

# Introduction to WebPACK 4.1 for CPLDs

Using Xilinx WebPACK Software to Create CPLD Designs for the XS95 Board

Release date: 10/29/2001

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

All XC-prefix product designations are trademarks of Xilinx.

#### **Table of Contents**

Wł	hat This Is and <i>Is Not</i>	1
CP	PLD Programming	3
Ins	stalling WebPACK	5
	Getting WebPACK	5
	Installing WebPACK	8
	Getting XSTOOLs	8
	Installing XSTOOLs	9
Ou	ır First Design	10
	An LED Decoder	10
	Starting WebPACK Project Navigator	12
	Describing Your Design With VHDL	18
	Checking the VHDL Syntax	25
	Fixing VHDL Errors	26
	Fixing VHDL Errors         Synthesizing the Logic circuitry for Your Design	
		29
	Synthesizing the Logic circuitry for Your Design	29 30
	Synthesizing the Logic circuitry for Your Design Fitting the Logic Circuitry Into the CPLD	29 30 32
	Synthesizing the Logic circuitry for Your Design Fitting the Logic Circuitry Into the CPLD Checking the Fit	29 30 32 34
	Synthesizing the Logic circuitry for Your Design Fitting the Logic Circuitry Into the CPLD Checking the Fit Constraining the Fit	29 30 32 34 39
	Synthesizing the Logic circuitry for Your Design Fitting the Logic Circuitry Into the CPLD Checking the Fit Constraining the Fit Viewing the Chip	29 30 32 34 39 45
	Synthesizing the Logic circuitry for Your Design Fitting the Logic Circuitry Into the CPLD Checking the Fit Constraining the Fit Viewing the Chip Generating the Bitstream	29 30 32 34 39 45 52
Hie	Synthesizing the Logic circuitry for Your Design Fitting the Logic Circuitry Into the CPLD Checking the Fit Constraining the Fit Viewing the Chip Generating the Bitstream Downloading the Bitstream	29 30 32 34 39 45 52 55
Hie	Synthesizing the Logic circuitry for Your Design Fitting the Logic Circuitry Into the CPLD Checking the Fit Constraining the Fit Viewing the Chip Generating the Bitstream Downloading the Bitstream Testing the Circuit	29 30 32 34 39 45 52 55 56
Hie	Synthesizing the Logic circuitry for Your Design Fitting the Logic Circuitry Into the CPLD Checking the Fit Constraining the Fit Viewing the Chip Generating the Bitstream Downloading the Bitstream Testing the Circuit	29 30 32 34 39 45 55 55 56

Adding a Counter	61
Tying Them Together	66
Checking the VHDL Syntax	97
Constraining the Design	98
Synthesizing the Logic Circuitry for the Design	99
Fitting the Logic Circuitry Into the CPLD	100
Checking the Fit	102
Checking the Timing	104
Generating the Bitstream	106
Downloading the Bitstream	112
Testing the Circuit	113
Going Further	114



### 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 a CPLD project.
- How to target a design to a particular type of CPLD.
- 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 a CPLD.
- How to check device utilization and timing for a CPLD.
- How to generate a bitstream for a CPLD.
- How to download a bitstream to program a CPLD.
- How to test the programmed CPLD.

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 a CPLD.
- It will not teach you what to do if your design doesn't fit in a particular CPLD.
- It will not show you every feature of the WebPACK software and discuss how to set every option and property.

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

## 1

### **CPLD Programming**

Implementing a logic design with a CPLD 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 a *fitter* program to map the logic gates and interconnections into the CPLD. The CPLD consists of several *function blocks* which can be further decomposed into *macrocells* that can perform logic operations. The function blocks and macrocells are interwoven with various *routing matrices*. The fitter assigns gates from your netlist to various macrocells in the function blocks and opens or closes switches in the routing matrices to connect the gates together.
- 4. Once the fitting 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 CPLD chip (usually embedded in some larger system). The electronic switches in the CPLD open or close in response to the binary bits in the bitstream. Upon completion of the downloading, the CPLD 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>XS95 Board</u> containing a Xilinx XC95108 CPLD.



## 2

## **Installing WebPACK**

#### **Getting WebPACK**

Before downloading the WebPACK software you will have to register at <u>http://www.Xilinx.com/xlnx/xil\_entry2.jsp?sMode=login&group=webpack</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.html</u> to begin downloading the WebPACK software. After entering the WebPACK homepage, click on the Design Configurations button as shown on the next page.



Next, click on the Select All button. This will select all the WebPACK software modules that cover both FPGA and CPLD designs.

🚰 Xilinx WebPACK - Microso	oft Internet Explorer						
<u>File E</u> dit <u>V</u> iew F <u>a</u> vorites	: <u>T</u> ools <u>H</u> elp						191 191
← → → ⊗ 🐼 🖓 (0	Q 🖻 🧭 🖏 - 🍠 🖻	- 📃 🗼					
Address 🛃 http://www.xilinx.c			• @	Go Links (	🗿 Quick Sea	arch.exe	»>
<b>E</b> XILINX	Products						
	HOME PRODUCTS	SUPPORT	EDUCATION	PURCHASE	CONTACT	SEARCH	
	Devices   Design Tools	Intellectual	Property   Sy	stem Solutio	ns   Literat	ure	
Design Connigurations	PLD Design PGA Design PGA Design	ducts   Suppo	t   <u>Education</u>		Intact   Searce	Design Type CPLD Des FPGADes Select A Select A	ry Choice ed Modules a: sign: sign

Then click on the Download button to begin downloading the WebPACK software.

Xilinx WebPACK - Micros	oft Internet Explorer					
	s <u>T</u> ools <u>H</u> elp					
	Q 🖻 🧭 🔁 🗗			A Database		
Address 🖉 http://www.xilinx.c	com/webpack/index.html	c	≻Go ∐Links ⊧	🧕 Quick Sea	arch.exe	
<b>XILINX</b>	Products					
	HOME PRODUCTS	SUPPORT EDUCATION	PURCHASE	CONTACT	SEARCH	
	Devices   Design Tools	Intellectual Property	System Solutio	ns   Literat	ure	
Design Conligurations	PLD Design PGA Design I Home   Proc I Devices   Desi	ducts   Support   Education gn Tools   Intellectual Prope	<u>Purchase</u>   <u>Cc</u>	ontact   Searc	Design Type CPLD Des FPGA Des Deselect	ry Choice ed Modules a: sign: sign TAIL Download

Click on the link to download the WebPACK software in the Download WebPACK window. You can use either the FTP or the HTTP link. (You can also download the demo version of the ModelSim HDL simulator but we will not discuss the operation of that software in this tutorial.)

🖉 Download Web	PACK - Microsoft Inter	net Explorer	<u> </u>
<b>E</b> XILINX	Download Manag	er	*
WebPACK installat save each file to yo	ule selections, you will ion files. Please select e ur hard disk. Then run e	ither the FTP or H ach installer execu	TTP link and table.
	s <u>Download Accelerato</u> ple to resume interrupte		timize your
WebPACK 4.1WP1.0	) - October 15, 2001		
WebPACK_41wp10 MXE_41wp10_full_		110.25M <u>F</u> 38.58M <u>F</u>	
Total download size	e:	148.83M	
Estimated Total Dov	wnload Time:	T1: 56k:	14 Min 354 Min
Disk Space Require Maximum Disk Spac	d: ce Required During Inst	430.87M	304 IVIIII
			<b>T</b>

#### Installing WebPACK

After the WebPACK software download completes, double-click the .EXE file. The installation script will run and install the software. Accept the default settings for everything and you shouldn't have any problems.

#### Getting XSTOOLs

If you are going to download your CPLD bitstreams into an XS95 Board, then you will need to get the XSTOOLS software from <u>http://www.xess.com/ho07000.html</u>. Just download the <u>xstools4.exe</u> file.

#### Installing XSTOOLs

Double-click the xstools4.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 CPLD 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	ľ
0010	2	5
0011	3	3
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 **Project** 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.

💫 Xilinx - Project Navigator		×
<u>File</u> <u>E</u> dit <u>V</u> iew P <u>r</u> oject <u>S</u> ou	urce <u>P</u> rocess <u>M</u> acro <u>W</u> indow <u>H</u> elp	
New Project Open P <u>r</u> oject Open Exa <u>m</u> ple		
Close Projec <u>t</u> Sa <u>v</u> e Project As		
<u>N</u> ew Ctrl+N Open Ctrl+O Close		
<u>S</u> ave Ctrl+S Save <u>A</u> s Print Ctrl+P		
Save Alj		11
Recent Projects		
E <u>x</u> it		11
		Ш
Process View		9
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.

New Project				×
Project <u>N</u> ame:   Project Device Options:	Project <u>L</u> ocation C:Wilinx_WebF		- <del>1</del> 2	
Property Name		Value		Т
Device Family		XC9500 CPLDs		
Device		XC95108 PC84		
Design Flow		EDIF		
OK.	Cancel			

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\cpld\_designs folder.

Browse for Folder	? ×
Select Directory	
	-
RECYCLED	
SC 👘	
🗄 💼 sdram	
🗈 💼 🛄 SNAPSHOT	
i i temp	
🗄 🔂 fpga_designs 🧏	
	<b>_</b>
OKC	ancel

Next we will give our LED decoder design the descriptive title of design1 by typing it into the Project name field.

ew Project				
Project <u>N</u> ame:		Project Location	n:	
design1		C:\tmp\cpld_d	esigns\design1	
Project Device Option	ns:			
	Property Name		Value	
			Value XC9500 CPLDs	

To set the family of CPLD devices we will target with this design, click in the Value field of the Device Family property. Select the XC9500 CPLDs entry in the pop-up menu that appears.

New Project			×
Project <u>N</u> ame: design1 Project Device Option	ns:	Project Location	
F	Property Name		Value
Device Family			XC9500 CPLDs
Device			Virtex2
Design Flow			VirtexE
			CoolRunner2 CPLDs
	OK	Cancel	XC9500 CPLDs
	N		XC9500XL CPLDs が
			XC9500XV CPLDs
			CoolRunner XPLA3 CPLD

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 XC95108 PC84 since this is the device used in the XS95 Board where we will test our design.

