





© 2001 by X Engineering Software Systems Corp., Apex, North Carolina 27502

All rights reserved. No part of this text may be reproduced, in any form or by any means, without permission in writing from the publisher.

The author and publisher of this text have used their best efforts in preparing this text. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this text. The author and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

XESS, XS40, and XS95 are trademarks of X Engineering Software Systems Corp. XILINX, Foundation, XC4000, and XC9500 are trademarks of XILINX Corporation. Other product and company names mentioned are trademarks or trade names of their respective companies.

The software described in this text is furnished under a license agreement. The software may be used or copied under terms of the license agreement.

3

Hierarchical Design

In this chapter you will learn how to:

- Create VHDL designs composed of a hierarchy of VHDL modules.
- Create hierarchical VHDL designs which also include schematics in the hierarchy.
- Create hierarchical schematics that include VHDL modules.
- Use LogiBlox to create modules for incorporation into hierarchical designs.

Hierarchy

Most complex systems possess a hierarchical structure. Hierarchy arises in man-made systems because people are good at decomposing problems into simpler problems whose solutions can be combined into a complete solution. In an eight-bit adder, for example, you might decompose the operation into a set of one-bit additions. Then you could design a logic circuit that adds binary bits using AND, OR, and NOT gates. Finally, you could combine the one-bit adder modules into a single eight-bit adder. But it doesn't have to stop there because you could use the eight-bit adder to build a 32-bit adder as part of a microprocessor chip that resides on a circuit board in a computer attached to a network...

Some of the advantages of building circuits from a hierarchy of modules are:

Design re-use: A module can be re-used across multiple designs so you do less work overall.

Information hiding: Encapsulating a circuit into a module lets you ignore its internal operational details while allowing you to concentrate on the interaction of the module's inputs and outputs with the rest of the system.

Replication: Building a large circuit by duplicating a small group of modules is much easier than building a large circuit by stitching together a large number of primitive gates.

In this chapter we will see how the Foundation software supports hierarchical design techniques. The example design I will use consists of a 28-bit binary counter whose four upper bits are displayed as a hexadecimal digit on a seven-segment LED (see Figure 7). When driven by a 50 MHz clock, the displayed digit will change every 2^{24} / 50,000,000 =

0.34 seconds. Foundation lets you describe the root of the design using an HDL (either VHDL or Verilog) or as a schematic, and you have this same flexibility with each module in the lower levels of the hierarchy. So you can change your design style to match the type of circuit you are building. We will build the design in three different ways:

- 1. The root and all lower level-modules are described using VHDL;
- 2. The root is described using VHDL, but the lower-level modules are designed using both VHLD and schematics.
- 3. The root is designed as a schematic that contains VHDL and schematic-based lower-level modules.



Figure 7: Design hierarchy for a counter display.

Hierarchical VHDL-Based Design

The first design (*dsgn3_1*) is started as an HDL-based project.

New Proje	ect		×
Name:	dsgn3_1		OK
Directory:	C:\PRAG21I		Cancel <u>B</u> rowse
Туре:	F2.1i	•	<u>H</u> elp
Flow:	C <u>S</u> chematic	● HDL	

Creating the VHDL Files for the Lower-Level Modules

We start by using the HDL Editor to design the lower-level modules for the 28-bit counter and the seven-segment LED decoder.

🐌 dsgn3_1 - design not in	nplemented - Project Manager	_ 🗆 🗵
<u>File D</u> ocument <u>V</u> iew <u>P</u> roj	ect <u>S</u> ynthesis <u>I</u> mplementation <u>T</u> ools <u>H</u> elp	
Files Versions	Flow Contents Reports	
🖃 🗖 dsgn3_1		-
🔤 🖯 dsgn3_1	dsgn3_1	-
	HDL Editor	
	IMPLEMENTATION ? VERIFICATION	
Pcm : Start Xilinx Found	ation F2.1i - Messages - Thu Mar 01 20:53:03 2001	
	project: c:\prag21i\dsgn3_1	
	c:\prag21i\dsgn3_1	<u> </u>
Console / HDL Errors / HI	DL Warnings / HDL Messages /	<u> </u>
Ready		

The VHDL code for the LED decoder is shown in Listing 1. The code looks very similar to what we saw in Chapter 1 with the following differences:

- Line 3: A new package from the IEEE library is used. The numeric_std package gives us access to some new VHDL types such as SIGNED and UNSIGNED bit vectors and the arithmetic operations that act on them.
- Lines 5–12: A new package (leddcd_pckg) is defined that contains a single component declaration for the LED decoder. The component declaration tells other VHDL modules about the types of inputs and outputs used to interface to the LED decoder.
- Line 12: The four-bit d input to the LED decoder has been declared as an UNSIGNED bit vector rather than as a STD_LOGIC_VECTOR. This will simplify the interface with the 28-bit counter module which also outputs UNSIGNED values.

Listing 1: VHDL code for the seven-segment LED decoder module.

```
1 library IEEE;
```

```
2 use IEEE.std_logic_1164.all;
```

```
3 use IEEE.numeric_std.all;
© 2001 by XESS Corp.
```

```
1
 2
     package leddcd pckg is
 3
 4
     component leddcd
 5
         port (
 6
             d: in UNSIGNED (3 downto 0);
 7
             s: out STD LOGIC VECTOR (6 downto 0)
 8
         );
 9
     end component;
10
11
     end leddcd pckg;
12
13
14
     library IEEE;
15
     use IEEE.std_logic_1164.all;
16
     use IEEE.numeric_std.all;
17
18
     entity leddcd is
19
         port (
20
             d: in UNSIGNED (3 downto 0);
21
             s: out STD LOGIC VECTOR (6 downto 0)
22
         );
23
     end leddcd;
24
25
     architecture leddcd arch of leddcd is
26
     begin
27
         with d select
28
             s <= "1110111" when "0000", -- 0
29
                  "0010010" when "0001", -- 1
30
                   "1011101" when "0010", -- 2
31
                  "1011011" when "0011", -- 3
32
                   "0111010" when "0100", -- 4
33
                  "1101011" when "0101", -- 5
34
                  "1101111" when "0110", -- 6
35
                  "1010010" when "0111", -- 7
36
                  "1111111" when "1000", -- 8
37
                  "1111011" when "1001", -- 9
38
                  "1111110" when "1010", -- A
39
                  "0101111" when "1011", -- b
40
                  "1100101" when "1100", -- C
41
                  "0011111" when "1101", -- d
42
                   "1101101" when "1110", -- E
43
                   "1101100" when others; -- F
44
     end leddcd arch;
```

In the **HDL Editor** window, select the File → Save As menu item and save the LED decoder VHDL into the leddcd.vhd file in the *dsgn3_1* project directory.

Save As			? ×
Save jn: 🔄 dsgn3_1 💽 🗈		Ċ	
dsgn3_1			
1			
File name: leddcd.vhd			Save
Save as type: VHDL Files (*.vhd;*.vhi;*.vho)	•		Cancel

The VHDL code for the counter is shown in Listing 1. The important parts of the code are as follows:

- Line 3: Once again we access the numeric_std package. We will need the arithmetic addition operator to increment the counter value.
- Lines 5–12: Another new package (cntr_pckg) is defined that contains a single component declaration for the counter.
- Lines 25–34: The counter interface consists of an input to reset the counter to zero, a clock input that increments the counter value on each rising edge, and an UNSIGNED bit vector that outputs the current value of the counter. Note that the number of counter output bits is determied by the generic parameter LENGTH declared on lines 26–28. The cntr module is customized when it is instantiated for a particular application by specifying the value of LENGTH.
- Line 37: An internal UNSIGNED bit vector is declared to hold the value of the counter.
- Lines 39–49: The actual counting operation is specified in the COUNT process. The process is triggered by changes in the clk input (line 39) and the counter changes value on the rising edge of clk (line 42). If the rst input is high on a rising clock edge, then the counter value is cleared to zero (lines 43–44). (The TO_UNSIGNED function from the numeric_std package converts an integer into an UNSIGNED bit vector.) If the rst input is not high, then the value in the counter register is incremented by one (lines 45–46).

