

Introduction to WebPACK 5.2 for FPGAs

Using Xilinx WebPACK Software to Create FPGA Designs for the XSB-300E Board

Release date: 9/30/2003

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

All XC-prefix product designations are trademarks of Xilinx.

Table of Contents

What This Is and <i>Is Not</i> 1
FPGA Programming
Installing WebPACK
Getting WebPACK5
Installing WebPACK6
Getting XSTOOLs6
Installing XSTOOLs7
Our First Design
An LED Decoder8
Starting WebPACK Project Navigator10
Describing Your Design With VHDL16
Checking the VHDL Syntax23
Fixing VHDL Errors24
Synthesizing the Logic circuitry for Your Design27
Implementing the Logic Circuitry in the FPGA28
Checking the Implementation
Assigning Pins with Constraints
Viewing the Chip40
Generating the Bitstream48
Downloading the Bitstream50
Testing the Circuit53
Hierarchical Design
A Displayable Counter54
Starting a New Design55
Adding the LED Decoder57

	Adding a Counter6	30
	Tying Them Together6	35
	Checking the VHDL Syntax	90
	Constraining the Design	91
	Synthesizing the Logic Circuitry for the Design	97
	Implementing the Logic Circuitry in the FPGA	98
	Checking the Implementation	98
	Checking the Timing10)0
	Generating the Bitstream10)1
	Downloading the Bitstream10)6
	Testing the Circuit11	15
Go	bing Further11	16



What This Is and *Is Not*

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

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

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

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

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

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

- It will not teach you how to design logic with VHDL.
- It will not teach you how to choose the best type of FPGA or CPLD for your design.
- It will not teach you how to arrange your logic for the most efficient use of the resources in an FPGA.
- It will not teach you what to do if your design doesn't fit in a particular FPGA.
- It will not show you every feature of the WebPACK software and discuss how to set every option and property.
- It will not show you how to use the variety of peripheral devices available on the XSB-300E Board.

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

1

FPGA Programming

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

- 1. You enter a description of your logic circuit using a *hardware description language* (HDL) such as VHDL or Verilog. You can also draw your design using a schematic editor.
- 2. You use a *logic synthesizer* program to transform the HDL or schematic into a *netlist*. The netlist is just a description of the various logic gates in your design and how they are interconnected.
- 3. You use the *implementation tools* to map the logic gates and interconnections into the FPGA. The FPGA consists of many *configurable logic blocks* which can be further decomposed into *look-up tables* that perform logic operations. The CLBs and LUTs are interwoven with various *routing resources*. The mapping tool collects your netlist gates into groups that fit into the LUTs and then the place & route tool assigns the gate collections to specific CLBs while opening or closing the switches in the routing matrices to connect the gates together.
- 4. Once the implementation phase is complete, a program extracts the state of the switches in the routing matrices and generates a *bitstream* where the ones and zeroes correspond to open or closed switches. (This is a bit of a simplification, but it will serve for the purposes of this tutorial.)
- 5. The bitstream is *downloaded* into a physical FPGA chip (usually embedded in some larger system). The electronic switches in the FPGA open or close in response to the binary bits in the bitstream. Upon completion of the downloading, the FPGA will perform the operations specified by your HDL code or schematic.

That's really all there is to it. Xilinx WebPACK provides the HDL and schematic editors, logic synthesizer, fitter, and bitstream generator software. The XSTOOLs from XESS provide utilities for downloading the bitstream into an <u>XSB-300E Board</u> containing a Xilinx XC2S300E SpartanIIE FPGA.



2

Installing WebPACK

Getting WebPACK

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



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

download completes, you also need to download <u>Service Pack 3</u> to get the most current WebPACK updates.

🖉 Home : ISE Logic Design Too	ols : Free ISE WebPACK - Micros	oft Internet Exp	lorer						<u>- ×</u>
File Edit View Favorites	Tools Help								11
🗢 Back 🔹 🤿 🔹 🙆	🕄 Search 📓 Favorites 🛞 🕅	iedia 🥶 🗟 🗸	9	87	• 📃				
Address 🕘 http://www.xilinx.cor	n/webpack/index.htm#sfd							▼ 200	Links »
	ISE WebPACK Single File Download Modules Below are the single-file download modules similar to those used in prior WebPACK releases. These software packages allow you to select the feature set that best suits your needs. Each self-extracting zip file contains its own installer. For more details, please visit the Module Description page.								
	ISE WebPACK Download Module (service pack required - see information below)	Download Size	CPLD Design Environment	FPGA Design Environment	CPLD Programming	FPGA Programming	ChipViewer, XPower HTML Reporting		
	Complete ISE WehPACK Software	187 Mb*	~	~	•	~	*		
	Complete CPLD Tool Set	101 Mb*	-		~		√		
	CPLD Design Environment	81 Mb*	-						
	CPLD Programming Tools	20 Mb*			~				
	CPLD Optional Tools	44 Mb*					√		
	Complete Programming Tools	70 Mb*			1	1			
	* Customers installing the ab download and install the lates <u>Pack 3</u> is available. Service p contents are located in the <u>R</u>	t Service Pack. ack installation <u>EADME</u> .	. At t	nis ti	me,	Servi	ice		T
) http://direct.xilinx.com/direct/w	ebpack/52/WebPACK_52_fcfull_i.ex	e						🔹 🚺 🔮 Internet	11.

Installing WebPACK

After the WebPACK software download completes, double-click the WebPACK_52_fcfull_i.exe file. The installation script will run and install the software. Accept the default settings for everything and you shouldn't have any problems. Then repeat this procedure with the Service Pack 3 install file 5_2_03i_pc.exe.

Getting XSTOOLs

If you are going to download your FPGA bitstreams into an XSB-300E Board, then you will need to get the XSTOOLS software from <u>http://www.xess.com/ho07000.html</u>. Just download the <u>xstools4_0_3.exe</u> file.

Installing XSTOOLs

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



Our First Design

An LED Decoder

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

Four-bit Input	Hex Digit	LED Display
0000	0	0
0001	1	1
0010	2	5
0011	3	Э
0100	4	ч
0101	5	5
0110	6	6
0111	7	٦
1000	8	8
1001	9	9
1010	A	8
1011	В	8
1100	С	C
1101	D	D
1110	E	E
1111	F	F

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



Starting WebPACK Project Navigator



We start WebPACK by double-clicking the Navigator icon, ,on the desktop. This will bring up an empty project window as shown below. The window has four panes:

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



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

🔉 Xilinx - Project Navigator - No Project	
File Edit View Project Source Process Window Help	
New Project Image: Constraint of the image: Con	
New Ctrl+N Open Ctrl+O Close	
Save All Recent Projects Recent Files	
Exit	
(Empty Log)	× •

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		x
Project Name:	Project Location C:\ise5\ISEexa	
Property Name		Value
Device Family		CoolRunner XPLA3 CPLD
Device		Auto xcr3000xl
Package		*
Speed Grade		_*
Design Flow		XST VHDL
	OK	Cancel Help

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

Browse for Folder	? ×
Select a Directory	
C:\tmp\fpga_designs	
OK Car	ncel

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

New Project		×
Project Name:	Project Location	n:
design1	C:\tmp\fpga_d	esigns\design1
Project Device Options:		
Property Name		Value
Device Family		CoolRunner XPLA3 CPLD
Device		Auto xcr3000xl
Package		*
Speed Grade		*
Design Flow		XST VHDL
	OK	Cancel Help

To set the family of FPGA devices we will target with this design, click in the Value field of the Device Family property. Select the Spartan2E entry in the pop-up menu that appears.

lew Project			2
Project Name: design1	Project Location C:\tmp\fpga_d		
Project Device Options:			
Property Name		Value	
Device Family		CoolRunner XPLA	3 CPL 💌
Device		CoolRunner XPLA	3 CPLDs
Package		CoolRunner2 CPL)s
Speed Grade		Spartan2	
Design Flow		Spartan2E	
		Spartan3 k	5
		Virtex2	
	ОК	Virtex2P	
L		VirtexE	
		XC9500 CPLDs	I

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 xc2s300E since this is the device used in the XSB-300E Board where we will test our design.

New Project]
Project Name: design1	Project Location	
Project Device Options:		
Property Name		Value
Device Family		Spartan2E
Device		xc2s50e
Package		xc2s50e
Speed Grade		xc2s100e
Design Flow		xc2s150e
		xc2s200e
		xc2s300e
	ОК	Cancel ^K Help

Now click in the Value field of the Package property and choose the PQ208 package style for the FPGA on the XSB-300E Board.