New Project			×
Project <u>N</u> ame: design1 Project Device Options:	Project <u>Locatior</u> C:\tmp\cpld_de		
Property Name		Value	
Device Family		XC9500 CPLDs	
Device		XC95108 PC84	•
Design Flow OK	Cancel	XC95108 PC84 XC95144 PQ160 XC95144 PQ100 XC95144 TQ100 XC95216 HQ208	<b>↓</b> 3
		XC95216 PQ160 XC95216 BG352	-

Finally, our design will be done using VHDL so click in the Value field of the Design Flow property and select XST VHDL from the pop-up menu. This enables the Xilinx VHDL synthesizer.

New Project			×
Project <u>N</u> ame: design1 Project Device Options:	Project <u>L</u> ocatior C:\tmp\cpld_de		
Property Name		Value	
Device Family		XC9500 CPLDs	
Device		XC95108 PC84	
Design Flow		XST VHDL	-
		EDIF	
ΟΚ	Cancel	XST VHDL	
		XST Verilog	15
		ABEL XST Verilog ABEL XST VHDL	

Once all the fields are set, click on OK in the **New Project** window. Now the Sources pane in the **Project Navigator** window contains two items:

- 1. A project object called design1.
- 2. A chip object called XC95108 PC84 XST VHDL.

Elle Edit View Project Source Process Macro Window Help     Sources in Project:	Xilinx - Project Navigator - C:\tmp\cpld_de	signs\design1\design1.npl
Sources in Project: Casigni XC95108 PC84XST VHDL Casigni XC95108 PC84XST VHDL Casigni Ca	<u>File E</u> dit <u>V</u> iew P <u>r</u> oject <u>S</u> ource <u>P</u> rocess <u>M</u> ac	ro <u>W</u> indow <u>H</u> elp
Sources in Project  Cesign1  CXC35108 PC84XST VHDL  CXC35108 PC84XST	Ш	
Processes for Current Source: (No Processes Available) Process View (Empty Log)	Sources in Project: design1 XC95108 PC84-XST VHDL	
	Processes for Current Source: (No Processes Available)	
For Help, press F1		

#### 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 XC95108 PC84 object in the Sources pane and select New Source ... from the pop-up menu as shown below.

💊 Xilinx - Project N	avigator - C:\tmp\cpl	l_designs\de	sign1\desig	n1.npl				_ D ×
<u>F</u> ile <u>E</u> dit <u>V</u> iew P <u>r</u> oj	ject <u>S</u> ource <u>P</u> rocess	<u>Macro W</u> indo	w <u>H</u> elp					
🛛 🗅 🖨 🗶	e e s   e s	1	<b>3</b> 🔊	१ <b>№</b>	× P	<b>B</b> 2 2	<b>1</b>	
Sources in Project:								
	New Source Add Source Add Copy of Source Elemove	Insert Shift+Insert Delete						
 ■\$ Module [	Move to Library							
	<u>O</u> pen <u>C</u> lose							
Processes for Curre	<u>T</u> oggle Paths Prop <u>e</u> rties							
Process View			-		_			
(Empty Log								× ×
Add a new source to the	e 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, just highlight the VHDL Module item. Then we type the name of the module, leddcd, into the File Name field and click on Next.

New	×
User Document VHDL Module Schematic Vhdl Library VHDL Package VHDL Test Bench Test Bench Waveform State Diagram	File Name: Ieddcd Logation: c:\tmp\cpld_designs\design1
< <u>B</u> ack N	ext > 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.

Define VHDL Source				×
				_
Entity Name leddo	:d			
Architecture Name Beha	vioral			_
P. ( Horas	<b>B</b> <sup>1</sup> <b>B</b> <sup>1</sup> <b>B</b> <sup>1</sup>		1.00	
Port Name	Direction	MSB	LSB	
d	in	3	0	
	in			
	1	+ +		
< <u>B</u>	ack <u>N</u> ext>	Cancel	H	Help

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

Define VHDL Source	ce				2
Entity Name	leddcd				
Architecture Name	Behavioral				
Port Name		Direction	MSB	LSB	-
d	in		3	0	
S	in		6	0	
	in				
	< <u>B</u> ack	<u>N</u> ext >	Car	ncel	Help

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

efine VHDL Source				>				
Entity Name leddod								
				_				
Architecture Name Be	enavioral							
Port Name	Direction	MSB	LSB	1				
d	in	3	0					
s	out	6	0					
	in							
	inout	·						
		1						
	in			_				
	in			_				
	in			_				
	in			_				
	in :-			_				
	in			_				
	in			-				
	Back <u>N</u> ext>	Canc	el	Help				

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

N	ew Source Inform	nation					×
	Project Navigator following specifica		a new sk	eleton source	e with the		
Source Type: VHDL Module Source Name: leddcd Entity Name: leddcd Architecture Name: Behavioral Port Definitions:							
		d s	vector: vector:	3:0 6:0	in out		
	Source Directory	r:					
		< <u>B</u> ac	k [	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 6-9. We will describe the logic operations of the decoder in the architecture section between lines 13 and 16.



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 15-30).

```
library IEEE;
use IEEE.STD LOGIC 1164.ALL;
use IEEE.STD LOGIC ARITH.ALL;
use IEEE.STD LOGIC UNSIGNED.ALL;
entity leddcd is
    Port ( d : in std logic vector(3 downto 0);
           s : out std logic vector(6 downto 0));
end leddcd;
architecture Behavioral of leddcd is
begin
    s <=
            "1110111" when d="0000" else
            "0010010" when d="0001" else
            "1011101" when d="0010" else
            "1011011" when d="0011" else
            "0111010" when d="0100" else
            "1101011" when d="0101" else
            "1101111" when d="0110" els
            "1010010" when d="0111" else
            "11111111" when d="1000" else
            "1111011" when d="1001" else
            "11111110" when d="1010" else
            "01011111" when d="1011" else
            "0001101" when d="1100" else
            "00111111" when d="1101" else
            "1101101" when d="1110" else
            "1101100"
end Behavioral;
```

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.

Xilinx - Project Navigator - C:\tmp\cpld_des		- D × - B ×
	🗷 🚾 🗛 😵 😵 🛛 X 🛍 🛍 🗠 🗠 🙀 🗠 n	•
Sources in Project: Control design1 Control Control	<pre>library IEEE; use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL; use IEEE.STD_LOGIC_UNSIGNED.ALL; entity leddcd is Port ( d : in std_logic_vector(3 downto 0); s : out std_logic_vector(6 downto 0)); end leddcd;</pre>	
Module 🖿 Snapshot 🖺 Library	architecture Behavioral of leddcd is	
Processes for Current Source: Design Entry Utilities Synthesize View Synthesis Report Analyze Hierarchy Check Syntax Implement Design Generate Programming File	begin s <= "1110111" when d="0000" else "0010010" when d="0001" else "1011101" when d="0010" else "1011011" when d="0011" else "0111010" when d="0100" else "1101011" when d="0101" else "1010010" when d="0111" else	- - -
Process View	V leddcd.vhd	
Done: completed successfully.		×
Hierarchy is up to date.	Ln 1, Col 1	

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 20. You can manually scroll to line 20 in the editor pane, or you can double-click on the error message in the log pane to go directly to the erroroneous line.



On line 20, 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\cpld\_designs\design1\design1.npl - [leddcd.vhd] - 🗆 × \_ 🖻 🗵 ile Edit View Project Source Process Macro Window Help 🤋 💦 X 🖻 🖻 🗅 📂 🔚 🎒 🗌 🚰 📝 國 🗐 🔮 🚿 🔳 📭 🗞 🛛 <u>12</u> <u>24</u> 🖣 cs\_n • ≜ × "1110111" when d="0000" else ٠ s <= Sources in Project: "0010010" when d="0001" else --- 🧧 design1 "1011101" when d="0010" else 🗄 📶 XC95108 PC84-XST VHDL "1011011" when d="0011" else 🛛 📝 leddcd (leddcd.vhd) "0111010" when d="0100" else "1101011" when d="0101" else "11011111" when d="0110" else 🗖 🛱 Module ... 💼 Snapshot... Library "1010010" when d="0111" else "11111111" when d="1000" else ≜ × "1111011" when d="1001" else "11111110" when d="1010" else Processes for Current Source: "0101111" when d="1011" else 1 Design Entry Utilities +.... "0001101" when d="1100" else ⊡-ā Synthesize "00111111" when d="1101" else ð View Synthesis Report "1101101" when d="1110" else Analyze Hierarchy C "1101100" 😋 🗙 Check Syntax Implement Design end Behavioral; Generate Programming File • Ð 🗖 🛱 Process View **V**1 leddcd.vhd × \* ERROR:HDLParsers:162 - C:/tmp/cpld designs/design1/leddcd.vhd Line 32. Read symbol END, CPU : 0.11 / 0.22 s | Elapsed : 0.00 / 0.00 s -Console Findin Files -Process 'exewrap -mode pipe -tapkeep -command C: /Xilinx\_WebPACK/bin/nt/xst.exe -ifn leddcd.xst -ofn ledi Ln 32, Col 1

The syntax checker now finds another error on line 31 of the VHDL code.

When we look at line 31 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 29. Adding the semicolon to the end of line 29 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\cpld_designs\design1\desig	n1.npl - [leddcd.vhd]	- 🗆 🗵
iw <u>File Edit View Project Source Process Macro Window H</u> elp		_ 8 ×
	🖇 😽 🖁 X 🖻 🖻 🗠 🗠 🙀 <mark>∞_n</mark>	•
Sources in Project: design1 XC95108 PC84XST VHDL Module Snapshot Library Processes for Current Source: Design Entry Utilities Synthesize	"1110111" when d="0000" else "0010010" when d="0001" else "1011101" when d="0010" else "101101" when d="0011" else "0111010" when d="0100" else "1101011" when d="0101" else "1101011" when d="0111" else "1101010" when d="0111" else "1111111" when d="1000" else "1111101" when d="1001" else "1111110" when d="1010" else "0101111" when d="1011" else	
	"0001101" when d="1100" else "0011111" when d="1101" else "1101101" when d="1110" else "1101100";	
Process View		
Done: completed successfully.	Ln 30, Col 23	×

The synthesizer will read the VHDL code and transform it into a netlist of gates. This will take only a minute. If no problems are detected, a  $\checkmark$  will appear next to the Synthesize process. You can double-click on the View Synthesis Report to see the various synthesizer options that were enabled and some simple statistics for the synthesized design.