Line 51: The value in the counter register is placed on the outputs.

Listing 2: VHDL code for the counter module.

1 library IEEE; 2 use IEEE.std_logic_1164.all; © 2001 by XESS Corp.

```
3
    use IEEE.numeric std.all;
 4
 5
     package cntr pckg is
 6
 7
    component cntr
 8
          generic (
9
                LENGTH: natural -- number of bits in counter
10
          );
11
        port (
          rst: in STD_LOGIC; -- synchronous reset
clk: in STD_LOGIC; -- counter clock
12
13
14
          cnt: out UNSIGNED(LENGTH-1 downto 0) -- counter output
15
        );
16
     end component;
17
18
     end cntr pckg;
19
20
21
     library IEEE;
22
    use IEEE.std logic 1164.all;
23
    use IEEE.numeric std.all;
24
25 entity cntr is
26
         generic (
27
                LENGTH: natural -- number of bits in counter
28
          );
29
       port (
          rst: in STD_LOGIC; -- synchronous reset
clk: in STD_LOGIC; -- counter clock
30
31
32
          cnt: out UNSIGNED(LENGTH-1 downto 0) -- counter output
33
        );
34
     end cntr;
35
36
     architecture cntr arch of cntr is
37
     signal cnt r: UNSIGNED(LENGTH-1 downto 0); -- counter register
38
    begin
39
           COUNT: process(clk)
40
           begin
41
                 -- change counter only on rising clock edges
42
                 if (clk'event and clk='1') then
43
                       if rst='1' then -- synchronous reset to 0
44
                            cnt r <= TO UNSIGNED(0,LENGTH);</pre>
45
                                      -- otherwise, increment counter
                       else
46
                             cnt r <= cnt r + 1;
47
                       end if;
48
                 end if;
49
           end process COUNT;
50
51
          cnt <= cnt_r; -- output register contents</pre>
52
     end cntr arch;
```

In the **HDL Editor** window, select the File→Save As menu item and save the counter VHDL into the cntr.vhd file in the *dsgn3_1* project directory.

Save As					?×
Save jn: 🔁	dsgn3_1	• 🗈		r	
dsgn3_1					
🚞 xproj					
≣ leddcd.vhd	I				
I					
File <u>n</u> ame:	ontr. vhd				Save
Save as <u>type</u> :	VHDL Files (*.vhd;*.vhi;*.vho)		•		Cancel

Adding a New Library to the Project

Now we have the lower-level modules defined, but we still have to add them to the project so they can be accessed by the root module. To do this, we will create a new library and then add the modules to the library. Select the Synthesis \rightarrow New Library... menu item to start this step.

🐌 dsgn3_1 - design not imple	emented - Project Manager	<u> </u>
<u>File D</u> ocument <u>V</u> iew <u>P</u> roject	Synthesis Implementation Tools Help	
D 🕞 🖯 🚯 🔍 🖤	Add Source File(s)	
Files Versions 1 □• □ dsgn3_1 □· □ dsgn3_1	A <u>n</u> alyze All Sources Eorce Analysis of All Sources Synthesize	÷
	New Library Options DESIGN ENTRY ?	
		•
Pcm : EXIT: Library Manage	r	<u> </u>
Pcm : Execute hde -p 2208 Pcm : Update: C:\Prag21i\tr	np\dsgn3_1\cntr.vhd (0, 0)	
Console / HDL Errors / HDL V		<u> </u>

Specify the library name as xslib in the New Library window and click on the OK button.

New Library	×
Library Name:	К
xslib	<u>C</u> ancel
	Help

Now you will see the xslib library has been added to the left-hand **Hierarchy** pane of the **Project Manager** window. There isn't anything in this new library yet, but we will fix that by right-clicking on the xslib icon and selecting the Add Source Files to "xslib"... menu item.

🕸 dsgn3_1 - design not implemented - Project Manager	. 🗆 🗵
<u>File D</u> ocument <u>View Project</u> Synthesis Implementation <u>I</u> ools <u>H</u> elp	
Files Versions Flow Contents Reports	
Brein degn3_1	-
Slib degn3_1	
Analyze	
Eorce Analysis	
Synthesize	
Add Source Files to "xslib"	
New Library	
View library Report	
Remove Del	
IMPLEMENTATION ? VERIFICATION	
	-
Pcm : EXIT: Library Manager	-
Pcm : Execute hde -p 2208	
Pcm : Update: C:\Prag21i\tmp\dsgn3_1\cntr.vhd (0, 0)	
Console / HDL Errors / HDL Warnings / HDL Messages /	
Add Source Files	

Highlight the cntr.vhd and leddcd.vhd files in the **Add Document** window that appears and then click on the Open button.

Add Documer	nt					?×
Look in: 🔂	dsgn3_1	•	£		<u>e</u> *	
dsgn3_1						
🚞 xproj						
entr.vhd leddcd.vhd						
	-					
I						
File <u>n</u> ame:	"leddod.vhd" "ontr.vhd"					<u>Open</u>
Files of <u>type</u> :	HDL (*.VHD;*.VER;*.VE;*.V)			•		Cancel
						<u>H</u> elp

The modules for the LED decoder and counter will be added to the xslib library. You can click on the + icon to the left of the xslib icon to view what modules are included in the library.

😓 dsgn3_1 - design not implemented - Project Manager					
<u>File D</u> ocument <u>V</u> iew <u>P</u> roje	ect <u>S</u> ynthesis <u>I</u> mplementation <u>T</u> ools <u>H</u> elp				
Files Versions	Flow Contents Reports	-			
⊡ [⊖] xslib ⊕ ⊈ y leddcd.vhd	dsgn3_1				
⊞ ⊴v cntr.vhd ⊂ Casgn3_1					
	DESIGN ENTRY				
	SYNTHESIS ? SIMULATION				
		<u>.</u>			
	11i\dsgn3_1\cntr.vhd	_			
Dpm : Done Pcm : Synopsys server i	nitialization	_			
Console / HDL Errors / HD		<u> </u>			
Ready					

Creating the Root Module

Now we have to build the root module that combines the counter and LED decoder modules to create the complete circuit. The VHDL code for the cntdisp module has the following features:

- Line 1: XSLIB is now included in the list of libraries used for this design.
- Lines 4–5: Each lower-level module in the library declared its own package, so we have to explicitly declare that we are going to use all the components in each package.
- Line 16: A constant for the number of bits in the counter is declared and set to 28.
- Line 17: An internal 28-bit bit vector is declared. The upper four bits of this vector will be used to transfer the upper counter bits to the LED decoder module.
- Lines 19–20: The counter module is instantiated. The generic length parameter is set to 28 and the reset and clock inputs of the counter module are attached to the

reset and clock inputs to the root module. The counter outputs are attached to the internal bit vector in the root module.

Line 21: The LED decoder module is instantiated. The upper four bits of the counter value are passed into the LED decoder and the outputs of the decoder are connected to the root-level outputs.

🔲 cr	ntdisp.vhd - HDL Editor
<u>F</u> ile	<u>E</u> dit <u>S</u> earch <u>V</u> iew Synthesis <u>P</u> roject <u>T</u> ools <u>H</u> elp
1	library IEEE,XSLIB;
2	use IEEE.std_logic_1164.all;
3	use IEEE.numeric_std.all;
4	<pre>use XSLIB.cntr_pckg.all;</pre>
5	<pre>use XSLIB.leddcd_pckg.all;</pre>
6	
7	entity cntdisp is
8	port (
9 10	rst: in STD_LOGIC; synchronous reset clk: in STD LOGIC; counter clock
11	s: out STD LOGIC; Counter Clock s: out STD LOGIC VECTOR(6 downto 0) outputs to LED segments
12);
13	end cntdisp;
14	end cheursp,
15	architecture cntdisp arch of cntdisp is
16	constant length: natural := 28;
17	<pre>signal cnt: UNSIGNED(length-1 downto 0);</pre>
18	begin
19	u0: cntr generic map(LENGTH=>length)
20	<pre>port map(rst=>rst, clk=>clk, cnt=>cnt);</pre>
21	u1: leddcd port map(d=>cnt(length-1 downto length-4), s=>s);
22	end cntdisp_arch;
	F
0 er	ror(s) 0 warning(s) found
Ë,	
Read	y Ln 1, Col 20 VHDL

The root module is stored in the cntdisp.vhd file and then that file is added to the project hierarchy. With all the source files in place, we can check for any VHDL syntax errors by selecting the Synthesis→Force Analysis of All Sources menu item. In this case there are no errors (as indicated by the green checkmarks by each source file name in the **Project Hierarchy** pane.) If errors were found, you could double-click the marked files to open them with the HDL Editor and make the necessary fixes.