New Project			×
Project Name: design1	Project Location C:\tmp\fpga_d		
Project Device Options:			
Property Name		Value	
Device Family		Spartan2E	
Device		xc2s300e	
Package		ft256	•
Speed Grade		ft256	
Design Flow		fg456	
	OK	Cancel	Help

Then set the speed grade property for the FPGA to -6.

New Project			×
Project Name: design1	Project Location C:\tmp\fpga_do		
Project Device Options:			
Property Name		Value	
Device Family		Spartan2E	
Device		xc2s300e	
Package		pq208	
Speed Grade		-7	•
Design Flow		-7	
	ОК	-6 Cancel	Help

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 Name: design1	Project Location C:\tmp\fpga_d		
Project Device Options:			
Property Name		Value	
Device Family		Spartan2E	
Device		xc2s300e	
Package		pq208	
Speed Grade		-6	
Design Flow		XST VHDL	•
	OK	EDIF NGC/NGO XST VHDL XST Verilog	

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

- 1. A project object called design1.
- 2. A chip object called xc2s300e-6pq208 XST VHDL.

🗞 Xilinx - Project Navigator - C:\tmp\fpga_designs\design1\design1.npl	
File Edit View Project Source Process Window Help	
	•
Sources in Project: design1 xc2s300e-6pq208 - XST VHDL Module View Snapshot View Processes for Current Source: (No Processes Available) Processes Available)	
(Empty Log)	× •
Hierarchy is up to date.	• //:

Describing Your Design With VHDL

Once all the project set-up is complete, we can begin to actually design our LED decoder circuit. We start by adding a VHDL file to the *design1* project. Right-click on the xc2s300e-6tpq208 object in the Sources pane and select New Source ... from the pop-up menu as shown below.

Xilinx - Project Navigator - C:\tmp\fpga_designs\design1\design1.npl	
File Edit View Project Source Process Window Help	
	•
Sources in Project: design1 New Source Add Source Shift+Insert Remove Delete	
Image: Module View Image: Move to Library Open Open Processes for Current Source Toggle Paths	
Design Entry L Properties Process View	
(Empty Log)	×
Add a new source to the project	- 🔶 //.

This causes a window to appear where we must select the type of source file we want to add. Since we are describing the LED decoder with VHDL, 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	X
User Document VHDL Module Schematic VHDL Library VHDL Package VHDL Test Bench Test Bench Waveform BMM File MEM File Implementation Constraints File State Diagram	File Name: leddcd Location: C:\tmp\fpga_designs\design1
< Back Next >	Cancel Help

Then the **Define VHDL Source** window appears where we declare the inputs and outputs to the LED decoder circuit. In the first row, click in the Port Name field and type in d (the name of the inputs to the LED decoder). The **d** input bus has a width of four, so click in the MSB field and increment the upper range of the input field to 3 while leaving 0 in the LSB field.

Define VHDL Source			×
Entity Name led	acd		
Architecture Name Beł	navioral		
,	1	I	
Port Name	Direction	MSB	LSB 🔺
d	in	3 3	
	in	7	
	in		
		_	
<	Back Next>	Cancel	Help

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

Define ¥HDL Source				×
Entity Name ledd	cd			
Architecture Name Beha	avioral			
Port Name	Direction	MSB	LSB	_
d	in	3	0	
s	in	6	0	
	in			-
	·			
		_		
< E	}ack Next≻	Cano	el H	elp 📗

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.

Define ¥HDL Source				×
Entity Name ledd	cd			
Architecture Name Beh	avioral			
Port Name	Direction	MSB	LSB	
d	in	3	0	
S	in 💌	6	0	
	in]		
	out			
	linout k	<u> </u>		
	in			_
	in			_
	in			_
	in			
	in			
	in			
	in			-
<	Back Next >	Cance	el Help	

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

New Source Information	×
Project Navigator will create a new skeleton source with the following specifications:	
Source Type: VHDL Module Source Name: leddcd.vhd Entity Name: leddcd Architecture Name: Behavioral Port Definitions:	<u> </u>
d vector: 3:0 in s vector: 6:0 out	
Source Directory: C:\tmp\fpga_designs\design1	×
< Back Finish Cancel	Help

After clicking on Finish, the editor pane of the **Project Navigator** window displays a VHDL skeleton for our LED decoder. (We also see the leddcd.vhd file has been added to the Sources pane.) Lines 1-4 create links to the IEEE library and packages that contain various useful definitions for describing a design. The LED decoder inputs and outputs are declared in the VHDL entity on lines 11-14. We will describe the logic operations of the decoder in the architecture section between lines 18 and 21.



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



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

Checking the VHDL Syntax

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



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

Fixing VHDL Errors

We can find the location of the error by scrolling the log pane at the bottom of the **Project Navigator** window until we find an error message. In this case, the error is located on line 25. You can manually scroll to line 25 in the editor pane, or you can double-click on the error message in the log pane to go directly to the erroneous line.



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



Xilinx - Project Navigator - C:\tmp\fpga_designs\de File Edit View Project Source Process Window		- D X > - B X
		•
Sources in Project: design 1 Carter Science: Module View Snapshot View Library View Processes for Current Source: User Constraints User Constraints View Synthesis Report View RTL Schematic Analyze Hierarchy	17 18 begin 19 20 s <= "1110111" when d="0000" else	
Check Syntax	Neddod.vhd	
	esigns/design1/leddcd.vhd in Library work. a_designs/design1/leddcd.vhd Line 37. parse error	, unexper
Process 'xst_sprjTOstx.tclprojnav/xst_sprjTOstx_tcl.rsp' fail	led with code: 00004 Ln 25 Col 32	• <i>[</i>]

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

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

📚 Xilinx - Project Navigator - C:\tmp\fpga_designs\d	esign1\design1.npl - [leddcd.vhd]	
File Edit View Project Source Process Window	Help	» _ B ×
Ш	▙ \$ \$ \$ \$	•
Sources in Project: Carter and the sign of the sign o	17 18 begin 19 20 s <="1110111" when d="0000" else	•
User Constraints User Constraints Synthesize View Synthesis Report View RTL Schematic Analyze Hierarchy Check Syntax The Check Syntax Process View	32 "0001101" when d="1100" else 33 "0011111" when d="1101" else 34 "1101101" when d="1110" else 35 "1101100"; 36 "1101100"; 38 • • • • • • • • • • • • • • • • • • • • • • •	×
Entity <leddcd> (Architecture <be Completed process "Check Syntax".</be </leddcd>	havioral>) compiled.	
Console Find in Files		
Process "Check Syntax" is up to date.	Ln 35 Col 14	

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.



The synthesizer will read the VHDL code and transform it into a netlist of gates. This will take les than 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 device utilization and timing statistics for the synthesized design.

Implementing the Logic Circuitry in the FPGA

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



You can watch the progress of the implementation process in the status bar at the bottom of the **Project Navigator** window. For a simple design like the LED decoder, the fitting is completed in seconds (on a 850 MHz Athlon PC with 768 MBytes). 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 Map process decomposes the netlist and rearranges it so it fits nicely into the circuitry elements contained in the FPGA device we selected. Then the Place & Route process assigns the mapped elements to specific locations in the FPGA and sets the switches to route the logic signals between them. If the Implement Design provcess had failed, a X would appear next to the subprocess where the error occured. You

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



Checking the Implementation

We have our design fitted into the XC2S100 FPGA, but how much of the chip does it use? Which pins are the inputs and outputs assigned to? We can find answers to these questions by double-clicking on the Place & Route Report and the Pad Report in the Process pane.



The device utilization of the LED decoder circuit can be found near the top of the place & route report. The circuit only uses 4 of the 3072 available slices in the XC2S300E FPGA. Each slice contains two CLBs and each CLB can compute the logic function for one LED segment output.

Device utilization summary:

Number of Number	External of LOCed			out out		142 11	7응 0응
Number of	SLICEs		4	out	of	3072	1%

The pad report shows what pins the inputs and outputs use. The **d** inputs have been assigned to pins 40, 44, 36 and 49. The **s** outputs which drive the LED segments have been routed through pins 45, 42, 41, 46, 43, 47 and 56. (The pad report was edited to remove unused pins and fields so it would fit into this document.)