#### Fitting the Logic Circuitry Into the CPLD

We now have a synthesized logic circuit for the LED decoder, but we need to fit it into the logic resources of the CPLD in order to actually use it. We start this process by highlighting the XC95108 PC84 object in the Sources pane and then double-clicking on the Implement Design process.

Xilinx - Project Navigator - C:\tmp\cpld_designs\design1\design1.npl - [leddcd.vhd]				
ile Edit ⊻iew Project Source Process	<u>M</u> acro <u>W</u> indow <u>H</u> elp	_ 뭔 ×		
	🗜 🖪 🖾 🗞 🦹 🛠 🗍 X 🖻 🖬 🗠 🗠 🙀 🗠 n	•		
Sources in Project: design XC95108 PC84-XST VHDL Module Snapshot Library Module Snapshot Library Processes for Current Source: Check Syntax Implement Design Check Syntax Implement Design Generate Timing Generate Post-Fit Simulal Generate Post-Fit Simulal Generate BIS Model Fit Process View	begin s <= "1110111" when d="0000" else "0010010" when d="0001" else "1011101" when d="0010" else "101101" when d="010" else "011101" when d="010" else "110111" when d="011" else "110101" when d="011" else "1101010" when d="100" else "111111" when d="100" else "111111" when d="100" else "111111" when d="101" else "111111" when d="101" else "001111" when d="101" else "001111" when d="1100" else "001111" when d="1100" else "001111" when d="1100" else "101101" when d="1100" else "101100"; ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓			
Done: completed successfully	r.	A V F		
Process "Synthesize" is up to date.	Ln 30, Col 23			

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). A successful implementation is indicated by 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 Fit process maps the netlist into the circuitry elements and routing matrices contained in the device we selected. If the Implement Design provcess had failed, a X would appear next to the subprocess where the error occured.

You may also see a <sup>1</sup>/<sub>2</sub> to indicate a successful completion but some warnings were issued or not all the subprocesses were enabled.



#### Checking the Fit

We have our design fitted into the XC95108 CPLD, 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 Fitter Report in the Process pane.

Xilinx - Project Navigator - C:\tmp\cpld_designs\design1\design1.npl - [leddcd.vhd]					
i <u>File E</u> dit <u>V</u> iew P <u>r</u> oject <u>S</u> ource <u>P</u> rocess <u>M</u>	<u>1</u> acro <u>W</u> indow <u>H</u> elp	_ 8 ×			
	🗉 🖪 🛐 🗞 🦹 🛠 📗 X 🖻 💼 🗠 🗠 🦓 🖙 n	•			
Sources in Project: 	begin s <= "1110111" when d="0000" else "0010010" when d="0001" else "1011101" when d="0010" else "1011011" when d="0101" else "1101011" when d="0101" else "1101111" when d="0110" else				
Processes for Current Source:	"1101111" when d="0110" else "1010010" when d="1000" else "1111111" when d="1001" else "1111011" when d="1001" else "1111110" when d="1011" else "0001101" when d="1100" else "0001101" when d="1100" else "1101101" when d="1101" else "1101100";				
Generate Timing	leddcd.vhd				
Done: completed successfully.		× v			
Process "Implement Design" is up to date.	Ln 30, Col 23	🔶 //.			

This brings up a window containing the fitting statistics for the LED decoder. The top few lines of the file show the LED decoder only uses 7 of the 108 available macrocells in the XC95108 CPLD. And it only uses 11 I/O pins (4 for input, 7 for output).

cpldfit: version E.30	Xilinx Inc.
	Fitter Report
Design Name: leddcd	Date: 10-20-2001, 2:03PM
Device Used: XC95108-7-PC84	
Fitting Status: Successful	
******	Resource Summary ************************************
Macrocells Product Terms	Registers Pins Function Block
Used Used	Used Used Inputs Used
7 /108 ( 6%) 24 /540 ( 4	%) 0 /108 ( 0%) 11 /69 ( 15%) 24 /216 ( 11%)
Further down in the fitting report we can see what pins the inputs and outputs use. The **d** inputs have been assigned to pins 72, 17, 83, and 44. The **s** outputs which drive the LED segments have been routed through pins 32, 45, 1, 6, 71, 14, and 57.

**************Resc	urces Us	ed by Su	ccessful	ly Ma	pped 1	Logic	*******	******
** LOGIC **								
Signal	Total	Signals	Loc	Pwr	Slew	Pin	Pin	Pin
Name	Pt	Used		Mode	Rate	#	Туре	Use
s<0>	4	4	FB5_2	STD	FAST	32	I/O	0
s<1>	3	4	FB6_2	STD	FAST	45	I/O	0
s<2>	3	4	FB1_2	STD	FAST	1	I/O	0
s<3>	2	4	FB1_9	STD	FAST	6	I/O	0
s<4>	4	4	FB2_2	STD	FAST	71	I/O	0
s<5>	4	4	FB3_2	STD	FAST	14	I/O	0
s<6>	4	4	FB4_2	STD	FAST	57	I/O	0
** INPUTS **								
Signal			Loc			Pin	Pin	Pin
Name						#	Туре	Use
d<0>			FB2_3			72	I/O	I
d<1>			FB3_5			17	I/O	I
d<2>			FB2_16			83	I/O	I
d<3>			FB5_17			44	I/O	I

End of Resources Used by Successfully Mapped Logic

The fitting report even lists the logic equations for each output:

```
; Implemented Equations.
```

```
/"s<1>" = /"d<0>" * "d<2>" * "d<3>"
      + "d<1>" * "d<2>" * "d<3>"
      + /"d<0>" * "d<1>" * /"d<2>" * /"d<3>"
/"s<2>" = "d<0>" * /"d<3>"
      + "d<0>" * /"d<1>" * /"d<2>"
      + /"d<1>" * "d<2>" * /"d<3>"
/"s<3>" = /"d<1>" * /"d<2>" * /"d<3>"
      + "d<0>" * "d<1>" * "d<2>" * /"d<3>"
/"s<4>" = "d<0>" * "d<1>" * "d<3>"
      + /"d<0>" * "d<1>" * "d<2>"
      + /"d<0>" * "d<2>" * "d<3>"
      + "d<0>" * /"d<1>" * "d<2>" * /"d<3>"
/"s<5>" = "d<0>" * "d<1>" * /"d<3>"
      + "d<0>" * /"d<2>" * /"d<3>"
      + "d<1>" * /"d<2>" * /"d<3>"
      + /"d<1>" * "d<2>" * "d<3>"
/"s<6>" = /"d<0>" * /"d<1>" * "d<2>"
      + /"d<1>" * "d<2>" * "d<3>"
      + "d<0>" * "d<1>" * /"d<2>" * "d<3>"
      + "d<0>" * /"d<1>" * /"d<2>" * /"d<3>"
/"s<0>" = "d<0>" * "d<1>" * "d<2>"
      + "d<0>" * /"d<1>" * /"d<2>" * /"d<3>"
      + /"d<0>" * "d<1>" * /"d<2>" * "d<3>"
      + /"d<0>" * /"d<1>" * "d<2>" * /"d<3>"
```

## Constraining the Fit

The problem we have now is that the inputs and outputs for the LED decoder were assigned to pins picked by the fitting process, but these are not the pins we actually want to use on the XS95 Board. The CPLD on the XS95 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	XS95 XC95108 CPLD Pin
d0	P46
d1	P47
d2	P48
d3	P50

Likewise, the XS95 Board has a seven-segment LED attached to the following pins of the CPLD:

LED Decoder Output	XS95 XC95108 CPLD Pin
s0	P21
s1	P23
s2	P19
s3	P17
s4	P18
s5	P14
s6	P15

How do we constrain the fitting process so it assigns the inputs and outputs to the pins we want to use? We start by highlighting the leddcd object in the Sources pane and then double-clicking the Edit Implementation Constraints (Constraints Editor) process.



The **Constraints Editor** window appears. Click on the Ports tab in the upper pane. A list of the inputs and outputs for the LED decoder will appear. We can enter our pin assignments here.

<b>∑Xilinx Constraints Editor - [Ports</b> <u>File Edit View Window H</u> elp	- leddcd.ngd / leddc	d.ucf]		
	<b>? №</b> ?			
Port Name	Port Direction	Location	Pad to Setup	Clock to Pad
d<0>	INPUT			N/A
d<1≻	INPUT			N/A
d<2>	INPUT			N/A
d<3>	INPUT			N/A
s<0>	OUTPUT		N/A	
s<1>	OUTPUT		N/A	
s<2>	OUTPUT		N/A	
s<3>	OUTPUT		N/A	
s<4>	OUTPUT		N/A	
s<5>	OUTPUT		N/A	
s<6>	OUTPUT		N/A	
☐ 1/0 Configuration Options	Pad Groups Group Name:		Create Group	]
Prohibit I/O Locations	Select Group:		Pad to Setup Clock to Pad	
Global Ports	Advanced	Misc		
// Template UCF file created by the Pro	oject Navigator			
UCF Constraints (read-write)	F Constraints (read-only)	Source Constra	aints (read-only)	
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: P46. Do this for each of the inputs and outputs using the pin assignments from the tables shown above. After doing this, the **Constraints Editor** window appears as follows.