Synthesizing the Netlist

Once we know there are no syntax errors, we can run the synthesis tools to extract a netlist for the circuit.

🐌 dsgn3_1 - design not in	nplemented - Project Manager	<u> – – ×</u>
<u>File D</u> ocument <u>View</u> Proje	ect <u>S</u> ynthesis <u>I</u> mplementation <u>T</u> ools <u>H</u> elp	
D 🕞 🖯 🐽 🔍		
Files Versions	Flow Contents Reports	
⊡ • 🗖 dsgn3_1 ⊕ • 🚭 cntdisp.vhd ⊡ • ⊖ xslib	dsgn3_1	÷
⊕ ⊈vleddcd.vhd ⊕ ⊈v <mark>cntr.vhd</mark> — ⊖ dsgn3_1	DESIGN ENTRY	
Usgn3_1		
		-
Dpm : Done	Note and 2. Alternative of (2. 0)	-
Pcm : Update: c:\prag21 Pcm : Synopsys server in	i\dsgn3_1\cntr.vhd (0, 0) nitialization	_
Console / HDL Errors / HD		<u> </u>
Ready		

We are going to target this design to the XS40-005XL Board so set the target device as shown below in the **Synthesis/Implementation settings** window. Then pull-down the list of modules in the Top level field and highlight the cntdisp entry. This tells the synthesizer tools that the **cntdisp** module is the root of the design.

Synthesis/Impl	ementatio	n settin	gs	×
Top level:	cntdisp		•	<u>R</u> un
Version name:	cntdisp cntr	4		ОК
Synthesis Settin	leddod gs:		SET	<u>C</u> ancel
				<u>H</u> elp
_ Target Device				
Family: XC4	DOOKL	•		
Device: 4005	XLPC84	•	Speed:	xl-3 🔹
Edit Synthesi				ation
🗖 Auto Run Ii	mplementati	on tools		
Physical Imple	mentation s	ettings —		
Revisi	ion name:	rev1		<u>O</u> ptions
Contro	ol Files:	SE	T	

After clicking on the Run button, the synthesis tools will process the VHDL in the three source files to create a netlist.

Create Version			
Mapping combinational logic in design '/ver1-Optimized'			
Cancel			

Assigning the I/O Signals to the FPGA Pins

Before mapping the synthesized netlist to the FPGA, we need to specify the pin assignments for the inputs and outputs of the circuit. The pins on the FPGA of the XS40 Board that are connected to the clock oscillator and the seven-segment LED are shown in Figure 8. We will also control the reset input of the circuit using the **D0** pin of the parallel port. That way we can reset the circuit using the PC attached to the XS40 Board.



Figure 8: Connection of the programmable oscillator, parallel port, and LED digit to the pins of the FPGA on the XS40 Board.

The pin assignments corresponding to Figure 8 are stored in the dsgn3_1.ucf file.

🔚 dsgn3_1.ucf - HDL Editor	
<u>File Edit Search View Synthesis Project</u>	<u>T</u> ools <u>H</u> elp
<pre>1 het rst loc=p44; 2 net clk loc=p13; 3 net s<0> loc=p25; 4 net s<1> loc=p26; 5 net s<2> loc=p24; 6 net s<3> loc=p20; 7 net s<4> loc=p23; 8 net s<5> loc=p18; 9 net s<6> loc=p19; 18</pre>	▲
·	
For Help, press F1	Ln 1, Col 1 TEXT

Implementing the Design

Once the pin assignments are in place, we can start the implementation tools.



Press the SET button in the **Synthesis/Implementation settings** window so the location of the pin assignments can be specified.

Synthesis/Implementation settings	×			
Top level: cntdisp	<u>▼ R</u> un			
Version name: ver1	ОК			
Synthesis Settings: SET	<u>C</u> ancel			
	<u>H</u> elp			
Target Device				
Family: XC4000XL 💽				
Device: 4005XLPC84 Sp	eed: 🛛 🚽			
Edit Synthesis/Implementation constra	aints			
☐ View Estimated Performance after Optimization				
✓ Auto Run Implementation tools				
Physical Implementation settings				
Revision name: rev1	<u>Options</u>			
Control Files: SET	Į			
· · · · · · · · · · · · · · · · · · ·	V			

Select Custom from the Constraints file field of the Settings window.

S	ettings	×			
	Implementation control files				
	Current Revision Control File Settings:				
	Use Constraints file from: None				
	Copy Guide file from:				
	Copy Floorplan files from: None				
	Current Revision Control file use:				
	OK Cancel <u>H</u> elp				

The name of the dsgn3_1.ucf file will already be listed in the Constraints File field of the **Custom** window that appears, so just click on the OK button.

Custom	×
Constraints File: dsgn3_1.ucf	Browse
<u> </u>	<u>H</u> elp

After returning to the **Synthesis/Implementation settings** window, click on Run to initiate the implementation process.

Synthesis/Implementation settings	×
Top level: cntdisp	Bun
Version name: ver1	ок
Synthesis Settings: SET	<u>C</u> ancel
	<u>H</u> elp
Target Device	
Family: XC4000XL	
Device: 4005×LPC84 Speed:	xl-3 🔽
Edit Synthesis/Implementation constraints	
🔲 View Estimated Performance after Optimizati	on
✓ Auto Run Implementation tools	
Physical Implementation settings	
Revision name: rev1	<u>O</u> ptions
Control Files:	

All the steps in the implementation process should complete without errors.

📲 dsgn3_1 (ver1->rev1) - Flow Engine	
Elow Yiew Setup Utilities Help	
XC4000XL Design Flow (rev1) Status: 0K	
Translate Map Place&Route Timing (Sim) Configure	
Completed Completed Running	
3 input LUTs: 0 Number of bonded IOBs: 9 out of 65 13% IOB Flops: 0 IOB Latches: 0 Number of clock IOB pads: 1 out of 12 8% Number of BUFGLSs: 1 out of 8 12% Total equivalent gate count for design: 567 Additional JTAG gate count for IOBs: 432 Writing design file "map.ncd" Mapping completed. See MAP report file "map.mrp" for details.	
par -w -ol 2 -d 0 map.ncd dsgn3_1.ncd dsgn3_1.pcf	-
For Help, press F1 XC4005XL-3-PC84 dsgn3_1.ucf	Π.

Project Manager	×
Flow Engine ver1->rev1 Completed Successfu	dly.
ОК	

The Project Manager window should look as follows after the implementation tools have terminated successfully.



Downloading and Testing the Design

At this point, the final bitstream for downloading into the XS40-005XL Board is available. Open the directory containing the *dsgn3_1* project files and drag-and-drop the dsgn3_1.bit file into the **gxsload** window. The bitstream will download into the XS40 Board attached to the parallel port.

gxsload gxsload Image: SVF, .HEX, and .EX0 files here to download to the XS or XSV Board	Gasgn3_1 File Edit Yiew Go Favorites Help Help <th></th>	
Recent Files:	dpm_net dsgn3_1.alb leddcd.vhd dsgn3_1 dsgn3_1 dsgn3_1.bak logiblox.ini lib dsgn3_1.bt time_sim.edn xproj dsgn3_1.ll sproj.ini cntdisp.er dsgn3_1.prj dsgn3_1.ucf cntdisp.vhd dsgn3_1.XNF cntdisp.vhd dsgn3_1.XNF contr.er specess.ini	
Reload EEPROM Port LPT1 -	Centr.log ileddcd.er Centr.vhd ileddcd.log Contr.vhd ileddcd.log Computer	

If pin D0 of the parallel port is at logic 1 after the downloading completes, the counter will be held in the reset state so only a static O is displayed. To release the reset, open the gxsport window and click on the D0 button until it displays a 0.