						1
Pin Number	Signal Name	Pin Usage	Pin Name	Direction	IO Standard	i
P36	d<2>	IOB	IO_L41N_YY	INPUT	LVTTL	
P40	d<0>	IOB	IO L40P Y	INPUT	LVTTL	
P41	s<2>	IOB	IO_VREF_6 L40N Y	OUTPUT	LVTTL	
P42	s<1>	IOB	110	OUTPUT	LVTTL	
P43	s<4>	IOB	IO	OUTPUT	LVTTL	
P44	d<1>	IOB	IO	INPUT	LVTTL	
P45	s<0>	IOB	IO VREF 6 L39P	OUTPUT	LVTTL	
P46	s<3>	IOB	IO_L39N	OUTPUT	LVTTL	
P47	s<5>	IOB	IO VREF 6	OUTPUT	LVTTL	
P49	d<3>	IOB	IO_L38N_YY	INPUT	LVTTL	
P56	s<6>	IOB	IO_L37P_YY	OUTPUT	LVTTL	

Assigning Pins with Constraints

The problem we have now is that the inputs and outputs for the LED decoder were assigned to pins picked by the implementation process, but these are not the pins we actually want to use on the XSB-300E Board. The FPGA on the XSB-300E Board has eight inputs which are driven by a DIP switch and we would like to assign the LED decoder inputs to four of these as follows:

LED Decoder Input	XSB XC2S300E FPGA Pin	
d0	P110	
d1	P111	
d2	P112	
d3	P113	

Likewise, the XSB-300E Board has two seven-segment LEDs, one of which is attached to the following pins of the FPGA:

LED Decoder Output	XSB XC2S300E FPGA Pin	
s0	P135	
s1	P141	
s2	P126	
s3	P116	
s4	P145	
s5	P120	
s6	P153	

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

💸 Xilinx - Project Navigator - C:\tmp\fpga_designs\design1\design1.npl - [leddcd.pad_txt (REAI		
li File Edit View Project Source Process Window Help	» _∃'×	
Sources in Project: 45 P41 \$<2> esign1 46 P42 \$<1> xc2s300e-6pq208 · XST VHDL 48 P44 d<1> xc2s300e-6pq208 · XST VHDL 49 P45 \$<0> xc2s300e-6pq208 · XST VHDL 49 P45 \$<0> xc2s300e-6pq208 · XST VHDL 49 P44 d<1> xc2s300e-6pq208 · XST VHDL 49 P45 \$<0> Module View sra Add Source Insert Add Source Add Source Shift+Insert Remove Delete P50 P51 P52 Move to Library P53 Design Entry Open P57 User Constra Toggle Paths P57 P57 Synthesize P59 Sequence 64 P60 65 P61 66 66 P61 66 67 P60 66 68 P62 Image: Path P10 P10 Renove Eddcd.vhd Eddcd.pad_t	IOB IOB IOB IOB IOB IOB IOB IOB IOB IOB	
Release 5.2.03i - PACE F.31 Copyright (c) 1995-2002 Xilinx, Inc. All rights reserved. Console Find in Files		
Add a new source to the project	.n 61 Col 1 🛛 💽 🎢	
Select Implementation Constraints File as the type of source file we want to add and type leddcd in the File Name field. Then click on the Next button.

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

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

Select	×
Associate with Source	
leddod	
< Back Next > Cancel Help	

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

New Source Information	×
Project Navigator will create a new skeleton source with the following specifications:	
Source Type: Implementation Constraints File Source Name: leddcd.ucf Association: leddcd	Ă
	T
Source Directory: C:\tmp\fpga_designs\design1	
< Back Finish Cancel	[
Kernel Cancel	Help

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

😹 Xilinx - Project Navigator - C:\tmp\fpga_designs\	design1\design1.npl - [leddcd.pad_txt (READ ONL	Y)]
File Edit View Project Source Process Window	Help	» _ d ×
	■ 🔉 🗞 🦻 🕺 🕺 🖬 🖬 🗠 🗠	
Sources in Project: Cources in Project: Cources in Project: Cources in Project: Cources: Cources: Cources: Cource	$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$	IOB IOB
Process View	61 P57 62 P58 63 P59 64 P60 65 P61 66 P62 ✓ ✓ ► Eddcd.vhd È leddcd.pad_t	IOB IOB IOB IOB IOB IOB
Release 5.2.031 - PACE F.31 Copyright (c) 1995-2002 xilinx,	Inc. All rights reserved.	
Hierarchy is up to date.	Ln 61 (Col 1 💽 🥠

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

Xilinx Constraints Editor - [Ports - le File Edit View Window Help	eddcd.ngd / leddc	d.ucf]		
	? №			
	<u> </u>			1
Port Name	Port Direction	Location	Pad to Setup	Clock to
d<0>	INPUT	P50		N/A
d<1>	INPUT	P48		N/A
d<2>	INPUT	P42		N/A
d<3>	INPUT	P47		N/A 🔍
•			·	
I/O Configuration Options Prohibit I/O Locations Global Ports	Pad Groups Group Name: Select Group: Advanced	Misc	Pad to Setup.	
NET "s<2>" LOC = "P62"; NET "s<3>" LOC = "P60"; NET "s<4>" LOC = "P46"; NET "s<5>" LOC = "P46"; NET "s<5>" LOC = "P57"; NET "s<6>" LOC = "P49"; UCF Constraints (read-write) UCF C	Constraints (read-only)	Source Constrai	nts (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: P110. Do this for each of the inputs and outputs using the pin assignments from the tables shown previously. After doing this, the **Constraints Editor** window appears as follows.

d<0> INPUT P110 N/A d<1> INPUT P111 N/A d<2> INPUT P112 N/A d<3> INPUT P113 N/A s<0> OUTPUT P135 N/A INPUT s<1> OUTPUT P141 N/A Integer s<2> OUTPUT P166 N/A Integer s<3> OUTPUT P145 N/A Integer s<4> OUTPUT P145 N/A Integer s<5> OUTPUT P153 N/A Integer s<6> OUTPUT P153 N/A Integer I/0 Configuration Options Group Name: Create Group Pad to Setup Belect Group: Select Group: Pad to Setup Clock to Pad	Port Name	Port Direction	Location	Pad to Setup	Clock to
d<2> INPUT P112 N/A d<3> INPUT P113 N/A s<0> OUTPUT P135 N/A s<1> OUTPUT P141 N/A s<2> OUTPUT P126 N/A s<3> OUTPUT P146 N/A s<3> OUTPUT P145 N/A s<4> OUTPUT P145 N/A s<5> OUTPUT P153 N/A s<6> OUTPUT P153 N/A	d<0>	INPUT	P110		N/A
3<3> INPUT P113 N/A s<0> OUTPUT P135 N/A s<1> OUTPUT P141 N/A s<2> OUTPUT P126 N/A s<3> OUTPUT P116 N/A s<4> OUTPUT P145 N/A output P145 N/A s<5> OUTPUT P120 output P133 N/A	¦<1>	INPUT	P111		N/A
Sector OUTPUT P135 N/A Sector OUTPUT P141 N/A Sector OUTPUT P126 N/A Sector OUTPUT P126 N/A Sector OUTPUT P116 N/A Sector OUTPUT P145 N/A Sector OUTPUT P145 N/A Sector OUTPUT P120 N/A Sector OUTPUT P153 N/A Pad Groups Group Name: Create Group Pad to Setup Pad to Setup Pad to Setup	! <2>	INPUT	P112		N/A
Sec1> OUTPUT P141 N/A Sec2> OUTPUT P126 N/A Sec3> OUTPUT P116 N/A Sec4> OUTPUT P145 N/A Sec5> OUTPUT P120 N/A Sec6> OUTPUT P153 N/A I/O Configuration Options Pad Groups Create Group Pad to Setup Select Group: Pad to Setup	1<3>	INPUT	P113		N/A
s<2> OUTPUT P126 N/A s<3> OUTPUT P116 N/A s<4> OUTPUT P145 N/A s<5> OUTPUT P120 N/A s<6> OUTPUT P153 N/A Pad Groups Group Name: Create Group Pad to Setup	s<0>	OUTPUT	P135	N/A	
S<3> OUTPUT P116 N/A S<4> OUTPUT P145 N/A S<5> OUTPUT P120 N/A S<6> OUTPUT P153 N/A Image: Create Group Pad Groups Group Name: Create Group Prohibit I/O Locations Select Group: Pad to Setup	s<1>	OUTPUT	P141	N/A	
s<4> OUTPUT P145 N/A s<5> OUTPUT P120 N/A s<6> OUTPUT P153 N/A Pad Groups Group Name: Create Group Pad to Setup Pad to Setup	s<2>	OUTPUT	P126	N/A	
s<5> OUTPUT P120 N/A s<6> OUTPUT P153 N/A Pad Groups Group Name: Create Group Pad to Setup Pad to Setup	s<3>	OUTPUT	P116	N/A	
s<6> OUTPUT P153 N/A	s<4>	OUTPUT	P145	N/A	
Pad Groups Group Name: Create Group Pad to Setup Pad to Setup	s<5>	OUTPUT	P120	N/A	
Pad Groups I/O Configuration Options Prohibit I/O Locations Select Group: Pad to Setup	s<6>	OUTPUT	P153	N/A	
I/O Configuration Options Group Name: Create Group Prohibit I/O Locations Select Group: Pad to Setup	•				
		Group Name:		Pad to Setup	