Xilinx Constraints Editor - [Ports - ile <u>E</u> dit <u>V</u> iew <u>W</u> indow <u>H</u> elp	- leddcd.ngd / ledd	cd.ucf*]		
D 😂 🖬 🗙 ≍ ៲⊃ 🗐 ଫ	<b>⊘ №</b> 1			
	8 7:			
Port Name	Port Direction	Location	Pad to Setup	Clock to Pad
d<0>	INPUT	P46		N/A
d<1>	INPUT	P47		N/A
d<2>	INPUT	P48		N/A
d<3>	INPUT	P50		N/A
s<0>	OUTPUT	P21	N/A	
s<1>	OUTPUT	P23	N/A	
s<2>	OUTPUT	P19	N/A	
s<3>	OUTPUT	P17	N/A	
s<4>	OUTPUT	P18	N/A	
s<5>	OUTPUT	P14	N/A	
s<6>	OUTPUT	P15	N/A	
I/O Configuration Options	Pad Groups Group Name:	P15	Create Group	]
Prohibit I/O Locations	Select Group:		Pad to Setup Clock to Pad	1
Global Ports Advanced Misc				
NET "s<5>" LOC = "P14"; NET "s<6>" LOC = "P15";				
UCF Constraints (read-write) UCF Constraints (read-only) Source Constraints (read-only)				
r Help, press F1				

After the pin assignments are entered, click on the button to save the pin assignment constraints. Then select File > Exit in the **Constraints Editor** window.

Since we are changing the constraints on our design, we are asked to reset the implementation process so it will be re-run with our new constraints. This just means we have to re-run the fitting process again if we want the design to use the pin assignments we just made. Click on Reset and then double-click the Implement Design process to re-fit the design with the new pin assignments.

Notice	
Reset the Implement Design process so that your U( be read?	CF changes will
The User Constraint File (UCF) has changed. As a re be possible to reproduce the same implementation re new UCF. To incorporate the new UCF at this time, ch to mark the Implement Design process out of date. T Implement Design process. Otherwise, choose RET/ current implementation results intact and not incorpo UCF at this time.	esults using the noose RESET Then re-run the AIN to keep the
Reset	etain

Next double-click on the Fitter Report process to view the pin assignments made by the fitter process. Looking through the fitter report, we see the following:

**************Resources Used by Successfully Mapped Logic************************************								
** LOGIC **								
Signal	Total	Signals	Loc	Pwr	Slew	Pin	Pin	Pin
Name	Pt	Used		Mode	Rate	#	Туре	Use
s<0>	4	4	FB3_11	STD	FAST	21	I/O	0
s<1>	3	4	FB3_12	STD	FAST	23	I/O	0
s<2>	3	4	FB3_8	STD	FAST	19	I/O	0
s<3>	2	4	FB3_5	STD	FAST	17	I/O	0
s<4>	4	4	FB3_6	STD	FAST	18	I/O	0
s<5>	4	4	FB3_2	STD	FAST	14	I/O	0
s<6>	4	4	FB3_3	STD	FAST	15	I/O	0
** INPUTS **								
Signal			Loc			Pin	Pin	Pin
Name						#	Туре	Use
d<0>			FB6_3			46	I/O	I
d<1>			FB6_5			47	I/O	I
d<2>			FB6_6			48	I/O	I
d<3>			FB6_8			50	I/O	I

End of Resources Used by Successfully Mapped Logic

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, you can get a graphical depiction of how the logic circuitry and I/O are assigned to the CPLD macrocells and pins. Just highlight the leddcd object in the Sources pane and then double-click the View Fitted Design (ChipViewer) process.



The ChipViewer window will appear containing two panes:

- 1. The left-hand pane lists the LED decoder inputs and outputs assigned to the various function blocks in the XC95108 PC84 CPLD.
- 2. The right-hand pane shows the 108 macrocells of the CPLD arranged into six groups of 18 cells each. The 69 I/O pins are also shown. (The 15 pins used for Vcc, GND, and programming are not shown.)

Xilinx ChipViewer - leddo	d.vm6
File Edit View Help	
<u> </u>	
leddcd.vm6 Post-	eddcd.vm6 Placement for XC95108-7-PC84
eddcd	
● FB2 ● <u> </u> FB3	
• FB4	
● FB5 ● 💼 FB6	
	비밀 털 타
Done	

The function blocks with  $\mathbb{H}$  icons next to them contain macrocells that are used in our design. (The remaining function blocks are not used by our design.) Clicking on the  $\mathbb{H}$  next to FB6, we can see that the inputs enter through pins attached to function block 6. Meanwhile, all the outputs are generated by macrocells in function block 3.

Xilinx ChipViewer - leddco File Edit ⊻iew Help	i. vm6
🖻 🖬 🎒 🔋 📢 🗨	
Ieddcd vm6 Post         FB1         FB2         FB3         N215         N231         N178         N200         N160         N99         N140         S<5>         \$<58         \$<5>         \$<5>         \$<5>         \$<5>         \$<5>         \$<5>         \$<5>         \$<5>         \$<5>         \$<5>         \$<5>         \$<5>         \$<5>         \$<5>         \$<5>         \$<5>         \$<5>         \$<5>         \$<5>         \$<5>         \$<5>         \$<50>         \$<50>         \$<50>         \$<50>         \$<50>         \$<50>         \$<50>         \$<50>         \$<50>         \$<50>         \$<50>         \$<50>         \$<50>         \$<50>         \$<50>         \$<50>         \$<50>   \$	leddcd.vm6 Placement for XC95108-7-PC84
Done	

We can see which inputs affect each output by right-clicking in the right-hand pane and selecting the Inputs Connection item from the pop-up menu.



This causes red connecting lines to be drawn from each green-colored input to the blue-colored outputs it affects as shown below. For the LED decoder, every input affects every output so there are seven lines connecting each input to every output.



We can see more detail by clicking on the button several times to expand the right-hand pane. We can see the name of each output and the pin it is assigned to. By placing the mouse pointer over a particular pin, we get some information on the settings for the configuration options for the pin and the attached macrocell. For example, macrocell 12 of function block 3 is configured in the standard power consumption mode, and pin 23 is set for the maximum slew rate. We can also double-click on a macrocell to expand it into a separate window where we can see a list of its inputs, the Boolean equation for its output, and a rather uninteresting block-level schematic.

Xilinx ChipViewer - leddc	d. vm6	-OX
<u>File Edit View H</u> elp		
🖻 🖩 🎒 💡 📢 🗨		
📓 leddcd.vm6 Post-💶 🗖	leddcd.vm6 Placement for XC95108-7-PC84	
<pre> leddcd     FB1     FB2     FB3     FB3     INPUTS     N215     N215     N231     N178     N200     N160     N99     N140     OUTPUTS     s&lt;5&gt;     s&lt;6&gt;     s&lt;3&gt;     s&lt;4&gt;     s&lt;2&gt;     s&lt;0&gt; </pre>	$ \begin{array}{c c c c c c c c c c c c c c c c c c c $	
FB4 FB5 FB6 INPUTS d<0> d<1> d<1> d<2> d<2> d<3>	32     32     32     45       33     33     34     46       34     34     46     47       35     46     47     43       36     46     48     42       36     403     403     50       37     403     50     51	)
Done		

#### Generating the Bitstream

Now that we have synthesized our design and fitted it to the CPLD 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 Configure Device (iMPACT) process.



The **iMPACT** window will appear and it will try and fail to establish a connection with the CPLD through the various ports of the PC. This is normal since the XS95 Board is not made to interface directly with the iMPACT software. Just click on the OK button and proceed.

🚚 Untitled - iMPACT	
<u>File</u> <u>E</u> dit Operations <u>O</u> utput ⊻iew	<u>H</u> elp
🗋 🗅 🚅 🖬 👗 X 🖻 💼 🗮 💥	
Boundary Scan Slave Serial	Select Map
<u>.</u>	MPACT Some could not be established. Please check the cable connections and cable power source.
Cable connection failed. Connecting to cable (COM4 Port).	
Cable connection failed.	
Elapsed time = 24 sec.	
Cable autodetection failed. =>	
Connecting to the selected cable	No Connection
connecting to the selected cable	INO CONNECTION J

Since iMPACT cannot determine how the CPLD is connected to the PC, it will ask you what type of communication link the connection uses. Since there is no appropriate link for the XS95 Board, click the Cancel button.

Cable Communication Setup	?×
Communication Mode	O Multilinx/Serial
C Multilinx/USB	
Baud Rate: 57600	Port:
<u> </u>	<u>H</u> elp

The **iMPACT** window now shows the JTAG chain of chips that are to be programmed. We only have one chip in our LED decoder design, so only one XC95108 CPLD is shown.

🚭 Untitled - iMPACT	- 🗆 🗵
<u>F</u> ile <u>E</u> dit Operations <u>O</u> utput <u>V</u> iew <u>H</u> elp	
D 🗳 🖬   X 🖻 🖻   🛱 💥   👯 🏭 🗮   🌐 🖽   🎒   隆	
Boundary Scan Slave Serial Select Map	
Right click device to select operations	
TDI xc95108 leddcd.jed TDO	
	-
 Device #1 selected Device #1 selected Device #1 selected => 	
For Help, press F1 No Connection	Þ

We proceed by selecting the destination for the bitstream. The XS95 Board has a separate utility called gxsload for programming the CPLD, so we need to store the bitstream into a file that gxsload can read. To do this, select Output  $\rightarrow$  Use File  $\rightarrow$  SVF File  $\rightarrow$  Create SVF File... as follows.

Untitled - iMPACT			
<u>File Edit Op</u> erations <u>O</u>	utput <u>V</u> iew <u>H</u> elp		
🗅 😅 🖬 👗 🖻 Boundary Scan 🗄	Cable <u>S</u> etup Cable <u>R</u> eset	∄ ☷   <b>雲</b> ,   № ap	
	Cable <u>D</u> isconnect		
	<u>U</u> se Cable		
	Use <u>F</u> ile 🔹 🕨	S <u>V</u> F File →	<u>C</u> reate SVF File
TDI — <b>Xilinx</b>	-	ST <u>A</u> PL File 🕨	Append to SVF File 상 Close SVF File
xc95108 leddcd.jed			
TDO			
, 			
Device #1 selected			_
Device #1 selected			
Device #1 selected			
Device #1 selected =>			
T			
Create an SVF file and direct	subsequent operations to it	No Connection	

Now a window appears where we can enter the name for the file that will hold the bitstream. We can click on the Save button to accept the default name of leddcd.svf.