🗶 gxsport	_ 🗆 🗵
1 1 1 1 1 1 1 0 D7 D6 D5 D4 D3 D2 D1 D0 Strobe Image: Count Port LPT1 Image: Count Port LPT1	Exit

Then click on the Strobe button so the logic 0 value is output on the D0 pin of the parallel port.

🗶 gxspor	t			_ 🗆 🗵
1 1	1 1	1 1 1	0	Exit
D7 D6	D5 D4	D3 D2 D1	DO	
Strobe	☐ <u>C</u> ount	Port LP1	1 -	

Now you should observe the seven-segment LED running through the sequence: O, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, C, d, E, F, ... with each digit being displayed for roughly 1/3 seconds.

Consolidating the Packages

It can be inconvenient to place each module in a package and then have to explicitly include each package in the root module. Instead, you can create a single VHDL file that contains the package declarations from each of the other modules, and then just include this single module in the root. This was done for project **dsgn3_1a** as shown below.



The package declarations were removed from cntr.vhd and leddcd.vhd and the component declarations from each file were incorporated into a single package in the xs_pckg.vhd file (Listing 2).

Listing 2: Consolidated package declaration for the counter and LED decoder modules.

```
1
     library IEEE;
2
     use IEEE.std logic 1164.all;
 3
     use IEEE.numeric std.all;
 4
 5
     package xs pckg is
6
7
     component leddcd
8
         port (
9
             d: in UNSIGNED (3 downto 0);
10
             s: out STD LOGIC VECTOR (6 downto 0)
11
         );
12
     end component;
     © 2001 by XESS Corp.
```

```
13
14
    component cntr
15
          generic (
16
                LENGTH: natural -- number of bits in counter
17
          );
18
        port (
                               -- synchronous reset
19
          rst: in STD LOGIC;
20
            clk: in STD LOGIC;
                                -- counter clock
21
            cnt: out UNSIGNED(LENGTH-1 downto 0) -- counter output
22
        );
23
    end component;
24
25
    end xs pckg;
```

Then the single xs_pckg package is included on line 4 in the root module (Listing 3).

Listing 3: VHDL source for the root module of dsgn3_1a.

```
1
     library IEEE, XSLIB;
 2
     use IEEE.std logic 1164.all;
 3
     use IEEE.numeric std.all;
 4
     use XSLIB.xs pckg.all;
 5
 6
     entity cntdisp is
 7
         port (
 8
           rst: in STD_LOGIC; -- synchronous reset
  clk: in STD_LOGIC; -- counter clock
 9
10
              s: out STD_LOGIC_VECTOR(6 downto 0)
                                                       -- outputs to LED
11
     segments
12
         );
13
     end cntdisp;
14
15
     architecture cntdisp arch of cntdisp is
16
     constant length: natural := 28;
17
     signal cnt: UNSIGNED(length-1 downto 0);
18
     begin
19
     u0: cntr generic map(LENGTH=>length)
20
                  port map(rst=>rst, clk=>clk, cnt=>cnt);
21
     u1: leddcd port map(d=>cnt(length-1 downto length-4), s=>s);
22
     end cntdisp arch;
```

Hierachical VHDL Design with Schematic-Based Modules

In the *dsgn3_2* project, we will replace the cntr.vhd module with a counter desscribed by schematics. Once again, the design is initiated as an HDL-based project.

New Proje	ect		×
Name:	dsgn3_2		ОК
Directory:	C:\PRAG21I		Cancel Browse
Туре:	F2.1i	•	<u>H</u> elp
Flow:	C <u>S</u> chematic	€ HDL	

Then we can add the xslib library and add the leddcd.vhd file to it as we did in the *dsgn3_1* project.

🕸 dsgn3_2 - design not implemented - Project Manager		
<u>File D</u> ocument <u>View</u> Project	Synthesis Implementation <u>T</u> ools <u>H</u> elp	
Files Versions	Flow Contents Reports	
⊡-⊡ dsgn3_2 ⊡-⊡ xslib ⊡-≦vleddcd.vhd	dsgn3_2	1
🖯 dsgn3_2		
		•
Pcm : Execute c:\fndtn\active\exe\sc.exe		
Pcm : START: Schematic Editor		
Console / HDL Errors / HDL Warnings / HDL Messages /		
Ready		

Adding a Predefined Library to the Project

Next we need to draw the schematic for the counter. But we need a library of parts with which to build the counter. Schematic part libraries are tied specific device families, so we need to add the appropriate part library to our project. Once again we will target the XS40-005XL Board with the XC4005XL FPGA. To add the library for this device, select the File → Project Libraries... menu item.

🔹 dsgn3_2 - design not implemented - Project Manager 📃		
<u>File</u> <u>D</u> ocument <u>V</u> iew <u>Project</u> <u>Synthesis</u> <u>Implementation</u> <u>T</u> ools <u>H</u> elp		
New Project Ctrl+N		
Open Project Ctrl+0		
Copy Project Ctrl+D	<u> </u>	
dsgn3 2	-	
Archive Project Restore Project		
Project Info		
Project Type Ctrl+T		
Preferences		
1 dsgn3_2 SYNTHESIS ? SIMULATION		
3 dsgn2_2		
Exit IMPLEMENTATION / VERIFICATION		
	<u>•</u>	
Pcm : Execute c:\fndtn\active\exe\sc.exe		
Pcm : START: Schematic Editor		
Console / HDL Errors / HDL Warnings / HDL Messages /		
Project Libraries		

Scroll down in the list of libraries in the **Project Libraries** window and highlight the xc4000x entry.

Project Libraries		×
Attached Libraries	с <u>Р</u>	roject Libraries:
test1 test2	▲ <u>A</u> dd >>	lsgn3_2
tripgeni virtex	<< Remo <u>v</u> e	
xabelsim xc4000e	Info	
xc4000x xc9500	↓ Lib <u>Manager</u>	
A <u>t</u> tach	<u>C</u> lose	<u>H</u> elp

Pro	ject Libraries			×
A	ttached Libraries:		Project Libraries:	
	est1 est2 ipgeni irtex abelsim c4000e c4000x c9500	Add >> << Remoye	dsgn3_2	
,	A <u>t</u> tach		<u>H</u> elp	

Then click the Add>> button to copy the xc4000x library of parts to the list of project libraries.

This is the only library we need for this project, so click on the Close Button.

P	oject Libraries			×
	Attached Libraries:		Project Libraries:	
	test1 🔺 test2	$\underline{A}dd >>$	dsgn3_2 xc4000x	-
	tripgeni virtex	<< Remo <u>v</u> e		
	xabelsim xc4000e	<u>I</u> nfo		
	xc4000x xc9500	Lib <u>M</u> anager		
	Attach		<u>H</u> elp	

Notice that now the xc4000x library icon now appears in the **Hierarchy** pane of the **Project Manager** window. We can now open the **Schematic Editor** window and begin to design the counter module.