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

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

Xilinx - Project Navigator - C:\tmp\fpga_designs\de		
File Edit View Project Source Process Window	Help	» <u> </u>
	E 🔉 🧞 🦹 🛠 🕺 Å 🛍 🖻 🗠 ↔ 🗛 🗖	•
Sources in Project: design1 design1 dedicd (leddcd vhd) dedicd (leddcd vhd) dedicd ucf Module View Snapshot View Library View Processes for Current Source: Design Entry Utilities User Constraints Synthesize Generate Programming File	46 P41 S<2> 46 P42 S<1> 47 P43 S<4> 48 P44 d<1> 49 P45 S<0> 50 P46 S<3> 51 P47 S<5> 52 P48 53 P49 d<3> 54 P50 56 P51 56 P51 57 P53 58 P54 59 P55 60 P56 S<6> 61 P57 62 P58 63 P59 64 P60 65 P61 66 P62	IOB IOB IOB IOB IOB IOB IOB IOB IOB IOB
C Process View	leddcd.vhd 📄 leddcd.pad_t	
NGDBUILD done. Completed process "Translate".		× ×
Process "Launching Constraints Editor to edit UCF File" is up to o	date. Ln 61 Col 1	

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

Pin Number	Signal Name	Pin Usage	Pin Name	Direction	IO Standard
P110	d<0>	IOB	IO	INPUT	LVTTL
P111	d<1>	IOB	IO_VREF_3_L23N_Y	INPUT	LVTTL
P112	d<2>	IOB	IO_L23P_Y	INPUT	LVTTL
P113	d<3>	IOB	IO	INPUT	LVTTL
P116	s<3>	IOB	IO_D6_L22P_Y	OUTPUT	LVTTL
P120	s<5>	IOB	IO_D5_L21N_YY	OUTPUT	LVTTL
P126	s<2>	IOB	IO_D4_L19P_Y	OUTPUT	LVTTL
P135	s<0>	IOB	IO_D3_L17N_Y	OUTPUT	LVTTL
P141	s<1>	IOB	IO_D2_L15P_YY	OUTPUT	LVTTL
P145	s<4>	IOB	IO_D1_L14N_Y	OUTPUT	LVTTL
P153	s<6>	IOB	IO_DIN_D0_L12N_YY	OUTPUT	LVTTL

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

Viewing the Chip

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



The FloorPlanner window will appear containing three panes:

- 1. The **Design Hierarchy** pane lists the LED decoder inputs, outputs and LUTs assigned to the various CLBs in the FPGA.
- 2. The **Design Nets** pane lists the various signal nets in the LED decoder.
- 3. The **Placement** pane shows the 32 × 48 array of slices in the FPGA. The I/O pins are also shown around the periphery. (The pins used for Vcc, GND, and programming are not shown.)

🕅 Xilinx Floorplanner - leddcd.fnf	<u> </u>
File Edit View Hierarchy Pattern Floorplan Window Help	
│ D ☞ 🖬 ⊜ 🤋 🕺 ■ 슭 🤁 ☜ ☞ 의 🦑 ❣ 🥨 ⋈ 🎥 🔏 ☜ ⊠ ↔ ♥ 🖉	
eddcd.fnf Design Hierarchy	
leddcd "leddcd" [11 IOBs, 7 FGs leddcd.fnf Placement for XC25300E-6-PQ208	
mrom_s_inst_lut4_61 [FG] as	
mrom_s_inst_lut4_51 [FG] os	
mrom_s_inst_lut4_41 [FG] o.s_	
mrom_s_inst_lut4_31 [FG] os	
mrom_s_inst_lut4_21 [FG] ors	
mrom_s_inst_lut4_11 [FG] os	
mrom_s_inst_lut4_01 [FG] as	
s<6> [10B] 0:s_6_obuf s<5> [10B] 0:s_5_obuf	
s<3>[10B]0:s_3_0000	
sty [105] 0.s_4_obui	
stor [lob] 0:s_0_obul	
s<1> [IOB] 0:s 1_obuf	
s<0> [10B] 0:s_0_obuf	
d<3> [10B] l:d_3_ibuf	
leddcd.fnf Design Nets	
s 5 obuf	
s_3_obuf	
s_2_obuf s 1 obuf	
s_0_obuf	
s_6_obuf	<u>↑</u>
	+ -
R33C1	

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

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

🐼 Xilinx Floorplanner - leddcd.fnf		- D ×
File Edit View Hierarchy Pattern Floorplan Window	Help	
🛯 🗅 🖨 🗑 🖇 😢 🔳 🕼 🕄 🔁 😵	& \$ ⊈ ⋈ \$ ∡ 4 ♥ X Q Q X Q \$	
leddcd.fnf Design Hierarchy	ldcd.fnf Editable Floorplan for XC25300E-6-PQ208(UCF Flow)	
📙 📮 🚬 leddcd "leddcd" (11 IOBs, 7 FGs 🗖 📗 [leddcd.fnf Placement for XC25300E-6-PQ208	
-> mrom_s_inst_lut4_61 [FG] o:s_		E
mrom_s_inst_lut4_51 [FG] o:s_		
} mrom_s_inst_lut4_41 [FG] α.s_		
mrom_s_inst_lut4_31 [FG] as		
mrom_s_inst_lut4_21 [FG] o:s_		
mrom_s_inst_lut4_11 [FG] o:s_		
mrom_s_inst_lut4_01 [FG] o:s_		
s<6> [IOB] 0:s_6_obuf		
s<5> [IOB] 0:s_5_obuf		
s<4> [IOB] 0:s_4_obuf		
s<3> [IOB] 0:s_3_obuf		
s<2> [10B] 0:s_2_obuf		
s<1> [IOB] 0:s_1_obuf		
s<0> [10B] 0:s_0_obuf	D D [] P122	
d<3> [10B] l:d_3_ibuf		
💐 leddcd.fnf Design Nets 📃 🗖 🗙		
s_4_obuf		
s_3_obuf s 2 obuf	P120	
s_1_obuf	P116	
s 6 obuf	P115	
d_3_ibuf		
	•	
	R6C1	

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

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

Now clicking on a CLB will cause the inputs and output for the CLB to appear as shown below.



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



By default, the **Placement** pane only shows FPGA resources that are used by the design. To see all the logic resources in each CLB, click on the Resources tab and check all the boxes as shown below.

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

Now the **Placement** pane shows all the LUTs, flip-flops, carry-chains and tristate buffers included in each CLB.

Xilinx Floorplanner - leder File Edit View Hierarchy	dcd.fnf Pattern Floorplan Win	ndow Help					
D 🗳 🖬 🎒 📍 📢	🛛 🔳 🕼 🔁 🔁	Þ 🕫 🧶 🖠	伝 翰 🖁	k 🚄 😵 .	XQQX	۹ 🗈	
👔 leddcd.fnf Design Hiera	rchy	📕 leddcd.fnf	Editable Floorp	lan for XC25	300E-6-PQ208	(UCF Flow)	
11 1 3 3	ut4_41 [FG] o:s_	📕 leddcd.	fnf Placement	for XC25300	E-6-PQ208		
	ut4_31 [FG] o:s_ ut4_21 [FG] o:s						
10-01-3-5	ut4_11 [FG] o:s(
11 1 1 1	ut4_01 [FG] o:s		<u> </u>			<u> </u>	
s<6> [10B] 0	s_6_obuf				VIIIIIX.		
s<5> [IOB] 0		Ц <u>4</u> 3	l G f		CANKA -		
s<4>[IOB]0							
s<3> [10B] 0							
s<2> [IOB] 0 s<1> [IOB] 0					VIIIXXIIIN		
s<1>[108]0							
d<3>[108]		1 .	4		1		
d<2>[IOB]					F M		
d<1>[IOB]	1_1_ibuf		11				
d<0> [IOB] [:	1_0_ibuf 🔽 🗌		1/			עביים די	
						l)	
💐 leddcd.fnf Design Nets					·	\mathbb{N}	
s_5_obuf							
s_4_obuf s_3_obuf		LF &				- N - Y - A -	
s_2_obuf		L L L				114	
s_1_obuf s_0_obuf						$\geq 1 \wedge 1$	
s_6_obuf d 3 ibuf							
d_2_ibuf	- II	Ŀ					-
d 1 ibuf						1 1	
<u>.</u>					R32C0		