Create a New	/ SVF File				? ×
Savejn: 🔂	design1	- 🗈	<u></u>	Ť	<b></b>
_ngo					
🛄 xst					
File <u>n</u> ame:	leddcd				<u>S</u> ave
Save as type:	SVF Files(*.svf)		•	(	Cancel
					//.

At this point we can generate the bitstream that will be stored in the leddcd.svf file. Click on the Operations → Program... menu item to initiate the actual bitstream generation process.

🔩 Untitle	ed - iMPACT			
<u>F</u> ile <u>E</u> dit	Operations <u>O</u> utput <u>V</u> iew <u>H</u> elp			
🗋 🗋 🗃	Program	🛱 🖽 📑 😽 🕷	?	
Bounda	⊻erify <sup>M</sup>	/ap		
	<u>E</u> rase Functional Test			
	Blank Check			
	 <u>R</u> eadback			
	Program XPLA <u>U</u> ES			
TDI-	- Get Device ID			
	Giet Device <u>C</u> hecksum			
	Get Device <u>S</u> ignature/Usercode Get ⊠PLA device UES			
TDO-	- IDCODE <u>L</u> ooping			
				•
Device #1 : Device #1 :				
Device #1 :				
Device #1 :	selected			
=>				
Programs th	ne selected devices	File   SVF		

A **Program Options** window appears where we can set some options for the generated bitstream. We usually check the Erase Before Programming box so that the Flash storage in the CPLD will be erased before we start loading a new design. (The only time we can leave this box unchecked is when we are programming a CPLD which we know is already erased.)

The other two options which are of interest are Write Protect and Read Protect. Checking the Write Protect box generates a bitstream which programs the CPLD so that it cannot be reprogrammed. (Don't worry, the device can still be erased if you want to re-use it.) The Read Protect option prevents anyone from getting the bitstream out of the CPLD so they can't steal the design. We won't enable either of these options.

Program Options	?×
☑ <u>E</u> rase Before Program	mming 🦳 Eunctional Test
☐ <u>V</u> erify ☐ <u>R</u> ead Protect —	PROM Skip user array
Virtex2	Load Fpga
<u>S</u> ecure Mode     Program <u>K</u> ey	Laraner Mode
PROM Usercode (8 H	ex Chars) FFFFFFF
XPLA UES: Enter up t	© 0 characters
	<u>C</u> ancel <u>H</u> elp

After we click OK in the **Program Options** window, the bitstream generation process begins. The progress is reported in the lower pane of the **iMPACT** window:

🚭 Untitled - iMPACT			
<u>File Edit Operations Output View H</u> elp			
🗋 D 🚅 🖬 👗 🛍 🖻 🕄 👯 👯 🔡 🖽	: 🗄 📑 😽	?	
Boundary Scan Slave Serial Select Map	<b>)</b>		
TDI <b>EXLINX</b> xc95108 leddcd.jed TDO			
'1': Programming device done. '1': Programming completed successfully. PROGRESS_END - End Operation. Elapsed time = 2 sec. =>			<b>▲</b>
For Help, press F1	File   SVF		

Once the bitstream file is generated, click on File $\rightarrow$ OK to close the window. (You will be asked if you want to save the configuration. Don't bother.)

# Downloading the Bitstream

Now we have to get the bitstream file programmed into the CPLD of the XS95 Board. The XS95 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 XS95 Board is programmed using the gxsload utility. We double click the GXSLOAD icon to bring up the following window:

🔀 gxsload		
	95-108 <u>•</u> 12 •	Load
FPGA/CPLD	RAM	Flash/EEPROM
High Address		
Low Address		
Upload Format	HEX 💽 🗋	HEX 💽 🗋

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

🔁 design1	🔀 gxsload 📃 🗖 🗙
Eile       Edit       View       Go       Favorites       He       Ne         ↓       ↓       ↓       ↓       ↓       ↓       ↓       ↓       ↓         ↓       ↓       ↓       ↓       ↓       ↓       ↓       ↓       ↓       ↓         ↓ <td< th=""><th>Board Type XS95-108  Port LPT2  Exit</th></td<>	Board Type XS95-108  Port LPT2  Exit
_ngo      projnav.log         xst       adesign1.jid         _impact.log      chipview.pl         aleddcd.svf       aleddcd.mfd         aleddcd.jed       aleddcd.pnx        impact.rsp       ileddcd.rpt         automake.log      cpldfit.rsp         tmperr.err      cpldfit.tcl	FPGA/CPLD       RAM       Flash/EEPROM         leddcd.svf       Image: Second s
1 object(s) selected	Upload Format HEX 💽 🗀 HEX 💽 🗀

1

Then we click on the Load button to initiate the programming of the XS95 Board. Downloading the leddcd.svf file will take under a minute.

👗 gxsload 📃 🔍 🗙
Board Type XS95-108
Port LPT2  Exit
FPGA/CPLD RAM Flash/EEPROM
leddcd.svf
High Address
Low Address
Upload Format 🛛 💌 🔁 🖿 🗀 🖿 🗀
🗶 gxsload 📃 🔍
Board Type XS95-108
FPGA/CPLD RAM Flash/EEPROM
leddcd.svf
High Address
Low Address
Upload Format 🛛 🖛 🔁 🛏 🖂 🗀
Configure CPLD
Downloading leddcd.svf

### Testing the Circuit

Once the CPLD on the XS95 Board is programmed, we can begin testing the LED decoder. The eight data pins of the PC parallel port connect to the CPLD 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 XS95 Board.



Double-clicking the GXSFORT 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 CPLD. For example, setting (D3,D2,D1,D0) = (1,1,1,0) will cause E to appear on the seven-segment LED of the XS95 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.



# **Hierarchical Design**

# A Displayable Counter

We went through a lot of work for our first CPLD 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→New Project... menu item. We name the project *design2* and store it in the same folder as the previous design: C:\tmp\cpld\_designs. The other properties in the **New Project** window retain the same values we set in the previous project.

New Project		x
Project <u>N</u> ame: design2	Project <u>L</u> ocation c:\tmp\cpld_de	
Project Device Options:		
Property	y Name	Value
Device Family		XC9500 CPLDs
Device		XC95108 PC84
Design Flow		XST VHDL
	OK Cancel	

Once we click on OK in the **New Project** window, the **Project Navigator** window appears as shown below.

Xilinx - Project Navigator - c:\tmp\cpld_designs\	s\design2\design2.npl	x
<u>File Edit View Project Source Process Macro W</u>	<u>√</u> indow <u>H</u> elp	
	🔄 🔉 🇞 🧣 🕺 🕺 🛍 🛍 🗠 🗠 🦮 💁	~
		n I
Sources in Project:		
CS5108 PC84-XST VHDL		
🗖 📲 Module View 💼 Snapshot View 🖺 Library View		
	-	
Processes for Current Source:		
(No Processes Available)		
Process View		
I (Empty Log)		
(Empty Log)		
	٩ ا	7
Console Find in Film		
Hierarchy is up to date.		11.

# Adding the LED Decoder

The first thing we do after getting the *design2* project started is to add the LED decoder module. We do this by right-clicking on the XC95108 PC84 object in the Sources pane and selecting Add Source ... from the pop-up menu.

📚 Xilinx - Project Navigator	<ul> <li>c:\tmp\cpld_design</li> </ul>	ns\design2\des	sign2.npl				- 🗆 🗵
<u>File E</u> dit <u>V</u> iew P <u>r</u> oject <u>S</u> our	rce <u>P</u> rocess <u>M</u> acro	<u>W</u> indow <u>H</u> elp					
		- 🖪 🖪 🎘	. 💡 📢	)    X @	6 <u>6</u> 2 2	e 📴 cs_n	7
							1
Sources in Project:							
C95108 PC84-XST	New Source		1				
	Add Source	Insert					
	Add Copy of Source						
- Snapsi	<u>R</u> emove	Delete					
	Move to Library						
	<u>O</u> pen						
Processes for Current Source	<u>C</u> lose						
	<u>T</u> oggle Paths						
	Prop <u>e</u> rties						
Process View							
(Empty Log)							×
Console Find in Files	/						
Add a file from another project							

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

Add Existing	Sources			? ×
Look jn: 🔂	design1	• <b>E</b>	M 🖄	
ngo xst				
leddcd.vho	1			
File <u>n</u> ame:	leddcd.vhd			<u>O</u> pen
Files of <u>type</u> :	Sources (*.txt;*.vhd;*.sch;*.tb	ow;*.dia;)	•	Cancel

After clicking on Open, a window appears that asks us the type of file we are adding to the project.



We select VHDL Module since the leddcd.vhd file contains a standard VHDL description of a circuit. (Packages contain extra syntactical elements for modules meant to be used as a library. Test benches contain VHDL code that exercises other VHDL modules through a sequence of tests.) After clicking OK, we see that the LED decoder module has been added to the Source pane of the **Project Navigator** window on the next page.

Xilinx - Project Navigator - c:\tmp\cpld_designs\design2\design2.npl	- D ×
<u>File Edit V</u> iew P <u>r</u> oject <u>S</u> ource <u>P</u> rocess <u>M</u> acro <u>W</u> indow <u>H</u> elp	
	~
Sources in Project: design2 XC95108 PC84-XST VHDL Veledded (C.\mp\cpld_designs\design1\ledded.vhd) Snapshot View Snapshot View Clibrary View Processes for Current Source: Synthesize Synthesize Module Programming File File Process View	
Done: completed successfully.	N N
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 XC95108 PC84 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.

New	×
User Document VHDL Module Schematic Vhdl Library VHDL Package VHDL Test Bench Test Bench Waveform State Diagram	File Name: counter Logation: c:\tmp\cpld_designs\design2
< <u>B</u> ack <u>N</u>	ext > 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 VHDL Source				
Entity Name co	ounter			
Architecture Name Be	ehavioral			_
Port Name	Direction	MSB	LSB	<b>_</b>
clk	in			
count	out	3	0	
	in			_
	in			
	< <u>B</u> ack <u>N</u> ext >	Canc	el	Help

Click on Next and check the information about the module.