🔹 dsgn3_2 - design not impler	nented - Project Manager	<u> </u>	
<u>File D</u> ocument <u>View</u> Project	<u>Synthesis</u> Implementation <u>T</u> ools <u>H</u> elp		
Files Versions	Flow Contents Reports		
🗆 🗖 dsgn3_2		4	
🖻 😑 xslib	dsgn3_2		
⊡ ∰ leddcd.vhd - ⊖ dsgn3_2 - ⊖ xc4000x			
		-	
Pcm : Execute c:\fndtn\active\exe\sc.exe			
Pcm : START: Schematic Editor			
Console / HDL Errors / HDL Warnings / HDL Messages /			
Ready			

Drawing the Lower-Level Counter Schematic

In the **Schematic Editor** window, we begin by adding a sixteen-bit counter (CB16RE), an eight-bit counter (CB8RE), and a four-bit counter (CB4RE) to get a total length of 28 bits. Then the clock-enables of the counters are connected as we did in the example in Chapter 2. All we have left to do is add the inputs and outputs to the counter. We will not use IPADs or OPADs for this since these correspond to pins on the actual FPGA device. It is better to use hierarchical connectors or terminals for I/O into or out of a lower-level module and then place all pin connections in the root module. That way we can defer the decision as to which signals enter and exit the chip when we design the top-level module.

To begin placing I/O terminals, click on the Hierarchy Connector button.



The **Hierarchy Connector** window will appear. The first input we will add is for the clock. Type the name of the input (CLK) into the Terminal Name field and click on the OK button.

Hierarchy Connector		
Terminal <u>N</u> ame : CL	K	
Terminal Type : INPUT		
	<u>R</u> epeat	
	Attributes	
OK Cancel Help		

Then click in the drawing area of the **Schematic Editor** window and the input terminal will appear.



Repeat this procedure to add the reset input terminal (RST). Then wire these terminals to the clock and reset inputs of the counters as shown below. This takes care of the inputs to the 28-bit counter. Now we will use a bus to get the outputs from the counter.



Start by drawing a bus at the far right of the schematic. When the last point of the bus is drawn, right-click the mouse and select Add Bus Terminal... from the pop-up menu that appears.



Type the bus name (CNT) into the Name field of the **Add Bus Terminal/Label** window that appears. Also set the upper and lower indices of the bus range to 27 and 0, respectively, to set the bus width to 28 bits. Finally, specify that this is an output bus by selecting Output from the Terminal Marker drop-down list. Then click on the OK button.

Add Bus Terminal/Label	×
<u>B</u> us Label (e.g. BUS[0:2]):	🔽 Simple Bus
Name: CNT	Range:
Terminal Marker: Output ▼ None Input O <u>K</u> BiDirectional	<u>Cancel</u> <u>H</u> elp

The name of the bus and its upper and lower indices will appear in the schematic. Now we need to tap the upper four bits of this bus and connect them to the four-bit counter. Click on the Draw bus taps button to start this operation.



Then click on the CNT bus to select the bus that will be tapped. Create the individual taps by clicking on the Q3, Q2, Q1, and Q0 outputs (in that order) of the CB4RE counter.

The eight and sixteen-bit counters will be connected to the output terminals using buses, so click on the Draw buses button.





Draw a bus from the output bus of the eight-bit counter to the CNT bus as shown below.
Repeat this operation to draw a bus from the output of the sixteen-bit counter to the CNT bus. Now the question arises: "Which of the 28 bus lines are the sixteen-bit and eight-bit counter outputs connected to?" There are some implicit rules that govern this, but it is clearer if we explicitly specify the bus connections. To do this, double-click on the bus connected to the CB16RE counter.



The **Edit Bus** window will appear. Specify the upper and lower index of the bus as 15 and 0, respectively. Then click on the OK button. This will connect the sixteen outputs of the CB16RE counter to the lower sixteen bits of the 28-bit CNT bus.

Edit Bus	×
<u>B</u> us Label (e.g. BUS[0:2]):	<mark>, </mark>
Name:	Range:
CNT	
Terminal None 🔽	
	OK Bus End
∱°°	<u>C</u> ancel <u>H</u> elp

The index range of the bus connected to the sixteen-bit counter will now appear in the schematic.



We can repeat this procedure to specify the bus connections for the eight-bit counter as indicated below. Now all the bits in the output bus are connected to the counters.



At this point we should save the schematic.

Schematic Editor - [Modified - Non-Project Eile Edit Mode Options Hierarchy View	
New Sheet Ctrl+N Open Ctrl+O Open Macro Ctrl+O Close Save Save All Save All Import ViewLogic Schematic Generate Schematic from Netlist Print Ctrl+P Print.er Setup Ctrl+R Import ViewLogic Schematic from Netlist Ctrl+P Print.er Setup Ctrl+Y Page Setup Table Setup Import ViewLogic Setup Ctrl+B Import ViewLogic Setup Ctrl+B	CNT[23:16]
Import ViewLogic Schematic Generate Schematic from Netlist Print Ctrl+P Printer Setup Ctrl+T	CNT[15:0] CB4RE
JSGN3_21	بر د
2.6 , 4.3	Select and Drag

Set the name of this module to CNTR and then click on the OK button.

Save As		×
File <u>n</u> ame: CNTR.SCH	<u>F</u> olders: c:\prag21i\dsgn3_2	OK Cancel
dsgn3_21.SCH	 C:\ PRAG211 PRAG211 DSGN3_2 dsgn3_2 ib xproj 	
Save file as <u>type:</u> Schematic (*.SCH)	Drives:	▼ Net <u>w</u> ork

Finally, we can exit the **Schematic Editor** window.



Adding the Lower-Level Counter Module to the Project

Once we are back in the **Project Manager** window, we can add the counter schematic to the project by clicking on the dsgn3_2 icon in the **Hierarchy** pane and selecting Add HDL Source Files... from the pop-up menu. (I know the counter is a schematic and not an HDL file, but this works anyway.)

🗘 dsgn3_2 - design not implemented - Project Manager	<u> – – ×</u>
<u>File D</u> ocument <u>V</u> iew <u>Project</u> <u>Synthesis</u> <u>Implementation</u> <u>T</u> ools <u>H</u> elp	
D 🕞 🖯 🕪 💌 🕒 🖬 🖻 🗄 📲	
Files Versions Flow Contents Reports Image: Synthesize Update Project gn3_2 Image: Synthesize Madd HDL Source Files Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Image: Synthesize Imag	4
Pcm : Execute: Im.exe -p 1104 -i /dsgn3_2 c:\prag21i\dsgn3_2\lib\dsgn3_2 Lm : Library Manager has terminated. Pcm : EXIT: Library Manager	-
Console / HDL Errors / HDL Warnings / HDL Messages /	<u> </u>
Add Source Files	

In the **Add Document** window, select Schematic (*.SCH) in the drop-down list attached to the Files of type field. Then highlight the CNTR.SCH file and click on the Open button.

Add Documer	nt					? ×
Look in: 🔂	dsgn3_2	•	£		e *	III III
dsgn3_2 ib sch xproj CNTR.SCH dsgn3_21.						
File <u>n</u> ame:	ļ					<u>O</u> pen
Files of <u>type</u> :	Schematic (*.SCH) HDL (*.VHD,*.VER,*.VE,*.V) Schematic (*.SCH) FSM (*.ASF)	2		•		Cancel <u>H</u> elp

Before adding the CNTR.SCH schematic to the project, Foundation will ask you to specify the target device for the schematic. As we stated before, we are targeting the XS40-005XL Board so set the device information as shown below.



After clicking on OK in the **Select Part** window, the Foundation software will extract the netlist from the schematic and add the cntr.sch schematic and cntr netlists library to the project. Clicking the + signs to the left of these elements will expand them so we can see their contents.



We note that the cntr.sch element lists the names of the four, eight, and sixteen-bit counters as its subcomponents as we would expect. The cntr netlists element also lists the Xilinx netlist files (XNF) for these counters as subcomponents as well as the XNF files for the toggle and D flip-flops that make up the counters.