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

Generating the Bitstream

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



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

Xilinx - Project Navigator - C:\tmp\fpga_designs\d File Edit View Project Source Process Window		(READ ONLY)]	- 🗆 ×	
	n 🔁 🔁 🗞 🖇 🕅 🖻 🖻	222 A	•	
Sources in Project: design1 Carlot Stresson St	141 P137 142 P138 143 P139 144 P140 145 P141 146 P142 147 P143 148 P144 149 P145 150 P146 151 P147		IOB IOB IOB IOB IOB IOB IOB	
Module View Snapshot View Library View Processes for Current Source: Design Entry Utilities User Constraints Synthesize Implement Design Generate Programming File Generate PROM, ACE, or JTAG File Configure Device (iMPACT)	152 P148 153 P149 154 P150 155 P151 156 P152 157 P153 158 P154 159 P155 160 P156 161 P157 162 P158 163 P159 164 P160 165 P161 168 P162 168 P162 169 P163 169 P165		IOB IOB IOB IOB IOB IOB IOB IOB	
Process View	leddcd.vhd leddcd.pad_t			
Bitstream generation is complete. Completed process "Generate Programming File".				
Process "Generate Programming File" is up to date.		Ln 158 Col 1	🔶 /h	

Downloading the Bitstream

Now we have to get the bitstream file programmed into the FPGA on the XSB-300E Board. The XSB-300E Board is powered with an ATX (or 9VDC) power supply and is attached to the PC parallel port with a standard 25-wire cable as shown below.





The XSB-300E Board is programmed using the gxsload utility. We double click the GX5LOAD icon to bring up the **gxsload** window:

<mark>X</mark> gxsload		_ 🗆 🗙
	3-300E 💌	Load Exit
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\fpga_designs\design1 in this case). We just drag-and-drop the leddcd.bit file from the **design1** window into the **gxsload** window.

🔁 C:\tmp\fpga_designs\design1	_ D ×		🗙 gxsload
File Edit View Favorites Tool	s Help 🏥		Board Type XSB-300E Load
ົ⇔•⇒∘©I©I©S%	6 B B X 10 🔳 -		Port LPT1 Exit
Address 🗋 design1	▼ 🔗 Go		
🛋 _prepar.rsp	🛋 leddcdprj		FPGA/CPLD RAM Flash/EEPROM
jprepar.rsp ≣automake.log	🔊 leddcdsprj		leddcd.bit
🔊 bitgen.rsp	🔊 leddcd.ana		
🔊 bitgen.ut	🗃 leddcd.bgn		
🔊 chkdata.err	💼 leddcd.bit		
🔊 design1.jid	🛋 leddcd.bld		High Address
design1.npl	🛋 leddcd.cel		
🔊 fpga.cel	🔊 leddcd.cmd_log		Low Address
🔊 fpga.ucf	🛋 leddcd.cup		Upload Format 🛛 🖛 🦳 🔂 🖂 🗖
	•		
Type: BIT File Size: 229 229 KB	🖳 My Computer		
17700 011 1 10 01201 229 (229 KD	C inv compater	2	

Then we click on the Load button to initiate the programming of the FPGA. Downloading the leddcd.bit file to the XSB-300E takes only a few seconds.

🔀 gxsload		
	3-300E 💌	Load Exit
FPGA/CPLD	RAM	Flash/EEPROM
leddcd.bit		
High Address		
Low Address		
Upload Format	HEX 💌 🗋	HEX 💌 🗋

Testing the Circuit

After the FPGA on the XSB-300E Board is programmed, we can test the LED decoder by placing different patterns on the input pins and viewing the response on the seven-segment LED digit. Changing the settings of DIP switches 1–4 will change the numeral displayed on the LED digit as shown below. (Note that each DIP switch forces a logic 1 on the FPGA input pin when it is pushed downward into the OFF position. It forces a logic 0 when it is pushed upward into the ON position.)





Hierarchical Design

A Displayable Counter

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



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

Starting a New Design

We can start a new project using the File→New Project... menu item. We name the project *design2* and store it in the same folder as the previous design: C:\tmp\fpga_designs. The other properties in the **New Project** window retain the same values we set in the previous project.

New Project	×
Project Name: design 2	Project Location: C:\tmp\fpga_designs\design2
Project Device Options: Property Name	Value
Device Family	Spartan2E
Device	xc2s300e
Package	pq208
Speed Grade	-6
Design Flow	XST VHDL
	OK Cancel Help

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

🗞 Xilinx - Project Navigator - C:\tmp\fpga_designs\design2\design2.npl	- D ×
File Edit View Project Source Process Window Help	
	•
Sources in Project: Constraints of the state of the stat	
Empty Log)	A -
Console Find in Files	
Hierarchy is up to date.	- 🔶 //.

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 xc2s300e-6pq208 object in the Sources pane and selecting Add Source ... from the pop-up menu.

እ Xilinx - Project Navigator - C:\tmp\fpga_designs\design2\design2.npl	- D ×
File Edit View Project Source Process Window Help	
	•
Sources in Project: design2 Ke223300e-6pq208 New Source Add Source Add Copy of Source Remove Delete Move to Library,	
Image: Comparison of the second se	
Processes for Current SouPropercies	
(Empty Log)	×
Add a file from another project	• <i>[i</i> ,

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

Add Existing 9	Sources	<u>?</u> ×
Look in: 🔂	design1 💌 🗢 🗈 📸 🎹 🕇	
projnav	🗐 Valid_Info.txt	
🔊 fpga.ucf		
eddcd.ucf		
	•	
I		
File name:	leddcd.vhd Oper	
Files of type:	Sources (*.txt;*.vhd;*.sch;*.tbw;*.bmm;*.elf;*.i 💌 Canc	

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

Choose Source Type	×
leddcd, vhd is which source type? The suffix is ambiguous as to type.	
VHDL Module VHDL Package VHDL Test Bench	OK Cancel Help

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\fpga_designs\design2\design2.npl		
File Edit View Project Source Process Window Help		
	•	
Sources in Project: design 2 xc2s300e-6pq208 - XST VHDL Wieddod (Vdesign1Vieddod vhd) Constraints Design Entry Utilities User Constraints Synthesize Design Entry Utilities Constraints Synthesize Constraints Con		
Contraction of the second seco		
Scanning c:\tmp\fpga_designs\design1\leddcd.vhd Writing leddcd.jhd.	× V	
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 xc2s300e-6pq208 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 BMM File MEM File Implementation Constraints File State Diagram	File Name: counter Location: C:\tmp\fpga_designs\design2
< Back Next >	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.

efine ¥HDL Source							
Entity Name co Architecture Name Be		1					
Port Name		Direction	Ι	MSB		LSB	<u> </u>
clk	in						
count	out		3		0		
	in						
	in						
	in						
	in						
	in						
	in						
	in						
	in						
	lin				1		
< E	Back -	Next >		Cancel		Help	1

Click on Next and check the information about the module.

New Source Information	×
Project Navigator will create a new skeleton source with the following specifications:	
Source Type: VHDL Module Source Name: counter.vhd Entity Name: counter Architecture Name: Behavioral Port Definitions: clk scalar in count vector: 3:0 out	×.
<u> </u>	
Source Directory: C:\tmp\fpga_designs\design2	
< Back Finish Cancel	Help

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

```
library IEEE;
                                                           2
     use IEEE.STD_LOGIC_1164.ALL;
 з
     use IEEE.STD_LOGIC_ARITH.ALL;
 4
    use IEEE.STD_LOGIC_UNSIGNED.ALL;
 5
 6
         Uncomment the following lines to use the decla
         provided for instantiating Xilinx primitive co
 7
 8
     --library UNISIM;
 9
     --use UNISIM.VComponents.all;
10
     entity counter is
Port ( clk : in std_logic;
11
12
                 count : out std_logic_vector(3 downto (
13
14
     end counter;
15
16
     architecture Behavioral of counter is
17
       signal cnt: std_logic_vector(27 downto 0);
18
     begin
       process(clk)
19
20
       begin
21
         if clk'event and clk='0' then
22
           cnt <= cnt + 1;
23
         end if;
24
       end process;
25
       count(3 downto 0) <= cnt(27 downto 24);</pre>
26
     end Behavioral;
27
•[]
                                                         ۲
⊡
   counter.vhd
```

Line 12 declares a 28-bit signal, **cnt**, that is the current value of the counter. The process on lines 19-24 controls when the counter increments. The condition clause of line 21 is only true when the value on the **clk** input goes from 0 to 1. Then the statement on line 22 replaces the value in **cnt** with its incremented value. (We can use the high-level addition operator instead of having to describe a 28-bit adder because on line 4 we have linked into the ieee.std_logic_unsigned.all package that supports unsigned arithmetic.) Finally, line 25 places the upper four bits of the current counter value onto the outputs of the module.

