OrCAD Interface/ Tutorial Guide

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

OrCAD Interface/ Tutorial Guide

X-BLOX Tutorial

Xilinx ABEL Tutorial

XACT-Performance and XDelay Tutorial

XEPLD Tutorial

Program Options

Error Messages

Warning Messages

OrCAD XEPLD Demonstration Procedure

Preface

About This Manual

This manual describes Xilinx's OrCAD interface programs, a set of tools used to translate your schematics from OrCAD into implemented design files and simulation files.

Before using this manual, you should be familiar with the operations that are common to all Xilinx software tools: how to bring up the system, select a tool for use, specify operations, and manage design data. These topics are covered in the *XACT Reference Guide* and the *XEPLD Reference Guide*. You should also be familiar with OrCAD and the OrCAD documentation.

Alternatively, if you are a new user, you might wish to turn to the tutorial chapters at the end of this manual for a set of tutorials that introduce the new user to the process of creating a Xilinx design with OrCAD. Other publications you can consult for related information are the *XACT Libraries Guide* and *The Programmable Logic Data Book*.

Manual Contents

This manual covers the following topics.

- Chapter 1, INTRODUCTION, introduces the Xilinx programs and the general task flow of the book.
- Chapter 2, GETTING STARTED, assists you in preparing your system to use the OrCAD interface software.
- Chapter 3, ORCAD SDT DESIGN TECHNIQUES, discusses schematic entry techniques common to both FPGA and EPLD designs.

- Chapter 4, FPGA DESIGN ISSUES, assists you in creating your FPGA design in view of the conversion to implemented design and simulation files.
- Chapter 5, EPLD DESIGN ISSUES, assists you in creating your EPLD design.
- Chapter 6, FUNCTIONAL SIMULATION, provides translation procedures for creating a functional netlist file for simulation as well as instructions on how to simulate your design.
- Chapter 7, DESIGN IMPLEMENTATION, discusses the translation of a schematic design into an implemented design file.
- Chapter 8, TIMING SIMULATION, provides translation procedures for creating a timing netlist as well as instructions on how to simulate your design.
- Chapter 9, ORCAD VST SIMULATION ISSUES, explains the simulation issues you should understand to simulate your design.
- Chapter 10, MANUAL TRANSLATION, explains how to manually translate your design into implemented design files and simulation netlists, as an alternative to the automated procedures.
- Chapter 11, SDT TUTORIAL, teaches the first-time user how to use OrCAD to enter an FPGA schematic design using Xilinx libraries.
- Chapter 12, VST TUTORIAL, teaches the first-time user how to prepare for simulation and do functional and timing simulations.
- Chapter 13, XILINX ABEL TUTORIAL, demonstrates the use of OrCAD with Xilinx ABEL.
- Chapter 14, X-BLOX TUTORIAL, demonstrates the use of OrCAD with X-BLOX^{TM} for FPGA designs.
- Chapter 15, XACT-PERFORMANCE AND XDELAY TUTORIAL, demonstrates the use of OrCAD with XACT-Performance^{TM} and XDelay for FPGA designs.
- Chapter 16, XEPLD^{TM} TUTORIAL, teaches the first-time user how to complete an XEPLD schematic design and perform simulations on the design.
- Appendix A, PROGRAM OPTIONS, lists the options used with the interface programs XDraft, SDT2XNF, and XNF2VST.

- Appendix B, ERROR MESSAGES, lists all error messages and suggested recovery techniques for SDT2XNF and XNF2VST.
- Appendix C, WARNING MESSAGES, lists all warning messages and suggested recovery techniques for XDraft, SDT2XNF, and XNF2VST.
- Appendix D, ORCAD XEPLD DEMONSTRATION PROCEDURE, is a summary of the commands used in the XEPLD tutorial.

Conventions

The following conventions are used in this manual's syntactical statements:

`Courier font regular`	System messages or program files appear in regular Courier font.
`Courier font bold`	Literal commands that you must enter in syntax statements are in bold Courier font.
italic font	Variables that you replace in syntax statements are in italic font.
`[ ]`	Square brackets denote optional items or parameters. However, in bus specifications, such as bus [7:0], they are required.
`{ }`	Braces enclose a list of items from which you must choose one or more.
. . .	A vertical ellipsis indicates material that has been omitted.
. . .	A horizontal ellipsis indicates that the preceding can be repeated one or more times.
|	A vertical bar separates items in a list of choices.
↵	This symbol denotes a carriage return.

Contents

Chapter 3 OrCAD SDT Design Techniques

Chapter 4 FPGA Design Issues

Chapter 10 Manual Translation

Chapter 11 SDT Tutorial

Chapter 12 VST Tutorial

Chapter 14 Xilinx ABEL Tutorial

Chapter 15 XACT-Performance and XDelay Tutorial

Chapter 16 XEPLD Tutorial

Appendix C Warning Messages

Appendix D OrCAD XEPLD Demonstration Procedure

OrCAD Interface/ Tutorial Guide

Introduction

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Chapter 1

Introduction

This book explains how to use Release 5.0 of the XACT OrCAD interface software to translate your FPGA and EPLD designs from OrCAD schematics into implemented design files and simulation files.

In order to make proper use of the information in this book, you should be familiar with the overall design task flow and the concept of interface software. Both concepts are defined in the following sections.

Defining the Design Flow

The process of designing FPGAs and EPLDs can be summarized as follows.

1. Enter your design with the OrCAD schematic editor making sure that you observe the Xilinx design requirements mentioned in this book.
2. Test the functionality of your design. Generate in one step a functional netlist file using the XSimMake program with the appropriate options and use the file for functional simulation.

   When the functional verification test proves that the functions of your design have been properly represented, proceed with the third step, design implementation.
3. Implement your FPGA or EPLD design. Generate the implemented design, an LCA™ or VMH file, automatically by executing the XMake program for an FPGA design or the XEMake program for an EPLD design.
4. Test the timing of your design. Generate a timing netlist in one step using the XSimMake program on your LCA file or VMH file

with the appropriate options. Use the VST output file for your timing simulation.

If you developed your EPLD design behaviorally instead of using a schematic and you wish to simulate using OrCAD / VST386+, refer to Chapter 8, "Timing Simulation."

5. Download your design to an FPGA or EPLD device.

The following figure proposes an overall view of the design process:

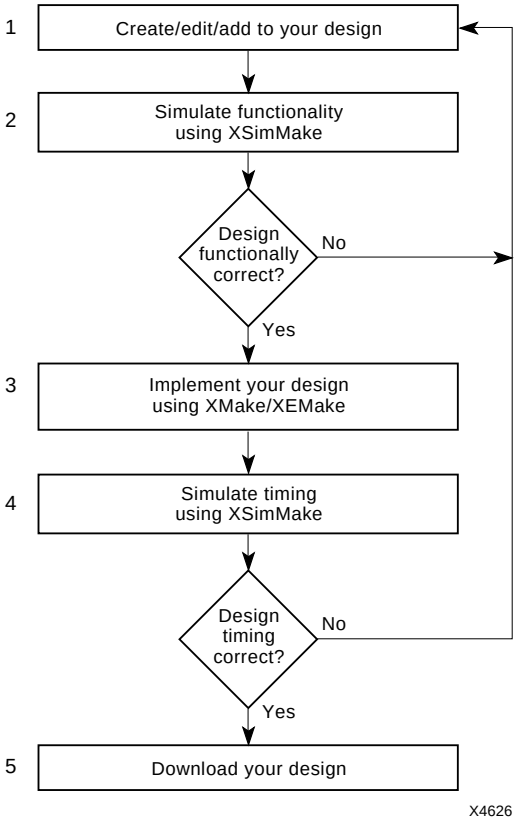


Figure 1-1 Design Process: Design Entry, Functional Simulation, Design Implementation, and Timing Simulation

Defining the Interface

Central to the interface is the concept of translation. The translation between OrCAD and Xilinx format is enabled by two Xilinx programs: SDT2XNF and XNF2VST.

The first interface program, SDT2XNF, converts your source schematic design from the OrCAD SDT format into a format that makes your design compatible with the Xilinx core tools. This format is called Xilinx Netlist Format (XNF).

The other interface program, XNF2VST, converts either the original or the processed design from the XNF format into an OrCAD simulation file (VST).

Whereas SDT2XNF and XNF2VST are the engines of the interface, there are other Xilinx programs that must be invoked to generate implemented design and simulation files. XMake / XEMake and XSimMake are the programs that automatically invoke the appropriate sequence of programs to create the implemented design and simulation files.

Also part of the interface are the Xilinx-supplied FPGA and EPLD libraries, which you must use to draw your schematics. These libraries determine the logic of the FPGAs and EPLDs you design.

What is New in this Release

This section summarizes the new features and enhancements of the OrCAD interface software Version 5.0.

Library Features

- The main new feature of this release is a new set of library symbols called Unified Libraries. The advantage of these new libraries is that they use the same component names, with the same footprints (shapes, pin names, and functionality) between product families, thus enabling you to convert your designs easily from one product family to another, EPLDs included.
- The names of the individual Unified Libraries are the XC2000, XC3000, XC4000, and XC7000 libraries. These libraries are installed under the XACT directory. You can still use the existing sdt2k, sdt3k, sdt4k, and sdt7k libraries for your existing designs,

but we recommend that you use the new libraries to create all new designs. When you start a new design, specify either the new or the old libraries, as it is not allowed to mix old and new library elements or components from different libraries. Each design you create must be composed of either old-library elements or new unified-library elements.

- The Unified SDT / VST Libraries are in OrCAD SDT / VST 386+ 32-bit format only.
- Relationally placed macros (RPMs) replace all Xilinx-supplied hard macros. RPMs are easier to use than hard macros; moreover, they are simulatable. If you need to use your user-created hard macros, you must convert them to RPMs using the HM2RPM utility. Refer to the *XACT Reference Guide* for more information on HM2RPM.
- EPLD designs use XNF-formatted netlists (not EDIF) for design capture. Functional simulation is now supported for EPLD designs.

XDraft Support

- XDraft now supports configuration for both SDT and VST. If you need to configure the SDT and VST configuration files separately, use the -s option for SDT and the -v option for VST. You can also configure SDT and VST together using the XDraft command with no options.
- In addition to supporting the new Unified SDT / VST Libraries, XDraft supports the old v4.2x SDT / VST libraries. Use option -L to configure the old libraries assuming they are under XACT.
- XDraft supports Xilinx EPLD configuration. Specify a 7 after the XDraft command to specify the XC7000 library.
- You can override the default XACT environment variable by using the -x command-line option.

SDT2XNF Enhancements

- You can run SDT2XNF on designs created with either the new Unified SDT Libraries or the old v4.2x SDT libraries. However, do not attempt to mix old and new library elements in your schematics.
- SDT2XNF now offers two ways of handling multiple-sheet designs:
  - Flat designs (multiple sheets at the top level)
  - Hierarchical designs (multiple sheets at different levels)
- SDT2XNF now generates multiple XNF files for Xilinx macros, user-defined hierarchical sheet symbols, and user-defined macros (schematics containing Xilinx primitives). XNFMerge merges all the XNF files into one flat XNF format file (XFF file).
- You can specify the search path for the INF files of user-defined sheet symbols and macros by using the new -u command-line option. The default value for this path is the current directory. (INF is the OrCAD netlist format produced by the OrCAD INET program.)
- You can specify the search path for the INF files of Xilinx library macros by using the new -s command-line option. The default value for this path depends on the XACT environment variable and the Xilinx part type used in the design.
- You can specify stimulus and trace information at any level in your schematic and SDT2XNF passes it to the XNF files.

XNF2VST Enhancements

- You can run XNF2VST to simulate designs created with elements from the new SDT/VST Unified Libraries or from the old v4.2x SDT/VST library. However, do not attempt to mix old and new library elements in your schematics.
- If you specify stimulus and trace information at any level in your schematic, XNF2VST generates AST (OrCAD ASCII Stimulus) and ATR (OrCAD ASCII Trace) files automatically.
- Because XNF2VST supports recycled aliasing in the Name Reference File (NRF) file, you can modify the reference names in

the NRF file by entering a symbol and signal name of your choice. XNF2VST uses the modified names during the translation process.

OrCAD Interface/ Tutorial Guide

Getting Started

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Chapter 2

Getting Started

This chapter describes software configuration and the initial steps you must perform before using the OrCAD Schematic Design Tools and the Xilinx interface software to design FPGA and EPLD devices.

The chapter is structured as follows:

- *Preparing Your System* tells you all the steps you must complete for your installation.
- *Partitioning Software Between Two Different Disks* explains how to install your Xilinx software on one drive and your OrCAD software on another drive.
- *Entering the OrCAD/ESP Design Environment* tells you the different ways of starting OrCAD.
- *Configuring the OrCAD/ESP Design Environment* directs you to the information source you should consult to configure the ESP environment.
- *Creating a Design Directory* explains how to create a design directory before designing a specific FPGA or EPLD device.
- *Using XDraft to Configure the OrCAD Environment* describes how to configure the schematic editor and the simulation tools.
- *Entering the Draft Schematic Editor* shows you how to enter the schematic editor.
- *Retargeting Your Design to a Different Family* discusses the conversion of designs created with the Unified Libraries symbols.

If there is any question regarding the proper use of OrCAD / SDT386+ when creating a design file, the information contained in this manual supersedes that contained in your OrCAD-supplied documentation.

Note: The Unified Libraries provided by Xilinx cannot be used with the OrCAD III or OrCAD IV software. You can only use them with OrCAD/SDT386+. However, you can still use the OrCAD IV Xilinx libraries with the new translators. As the old libraries are no longer provided, you must copy them into the XACT directory where the new OrCAD interface software is installed.

Preparing Your System

You must complete four basic steps before you can begin working with your OrCAD software. The following is a summary of how to prepare your system to use the OrCAD interface software. This chapter documents only system configuration, which is defined below in step 4. For instructions on software installation, refer to the *XACT Installation Guide.*

1. Install the OrCAD-supplied software:
   - OrCAD/ESP
   - OrCAD/SDT386+
   - OrCAD/VST386+ (optional)
2. Install the OrCAD interface software supplied by Xilinx into your XACT directory. The following shows you the structure of your typical XACT directories and files. Refer to the *XACT Installation Guide* for information on how to complete the installation.

\XACT
inf2xnf.exe
macro3.mac
sdt2xnf.exe
vstmac.mac
xcfg.exe
xdraft.exe
xnf2inf.exe
xnf2vst.exe
\data
\msg
\xc2000
\xc3000
\xc4000
\xc7000

3. Install the Xilinx XACT core tools as explained in the documents provided with the software.
4. Verify that your system configuration files include the required OrCAD environment variables and path names.
   - Enter the OrCAD/ESP design environment.
   - Configure ESP, referring to the OrCAD documentation for instructions.
   - Create a design directory for your new design. This is a requirement of the OrCAD/ESP software.
   - Configure SDT and VST automatically by running the XDraft program as explained in this chapter or manually by following the instructions in your OrCAD documentation.

Partitioning Software Between Two Different Disks

If you do not have a disk partition large enough to hold the Xilinx Development System software, the Xilinx OrCAD interface and libraries, and the OrCAD software, you can partition the software between two different drives.

The simplest approach is to load the Xilinx Development System software and the Xilinx OrCAD interface and libraries on the same drive, and the OrCAD software on another drive. In this case, system setup is straightforward.

Alternatively, you can load the Xilinx Development System software on one drive and the Xilinx OrCAD interface and libraries with the OrCAD software on another drive. For example, if you install the Xilinx core software on drive C: and the OrCAD interface and libraries with the OrCAD software on drive D:, your autoexec.bat file resembles the following:

```
path=...;d:\xact;c:\xact;d:\orcadexe
set xact=d:\xact;c:\xact
set ORCADEXE=d:\orcadexe\
set ORCADESP=d:\orcadesp\
set ORCADPROJ=d:\orcad\
set ORCADUSER=d:\orcadesp\
```

The config.sys file is unchanged.

Configure the OrCAD design directory using XDraft as explained in the section “Using XDraft to Configure the OrCAD Environment.” You can make corrections to the file using any text editor. For this example, the sdt.cfg file must contain the following lines:

```
{ OrCAD/SDT386+ Configuration File }
PDRV   = 'd:\ORCADESP\DRV\'
PSCH   = ''
PLIB   = 'd:\xact\xc3000\*.LIB'
DD     = 'VGA640.DRV'
PRD    = ''
PLD    = ''
LIB    = 'XC3000.lib'
DMF    = 'd:\xact\macro3.mac'
DIM    = '\I'
.
.
.
```

Entering the OrCAD/ESP Design Environment

This section assumes that you have completed your software installation. To configure your software, you need to enter the OrCAD/ESP environment. You can enter OrCAD/ESP either from a DOS prompt or from the XACT Design Manager (XDM™).

From a DOS Command Prompt

Provided that your active path includes the OrCAD executable directory, you can enter the OrCAD/ESP Design Environment by entering the following command from a DOS command prompt:

```
orcad↵
```

You can enter the OrCAD/ESP Design Environment directly from DOS and execute all the OrCAD-supplied programs.

You can execute all the Xilinx-supplied programs from the XACT Design Manager or from the DOS command line. It is a good idea to run your programs from DOS if you run into memory problems.

To get information on the Xilinx program options from DOS, type the program name at the DOS prompt and press the return key. Appendix A supplies the syntax as well as a list of program options for XDraft, SDT2XNF, and XNF2VST.

From the XACT Design Manager

There are two ways to invoke the OrCAD software from within XDM. One way is to select it from the Design Entry menu. The other way is to type it at the command line, which is located at the bottom of the XDM screen. Help is available by pressing F1 from any XDM menu option.

As an XDM Menu Selection

To enter the OrCAD/ESP Design Environment as an XDM menu selection, complete these steps:

1. Open the **`Design Entry`** menu.
2. Select **`OrCAD`** from the menu.

Note: The menu items listed in your Design Entry menu are based on

the Xilinx-supported products installed on your system. If OrCAD does not appear in your Design Entry menu, it might be necessary to perform a ScanDisk by selecting ScanDisk from the Utilities menu.

At the XDM Command Line

To enter the OrCAD/ESP Design Environment at the XDM command line, enter the following at the XDM command line:

```
orcad↵
```

Configuring the OrCAD/ESP Design Environment

After invoking OrCAD from either DOS or XDM, the OrCAD/ESP Design Environment Main Screen appears. You should refer to the OrCAD Schematic Design Tools documentation to familiarize yourself with your OrCAD software and to customize the configuration of the OrCAD/ESP Design Environment.

Refer to your OrCAD documentation for an explanation on how to configure the options for the driver, editor, design, and printer/plotter output, as well as the color and pen plotter table.

Creating a Design Directory

Before entering your design, you need to create a design directory, also called a working directory. You need to configure the directory to use the device family you want. You can create a design directory from DOS or from the OrCAD graphical user interface.

From DOS

To create a design directory from DOS, follow the steps outlined below:

1. Change to your ORCADPROJ directory, which is \orcad by default. Type:

   ```
   cd \orcad↵
   ```

2. Use the mkdir command followed by the directory name:

   mkdir *directoryname* ↵

3. Copy the OrCAD template files into your new design directory:

   `copy \orcad\template\*.*` ***directoryname*** ↵

From the Graphical User Interface

To create a design directory from the graphical user interface, follow the steps below:

1. Invoke OrCAD.
2. Click on the **`Design Management Tools`** button. A menu appears in the upper-left corner of your display.
3. Select **`Execute`** from the menu to display the screen for the Design Management Tools.
4. With the Design View tools displayed, click the left mouse button once on the **`Create Design`** selection to bring up another window, and click again in the box to the right of "New Design Name."
5. Type the name of your design and press **`Enter`**. Then, click on **`OK`** to confirm.

   ESP creates the design directory under the path specified by the ORCADPROJ environment variable. The default setting for ORCADPROJ is *drive*:\orcad.

   For example, if ORCADPROJ=c:\orcad\ and a new design, FIFO, is specified, the directory c:\orcad\fifo is created. ESP also copies all the template configuration files into the new design directory.
6. Click on the **`OK`** button to return to the ESP main menu.

Using XDraft to Configure the OrCAD Environment

You can only use one product family per logic design. To use OrCAD/SDT386+ with the specific Xilinx product family required by your logic design, you need to reconfigure your OrCAD Schematic Tools every time you create a new design.

After creating a new design directory, modify the OrCAD/SDT386+ configuration file (sdt.cfg) to support logic development for the device family targeted for your design. You can also modify the OrCAD/VST386+ configuration file (vst.cfg) now, if you wish to

prepare for later simulation. You can perform both configurations automatically and in a single step using the XDraft utility, or you can perform them without XDraft as described in the OrCAD documentation.

You can use the XDraft option under the XDM Translate menu or enter the XDraft syntax from DOS as explained in the next section.

Note: Automatic configuration with XDraft enables you to configure the schematic tools and the simulation tools separately or together.

Running XDraft

To modify your configuration files automatically from the ESP Design Environment, click on the top of the screen and select **Suspend to System**. This generates a DOS shell with the current directory corresponding to the current design.

Note: To run XDraft, you must have sdt.cfg and vst.cfg in your local directory. If they are not in this directory, copy them from the OrCAD Template directory into your current design directory. Otherwise, XDraft issues a warning that it cannot configure, because it cannot find the configuration files. By default, XDraft configures the environment for both schematic entry and simulation. There are a number of options that you can use to limit your configuration.

Use the following DOS prompt syntax to configure the Draft schematic editor and the simulation tools:

xdraft ***number*** **[*options*]**↵

where *number* is 2, 3, 4, or 7.

Number stands for the device family you wish to use for your design. For example, entering a "2" configures the schematic editor (SDT) and the simulation tools (VST) for use with XC2000 design files. You do not need to use the XDraft utility again unless you need to change the configuration for a different device family. In most cases, XDraft is needed only when creating a new design.

The following configuration options are available:

- To configure SDT only, add the -s command-line option to the preceding syntax:

  xdraft ***number*** **-s**↵

where *number* is 2, 3, 4, or 7.

- To configure VST only, add the -v command-line option to the preceding syntax:

  xdraft *number* -v ↵

where *number* is 2, 3, 4, or 7.

Warning: To configure SDT and VST to use the old v4.2 libraries, specify the -L command-line option after the XDraft command:

xdraft *number* -l ↵

where *number* is 2, 3, 4, or 7.

If you ran the XDraft utility from DOS within the Draft schematic editor, you must exit from Draft and invoke it again to reconfigure the Draft menus.

Warning: XDraft does not update the Connectivity Database extension for your VST configuration. Refer to the section "Connectivity Database Extension" later in this chapter for information on how to make this change.

Changes Made to SDT

XDraft makes the following modifications to the SDT configuration in the sdt.cfg file for the current design:

- *Library Options* — The library path is set for the XC2000, XC3000, XC4000, or XC7000 library depending on which family you select, as in the following example:

  ```
  PLIB = 'c:\XACT\XC4000\'
  ```

  This example displays the library path for the XC4000 family.

- *Macro Options* — In the sdt.cfg file, the macro3.mac file is specified as the default macro library and the \I macro (Alt - I) is set as the initial macro. The \I macro is a user-defined macro.

  ```
  DMF = 'c:\XACT\MACRO3.MAC'
  DIM = '\I'
  ```

- *User Part Fields* — The user part fields are set according to the family specified by the XDraft option used. Tables 2-1 and 2-2 indicate the field names for each FPGA device family. The part

fields for the XC7000 EPLD family are simply called Part Fields 1 through 8.

Table 2-1 XC2000 and XC3000 Field Names

Part Value Field	Field Name
1st Part Field	LOC, OPTIONS
2nd Part Field	BLKNM
3rd Part Field	BASE
4th Part Field	CONFIG
5th Part Field	EQUATE_F
6th Part Field	$FCONT
7th Part Field	EQUATE_G
8th Part Field	$GCONT

Table 2-2 XC4000 Field Names

Part Value Field	Field Name
1st Part Field	OPTIONS_1
2nd Part Field	OPTIONS_2
3rd Part Field	INIT
4th Part Field	BASE
5th Part Field	CONFIG
6th Part Field	EQUATE_F
7th Part Field	EQUATE_G
8th Part Field	EQUATE_H

Sample Sdt.cfg File

Ensure your configuration file sdt.cfg includes the following lines:

```
{ OrCAD/SDT386+ Configuration File }
PDRV    = 'c:\ORCADESP\DRV\'
```

```
PSCH    = ''
PLIB    = 'c:\XACT\XC4000\'
DD      = 'VGA640.DRV'
LIB     = 'XC4000.lib'
LIB     = 'XBLOX.lib'
DMF     = 'c:\XACT\MACRO3.MAC'
DIM     = '\I'
```

Note: This example is for a configuration file that uses the XC4000 libraries. For an XC2000, XC3000, or XC7000 design, replace the XC4000 library names with those of the libraries that match your design.

Changes Made to VST

XDraft makes the following modifications to the VST configuration in the vst.cfg file for the current design:

- *Library Options* — The library path is set for the XC2000, XC3000, XC4000, or XC7000 library depending on which family you select, as follows:

  ```
  PLIB = 'c:\XACT\XC4000\'
  ```

  In this instance, we show the library path for the XC4000 family.

- *Macro Options* — The vst.cfg file sets the vstmac.mac file as the default macro library.

  ```
  MAC = 'c:\XACT\VSTMAC.MAC'
  ```

- *Simulator Options Prefix* — XDraft updates the simulator options prefix with a "G".

  ```
  PREFIX = 'G'
  ```

Warning: Without the "G" prefix, your outputs remain undefined.

Connectivity Database Extension

XDraft does not update the Connectivity Database extension for your VST configuration. You need to manually update this extension from INF to VST so that it specifies the VST file created by the XNF2VST program.

1. Enter the OrCAD ESP design environment from DOS or from XDM.

2. On the OrCAD/ESP main screen, select **Design Management Tools**. Check that your design file is selected on the Design Management Tools screen. Click **OK** to save the settings and exit.
3. Select **Digital Simulation Tools** on the OrCAD/ESP Main Screen.
4. Select the menu command **Simulate → Local Configuration → Configure Simulate**.
5. The Configure Simulate screen appears. The first field on the Configure Simulate screen is the File Options field. Click on the box next to Connectivity Database, and change the INF extension of your schematic file to VST.
6. Click on **OK** to save the changes.

Sample Vst.cfg File

Ensure your configuration file, vst.cfg, includes the following lines:

```
{ OrCAD/VST386+ Configuration File }

PDRV    = 'c:\ORCADESP\DRV\'
DD      = 'VGA640.DRV'
PLIB    = 'c:\XACT\XC4000\'
PREFIX  = 'G'
MAC     = 'c:\XACT\VSTMAC.MAC'
IMAC    = '\F6'
```

This example is for a configuration file that uses the XC4000 libraries. For an XC2000, XC3000, or XC7000 design, replace the XC4000 library name with that of the library that matches your design.

Macro Files

OrCAD/SDT386+ comes with two Draft macro files, macro1.mac and macro2.mac. Your Xilinx translation software includes a third macro file, macro3.mac, which supports OrCAD/SDT386+ for the Xilinx design process, and a file called vstmac.mac, which supports OrCAD/VST386+ for simulation.

Table 2-3 lists all macro keys and functions available when using the macro3.mac file. Table 2-4 lists the macros and functions available when using the vstmac.mac file. These files are copied into the XACT directory during software installation.

Note: Before executing a key macro, verify that you are at the main menu level and that you did not select any menus or commands. If you are not at the main menu level, the process executed by a macro is unpredictable.

Some key macros require that the cursor point at an intended object before execution. To cancel a macro requiring user interaction while it is executing, press the Escape and Enter keys, as necessary, until you are once again at the root-menu level.