🐌 dsgn3_2 - design not imple	mented - Project Manager	- 🗆 🗵
<u>File D</u> ocument <u>V</u> iew <u>P</u> roject	<u>Synthesis</u> Implementation <u>T</u> ools <u>H</u> elp	
D 🕞 🖯 🚺 🛤 🔳 🎙		
Files Versions	Flow Contents Reports	
🗆 🗖 dsgn3_2		-
🖻 🕏 cntr.sch	dsgn3_2	-
🕀 🔂 \$i1 - cb16re		
⊞ 🛨 \$i2 - cb8re ⊞ 🛨 \$i3 - cb4re		
⊞ ⊖ xslib	DESIGN ENTRY	
😑 🖯 cntr netlists	•	
E 🐨 CB16RE.XNF	🖹 🔸 🦫 🔥 📩	
	SYNTHESIS	
E ▼FDRE.XNF		
E TVFD.XNF	🛃 🕹 🖗 🔠 📭 🔛 🔛	
dsgn3_2	IMPLEMENTATION ? VERIFICATION	
🛄 🖯 xc4000x		
		<u> </u>
Dpm : Done Pcm : Document C:\PRAG21	II\DSGN3_2\CNTR.SCH added	-
Pcm : Synopsys server initia	—	-
Console / HDL Errors / HDL W	arnings / HDL Messages /	• •
cntr netlists - CNTR NETLISTS	3	

Modifications to the VHDL Code of the Root Module

The VHDL for the top-level root module is entered in the **HDL Editor** window as shown below. The main differences between this root module and the one from *dsgn3_1* project are:

- Line 4: Only the leddcd_pckg is included in this module because no VHDL package was created for the counter schematic.
- Lines 15–17: The component declaration for the 28-bit counter is directly incorporated into the architecture section of the root module. This is the simplest way to do it since the counter is only used in one place. For more complex designs, you could package the component declaration in a file that could be included anywhere the counter was needed.



The root module is stored in the cntdisp.vhd file, and this file is added to the *dsgn3_2* project.

Synthesizing the Netlist

Now the synthesis tool can be run on the project files to extract the netlist.



As before, set the target device appropriately and select the cntdisp entry as the top-level module for the synthesizer. Then click on the Run button and the synthesizer will do its job.

Synthesis/Impl	ementatior	n settin	gs	×
Top level: Version name: Synthesis Setting	CB16RE CB8RE CB8RE_0 cntdisp cntr		× 	<u>R</u> un OK <u>C</u> ancel
Target Device Family: XC40 Device: 4005	000×L ×LPC84	• •	Speed:	Help xI-09
View Estimate	ed Performar	nce after m tools :ttings—		
Revisi	on name: ol Files:	rev1 SE	T	<u>Options</u>

Assigning the I/O Pins and Implementing the Design

After the netlist is synthesized, place the pin assignments for the XS40 Board into the dsgn3_2.ucf file as shown below. Then specify this file as the constraints file when the implementation tools are run.

sgn3_2.ucf - HDL Editor
<u>File E</u> dit <u>S</u> earch <u>V</u> iew Synthesis <u>P</u> roject <u>T</u> ools <u>H</u> elp
1 net rst loc=p44;
2 net clk loc=p13;
3 net s<0> loc=p25;
4 net s<1> loc=p26;
5 net s<2> loc=p24;
6 net s<3> loc=p20;
7 net s<4> loc=p23;
8 net s<5> loc=p18;
9 net s<6> loc=p19;
For Help, press F1 Ln 1, Col 1 TEXT

Downloading and Testing the Design

At this point, the final bitstream for downloading into the XS40-005XL Board is available. Open the directory containing the *dsgn3_2* project files and drag-and-drop the dsgn3_2.bit file into the **gxsload** window. The bitstream will download into the XS40 Board attached to the parallel port.

		🔁 dsgn3_2		_	
		<u>F</u> ile <u>E</u> dit ⊻iew	<u>G</u> o F <u>a</u> vorites	<u>H</u> elp	
🗶 gxsload			IX PC	$\mathfrak{O} \times \mathfrak{F} =$	•
Drop .BIT, .SVF, .HEX, and .EXO files here to download to the	Exit	Address 🗀 C:\Prag	g21i\dsgn3_2		•
XS or XSV Board.		🔁 dpm_net	👪 cntdisp.log	🖻 dsgn3_2.XNF	🖻 Fl
Recent Files:		dsgn3_2	🗒 cntdisp.vhd	🔄 DSGN3_21.BSC	🔄 FI
		lib 🔁	🔊 CNTR.alb	🔄 dsgn3_21.SCH	🔄 FI
		sch	CNTR.SCH	💽 exp_net.log	🔊 FI
		🔁 xproj	🔊 cntr.xnf	📓 express.ini	🔊 FI
		CB16RE_0.xnf	🛋 dsgn3_2.alb	🛋 FD_0.xnf	🔊 FI
, Line and L		CB4RE_0.xnf	🛋 dsgn3_2.bit	🔄 FD_0_0.xnf	🔊 FI
** +		CB8RE_0.xnf	🔊 dsgn3_2.ll	🔄 FD_0_1.xnf	🔄 FI
		🔊 cntdisp.bak	🙍 dsgn3_2.prj	🔄 FD_0_10.xnf	🔊 FI 🔊 FI
		🛋 cntdisp.er	🗐 dsgn3_2.ucf	🛋 FD_0_11.xnf	🖻 Fl
	LPT1 -	•			►
		1 object(s) selected	d	🛄 My Compute	н <u>//</u>

If pin D0 of the parallel port is at logic 1 after the downloading completes, the counter will be held in the reset state so only a static O is displayed. To release the reset, open the **gxsport** window and click on the D0 button until it displays a 0.

🗶 gxsport		
1 1 1 D7 D6 D5 <u>Strobe</u> □ <u>C</u>	1 1 1 1 D4 D3 D2 D1 ount Port LP1	DO Exit

Then click on the Strobe button so the logic 0 value is output on the D0 pin of the parallel port.

1 1 1 1 1 1 1 0 Exit D7 D6 D5 D4 D3 D2 D1 D0 Strobe □ Count Port LPT1 ▼	🔼 gxsport			<u>- 🗆 ×</u>
	1 1 D7 D6 <u>S</u> trobe	1 1 D5 D4	 	Exit

Now you should observe the seven-segment LED running through the sequence: O, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, C, d, E, F, ... with each digit being displayed for roughly 1/3 seconds.

Hierarchical Schematic-Based Design with VHDL Modules

In the *dsgn3_3* project, we will replace the root VHDL module with a schematic. This design will be tested with an XS95-108 Board, so start a schematic-based project targeted at an XC95108 CPLD.

New Proje	ect		×
Name:	dsgn3_3		ОК
Directory:	C:\PRAG211		Cancel
Туре:	F2.1i	•	<u>H</u> elp
Flow:		⊂ HD <u>L</u>	
×C9500	■ 95108PC84	4 🗾	20 💌

Creating the Schematic-Based Counter Macro

Open a **Schematic Editor** window and create a 28-bit counter as we did in the previous section and save it in the cntr.sch file.



Since we will want to include this 28-bit counter in the top-level schematic, we need to create a macro symbol to represent the counter in the list of parts. This is done using the Hierarchy→Create Macro Symbol from Current Sheet... command.



Type the name for the macro (CNTR) in the Symbol Name field of the **Create Symbol** window and click on the OK button.

Create Sy	mbol	×
<u>S</u> ymbol N Co <u>m</u> ment		ference: H
<u>S</u> heet:	C:\PRAG21I\DSGN3_3\CNTR.SC	Browse
<u>– P</u> in: ——		
<u>I</u> nput:	CLK, RST	<u>E</u> dit
<u>O</u> utput:	==CNT[27:0]==	
<u>B</u> idir:		Cancel
<u>P</u> WR:		<u>H</u> elp

A symbol for the counter will be added to the *dsgn3_3* project library and you will be asked if you want to edit it. Click the No button and return to the **Project Manager** window.



Examining the Project Library

Now we can view what has been added to the project library by double-clicking the dsgn3_3 library symbol.



The **Library Manager** window will appear and show that the dsgn3_3 library contains the single CNTR object. We can see the symbol for this component by double-clicking the CNTR entry in the window.