Why are we building a 28-bit counter and using only the upper four bits? The counter will be driven by the oscillator on the XSB-300E 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 10⁶) = 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.



Tying Them Together

We have the LED decoder and the counter, but now we need to tie them together to build the displayable counter. We will do this by connecting the counter to the LED decoder in a top-level schematic. Before we can do this, we have to create schematic symbols for both the counter and LED decoder 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.

Xilinx - Project Navigator - c:\tmp\fpga_d		
I File Edit View Project Source Process		리지
	1 🖪 🖪 🎘 😵 🕺 🕺 🖻 🗟 🗠 🗠 🙀 🗠 n	•
Sources in Project: design2 xc2s100-5tq144-XST VHDL counter (counter.vhd) leddcd (C:\tmp\fpga_designs\desic Module	<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;</pre>	A
Processes for Current Source: Design Entry Utilities User Constraints Create Schematic Symbol Launch ModelSim Simulatc View VHDL Instantiation To Synthesize Implement Design Generate Programming File Process View	<pre>architecture Behavioral of counter is signal cnt: std_logic_vector(27 downto 0); begin process(clk) begin if clk'event and clk='0' then cnt <= cnt + 1; end if; end process; count(3 downto 0) <= cnt(27 downto 24); end Behavioral; counter.vhd </pre>	¥.
	ţ	÷ •
Process "Create Schematic Symbol" is up to date.	Ln 15, Col 10	• //

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 xc2s300e-6pq208 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 BMM File MEM File Implementation Constraints File State Diagram	File Name: disp_ont Location: C:\tmp\fpga_designs\design2
< Back Next >	Cancel Help

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

New Source Information	×
Project Navigator will create a new skeleton source with the following specifications:	
Source Type: Schematic Source Name: disp_cnt.sch	A
T	▼
Source Directory: C:\tmp\fpga_designs\design2	
< Back Finish Cancel	Help
Now the disp_cnt schematic object has been added to the Sources pane. We can double-click it to begin creating the schematic, but a schematic editor window should open automatically once the file is created.



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



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

Context Context (Context Context Conte	w He	elp				-	-		-	-										미× 리×
D 😂 🖬 🍠 🎒 👯 X 🖻	C.	0 C			Q (0)	X	X	Q [\$		Ē		٦.							
♥ ♥ ♥ ► \$ \$ ₺		0 \		A	?	Ç ⊿	k ti	• •	/											
Options Symbols	IF	1		2		3			6		5		6	;		7			8	
<u>C</u> ategories	Ш																			
<-All Symbols-> <c: design2="" fpga_designs="" tmp=""></c:>	A					•														A
Arithmetic Buffer Carry Logic				· ·	•	•	· ·	•	•	•	•	· ·		•			•	•	•	
Comparator Counter	Ш																			
Decoder Flip_Flop ▼			•	· ·	•	•	· ·	•	•	•	•	· ·	•	•	•	•	•	•	•	8
	Ш																			
Symbols counter] .															•			
leddod 🔥			•	· ·	•	•	· ·	•	•	•	•	· ·		•					•	с
		.																		
	Iŀ	1:	•	•••	•	•	• •	•	•	•	•	• •	•	•	•	•	•	•	•	
Symbol <u>N</u> ame Filter																				
						•													•	
<u>O</u> rientation		1		2		3			£.		5		6	5		7			8	
Rotate 0	Ľ	disp_	_cnt.sch																	
																		[188	30,-80	0] //

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.







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

Xilinx ECS - [disp_cnt.sch]	N I	Help)			-												_ D ×
 D 😅 🖬 🗿 🚭 🛠 % 🗈 (Ŋ	C)			€	Q)	< X	Q [•	-	e é						
11 11 11 11 11 11	$\overline{}$		>] A	. ?] Ç	⊿⊾ 1.	↓ √	·								
Options Symbols		·	_	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
Add Wire Options		·		•														·
When you add a wire		·		•		•		·		•		•		•		•		•
 Use the Autorouter to add one or more line segments 		.						•										
between the points you indicate		·		·		•		•		·		•		•	•	·		· II
C Use the Manual method to add single line segments between the points you		·					our	nter				lec	ddc	d ·				·
indicate		·		·	a	cik		count(3:0	<u> </u>	}{		(3:0)		s(81)	=	•		·
Use a press - drag - release action to specify a single pair of wire points.		·		•		•												·
Use a click - click action to specify a connected sequence of wire points.		·		·		•		•		·		•		•		·		·
Terminate the sequence with a double click.		·		•		•									•			·
		·		·		•		•		•		•		•	•	•		•
		·		•											•			·
				·	·	•		•	•	•		•		·	•	·	·	ا ت
	.		disp_	ont.s	ch													
Ready		_															[1653,	1301] //

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. Click on the Add an output marker button in the Options pane and then click on the free end of the wire segment that we just added.



Then click on the other end of the newly-added bus segment to create a byte-wide set of output pins.

Xilinx ECS - [disp_cnt.sch] File Edit View Add Tools Windo	lelp		_ 므
-	∽ ∝ M Q Q X X Q @	4 6 2 4 4 0	
▼ 1 ♥ ♥ ► ☜ ☵ ≞	○ ∖ 🗆 A 🛛 ♀ ₄ 🖬 🖌 🧹		
Dptions Symbols	• • • • • • •	• • • • •	
Select Options			
When you click on a branch	• • • • • • • •	• • • • • •	
 Select the entire branch Select the line segment 			
	• • • • • • • •	• • • • • •	•
When you move an object • Keep the connections to other	leddcd		
objects Break the connections to		XLXN_5(7:0)	•
other objects		. k	
When you use the area select tool, select the objects that	• • • • • • • •	• • • • • •	· [
Are enclosed by the area Intersect the area			
	• • • • • • • •	• • • • •	•
When you use the area select tool, select			
 Objects excluding attribute windows 	• • • • • • • •	• • • • •	• •
C Attribute windows only	disp_cnt.sch		
ady		[1]	826,1309]

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.

File Edit View Add Tools Wine		Help			1.00					1.0								_ 6
) 😂 🖬 🗿 🚭 🌾 🗍 % 🖻) C	ŝ	CH	M]]]	€€	XX	X	R 🗘]]]]		Ì 🗖		= 0				
. ↓ ♥ ☜ ┝ ☜ ☵ ∄	- ~	N C	\sim		Α	?	Ģ ⊿	⊾ t∔	\checkmark									
	×	_																
tions Symbols			•		•		·	·	·	•	•		•		·	·	·	
Select Options																		
Select Options																		
When you click on a branch			•		•		•		•		•		•		·		•	
 Select the entire branch 																		
Select the line segment																		
			•		•		•		•		•		•		·		•	
When you move an object		Ι.	ام م	- L - L -	Л.													
Keep the connections to other		\square	ea	dco	<u>ار</u>		Ċ		IE8									
objects © Break the connections to		H(3:0)		:(CD)		-È	\$≥	<u> </u>		XLXN	1517	Cut				•	
other objects													Сору					
													Paste					
When you use the area select			•		•		•		•		·		Delete	,			•	
tool, select the objects that													Zoom			•		
Are enclosed by the area Intersect the area				·			·			·		_	Select	Object	(s)			
			•		•		•		•		•		Select		(-)		•	
Authors and the same sets t													Unsele	ect All				
When you use the area select tool, select				-	-	-	-	-	-				Mirror				-	
Objects excluding attribute			•	•	•	•	•		•	•	•		Rotate	в			•	ŀ
windows Attribute windows only		┛										_	Renan	ne Port				▶
- The action of the second second			lien o	nt.scł	. Г							_						