Table 2-3 Macros in the Macro3.mac File

Key	Function	Key	Function
F1	Get Component	Shift-F1	Abandon and Quit
F2	Place Wire	Shift-F2	Clear the Entire Sheet
F3	Copy Object at Cursor	Shift-F3	Import Block
F4	Delete Object at Cursor	Shift-F4	Export Block
F5	Drag Object at Cursor	Shift-F5	Drag Block
F6	Place Bus	Shift-F6	Save and Continue
F7	Place Label	Shift-F7	Save File as...
F8	Place Module Port	Shift-F8	Print Schematic
F9	Place Sheet Symbol	Shift-F9	DOS Shell
F10	Redraw Screen	Shift-F10	Save and Exit
Ctrl-F1	List Library Directory	Alt-F1	Set Auto Pan Off
Ctrl-F2	Place Junction	Alt-F2	Set Auto Pan On
Ctrl-F3	Copy Block	Alt-F3	Zoom to Window
Ctrl-F4	Delete Block	Alt-F4	Undo Delete
Ctrl-F5	Move Block	Alt-F5	Zoom In
Ctrl-F6	Edit Net or Module Port	Alt-F6	Zoom Out
Ctrl-F7	Edit Title Block	Alt-F7	Save and Enter Sheet
Ctrl-F8	Set Work Sheet Size	Alt-F8	Save and Leave Sheet
Ctrl-F9	Place Text	Alt-F9	Turn Grid Off
Ctrl-F10	Show Condition	Alt-F10	Turn Grid On
Left Mouse	Enter (select)	Alt-E	Edit Label or Module Port
Middle Mouse	Place Wire	PgUp	Zoom Out
Right Mouse	Escape	PgDn	Zoom In

Table 2-4 Macros in the Vstmac.mac File

Key	Function
F1	Run simulation for 100 units (10 ns)
F2	Run simulation for 1,000 units (100 ns)
F3	Run simulation for 10,000 units (1,000 ns)
F4	Toggle on/off Setup, Hold, and Pulse Checking
F5	Initialize simulator, spool overflow simulation data to file *spool.tdf*
F6	Zoom to time 0
F7	Change trace view to 1 (.1 ns/grid)
F8	Change trace view to 10 (1 ns/grid)
F9	Change trace view to 1,000 (100 ns/grid)

Entering the Draft Schematic Editor

Verify your configuration by invoking the Schematic Editor. To invoke the Draft schematic editor, first ensure the current design displayed on the ESP main screen is the one you want to use. Then, press the **`Schematic Design Tools`** button on the ESP main screen and select **`Execute`** from the menu. Finally, press the **`Draft`** button in the upper-left corner of the screen, and select **`Execute`** from the menu. If you can access the SDT Draft menu, your configuration was successful.

To enter the Draft schematic editor from DOS, use the following syntax:

`draft` ***designname***↵

Retargeting Your Design to a Different Family

The Unified Libraries allow you to retarget your designs from one family to another as long as both your source and target designs only include symbols from the Unified Libraries. Most of the symbols of the Unified Libraries have the same footprint and name from one device family to another, hence their ability to be easily converted.

To retarget or convert a design, follow these simple steps:

1. Change to your ORCADPROJ directory, which is \orcad by default. Type:

   cd \orcad↵

2. Create a new design directory for your target design.

   mkdir *newdesign*↵

3. Copy the template files and the design you wish to retarget from the old directory to your new design directory:

   copy *olddesign newdesign*↵

4. Change directories to the new design directory:

   cd *newdesign*↵

5. Run XDraft on the new design directory, specifying the target family number 2, 3, 4, or 7. For example, to retarget an XC3000 family design to the XC4000 family, execute XDraft and specify 4 to designate the XC4000 target family.

   xdraft *4*↵

6. Run the OrCAD Draft schematic editor and check each schematic, making sure all the symbols have been converted to the target family.

   draft *filename.sch*↵

   In your target design, the symbols that are common to the source and target families maintain their relative location and pin position in the schematic. Pins on these symbols retain their connectivity to the nets they were attached to in the source design.

   Symbols that are not common to your source and target families appear as empty spaces at the places where they used to be displayed on the source schematic. Draft also issues a warning that it could not find these components.

7. Replace with equivalent logic the symbols that have not been converted to symbols in your target design.

Example

To translate an XC3000 family ACLK (Alternate Clock Buffer), which is not available in the XC4000 family, to an XC4000 symbol, replace it with a BUFGS (Secondary Global Buffer). Because both the ACLK and BUFGS modules are secondary global buffers, you can replace one with the other. Place this macro in the blank space that was occupied by the ACLK macro in the source design. Connect the pins to the appropriate nets to emulate the original symbol. For some symbols, you might have to change the length of the nets.

Repeat these steps for all underlying schematics in your design until you have filled in all the missing parts of your design.

OrCAD Interface/ Tutorial Guide

OrCAD SDT Design Techniques

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Chapter 3

OrCAD SDT Design Techniques

This chapter discusses various schematic drawing techniques and requirements common to both FPGA and EPLD designs.

- *Naming Conventions* lists the naming conventions you must follow for nets and symbols in your schematics.
- *Multiple-Sheet Designs* explains how to create flat designs and hierarchical designs.
- *User-Created Libraries* explains how to create library elements and add them to a user library.

Naming Conventions

This section lists the valid characters and discusses the purposes and conventions for naming nets and symbols used in an FPGA or EPLD logic design.

Reserved Names

For FPGA designs, you cannot use the physical names associated with every resource on every FPGA part to name signals and symbols. Reserved names include CLBs, IOBs, clock buffers, TBUFs, oscillators, package pin names, “CCLK,” “DP,” “GND,” “VCC,” “M0RT,” “PWRDN,” and “RST.” Examples of reserved names include CLB names such as AA and AB, pin names such as P1 and P2, pad names such as PAD1 and PAD2, and primitive names such as TDO, BSCAN, M0, M1, M2, or STARTUP.

For EPLD designs, you cannot use “PRLD,” “MRESET,” or any name beginning with “_N” (underscore N) as a net label.

Valid Characters

All names given to symbols and signals in an FPGA or an EPLD design must be valid for the XACT Development System. Names for nets, buses, and symbols must follow these conventions:

- Only A-Z, a-z, 0-9, "_," and "-" are allowed in user-defined names.
- Two other characters — the square open and square closed brackets ([and]) — can only be used to represent a bus in the OrCAD / SDT386+ software.
- Names, which are not case-sensitive, must contain at least one non-numeric character, and can begin with any legal character.
- Names can be up to 1024 characters in length, including the hierarchical paths.

Naming Nets and Subnets

Putting net names on your signals is always encouraged for easy tracking and understanding of the design. Labeling important signals provides future reference for the simulator. If a net has no label, the interface assigns a default name, for example, $U10_O or $U25_Q. The dollar sign ($) is part of the default name. The reference designator and the pin name of a library part or the sheet name and pin name of a sheet symbol determine the default net name.

If a net name appears before partitioning but disappears after partitioning, the signal might have been pulled into the functional block of the CLB, merged into another net through a minimizing process, or removed because it was either loadless or sourceless.

Knowing how to reference subnets deep within a hierarchical design is very important for simulation and debugging. During the flattening process of a hierarchical design, the translator creates and prefixes each of the subnets with a hierarchical path. The path is always the sheet name of a sheet symbol or the reference designator of a library part.

In Figure 3-1, for example, reference a subnet called MYNET in the BOTTOM module in the following manner:

```
middle/bottom/mynet
```

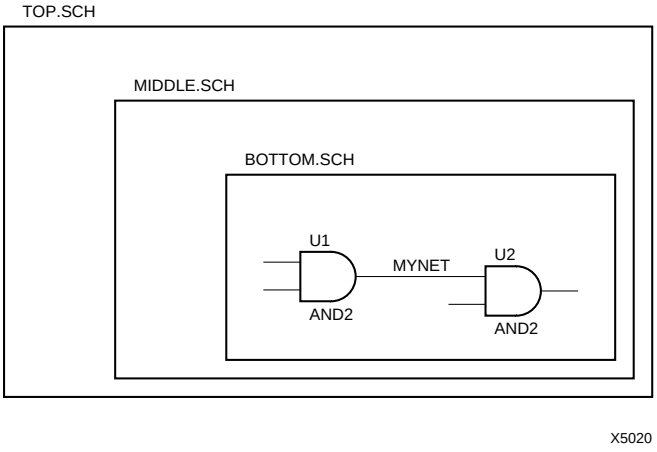


Figure 3-1 Referencing Subnets in a Hierarchical Design

If a net connects hierarchically, it retains the net name at the parent level, regardless of any lower-level names. For example, if a signal from the root called TOPCLK is connected to a signal called MIDCLK of the middle level and to a signal called BOTCLK of the bottom level, it is simply referenced as TOPCLK.

Multiple-Sheet Designs

The OrCAD interface supports two ways of handling multiple-sheet designs:

- Flat designs (multiple sheets at the top level)
- Hierarchical designs (multiple sheets at different levels)

Flat Designs

Flat design refers to the method of extending a design on multiple sheets at the root level. This is achieved by means of module ports and the |LINK keyword.

To link the worksheets together, first place input and output module ports on the nets of your designs so that the output signals of one sheet can be connected to the input signals of another sheet laterally.

To attach module ports to your nets, do as follows:

1. Invoke the **Place → Module Port** command.
2. Specify the name of the port at the `Module Port Name?` prompt.
3. Press **Enter**.
4. Specify the type of port: **Input**, **Output**, or **Bidirectional**.
5. Place the module port on the appropriate net using the **Place** command.
6. When you have placed all the module ports, you can connect the worksheets together.

Use the |LINK keyword and pipe character on the root level schematic to tell the schematic tools which worksheets are included in the flat design. In the following example, Figure 3-2, the TOP schematic is first entered as a single design. Then, in an equivalent implementation, the same circuit is entered as two separate sheets. The |LINK keyword is used to link the root worksheet (TOP.SCH) to SHEET1.SCH. (See Figure 3-3 and Figure 3-4.)

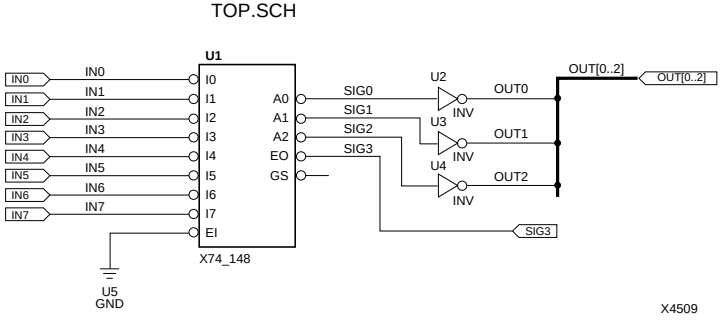


Figure 3-2 Top Schematic Design: Single Sheet Design

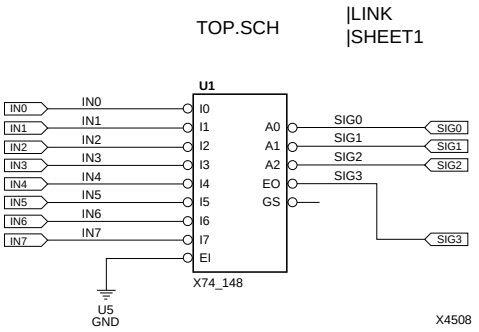


Figure 3-3 Linking Multiple Sheets Together: Primary Design

SHEET1.SCH

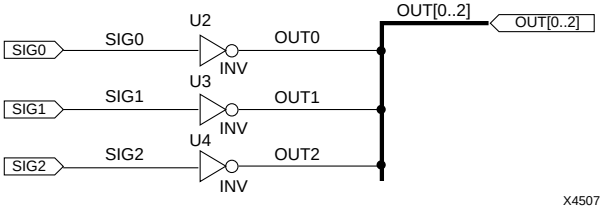


Figure 3-4 Linking Multiple Sheets Together: Second Design

To place the |LINK keyword for the design names to be included on your worksheet, do as follows:

1. From the top root-level design, select the **Place → Text** command.
2. At the `Text?` prompt, type **|LINK**.
3. Press **Enter** and place the keyword on the worksheet using the **Place** command.

4. Use the **Place → Text** command and type the name of the sheet you wish to connect to your primary design at the Text? prompt. Precede the design name with the pipe character, as follows.

   |*sheetname*

5. Press **Enter** and use the **Place** command to place the design name exactly under the |LINK keyword. Make sure the pipe characters are aligned as you place each sheet name under the keyword, as shown in Figure 3-3.

To add more sheets to your design, connect the input and output ports as described above and include the schematic file name under the |LINK keyword.

Hierarchical Designs

There are two ways to represent a lower-level drawing. One way is to use a sheet symbol, and the other is to use a sheet-path part, or library symbol.

Specifying a Sheet Symbol

There are two commands that are critical when creating a sheet symbol. They are the Name and Filename commands.

The Name command specifies the name of a sheet symbol. Because Xilinx uses the name as a part of the hierarchical path, the name of the sheet symbol must follow the Xilinx naming conventions. The name of the sheet must always be unique. The same file name, however, can be used for more than one sheet symbol.

Note: In sheet names that include more than one word, do not use spaces to separate words; use underscore characters instead.

With the Filename command, you can specify the file representing the hierarchical worksheet. Do not use the default file name. Always specify your own file name and be sure the file extension is SCH. This is the extension XMake (the automatic design processor) expects. The same rule applies for complex hierarchical structures.

For more information, refer to the *OrCAD/Schematic Design Tools Reference Guide*.

Sheet-Path Part

A flip-flop of type D, called an FD, in the XC3000 library (XC3000.lib) is an example of a sheet-path part. It is also called a library macro. Every sheet-path part consists of two parts: the worksheet, for example, FD.SCH, and the library symbol (FD symbol in XC3000.lib).

The worksheet is a drawing that contains the logic represented by the symbol. All input and output signals must be terminated by a module port.

The symbol is created in LibEdit. The input and output pin names match the defined module port names.

User-Created Libraries

As described in the previous section, OrCAD includes two different types of symbols: sheet symbols and library symbols, also known as sheet-path parts. Sheet symbols are block-shaped symbols representing lower-level hierarchies. Library symbols are symbols that have been incorporated into a library. Each time you place a sheet symbol, a new symbol must be drawn. In some cases it can be more convenient to take a duplicated sheet and make it part of a user-created library. This section details how to create a library symbol and add it to a library. Any library symbol can be placed using the Get command.

A user-created library consists of a LIB file containing any number of symbols, and an INF file for each symbol. For documentation purposes, it is a good idea to keep an SCH file, also. It is recommended that you put the files in the working directory for each design: by default, XMake and SDT2XNF look for these files in the working directory.

Note: If you place the LIB and INF files for the user-defined libraries in any directory other than the working directory, you must specify the -u option for SDT2XNF when you run XMake or SDT2XNF manually. If you place the files in the working directory, you do not need to specify the -u option for SDT2XNF.

Using OrCAD to create and save library symbols, you can load the symbols into a design in the same manner as Xilinx library elements. To create such a library element and to add it to a user library, continue with the sections:

- Creating Schematic and Netlist Files
- Creating a Symbol for the Schematic
- Saving the Symbol
- Changing Your Search Path

Creating Schematic and Netlist Files

The first step in creating a symbol is to generate a schematic file and a netlist file.

1. Create a schematic for the element using Xilinx library elements. Do not use the name of any Xilinx library component as your schematic name. Suppose, for example, that the schematic has been saved as example.sch.
2. Run the Annotate and INET programs to create an INF file.

Creating a Symbol for the Schematic

Next, create a symbol using OrCAD's LibEdit utility. Although LibEdit is described in detail in the OrCAD documentation, the steps involved in creating a symbol are given here. The library in this example is called userlib.

1. Go to the OrCAD ESP Design Environment, and select **`Schematic Design Tools`** → **`Execute`** → **`Edit Library`** → **`Execute`**.

   Alternatively, you can type **`libedit`** ↵ at the DOS prompt to enter the OrCAD LibEdit utility.

2. At the `Read Library?` prompt, type **`.\userlib.lib`** ↵.

   If this library does not already exist, the words `New Library` appear at the top of a blank sheet. If the library already exists, the sheet is still blank, and all other steps remain unchanged.

Note: To install the new library in the recommended directory, that is in the current working directory, type **`.\userlib.lib`**↵. To place the library in the same directory as the Xilinx-supplied libraries, type **`userlib.lib`** ↵. In this case, you must copy the SCH and INF files to the PLIB directory and always run XMake or SDT2XNF with the -u option for SDT2XNF.

3. To create the symbol, select **Body → Block**.
4. You are then asked to select the `Number of Parts per Package`. *You must select 0*. If you select any other number, pin numbers as well as pin names are assigned to the symbol pins. The presence of pin numbers causes serious problems with the translation software.
5. Draw the box, select **Place**, and press the **Escape** key.

   The body of the symbol is now complete.
6. Position the cursor where you want to add a pin, and select **Pin → Add**.
7. Enter a name for the pin. This name must be the same as a name on a module port in the schematic.
8. Select the appropriate pin type. You must select either **input**, **output**, or **bidirectional**. Do not choose any of the other options.
9. Select any of the possible pin shapes.

   The pin appears on the symbol with a name beside it. If a number also appears, it means you did not select 0 for the number of parts per package. This error is difficult to correct, so you might want to exit LibEdit and start the process over.
10. Repeat the last few steps until you have added all the pins.
11. Add a name to the symbol. Select **Name → Add**.
12. Type the name for the symbol, in this case, **example** ↵. The symbol name you use must be the same as the name of the corresponding schematic.
13. When prompted by `Sheetpath?`, *do not add a sheetpath name*.
14. Press ↵ and the **Escape** key.

Saving the Symbol

After creating a symbol, you must save it into the library.

1. Select **Library → Update Current**.
2. To save the library to disk, select **Quit → Update File**.

3. The library is automatically saved as userlib.lib in the working directory.
4. To exit LibEdit, select **Abandon Edits** from the **Quit** menu.

Changing Your Search Path

After saving the symbol, add the library to your search path.

1. Select **Draft → Configure Schematic Tools**.
2. Scan the list of available libraries. Click on **.\USERLIB.LIB → Insert**.

   The new library appears in the Configured Libraries list at the right.
3. Return to the top of the screen and click on **OK**.

   An alternative approach is to manually edit the sdt.cfg file using any text editor in DOS, and add a line similar to the other "LIB" statements, that is, **LIB='.\userlib.lib'**.

   To verify that the library symbol is actually in the library, you can go into Draft, select **Get**, and press ↵. A list of available libraries appears. Click on **.\Userlib.lib** and then **EXAMPLE**.

Besides the LIB file, the INF file for the block must be saved in the same directory as the LIB file. Additionally, it is a good idea to save a copy of the schematic, in case you ever wish to consult it or change it. However, neither of these rules applies in the case of a library symbol without a schematic. These exceptions are discussed in the section, "Merging Design Files from Other Sources" in Chapter 4.

If you have questions on the LibEdit program or other aspects of user-created libraries, see the *Schematic Design Tools Users Guide* and *Reference Guide* from OrCAD.

Note: Save the new library symbol to a user-created library file. Do not edit the XC3000.lib, XC2000.lib, XC4000.lib or XC7000.lib files, because future updates to the Xilinx library will overwrite user library symbols.

OrCAD Interface/ Tutorial Guide

FPGA Design Issues

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Chapter 4

FPGA Design Issues

This chapter explains the different Xilinx design requirements you should be familiar with as you complete an FPGA design.

- *Xilinx-Supplied Primitives and Macros* defines the FPGA libraries of primitives and macros.
- *Entering Xilinx Attributes* tells you how to attach attributes to macros and primitives.
- *Entering XACT-Performance Attributes* details how to enter the TIMESPEC, TIMEGRP, TNM, and TS attributes on your design.
- *Entering a Parttype Record* explains how to specify the Parttype record on your worksheet.
- *Using the Draft Edit Menu* covers all the field names available for the XC2000, XC3000, and XC4000 FPGA families.
- *Symbol and External I/O Attributes* defines the different attributes you can enter on symbols and input/output buffers.
- *Signal Attributes* lists and describes the flags you can connect to nets.
- *Representing Power and Ground Signals* explains how to tie signals High or Low in your design.
- *Merging Design Files from Other Sources* explains how to include a non-OrCAD netlist in your design.
- *Using X-BLOX Symbols* tells you how to use X-BLOX modules in your OrCAD design.

Xilinx-Supplied Primitives and Macros

The function of each Xilinx primitive and macro is described in the *XACT Libraries Guide*. This section provides OrCAD-specific information for some of the symbols.

Existing customers have two sets of libraries: old libraries and the new Unified Libraries. Only the new libraries are shipped with the new systems. You must do your design using either the old symbols or the new Unified Libraries symbols. We strongly recommend that you create all new designs using the new unified symbols. The old libraries are supported in this release and allow you to continue to process existing schematics without redrawing them. However, you must not mix and match elements from the old libraries with elements of the new libraries in the same design.

The Xilinx libraries contain primitives and macros, the building blocks of any FPGA design. Primitives are basic logic elements, such as gates and flip-flops. Macros are combinations of various primitives or other macros, used to implement common functions. For example, the ADSU8 macro represents an 8-bit adder. Macros are classified as either “soft” or “relationally placed” macros (RPMs).

Soft macros are neither partitioned nor routed. This makes them more malleable when fitted with other macros.

An RPM is a new type of macro designed to replace the Xilinx hard macros. RPMs constitute a super-set of soft macros that support FMAPs, HMAPs, carry-logic schematic elements, and absolute location (LOC) or relative location (RLOC) constraints. As most symbols in the Unified Libraries have relative location constraints, you can easily design RPMs using symbols in these Xilinx libraries. Moreover, you can simulate RPMs. RPMs are currently supported for the XC4000 family only.

Warning: User-created hard macros must be converted for use with the new Xilinx core tools. You must convert all your hard macros to RPMs using the HM2RPM utility. Refer to your *XACT Reference Guide* for more information.

The Xilinx OrCAD-FPGA library includes numerous macros, which implement common functions as well as TTL-equivalent components.

Entering Xilinx Attributes

Some Xilinx macros and primitives require that you provide attributes and attribute values. This section explains how to enter these attributes.

The different types of attributes include:

- Symbol and External I/O attributes
- Signal attributes
- User attributes

Note: The Xilinx Pin attributes are not supported in OrCAD.

After placing a symbol, you can enter attributes with the Draft Edit menu. Refer to the "Symbol and External I/O Attributes" section for a list of the different attributes that you can use with each library.

Enter the signal attributes using the Draft Get command. Refer to the "Signal Attributes" section for a list of the different signal attributes that you can use with each library.

The following figure is used to illustrate how to place attributes on your schematic. Refer to it throughout this section.

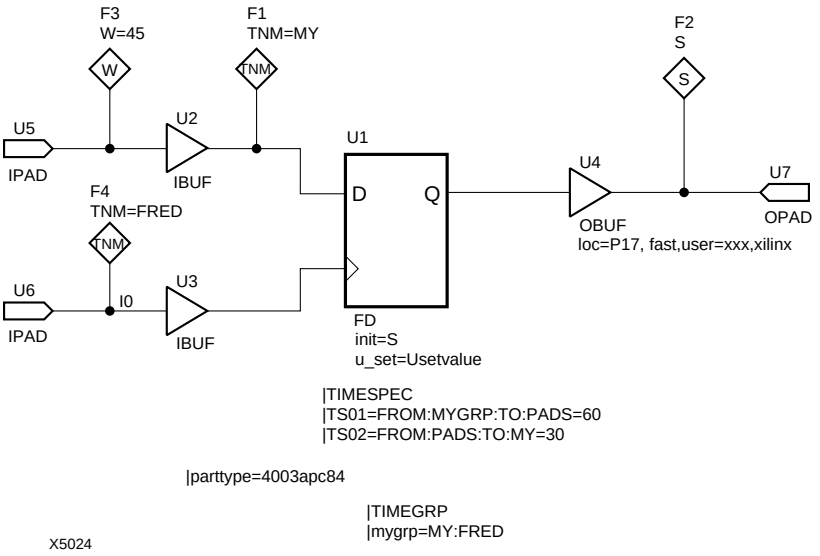


Figure 4-1 Entering Different Types of Attributes

Entering Symbol and External I/O Attributes

Symbol attributes define a symbol and its connections to signals. An external attribute is an attribute you connect to an I/O symbol, such as an IBUF, OBUF, INFF, INLAT, INREG, OUTFF, OUTFFT, or pad symbol. To enter a symbol attribute or an external attribute, do as follows:

1. Place the cursor over the symbol you wish to edit.

   Use the menu command **Edit → Edit → OPTIONS_1 → Name**, or **Edit → Edit → OPTIONS_2 → Name** for XC4000 devices.

   Use the menu command **Edit → Edit → LOC,OPTIONS → Name** for XC2000 family and XC3000 family devices.

2. At the prompt, enter an attribute name and a valid value separated by the equal sign.

   For example:

   OPTIONS_1? loc=P17

3. To specify multiple attributes, use a comma to separate the different attribute definitions:

   `OPTIONS_1? loc=P17,init=S,U_set=Usetvalue`

Some attributes, such as FAST, do not require a value. Check the *XACT Libraries Guide* for the correct attribute syntax.

Those attributes that do require a value and are listed as selections in the Edit menu can be entered from the Options field as well. In this case, you must enter the attribute, the equal sign, and the attribute value at the prompt. Alternatively, each of these attributes can be entered from the field that bears its name. In this case, you only need to enter a valid value at the prompt, as in the following example.

`OPTIONS_1? INIT=S`

or

`INIT? S`

This last command is executed by selecting **Edit → Edit → Init → Name**.

If you attempt to enter an attribute in a particular field designed for another type of attribute, your attribute name supersedes the existing field name. However, this only works if the attribute you enter is accompanied by a valid value.

Use:

`Base? U_SET=Usetvalue.`

Do not use:

`Base? U_SET`

Note: For the XC2000 and XC3000 menus, the part field used to enter attributes is called LOC, OPTIONS. If you enter a value without an attribute in this field, the value defaults to the LOC field. If you specify an attribute name with a value or an attribute that does not require a value, the specification defaults to the OPTIONS field.

Entering Signal Attributes

A signal attribute is a symbol, or flag, that you connect to a net, or signal. Use the Get command to get a flag, place the flag next to a signal, and connect the flag to the signal with a net and a junction dot.

The following is a step-by-step procedure to enter this type of attribute.

Note: The Xilinx Pin attributes are not supported in OrCAD.

1. Use the **Get** command to get the flag for the attribute.
2. At the Get? prompt, enter the flag name:

   Get? ***symbolname*** ↵

3. Use the **Place** command to fix the flag on the worksheet next to a signal.
4. Connect the flag to the net by drawing another net using the **Place** → **Wire** command.
5. Place a junction (programming intersection point) where the new net connects with the designated signal.

For the W and TNM symbols, continue with the following steps:

1. Place the cursor over the flag you have just entered.
2. Use the menu command **Edit** → **Edit** → **PartValue** → **Name** to specify a value for your attribute. At the Value? W or Value? TNM prompt, type an equal sign followed by your attribute value.

   Value? **W=45** ↵

   or

   Value? **TNM=groupname**

For a TS symbol, continue with the following steps:

1. Place the cursor over the TS symbol you have just entered.
2. Use the menu command **Edit** → **Edit** → **PartValue** → **Name** and, at the Value? TS prompt, enter the TS value without the equal sign:

   Value? **TS03** ↵

Entering User Attributes

User attributes are user specified. You can enter any specifications you want. User attributes are ignored by the Xilinx software and are available for your own documentation purposes only. Enter these attributes on library symbols by performing the following steps.

1. Place the cursor over the symbol that you wish to edit.
2. Use the menu command **Edit → Edit → OPTIONS_1 → Name**, or **Edit → Edit → OPTIONS_2 → Name** for XC4000 devices.

   Use the menu command **Edit → Edit → LOC,OPTIONS → Name** for XC2000 family and XC3000 family devices.
3. Specify your own attributes after any existing Xilinx attributes that you already specified.

   loc=P17,fast,user=xxx,xilinx

   In this example, the first two attributes are Xilinx-specific attributes. The last two attributes are user specified.
4. Press **Enter** to place the attributes on the library symbol.

Entering XACT-Performance Attributes

The following briefly explains the steps used to enter the TIMESPEC, TIMEGRP, TNM and TS attributes.

For a complete discussion of XACT-Performance, refer to the *XACT Reference Guide* and the "XACT-Performance and XDelay Tutorial" chapter in this manual.

Note: As you place the text on the worksheet, separate each attribute definition from other attribute definitions, as shown in Figure 4-1. Do not put two attribute definitions on the same line or in the same column.

TIMESPEC

To specify a TIMESPEC attribute:

1. Use the menu command **Place → Text.**
2. At the `Text?` prompt, enter the pipe character followed by the TIMESPEC keyword:

   `Text?`**`|TIMESPEC`**
3. Press **Enter** and select **Place** to place the text on your worksheet.
4. Use the same menu command again, **Place → Text**, to specify a value for your TIMESPEC. Enter a valid value as follows:

```
Text?|TSidentifier=specification
```

For example, enter a value such as |TS01=FROM:FFS:TO:PADS=30.

5. Press **Enter** and drag the specification under the TIMESPEC keyword on your worksheet, aligning it with the keyword.
6. When it is aligned, select **Place** to place the specification on your worksheet.

```
|TIMESPEC
|TS01=FROM:FFS:TO:PADS=30
```

Note: You can have several TS specifications. However, align all the pipe characters as you place the specifications under the TIMESPEC keyword.