🗿 Library Manager				<u>_ D ×</u>
<u>File Library Object View H</u>	<u>H</u> elp			
	s 🔊 🖻 🕻	1 ♦ ĕ H H H A	8 🖪	5 ?
🌮 Libraries 👁 Objects				
Logical Name (+ Physical Name	Comment	Object Type	Attributes	Library
		NET+SCH+SYM	0000000	DSGN3_3
Ready.		1 object(s) selecte	d	

The Symbol Editor window shows the 28-bit counter with the reset and clock inputs arranged along the left-hand side and the 28-bit counter output bus on the right-hand side. This completes the process of adding the 28-bit counter macro to the project so close the **Symbol Editor** window.

📳 Symbol Editor - [CNTR (DSGN3_3)]		<u>- 0 ×</u>
≇ Pile Edit View Symbol Window Help		_ 8 ×
Symbol CNTR	Number: Name:	-
Library DSGN3_3		<u> </u>
PCB Footprint Ref Prefix H Sections 0		_
Description	RST	
SPICE Model Info	CLK CNT[27:0]	_
Pins		
CLK IN RST IN CNT[27:0] OUT		-
J <u>r</u> For help, press F1	200% 6.3, -4.3	

Adding a VHDL-Based Macro to the Project Library

Next we need to turn the VHDL code for the seven-segment LED decoder into a macro. Enter the source code as shown below and then execute the Project→Create Macro command.

📄 le	ddcd.vhd - HDL Editor
<u>F</u> ile	<u>E</u> dit <u>S</u> earch <u>V</u> iew Synthesis <u>P</u> roject <u>T</u> ools <u>H</u> elp
	🖆 🖬 🗿 👔 🚹 🚹 🛕 🖾 🕹 🖉 🦿
1 2 3	library IEEE; Create Macro use IEEE.std_logic_1 Update Macro
4	entity leddcd is
5	port (
6	d: in STD_LOGIC_VECTOR (3 downto 0);
7	s: out STD_LOGIC_VECTOR (6 downto 0)
8); and ladded:
10	end leddcd;
11	architecture leddcd_arch of leddcd is
12	begin
13	with d select
14	s <= "1110111" when "0000",
15	''0010010'' when ''0001'',
16	"1011101" when "0010",
17	"1011011" when "0011",
18	"0111010" when "0100",
19	"1101011" when "0101",
20	"1101111" when "0110",
21	"1010010" when "0111",
<u> </u>	<u></u>
Creat	es library macro Ln 1, Col 1 VHDL

The **DPMCOMP** window shows the progress as the synthesizer tool processes the VHDL and deposits the netlist into the project library.

🛃 DPMCOMP	<u> </u>	<
Initialize 1	DPM	
Checking lie	cense	
License che	sk OK.	
	C:\Prag21i\dsgn3_3\leddcd.vhd. XC9500. 95108PC84-20.	
Create proje Create file Analyze sou: Create chip Optimize ch:	rce file	



After the synthesis completes, we can double-click the dsgn3_3 library icon and see that the LED decoder has now been added to the library.

🗿 Library Ma	nager				- D ×
<u>File L</u> ibrary <u>(</u>	<u>⊇</u> bject <u>V</u> iew <u>I</u>	<u>H</u> elp			
			はくう形法な	80	6 ?
🌮 Libraries	⊅ Objects				
Logical Name (Physical Name	Comment	Object Type	Attributes	Library
CNTR 2	CNTR		NET+SCH+SYM	00000000	DSGN3_3
📒 LEDDCD	LEDDCD	# ,	NET+SYM	00000000	DSGN3_3
					I
Ready.			2 object(s)		

Double-clicking the LEDDCD entry in the list of library objects shows the symbol for the LED decoder.

🐌 Symbol Editor - [LEDDCD (DSGN3_3)]		- D ×
Tile Edit ⊻iew Symbol Window Help		-8×
Symbol LEDDCD	Number: Name:	
Library DSGN3_3		-
Ref Prefix U Sections 0		
Description #,	Vhdl code D[3:0] \$[6:0]	
SPICE Model Info	-ferel - eferel	
Pins		
S(6:0) OUT D(3:0) IN		
<u> </u>		
For help, press F1	200% 14.0, -	8.4

Placing the Lower-Level Macros in the Root Schematic

Now that the lower-level modules are designed, we can open a schematic for the toplevel module. Notice that the list of parts available for use in creating the root module now contains the 28-bit counter and LED decoder macros.



We can drag-and-drop the CNTR and LEDDCD macros into the drawing area of the **Schematic Editor** window and connect them with a 28-bit bus. But which of the 28 outputs from the counter macro are connected to the four inputs of the LED decoder? The rule is that the pins are connected starting at the left-most index and proceeding to the right. So CNT27 connects to D3, CNT26 connects to D2, CNT25 connects to D1, and CNT24 connects to D0.



Examining Lower-Level Macros in the Hierarchy

We can actually view what is inside these macros using the Hierachy Push/Pop button.



An **H** will be attached to the cursor indicating that it can be used to descend through the project hierarchy. Double-clicking the LEDDCD symbol loads the VHDL code describing this macro into an **HDL Editor** window. You can modify and update the macro symbol if needed.

E	s <mark>hematic Editor - [DSGN3_31.SCH]</mark> ile <u>E</u> dit <u>Mode Options Hi</u> erarchy <u>V</u> iew Displ 2 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 		× &×
ᆇ▤▯▯◨◥▥◁▯▯Ⴝ	H1		
	RST CLK CNT[27:0	CNT[27:0]	U1 Vhdl code D[3:0]
⊳ Ga	CNTR]	
ļ	✓ DSGN3_31		بر ک
	5.9 , 2.9		Hierarchy Push/Pop

📄 le	ddcd. vhd - HDL Editor	×
<u>F</u> ile	<u>E</u> dit <u>S</u> earch <u>V</u> iew Synthesis <u>P</u> roject <u>T</u> ools <u>H</u> elp	
D		?
1 2 3 4 5 6 7 8 9 10	<pre>library IEEE; use IEEE.std_logic_1164.all; entity leddcd is port (d: in STD_LOGIC_VECTOR (3 downto 0); s: out STD_LOGIC_VECTOR (6 downto 0)); end leddcd;</pre>	4
11 12	architecture leddcd_arch of leddcd is	
13	begin with d select	
14	s <= "11101111" when "00000",	
15	"0010010" when "0001",	
16	"1011101" when "0010",	
17	"1011011" when "0011",	
18	"0111010" when "0100",	
19	"1101011" when "0101",	
20	"1101111" when "0110",	
21	"1010010" when "0111",	تے.
		<u> </u>
For H	elp, press F1 Ln 1, Col 1 VHDL	11.

Assigning the I/O Ports to the CPLD Pins and Exporting the Netlist

Since the design is being targeted to an XS95-108 Board, the inputs and outputs of the top-level module have to be connected to the CPLD pins as shown in Figure 9.



Figure 9: Connection of the programmable oscillator, parallel port, and LED digit to the pins of the CPLD on the XS95 Board.

The completed schematic with all I/O pads and their pin assignments is shown below. The netlist for the entire design is created using the Options \rightarrow Create Netlist command.



Then the netlist is exported into the dsgn3_3.alb file in EDIF 2.0 format using the Options→Export Netlist... command. Then the **Schematic Editor** window can be closed.

Export Netlis	t		? ×
Look jn: 🔂	dsgn3_3	· 🗈 💆	📸 📰 📰
	P.TMP		
lib D xproj			
dsgn3_3.a	lb		
1			
File <u>n</u> ame:	dsgn3_3.alb		<u>Open</u>
Files of <u>type</u> :	Edif 200 [*.EDN]	•	Cancel

Implementing the Design

Now the implementation tools are run to map the netlist to the XC95108 CPLD chip.