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

B Object Properties				×
Category	Viet	Net Attri w and edit the attribut		ed nets
^I XLXN_5(7:0)	Name	Value	Visible	New
	Name	XLXN_5(7:0)	Add	
	PortPolarity	Output	Add	Edit Traits
				Delete
	ОК	Cancel	Apply	Help

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

Content in the second s				×
Category				
□Nets ↓XL×N_5(7:0) □IO Markers	Viev	Net Attri w and edit the attribut		ed nets
XLXN_5(7:0)	Name	Value	Visible	New
	Name	S(6:0)	Add	
	PortPolarity	Output	Add	Edit Traits
				Delete
	ОК	Cancel	Apply	Help



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

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

Xilinx ECS - [disp_cnt.sch] File Edit View Add Tools Wind	ow Help												_ D
Lu	Вю 0	M	€. €	\mathbf{x}	K 🔍 [\$	- e	1		3 00			
 ▼ J ლ ლ ┝ ☜ 🕮 ≞	1			Q AL	ti 🗸	1							
	-					R-		_					
)ptions Symbols	í I 🔽	• •		•	· .	Check 9	ichemati	ic .	•		•		•
Select Options	·				· ·								
When you click on a branch	11 ·	•		•	• •		•		•		•		
 Select the entire branch Select the line segment 												•	
When you move an object Keep the connections to other objects Break the connections to other objects	<mark>lec</mark> ແໜ	idcd (شهر)]]		BUE	8	- <u>S(6:0</u>	D		•	•		
When you use the area select tool, select the objects that	·	•		•	•		•		•		•		•
 Are enclosed by the area Intersect the area 		· ·		•	· ·	•	•	•	•				
When you use the area select tool, select													
 Objects excluding attribute windows 	· ·	•		•	•••		•		•		•		· •
C Attribute windows only		cnt.sch											
eck the active schematic												[2:	317,849]

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.

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

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



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



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



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

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

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



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

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

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

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



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



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 ^Y that appears after the Analyze Hierarchy process completes shows we have no major errors, but there are some warnings. These warnings concern the location of the VHDL file that defines the LED decoder; it has been changed as a result of generating the schematic symbol for this module. This warning is of no concern at this time, but may have to be considered later if we modify the VHDL file for the LED decoder since we want to make sure we get the right file.



Constraining the Design

Before synthesizing the displayable counter, we need to assign the pins which the inputs and outputs will use. We start by right-clicking the disp_cnt object in the Sources pane and selecting New Source... from the pop-up menu.



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

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

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

Select		×
Associate with Source		
counter leddcd		
disp_cnt		
		_
< Back	Next > Cancel Help	

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

New Source Information	×
Project Navigator will create a new skeleton source with the following specifications:	
Source Type: Implementation Constraints File Source Name: disp_cnt.ucf Association: disp_cnt	<u> </u>
Source Directory: C:\tmp\fpga_designs\design2	
< Back Finish Cance	el Help

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



In the Ports tab of the **Constraint Editor** window, set the pin assignments for the clock input and LED segment drivers as follows:

Xilinx Constraints Editor - [Ports -	disp_cnt.ngd / dis	p_cnt.ucf*]		_ 🗆 ×
	? ∖?			
Port Name	Port Direction	Location	Pad to Setup	Clock to Pa
clk	INPUT	P77	N/A	N/A
s<0>	OUTPUT	P135	N/A	
s<1>	OUTPUT	P141	N/A	
s<2>	OUTPUT	P126	N/A	
s<3>	OUTPUT	P116	N/A	
s<4>	OUTPUT	P145	N/A	
s<5>	OUTPUT	P120	N/A	
s<6>	OUTPUT	P153	N/A	
	Pad Groups Group Name: Select Group: Advanced	Misc	Create Group Pad to Setup Clock to Pad	
NET "s<5>" LOC = "P120"; NET "s<6>" LOC = "P153";	Constraints (read-only)	Source Constr	aints (read-only)	

Assigning the **clk** input to pin P77 lets us use the onboard oscillator of the XSB-300E Board to drive the counter. The output assignments connect the displayable counter to one of the seven-segment LEDs on the XSB-300E Board as in the previous design example.

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

Synthesizing the Logic Circuitry for the Design

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



Implementing the Logic Circuitry in the FPGA

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



Checking the Implementation

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

Device utilization summary:

Number of External GCLKIOBs	1 out of 4 25%
Number of External IOBs	7 out of 142 4%
Number of LOCed External IOBs	7 out of 7 100%
Number of SLICEs	18 out of 3072 1%

The displayable counter consumes 18 of the 3072 slices in the FPGA. Each slice contains two CLBs, so the displayable counter uses a maximum of 36 CLBs. The 28-bit counter requires at least 28 CLBs and the LED decoder requires 7 CLBs so this totals to 35 CLBs.

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



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

Pin Number	 Signal Name	 Pin Usage	 Pin Name	 Direction	 IO Standard
 Р77			 GCK1	 INPUT	 LVTTL
P120	s<5>	IOB		OUTPUT	LVTTL
P126	s<2>	IOB	IO D4 L19P Y	OUTPUT	LVTTL
P135	s<0>	IOB	IO_D3_L17N_Y	OUTPUT	LVTTL
P141	s<1>	IOB	IO_D2_L15P_YY	OUTPUT	LVTTL
P145	s<4>	IOB	IO_D1_L14N_Y	OUTPUT	LVTTL
P153	s<6>	IOB	IO DIN DO L12N YY	OUTPUT	LVTTL

Checking the Timing

We have the displayable counter synthesized and implemented in the XC2S300E FPGA with the correct pin assignments. But how fast can we run the counter? To find out, double-click on the Generate Post-Place & Route Static Timing process. This will determine the maximum delays between logic elements in the design taking into account logic and wiring delays for the routed circuit.



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



From the information shown in the timing report, we see the minimum clock period for this design is 5.492 ns which means the maximum clock frequency is 182.1 MHz. The default clock frequency on the XSB-300EBoard is 50 MHz which is well below the maximum allowable frequency for this design.

Clock to Setup on destination clock clk

Source Clock				Src:Fall Dest:Fall
clk	' +	' +	' +	5.492

Generating the Bitstream

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

In this example, rather than use the gxsload utility we will employ the downloading utilities built into WebPACK. The iMPACT programming tool downloads the bitstream through the JTAG interface of the FPGA so we need to adjust the way the bitstream is generated to account for this. Right click on the Generate Programming File process and select the Properties... entry from the pop-up menu.



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

General Options Configuration options	Startup options	Readback options	
Property Name		Value	
FPGA Start-Up Clock		JTAG Clock	-
Enable Internal Done Pipe		CCLK	
Done (Output Events)		User Clock	
Enable Outputs (Output Events)		JTAG Clock	
Release Set/Reset (Output Events)		Default (6) 🤸 🕅	
Release Write Enable (Output Events)		Default (6)	
Release DLL (Output Events)		Default (NoWait)	
Drive Done Pin High		Γ	

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

Xilinx - Project Navigator - C:\tmp\fpga_designs\de File Edit View Project Source Process Window		_□× »_₽×
	┗┓┓╠₀ १ 🕺 💧 🠇 🛍 💼 🗠 🗠 🙀	•
Sources in Project: design2 design2 disp_cnt.ucf counter (counter.vhd) disp_cnt.ucf counter (counter.vhd) disp_cnt.ucf disp_cnt.ucf disp_cnt.ucf disp_cnt.ucf disp_cnt.ucf disp_cnt.ucf disp_cnt.ucf disp_cnt.ucf disp_cnt.ucf Design1\leddcd.vhd) Processes for Current Source: Design Entry Utilities Design Entry Utilities D	73 P69 74 P70 75 P71 76 P72 77 P73 78 P74 79 P75 80 P76 81 P77 82 P78 83 P79 84 P80 85 P81 86 P82 87 P83 88 P84 89 P85 90 P86 91 P87 92 P88 93 P89 94 P90 95 P91 96 P92 97 P93 98 P94 99 P95 100 P96 101 P97 ** ************************************	IOB IOB IOB DLLIOB GCLKIOF GCLKIOF IOB IOB IOB IOB IOB IOB IOB IOB IOB IOB
Loading device for application Bi	cation Bitgen from file "disp_cnt.ncd". 2.37, device xc2s300e, package pq208, speed -6 tgen from file '2s300e.nph' in environment	•
Generate Bit file for module disp_cnt	Ln 98 Col 1 50 9	%
Within a few seconds, a vill appear next to the Generate Programming File process and a file detailing the bitstream generation process will be created. A bitstream file named disp_cnt.bit can now be found in the design2 folder.