TIMEGRP

To specify a TIMEGRP attribute:

1. Use the menu command **Place → Text.**
2. At the Text? prompt, enter the pipe character followed by the TIMEGRP keyword:

   ```
   Text?|TIMEGRP
   ```

3. Press **Enter** and select **Place** to place the text on your worksheet. Use the same menu command again, **Place → Text**, to specify a value for your TIMEGRP and enter a valid value:

   ```
   Text?|groupname1=set_definition1
   ```

4. Press **Enter** and drag the specification under the TIMEGRP keyword on your worksheet, aligning it with the keyword.
5. When they are aligned, select **Place** to place the specification on your worksheet.

Note: You can specify several group names for the TIMEGRP keyword. Align all pipe characters as you place the group names under the TIMEGRP attribute. Also, make sure you do not place a TIMEGRP and a TIMESPEC specification on the same row or column. (See Figure 4-1.)

TNM

TNM is an attribute that defines groups of nets or symbols that can be referenced by TIMESPEC or TIMEGRP specifications, as shown in Figure 4-1. Refer to the "Entering Signal Attributes" section above for information on how to attach the TNM symbol to a net and assign its attribute value.

Note: You cannot enter the TNM attribute on a pin, as none of the Xilinx pin attributes are supported in OrCAD.

TSidentifier

TS*identifier* is an attribute that identifies individual XACT-Performance timing specifications. Refer to the "Entering Signal Attributes" section for information on how to attach the TS symbol to a net and assign an attribute value.

Entering a PARTTYPE Record

The PARTTYPE record is a statement that you can enter on your schematic to specify the family and speed of your device. If you do not specify a part type for your device, the Xilinx software uses a default device for each family.

To specify a PARTTYPE record:

1. Use the menu command **Place → Text.**
2. At the Text? prompt, enter the pipe character followed by the PARTTYPE or PART keyword and its value:

   Text?**|PARTTYPE=4005pg156-7**
3. Press **Enter** and select **Place** to place the specification on your worksheet.

   |PARTTYPE=4005pg156-7

   or

   |PART=4005pg156-7

Using the Draft Edit Menu

This section discusses the Edit Part fields that were configured by the XDraft utility. Use this menu to enter attributes on Xilinx macros and primitives in your design.

XC2000/XC3000 Field Names

If you configure the Edit Part menu to support an XC2000 or XC3000 design file, the Edit Part menu, when selected, generates a menu with the following options:

LOC,OPTIONS
BLKNM
BASE
CONFIG
EQUATE_F
$FCONT
EQUATE_G
$GCONT

Location and Options (LOC,OPTIONS)

The LOC, OPTIONS field is used to enter Xilinx symbol attributes. LOC, which is short for location, is used to lock a component to a specific location and is described in the section on how to control block placement in the *XACT Libraries Guide*. Each field retains the attributes and attribute values you specify even after you place the attribute or string of attributes on your worksheet symbols. Therefore, the number of attributes you can enter in each Options field is limited to the extent of the field. To enter additional attributes, you can use the BASE, CONFIG, CONT, and EQUATE fields.

Note: If you enter a value without an attribute in the LOC,OPTIONS field, it is presumed to be a location. If you specify an attribute name with a value or an attribute that does not require a value, the specification defaults to the OPTIONS field.

Available options are listed in the section “Symbol and External I/O Attributes,” below.

Block Name (BLKNM)

This field specifies a logical block name to be assigned to a CLB or an IOB. However, it does not apply to Xilinx macros.

BASE and CONFIG Fields

Use these fields to enter CLB and IOB configurations. The BASE field sets the base configuration of any IOB or CLB. The CONFIG field specifies logic element inputs and the storage element function.

EQUATE_F, EQUATE_G, $FCONT, $GCONT Fields

Use these fields to enter CLB configurations. The EQUATE_F and EQUATE_G fields let you configure CLB logic functions by entering logic equations directly. To enter logic directly into the F function of the CLB, select **EQUATE_F** (likewise for the G function) and **Name** from the Edit Part submenus; then, type the function at the prompt.

The $FCONT and $GCONT fields in the Edit Part submenu let you continue the function definition for Boolean expressions exceeding the length of the EQUATE_F and EQUATE_G fields, respectively. Refer to the *XACT Libraries Guide* for valid tags and options for XC2000 family and XC3000 family designs.

XC4000 Field Names

If you configure the Edit Part menu to support an XC4000 design file, the Edit Part menu appears with the following options:

Reference
Part Value
OPTIONS_1
OPTIONS_2
INIT
BASE
CONFIG
EQUATE_F
EQUATE_G
EQUATE_H
SheetPart Name
Orientation

Options (OPTIONS_1 and OPTIONS_2)

The Options fields are generic. You can specify one or more of the symbol attributes listed in the section "Symbol and External I/O Attributes," following, using either of these two fields. Each field retains the attributes and attribute values you specify even after you place the attribute or string of attributes on your worksheet symbols. Therefore, the number of attributes you can enter in each Options field is limited to the extent of the field. To enter additional attributes, use the BASE, CONFIG, and EQUATE fields.

Initialization State (INIT)

This field indicates the start-up state of an IOB flip-flop and the ROMs but not the RAMs. To initialize a RAM, you must provide additional circuitry.

BASE, CONFIG, and EQUATE Fields

These fields are generic. Use them as additional Options fields when entering XC4000 family designs.

Symbol and External I/O Attributes

This section lists the symbol and external I/O attributes that you can use in your FPGA designs. It also includes a table, Table 4-1, that specifies the availability of each option for each FPGA family. For more details, refer to the *XACT Libraries Guide*.

For instructions on how to enter these attributes, refer to the section "Entering Symbol and External I/O Attributes."

Table 4-1 Symbol and External I/O Attributes

	XC2000/L	XC3000/A/L	XC3100/A	XC4000/A/D	XC4000H
BLKNM=	X	X	X	X	X
CAP					X
CMOS					X
DECODE				X	X
DEF=	X	X	X	X	X
DOUBLE		X	X		
FAST		X	X	X	
FILE=	X	X	X	X	X
HBLKNM=	X	X	X	X	X
HU_SET=				X	X
INIT=				X	X
LOC=	X	X	X	X	X
LOC<>	X	X	X	X	X
MAP=	X	X	X	X	X
MEDFAST				XC4000A	
MEDSLOW				XC4000A	
NODELAY				X	
RES					X
RLOC =				X	X
RLOC_ORIGIN=				X	X
RLOC_RANGE=				X	X
SLOW		X	X	X	
TNM=		XC3000A/L	XC3100A	X	X
TSidentifier=		XC3000A/L	XC3100A	X	X
TTL					X
USE_RLOC=				X	X
U_SET=				X	X

- BLKNM=*blockname*

  Use this field to specify the block name of a CLB or an IOB. You can also use it on primitive I/O symbols, primitive flip-flop symbols, and IOB and CLB primitives; it does not work on macros. The default block name is always the output net name of the IOB or CLB.

- CAP

  Use the capacitive, "softedge" mode on the 4000H to connect an output to a capacitive mode or to avoid ground bounce by turning off the pull-down transistor as the output is pulled to ground. Contrast this attribute with the RES attribute, which specifies the resistive mode.

- CMOS

  Use this attribute to configure output drivers on the 4000H to drive CMOS-compatible levels, and IOBs to be CMOS-compatible inputs. Contrast this attribute with the TTL attribute, which specifies TTL I/O voltage levels. Do not combine a TTL output with a CMOS input on a bidirectional pad, as this configuration is not supported.

- DECODE

  Use this attribute to indicate the function to be implemented with the dedicated decoding logic at the edges of the XC4000 die.

- DEF=*type*

  Use this attribute to indicate a Xilinx ABEL or MemGen netlist. Valid values include XABEL and MEM.

- DOUBLE

  Use this attribute to specify double pull-up resistors on a horizontal longline. If your device is processed with PPR, you do not need to use this attribute. PPR automatically uses two pull-up resistors where appropriate.

- FAST

  Use this attribute to specify a fast output pin on a pad or an I/O buffer. The output pins default to slew-rate-limited output.

- FILE=*filename*

  Use this attribute to indicate that an XNF representation exists for the symbol, that is, there exists an XNF file with the name of "filename" for the symbol. This name is the file name of a logic block that is not represented by an OrCAD schematic, for example, a Xilinx ABEL file.

- HBLKNM=*name*

  Use the HBLKNM attribute on a primitive in a macro that is instantiated several times in a design. The design-flattening process creates unique hierarchical HBLKNM values for each occurrence of the symbol.

- HU_SET=*name*

  Use this hierarchical attribute on a symbol that already includes RLOC properties. The HU_SET attribute is the same as the U_SET attribute except that it will be applied not only to the symbol with the attribute, but to all symbols below it in the hierarchy. It defines a set of symbols that will be considered related by their RLOC attributes.

- INIT=*value*

  Use this attribute to initialize XC4000 ROMs and IOB flip-flops. The values used on a ROM are 4 or 8 HEX digits. The values used on an IOB flip-flop are S (Set) or R (Reset, default).

- LOC=*location(s)*

  Use this attribute to specify a particular location or a range of locations for one or more CLBs, IOBs, flip-flops, FMAPs, HMAPs, CLBMAPs, or I/O symbols.

- LOC<>*location(s)*

  Use this attribute to avoid a particular location or a range of locations.

- MAP=*value*

  Use this attribute on CLBMAP, FMAP, and HMAP symbols to specify explicit control over partitioning. Supported values include PLC (pins locked and closed), PLO (pins locked and

open), PUC (pins unlocked and closed), and PUO (pins unlocked and open). The default value is MAP=PUC.

- MEDFAST / MEDSLOW

  Use these attributes to specify the transition time of the output drivers. MEDFAST decreases the output transition time from the default SLOW value and is slightly faster than MEDSLOW.

- NODELAY

  Use this attribute to remove the input delay from the IOB flip-flop. The default is to include the Delay buffer. The NODELAY attribute changes the setup and hold time specifications for the input flip-flop. See the *Programmable Logic Data Book* for additional information.

- RES

  Use the RES attribute on the 4000H to specify the resistive mode for your output driver. In this mode, the output is faster and draws more power. Contrast this attribute with the CAP attribute.

- RLOC=*value*

  Use the RLOC attribute to define the spatial relationships between two or more symbols. The syntax of the RLOC attribute value is RnCn, where Rn and Cn denote the row and column numbers of the LCA grid array. R0C0 is the upper left-hand corner of the group of related symbols.

- RLOC_ORIGIN=*value*

  Use this attribute on a symbol to establish the absolute position of a set of symbols already in a relative location relationship on the LCA. The syntax of the RLOC_ORIGIN attribute value is RnCn, where Rn and Cn denote the row and column numbers of the LCA grid array. The RLOC_ORIGIN is applied to the symbol with RLOC=R0C0.

- RLOC_RANGE=*value*

  Use this attribute on a symbol to establish a range within which a set of symbols, already in a relative location relationship, can be placed on the LCA. The syntax of the RLOC_RANGE attribute value is RnCn:RnCn, where Rn and Cn denote the row and column numbers of the LCA grid array.

- SLOW

  Use this attribute on the XC3000 and XC4000 families to specify the transition time of the output drivers. SLOW is the default.

- TNM=*namelist*

  Use the TNM attribute on a macro symbol, a primitive, or a net. If you specify a TNM attribute on a macro symbol, it propagates down to all lower-level symbols. You cannot put this attribute on pins because of an OrCAD limitation. The TNM attribute defines groups of nets or symbols to which an XACT-Performance (TIMESPEC) attribute can be applied. (See Figure 4-1.) Refer to the XACT-Performance chapter in the *XACT Reference Guide* for more details.

- TS*identifier=specification*

  Use the TS*identifier* attribute to convey timing information to the Xilinx place and route program (PPR). Place it in alignment with the TIMESPEC text specification. See Figure 4-1 for an example.

- TTL

  Use this attribute to configure output drivers on the 4000H to drive to TTL-compatible levels and IOBs to be TTL-compatible inputs. Contrast this attribute with the CMOS attribute, which specifies CMOS I/O voltage levels. Do not combine a TTL output with a CMOS input on a bidirectional pad, as this configuration is not supported.

- USE_RLOC=*value*

  Use this attribute on a macro symbol to tell the XNFMerge program whether the RLOC information used inside the macro symbol should be used or ignored. Valid values are True and False. The default value is True.

- U_SET=*name*

  Use this attribute on a symbol that already includes RLOC properties. It defines a set of symbols that will be considered related by their RLOC attributes.

Signal Attributes

The symbols for signal attributes are in the Xilinx library. When used, these symbols must make contact with the net to which they apply by means of a net and a junction dot. The table below lists the signal attributes (also known as flags) that apply to each FPGA family. For instructions on how to enter these attributes, refer to the section "Entering Signal Attributes."

Table 4-2 Signal Attributes

FPGA Families	Signal Attributes											
XC2000 / L	C	G	I	K	L	N	P	S				X
XC3000, XC3100	C				L	N	P	S				X
XC3000A / L, XC3100A	C					N	P	S	TNM	TS	W	X
XC4000 / A / D / H									TNM	TS	W	X

C — Critical

The C flag is used to identify a net as critical. A C flag assigns a weight of 100 to the attached net, which gives it the highest routing priority.

G — G-Function Output

Any CLB clocks driven by this net are connected to the G function generator output.

I — Input

Any CLB clocks driven by this net are connected to the C input.

K — Input

Any CLB clocks driven by this net are connected to the K input.

L — Long-Line Net

The APR software attempts to use a long line to route this net; this is useful for nets with high fan-out and needing low skew.

N — Non-Critical

The N flag is used to identify a net as non-critical. An N flag assigns a weight of zero to the attached net, which gives it the lowest routing priority.

P — Pinlock (CLBMAP Primitives Only)

This attribute specifies that APR is not to move the CLBMAP pin connected to the associated net. This is useful for aligning CLB inputs with a specified longline.

S — Save

This attribute is used to prevent XNFMAP from removing unconnected signals. If you do not have the S attribute on a net, XNFMAP removes the signal if it is not connected to some logic or an I/O primitive. The Save flag is useful for placing and routing a design with some logic blocks missing or incomplete, for preliminary testing or simulation.

TNM — Group Nets

This attribute is used to group paths with the same Timespec definition.

TS — Apply TS Attribute

This attribute is used to apply a specific TS attribute to a net.

W — Weight (Relative Routing Priority)

The W flag is used to assign a relative routing priority to a net. Attach the W flag (symbol name: W) to the desired net, placing a junction at the T connection. To specify the net weight value you must edit the part value of the W symbol, add an equal sign (=) and add a weight value to the existing text. The weight value is only legal from 1 to 99. For example, W=34 will assign the attached net a weight of 34.

X — External

The X flag is used to identify a net as external. An external net is one that exists at a CLB output, and will not be absorbed into a CLB. For

example, an external net between a logic gate and a flip-flop will force APR or PPR to place the combinatorial logic and the flip-flop in different CLBs. This may make the partitioning of the design less efficient, but will guarantee that the external net exists at a CLB output.

Representing Power and Ground Signals

The following sections describe the VCC and GND symbols used to tie a net High or Low in an OrCAD design.

VCC — Logic High

The VCC symbol ties a net to a logic High state. Get the VCC symbol (symbol name: VCC) from the library and place it on the schematic. Attach it to the desired net, using a junction dot if a T connection is formed. A net tied to VCC cannot have any other source.

When APR or PPR encounters a net tied to VCC, it removes any logic that is disabled by the VCC signal. The VCC net is only physically implemented in the FPGA if APR or PPR cannot remove the logic sourced by the signal.

GND — Ground

The GND symbol ties a net to a logic Low state. Get the GND symbol (symbol name: GND) from the library and place it on the schematic. Attach it to the desired net, using a junction dot if a T connection is formed. A net tied to GND cannot have any other source.

When APR or PPR encounters a net tied to GND, it removes any logic that is disabled by the ground signal. The GND net is only physically implemented in the FPGA if APR or PPR cannot remove the logic sourced by the signal.

Merging Design Files from Other Sources

You can enter your design as a simple set of schematics. Alternatively, you can enter part of your design in some form other than schematics, such as text entry or a RAM or ROM description. You can also bring in netlist files produced by interface software other than OrCAD. Whatever the form of entry, the starting point for inclusion

into an OrCAD schematic design must be a netlist file in XNF format. *This file must be located in the working directory*. Without the XNF file, this portion of the design cannot be included; with it, the origin of the logic becomes irrelevant.

Suppose that you choose to include an XNF file that you generated using software from one of Xilinx's Alliance partners. You have a file called, for example, example.xnf. This section describes how to incorporate an XNF netlist into your schematic.

Creating a Symbol for the XNF File

The simplest approach is to add the element to a user-created library. Use the procedure outlined in the "User-Created Libraries" section of Chapter 3. Be sure to add any new library to your search path.

Alternatively, if the XNF file was created by Xilinx ABEL or MemGen, you can bypass much of this procedure using Xilinx utilities that automatically create a symbol for you. Specifically, the following steps can be substituted for all steps outlined in the "User-Created Libraries" section, up to the "Saving the Symbol" section of Chapter 3.

1. If this is a Xilinx ABEL design and you have already produced an XNF file for it, type the following at the DOS prompt:

   `symgen example -o`↵

   where *example* is the name of a module for which you are trying to build the symbol. Symgen produces the example.cmd file.

   Do not run Symgen if you are using MemGen to create a memory block, as MemGen automatically creates the CMD file.

2. Go into LibEdit by typing **`libedit .\userlib`** ↵ at the prompt, assuming you chose "userlib" for your library name.

3. Create the new library element for the XNF file using the example.cmd file created in step 1. To execute the command file, select **`Import`** and type **`example.cmd`**↵.

   This procedure automatically creates the OrCAD symbol for you from the command file.

4. Continue with "Saving the Symbol," as described in the ''User-Created Libraries" section of Chapter 3.

Adding a Symbol to Your Schematic

Whenever a library symbol without a schematic is used in a design, you must specify the name of the XNF file. To specify the XNF file name, follow these instructions each time the symbol is placed.

1. Position the cursor on the symbol and select: **Edit → Edit → LOC,Options → Name** for an XC2000 or XC3000 design,

   or

   Edit → Edit → OPTIONS_1 → Name for an XC4000 design.

2. Type:

   FILE=example↵

   where *example*.xnf is the file representing that symbol.

Note: Do not specify the .xnf file extension after the symbol name.

3. If the file is a Xilinx ABEL file, use the same command sequence to append the string **,DEF=XABEL**. This attribute informs XSimMake that the file was generated by Xilinx ABEL.

4. If the file is a MemGen file, use the same command sequence to append the string **,DEF=MEM**. This attribute informs XSimMake that the file was generated by MemGen.

Using X-BLOX Symbols

You can use X-BLOX modules in your OrCAD XC3000A / L, 3100A, or XC4000 family design. The installation program places xblox.lib and related files in the \xact\xc4000 and \xact\xc3000 directories. You can use the other libraries available in these directories, either the XC3000 or the XC4000 library, with the X-BLOX library.

To retrieve a library component, invoke the Get command and enter the name of the X-BLOX component that you need. To choose a component, you can consult a list of available X-BLOX modules by using the Browse command and selecting the XBLOX.LIB library.

Connecting a Wire

X-BLOX buses must always be placed as wires, not buses. Therefore, do not attempt to use the Draft menu command Place Bus to connect a bus to an X-BLOX pin.

Adding Attributes

To add attributes to an X-BLOX component, find the name and proper value of the attribute in the *X-BLOX User Guide*. Use the Edit command to add the attribute using either the OPTIONS_1 or OPTIONS_2 field for an XC4000 family part, or the LOC, OPTIONS field for an XC3000A / L or XC3100A part. If you need to add more than one attribute on an XC4000 family part, use both Options fields or, if you decide to specify the attributes in the same field, use a comma to separate the attributes defined in the same field.

Processing the Design

Once an XNF file is generated, simply follow the procedure used to process X-BLOX modules. If you run XMake, this program recognizes X-BLOX designs and automatically processes them using the X-BLOX flow.

OrCAD Interface/ Tutorial Guide

EPLD Design Issues

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Chapter 5

EPLD Design Issues

This chapter explains how to create your EPLD schematic design so that it can be fitted to an EPLD device. The chapter covers the following topics:

- *Using the Schematic Library Components* explains how to take advantage of the EPLD features and how to adhere to the EPLD design rules.
- *Xilinx-Supplied Primitives and Macros* contrasts XC7000 primitives with XC7000 macros.
- *User-Defined Primitives and Macros* briefly tells you where to store your user-defined library primitives and macros.
- *Representing Power and Ground Signals* explains how to assign logical High and Low values to unconnected inputs and symbols.
- *Entering Xilinx Attributes* explains how to control logic optimization and other fitting options.

Using the Schematic Library Components

When creating an EPLD design, you must use only the components included in the XC7000 library. The XC7000 library contains components common to EPLD and FPGA designs. It also includes a few components that are specific to EPLDs. This section describes how to use the common and EPLD-specific components of the XC7000 library.

There are three basic types of components:

- Buffers or pads — These components define the input and output ports that represent the physical pins on the device.
- Standard components — These components represent the fixed

logic functions, such as the gates, adders, and counters.

- PLD components — You can define these components by means of a PLUSASM^{TM} equation file.

Many of the components in the XC7000 library are also implemented with special features that take advantage of the EPLD architecture. Such components are referred to as EPLD-specific components. Refer to Table 5-1 in the "EPLD-Specific Components" section for a list of the common modules along with their EPLD-specific counterparts. The sections that follow describe some of the features offered by the XC7000 library.

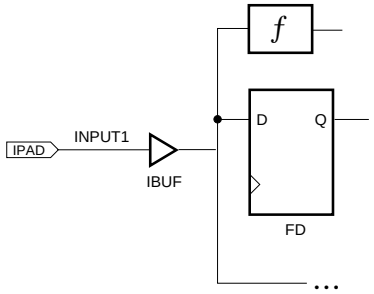
Buffers and Pads

Input and Output Buffer Connections

To represent an ordinary device input pin, use an IPAD connected to one IBUF buffer; you can then connect the IBUF to any number of on-chip logic symbol inputs. See Figure 5-1. An IBUF can drive clocks and 3-state output enables, except for OBUFEX1. There are special-purpose buffer symbols in the library, BUFG and BUFFOE, that you can use instead for these functions.

To take advantage of the input-pad registers and latches available in EPLD devices, use one of the IFD, IFDX1, or ILD symbols instead of the IBUF. Do not connect an IBUF to the D-input of an IFD/ILD symbol. Refer to the *XACT Libraries Guide* for specific application rules for the symbols.

To represent an ordinary device output pin, use an OBUF buffer that is driven by one and only one on-chip logic source. Connect the output of the OBUF to an OPAD symbol. You could also use one of the 3-state output buffers (OBUFE, OBUFT) instead of an OBUF. Drive the output enable control input (E or T) using any on-chip logic source or input signal from an IBUF. The EPLD fitter attempts to assign the enable signal to one of the EPLD's FOE global enable lines.



OR

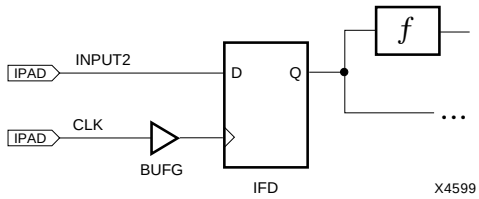


Figure 5-1 Input Buffers

To take advantage of an FOE global line explicitly, use a BUFFOE input buffer instead of an IBUF, and connect it to an OBUFEX1 output buffer instead of an OBUFE, as shown in Figure 5-2.

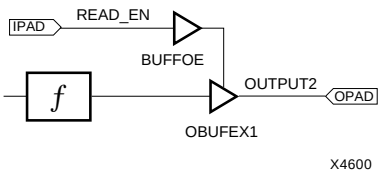


Figure 5-2 Assigning an FOE Line

Note: Always label wires connecting PAD symbols and input/output buffer symbols. These names are used by the software to refer to your device pins in the reports and during simulation.

To represent a bidirectional I/O pin, use an IOPAD symbol connected

to both the input of an IBUF or IFD / ILD and the output of an OBUFE, OBUFT, OBUFEX1, and so on as shown in Figure 5-3.

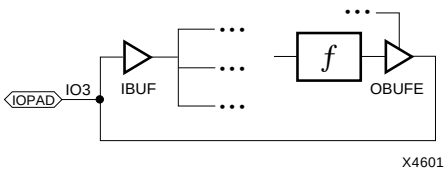


Figure 5-3 Bidirectional Pin

Output Buffers and 3-State Buffers

If a signal going into a common output buffer (OBUF) is generated by any component containing a 3-state buffer, such as a BUFE or a PLD, the 3-state control signal is used to enable and disable the device output pin driver. This behavior is unique to EPLDs and is not reproduced in FPGAs.

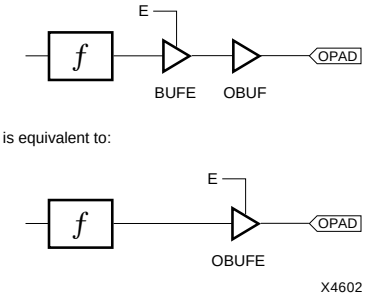


Figure 5-4 Output Enable Behavior in EPLDs

Note: Inserting a buffer (BUF) or inverter (INV) between the 3-state buffer (BUFE) and the output buffer (OBUF) does not prevent the 3-state buffer from controlling the device pin. The XEPLD fitter optimizes all buffers and inverters into the 3-state buffers unless you place an OPT=OFF attribute on the 3-state buffer.

If you use a PLD symbol in your schematic and connect one of its outputs to an output buffer such as OBUF, you can control the EPLD

device output pin using a 3-state control equation in the PLD, as shown in Figure 5-5.

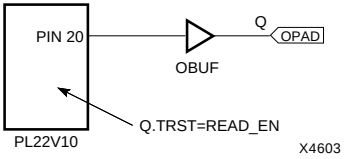


Figure 5-5 Controlling Output Using a PLD Equation

If you want to use a PLD output with a TRST equation to control a bidirectional I/O pin of the EPLD, connect the OBUF output to an IOPAD and an IBUF or an IFD/ILD. If the same PLD symbol that generates the output is also to receive the I/O pin input, you must use a separate pin of the PLD symbol to receive the signal from the IBUF. Do not tie the output of an IBUF to the input of the OBUF of the same IOPAD as shown in Figure 5-6; these input and output wires must remain separate as shown in Figure 5-7.

Rules for connecting PLD symbols also apply to any custom symbols defined by equation files or macro schematics.

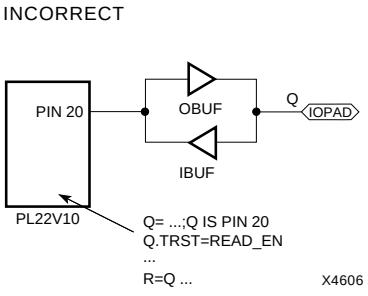


Figure 5-6 PLD Component Controlling a Bidirectional Pin (Incorrect Way)

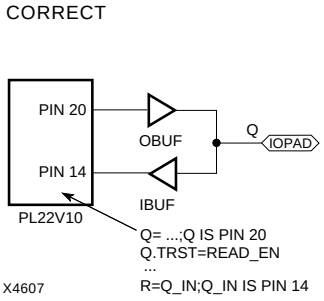


Figure 5-7 PLD Component Controlling a Bidirectional Pin (Correct Way)

If your design calls for 3-state multiplexing of multiple output sources, it is best to output each signal source on its own set of 3-state output pins and tie the signals together off-chip. You cannot connect more than one signal source to the same OBUF or OPAD, as shown in Figure 5-8. Instead, use the connection shown in Figure 5-9.

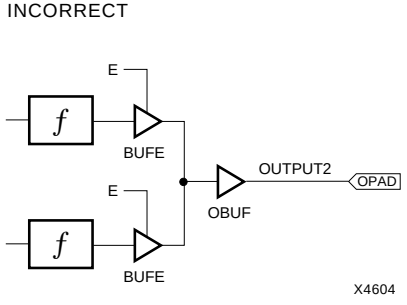


Figure 5-8 Incorrect 3-State Multiplexing of EPLD Outputs

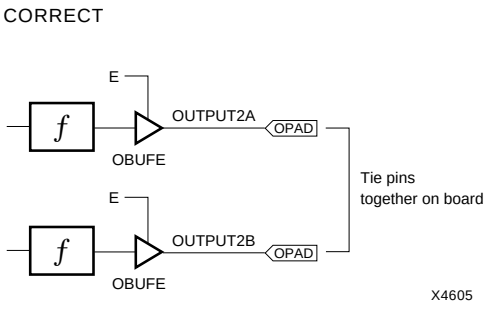


Figure 5-9 Correct Off-Chip 3-State Multiplexing of EPLD Outputs

On-Chip 3-State Multiplexing

EPLD components emulate 3-state signals internally by gating the macrocell feedback to the UIM™. (Macrocell feedback signals are never physically in a high-impedance state.) You can tie together the outputs of multiple 3-state buffer symbols, such as BUFE or BUFT, or 3-state PLD outputs to multiplex these signals on-chip as shown in Figure 5-10. You cannot connect such tied signals to an output buffer. You must connect the tied signal to a logic symbol, such as BUF before driving an output port.