New Source Info	ormation						×
Project Navigat following specifi		e a new sk	eleton sou	irce with the			
Source Type: VI Source Name: co Entity Name: con Architecture Nam Port Definitions:	ounter unter						
	clk count	scalar vector:	3:0		in out		
Source Direct	ory:						
	< <u>B</u> a	ick	Finish	Cano	cel	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:

```
library IEEE;
 use IEEE.STD LOGIC 1164.ALL;
 use IEEE.STD LOGIC ARITH.ALL;
 use IEEE.STD LOGIC UNSIGNED.ALL;
 entity counter is
      Port ( clk : in std logic;
             count : out std logic vector(3 downto 0));
 end counter;
 architecture Behavioral of counter is
      signal cnt: std logic vector(27 downto 0);
 begin
      process(clk)
     begin
          if clk'event and clk='1' then
              cnt <= cnt + 1;
          end if.
      end process;
      count(3 downto 0) <= cnt(27 downto 24);</pre>
 end Behavioral;
💙 counter.vhd *
```

Line 12 declares a 28-bit signal, **cnt**, that is the current value of the counter. The process on lines 15-20 controls when counter increments. The condition clause of line 16 is only true when the value on the **clk** input goes from 0 to 1. Then the statement on line 17 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 22 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 XS95 Board which has a default frequency of 50 MHz. The LED display would be changing much too quickly at this frequency. By connecting the LED decoder to the upper four bits of the 28-bit counter, the display will only change once in every  $2^{24}$  clock cycles. So the LED display will change every  $2^{24}$  / (50 x 10<sup>6</sup>) = 0.336 seconds which is slow enough to be seen.

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.

😹 Xilinx - Project Navigator - c:\tmp\cpld_designs\design2\design2.npl - [counter.vhd]				
▶ File Edit View Project Source Process	<u>M</u> acro <u>W</u> indow <u>H</u> elp	- 8 ×		
	\$ E 🖪 🏊 🗞 😵 📢 🐰 Fe 💼 🗠 🗠 🖓 😋 n	•		
Sources in Project: design2 Counter (counter.vhd) deddd (C:\tmp\cpld_designs\desig Module Snapsho Library Processes for Current Source:	<pre>library IEEE; use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL; use IEEE.STD_LOGIC_UNSIGNED.ALL; entity counter is Port ( clk : in std_logic; count : out std_logic_vector(3 downto 0)) end counter; architecture Behavioral of counter is signal cnt: std_logic_vector(27 downto 0);</pre>	;		
Design Entry Utilities     Synthesize     View Synthesis Report     Analyze Hierarchy     Check Syntax     Implement Design     Generate Programming File	<pre>begin     process(clk)     begin         if clk'event and clk='1' then             cnt &lt;= cnt + 1;         end if;     end process;     count(3 downto 0) &lt;= cnt(27 downto 24);</pre>			
Process View	end Behavioral;			
Done: completed successfully		×		
Process "Check Syntax" is up to date.	Ln 21, Col 16	- 🔶 //.		

### 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 toplevel schematic. Before we can do this, we have to create schematic symbols for both the counter and LED decoder 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 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 XC95108 PC84 object and select New Source... from the pop-up menu. Then highlight the Schematic entry in the **New** window and name the schematic **disp\_cnt**. Then click on Next.

New	×
User Document VHDL Module Schematic Vhdl Library VHDL Package VHDL Test Bench Test Bench Waveform State Diagram	File Name: disp_ont Logation: c:\tmp\cpld_designs\design2
< <u>B</u> ack <u>N</u>	ext > 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 Source Information	×
Project Navigator will create a new skeleton source with the following specifications:	
Source Type: Schematic Source Name: disp_cnt	
Source Directory:	
< <u>B</u> ack Finish Cancel Help	

Now the disp\_cnt schematic object has been added to the Sources pane. Double-click it to open a schematic window.

Xilinx - Project Navigator - c:\tmp\cpld_de		- 🗆 ×
	( III 🔤 👧 🇞 🧣 🕺 🕺 🛍 🛍 🗠 🗠 🏘 😋 n	•
Sources in Project: design2 Counter (counter, vhd) disp_cnt (disp_cnt.sch) leddcd (C:\tmp\cpld_designs\desig Library Processes for Current Source: Design Entry Utilities Synthesize Design Entry Utilities Generate Programming File Process View	<pre>library IEEE; use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL; use IEEE.STD_LOGIC_UNSIGNED.ALL; entity counter is Port ( clk : in std_logic; count : out std_logic_vector(3 downto 0)) end counter; architecture Behavioral of counter is signal cnt: std_logic_vector(27 downto 0); begin process(clk) begin if clk'event and clk='1' then cnt &lt;= cnt + 1; end if; end process; count(3 downto 0) &lt;= cnt(27 downto 24); end Behavioral; .</pre>	
Done: completed successfully		
		×
Hierarchy is up to date.	Ln 21, Col 16	<i> 1</i> ,

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/cpld\_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.

	Xil Ei	inx I	_	_	_	_					Win	dow	H	elo																_	
		, . 2			<u>-</u> 10.	~ E			<u>•</u> ?	1.0				l K	) (	5a	黄		Ð	Q	X	C (H	0	) <b>(</b>	2		2 6	î d	2 🔁		
	ĸ	-	<b>j</b> :	_	_	2	~	þ		 £	퐌																		nches		-
		1																						- 1		C-1-					
ŀ		1			2	2			3			4			5		6			7			8	-1			egorie I Syml				
																										KC:	/tmp/	'opld	desig	ns/d	esi
	A																							A		Buf					
																										Cou	mpara unter				
		•				•	•	•						•	•			•				•	•	H		Flip	coder _Flop				
																										Ger	neral				⊻
	8																							в		Sym	bols				
		•									•			•	•		•				•			H		COL	Inter		7		
						·																		Ш			dcd		.0		
	c																							с							
																								Ш							
	-	•				•	•				•			•	•		•	•			•	•	•	H							
			•			·	•											•													
	D																							D		Sym	bol <u>N</u>	ame	Filter		
		1			2	2			3			4			5		6			7			8			0.4-					
ļ						_																				_	ntatio				Ţ
	d	isp_(	ent.	sch	1	J																				Гно	tate O				<b>_</b>
																													[3	544,4	482]

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 result shown below.





Next, click on the <sup>1</sup> button to begin adding wires to the schematic.

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. Either click in the same endpoint or hit the ESC key to stop adding segments to the bus.



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.

Now click on the 🔤 button for adding I/O markers.



Then click on the other end of the newly-added bus segment to create 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			
⊡ <mark>Nets</mark> XLXN_4(7:0)		Net Attributes	
(1.0)	Name	Value	<u>N</u> ew
	Name	XLXN_4(7:0)	
	PortPolarity	Input	<u>E</u> dit Traits
			<u>D</u> elete
	OK	Cancel <u>Apply</u>	<u>H</u> elp

Replace the existing bus name with a seven-bit bus for driving the LED segments: **S(6:0)**. Then set the direction of the bus pins to Output.

🔀 Object Properties			×
Category			
⊡ Nets S(6:0)		Net Attributes	
0(0.0)	Name	Value	<u>N</u> ew
	Name	S(6:0)	
	PortPolarity	Output 🗸	<u>E</u> dit Traits
		Input	Delete
		Output Bidirectional	
		Didirectional 1%	
	OK	Cancel <u>A</u> pply	<u>H</u> elp

Next click on the OK button to close the **Object Properties** window.

🙀 Object Properties			×
<u>C</u> ategory			
□ Nets S(6:0)		Net Attributes	
-()	Name	Value	New
	Name	S(6:0)	
	PortPolarity	Output	<u>E</u> dit Traits
			Delete
I	1		
	ОК	Cancel <u>Apply</u>	<u>H</u> elp



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.



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 clicking on the Zoom In button to see an enlarged view of the area where the error lies.

🖁 Schematic	×	
Error No.	Error Msg	C <u>e</u> nter
1	Error: Pin 's(6:0)' is connected to a bus of a different width	Zoom <u>I</u> n
2	Error: Pin 'O(7:0)' is connected to a bus of a different width	Zoom <u>O</u> ut
		<u>C</u> lose
		<u>H</u> elp

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.



We could try solve these problems by using subsets of the buses or by adjusting the width of the output buffers. But the simplest solution is to remove the byte-wide output buffer and

replace it with a group of seven output buffers. To start this process, click on the button and then click on the OBUF8 symbol. Then press the delete key. Then click on the button and add bus segments to the schematic as shown below.



At this point it is a good idea to rename the bus connected to the LED decoder output so it has a less cumbersome name. Double-click on the bus and the **Object Properties** window will appear.

🔀 Object Properties			×
<u>C</u> ategory			
		Net Attributes	
	Name	Value	New
	Name	XLXN_11(6:0)	
	PortPolarity	Not a port	<u>E</u> dit Traits
			<u>D</u> elete
I	1		
	OK	Cancel <u>Apply</u>	<u>H</u> elp

Replace the automatically-assigned name for the seven-bit bus, **XLXN\_11(6:0)**, with the bus name **A(6:0)**. Then click on the OK button.

🔀 Object Properties			×
<u>C</u> ategory			
⊡Nets 		Net Attributes	
	Name	Value	New
	Name	A(6:0)	
	PortPolarity	Not a port	<u>E</u> dit Traits
			<u>D</u> elete
	OK	Cancel <u>A</u> pply	<u>H</u> elp
	h	<u>,</u>	

Next select a single-bit output buffer, obuf, from the list of symbols and drop seven of them into the schematic drawing area.



Now the output buffers have to be attached to the buses. Click on the button and attach seven bus taps to each bus as shown below. (Use the rotation button to rotate the bus tap symbol.)





Once all the bus taps are in place, attach the output buffers to the taps using wire segments.

