPLD will be reprogrammed in less than a minute.

	🔀 gxsload
File Edit View Favorites Tools H Image: Second s	Board Type XSA-50 Load Port LPT1 Exit
 dwnldpar.svf dwnldpar-2_1.svf scats100.bit fcnfg.svf xsats100.exo fcnfg100.svf xsats50.bit fintf.svf xsats50.exo fintf100.svf ram100.bit 	FPGA/CPLD RAM Flash/EEPROM Image: Construction of the state of the
Type: SVF 156 KB 🖳 My Computer 🥢	

After the piiijtag.svf file is loaded into the XSA-50 Board, move the shunt on jumper J9 from the **xs** to the **xi** position. The XSA-50 Board is now setup so the FPGA can be configured through its boundary-scan pins with the iMPACT programming tool. Note that this process only needs to be done once because the CPLD on the XSA-50 Board will retain its configuration even when power is removed from the board. (If we want to go back to using the gxsload programming utility, we must move the shunt on J9 back to the **xs** position and download the dwnldpar.svf file into the CPLD.)

🙀 Xilinx - Project Navigator - C:\tmp\fpga_designs\design2\design2.npl - [disp_cnt.twr (READ ONLY)] - 🗆 🗵 📄 File Edit View Project Source Process Window Help 🗅 📂 🔚 🎒 📑 📝 國 🗉 🗈 🌃 🔳 🔁 🔊 🗞 🤋 💦 Ж Ba 🖪 <u>n</u> **P** • A - × Destination to PAD Internal Clock(s) Phi 🔺 Sources in Project: 33 17.427(F) XLXN_5 17.918(F) XLXN_5 17.062(F) XLXN_5 17.466(F) XLXN_5 15.961(F) XLXN_5 16.751(F) XLXN_5 16.751(F) XLXN_5 🧧 design2 34 35 S<0> 0. S<1> 0. 🖻 – 🛄 xc2s50-5tq144 36 S<2> S<3> ō. 🖃 🚯 disp_cnt (disp_cnt.sch) 37 0. 🚺 disp_ont.ucf 38 S<4> ο. 39 S<5> counter-behavioral (counter.vhd) 0. V 40 17.084(F) XLXN_5 ō. S<6> **⊡** leddcd-behavioral (leddcd.vhd) 41 42 43 44 45 📲 🕻 Module View 📔 📩 Snapshot View 🗌 Library View Clock to Setup on destination clock clk l N Src:Rise| Src:Fall| Src:Rise| Src Dest:Rise|Dest:Rise|Dest:Fall|Des1 46 Source Clock Processes for Source: "disp_cnt" ÷..... 48 49 User Constraints c1k 🗄 --- 🔂 🖉 Synthesize - XST 50 51 52 53 54 55 🗄 🖳 🖸 🖌 Implement Design Analysis completed Thu Oct 30 00:18:02 2003 Ė----Ū denerate Programming File 📄 🖋 Programming File Generation Repo Generate PROM, ACE, or JTAG Fi Peak Memory Usage: 40 MB Configure Device (iMPACT) Ŧ ۲ 31 V counter.vhd 📄 disp_cnt.par... 📄 disp_cnt.pa... 📄 disp_cnt.twr 📲 🛱 Process View Started process "Generate Programming File". × Completed process "Generate Programming File". • [▲ ▲ ▶ ▶ Console ∕ Find in Files ∕ Warnings ∕ Errors / **•** Process "Generate Programming File" is up to date. Ln 52 Col 14

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

The **Configure Devices** window now appears. Previously, we programmed the CPLD on the XSA-50 Board so it would support programming of the FPGA through its boundary-scan pins. So select the Boundary-Scan Mode option and click on the Next button.

Configure Devices	×
I want to configure device via :	
Boundary-Scan Mode	
Slave Serial Mode	
SelectMAP Mode	
Desktop Configuration Mode	
< Back Next > Cancel	Help

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. We know there is just a single FPGA on the XSA-50 Board and we 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 we select the automatic identification option and click on the Finish button.

Boundary-Scan Mode Selection	×
 Automatically connect to cable and identify Boundary-Scan chain 	
Enter a Boundary-Scan Chain	
	_
< Back Finish Cancel Help	

The iMPACT software will probe the boundary-scan chain and find there is a single FPGA in it. Now we need to tell iMPACT what bitstream file we want to download into this FPGA. Click on the OK button to proceed.