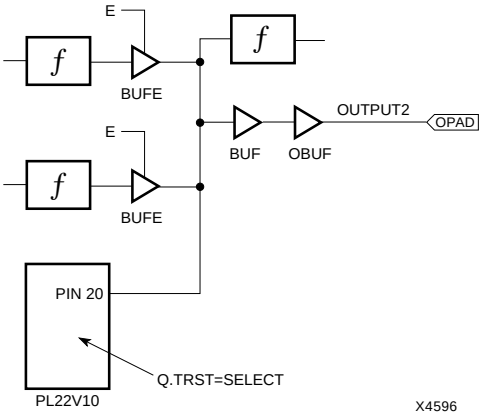


Figure 5-10 On-Chip 3-State Multiplexing

Input Buffers, Clocks, and Global Control Nets

You can connect the clock pin of any FD component or registered component to an ordinary logic signal, an IBUF, or a BUFG (FastCLK^{TM}) unless otherwise specified in the *XACT Libraries Guide.*

The input of a BUFG component must connect directly to a clock pin. There can be no other components between the pin and the BUFG.

IFD and ILD components must have a BUFG clock input.

After assigning any BUFGs to FastCLK pins, the XEPLD implementation software tries to assign IBUF signals driving clock inputs onto FastCLK pins.

The XEPLD software also attempts to optimize FD (D-type flip-flop) components into IFDs on the input pads. No other registers are ever optimized into the input pad.

If your design requires a global clock enable, you must use IFDX1 components. The CE input to these components can only be driven by a BUFCE, and the clock must be driven by a BUFG as shown in Figure 5-11.

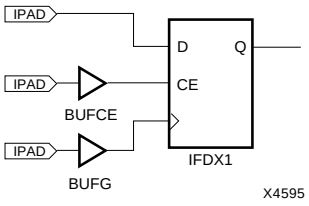


Figure 5-11 Use of the IFDX1 Symbol

Note: You can prevent input register optimization for the whole design using the REG_OPT=OFF global attribute. You can prevent clock optimization for the design using the CLOCK_OPT=OFF global attribute.

EPLD-Specific Components

In general, it is best to use EPLD-specific components whenever their features apply to your application. They are more efficient as they are designed to take advantage of special architectural features. For example, EPLD-specific adders take advantage of fast carry lines. The EPLD-specific library symbols are listed in Table 5-1.

However, if you want your design to be retargetable to either an EPLD or an FPGA device, you should use only the common library components. Most EPLD-specific components have at least one common counterpart.

Note: Each common component functions the same way in EPLD and FPGA devices. However, there might be a significant difference in efficiency or performance between the families. Whenever you map a design to a new device, you should do the following:

- Carefully check the reports created during integration to verify that the style you used to draw the design does not consume excessive logic resources in your target device.
- Perform timing analysis or simulation to catch any inefficient parts of the design.

If you need further information on XEPLD software and design techniques, see the *XEPLD Design Guide*.

Table 5-1 Common and EPLD-Specific Symbols

Common Symbols	EPLD-Specific Symbols
Accumulators	
ACC1	ACC1X1 or ACC1X2
ACC4	ACC4X1 or ACC4X2
ACC8	ACC8X1 or ACC8X2
ACC16	ACC16X1 or ACC16X2
Adders	
ADD1	ADD1X1 or ADD1X2
ADD4	ADD4X1 or ADD4X2
ADD8	ADD8X1 or ADD8X2
ADD16	ADD16X1 or ADD16X2
ADSU1	ADSU1X1 or ADSU1X2
ADSU4	ADSU4X1 or ADSU4X2
ADSU8	ADSU8X1 or ADSU8X2
ADSU16	ADSU16X1 or ADSU16X2
Counters	
CB2CLED	CB2X1
CB4CLED4	CB4X1
CB8CLED4	CB8X1
CB16CLED4	CB16X1
Input Registers	
IFD	IFDX1
IFD4	IFD4X1
IFD8	IFD8X1
IFD16	IFD16X1
Output Buffers	
OBUFE	OBUFEX1

Counters

Up / down counters in the common library, such as CB8CLED, have a single CEO output, which changes in response to the up / down direction input. Gating of this CEO function in this manner does not allow it to be optimized for zero delay. As a result, if common up / down counters are cascaded, they cannot operate at their maximum original frequency.

The EPLD-specific up / down counters, such as CB8X1, have separate outputs for the up and down directions, CEOU and CEOD, which are optimizable. You can cascade these components without impacting their maximum frequency.

Arithmetic Components

The ADD, ADDSU, and ACC types of common library components use the EPLD macrocell carry-chain between outputs of the same component, but they do not use it for cascading. If you cascade these components, the CI and CO carry signals go through the UIM and incur an extra macrocell delay. The CI and CO pins can be connected to any ordinary logic components or I / O ports but not to the CI and CO pins of EPLD-specific arithmetic components.

EPLD-specific adders and accumulators — whose names end in X1 or X2, such as ADSU8X2 — use the EPLD macrocell carry-chain for cascading without incurring a UIM delay. You can connect their CI and CO pins only to the CI and CO pins of another EPLD-specific arithmetic component or of the PLFB9 symbol.

Note: None of the accumulator symbols are supported by the XC7272 devices.

PLD Components

The XC7000 library contains symbols representing industry standard Programmable Logic Devices (PLDs) such as the PL22V10 and the PL20V8. When using these PLD devices, you must link the PLD component instance in your schematic to the associated PLD equation file or to the imported JEDEC file. The PLD equation file, processed by the PLUSASM assembler, defines the functionality of the PLD component. For instructions on how to link the PLD symbol and its equation file, see the description of the PLD attribute later in this chapter.

Because the functionality of a PLD is defined outside the schematic, designs containing PLDs cannot be functionally simulated prior to being integrated into a device using the FITNET program. Instead of performing functional simulation, integrate your design and perform timing simulation on the completed design.

The PLD components are PL20V8, PL22V10, PL20PIN PL24PIN, PL48PIN, PLFB9, and PLFFB9.

Components Not Supported by Some EPLD Devices

XC7272 devices do not support any of the common or EPLD-specific accumulator components.

The following components require features implemented only in High-Density Function Blocks. Therefore, they are not supported by the XC7336 devices, which contain only Fast Function Blocks.

- ACC symbols
- ADD symbols
- ADSU symbols
- BUFCE
- BUFT
- COMPM
- FDCP, FDCPE
- IFD
- IFDX1
- ILD
- LD
- OBUFT
- OFDT
- PLFB9
- XOR7, XOR8, XOR9

Note: The BUFE component is allowed for external outputs only and must allow FOE optimization.

Xilinx-Supplied Primitives and Macros

Each symbol in the library is either a primitive or a macro. The primitive logic components in the XC7000 library are all implemented using PLUSASM equation files included in the software. The single I/O pads, buffers, and input registers are also primitives, but they have no PLUSASM descriptions.

The PLUSASM equations which define most of the XC7000 library components are supplied for your reference in the \xact\examples\behavior\library directory. You can copy and edit these equation files as a convenient way to implement customized logic components.

A macro refers to any symbol represented by an OrCAD schematic included in the XC7000 library. The macro schematics contain either XC7000 primitives or other macro symbols, or both. When your design schematic is read by the software, the macro symbols are expanded into their underlying schematics. The components actually processed and listed by the software in the reports are the underlying primitives, referenced by their hierarchical instance names, after macro expansion has taken place.

A symbol implemented as a primitive in the EPLD library can be implemented as a macro in the library of another device family, and vice versa.

User-Defined Primitives and Macros

The procedure used to create a custom macro symbol is the same for EPLDs and FPGAs, as described in the section “Creating a Symbol for the Schematic” in Chapter 3. You can create a custom library file containing your custom macro and primitive symbols using the OrCAD LibEdit utility. You can draw macro schematics consisting of other XC7000 and custom library symbols. Store your library file and your macro schematics either under the \xact\xc7000 library directory or in your local design directory. Do not add to or modify the xc7000.lib file or any of the library macro schematics supplied by Xilinx.

User-defined primitive symbols are defined very much like PLD library symbols, except that you create your own custom symbols. Create and store your custom symbol in your user library file as you

would a custom macro symbol, but do not draw a macro schematic for it. Instead, create a PLUSASM equation file that defines the desired functionality and place it in your design directory.

Assemble your equation file using the PLUSASM command from the XDM Translate menu to debug the file and prepare it for use in your designs. In your PLUSASM equation file, specify the keyword COMPONENT as the target PAL type in the CHIP statement. This definition tells the software to look for pins on your custom symbol with the same names as the signals used in your equation file.

If you want to use a custom primitive in more than one design, you must copy and assemble your PLUSASM equation file in each design directory.

Representing Power and Ground Signals

Connect all unused inputs on symbols; warnings are issued by the software if you do not connect them. Tie unused inputs to a constant High or Low logic level in the schematic. The following sections describe the VCC and GND symbols used to tie a net High or Low in an OrCAD design.

VCC — Logic High

Use the VCC symbol to tie a net to a constant logic High state. Get the VCC symbol from the XC7000 library and place it on the schematic. Attach it to the desired net, using a junction dot if a T connection is formed. A net tied to VCC cannot have any other source. As an alternative, you can specify a constant High value by connecting a wire segment to a component pin and labeling the wire VCC.

When FITNET encounters a net tied to VCC, it removes any logic that is disabled by the VCC signal. A VCC signal is only physically implemented in the EPLD if FITNET cannot remove the logic sourced by the signal.

GND — Ground

Use the GND symbol to tie a net to a constant logic Low state from the same library to specify a constant Low logic level. Get the GND symbol from the library and place it on the schematic. Attach it to the desired net, using a junction dot if a T connection is formed. A net

tied to GND cannot have any other source. As an alternative, you can specify a constant Low value by connecting a wire segment to a component pin and labeling the wire GND.

When FITNET encounters a net tied to GND, it removes any logic that is disabled by the ground signal. A GND signal is only physically implemented in the EPLD if FITNET cannot remove the logic the signal sources.

Note: One exception is that unused pins of PLD symbols (those marked NC in the equation file pinlist) should remain unconnected. Another exception is that unused channels of multibit I/O ports and buffers may be left unconnected. If, for example, only six inputs of an IPAD8 and IBUF8 are used in the design, you can leave two of the input and output pins of each symbol unconnected. If inputs are tied High or Low, or if they are left unconnected, the software removes the logic associated with those inputs.

Entering Xilinx Attributes

To control the way the software processes your design, use attributes on your schematic. Areas you can control include:

- Linking of PLD symbols and PLUSASM equation files
- Pin assignments
- Power consumption
- Optimization of logic, registers, and control signals
- Allocation of Fast Function Block resources

When you integrate your schematic design using the XEMake or FITNET command, the schematic is converted to a PLUSASM equation file. Many of the attributes in a schematic are translated to PLUSASM declarations in the equation file.

Attributes enable you to express information that is specific to each design, as opposed to run-time options, which you enter through the XDM menu or command line to modify the way a specific program processes your design at a particular time. Attributes are classified according to the manner in which you place them on your schematic:

- Place component attributes, such as PLD, OPT, and LOC, on selected symbols. Attributes of this type affect only the component instances on which they are placed.
- Place global attributes, such as PRELOAD_OPT and LOGIC_OPT, into the schematic as text items. This type of attribute affects the entire design.
- Place the PARTTYPE attribute as a text item anywhere on your schematic.
- Place signal attributes, such as F and H on wires. This type of attribute affects individual component inputs or outputs and is represented by library symbols that are connected via wires.

Component Attributes

The component attributes specific to EPLD designs include:

- PLD=*filename*
- LOC=*pinname*
- LOWPWR=*value*
- OPT=*value*

  where *value* is On or Off.

Use the OrCAD Edit command to assign a component attribute by using the following procedure:

1. Position the cursor over the schematic symbol.
2. Select **Edit → Edit**.
3. Select any of the eight available part fields.
4. Select **Name**.
5. At the `Name?` prompt, enter an attribute name and a valid value separated by the equal sign.

   attributename=value↵

   For example:

   loc=P17

6. To specify multiple attributes, you can use multiple part fields or you can use commas to separate multiple attribute definitions in the same part field, as follows:

   PLD=RCVR,OPT=OFF,LOWPWR=ON

PLD Attribute: PLD Equation File Name

The PLD=*filename* attribute on a PLD symbol specifies the name of the file with the logic equations for that PLD. This attribute is valid on custom primitive symbols (where you would use COMPONENT as the target in PLUSASM) and on the following PLD symbols from the library: PL20V8, PL22V10, PL20PIN, PL24PIN, PL48PIN, PLFB9, and PLFFB9. Do not specify the file extension in the PLD=*filename* attribute.

You must also specify this filename as the first parameter of the CHIP statement inside the equation file, as described in the PLUSASM Language Reference section of the *XEPLD Reference Manual*. For example:

CHIP *filename* PL22V10

Within the PLD file, the pin list must contain the names of all the signals connected to all the PLD pins, in the proper order. For example, if you have the signals shown in Figure 5-12, include the following pin list in the equation file:

```
TITLE CNTR6

CHIP   CNTR6   P16V8;

;PINLIST (Highest pin number = 20)
       BUFG NC NC   rd cs NC NC NC NC NC
       NC   NC read c5 c4 c3 c2 c1 c0 NC
```

The format of the equation file must conform to PLUSASM syntax that is valid for PLD components in schematics; it should not contain declaration statements reserved for stand-alone behavioral designs. You can generate the PLUSASM file manually (using a text editor) or translate it from a higher-level compiler, such as ABEL, Xilinx ABEL, CUPL, or PALASM, or from a PAL JEDEC file.

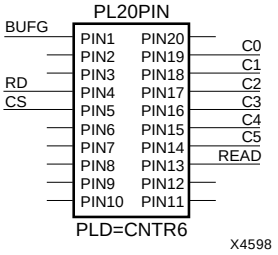


Figure 5-12 PLD Pins

All PLD components in your schematic design must include the PLD attribute. When you run XEMake, the program automatically assembles the equation files referenced by the PLD=*filename* attributes on the schematic. If you do not use XEMake, you must assemble each PLD file in the design with the PLUSASM command before you run FITNET.

Like PLDs, user-specified primitives are defined by PLUSASM equation files. The PLD=*filename* attribute is not required but can be applied as a convenient way to have your equation file automatically assembled when you invoke XEMake. If you omit the PLD attribute, FITNET expects to find a bitmap file for the symbol (*symbolname*.vmh) in your local clib subdirectory.

LOC Attribute: Pin Assignments

Use the LOC=*pinname* attribute on a PAD symbol to assign the signal to a specific pin. The PAD symbols are IPAD, OPAD, IOPAD, and UPAD. The pin name is P*nn* for PC packages, where *nn* is the pin number. The pin name is *rc* (RowColumn) for PG packages. Examples are LOC=P24 and LOC=G2.

Pin assignments are unconditional in that the software does not attempt to relocate a pin if it cannot achieve the specified assignments. You can apply the LOC attribute to as many PADs in your design as you like. However, each pin assignment further constrains the software as it automatically allocates logic and I/O resources to internal nodes and I/O pins with no LOC attributes.

To save all resulting pin assignments to preserve them for the next time you modify and re-integrate the design, use the PINSAVE command in the XDM Translate menu. This command saves the pin assignments to a *designname*.vmf file. You can override individual pin assignments saved in the VMF file by changing or adding LOC=*pinname* attributes in the schematic.

Note: Pin assignments using the LOC attribute are not supported for bus pad symbols such as OPAD8.

LOWPWR Attribute: Power Setting

This attribute is valid only for XC7300 designs. You can use the LOWPWR attribute either as a global or a component attribute.

The default is LOWPWR=OFF (high speed) for all macrocells used in the design unless otherwise specified.

To determine the power setting of the macrocells used by an individual symbol, use LOWPWR=ON, or use LOWPWR=OFF if the global LOWPWR=ALL was used. This attribute is ignored if it is assigned to a symbol that uses no macrocells, such as an inverter, an optimized AND/OR gate, or an input register. To change the default power for an entire design, refer to the global attribute LOWPWR.

OPT: Logic Optimization Attributes

Use the logic optimization attributes to control the optimization of part or all of your design.

The OPT=OFF component attribute inhibits logic optimization of all macrocells used by a symbol. OPT=ON can override the LOGIC_OPT=OFF global attribute for individual symbols.

The logic optimizer collapses the levels of logic to remove intermediate nodes. Components are optimized forward into components connected to their outputs.

If you build combinatorial logic using low-level gates and multiplexers, the software attempts to pack all logic bounded between device I/O pins and registers into a single macrocell.

The logic optimizer moves logic forward into components connected to their outputs. If collapsing an expression into all fanouts succeeds, the original macrocell logic becomes unused and is removed. The

logic optimizer does not collapse an expression into its fanouts if the resulting expression uses too many product terms or inputs.

The logic optimizer also moves forward any logic, whether combinatorial or sequential, that is buffered by a 3-state buffer. However, logic that itself contains a 3-state control is not moved forward.

Specifying OPT=OFF prevents the software from optimizing a component forward into any of its fanouts or into any non-macrocell resource, such as the UIM. Placing OPT=OFF on a component does not prevent the logic of another component from being collapsed into one of its inputs.

The OPT attribute has no effect on any symbol that contains no macrocell logic, such as an I/O buffer.

To change the logic optimization default for the entire design, refer to the global attribute LOGIC_OPT.

Global Attributes

The global attributes specific to EPLD designs include:

- LOWPWR=ALL
- LOGIC_OPT=OFF
- MRINPUT=ON
- MINIMIZE=OFF
- UIM_OPT=OFF
- FOE_OPT=OFF
- CLOCK_OPT=OFF
- REG_OPT=OFF
- PRELOAD_OPT=OFF

Global Attributes

To use one or more EPLD-specific global attributes, you must first place the keyword text string |GLOBAL into your schematic. Then, list each global attribute beneath the |GLOBAL keyword. Precede each attribute string with the pipe character (|). Align the pipe

characters of all attribute strings directly beneath the pipe character of the |GLOBAL keyword and leave no blank spaces between text lines. For example:

```
|GLOBAL
|LOWPWR=ALL
|PRELOAD_OPT=OFF
```

LOWPWR=ALL Attribute: Power Setting

This attribute is valid only for XC7300 designs. You can use the LOWPWR attribute as either a global or a component attribute.

To specify the default power setting for the entire design, place the global attribute LOWPWR=ALL in the schematic. To set LOWPWR off for individual components, use the LOWPWR=OFF component attribute.

LOGIC_OPT Attribute: Logic Optimization

To inhibit logic optimization by default for the entire design, apply the global attribute LOGIC_OPT=OFF. If you do not use this attribute, the default LOGIC_OPT=ON is used. You can override this setting for individual symbols using the OPT=ON component attribute.

MRINPUT Attribute: Master Reset Pin

Specifying the MRINPUT=ON global attribute in an XC7354 or XC7336 design changes the Master Reset pin to an ordinary input pin. If this attribute is specified, the EPLD device is initialized only at power-up.

MINIMIZE Attribute: Logic Minimization

Use the MINIMIZE=OFF global attribute to inhibit logic minimization for the whole design. If this attribute is not specified, any redundant or non-effective logic found in any user-specified equation files is eliminated through Boolean minimization.

UIM_OPT Attribute: UIM Optimization

To inhibit UIM optimization for the entire design, apply the UIM_OPT=OFF global attribute.

UIM optimization extracts AND expressions and inverters out of macrocell logic functions and moves them into the UIM, which reduces the use of Function Block resources.

FOE_OPT Attribute: Fast Output Enable Optimization

To inhibit FOE (Fast Output Enable) optimization for the entire design, apply the FOE_OPT=OFF global attribute.

FOE optimization generally applies only to BUFE, OBUFE, or 3-state PLD outputs driving an OBUF. FOE optimization changes a product-term 3-state signal to an FOE global control signal. Like the FastCLK assignment, this attribute reduces the number of UIM inputs and product terms required by each Function Block.

CLOCK_OPT Attribute: FastClock Optimization

To inhibit FastCLK optimization for the entire design, apply the CLOCK_OPT=OFF global attribute.

FastCLK optimization changes an ordinary clock signal to a FastCLK global signal. This optimization reduces the number of UIM inputs and product terms required by each Function Block.

REG_OPT Attribute: Input Register Optimization

To inhibit input register optimization for the entire design, apply the REG_OPT=OFF global attribute.

Input register optimization reduces the number of macrocells in a design by moving simple FD registers connected to IBUFs into a pad register, provided that the IBUF has no other fanouts. The clock by which the input register is controlled must be a FastCLK or an input that can be assigned to a FastCLK pin.

PRELOAD_OPT Attribute: Preload Values

Apply the PRELOAD_OPT=OFF global attribute to prevent the XEPLD software from changing the preload values of registered components in the design to match the preload values supported by specified device resources such as FFBs and input registers. The default, PRELOAD_OPT=ON, allows the XEPLD software to map your design most efficiently using the device resources most suited to the elements of your design. Unless you specify

PRELOAD_OPT=OFF, the software is free to change the initial register states of any component, including PLD custom components defined in PLUSASM. Use PRELOAD_OPT=OFF to preserve the initial states specified in the *XACT Libraries Guide* for library components and in the PRLD equations in your PLUSASM file for PLD or custom components.

You can set a High or Low preload for High-Density Function Blocks. The preload value of Fast Function Blocks depends on the use of set or reset. Input register preload values are fixed at 1, except for those on the XC7272, which are undefined.

The PARTTYPE Attribute

You can place the PARTTYPE attribute on your schematic to select the target EPLD device for your design. Refer to the Release Notes or the XDM Part menu for a list of EPLD device names supported by the software. Do not include the XC prefix in the PARTTYPE attribute.

You can override any PARTTYPE attribute already defined in your schematic by selecting a part type other than InDesign from the XDM menu before invoking the Fitter program.

Unlike global attributes used for EPLD designs, the PARTTYPE attribute is a stand-alone text string that can be placed anywhere in the schematic and does not need to be placed beneath the |GLOBAL keyword in the schematic. Simply place the text |PARTTYPE=dddd-sspppp anywhere on your schematic.

The default part type used for the XC7000 family, if it is not already specified in the schematic or in XDM, is 73108-12BG225, the highest density EPLD at the time of this release.

Signal Attributes

The signal attributes used by EPLD include:

- F — Fast Function Block or FastInput
- H — High-Density Function Block

A signal attribute is a symbol, or flag, that you connect to a net, or signal. Use the Get command to get a flag, place the flag next to a signal, and connect the flag to the signal with a wire and a junction dot if a T connection is formed. The following is a step-by-step

procedure to enter this type of attribute.

1. Select **Get** to get the flag symbol for the attribute.
2. At the Get? prompt, enter the flag name:

   symbolname ↵
3. Select **Place** to fix the flag on the worksheet next to a signal.
4. Connect the pin on the flag to the signal with a wire using the **Place → Wire** command.
5. Place a junction dot where the new wire connects with the designated signal as shown in Figure 5-13.

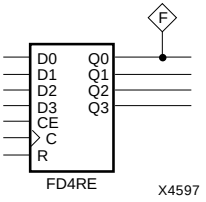


Figure 5-13 Applying the F Signal Attribute

F/H

Use the F or H signal attribute in an XC7300 device to specify whether a macrocell implementing a component output should be placed in a Fast Function Block (F) or a High-Density Function Block (H). These attributes are represented in OrCAD by a primitive symbol, called F or H, which is connected to the same wire that connects to the desired component output.

You can also use the F attribute on a wire connected to an IBUF to indicate a FastInput signal. Only a component implemented in a Fast Function Block receives this signal from the FastInput path. Any other high-density function block (HDFB) component receives the same input from the UIM. Except for using F on an IBUF signal, it is not valid to use F or H attributes on signals originating from any I/O buffer symbol, such as IFD or OBUF.

The F attribute is not valid on the outputs of components that require features implemented only in High-Density Function Blocks, such as the following types of symbols:

- ACC
- ADD
- ADSU
- COMPM
- FDCP, FDCPE
- LD
- PLFB9
- XOR7, XOR8, XOR9

Note: A BUFE symbol can only be implemented as an FFB output when it is driving an OBUF, and it must allow for FOE optimization.

The H attribute is not valid on the outputs of a PLFFB9.

For logic that is not labeled with F or H attributes, the XEPLD software attempts to put as much logic as possible in the Fast Function Blocks first, then it fills the High-Density Function Blocks.

OrCAD Interface/ Tutorial Guide

Functional Simulation

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Chapter 6

Functional Simulation

This chapter shows you how to create files so that you can use Simulate, the OrCAD/VST386+ simulation program, to simulate an FPGA or EPLD design. Included are instructions for creating files for functional simulation. The process is the same for FPGA and EPLD designs unless otherwise indicated. For instructions on how to use Simulate to simulate your designs, refer to the section "Simulating Your Design" in this chapter.

Note: You can specify trace and stimulus information on your schematic or using the simulation tool in OrCAD. Refer to your OrCAD documentation for information on how to specify trace and stimulus information from OrCAD/VST386+ or see the "VST Tutorial" chapter of this manual.

This chapter contains the following sections:

- *Creating a Functional Simulation Netlist* — This section tells you how to create a functional simulation netlist.
- *XSimMake Summary* — This section summarizes the steps performed by XSimMake, the tool used to create a simulation netlist.
- *FPGA Designs with IOB and CLB Elements* — This section explains how to create a functional simulation netlist for a design that contains IOB and CLB primitives.
- *Simulating Your Design* — This section tells you how to simulate a design functionally.

Creating a Functional Simulation Netlist

Use XSimMake to invoke the required translation programs automatically and convert the top-level design into VST, AST, and

ATR files. The whole translation process is automatically completed for you in a single step.

You can invoke XSimMake as a command line entry from the DOS prompt or through the XACT Design Manager. The steps for invoking XSimMake using the two methods are presented below.

From the XACT Design Manager

To create a functional netlist file, follow these simple steps:

1. Invoke **XSIMMAKE** from the XDM **Verify** Menu to bring up the XSimMake options menu.
2. Select the **-F** option.
3. Choose **OrCAD_Fpga_Func** for FPGA designs or **OrCAD_Epld_Func** for EPLD designs from the list of flows.
4. Select **Done**.
5. Select the file name for the design that you wish to simulate.

This process automatically creates a functional simulation netlist. For information on the steps performed by XSimMake, refer to the next section, "XSimMake Summary."

From the DOS Prompt

To run the XSimMake program from DOS, use the command line syntax shown below.

To process an FPGA design, use the following command:

xsimmake -f orcad_fpga_func [-*options*] *designname*↵

or

xsimmake -f off [-*options*] *designname*↵

To process an EPLD design, use the following command:

xsimmake -f orcad_epld_func [-*options*] *designname*↵

or

xsimmake -f oef [-*options*] *designname*↵

Note: This command can also be executed from the XACT Design Manager command line.

XSimMake Options

The following are the XSimMake program options:

-f — Flow Name

This option enables you to select the flow to run (functional or timing simulation flow for FPGA or EPLD designs).

-h — Help

This option provides you with help on XSimMake commands.

-l — List Flows

This option lists all available flows. Possible flows include FPGA timing, EPLD functional, and so forth.

-o — Old Libraries

This option tells XSimMake to use the old (pre-unified) libraries. Use this option only if your design was created using the old libraries.

Note: EPLD functional simulation is not supported for old libraries; use timing simulation instead.

-p — Parttype

This option enables you to select a part type. The designated part type overrides any part type already specified in your design.

-r — Force Re-execution

This option forces re-execution of all programs in the flow whether or not the design has changed.

-v — Verbose

This option enables you to view the progress of the program.

Converting Trace and Stimulus Files

Run ASCTOVST from the Verify menu to convert the AST and ATR files to STM and TRC files. If AST and ATR files already exist in your project directory, they are not overwritten. To overwrite existing AST and ATR files, you must select the -w option for XNF2VST in your XDM Profile menu.