Now the question becomes: "How do we know each output buffer is attached to the right LED decoder output and output pin?" The answer is: "We don't!" We have to manually set the connections to the buses to make sure they are correct. Double-click on the wire segment connecting the output buffer to the S(6:0) output bus. (Make sure you double-click the wire segment and not the bus tap symbol or the OBUF symbol.) The **Object Properties** window will appear with the name for the wire segment that was automatically assigned by the schematic editor.

🚜 Object Properties			×
<u>C</u> ategory			
□Nets XLXN_19		Net Attributes	
	Name	Value	New
	Name	XLXN_19	
	PortPolarity	Not a port	<u>E</u> dit Traits
			<u>D</u> elete
		_	
	OK	Cancel <u>A</u> pply	<u>H</u> elp

Change the name of the wire segment to S(0) which is the least-significant bit of the output bus. Then click on OK.

🙀 Object Properties			×
<u>Category</u>			
ENets		Net Attributes	
	Name	Value	New
	Name	S(0)	
	PortPolarity	Not a port	<u>E</u> dit Traits
			<u>D</u> elete
	1		
	ОК	Cancel Apply	<u>H</u> elp

Repeat the process to rename each wire segment as shown below. (The visible labels for each wire segment were added afterward. The wire segment labels will not be shown by the schematic editor.)



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

🔒 Schematic	Check Errors	×
Error No.	Error Msg	C <u>e</u> nter
1	No errors detected	Zoom <u>I</u> n
		Zoom <u>O</u> ut
		<u>C</u> lose
		<u>H</u> elp

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

Xilinx - Project Navigator - c:\tmp\cpld_de File Edit View Project Source Process	
	\$ E 🖪 🖪 🔁 😵 🛛 X 🖻 E 🗠 🗠 🖓 😋 n 💽
Sources in Project: design2 Counter (counter.vhd) leddcd (C:\tmp\cpld_designs\d Module Snapsho Library Processes for Current Source: Counter So	<pre>library IEEE; use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL; use IEEE.STD_LOGIC_UNSIGNED.ALL; entity counter is Port ( clk : in std_logic; count : out std_logic_vector(3 downto 0)); end counter; architecture Behavioral of counter is signal cnt: std_logic_vector(27 downto 0); begin process(clk) begin if clk'event and clk='1' then cnt &lt;= cnt + 1; end if; end process; count(3 downto 0) &lt;= cnt(27 downto 24); end Behavioral;</pre>
Process View	counter.vhd
Done: completed successfully	▼. ▼ ▼
Hierarchy is up to date.	Ln 21, Col 16 🖉 🥢

#### Checking the VHDL Syntax

We can check the entire design by highlighting the disp\_cnt object in the Sources pane and then double-clicking the Analyze Hierarchy process. This checks the VHDL for each module and their interconnections with each other. The 🖋 that appears after the Analyze Hierarchy process completes shows we have no syntax problems in our modules.



#### Constraining the Design

Before synthesizing the displayable counter, we need to assign the pins which the inputs and outputs will use. Highlight the disp\_cnt object in the Sources pane and then double-click the Edit Implementation Constraints (Constraints Editor) process. In the Ports tab of the **Constraint Editor** window that appears, set the pin assignments for the clock input and LED segment drivers as follows:

0> 0> 0 1> 0 2> 0 3> 0 4> 0 5> 0 6> 0	Port Direction NPUT DUTPUT DUTPUT DUTPUT DUTPUT DUTPUT DUTPUT DUTPUT DUTPUT	Location           P9           P21           P23           P19           P17           P18	Pad to Setup           N/A           N/A           N/A           N/A           N/A           N/A           N/A	Clock to Pad N/A N/A
I>     C       2>     C       3>     C       4>     C       5>     C	DUTPUT DUTPUT DUTPUT DUTPUT	P23 P19 P17	N/A N/A N/A	
2> 0 3> 0 4> 0 5> 0	DUTPUT DUTPUT DUTPUT	P19 P17	N/A N/A	
	OUTPUT OUTPUT	P17	N/A	
(4> ○ (5> ○ (6> ○) ○	OUTPUT			
<5> () <6> ()		P18	N12.0	
	OUTPUT		NA	
-		P14	N/A	
	OUTPUT	P15	N/A	_
I/O Configuration Options Prohibit I/O Locations	Pad Groups Group Name: Select Group:		Pad to Setup Clock to Pad.	
Global Ports A	dvanced	Misc		

Assigning the **clk** input to pin P9 lets us use the onboard oscillator of the XS95 Board to drive the counter. The output assignments connect the displayable counter to the seven-segment LED on the XS95 Board as in the previous design example.

## 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.

Xilinx - Project Navigator - c:\tmp\cpld_d	esigns\design2\design2.npl - [disp_cnt.vhf]
▶ File Edit View Project Source Process	Macro Window Help
	14 🗉 🖾 🎘 🛠 🕺 X 🖻 💼 🗠 🗠 🖓 😋 n 💽
Sources in Project: design2 Scounter (counter.vhd) disp_cnt (disp_cnt.sch) leddcd (C:\tmp\cpld_designs\d Module Snapsho Library Processes for Current Source: Design Entry Utilities Synthesize View Synthesis Report Analyze Hierarchy Compared to be a sign Generate Programming File	Vhdl model created from schematic C:\Xilinx_WebPACK\(* LIBRARY ieee; LIBRARY UNISIN; USE ieee.std_logic_1164.ALL; USE uwiSIM.Vcomponents.ALL; ENTITY OBUF8_MXILINX IS PORT ( I : IN STD_LOGIC_VECTOR (7 DOWNTO 0); O : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)); end OBUF8_MXILINX; ARCHITECTURE SCHEMATIC OF OBUF8_MXILINX IS ATTRIBUTE fpga_dont_touch : STRING; ATTRIBUTE fpga_dont_touch OF I_36_30 : LABEL IS "true ATTRIBUTE fpga_dont_touch OF I_36_31 : LABEL IS "true ATTRIBUTE fpga_dont_touch OF I_36_32 : LABEL IS "true ATTRIBUTE fpga_dont_touch OF I_36_33 : LABEL IS "true ATTRIBUTE fpga_dont_
ISE Auto-Make Log File	
	۲ ۲
For Help, press F1	Ln 1, Col 1
rorneip, piess ri	

#### Fitting the Logic Circuitry Into the CPLD

Once the netlist is synthesized, we can begin the process of fitting it into the CPLD. Before activating the fitting process, however, we will give the fitter some information on the speed of the CPLD we are targeting. The XC95108 on the XS95 Board has a -20 speed grade which means that the pad-to-pad delay through a single macrocell is 20 ns. To set the device speed, right-click on the Implement Design process and select the Properties item in the pop-up menu.



In the **Process Properties** window that appears, we set the speed to -20 in the Speed Grade field of the Design tab. Then we click on OK. There are many other parameters we can adjust to affect the fitting process, but we don't need to alter any of them from their default values for this design. (We would probably adjust these parameters if we were pushing the CPLD to the limit in terms of logic density or operating speed.)

Ρ	rocess Properties	×
	Design Basic Advanced	,
	Property Name	Value
	Implementation User Constraints File	
	Macro Search Path	
	Speed Grade	-20
	Implementation Template	-7
		-10
		-15
		-20
		~
Ľ		
	OK Cancel	<u>D</u> efault Help

Once the speed grade of the CPLD is set, we can double-click on the Implement Design process to initiate the fitting process. The  $\checkmark$  that appears indicates that the fitting process was successful.



#### **Checking the Fit**

After the fitting process is done, we can check the logic utilization by double-clicking on the Fitter Report process. At the top of the file we find:

cpldfit: version E.30		Xilinx Inc.
	Fitter Repor	t
Design Name: disp_cnt Device Used: XC95108-2 Fitting Status: Succes		Date: 10-21-2001, 11:49AM
*****	***** Resource Summary	*****
Macrocells Product	Terms Registers	Pins Function Block
Used Used	Used	Used Inputs Used
35 /108 ( 32%) 52 /54	0 ( 9%) 28 /108 ( 25%)	8 /69 ( 11%) 48 /216 ( 22%)

The displayable counter consumes 35 of the 108 macrocells: 28 for the four-bit counter and 7 for the LED decoder. Looking further, we find the pin assignments for the clock input and LED decoder outputs match the assignments we made in the Constraint Editor:

***************Resc	uwana IIa	ad by Cu	aacaaful	Jrr Mor	anad 1	[ agi g	*****	* * * * * * * * *
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA	ources us	ed by Su	ccessiui	ту мај	pped	LOGIC		
** LOGIC **								
Signal	Total	Signals	Loc	Pwr	Slew	Pin	Pin	Pin
Name	Pt	Used	200		Rate		Туре	Use
N52	1	24	FB3 18	STD	nace		(b)	(b)
N53	1	25	FB3 17	STD		31	(2) I/O	(b)
N54	1	26	FB3 16	STD		26	I/0	(b)
N55	1	27	FB3 15	STD		25	I/0	(b)
s<0>	4	4	FB3 11	STD	FAST		I/0	0
s<0>	4 3	4	FB3_11 FB3_12	STD	FAST		I/O	0
s<1>	3	4	FB3_12 FB3_8	STD	FAST		I/O	0
s<2>	2	4	FB3_0 FB3_5	STD	FAST		I/O I/O	0
s<3>	4	4	FB3_5 FB3_6	STD	FAST		I/O	0
			—					
s<5>	4	4	FB3_2	STD	FAST		I/O I/O	0
s<6>	4	4	FB3_3	STD	FAST	15	I/O	0
xlxi_1/cnt_0	1	1	FB2_18	STD		~ 4	(b)	(b)
xlxi_1/cnt_1	1	1	FB2_17	STD		84	I/O	(b)
xlxi_1/cnt_10	1	10	FB1_18	STD			(b)	(b)
xlxi_1/cnt_11	1	11	FB1_17	STD		13	I/O	(b)
xlxi_1/cnt_12	1	12	FB1_16	STD		12	GCK/I/O	(b)
xlxi_1/cnt_13	1	13	FB1_15	STD		11	I/O	(b)
xlxi_1/cnt_14	1	14	FB1_14	STD		10	GCK/I/O	(b)
xlxi_1/cnt_15	1	15	FB1_13	STD			(b)	(b)
xlxi_1/cnt_16	1	16	FB1_12	STD		9	GCK/I/O	GCK
xlxi_1/cnt_17	1	17	FB1_11	STD		7	I/O	(b)
xlxi_1/cnt_18	1	18	FB1_10	STD			(b)	(b)
xlxi_1/cnt_19	1	19	FB3_14	STD		24	I/O	(b)
xlxi_1/cnt_2	1	2	FB2_16	STD		83	I/O	(b)
xlxi_1/cnt_20	1	20	FB3_13	STD			(b)	(b)
xlxi_1/cnt_21	1	21	FB3_10	STD			(b)	(b)
xlxi_1/cnt_22	1	22	FB3_9	STD		20	I/O	(b)
xlxi_1/cnt_23	1	23	FB3_7	STD			(b)	(b)
xlxi_1/cnt_3	1	3	FB1_9	STD		6	I/O	(b)
xlxi_1/cnt_4	1	4	FB1_8	STD		5	I/O	(b)
xlxi_1/cnt_5	1	5	FB1_7	STD			(b)	(b)
xlxi_1/cnt_6	1	6	FB1 6	STD		4	I/O	(b)
xlxi <sup>1</sup> /cnt <sup>7</sup>	1	7	FB1 5	STD		3	I/O	(b)
xlxi <sup>1</sup> /cnt <sup>8</sup>	1	8	FB1 4	STD			(b)	(b)
xlxi <sup>1</sup> /cnt <sup>9</sup>	1	9	FB1 3	STD		2	I/O	(b)
			_					
** INPUTS **								
Signal			Loc			Pin	Pin	Pin
Name						#	Туре	Use
clk			FB1_12			9	GCK/I/O	GCK
			_					