Boundary	-Scan Chain Contents Summary	×
٩	There was one device detected in the boundary-scan chair iMPACT will now direct you to associate a programming or BSDL file with this device	n.
	ок	

The **Assign New Configuration File** window now appears. Go to the tmp\fpga_designs\design2 folder and highlight the disp_cnt.bit file. Then click on the Open button.

Assign New Co	nfiguration File	<u>?</u> ×
Look in: 🔂	design2 💌 🖛 🗈 📸 🎫 -	
projnav		
disp_cnt.bi	t	
1		_
File name:	disp_cnt.bit Open	
Files of type:	FPGA Bit Files(*.bit) Cancel	
	Cancel <u>A</u> ll Bypass	

The main **iMPACT** window appears with a boundary-scan chain consisting of a single XC2S50 FPGA.

File Edit View Mode Operations Output Help Image:	
Boundany-Scan Claus Carial CalastMAD Dealder Configuration	
Boundary Scan Slave Senal SelectMAP Desktop Configuration	
Right click device to select operations	
xc2s50 disp_cnt.bit TDO	
'1': Loading file 'C:\tmp\fpga_designs\design2\disp_cnt.bit' done. INFO:iMPACT:501 - '1': Added Device xc2s50 successfully. 	
For Help, press F1 Configuration Mode Boundary-Scan Parallel III lpt1 2	.00 KI

Now right-click on the XC2S50 icon and select the Program... item on the pop-up menu.

untitled [Configuration Mode] - iMPACT					<u>_ </u>
File Edit View Mode Operations Output Hele Image:	8 🛱 🗉 🛱 i	· · · ·			
TDI	ercode				
done. INFO:iMPACT:501 - '1': Added Device xc2s50 succ Device #1 selected For Help, press F1	cessfully.	Boundary-Scan	Parallel III	lpt1	200 KI

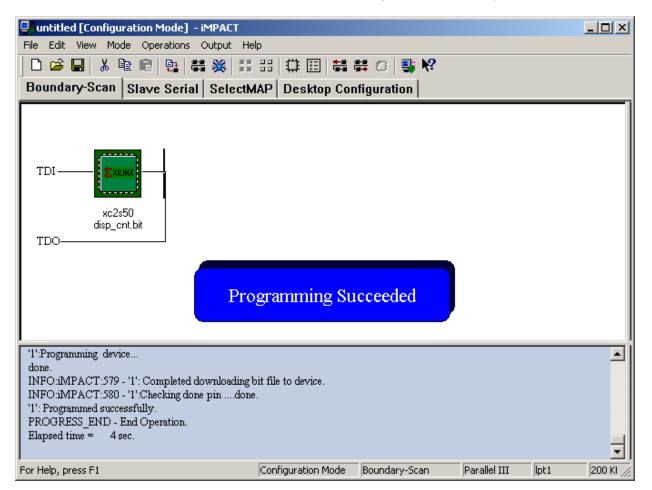
The **Program Options** window will appear. All we need to do at this point is click on the OK button to begin loading the disp_cnt.bit file into the FPGA.

Program Options	<u>? ×</u>
🗖 Erase Before Programming	Functional Test
Verify	🔲 On-The-Fly Program
Read Protect	
Virte Protect	PROM Load FPGA. Parallel Mode
Program Key	Use D4 for CF
PROM/CoolRunner-II Userco	
XPLA UES: Enter up to 13	characters
OK Cance	el Help

The progress of the bitstream download will be displayed. The download operation should complete within twenty seconds.

Operation Status	
Executing command	
Abort	

After the download operation completes, we can check the status messages in the bottom pane of the **iMPACT** window to see the FPGA was configured successfully.



Testing the Circuit

Once the XC2S50 FPGA on the XSA-50 Board is programmed, the circuit will begin operating without any further action from us. The LED display should repeatedly count through the sequence 0, 1, 2, 3, 4, 5, 6, 1, 8, 9, 8, 8, C, D, E, F with a complete cycle taking 5.4 seconds.



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:

- In the Project Navigator window, select Help→ISE Help Contents. You will be presented with a browser window containing topics that will let you learn more about the WebPACK software.
- 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.
- Go to the Xilinx web site and read their application notes and data sheets.
- Read the *comp.arch.fpga* newsgroup for helpful questions and answers about programmable logic design.