XSimMake Summary

The following table summarizes all the steps that XSimMake performs automatically for the functional flow of all designs:

XSimMake Command Summary: Functional Flow		
XSimMake performs the following steps:		
Annotate	Adds reference designators to symbols in your schematic.	All designs
INET	Writes an OrCAD netlist format file (INF file) for each schematic.	
SDT2XNF/INF2XNF	Translates each INF file into an XNF file.	
XNFMerge	Merges all XNF files into one flat XNF format file (XFF file). XNF files generated by Xilinx ABEL and MemGen are also merged for FPGA designs.	
If there are X-BLOX modules, XSimMake performs these additional steps:		
XNFPrep drc_only=true	Runs an Electrical Rule Check and generates an XTG file. Does not perform logic trimming.	XC3000A / L, XC3100A, and XC4000 designs
X-BLOX mergeio=false archopt=false	Synthesizes the X-BLOX modules and generates an XNF format file (XG file).	
For all designs, XSimMake continues with the following steps:		
XNF2VST/XNF2INF option -u	Generates a VST file. Performs unit delay translation (creates no DBA file). Generates AST and ATR files if you specified trace and stimulus information in your design and the files do not already exist.	All designs

Note: Do not use the INF files generated by the INET program for functional simulation of Xilinx designs.

You can only perform functional simulation on an EPLD design if the design consists entirely of self-defined library components. If your design contains any PLD symbols or user-specified primitive symbols defined by an equation file, you cannot perform a functional simulation. Instead, implement the design and perform a timing simulation as described in Chapter 8, "Timing Simulation."

For EPLD designs, you must use XSimMake to prepare a functional simulation netlist file (VST file). Do not use the individual netlist translation programs, SDT2XNF and XNF2VST, as they do not process EPLD schematics in a manner suitable for functional simulation. You can only use STD2XNF to create files for implementation. XSimMake creates special XNF files for EPLD library primitives in a separate subdirectory named "func," which is used to prepare the functional simulation netlist. The XNF files in the "func" directory must not be mixed with the XNF files used for design implementation in the main design directory.

FPGA Designs with IOB and CLB Elements

You can only run XSimMake on designs that do not include CLB or IOB primitives. If your FPGA design does include CLB or IOB primitives, you must run XMake and then run the verification programs manually rather than using XSimMake, or implement your design with XMake and perform a timing simulation.

To manually create a functional simulation netlist, perform the following steps:

1. Invoke **XMake** from the XDM **Translate** menu or from DOS. This generates a placed and routed LCA file. For more information on creating an LCA file, refer to Chapter 7, "Design Implementation."
2. Invoke **LCA2XNF** from the **Verify** menu or from DOS, specifying the **-u** option. LCA2XNF translates your LCA file into an XNF file.
3. Run **XNFBA** from the **Verify** menu to combine the pre-route XFF or XG (X-BLOX) file and post-route XNF file into a new file. By default, XNFBA creates an output file called xnfba.xnf.
4. Select **XNF2VST** from the **Verify** menu, specifying the **-u** option.

The -u option performs unit delay translation. The program also generates AST and ATR files if you specified trace and stimulus information in your design and the files do not already exist in your working directory.

5. Select **ASCTOVST** from the **Verify** menu to convert the AST and ATR files to STM and TRC files.

For information on the syntax of the Xilinx-specific programs used, refer to your *XACT Reference Guide* and Chapter 10, "Manual Translation," in this book.

Simulating Your Design

This section describes how to simulate your design from DOS or from OrCAD/ESP. Included are instructions on how to enter the OrCAD/ESP environment, how to configure the VST environment, and how to simulate your designs. The "VST Tutorial" chapter guides the new user step-by-step through a simple simulation.

Configuring the OrCAD/VST386+ Software

Before you can simulate your design, you must configure OrCAD/VST386+ for use with your Xilinx software. You only need to configure the software once in each design directory, as these settings are stored in the vst.cfg file for the design on which you are working.

There are two ways to configure VST: automatically, using XDraft, and manually, using the OrCAD/ESP menus. To configure with XDraft, refer to "Using XDraft to Configure the OrCAD Environment" in Chapter 2. To configure VST manually, refer to your OrCAD documentation.

Before beginning your simulation, verify that you have changed the Connectivity Database extension to VST from the default extension INF, as described in the "Connectivity Database Extension" subsection of Chapter 2.

If you already ran XDraft in your current working directory and configured both the schematic tools and the simulator as explained in Chapter 2 before entering your design, you do not have to reconfigure VST.

You can simulate your design either from DOS or from XDM. In

either case, follow the simulation instructions in the OrCAD manuals. Refer to the chapter "OrCAD VST Simulation Issues" in this manual for specific information on using the Simulate program to simulate FPGA and EPLD designs.

Simulating from DOS

You can access the OrCAD software and execute all Xilinx-supplied programs directly from DOS. It is a good idea to run your programs from DOS if you run into memory problems.

To simulate from DOS, enter the following command:

```
simulate design.vst /t↵
```

Option **/t** reads the trace and stimulus files: *design*.trc and *design*.stm.

Note: To simulate, you must specify the .vst extension. Otherwise, the simulator uses the .inf extension, which is the default. If the default is used, the simulator issues an error message indicating that it could not find a symbol in X3K_LIB.

Simulating from the Graphical User Interface

If you decide to simulate your design from OrCAD/ESP, use the instructions below.

Entering the OrCAD/ESP Design Environment

To simulate your design with OrCAD/VST386+, first enter the OrCAD/ESP design environment. You can enter OrCAD/ESP either from a DOS prompt or from the XACT Design Manager (XDM). Refer to the chapter "Getting Started" for instructions.

Entering the OrCAD/VST386+ Environment

After you enter the OrCAD/ESP environment, you need to enter the VST environment. Click the left mouse button on the Digital Simulation Tools button on the ESP main screen. The Digital Simulation Tools screen appears (see Figure 6-1).

Digital Simulation Tools provides five tool sets — editors, processors, libraries, reporters, and transfers — and a set of user-definable buttons. Refer to your OrCAD documentation for information about the Digital Simulation Tools tool sets and user-definable buttons.

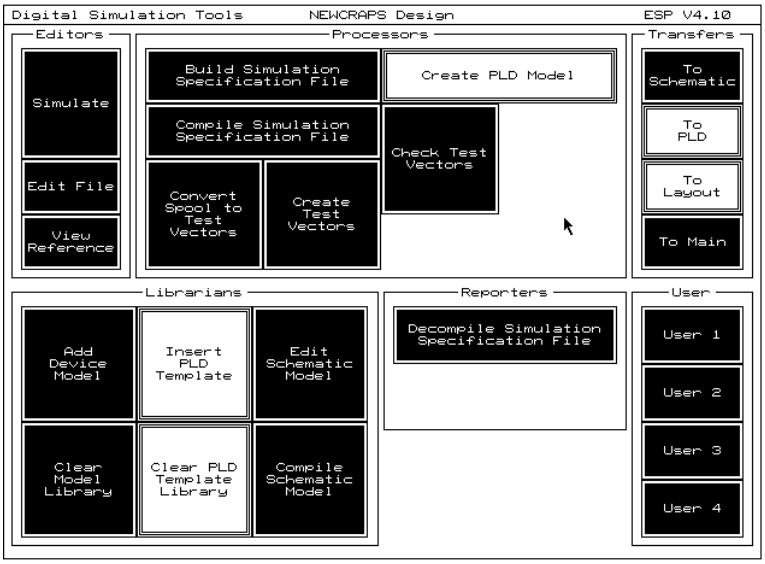


Figure 6-1 OrCAD Digital Simulation Tools Screen

Simulating a Design

To simulate a design from OrCAD/VST386+, follow these simple steps:

1. Click on the **Simulate** button; the Simulate menu appears.
2. Select **Local Configuration** from the menu. The Select Configuration menu appears.
3. Select **Configure SIMULATE**. The Configure Simulate screen appears.
4. Ensure that VST, rather than INF, is specified as the extension of the input simulation netlist file.
5. Click on the **OK** button at the top of the screen. You return to the Digital Simulation Tools screen.
6. Click on the **Simulate** button. The Simulate menu appears.
7. Click on **Execute** to invoke the Simulate program.

Follow the simulation instructions in the OrCAD manuals. Refer to the chapter, “OrCAD VST Simulation Issues,” for specific information on using Simulate to simulate FPGA and EPLD designs. Chapter 12, “VST Tutorial,” provides an example simulation session.

OrCAD Interface/ Tutorial Guide

Design Implementation

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Chapter 7

Design Implementation

This chapter discusses XMake, a Xilinx program used to translate an FPGA schematic design into a logic cell array (LCA) file, and XEMake, a Xilinx program used to translate an EPLD design into a VMH file. The chapter is structured as follows:

- *Translating Your FPGA Design* — This section explains how to translate FPGA designs into LCA and BIT files.
- *Translating Your EPLD Design* — This section describes how to translate EPLD designs into VMH and PRG files.
- *Valid File Formats* — This section lists the input files you can use to run XMake and XEMake, and the output files generated by these programs.
- *Reprocessing the Design after Minor Changes* — This section outlines the steps used to reprocess your design after minor modifications that do not affect the hierarchy.

An LCA file refers to the implemented version of an FPGA design, that is, a design that was placed and routed by the Xilinx APR core tools (XC2000, XC3000, and XC3100 designs) or PPR core tools (XC3000A/L, XC3100A, and XC4000 designs). You can download your LCA design using the MakeBits-generated bit file as input to the XChecker™ or MakePROM programs.

A VMH file refers to a database EPLD file that was created by the FITNET or FITEQN program for an XC7000 design. FITNET reads the design file in XNF format and implements it into an EPLD device. XEMake optionally creates a programming file in Intel HEX format (PRG file). You can also create a JEDEC format file (JED file) using the MakeJED program.

Before translating schematics, memory compilers, state machines, Boolean equations, or other design entry format files into an LCA file for FPGAs, you must first convert the design entry files into Xilinx Netlist Format (XNF) files. For EPLDs, only the schematic is converted to XNF. An XNF file describes each logic gate in a design, its associated pins, the connections between the gates, and any parameters you have specified on the symbols.

With the XACT Design Manager, you can translate and implement your design in one step automatically. You can also do it manually by invoking all the necessary translation programs as stand-alone processes. See Chapter 10, "Manual Translation," for a step-by-step description.

Note: The information contained in this chapter on OrCAD/SDT386+ file translations supersedes any similar information contained in the OrCAD/SDT386+ manuals.

Translating Your FPGA Design

The XMake program supports the automatic translation of design files into XNF files. XMake automatically converts a design file into a placed and routed design file using the Annotate, INET, SDT2XNF, and XNFMerge programs and the appropriate design implementation tools in succession. Use XMake to convert your FPGA design into an LCA file and a BIT file. The program can process your design successfully given a schematic file (SCH).

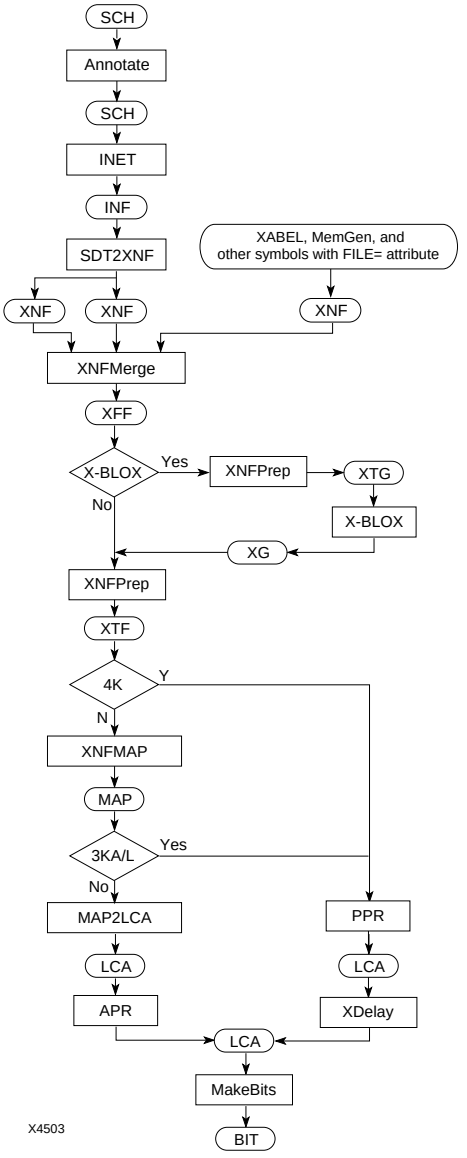


Figure 7-1 Implementing Your FPGA Design with XMake

Translating Automatically with XMake

The XMake program automatically executes the translation programs needed to convert a design into an LCA file. Even designs that contain both schematic and non-schematic modules, such as memory modules and Boolean equations, are automatically translated.

Invoking XMake

You can invoke XMake as a command line entry from the DOS prompt or from the XACT Design Manager. The steps for invoking XMake using each of these methods are presented below.

From the XACT Design Manager

If no part type is specified in the root schematic, use the mouse to select a supported part from the menu for your design. Then, perform the following sequence of commands:

1. Select **XMake** from the **Translate** menu.
2. Select any desired XMake options from the displayed menu. To process designs created with the old libraries, use option -L.
3. Select **Done**.
4. Select the top-level schematic (*toplevel*.sch) or MAK file from the displayed menu.

From the DOS Prompt

The command-line syntax for the XMake program, when invoked from the DOS prompt, is illustrated below:

```
xmake [-options] toplevel↵
```

or

```
xmake [-options] toplevel.mak↵
```

Note: This command can also be executed from the XACT Design Manager command line.

You can use the -p option to specify the intended part type. Option -L is required to process designs that reference the old libraries. Option -L also ensures that ABL2XNF and MemGen produce a version 4 XNF file, which is a file compatible with the old libraries. If

you do not specify option -L, XMake generates XNF files with new symbols from the Unified Libraries; these files clash with the old library symbols used in the rest of your design.

XMake automatically invokes the appropriate translators to convert the top-level design into an LCA file.

When a design file is used as an input file, XMake saves the translation commands into a file called *toplevel*.mak, which you can then use as an input to subsequent runs of XMake, provided that you have not modified the hierarchical structure of your design. Refer to the section "Reprocessing the Design After Minor Changes" for more information.

XMake Summary

XMake creates a routed LCA file, and, optionally, a BIT file. It first translates the design into a Xilinx Netlist Format (XNF) file to make the design compatible with the Xilinx core tools. Then, it runs the Xilinx core tools that implement the design.

Note: You can make minor changes to your design and recompile it using the original design as a guide file for any of the FPGA families. For information on this capability, see the *XACT Reference Guide*. An example is provided in the "SDT Tutorial" chapter.

XMake Command Summary: FPGA Design Implementation		
XMake performs the following steps:		
Annotate	Adds reference designators to your design symbols.	All designs
INET	Writes an OrCAD netlist format file (INF file) for each schematic.	
SDT2XNF	Translates each INF file into an XNF file.	
XNFMerge	Merges all XNF files into one flat XNF format file (XFF file). XNF files generated by Xilinx ABEL and MemGen are also merged.	
If there are X-BLOX modules, XMake performs these additional steps:		
XNFPrep	Performs DRC checking and trims unused / disabled logic. Generates an XTG file, which is a trimmed XNF format file.	XC3000A / L, XC3100A, and XC4000 designs
X-BLOX	Synthesizes the X-BLOX modules and generates an XNF format file (XG file).	
For all designs, XMake continues with the following steps:		
XNFPrep	Performs DRC checking and trims unused / disabled logic. Generates an XNF format file (XTF file).	All designs
XNFMAP	Maps the XTF file logic into logic and I / O resources of the LCA. Generates a netlist (MAP file) mapped into CLBs, IOBs, TBUFs, and clock buffers.	XC2000, XC3000 / A / L, XC3100 / A designs
MAP2LCA	Performs initial placement, translates MAP file into an unrouted LCA file.	XC2000, XC3000, and XC3100 designs
APR	Routes design, includes delays in your final LCA file.	
PPR	Generates a mapped, placed and routed LCA file.	XC3000A / L, XC3100A, and XC4000 designs
XDelay **option -dw**	Includes delays in your LCA file.	
MakeBits	Processes the LCA file and generates a BIT file.	All designs

For additional information on XMake and design implementation, refer to the *XACT Reference Guide.*

XMake Options

Some options control how MAK files are created. These options are used by programs invoked automatically by XMake. Except for the -r option, these options are only valid when you specify a design file. They do not apply if you invoke XMake with a MAK file, since the MAK file already describes which programs and options to use. Refer to your *XACT Reference Guide* for information on the XMake options.

Translating Your EPLD Design

The XEMake program supports the automatic translation of design files into XNF files. XEMake automatically converts a design file into an implemented file using the Annotate, INET, SDT2XNF, and XNFMerge programs and the appropriate design implementation tools in succession. Use XEMake to convert an EPLD design into a VMH file and optionally a HEX file. The program can process your design successfully given a schematic file (SCH).

Note: If you want to create a JEDEC file instead of a HEX file, run the MakeJED program on the VMH file.

The XEMake program automatically executes the translation programs needed to convert a design into VMH and HEX files. Even designs that contain both schematic and non-schematic modules, such as Boolean equations, can be automatically translated.

If you have existing schematics that were created with an earlier version of the XEPLD OrCAD library, you can continue to use your earlier library to process your designs. However, you must create new XNF netlists from your schematics using XEMake. XEPLD no longer accepts EDIF formatted netlist files.

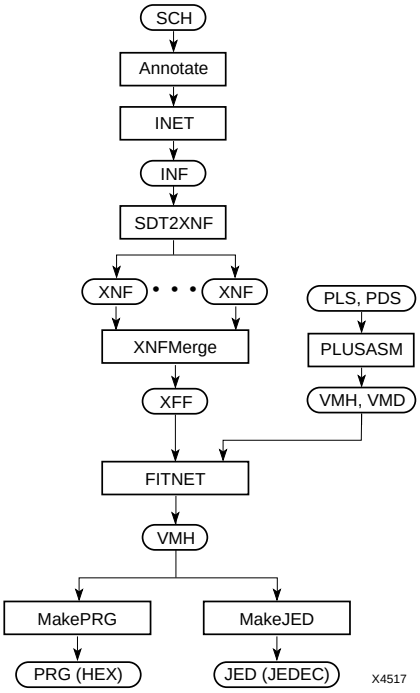


Figure 7-2 Implementing Your EPLD Design

Invoking XEMake

XEMake can be invoked as a command line entry from the DOS prompt or through the XACT Design Manager. The steps for invoking XEMake using each of these methods are presented below.

From the XACT Design Manager

If no part type is specified in the root schematic, use the mouse to select a supported part from the menu for your design. Then, perform the following sequence of commands.

1. Select **XEMAKE** from the **Translate** menu.
2. When the options menu appears, accept the default processing by pressing **Done**.

3. Select the design file you wish to implement.

   Files with the extension .sch are your schematic files.

4. Indicate the target for XEMake: either **`Make Intelhex bitmap`** or **`Make design database`**. The first option generates a design database file and an Intelhex bitmap file (VMH and HEX files). The second option generates a design database file only (VMH file). By default, XEMake generates a design database file only.

Note: To create a JEDEC file, run XEMake first. Invoke the MAKEJED program from the Verify menu to process the VMH file.

From the DOS Prompt

To run the XEMake program from DOS, use the command-line syntax shown below:

`xemake [-`*`options`*`] `*`toplevel`*` [`*`target`*`.prg]`↵

or

`xemake [-`*`options`*`] `*`toplevel`*`.mak` ↵

If you want to create an Intel HEX file, specify a target file name with a .prg extension.

You can use the -p option to specify the intended part type.

XEMake automatically invokes the appropriate translators to convert the top-level design into a VMH file.

When a design file is used as an input file, XEMake saves the translation commands into a file called *toplevel*.mak, which you can then use as an input to subsequent runs of XEMake, provided that you have not modified the hierarchical structure of your design. Refer to the section “Reprocessing the Design After Minor Changes” for more information.

XEMake Summary

The following table summarizes the steps performed by XEMake when implementing an EPLD design:

XEMake Command Summary: EPLD Design Implementation		
XEMake performs the following steps:		
Annotate	Adds reference designators to your design symbols.	All designs
INET	Writes an OrCAD netlist format file (INF file) for each schematic.	
SDT2XNF	Translates each INF file into an XNF file.	
XNFMerge	Merges all XNF files into one flat XNF format file (XFF file).	
FITNET	Generates a VMH file containing the implemented EPLD design.	
MAKEPRG (optional)	Generates a programming list file (PRG) in HEX format.	

For information on XEMake options, refer to the *XEPLD Reference Guide*.

Valid File Formats

XMake and XEMake can use either of two types of input files, and they produce four major output files in addition to many intermediate files. These input and output files are described in detail in the following subsections.

Input Files

When translating a design from OrCAD/SDT386+, XMake and XEMake require one of two possible file formats as input. These files include the SCH file or an XMake/XEMake-generated MAK file.

SCH File

- *toplevel*.sch is the FPGA or EPLD schematic design file created with the Draft schematic editor.

MAK File

- *toplevel*.mak is the file created by XMake or XEMake detailing the sequence of implementation programs to be executed.

  The sequence of programs run by XMake or XEMake varies according to the structure and hierarchy of your design. Each program uses the program options defined in the XDM profile file (xdm.pro). It then records the program options it uses and the steps it performs in a MAK file. After creating the MAK file, the program uses the MAK file as an input file. You can perform faster compilation using the MAK file as an input, rather than using an SCH file, provided the input-design hierarchy has not changed since the MAK file was created.

Output Files

Both XMake and XEMake perform a complete implementation of the design. Output files from each program are discussed in this section.

Report File

XMake generates several report files, the most comprehensive of which is *toplevel*.out.

- *toplevel*.out is an ASCII text file containing the screen outputs of the various translation programs invoked by XMake.

XEMake generates several report files. Refer to the *XEPLD Reference Guide* for more information.

Note: Since the OUT file contains all warnings and error messages that occur during design processing, you should always review the OUT file after XMake has run to ensure that your design is error-free.

Design File

Both XMake and XEMake generate a completed design file.

- *toplevel*.lca is the name of the design file generated by XMake.

  XMake creates an LCA file that is mapped, placed, and routed by either the APR (XC2000, XC3000, and XC3100 designs) or PPR (XC3000A/L, XC3100A, and XC4000 designs) program, unless it is disabled with the XMake -n option.

- *toplevel*.vmh is the name of the design file generated by XEMake.

  XEMake creates a VMH file, which is a partitioned file fitted by the FITNET program. Included are the PLD equations integrated by FITNET.

Programming File

Both XMake and XEMake generate a programming file.

- *toplevel*.bit is the name of the bitstream file generated by XMake.

  For all XC3000A/L, XC3100A, and XC4000 designs that PPR successfully routes, or XC2000, XC3000, and XC3100 designs that APR successfully routes, XMake creates a bitstream that can be downloaded to an FPGA device. The configuration options for the bitstream generator are determined by the options set in the XACT Design Manager profile file, xdm.pro.

- *toplevel*.prg is the name of the bitmap file generated by XEMake.

  XEMake creates a bitmap file that can be programmed into an EPLD device through an EPLD programmer. The PRG file is in HEX format.

MAK File

XMake and XEMake both create a MAK file.

- *toplevel*.mak

  When you run XMake or XEMake on a Draft schematic file, the program creates a text MAK file (*toplevel*.mak) that documents how each design submodule is processed, including which options were used by the translation programs. The information in the MAK file serves three purposes:

  - It is used as a script for XMake/XEMake when first translating the design.
  - It documents the commands used to translate the design into an LCA file or a VMH file. By examining the MAK file, you can determine exactly which programs and options XMake/XEMake ran on each design submodule.
  - It can be used as an input file the next time XMake/XEMake is invoked on the same design. Discerning which schematic files

have been updated, XMake/XEMake only invokes the necessary processes to obtain the final result using minimal processing time.

Reprocessing the Design After Minor Changes

Your implementation program, XMake or XEMake, performs the following operations.

- It creates a MAK file that describes which programs and options to use when processing the design.
- It translates the design to an XNF file and an LCA or VMH file, using the command sequence defined in the MAK file.
- It creates a configuration bitstream or program bitmap file.

To reprocess the design after making logic changes, you can invoke XMake/XEMake using the MAK file that these programs created in the process outlined above, provided that the changes do not affect the hierarchy of the design.

For example, use your top-level design file called "mydesign.sch" and follow these steps to reprocess your design:

1. Select **XMake** or **XEMake** from the **Translate** menu depending on the type of device you are designing.
2. Select **Done** to indicate that you do not want to change any program options.
3. Select **mydesign.mak** from the design menu, that is, instruct XMake/XEMake to use the MAK file to decide which programs and options to use when processing your design.

Using an existing MAK file has two benefits:

- Design processing is faster, because XMake/XEMake does not have to process the entire design; it processes only those files that have changed since the design was last processed.
- You can edit the MAK file to modify program options, then reprocess the design to reflect these changes.

OrCAD Interface/ Tutorial Guide

Timing Simulation

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Chapter 8

Timing Simulation

This chapter shows you how to create files so that you can use Simulate, the OrCAD/VST386+ simulation program, to simulate an FPGA or EPLD design. Included are instructions for creating files for timing simulation. The process is the same for FPGA and EPLD designs unless otherwise indicated. It is assumed that you have already created a routed LCA file using XMake or a VMH file using XEMake. (See Chapter 7, "Design Implementation.")

For instructions on how to use Simulate to simulate your designs, refer to the section "Simulating Your Design" in this chapter.

Note: You can specify trace and stimulus information on your schematic or using the simulation tool in OrCAD. Refer to your OrCAD documentation for information on how to specify trace and stimulus information from OrCAD/VST386+, or see the "VST Tutorial" chapter of this manual.

This chapter contains the following sections:

- *Creating a Timing Simulation Netlist* — This section tells you how to create a timing simulation netlist for FPGA, EPLD, and EPLD behavioral designs.
- *XSimMake Summaries* — This section summarizes the steps performed by XSimMake, the tool used to create a simulation netlist.
- *Simulating Your Design* — This section tells you how to simulate your design.

Creating a Timing Simulation Netlist

Use XSimMake to invoke the required translation programs automatically and convert the top-level design into VST, AST, and ATR files. The whole translation process is automatically completed for you in a single step.

You can invoke XSimMake as a command line entry from the DOS prompt or through the XACT Design Manager. The steps for invoking XSimMake using the two methods are presented below.

From the XACT Design Manager

If there is no part type specified from the root schematic, use the mouse to select a supported part from the menu for your design.

To create a timing netlist file, follow these simple steps:

1. Invoke **XSIMMAKE** from the XDM **Verify** Menu to bring up the XSimMake options menu.
2. Select the **-F** option.
3. Choose **OrCAD_Fpga_Timing** for FPGA designs from the list of flows. Choose **OrCAD_Epld_Timing** for EPLD designs.
4. Select **Done**.
5. Select the file name for the design that you wish to simulate.

These steps create a timing simulation netlist. For information on the steps performed by XSimMake, refer to the next section, "XSimMake Summaries."

From a DOS Prompt

To run the XSimMake program from DOS, use the command line syntax shown below.

xsimmake -f orcad_fpga_timing [-*options*] *designname* ↵

or

xsimmake -f oft [-*options*] *designname* ↵

To process an EPLD design, use the following command:

xsimmake -f orcad_epld_timing [-*options*] *designname* ↵

or

xsimmake -f oet [-*options*] *designname* ↵

Note: This command can also be executed from the XACT Design Manager command line.

XSimMake Options

The following options are available for XSimMake:

-f — Flow Name

This option enables you to select the flow to run (functional or timing simulation flow for FPGA or EPLD designs).

-h — Help

This option provides you with help on XSimMake commands.

-l — List Flows

This option lists all available flows. Possible flows include FPGA timing, EPLD functional, and so forth.

-o — Old Libraries

This option tells XSimMake to use the old (pre-unified) libraries. Use this option only if your design was created using the old libraries.

-p — Parttype

This option enables you to select a part type. The designated part type overrides any part type already specified in your design.

-r — Force Re-execution

This option forces re-execution of all programs in the flow whether or not the design has changed.

-v — Verbose

This option enables you to view the progress of the program.

Converting Trace and Stimulus Files

Run ASCTOVST from the Verify menu to convert the AST and ATR files to STM and TRC files. If AST and ATR files already exist in your project directory, they are not overwritten. To overwrite existing AST and ATR files, you must select the -w option for XNF2VST in your XDM Profile menu. AST and ATR files are not generated by the EPLD Timing simulation flow; you must run the EPLD Functional simulation flow to extract them from your schematic.

EPLD Behavioral Designs

If you do not use OrCAD / SDT386+ to enter your design, you can enter your design in a behavioral format. To generate a timing simulation netlist for a behavioral design, perform the steps outlined below. A behavioral design refers to a design entirely represented by an equation file instead of a schematic. Refer to the *XEPLD Design Guide* for more information on behavioral designs.