Eile Document Yiew Project Implementation I cols Help Image: Second structure Imag
Files Versions Flow Contents Reports Synthesis
Image: Second
Conv : Total number of nets: 469 Conv : EDIF netlist exported to file - C:\PRAG21I\DSGN3_3\dsgn3_3.edn Pcm : EXIT: Schematic Editor
Console CONTR - DSGN3_31/H1

The appropriate CPLD part must be specified in the Device and Speed fields of the **Implement Design** window that apears. There is no need to specify a constraint file with the pin assignments since these have already been added to the top-level schematic.

Implement Desi	gn				×
Device 95108	PC84	• s	peed	20	•
Version name:	ver1				
<u>R</u> evision name:	rev1				
Control Files:	<u>S</u> et			<u>O</u> ptior	IS
Bun	ОК	<u>C</u> ance	:	<u>H</u> elp	>

After this is done, the programming tools are used to generate an SVF file that can be downloaded into the XS95-108 Board.

Downloading and Testing the Design

Open the directory containing the *dsgn3_3* project files and drag-and-drop the dsgn3_3.svf file into the **gxsload** window. The bitstream will download into the XS95 Board attached to the parallel port.

	🔁 dsgn3_3	×
🗙 gxsload	<u>File E</u> dit <u>V</u> iew <u>G</u> o F <u>a</u> vorites <u>H</u> elp	
	↓+ • + • ⊡ 🐰 🖻 🗳 🖂 🖂 🖼	•
Drop .BIT, .SVF, .HEX, and .EX0 Exit	Address C:\Prag21i\dsgn3_3	•
XS or XSV Board.	DPMCOMP.TMP 🖻 dsgn3_3.xbt 🔳 leddcd.vhd	
Recent Files:	📄 lib 🛛 📓 dsgn3_31.SCH 🔄 leddcd.xnf	
	📄 🗀 xproj 🛛 📲 ERRLOG.LOG 🔄 🖻 leddcd.xsf	
	🖬 CNTR.SCH 🛛 🖼 exp_EDIF.log 🛛 🐻 logiblox.ini	
	🔊 📾 dsgn3_3.alb 🛛 📾 leddcd.alr 🛛 👪 NETLIST.LOG	
	📋 dsgn3_3.EDN 🛛 🔏 leddcd.ASX 🛛 🖺 time_sim.edn	
	📓 dsgn3_3.jed 🛛 📓 leddcd.ENT 🖉 types.dir	
	📓 📾 dsgn3_3.prj 🛛 📾 leddcd.ER 🛛 📾 VHDL.LST	
	📕 🖻 dsgn3_3.svf 🛛 🔣 leddcd.log 🛛 🐻 xproj.ini	
	🔄 🗐 dsgn3_3.ucf 🛛 🛤 leddcd.opt	
Reload EEPROM Port LPT1 -		
	1 object(s) selected 🛄 My Computer	11.

If pin D0 of the parallel port is at logic 1 after the downloading completes, the counter will be held in the reset state so only a static O is displayed. To release the reset, open the **gxsport** window and click on the D0 button until it displays a 0.



Then click on the Strobe button so the logic 0 value is output on the D0 pin of the parallel port.

🗶 gxsport	<u>- 🗆 ×</u>
1 1	Exit

Now you should observe the seven-segment LED running through the sequence: O, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, C, d, E, F, ... with each digit being displayed for roughly 1/3 seconds.

Creating a Macro Using LogiBLOX

Xilinx Foundation contains the LogiBLOX tool that lets you create many types of commonly-used logic functions. Rather than having to create an entire schematic to design a 28-bit counter, LogiBLOX lets you do it with a few mouse clicks. We will modify the *dsgn3_3* project to use a counter created with LogiBLOX. To start, make a copy of the dsgn3_3 project using the File → Copy Project menu item.

ዾ dsgn3_3 - 95108-20PC84 - Project Manager 📃	
<u>File D</u> ocument <u>View Project</u> Implementation <u>T</u> ools <u>H</u> elp	
New Project Ctrl+N 🔁 🚳 🗭 💀	
Open Project Ctrl+0	_
Copy Project Contents Reports Synthesis	
ver1 rev1 (XC95108-20-PC84)	
Archive Project	
Restore Project Project Info	
Project Libraries Ctrl+L DESIGN ENTRY SIMULATION Project Type Ctrl+T	
Preferences	
1 dsgn3_3 IMPLEMENTATION VERIFICATION	
2 dsgn3_3	
4 dsgn3_1a	
Pcm : Copying files	-
Pcm : Cannot copy Xilinx project with different name. Pcm : Project 'dsgn3_3' in directory c:\prag21i has been copied into c:\prag21i\dsgn3_3a	
Console	-
Copy project	

In the **Copy Project** window, name the new project dsgn3_3a.

Copy Project	x X
Source Project:	c:\prag21i\dsgn3_3.pdfBrowse
- Destination Name: Directory:	dsgn3_3a c:\prag21i Browse
	OK Cancel Help

Open the *dsgn3_3a* project and then open the root module schematic (which still has the name dsgn3_31.sch). In the **Schematic Editor** window, issue the Tools→LogiBLOX Module Generator command to begin creating a new counter macro.



The **LogiBLOX Module Selector** window will appear. A pull-down list attached to the Module Type field shows the range of functions that LogiBLOX can generate. Highlight the Counters entry since that is what we wish to build.

LogiBLOX Module Se	lector		_ 🗆
Selection		5	ОК
Module <u>N</u> ame:	Module Type:	Bus <u>W</u> idth:	
	Accumulators	- 4 -	Cancel
D 1 1	Accumulators Adders/Subtracters		Setup
Details	Clock Dividers		
Add/Sub —	- Comparators		User P <u>r</u> efs
Carry Input 🔽 —	Constants Counters		
	Data Registers		<u>H</u> elp
	Decoders		
	Inputs/Outputs		
B	Memories Multiplexers		
Load 🗖	Pads	Overflow	
Clock Enable 🔽 —	Shift Registers	Carry Output	
Clock —	Simple Gates Tristate Buffers		
Async. Control 🔲		Uveniow	
Sync. Control	└──── 🔽	Carry Output	
Sync. Control j			
<u>C</u> Value	=		
<u>Operation</u>	= Add/Subtract	•	
<u>S</u> tyle	= Maximum Speed	•	
<u>E</u> ncoding	= Unsigned	•	
<u>A</u> sync, Val	=		
Sync. <u>V</u> al	=		
	,		

Now set the following fields in the window:

- 1. Enter CNTR2 in the Module Name field (since we already have a macro named CNTR).
- 2. Set the Bus Width field to 28.
- 3. Remove the checkmark in the D_IN box because we do not need to load arbitrary values into the counter.
- 4. Remove the checkmark in the Clock Enable box because we do not need to disable the incrementing of the counter.
- 5. Place a checkmark in the Sync. Control box that will be used as a synchronous reset.
- 6. Enter 0 into the Sync. Val field. This value is loaded into the counter whenever a rising clock edge occurs and the Sync. Control input is a logic 1. (Thus, the Sync. Control acts as a synchronous reset input.)
- 7. Set the Operation field to Up since the counter only needs to be count in one direction.

đ,	LogiBLOX Module Selector	_ 🗆 🗙
	Selection Module Name: Module Type: CNTR2 Counters	OK Cancel
[- Details-	Set <u>u</u> p
		User P <u>r</u> efs
	Async. Control 🗖 Sync. Control 🔽 —— Clock Enable 🗖 Clock ——> 🗖 Terminal Count	<u>H</u> elp
	Operation = Up	
	Style = Maximum Speed	
	Encoding = Binary	
	Count Limit =	
	<u>A</u> sync. ∀al =	
	Sync. <u>V</u> al = 0	
	Async. Count =	
	Sync. <u>C</u> ount =	
l		

After clicking OK in the **LogiBLOX Module Selector** window, the CNTR2 macro will appear in the part list in the **Schematic Editor** window.



The CNTR2 macro can be dropped into the schematic drawing area and attached as shown below. Then the netlist for the schematic can be extracted, exported, implemented, downloaded, and tested using the XS95-108 Board as was done with the *dsgn3_3* project.