End of Resources Used by Successfully Mapped Logic

Also note that there are 28 signals in addition to the input and outputs. There are the upper four output bits from the counter (**N26**, **N27**, **N28**, and **N29**) and the twenty-four lower bits of the counter (**xlxi\_1/cnt\_\***). They will not appear on the pins of the CPLD because their macrocells have been buried. In effect, a macrocell is buried when the output buffer from the macrocell to its associated I/O pin is placed in a high-impedance state.

#### Checking the Timing

We have the displayable counter synthesized and fitted to the XC95108 CPLD with the correct pin assignments. But how fast can we run the counter? To find out, double-click on the Generate Timing process.



After the static timing delays are calculated, double-click the Timing Report to view the results of the analysis.

Xilinx - Project Navigator - c:\tmp\cpld_de <u>File</u> Edit <u>View</u> Project <u>Source</u> Process 1	signs\design2\design2.npl - [disp_cnt.tim (READ ONLY)]
	E 🖪 📴 🗞 💡 📢 🕺 🏦 🖻 🖆 🗠 🖓 💽 n 💽
Sources in Project: design2 CS5108 PC84×ST VHDL disp_ent (disp_ent.sch) ledded (C:\tmp\cpld_designs\d Module Snapsho Library	Design: disp_cnt Device: XC95108-20-PC84 Speed File: Version 3.0 Program: Timing Report Generator: version E.30 Date: Sat Oct 20 18:57:09 2001 Performance Summary: Clock net 'clk' path delays:
Fit → C Lock Pins → C Generate Timing	Clock Pad to Output Pad (tCO) : 22.( Clock Pad 'clk' to Output Pad 's<0>' Clock to Setup (tCYC) : 20.( Clock to Q, net 'xlxi_1/cnt_0.Q' to TFF Setup(D) at 'NS&
🖹 🖋 Timing Report Analyze Posthigt S	Minimum Clock Period: 20.0ns
Generate Post-Fit Simula_ ⊕ Generate IBIS Model ↓	Maximum Internal Clock Speed: 50.01 (Limited by Cycle Time)
Process View	Counter.vhd disp_cnt.vhf disp_cnt.tim
Done: completed successfully	
For Help, press F1	Ln 1, Col 1 🛛 💽 🎢

The timing report contains the following information:

Minimum Clock Period: 20.0ns

Maximum Internal Clock Speed: 50.0Mhz (Limited by Cycle Time)

This table says that there is a 22.0 ns time delay between the rising edge on the clk input and a change in any one of the LED driver outputs. (This is the reason we changed the speed grade property in the fitter: so the reported delay would be accurate for our chip.) The minimum clock period is stated to be 20 ns so the design should run at 50 MHz.

#### Generating the Bitstream

Now we are ready to generate the bitstream for the displayable counter. To initiate the programmer, we highlight the disp\_cnt object in the Source pane and double-click on the Configure Device (iMPACT) process.



The **iMPACT** window will appear and once again it will try and fail to establish a connection with the CPLD through the various ports of the PC. As in the previous design example, just click on the OK button and proceed.

🔩 Untitled - iMPACT	
<u>Eile Edit Operations Output S</u>	<u>/iew H</u> elp
🗋 🗅 🚅 🔚 👗 🖬 💼 👹	2 梁 🔢 22 🗇 🖽 鸀 📢
Boundary Scan Slave Se	rial Select Map
	Inx iMPACT Communications with the cable could not be established. Please check the cable connections and cable power source. OK
Cable connection failed. Connecting to cable (COM4 Port).	▲ ▲
Cable connection failed.	
Elapsed time = 24 sec.	
Cable autodetection failed. =>	
T	×
Connecting to the selected cable	No Connection

Click the Cancel button in the next window and proceed.

Cable Communication Setup	? ×
Communication Mode Parallel Multilinx/USB	O Multilinx/Serial
Baud Rate: 57600	Port: Ipt1
KCancel	<u>H</u> elp

The **iMPACT** window now shows the JTAG chain of chips that are to be programmed. We only have one chip in our LED decoder design, so only one XC95108 CPLD is shown. Click on the

xc95108 icon to highlight it. Then continue by selecting an SVF file as the destination for the bitstream so we can use GXSLOAD for programming the CPLD in the XS95 Board.

🖵 Untitled - iMPACT			
<u>File Edit Operations O</u>	utput ⊻iew <u>H</u> elp		
Boundary Scan	Cable <u>S</u> etup Cable <u>R</u> eset	∄ ⊞   <b>∰</b>   <b>№</b> ap	
	Cable <u>D</u> isconnect		
	<u>U</u> se Cable		
	Use <u>F</u> ile ▶	S <u>V</u> F File ►	<u>C</u> reate SVF File
		ST <u>A</u> PL File →	Append to SVF File い Close SVF File
xc95108			
disp_cnt.je TDO			
done.			<b>•</b>
Device #1 selected			
Device #1 selected =>			
1			
Create an SVF file and direc	t subsequent operations to it	No Connection	

Now a window appears where we can enter the name for the file that will hold the bitstream. We can click on Save to accept the default name of disp\_cnt.svf.

Create a New	≠SVF File					?×
Savejn: 🔂	design2	•	£	<u></u>	<del>č</del> *	<b></b>
_ngo						
🛄 xst						
File <u>n</u> ame:	disp_cnt			_		Save N
Save as <u>t</u> ype:				Ţ		Cancel
0010 do <u>o</u> po.	1041 ( 100( .341)					

We then proceed to generate the bitstream by selecting the Operations→Program menu item.

🔩 Untitle	d - impact		
<u>F</u> ile <u>E</u> dit	Operations Output View Help		
Bounda	<u>E</u> rase	(‡) [⊞   <b>∰</b>   <b>№</b> ? /lap	
TDI-	Eunctional Test Blank Check Readback Program XPLA UES Get Device ID Get Device Checksum Get Device Signature/Usercode Get XPLA device UES		
TDO-	IDCODE <u>L</u> ooping		
done.  Device #1 s Device #1 s =>	elected		× ×
Programs th	e selected devices	File   SVF	11.

A **Program Options** window appears where we can set the Erase Before Programming option for the generated bitstream. Once we click OK in the **Program Options** window, the bitstream generation process begins.

Program Options	? ×
Erase Before Programming	Eunctional Test
□ Verify □ <u>R</u> ead Protect □ Write Protect	PROM Skip user array
Virtex2	Load Fpga <u>P</u> arallel Mode     Use D4 for CF
PROM Usercode (8 Hex Chars)	
XPLA UES: Enter up to 0 ch	aracters
	<u>H</u> elp

The progress is reported in the lower pane of the **iMPACT** window:



Once the bitstream file is generated, we click on File $\rightarrow$ Exit to close the window.

## Downloading the Bitstream

Now we download the bitstream file into the CPLD of the XS95 Board. We double click the



GXSLOAD icon to bring up the **gxsload** window. Then we drag the disp\_cnt.svf file from the C:\tmp\cpld\_designs\design2 folder and drop it into the **gxsload** window.

🔁 design2	🕻 🚺 gxsload 📃 🗖 🗙
<u>File Edit View Go</u> F <u>a</u> vorites <u>H</u> elp	Board Type XS95-108 - Load
↔ • → · 🖻   ½ 🖻 🗳 🖂 🗙	Port LPT2 Exit
Address 🗀 C:\tmp\cpld_designs\design2	
Image: Second system     Image: Second system       Image: Second system     Ima	FPGA/CPLD     RAM     Flash/EEPROM       disp_cnt.svf
i isp_ont.prj i isp_ont.vhf I disp_ont.rpt i isp_ont.vm6	
≝ disp_ont.rpt 🔊 disp_ont.vm6	Low Address
	Upload Format HEX 💽 🗀 HEX 💽 🗀
1 object(s) selected My Co	

Click the Load button and the XC95108 CPLD programming starts and completes in about a minute.

🔀 gxsload 📃 🗆 🔪	3
Board Type XS95-108	
Port LPT2  Exit	
FPGA/CPLD RAM Flash/EEPROM	
High Address	
Low Address	
Upload Format 🛛 💌 🎦 🛏 💌 🗀	
Configure CPLD	
Downloading disp_cnt.svf	

## **Testing the Circuit**

Once the XC95108 CPLD on the XS95 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, 0, 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.