1. Use the **Translate → PALCONVT** or **FITTER → FITEQN** command to integrate your design.
2. Select **Verify → XSIMMAKE** from the XDM menus.
3. Choose **OrCAD_Epld_Timing** as the flow.
4. Enter OrCAD and perform simulation in VST using the standard procedure.

XSimMake Summaries

With XSimMake, the process of translating your implementation file into a VST file is automatically completed in a single step.

FPGA Designs

The following table summarizes all the steps that are performed by XSimMake for an FPGA design:

XSimMake Command Summary: FPGA Timing Flow		
XSimMake performs the following steps:		
LCA2XNF	Translates the LCA file into an XNF file that includes delays.	All designs
XNFBA	Combines pre-route XFF or XG (X-BLOX) file and post-route XNF file into a new file so that the symbol and signal names correspond to the names in the original schematics.	
XNF2VST/XNF2INF	Generates VST, DBA, and NRF files. Also generates AST and ATR files if you specified stimulus and trace information in your schematic and the files do not already exist.	

EPLD Designs

The following table summarizes the steps XSimMake performs for an EPLD design:

XSimMake Command Summary: EPLD Timing Flow		
XSimMake performs the following steps:		
VMH2XNF	Translates the VMH file into an XNF file that includes delays.	All designs
XNF2VST	Generates VST, DBA, and NRF files.	

Simulating Your Design

This section describes how to simulate your design from DOS or from OrCAD/ESP. Included are instructions on how to enter the OrCAD/ESP environment, how to configure the VST environment, and how to simulate your designs. The "VST Tutorial" chapter guides the new user step-by-step through a simple simulation.

Configuring the OrCAD/VST386+ Software

Before you can simulate your design, you must configure OrCAD/VST386+ for use with your Xilinx software. You only need to configure the software once in each design directory, as these settings are stored in the vst.cfg file for the design on which you are working.

There are two ways to configure VST: automatically, using XDraft, and manually, using the OrCAD/ESP menus. To configure with XDraft, refer to "Using XDraft to Configure the OrCAD Environment" in Chapter 2. To configure VST manually, refer to your OrCAD documentation.

Before beginning your simulation, verify that you have changed the Connectivity Database extension to VST from the default extension INF, as described in the "Connectivity Database Extension" subsection of Chapter 2.

If you already ran XDraft in your current design directory and configured both the schematic tools and the simulator as explained in Chapter 2 before entering your design, you do not have to reconfigure VST.

You can simulate your design either from DOS or from XDM. In either case, follow the simulation instructions in the OrCAD manuals. Refer to the chapter "OrCAD VST Simulation Issues" in this manual for specific information on using the Simulate program to simulate FPGA and EPLD designs.

Simulating from DOS

You can access the OrCAD software and execute all Xilinx-supplied programs directly from DOS. It is a good idea to run your programs from DOS if you run into memory problems.

From DOS, enter the following command:

```
simulate design.vst /t /a↵
```

Option /t reads the trace and stimulus files: *design*.trc and *design*.stm.

Option /a specifies full-timing simulation.

Note: To simulate, you must specify the .vst extension. Otherwise, the simulator uses the .inf extension, which is the default. If the default is used, the simulator issues an error message indicating that it could not find a symbol in X3K_LIB.

Simulating from the Graphical User Interface

If you decide to simulate your design from OrCAD/ESP, use the instructions below.

Entering the OrCAD/ESP Design Environment

To simulate your FPGA design with OrCAD/VST386+, first enter the OrCAD/ESP design environment. You can enter OrCAD/ESP either from a DOS prompt or from the XACT Design Manager (XDM). Refer to the chapter "Getting Started" for instructions.

Entering the OrCAD/VST386+ Environment

After you enter the OrCAD/ESP environment, you need to enter the VST environment. Click the left mouse button on the Digital Simulation Tools button on the ESP main screen. The Digital Simulation Tools screen appears (see Figure 8-1).

Digital Simulation Tools provides five tool sets — editors, processors, libraries, reporters, and transfers — and a set of user-definable buttons. Refer to your OrCAD documentation for information about the Digital Simulation Tools tool sets and user-definable buttons.

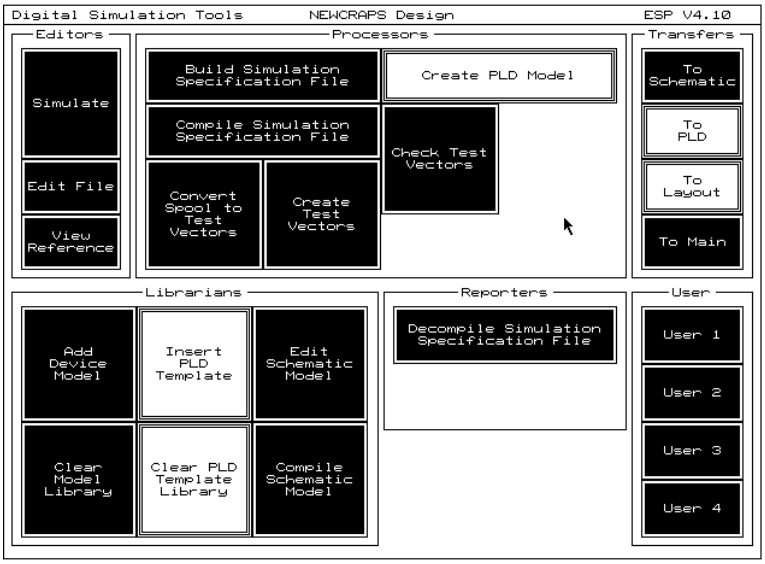


Figure 8-1 OrCAD Digital Simulation Tools Screen

Simulating a Design

To simulate a design from OrCAD/VST386+, follow these simple steps:

1. Click on the **Simulate** button; the Simulate menu appears.
2. Select **Local Configuration** from the menu. The Select Configuration menu appears.
3. Select **Configure SIMULATE**. The Configure Simulate screen appears.
4. Ensure that VST, rather than INF, is specified as the extension of the input simulation netlist file.
5. Enable the **Use Delay Annotation** option to include delays in your simulation.
6. Click on the **OK** button at the top of the screen. You return to the Digital Simulation Tools screen.
7. Click on the **Simulate** button. The Simulate menu appears.

8. Click on **Execute** to invoke the Simulate program.

Follow the simulation instructions in the OrCAD manuals. Refer to the chapter, "OrCAD VST Simulation Issues," for specific information on using Simulate to simulate your designs. Chapter 12, "VST Tutorial" provides an example simulation session.

OrCAD Interface/ Tutorial Guide

OrCAD VST Simulation Issues

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Chapter 9

OrCAD VST Simulation Issues

FPGA Devices

This section contains important information that you should consider when using the Simulate program to simulate your FPGA designs.

Unconnected Control Pins

When left unconnected, control pins such as CE, CLR, PRE, and R on Xilinx library macros can cause unknown output values during functional simulation. The solution is to tie unused CE pins to VCC, and unused CLR, PRE, and R pins to the inactive values. The logic is later trimmed during the implementation process, so the extra VCC and GND nets do not use any additional routing resources, and this problem does not occur during timing simulation.

If there are unconnected control pins in the input netlist, XNF2VST issues a warning message.

Global Reset and 3-State Signals

By default, the global reset signals are active at time zero and the global 3-state signal is inactive. Therefore, all flip-flops in your design remain in the reset state until you deactivate the reset signal. For XC2000, XC3000, and XC3100 designs, drive GR High after 100 nanoseconds. For XC4000 designs, drive GSR Low.

There is no pulse-width check on the global reset and 3-state signals.

Simulation Time Units

To simulate sub-nanosecond delays, the interface to VST passes data that is scaled by one tenth. Consequently, VST uses .1 ns units. For example, to simulate for 10 ns, select Run Simulation and type 100↵.

Using Traces and Stimuli

Trace and stimulus symbols can be placed in your design at any level of the schematic. Refer to the "SDT Tutorial" chapter for an example of how to enter trace and stimulus symbols. If you follow the functional simulation design flow, the stimulus and trace information associated with these symbols is usable when simulating with OrCAD/VST386+. If you follow the timing simulation design flow, schematic-level trace and stimulus information is only usable if you back-annotate your net names using the XNFBA back-annotation program. If you use XSimMake, the timing simulation flow automatically runs XNFBA.

Note: At this time, stimulus and trace elements on buses are not supported.

Simulating High-Impedance Inputs

OrCAD "test vector" input format is required when simulating bidirectional signals. If a stimulus is specified directly in the Stimulus Editor, VST treats high-impedance inputs incorrectly.

Pulse-Widths Smaller than the Routing Delay

The simulator invalidates (displays as undefined) pulses that have a width less than the net delay time. In the actual FPGA device, however, pulses with widths less than the net delay time are not always absorbed.

No Weak-keeper

Weak-keeper on bus lines is not implemented. The bus goes to High-Impedance when all sources are enabled to 3-state.

Simulating the OSC, OSC4, and GXTL Oscillators

These oscillators cannot be simulated by OrCAD/VST386+. When using these signals, you must create the stimulus on the oscillator output using the Stimulus Editor in the Simulate program or specify the stimulus in the schematic.

Hold Violations

During simulation, you may encounter hold violations on flip-flops used in your design. According to *The Programmable Logic Data Book*, there is no hold time for the flip-flops in a CLB unless the DIN pin is used. The lack of hold-time requirements is reflected in the modeling in the routed XNF file similar to the following.

```
SYM, Q1.QX, DFF, INIT=R
  PIN, Q, 0, Q1.QX, 5.0
  PIN, D, I, Q1.F
  PIN, C, I, CAB, 6.1
  PIN, CE, I, BCE, 0
  PULSE, C, +, 4.0
  SETUP, D, C, +, 0.0, 4.5
END
```

The SETUP line indicates that the input pin D is clocked by C, which is positive-edge-triggered, and has a setup time of 0 ns and a hold time of 4.5 ns. This XNF file is for an XC4000 device with a speed grade of 5. If you look up the setup time for a CLB flip-flop in *The Programmable Logic Data Book*, the setup time is 4.5 ns, and the hold time is 0 ns.

The setup and hold time specifications in the Xilinx data sheets are based on a comparison of the CLB input signal and the CLB clock input, which is also the clock input of the flip-flop. This comparison is illustrated as (a) in Figure 9-1.

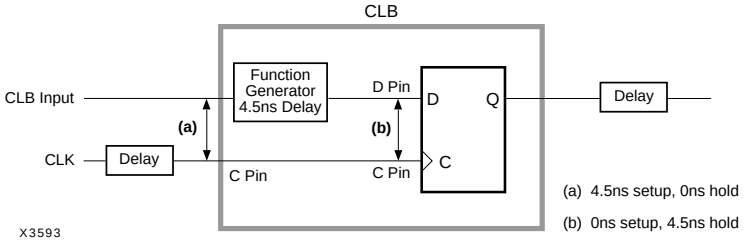


Figure 9-1 Hold Violation Example Circuit

Before OrCAD reads your netlist, the CLB is broken down into gates and flip-flops. OrCAD makes its setup and hold-time checks by comparing the D input of the flip-flop and the clock input, shown as comparison (b) in Figure 9-2.

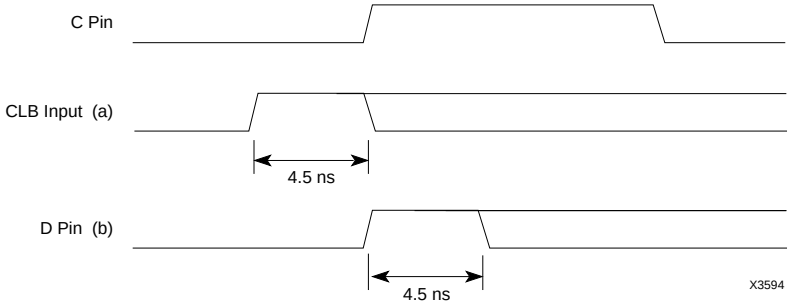


Figure 9-2 Hold Violation Waveform Output

Consequently, what appears to be a setup violation from the viewpoint of the CLB specification may be reported by OrCAD as a hold violation. Since the sum of the setup and hold-time requirements is the same in either case, whether the violation is reported as a setup or hold-time violation is immaterial.

Simulating Large ROMs in XC4000 Devices

If your XC4000 design requires more than 100 ROM16X1 or 50 ROM32X1 symbols, you need to add information from the xmem2.dsf file to the model.dat and model.ndx files. Add the information by using the Add Device Model button in the Librarians section of the Digital Simulation Tools screen.

If you use the Clear Model Library button to clear model data from the model.dat and model.ndx files, you might need to reload the VST interface. You need to reload the VST interface because there is no source (DSF) file available for most FPGA / VST models.

EPLD Devices

Using PRLD for Initialization

The PRLD (preload) signal is an input to your design that does not appear on your schematic; it is included in the models of registered components and is automatically added to both the functional and timing models by XSimMake. You must include the PRLD signal in your OrCAD simulation stimulus file (STM) to ensure that the registers in your design are simulated properly. However, it is not necessary to display this signal in your OrCAD trace file (TRC).

Before applying simulation stimuli, you must initialize the device by pulsing PRLD High for at least 1 time unit (0.1ns). Before initialization, all registers are in an unknown state (U) which usually prevents any meaningful simulation results. PRLD simulates the Master Reset signal (or power-on-reset) of the EPLD device and forces all registers to a predefined state.

All input signals to the device should be set to a known logic state before PRLD is returned Low, otherwise some internal nodes may become trapped in an unknown state. You must return PRLD to a Low state before the design will respond properly to input stimulation.

For functional simulation, all registered components initialize to the state defined in the *XACT Libraries Guide*. During implementation, the FITNET program might alter the initial states of register to take optimal advantage of device resources unless you inhibit Preload optimization. Timing simulation will exhibit the actual register

preload values implemented by the software. You can control the preset state of XEPLD registers in PLD components, so they are forced either High or Low, by using the "*output*.PRLD" equation in PLUSASM. See the *XEPLD Reference Manual* for more information on the PRLD extension.

You can use the PRELOAD_OPT global attribute to determine whether the preload value can be changed for parts of the design to allow logic to fit as efficiently as possible into device resources (ON), or the logic is mapped to device resources according to the established preload values (OFF). PRLD equations in your PLD components are only effective if PRELOAD_OPT is turned OFF. For more information, see Chapter 5, "EPLD Design Issues."

Under some conditions, the first simulation cycle after PRLD is brought Low produces setup and hold violations. This is due to the asynchronous nature of the PRLD signal. The resulting warnings are usually not indicative of a circuit problem.

You should analyze your design for any potential initial-state problems that could result when the device comes out of the power-on-reset or the external Master Reset (MR). For example, if the device recovers from its reset cycle and becomes operational coincident with the rising edge of a free-running clock input, not all registers in the device might respond to this first clock cycle. As a result, an invalid internal state might occur. Such situations are not detectable during simulation.

See the XEPLD Tutorial for an example of how to initialize the device during simulation.

3-State Outputs

When performing functional simulation, remember that high-impedance states (Z) only exist on the wire connected to a 3-state symbol output. For example, if you connect a BUFE 3-state buffer symbol to an OBUF, which is valid in EPLD designs, the output of the OBUF becomes undefined ("U") when the BUFE is disabled, as shown in Figure 9-3. This undefined output occurs because the OrCAD model used to represent the OBUF symbol cannot pass a High-Z signal through it. When such a design is actually implemented, the EPLD device pin goes to High-Z when the BUFE is

disabled, and timing simulation accurately demonstrates this behavior.

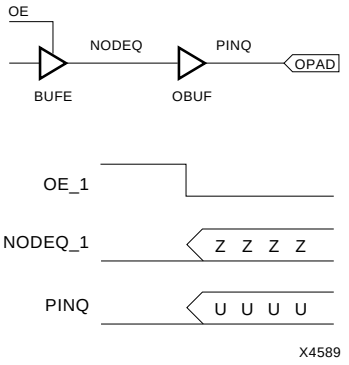


Figure 9-3 Trace Waveforms of 3-State Signal Passed Through OBUF

OrCAD Interface/ Tutorial Guide

Manual Translation

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Chapter 10

Manual Translation

If you need to execute manually as stand-alone processes the programs executed automatically by the XMake and XSimMake programs, follow the instructions in this chapter. This information is also useful if you run into problems with XMake, XEMake, and XSimMake because it explains the whole translation process. This chapter details each type of translation and is structured as follows:

- *Creating an XNF File* tells you how to create an XNF file. Use the XNF file as a starting point for implementing your design and creating simulation files. The Annotate, INET, SDT2XNF, and XNFMerge programs are described in detail in this section.
- *Creating Functional Simulation Files* tells you how to generate a functional simulation netlist (VST file) using an XNF file for FPGA designs. You must use XSimMake for EPLDs.
- *Creating Implemented Design File*s outlines the steps for creating a routed LCA or VMH file and a downloadable BIT or PRG file.
- *Creating Timing Simulation Files* describes how to generate timing simulation input files (VST and DBA files) using an implemented LCA or VMH file.
- *Translation Programs for Simulation* details the program options, syntax, and files needed to use XNF2VST and ASCTOVST.

The following figure shows the different types of files created in each translation sequence.

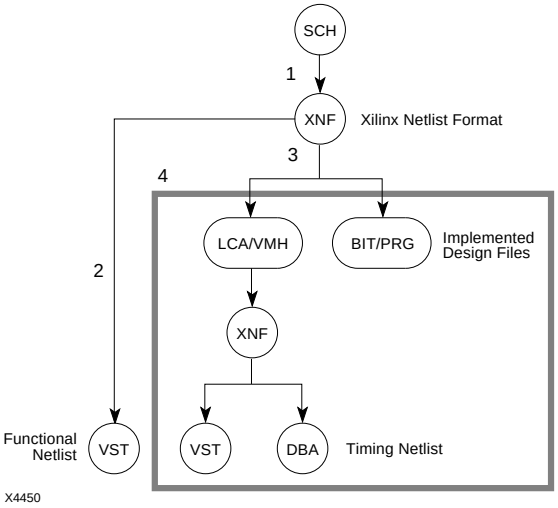


Figure 10-1 Four Types of Manual Translation

Although the objective of this chapter is to provide the syntax used to execute each translation program from DOS, you can use the same program and option information to execute the programs from the XDM menus. To execute a program from XDM, simply select the program from XDM and enable the options referenced in the command-line syntax using the left mouse button.

The programs used to generate the different types of XNF files, such as XFF, XG, and XTF files, are located under the XDM Translate menu. Place and route programs are located under the XDM PlaceRoute menu. The programs used to generate simulation files are located under the XDM Verify menu.

Creating an XNF File (SCH → XNF)

You must translate your design into an XNF file before you can create an implemented design file or a simulation file. The Annotate, INET, SDT2XNF, and XNFMerge programs are used sequentially to create a flattened XNF format file (XFF) from your Draft schematic file. (See Figure 10-2.)

- Annotate — The Annotate program assigns or updates the reference designators on all library symbols used in your design. The original schematic files are renamed as backup (BAK) files. The annotated file is then saved as the schematic (SCH) file. No options are required by the Annotate program.
- INET — The INET program creates an INF file (OrCAD netlist format) for each schematic in the design. No options are required by the INET program.
- SDT2XNF — This program uses the INF files created by INET as input files, and converts them into corresponding XNF files.
- XNFMerge — XNFMerge merges all XNF files for the design into a single top-level XNF format (XFF) file. XNF files for blocks without schematics, such as files created by Xilinx ABEL or MemGen, are also merged.

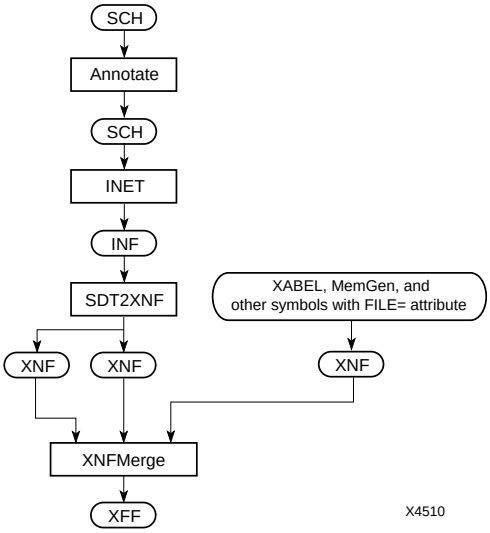


Figure 10-2 Creating XNF Format Files

Annotate Program

Annotate is an OrCAD program that reads the drawing file, assigns a unique reference designator to any unassigned component, and then saves the result to the original drawing. Use .sch as your drawing file extension. A backup of the original drawing with a .bak extension is also created. Typically, no command-line options are required.

Syntax

To execute the Annotate program on hierarchical or single-level designs, use the following syntax:

annotate ***toplevel*.sch [/*options*]**↵

The file name *toplevel*.sch is the OrCAD schematic file name submitted to the system. It represents the top-level schematic file in a hierarchical design. Structured designs and user-created macros use the same syntax.

Note: You must specify the .sch extension.

Options

You can modify the operation of the Annotate program using the following options.

/l — Last Reference Designators

This option creates a report listing the last reference designators assigned by Annotate. If you do not specify a destination file, the report is placed in a file called *toplevel*.end.

/o — One Sheet

This option treats the selected file as a single sheet.

/q — Quiet Mode

This option runs the program in quiet mode without echoing the tracking information on the screen.

/r — Remove Annotations

This option removes all annotations on the schematic. All reference designators are set to “U?”.

/s — Preserve Sheet Number

This option preserves the sheet number. If this switch is not set, the sheet numbers are changed to reflect the current sheet in the design.

/u —Change References Unconditionally

This option changes references unconditionally. Use this option only if there are multiple components that contain the same reference on the same sheet.

/z — Ignore Warning Messages

This option causes warning messages to be ignored.

INET Program

INET is an OrCAD program that reads the root schematic and any other related schematics, and converts them into individual netlist (INF) files.

Syntax

To execute the INET program on a hierarchical or single-level structure design, use the following syntax:

inet *toplevel*.sch [*/options*]↵

The file name *toplevel*.sch is the OrCAD schematic file name submitted to the system. It represents the top-level schematic file in a hierarchical design. Use the same syntax for flat-structured designs and user-defined macros.

Note: You must specify the .sch extension.

Options

You can modify the operation of the INET program using the following options:

/g — Check for Objects off the Grid

This option checks the worksheet for parts, sheets, labels, module ports, and power objects placed off the grid. A report is placed in a GRD file.

/l — Report Labels and Module Ports

This option reports all connected labels and module ports. The report is placed in a LAB file.

/n — Prevent INF File Rebuild

This option prevents the INF files from being rebuilt. It can be used in conjunction with the -g, -l, -u, and -w options.

/q — Quiet Mode

This option causes the program to run in quiet mode without echoing tracking information on the screen.

/t — Rebuild Database

This option causes the entire database to be rebuilt. It recreates all connected databases and all connecting database files.

/u — Unconnected Wires and Pins

This option reports all unconnected wires and pins. The report is placed in an NC file. This option also checks for pins, module ports, and power objects that are overlapping.

/w — Electrical Rules Check

This option runs an electrical rules check on all files that are netlisted. If a destination is supplied, then the output is placed in the indicated file; otherwise, standard out is used. Module ports are checked for correctness after all incremental netlisting and electrical rules checking is complete.

Note: If the /w option is used and INET issues severe warning messages, INET might not generate an INF file. To force the generation of an INF file with the /w option in use, turn on option /z, Ignore Warning Messages.

/z — Ignore Warning Messages

This option causes the program to exit normally and ignore all warning messages.

SDT2XNF Program

The SDT2XNF program invokes INF2XNF to convert the INET-generated INF file into an XNF file (*filename*.xnf). After creating INF files for each schematic file in your design, execute SDT2XNF on your top-level schematic. The SDT2XNF program follows the hierarchical structure of your design. As a result, any INF files representing drawing files beneath the top-level design file, including Xilinx macros, are also translated into individual XNF files.

Under some circumstances, SDT2XNF does not incorporate lower-level INF files. If, for example, FILE= is used for a selected module, the INF file for that module is not read.

Syntax

To execute the SDT2XNF program from the command line, use the following syntax:

sdt2xnf ***inffile*[*.inf*] [*xnffile*[*.xnf*]][*-options*]**↵**

- Input File

  inffile.inf

  INET generates this file from your schematic file. The *inffile* file must be your top-level schematic file. The INF extension is optional.

- Output File

  xnffile.xnf

  This is the XNF file that corresponds to the INF input file. The XNF extension is optional. If no output file name is specified, SDT2XNF uses the input design name with the XNF extension.

Options

You can modify the operation of the SDT2XNF program using the following options:

-d — Output Directory for XNF Files

The -d option specifies in which directory the XNF files are placed. If you do not specify this argument, it is set to the current directory.

-m — Macro Option

When the -m option is specified, SDT2XNF reads all symbols for which an INF file exists as macros and generates an XNF file for each of these symbols.

-p — Part Type

The -p option is used to specify the FPGA or EPLD part type used in the translation process. Specify this argument to set or override the part type in the schematic.

The -p option can also be used to specify the speed grade. The XC3000A/L, XC3100/A, and XC4000 family speed grades are of the form –6 and –5, which represent CLB combinatorial block delays of 6 and 5 ns, respectively. The XC3000 and XC2000 family speed grades are -70, –100, and –125, which represent the flip-flop toggle frequencies (MHz).

The following examples illustrate 125 MHz and 5 ns speed grade specifications for XC2000/XC3000 and XC4000 devices, respectively:

```
... -p 3020pc68–125
... -p 4005pg156–5
```

-s — Search Path for Xilinx-Defined INF Files

The -s option specifies the path used by SDT2XNF to search for Xilinx-defined INF files that include the Xilinx soft macros and smart macros. If you do not specify this option, the path is set to the appropriate Xilinx-library path.

-u — Search Path for User-Defined INF Files

The -u option specifies the path used by SDT2XNF to search for user-defined INF files that include the child-sheets and link-sheets. If you do not specify this argument, it is set to the current directory.

XNFMerge Program

You must merge the XNF files into a single XNF-format file (XFF file) representing the entire design using the XNFMerge program.

XNFMerge merges all of the XNF files created by SDT2XNF, as well as the XNF files for blocks without schematics, such as Xilinx ABEL or MemGen blocks. The output of XNFMerge is a single XNF format file with an XFF extension.

Syntax

To execute the XNFMerge program from the command line, use the following syntax:

xnfmerge [-*options*] *input_file*[.xnf][*output_file*[.xff]]↵

Options

You can modify the operation of the XNFMerge program using the following options:

-a — Abbreviate Report

This option abbreviates the file-reading report.

-d — Directory

This option uses the XNF files in the directory that you specify.

-f — Hierarchy Information

This option specifies that you do not wish to maintain hierarchy information in your XNF file.

-i — Ignore RLOC

This option ignores RLOC-related information.

-o — Merge Report

This option is used to send the merge report to the file that you specify. The MRG extension is used for this file.

-p — Partname

This option is used to specify the part type that you want in the output XFF file. Do not include the XC prefix when specifying the part type from the command line.

-q — Suppress Messages

This option suppresses messages about unresolved symbols. This is the default for XC7000 parts due to the nature of the EPLD library.

Creating Functional Simulation Files (XNF → VST)

This section explains how to execute as stand-alone processes each program normally executed by XSimMake as it translates your XNF file into a functional simulation netlist for FPGA designs. For EPLD designs, you must use XSimMake to prepare the functional simulation netlist file. XSimMake creates special XNF files for EPLD library primitives in a separate subdirectory, "func," which must not be mixed with the XNF files used for design entry.

A file used for functional simulation does not require any delay information, as the functional netlist file is only used to verify the logic in your design. The translation of the XNF format file into a functional netlist is achieved by the Xilinx program XNF2VST.

To create a functional simulation file in the OrCAD design flow, use one of the following files:

- An XFF file created by SDT2XNF and XNFMerge. Designs with Xilinx ABEL blocks can be simulated from the XFF file.
- An XG file, created by running XNFPrep with the drc_only=true option and X-BLOX, if your design contains X-BLOX modules.
- A placed and routed LCA file produced by XMake. If XMake is not used, this file is created by either APR (XC2000, XC3000, and XC3100 designs) or PPR (XC3000A / L, XC3100A, and XC4000 designs). You must use an LCA file as a simulation starting-point if your FPGA design contains CLB or IOB primitives.

Design Flows

This section discusses the functional simulation flows used for FPGA designs.