Xilinx - Project Navigator - C:\tmp\fpga_designs\d File Edit View Project Source Process Window		_ I × > _ B ×
	E 🏹 🇞 💡 🛠 🗍 X 🖻 🖻 🗠 🗠 🙀 🦳	•
Sources in Project: design2 curces in Project: disp_cnt (disp_cnt.sch) disp_cnt.ucf counter (counter.vhd) leddcd (\design1\leddcd.vhd) Module View Snapshot View Library View Processes for Current Source: Processes for Current Source: Design Entry Utilities User Constraints Synthesize Design Entry Utilities User Constraints Synthesize Programming File Programming File Configure Device (iMPACT)	73 P69 74 P70 75 P71 76 P72 77 P73 78 P74 79 P75 80 P76 81 P77 82 P78 83 P79 84 P80 85 P81 86 P82 87 P83 88 P84 89 P85 90 P86 91 P87 92 P88 93 P89 94 P90 95 P91 96 P92 97 P93 98 P94 99 P95 101 P97	IOB IOB IOB IOB IOB DLLIOB GCLKIOF DLLIOB IOB IOB IOB IOB IOB IOB IOB IOB IOB
Process View	counter.vhd	
Bitstream generation is complete. Completed process "Generate Progr	amming File".	
Process "Generate Programming File" is up to date.	Ln 98 Col 1	

Downloading the Bitstream

Before downloading the disp_cnt.bit file, we must configure the interface CPLD on the XSB-300E



board so it will work with the iMPACT programming tool. Double click the ^{GXSLOAD} icon and then drag & drop the piijitag.svf file from the C:\XSTOOLS4\XSA folder into the **gxsload** window. (Yes, we do mean the XSA folder; the XSA and XSB-300E Boards use the same CPLD configuration to support iMPACT.) Then click on the Load button and the CPLD will be reprogrammed in less than a minute.

	🖍 gxsload 💶 🖂 🗶
File Edit View Favorites Tools > + + → + 1 + + + + + >	Board Type XSB-300E Load
Address 🗋 XSA 💽 🔗 Go	Port LPT1 Exit
🔊 dwnldpar.svf 🔊 ram50.bit	FPGA/CPLD RAM Flash/EEPROM
🔊 dwnldpar-2_1.svf 🛛 🔊 xsats100.bit	piiijtag.svf
🖻 fcnfg.svf 🛛 🗒 xsats100.exo	
🔊 fcnfg100.svf 🛛 🙍 xsats50.bit	
🖻 fintf.svf 🔲 xsats50.exo	
intf100.svf	High Address
🔊 piiijtag.svf	
🔊 ram100.bit	Low Address
	Upload Format HEX 🔽 🗀 HEX 💌 🗀
Type: 156 KB 📃 My Computer 🥢	

After the piiijtag.svf file is loaded into the XSB-300E Board, move the shunt on jumper JP1 from the **xs** to the **xi** position. The XSB-300E Board is now setup so the FPGA can be configured through its boundary-scan pins with the iMPACT programming tool. Note that this process only needs to be done once because the CPLD on the XSB-300E Board will retain its configuration even when power is removed from the board. (If we want to go back to using the gxsload programming utility, we must move the shunt on JP1 back to the **xs** position and download the dwnldpar.svf file into the CPLD.)

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



In the **Operation Mode Selection** window, select the Configure Devices option since we are going to configure the FPGA on the XSB-300E Board (i.e., download a bitstream file to it). Then click on the Next button.

Operation Mode Selection	×
What do you want to do first?	
Configure Devices	
C Prepare Configuration Files	
C Load Configuration File (.cdf, .pdr)	
< Black Next > Cancel Help	

Now the **Configure Devices** window appears. Previously, we programmed the CPLD on the XSB-300E Board so it would support programming of the FPGA through its boundary-scan pins. So select the Boundary-Scan Mode option and click on the Next button.

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

Boundary-scan mode allows the configuration of multiple FPGAs connected together in a chain. To do this, the iMPACT software needs to know the types of the FPGAs in the chain. We know there is just a single FPGA on the XSB-300E Board and we could easily describe this to iMPACT. But iMPACT can also probe the boundary-scan chain and automatically identify the types of the FPGAs. This is even easier, so we select the automatic identification option and click on the Finish button.

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

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



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

Assign New Co	nfiguration File					? ×
Look in: 🔂	design2	•	🗢 🖻			
Digitalprojnav						
🗀 xst						
🛋 disp_cnt.bi	1					
File name:	disp_ont.bit				Open	Ę
Files of type:	FPGA Bit Files(*.bit)		•		Cancel	
			Car	ncel All		

Now iMPACT may warn us that the bitstream file we selected is for an XC2S300E FPGA but it detected an XCV300E FPGA in the boundary-scan chain. The XC2S300E FPGA was developed as a low-cost variant of the XCV300E FPGA and they both share the same boundary-scan identification code. We know the XSB-300E Board has an XC2S300E FPGA that is compatible with the bitstream file, so we can click on the Yes button and move on.

Xilinx iMP	PACT 🔀
?	A BIT file describing an xc2s300e is about to be assigned to a device previously identified as an xcv300e. Are you sure you want to do this?
	Yes No

Now the main **iMPACT** window shows a boundary-scan chain consisting of a single XC2S300E FPGA.

Untitled [Configuration Mode] - iMPACT				
File Edit Mode Operations Output View He	lp			
🗋 🗅 🚅 🔚 👗 🛍 🛍 🛍 🛍	H 🛱 🖽 🛱 🛱	# 13 📑 隆 👘		
Boundary Scan Slave Serial Selective	IAP Desktop Cont	figuration		
Right click device to select operations				
TDI				
INFO:iMPACT:501 - '1': Added Device xc2s300e su	ccessfully.			<u> </u>
=>				1
न				
For Help, press F1	Configuration Mode	Boundary-Scan	Parallel/PC3	lpt1

Now right-click on the xc2s300e icon and select the Program... item on the pop-up menu.

untitled [Configuration Mode] - iMPACT					<u>_ </u>
File Edit Mode Operations Output View					
0 🛥 🖬 👗 🖻 🖻 😫 🚟 💥					
Boundary Scan Slave Serial Selec	ctMAP Desktop Co	nfiguration			
TDI xc2s300 disp_ort) TDO TDO Assign New Configur					
					-
Device #1 selected =>					Ţ
For Help, press F1	Configuration Mode	Boundary-Scan	Parallel/PC3	lpt1	

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

Program Options	<u>? ×</u>
Erase Before Programming Verify Read Protect Write Protect Virtex-II/Pro SSecure Mod Program Key	Functional Test On-The-Fly Program ROM Skip user array Load Fpga Parallel Mode Use D4 for CF
PROM/CoolRunner-II Usercod	
XPLA UES: Enter up to 13 ch	aracters
OK Cancel	Help

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

Operation Status
Executing command
Abort

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

untitled [Configuration Mode] - iMPACT				
File Edit Mode Operations Output View He	elp			
📙 🗅 🚅 🔚 🖌 🖻 🖻 😫 🗱 👪 🚼	88 🛱 🖽 👹	🛱 🗇 📑 🎀 👘		
Boundary Scan Slave Serial Select	AP Desktop Cor	figuration		
TDI x2e300e disp_cnt.bit TDO				
INFO:IMPACT:579 - '1': Completed downloading b INFO:IMPACT:580 - '1':Checking done pindone '1': Programmed successfully. PROGRESS_END - End Operation. Elapsed time = 14 sec.				1
=> (▼
For Help, press F1	Configuration Mode	Boundary-Scan	Parallel/PC3	lpt1

Testing the Circuit

Once the XC2S300E FPGA on the XSB-300E Board is programmed, the circuit will begin operating without any further action from us. The LED display should repeatedly count through the sequence 0, 1, 2, 3, 4, 5, 6, 1, 8, 9, 8, 8, C, D, E, F with a complete cycle taking 5.4 seconds.



Going Further...

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

- In the Project Navigator window, select Help⇒ISE Help Contents. You will be presented with a browser window containing topics that will let you learn more about the WebPACK software.
- Get Essential VHDL (ISBN:0-9669590-0-0) or The Designer's Guide to VHDL (ISBN:1-55860-270-4) to learn more about VHDL for logic design.
- Go to the Xilinx web site and read their application notes and data sheets.
- Read the *comp.arch.fpga* newsgroup for helpful questions and answers about programmable logic design.