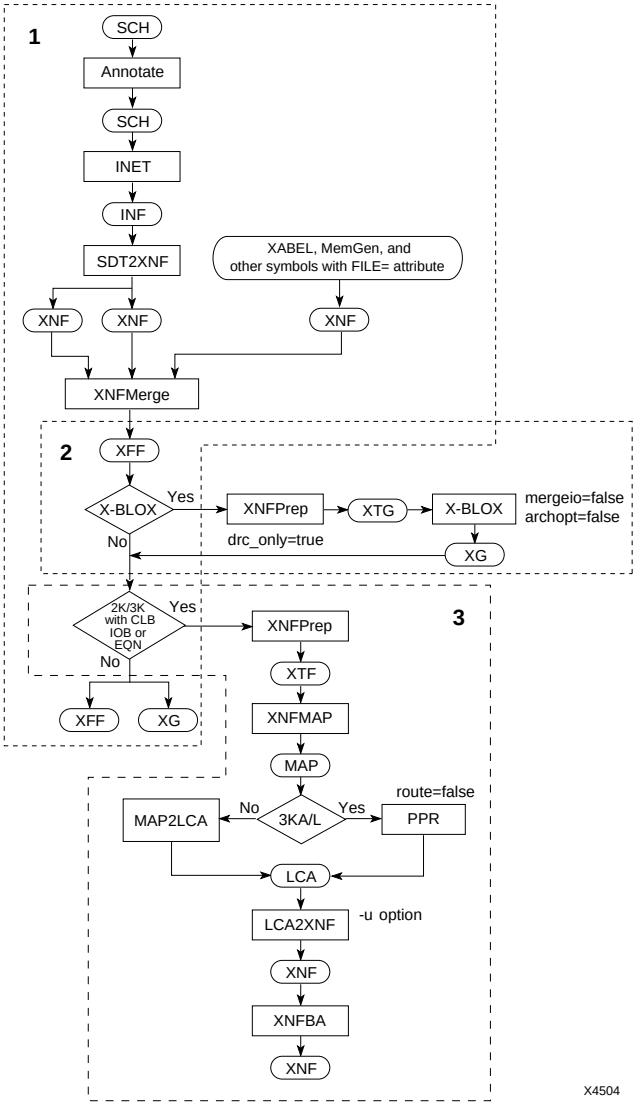


Figure 10-3 Creating XNF Files for Functional Simulation (FPGA)

FPGA Design Flow

Before using the translator to generate the actual simulation file, you must ensure that all XNF files generated from other HDL languages are merged into one flat XNF format file (XFF file) with XNFMerge; all X-BLOX modules are processed and all IOB and CLB designs are mapped with XNFPrep, XNFMAP, and MAP2LCA.

Figure 10-3 illustrates the different types of XNF format files (XFF, XG, and XNF) that are created depending on the type of FPGA modules used in the design: logical primitives only, X-BLOX modules, or IOB and CLB primitives.

- Area 1 shows the flow used for designs containing only logical primitives and Xilinx macros.
- Area 2 shows the additional steps necessary to process X-BLOX modules.
- Area 3 shows the additional steps necessary to process XC2000, XC3000, and XC3100 designs that include IOB and CLB primitives. In this last case, the XNF file must be mapped but need not be routed.

Use the final XNF format file to run the simulation translator (XNF2VST). In the case of designs with IOB and CLB primitives, you can only use the final XNF file output by LCA2XNF and by XNFBA for functional simulation.

Note: Only use XSimMake to generate a functional netlist file for EPLD devices. Refer to the chapter, "Functional Simulation," for more information.

Translating XFF Files Created with SDT2XNF and XNFMerge

Follow the step below to convert a file created with SDT2XNF and XNFMerge into a file used for FPGA functional simulation.

1. Execute the XNF2VST program from the command line using the following syntax:

   `xnf2vst` ***filename*****`[.xff] -u`**↵

XNF2VST creates all files necessary to perform functional simulation on your design. The -u option selects unit-delay translation; any

delay information included in the XNF file is ignored.

If you have either X-BLOX modules or IOB and CLB primitives in your design, proceed to the next subsections.

FPGA Designs with X-BLOX Modules

If you are designing an XC3000A/L, XC3100A, or XC4000 family device and your design contains X-BLOX modules, you need to run XNFPrep with the drc_only option to perform a DRC check without trimming logic and write an XTG file before running the X-BLOX software.

1. Enter the following from the command line:

   xnfprep ***filename*****[.xff] drc_only=true**↵

   xblox ***filename*****[.xtg] mergeio=false archopt=false**↵

   The mergeio and archopt options must be set to false to prevent the X-BLOX program from doing any optimization.

2. From the command line, run XNF2VST with the -u option on the XG file (XNF format) that you just created:

   xnf2vst ***filename*****[.xg] -u**↵

FPGA Designs with IOB and CLB Primitives

If your XC2000, XC3000, or XC3100 family design contains IOB or CLB primitives, you need to run XNFPrep to generate an XTF file and XNFMAP to map the XTF file into logic and I/O resources of the FPGA.

1. Enter the following from the command line as shown below:

   xnfprep ***filename*****[.xff]**↵

   xnfmap ***filename*****[.xtf]**↵

   XNFMAP generates a netlist mapped into CLBs, IOBs, TBUFs, and clock buffers.

2. Run PPR with the options route=false and placer_effort=1 if your design is an XC3000A/L or an XC3100A design. Otherwise, run MAP2LCA.

Use one of the following commands:

ppr *filename*[.map] route=false placer_effort=1↵

or

map2lca *filename*[.map]↵

3. After mapping the design as explained in step 2, run LCA2XNF to create an XNF file. Specify the -u option to create a unit-delay netlist. Specify a new name for the output file to avoid overwriting the original XNF file.

   lca2xnf -u *filename*[.lca] *newname*[.xnf]↵

4. Run XNFBA to back-annotate your file.

   xnfba *filename*[.xff,.xg] *newname*[.xnf]↵

   XNFBA requires two input files: the unrouted netlist (XFF or XG) and the routed XNF file created by LCA2XNF. Names in the new XNF file are changed to match the names in the pre-routed XNF format file. By default XNFBA creates an output file called xnfba.xnf.

5. Use XNF2VST to translate the output XNF file into a functional simulation netlist.

   xnf2vst xnfba.xnf *filename*[.vst] -u↵

Translating LCA Files Created with XMake

XMake creates a placed and routed LCA file. Follow the steps outlined below to convert an LCA file into a file for use with the Simulate program.

1. Execute LCA2XNF with the -u option from the command line. Specify a new name for the output file to avoid overwriting the original XNF file.

   lca2xnf -u *filename*[.lca] *newname*[.xnf]↵

   Option -u selects unit-delay translation.

   At this point you can skip to step 3. However, your LCA file might contain signal names changed by the XNFMAP or PPR programs. Follow Step 2 if you wish to restore any signal names that changed during design processing.

2. Execute XNFBA from the command line as shown below:

   xnfba *filename*[.xg,.xff] *newname*[.xnf]↵

   XNFBA requires two input files: the unrouted netlist (XFF or XG) and the routed XNF file created by LCA2XNF. Names in the new XNF file are changed to match the names in the pre-routed XNF format file. By default, XNFBA creates an output file called xnfba.xnf.

3. Run **XNF2VST** with the **-u** option enabled from the command line as shown below:

   xnf2vst xnfba.xnf *filename*[.vst] -u↵

Creating Implemented Design Files

This section gives you a brief summary of how to generate implemented FPGA and EPLD designs.

FPGA Designs (XNF → LCA → BIT)

To translate the XNF file (XFF, XG, or XNF) of an FPGA design into an implemented LCA file, use the Xilinx implementation tools. For more information, refer to Figure 7-1 in the "Design Implementation" chapter of this manual and to your *XACT Reference Guide.*

You must implement your design before you can create a timing simulation file.

EPLD Designs (XNF → VMH → PRG or JED)

Run FITNET on your schematic design to implement your EPLD design. If you have PLD components or custom primitives in your design, you must assemble each equation file using PLUSASM before running FITNET. Implementing your design is a requirement before generating a timing simulation file. To generate a programming file in Intel HEX format (PRG file), run MakePRG. To generate a JEDEC format file (JED file), use the MakeJED program. Refer to the *XEPLD Reference Guide* for details.

Creating Timing Simulation Files

A file used for timing simulation must contain delay information, which is generated when the design is placed and routed. Follow the steps below to create a timing simulation file from a placed and routed file.

FPGA Designs (LCA → XNF → VST + DBA)

This section details the procedure used to generate a timing simulation file for FPGA designs.

For an XC3000A / L, XC3100A, or XC4000 family design, if you created the LCA file without using XMake, you need to run XDelay with the -dw option enabled on your LCA file; this option adds routing delay information to the LCA file. Then perform the following steps on the LCA file created by XDelay.

1. Type LCA2XNF from the command line. Specify a new name for the output file to avoid overwriting the original XNF file.

   `lca2xnf` ***filename*****`[.lca]`** ***newname*****`[.xnf]`**↵

   At this point you can skip to step 3. However, your LCA file might contain signal names changed by XNFMap (XC2000, XC3000, and XC3100) or PPR (XC3000A / L, XC3100A, and XC4000). Perform step 2 if you wish to restore any signal names that changed during design processing.

2. Run XNFBA from the command line as shown below:

   `xnfba` ***filename*****`[.xff,.xg]`** ***newname*****`[.xnf]`**↵

   XNFBA requires two input files: the unrouted netlist (XFF or XG) and the routed XNF file created by LCA2XNF. Names in the new XNF file are changed to match the names in the pre-routed XNF format file. By default XNFBA creates an output file called xnfba.xnf.

3. Execute XNF2VST from the command line as shown below:

   `xnf2vst xnfba.xnf` ***filename*****`[.vst]`**↵

At this point, the necessary files for timing simulation have been created. If you have included trace and stimulus information in your schematic, you can also run the ASCTOVST program on the ASCII

trace and stimulus files produced by XNF2VST. ASCTOVST converts ASCII trace (ATR) files to binary trace (TRC) files, ASCII stimulus (AST) files to binary stimulus (STM) files. ASCTOVST also converts binary files to ASCII files. (See Figure 10-4.)

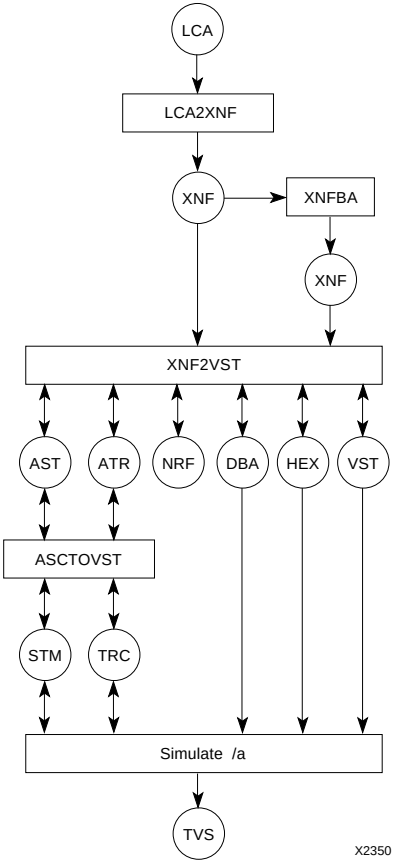


Figure 10-4 Translation Paths for Timing Simulation of FPGAs

EPLD Designs (VMH → XNF → VST + DBA)

This section outlines the steps used to create a timing simulation file for EPLD designs.

To perform timing simulation, you must first integrate your design using XEMake, FITNET, or FITEQN for behavioral designs. The implemented design database file resulting from any of these procedures has a VMH extension for all EPLD part types except XC7272, which uses a VMD file extension.

Then, perform the following steps on the VMH / VMD file:

1. Run **VMH2XNF** from the command line as shown below:

   vmh2xnf ***filename***↵

2. Run **XNF2VST** from the command line as shown below:

   xnf2vst ***filename***↵

At this point, the necessary files for timing simulation have been created. You can run the ASCTOVST program if you have ASCII trace and stimulus files. ASCTOVST converts ASCII trace (ATR) files to binary trace (TRC) files, and ASCII stimulus (AST) files to binary stimulus (STM) files. ASCTOVST also converts binary files to ASCII files. (See Figure 10-5.)

Note: The original symbols and interconnections contained in your OrCAD schematic are not used in the timing simulation netlist (XNF) file. Instead, VMH2XNF creates an original netlist based on the target EPLD device architecture as it is configured by your design. The names of all of your external wires and internal nodes that were not optimized by the software are preserved so that you can trace them during simulation.

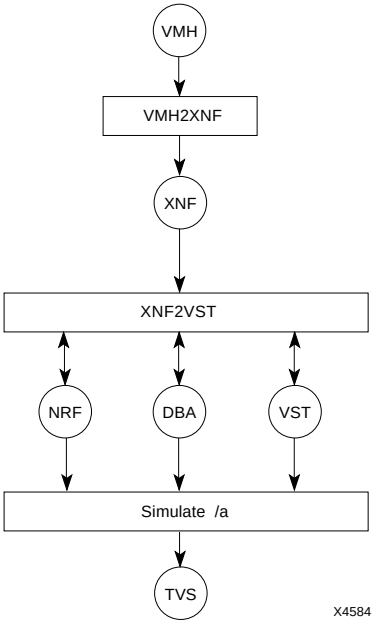


Figure 10-5 Translation Paths for Timing Simulation of EPLDs

Translation Programs for Simulation

This section includes information about the programs used to create an OrCAD simulation file. Included are sections on the XNF2VST and ASCTOVST programs. For information about the XNFBA and LCA2XNF programs, which are also used in the translation process, refer to the *XACT Reference Guide*. For information on the EPLD-specific program, VMH2XNF, refer to the *XEPLD Reference Guide*. This section applies to both functional and timing simulation programs.

XNF2VST Program

XNF2VST accepts either a routed or an unrouted XNF file and invokes XNF2INF to translate the XNF file into a VST file, which is read by the Simulate program. Any delay information included in the XNF file is passed to the DBA file, thus supporting timing simulation

of your design. Use the syntax shown here to create the VST file and other relevant simulation files from your XNF file:

xnf2vst *xnffile*[.xnf] [*vstfile*[.vst]] [*options*]↵

Input Files

- *xnffile*.xnf

  This file is an XNF format file for the current design. It can be either a routed XNF file or an XNF format file created prior to routing your current design. If your objective is to perform timing simulation on your design, you must use a routed XNF file, preferably back-annotated by the XNFBA program. The file extension is optional. If an extension is not specified, XNF is assumed. Valid extensions are XNF, XFF, and XG.

- *xnffile*.nrf

  XNF2VST reads this file if the -r option is specified; however, it is not a required input file.

Output Files

- *vstfile*.vst

  This file is the primary data-input file for the Simulate program. The VST file is a flattened netlist for the design, used for simulation purposes only.

- *vstfile*.ast

  The AST file is an ASCII file that contains all the stimulus information from your schematic design file. This file is not created if there is no stimulus information entered from the schematic editor.

- *vstfile*.atr

  The ATR file is an ASCII file that contains all the trace information from your schematic design file. This file is not created if there is no trace information entered from the schematic editor.

Note: AST and ATR files are created automatically the first time XNF2VST is run on your design. When XNF2VST is invoked again, it does not create new files. To create new files and overwrite the existing ATR and AST files, specify the -w option.

- *vstfile*.dba

  This file is an ASCII file that contains all timing delays within the design. It also includes the delays generated by routing. This file is created unless the -u option is specified. VST reads this file whenever you execute the Simulate program with the /a option.

- *vstfile*.nrf

  The name reference file (NRF) records any changes in the symbol reference names, net names, and any added net names. Browse through this file if you cannot find the original name you used. See the "XNF2VST Signal Names" and the "XNF2VST and FPGA/OrCAD Naming Conventions" sections, following, for more information on how XNF2VST handles names.

  XNF2VST always creates a new NRF file, which is based on the input NRF file if the -r option is specified. If the -r option is not specified, XNF2VST does not read the existing NRF file.

- *vstfile*.hex

  This is an Intel MCS hex file. It is created whenever the input XNF file contains one or more ROMs.

Options

You can modify the operation of XNF2VST using the following options:

-u — Unit-Delay Translation

Select this option to create a file for functional simulation only. Any delay information included in the XNF input file is ignored, and XNF2VST does not generate a DBA file. Without a DBA file present in your design directory, the Simulate program assumes a default delay of 0.1 ns (1 VST unit) for each gate-level logic element and a 0 ns delay for each net.

-r — Read Existing Name Reference File (NRF)

If the -r option is specified, XNF2VST reads the existing NRF file. A new NRF file is generated that is based on the existing NRF file. Previously assigned name references are retained.

-w — Overwrite Existing AST and ATR Files

If the -w option is specified, XNF2VST overwrites the existing AST and ATR files using the stimulus and trace information from the XNF file. Otherwise, XNF2VST generates new AST and ATR files only if stimulus and trace information exists in the XNF file and no AST and ATR files exist in the project directory.

-x — XACT Path

Use this option to specify the Xilinx XACT path. If you do not specify this option, the path is set to the value of the XACT environment variable.

XNF2VST Signal Names

Listed below are XNF2VST-generated signal names, which XNF2VST adds to the VST file when necessary.

- GR —The GR signal is an active Low signal that clears the contents of all flip-flops when you start simulation. It is only used in XC2000 family and XC3000 family designs.
- GSR — The GSR signal is similar to the GR signal, except that it is active High and is only used for XC4000 designs.
- GTS — GTS represents the global 3-state signal. When GTS is High, then all outputs are 3-state; when GTS is Low, then all outputs are active. GTS is only used for XC4000 designs.
- PRLD — The preload (PRLD) signal is active-High and initializes the flip-flops in an XC7000 design. XNF2VST adds PRLD when making an EPLD functional simulation netlist. VMH2XNF adds PRLD when making an EPLD timing netlist.
- VCC, GND — These names represent logic High and logic Low signals.
- *signal_name_INV#_I* — This naming convention is used whenever an inversion of an input pin is required. (This does not apply to logic gates.)
- *signal_name_INV#_O*— This naming convention is used whenever an inversion of an output pin is required. (This does not apply to logic gates.)

- OSC4_IN — This signal is created whenever the design uses an OSC4 primitive. Since the model cannot generate the fixed frequency by itself, you have to "drive" the OSC_4. The OSC4_IN signal directly drives the F8M output and all other outputs after it has been divided down properly. To get an exact 8 MHz waveform, set High and Low time for OSC4_IN to be 625 units (62.5 ns).

XNF2VST and FPGA/OrCAD Naming Conventions

When simulating an FPGA design with OrCAD/VST386+, all signals and symbols in the design must conform to both OrCAD/VST386+ and FPGA naming conventions (listed in the "Naming Conventions" section of Chapter 3). If a name does not conform to these conventions, then XNF2VST changes it; both the original name and the new name can be found in the NRF file. XNF2VST performs the following changes.

Symbol Names

- All symbols are renamed to names that begin with a "G".
- Lowercase characters are replaced with uppercase characters.

Signal Names

- Names with more than 14 characters are shortened to names that begin with a "$". The 14-character limitation is imposed by the OrCAD simulator.
- Lowercase characters are replaced with uppercase characters.

The following is a typical NRF file that provides examples of how XNF2VST changes names.

```
# ALL XILINX SYMBOLS RENAMED TO UNIQUE REFERENCE DESIGNATOR
# NAMES

G16  = U12
G17  = U10
G18  = U8
G70  = DEFVCC
G71  = DEFGND
G72  = U11
G73  = U9
G74  = U7
G75  = /atest/U13
G76  = /atest/another/U15
G77  = /atest/U14
G79  = /atest/another/U18
G80  = /atest/another/U17
G81  = /atest/another/U16

#  LONG XILINX SIGNAL NAMES RENAMED TO SHORT ORCAD NAMES

$4  = /atest/another/SIG4
$5  = /atest/another/SIG5
```

Recycled Aliases

XNF2VST supports recycled aliases in the Name Reference File. Therefore, you can modify the reference designator names directly in the NRF file by entering a symbol or signal name of your choice. The new name is used by XNF2VST during the translation process if you specify the -r option.

For instance, if the G75 reference designator name shown in the example above is not detailed enough, you can preserve the hierarchical structure of the original name (/atest/u13) by using the name GAU13 instead of G75. “G” is the required first character of the symbol name. “A” stands for the “atest” top-level portion of the schematic. “U13” is the lower-level primitive name.

When you rename a signal or a symbol, the maximum number of characters you can use is 14 characters for a signal and 5 characters for a symbol. All modified reference designator names must end with a number.

To rename signal name $4 in the NRF file using more details, you could call it $SIG4.

To keep the modified signal and symbol names in the NRF file and propagate the names back to your schematic file, use the -r option with XNF2VST. The existing NRF file is read, any necessary modifications are made, and a new NRF file is written.

ASCTOVST Program

ASCTOVST, provided with OrCAD/VST386+, is an ASCII-to-binary translator that converts ASCII trace (ATR) files to binary trace (TRC) files, and ASCII stimulus (AST) files to binary stimulus (STM) files. It also converts binary files to ASCII. You can select this program from the XDM Verify menu.

Input and Output Files

- *filename*.ast

  The AST file is an ASCII file that contains all the stimulus information from your schematic design file.

- filename.atr

  The ATR file is an ASCII file that contains all the trace information from your schematic design file.

- *filename*.stm

  The STM file is a binary file that contains all the stimulus information from your schematic design file. This file can be used as input to the VST simulator.

- *filename*.trc

  The TRC file is a binary file that contains all the trace information from your schematic design file. This file can be used as input to the VST simulator.

OrCAD Interface/ Tutorial Guide

SDT Tutorial

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Chapter 11

SDT Tutorial

This chapter steps through a typical Field Programmable Gate Array (FPGA) design procedure from schematic entry to completion of a functioning device using OrCAD's SDT Schematic Editor. Although the design is fairly simple, it demonstrates many development system features that you can use for more complex FPGA designs.

Design Flow

The Xilinx design flow is represented in Figure 11-1 and is described in the following steps. Most of the process is handled by the XMake program. You control each step using the XACT Design Manager (XDM), a graphic interface for Xilinx software.

- Create a schematic with the OrCAD schematic editor, using symbols from Xilinx libraries.
- XMake reads the SCH files and performs the following steps:
  - Runs OrCAD's Annotate program to assign unique instance names to all symbol placements in the schematic
  - Creates an OrCAD netlist (INF) file for each schematic using OrCAD's INET program
  - Translates INF files to Xilinx Netlist Format (XNF) with SDT2XNF
  - Merges the XNF files into a single top-level XNF file
  - Trims unused logic
  - Performs an electrical rule check on the XNF file
  - Maps the logic into FPGA device resources, configurable logic blocks (CLBs), and input/output blocks (IOBs)

  - Places the blocks and routes the connections between them
  - Creates a configuration bitstream
- Download the bitstream into one of the Xilinx demonstration boards to test the design

In this tutorial, the processing sequence is executed once, then a small change is made to the design, and it is processed again to show you how to do incremental design. A second tutorial steps through simulation of this design (see the "VST Tutorial" chapter in this manual), and advanced tutorials use optional blocks to demonstrate topics such as X-BLOX and Xilinx ABEL (see the "X-BLOX Tutorial," "Xilinx ABEL Tutorial," and "XACT-Performance and XDelay Tutorial" chapters of this manual).

You choose whether to target the design for an XC3000A, XC3000, XC4000A, or XC4000 device. Make sure you have the Xilinx demonstration board with the correct device, and the right software for the part you choose. (The Release Document that comes with your software package describes which devices it can support.) Then, using the appropriate demonstration board, you can observe your design in operation as part of a functioning system.

Note: This tutorial frequently references design flows that differ depending on which device you are using. To simplify the text, the phrase "XC3000 family" is used to reference all of the XC3000, XC3000A, XC3000L, XC3100, and XC3100A devices. When simply "XC3000" is used, it means that the XC3000A, XC3000L, XC3100, and XC3100A devices are not included. Similarly, "XC4000 family" means XC4000, XC4000A, XC4000D, and XC4000H parts, and "XC2000 family" includes both XC2000 and XC2000L devices.

For more information about the Xilinx design flow, see your *XACT User Guide*. For more information about the Xilinx / OrCAD interface, see the other chapters of this manual. They include useful information about the Xilinx / OrCAD interface and creating XNF files.

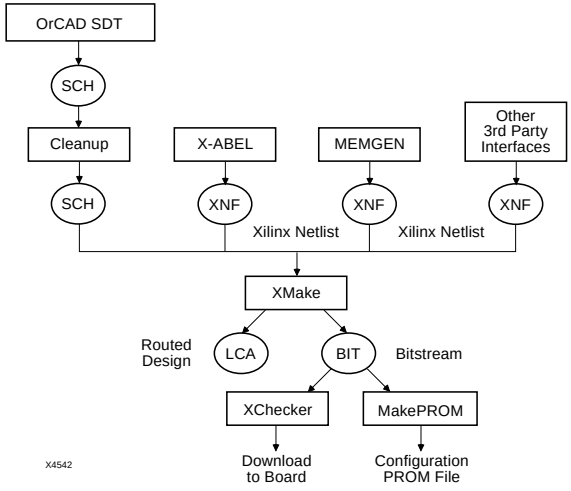


Figure 11-1 FPGA Design Flow

Required Software

This tutorial assumes that you are using the following versions of the development software.

- OrCAD / SDT 386+ — any version
- OrCAD / SDT Schematic Interface — SDT2XNF version 5.00 or later
- XACT Design Manager — XDM version 5.00 or later

Before Beginning the Tutorial

Before beginning the tutorial, you must set up your PC to use OrCAD 386+ and the XACT Development System software.

1. Verify that your PC is properly configured — consult the "Development System Hardware Requirements" section of *The Programmable Logic Data Book.*
2. Install SDT 386+.

3. Install the XACT Development System, following the installation instructions in the release documents provided with the software. You can install the Base Development System for OrCAD, which includes the Base Development System FPGA core tools and the OrCAD interface and libraries. Alternatively, you can load the OrCAD SDT / VST Interface (DS35) and the XACT Development System, DS502.

   When you have finished the installation, verify that your autoexec.bat file contains, among other things, lines similar to the following.

```
set XACT=c:\xact
set ORCADEXE=c:\orcadexe\
set ORCADESP=c:\orcadesp\
set ORCADPROJ=c:\orcad\
set ORCADUSER=c:\orcadesp\
PATH ...;c:\xact;c:\orcadexe;...
```

Note: This tutorial assumes that your Xilinx and OrCAD software are loaded on the c: drive, but you can substitute any other drive. If your PC is configured with only one drive, it is not necessary to specify a drive. You can load the Xilinx software on one drive and the OrCAD software on another drive, but it is recommended that you load the OrCAD / Xilinx Interface on the drive with your Development System software rather than on the drive with the OrCAD software. If you load the interface to a drive other than the one containing the Development System, you create a second XACT directory containing your Xilinx libraries. If you must create a second XACT directory because of drive size limitations, see the "Partitioning Software Between Two Different Disks" section in the "Getting Started" chapter.

   Your config.sys file should include lines similar to the following.

```
files=25
buffers=25
REM make extra space to accommodate more
REM environmental variables
shell=c:\dos\command.com c:\dos\ /e:1024 /p
```

   After installing the software, reboot your PC so that the changes are implemented. Changes to the autoexec.bat file take effect if you simply run the file by typing **`c:\autoexec`** ↵, but changes to

the config.sys file do not take effect until the PC is rebooted.

Installing the SDT Tutorial

The tutorial files are optionally installed when you install the Xilinx/OrCAD interface software. If you have already installed the software but are not sure whether you specified tutorial installation, check for the c:\xact\tutorial\orcad\calc directory. This directory contains the tutorial files.

Creating the Project Directory

OrCAD project directories are always placed in the ORCADPROJ directory as defined in your autoexec.bat file.

1. To create an OrCAD design directory using the ESP design environment, type **orcad** ↵ from any directory. The ESP executive screen appears.
2. Click twice with the left mouse button on the Design Management Tools button at the lower right. The first click selects the **Design Management Tools**, and the second selects **Execute** from the menu that appears in response to the first click.
3. Click on **Create Design**.
4. Click on the box next to New Design Name, and type **calc** ↵.
5. Click on the **OK** button.

   This procedure creates the c:\orcad\calc directory and copies several OrCAD configuration files into that directory from the c:\orcad\template directory.

Note: This tutorial assumes that your ORCADPROJ variable is set to c:\orcad\. You need not follow this convention.

6. Click on the **OK** button to exit the Design Management Tools.
7. Click on **Exit ESP** twice to exit the OrCAD ESP environment.

Configuring the Project Directory

The next step is to configure the Calc design directory for use with Xilinx designs. The design directory has been created and the configuration files installed, but the files do not contain the correct

information about the Xilinx libraries, the attributes allowed for Xilinx devices, and so forth.

Running XDraft

Xilinx provides a program, XDraft, that configures the project directory for either XC2000, XC3000, XC4000 or XC7000 (EPLD) family devices. Use XDraft with the correct modifier to target the Calc design to either the XC3000 or XC4000 family.

1. Type **`cd c:\orcad\calc`**↵.
2. Type **`xdraft 3`**↵ or **`xdraft 4`**↵, depending on whether you are targeting an XC3000 or an XC4000 device.

Sdt.cfg File

You can create an OrCAD project directory using ESP, as you did in the "Creating the Project Directory" section, or by creating a directory under ORCADPROJ with the mkdir command. However, there is one file that must be present before you can open the OrCAD schematic editor, SDT. You cannot bring up SDT unless you have an OrCAD configuration file, sdt.cfg, in the project directory.

XDraft's only function is to edit the sdt.cfg file and its simulation counterpart, vst.cfg. See the "Vst.cfg File" section of the "VST Tutorial" chapter of this manual for a discussion of the vst.cfg file.

Using a text editor, look at the sdt.cfg file, which is in ASCII format. (There is a binary version of the file, automatically created by the software, called sdt.bcf. You can ignore the sdt.bcf file or even delete it, but SDT regenerates it.) The sdt.cfg file is the most important file in the design directory. In fact, you could create a design directory simply by creating a directory under c:*orcadproj* in DOS, and copying the sdt.cfg file from c:*orcadproj*\calc. You would not need any other files, nor would you have to modify the file in any way to enter another Xilinx design, as long as the sdt.cfg file is copied from a directory configured for the same Xilinx device family.

XDraft makes several changes to the sdt.cfg file. The PLIB, LIB, DMF, DIM, and FN lines are modified during the configuration process. PLIB, the Library Prefix, points to the library files listed in the LIB statements. DMF points to the Macro File where keystroke macros are defined. DIM indicates the Initial Macro. The FN lines define the

parameters that can be used in Xilinx designs, such as LOCation and BLKNM. The PLIB, LIB and FN lines differ between the product families.

A portion of a sample sdt.cfg file for the XC3000 family follows.

```
{ OrCAD/SDT IV Configuration File }
PDRV   = 'C:\ORCADESP\DRV\'
PSCH   = ''
PLIB   = 'c:\xact\xc3000'
LIB    = 'XC3000.LIB'
LIB    = 'XBLOX.LIB'
DD     = 'VGA640.DRV'
PRD    = ''
PLD    = ''
DMF    = 'c:\xact\MACRO3.MAC'
DIM    = '\I'
.
.
.
FN1    = 'LOC, OPTIONS'
FN2    = 'BLKNM'
FN3    = 'BASE'
FN4    = 'CONFIG'
FN5    = 'EQUATE_F'
FN6    = '$FCONT'
FN7    = 'EQUATE_G'
FN8    = '$GCONT'
.
.
.
```

You can edit the sdt.cfg file with any text editor to add new libraries to the search path just by adding additional LIB lines. OrCAD also has automated methods of adding libraries. For a discussion of user-created libraries, see the "User-Created Libraries" section in the "OrCAD SDT Desgin Techniques" chapter.

Note: If you manually edit the sdt.cfg file, it is a good idea to delete the binary configuration file, sdt.bcf, before entering OrCAD again. Otherwise, your changes may not be seen by the software.

Copying the Tutorial Design Files

The SDT tutorial installation creates a directory on the target drive called c:\xact\tutorial\calc (assuming that c:\xact is your XACT directory), and copies over the files you need to complete the tutorial. However, you must copy the design files to your calc project directory.

1. Make sure you are still in the c:\orcad\calc directory.
2. Copy the files you need to perform the tutorial. Type:

   copy c:\xact\tutorial\orcad\calc ↵

   The tutorial files are copied into the project directory.

Solutions Directories

Not all of the files for the complete Calc design are placed in the project directory, because you create some files during the tutorial. However, solutions directories with all input and output files are provided. They are located in the c:\xact\tutorial\orcad\calc directory and are named as follows:

soln_3ka — solution files forXC3020APC68
soln_4ka — solution files forXC4003APC84
soln_3k — solution files forXC3020PC68
soln_4k — solution files forXC4003PC84

The solutions directories contain all pertinent design files from the completed tutorial, including all schematics and the bitstream generated by the MakeBits program.

Brief descriptions of the solutions files are given in Figure 11-2. Most of these files are created in the course of this tutorial; therefore, only a few of them are placed directly into the c:\xact\tutorial\calc directory. However, some may be needed for later reference, so do not write over any files in the solutions directories.

Many of the files listed in Figure 11-2 are intermediate files that are listed here for information only. Not all intermediate files are created for all FPGA families. Not all intermediate files are included in all solutions directories.

File	Description
stat_abl.abl	Xilinx ABEL file for state controller module, replaces Statmach schematic (advanced tutorial)
stat_abl.xsf	Symbol definition file for Xilinx ABEL block (advanced tutorial)
calc_4k.cst	Constraints file that sets pad locations for 4K parts
calc.sch	Top-level schematic file for design
calcsim.sch	Top-level schematic with stimulus and trace data
control.sch	Schematic file for control module
statmach.sch	Schematic file for state controller module
alu.sch	Schematic file for ALU module
alu_blox.sch	Schematic file for ALU module, X-BLOX version (incomplete, advanced tutorial)
bloxsoln.sch	Schematic file for ALU module, X-BLOX version (complete, advanced tutorial)
muxblk2.sch andblk2.sch orblk2.sch xorblk2.sch	Schematic files for arithmetic functions, used in ALU
muxblk5.sch	Schematic file for multiplexing arithmetic outputs, used in ALU
stack.sch	Schematic file for stack
stack_4k.sch	Schematic file for stack, optional 4K version using on-chip memory, replaces Stack schematic (use with 4K parts only)
led_inv.sch	Schematic file for pad interface to LED bar on 3K/4K and 4K demo boards
led_tru.sch	Schematic file for pad interface to LED bar on 3K demo board
7segdec.sch	Schematic file for 7-segment decoder
7seg_inv.sch	Schematic for pad interface to 7- segment display on 3K/4K and 4K demo boards
7seg_tru.sch	Schematic for pad interface to 7-segment display on 3K demo board
sw7.sch	Schematic for pad interface to seven of eight switches on demo boards
debounce.sch	Schematic for debounce circuit
osc_3k.sch	Schematic for pad interface to RC circuit on demo boards
osc_4k.sch	Schematic for 4K on-chip oscillator
calcsim.ast	ASCII stimulus data for advanced tutorials

calcsim.stm	Binary stimulus data for advanced tutorials
calcsim.atr	ASCII trace data for advanced tutorials
calcsim.trc	Binary trace data for advanced tutorials
*.inf	OrCAD netlist files generated by INET
calc.inx	File listing generated by INET
xnf\stat_abl.xnf	Xilinx netlist format file created by Xilinx ABEL (advanced tutorial)
xnf\stat_abl.xas	Simulation version of XNF file created by Xilinx ABEL (advanced tutorial)
xnf*.xnf	Xilinx netlist format files created by SDT2XNF
calc.xff	Output of XNFMerge, netlist of merged design
calc.mrg	Merge report file generated by XNFMerge
calc.xtf	Output of XNFPrep, netlist of trimmed design
calc.prp	Report file generated by XNFPrep
xnfprep.log	Log file generated by XNFPrep
calc.map	Mapped logic file generated by XNFMAP
calc.pgf	Paritioning guide file generated by XNFMAP, needed for doing incremental design
calc.crf	Cross-reference file generated by XNFMAP
calc.rpt	Routing report file generated by PPR
calc.rpf	Routing report on file without delay optimization
ppr.log	Log file generated by PPR
calc.lca	Placed and routed LCA file generated by PPR
calc.lcb	Earlier version of LCA file generated by XMake
calc.odf	Intermediate version of LCA file generated by PPR
calc.tna	Logic mapping table created by PPR
calc.bit	Bitstream for downloading to FPGA generated by MakeBits
calc.mbo	Bitstream configuration file generated by MakeBits
calc.mak	Script file generated and used by XMake
calc.out	XMake report file

Figure 11-2 Tutorial Files in the Solutions Directories

Loading the CALC Schematic into OrCAD SDT

You have created and configured the project directory, and copied over existing schematics for the Calc design. You are ready to load the CALC schematic into the OrCAD schematic editor, SDT, and begin working on the design.

Starting the XACT Design Manager (XDM)

The XACT Design Manager (XDM) is a menu–based shell program that interfaces to all Xilinx development software. In addition, many third–party software packages (including OrCAD) can be accessed through XDM. Thus, you can control the entire design process, from schematic entry in OrCAD to downloading the bitstream with XChecker, from inside XDM.

1. To start XDM from the DOS prompt, type **`xdm`** ↵.

   XDM provides a context–sensitive help command that provides explanations of available software options.

2. To use XDM Help, position the cursor over the menu selection in question and press the **`F1`** function key. A text window briefly explaining the command function and usage opens on the screen. To return to the XDM menu, press **`F1`** again.

Accessing OrCAD from XDM

The next step is to access the OrCAD software.

1. To start the OrCAD design environment from the XDM menu, click the left mouse button once on **`DesignEntry`**.

2. Select **`ORCAD.`**

   The ESP executive screen appears.

Note: If OrCAD does not appear in the DesignEntry menu, check the autoexec.bat file. Verify that the ORCADEXE directory appears on your path, with the correct drive designator. If the ORCADEXE directory is on your search path, but OrCAD does not appear in the DesignEntry menu, XDM may not know where to look for it. Select **`Utilities → Scandisk`** from the XDM menus; XDM scans your system for supported software.

Selecting Calc as the Active Design

You must select Calc as your active design. Currently the active directory is set to c:\orcad\template, which is the directory where OrCAD stores templates for all of its configuration files.

1. Select **`Design Management Tools → Execute`**.

A list of all OrCAD project directories appears.

2. Select the Calc design from the list of design directories.
3. Select **OK**.

   The words `CALC Design` appear at the top of the ESP screen.

Changing the Default Design

If the startup design is set back to Template the next time that you go into OrCAD, you must change it to Calc the same way you did the first time. It is easier to set the default startup design from the OrCAD ESP Design Environment to avoid having to set the design each time you go into OrCAD. To make this change:

1. Select **Schematic Design Tools → Configure ESP**.
2. Go down to Design Options and set the `Startup Design` to **CALC**. Click **OK** to save the change.

This procedure is equivalent to changing the c:\orcad\template\esp.cfg file so that one line reads **DESIGN = 'CALC'**.

The Calc design remains the default design for the ESP Design Environment until you change the default again.

Accessing SDT, the OrCAD Schematic Editor

The schematic editor portion of the OrCAD/SDT package is called Draft.

1. To enter the schematic editor, select **Schematic Design Tools → Execute**.
2. Click on **DRAFT → Execute**.

Note: There are other ways of entering Draft. You can type **orcad** ↵ directly at the DOS command line, which bypasses XDM entirely but still uses the ESP Design Environment. You can also type **draft calc** ↵ at the DOS prompt, thereby bypassing all frameworks.

The top-level schematic for Calc appears on the screen. The drawing consists of nine functional blocks, each labeled to describe the operation of a logic diagram on a lower level, and one pad with an

input buffer. The root drawing of the design is already complete. You will add schematic information to the lower levels.

Using the mouse, pan around the schematic to see the entire top–level drawing. As the cursor is moved toward the edge of the screen, a different section of the drawing scrolls into view. At first, the schematic is displayed at a low level of magnification, which can be changed by using the OrCAD Zoom command.

Using OrCAD Commands

In the OrCAD schematic editor, you execute commands by choosing options from a series of pulldown menus. As shown in Figure 11-3, each submenu contains a group of related commands. The diagram in this figure shows the main Draft menu, labeled OrCAD/SDT, and several submenus that are used in this tutorial.

Entering OrCAD Commands with the Mouse

The Zoom command brings up the Zoom menu, which changes the magnification level of the drawing. The Zoom command is used here to demonstrate how to execute commands and access menus using the mouse.

1. To bring up the command menu, click the left mouse button anywhere on the schematic, or type ↵.

   The OrCAD/SDT menu shown in Figure 11-3 appears on the screen.

2. Move the menu cursor to **Zoom** and click the left mouse button again; the Zoom submenu appears.

Note: To escape from an incorrectly selected menu without executing a command, either click the right mouse button or press the **Escape** key.

3. Use the mouse to select the **In** option to execute the Zoom In command.

   The drawing is now displayed at its highest level of magnification (scale = 1), so that you can read all of the text labels.

   To zoom back out and view more of the drawing on one screen, use the Zoom Out command.

4. Select **Zoom** from the main menu and then **Out** from the submenu.

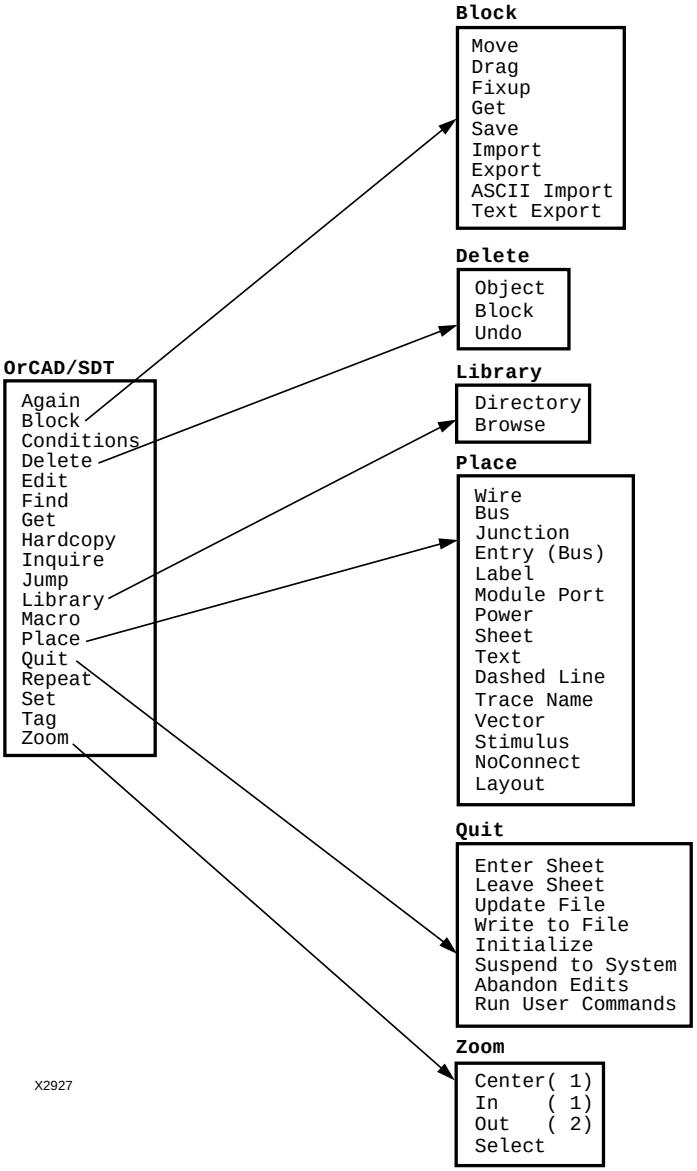


Figure 11-3 OrCAD Menu Structure

Entering OrCAD Commands from the Keyboard

As an alternative to using the mouse to select menu commands, you can execute a command by typing the first character of a menu command. If two commands begin with the same letter, a different letter from the second command is highlighted in the menu. Use the highlighted letter to select the second command using the keyboard. Executing commands from the keyboard is usually faster and more convenient than using the menus, once you are familiar with available commands.

Although it is not necessary to have the main menu on the screen to enter a command, the Enter (↵) key calls it up for reference.

1. To execute the Zoom Out command without using the mouse, first press ↵ to call up the main menu. (This step is optional.)
2. Type **z** to execute the Zoom command.

   The Zoom submenu appears on the screen.
3. Type **o** to execute the Out option.

The magnification level of the drawing is reduced.

Using OrCAD Key Macros

Although this menu organization provides a logical structure for the many OrCAD commands, repetitive operations such as drawing wires can be somewhat tedious. For this reason, OrCAD allows the PC function keys to be defined as key macros, each of which triggers a sequence of menu selections. All available key macros are listed in the "Macro Files" section of the "Getting Started" chapter of this manual. They are defined in the c:\xact\macro3.mac file, which is the macro file referenced by the sdt.cfg file; see the discussion of the sdt.cfg file in the "Before Beginning the Tutorial" section earlier in this chapter.

In this tutorial, most commands are presented in menu form, since familiarity with the menu structure is vital to understanding OrCAD. However, equivalent key macros are available for many command sequences. Key macros performing functions used in this tutorial are listed in Figure 11-4, and each of these key macros is referenced as a "Keyboard Shortcut" the first time the command sequence appears in the tutorial.

Note: You can terminate a keyboard macro with the Escape key; however, you may find that the interrupted command sequence leaves you in an intermediate state. In this case, click the left mouse button once, then the right button once, to exit the macro.

Block → Drag → Begin → End	F5
Block → Export	Shift-F4
Block → Import	Shift-F3
Block → Move	Ctrl-F5
Block → Save → Begin → End → Block → Get	F3
Delete → Block	Ctrl-F4
Delete → Object	F4
Edit → Edit → Name	Alt-E
Get ↵	F1
Library → Directory	Ctrl-F1
Place → Bus → Begin	F6
Place → Junction	Ctrl-F2
Place → Label	F7
Place → Module port	F8
Place → Sheet	F9
Place → Text	Ctrl-F9
Place → Wire → Begin	F2
Quit → Suspend to System	Shift-F9
Quit → Update File	Shift-F6
Quit → Update File → Abandon Edits	Shift-F10
Quit → Update File → Enter Sheet → Enter → Escape	Alt-F7
Quit → Update File → Leave Sheet	Alt-F8
Quit → Write to File	Shift-F7
Zoom → In	PgDn or Alt-F5
Zoom → Out	PgUp or Alt-F6

Figure 11-4 OrCAD/SDT Key Macros Used in This Tutorial

Design Description

The top-level schematic for the tutorial design, Calc, is shown in Figure 11-5. The Calc design consists of a 4-bit processor with a stack. The processor performs functions between an internal register and either the top of the stack or data input from external switches. The results of the various operations are stored in the register and displayed in hexadecimal on a 7-segment display. The top value in the stack is displayed in binary on a bar LED.

The design consists of nine basic functional blocks:

- ALU

  The arithmetic functions of the processor are performed in this block.

- CONTROL

  The opcodes are decoded into control lines for the stack and ALU in this module.

- STACK

  The stack is a 4-nibble storage device. It is implemented using flip-flops in the device-independent design. You can substitute a RAM module called STACK_4K in the XC4000-specific design to take advantage of the on-chip RAM capability of the XC4000 family.

- OSC_3K

  This module is used in XC3000 family designs. It generates a clock signal using the RC oscillator circuit on the FPGA (XC3000/XC4000) and XC3000 demonstration boards. It is replaced by the OSC_4K internal oscillator circuit for the XC4000-specific design.

- DEBOUNCE

  This circuit debounces the “execute” switch, providing a one-shot output.

- SW7

  The switch connections for opcode and data input are implemented within this module.

- 7SEGDEC

  This block decodes the output of the ALU for display on the 7-segment decoder.

- 7SEG_INV

  This module implements the connections to the 7-segment display on the FPGA (XC3000/XC4000) and XC4000 demonstration boards. It is replaced by the 7SEG_TRU circuit if you are using the XC3000 demonstration board.

- LED_INV

  The value at the top of the stack is displayed in binary on the LED bank of the FPGA (XC3000 / XC4000) and XC4000 demonstration boards using this block. It is replaced by the LED_TRU circuit if you are using the XC3000 demonstration board.

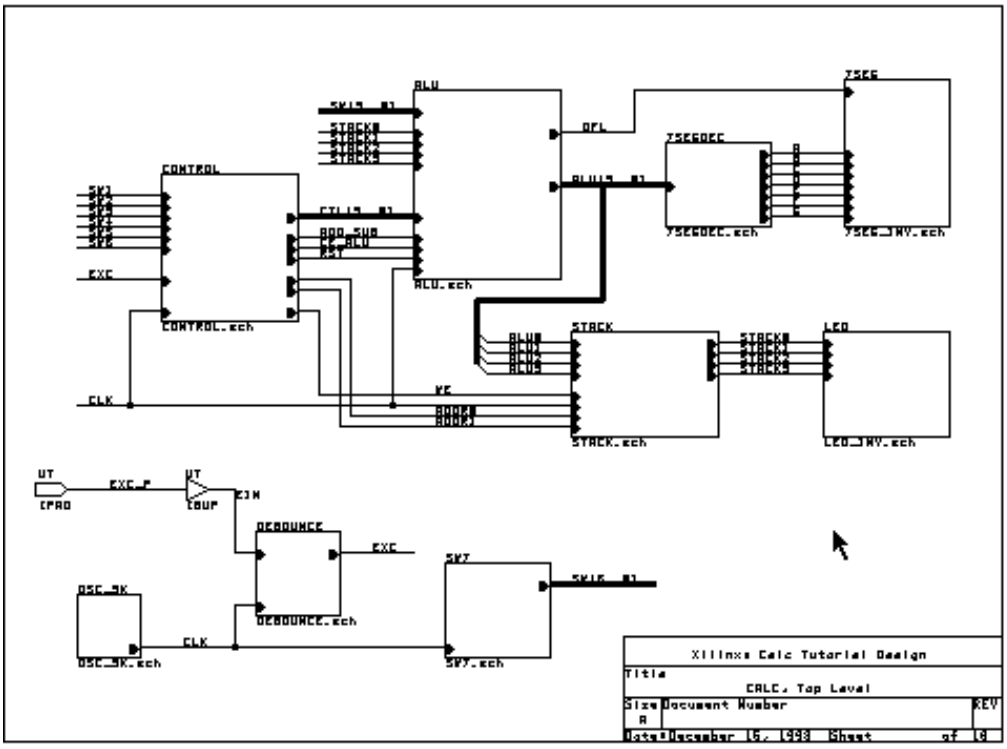


Figure 11-5 Top-Level Schematic for CALC

Exploring OrCAD Symbols

Each of the blocks shown in Figure 11-5, such as CONTROL or ALU, is linked to a second-level module that describes its logic. In turn, any second-level module can contain another block that references a third-level drawing. This organization is called a hierarchical structure.

There are two types of symbols used in OrCAD schematics.

CONTROL and ALU are placements of OrCAD "sheets." A sheet is a schematic that you create. You must draw or copy a new symbol each time you place a sheet in a schematic.

The other type of symbol is a library element. A library element has a single symbol, which you place using the Get command. You can place the symbol as many times as you wish. The symbol is stored as part of a .lib file, such as the ones you received from Xilinx. You can also create your own library symbols. If you plan to create a sheet that will be placed many times in a design, or one that will be used in more than one design, it is a good idea to put it into a user-created library. See the "User-Created Libraries" section of the "OrCAD SDT Design Techniques" chapter for instructions on how to make your own libraries.

The IPAD and IBUF symbols in the Calc schematic are missing the colors that characterize the other blocks in the design, because they are elements from a library and not sheets in the design directory. Placements of sheets from the design directory always appear in color. The presence or absence of these colors is a simple and effective way to tell the difference between library elements and schematic sheets.

Completing the ALU Schematic

In this section, you push into the ALU schematic. The ALU schematic is incomplete; you complete the schematic by adding two sheet symbols and one element from the Xilinx library.

Pushing into the ALU Schematic

From the CALC schematic, push into the ALU block.

1. Select **Zoom → In** to bring the drawing to its highest level of magnification.

Keyboard Shortcut: **Zoom → In** corresponds to the "PgDn" key or the Key Macro "Alt-F5." You can perform step 1 by pressing the **PgDn** key or by holding down the **Alt** key and pressing the **F5** function key.

2. Move the cursor inside the ALU sheet symbol.

3. Press ↵ to bring up the main menu, and select **Quit**. Alternatively, you can just type **q**.
4. Select **Enter Sheet** from the submenu.

   A list of command options appears horizontally across the top of the screen. This command list is different from a regular OrCAD menu; only the keyboard can be used to make a selection. Simply type the first letter of the desired option.

   Alternatively, to call up an equivalent menu for this command list, click the left mouse button; a menu with exactly the same options appears.
5. Using either method just described, choose **Enter** from the menu to finish the Enter Sheet command.

   Draft now displays the drawing alu.sch, which is the schematic represented by the ALU block in calc.sch.
6. Press the **Escape** key or click the right mouse button to exit the Enter Sheet command list.

Keyboard Shortcut: **Quit → Enter Sheet → Enter → Escape** is similar to the Key Macro "Alt-F7," which saves the current schematic and then enters the new sheet. Alt-F7 actually corresponds to **Quit → Update File → Enter Sheet → Enter → Escape**. You can perform steps 3 through 6 by holding down the **Alt** key and pressing the **F7** function key.

The incomplete schematic for the ALU is shown in Figure 11-6. This schematic contains both library elements and sheet symbols that reference third-level drawings. You will add two sheet symbols and a library element to complete the schematic.

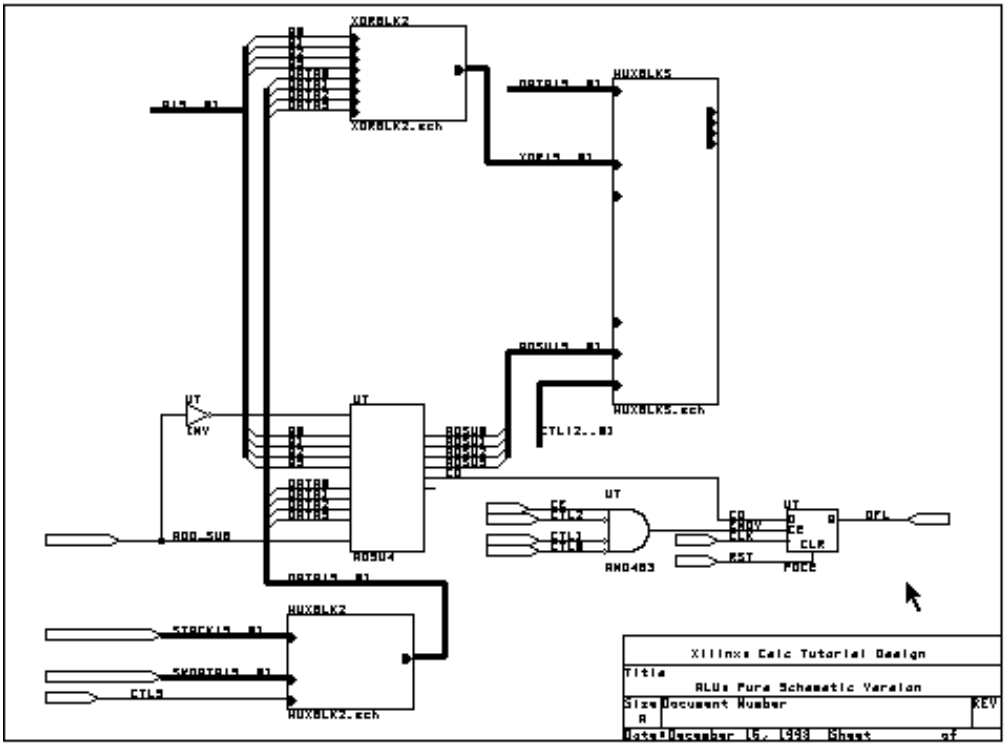


Figure 11-6 Incomplete Schematic Drawing for ALU

Enabling X and Y Coordinates

Before creating the sheet symbols for the ALU, enable the display of the X and Y coordinates.

1. Select **Set** from the main menu.
2. Click on **X,Y Display** on the Set submenu.
3. Select **Yes**.

You can use either the mouse or the keyboard to enter these commands.

Now move the cursor around the drawing, and observe that the absolute X,Y coordinate location of the cursor is displayed in the

upper right corner of the screen. This display makes sheet symbol placement easier.

Defining a Sheet Symbol

Now you are ready to begin creating the ANDBLK2 sheet symbol. As a guide, refer to Figure 11-7, which shows the completed symbol for ANDBLK2.

1. Click on **Place → Sheet**.

Keyboard Shortcut: **Place → Sheet** corresponds to the Key Macro "F9." You can perform step 1 by pressing the **F9** function key.

2. Position the cursor at location 3.30, 2.60, using the coordinate display as a guide. Establish this point as the upper left corner of the sheet symbol by typing **b** to select Begin.

3. Now move the cursor to position 4.40, 3.50; the sheet symbol changes size as the cursor is dragged to the lower right corner. Establish this corner by selecting **End**.

   This step creates a rectangular block with a "?" displayed above the top left-hand corner. Move the mouse and observe that the cursor now moves only along the sides of the block, facilitating the placement of input and output pins. A new command list appears displaying the options used in creating pins.

4. To define the first input pin, position the cursor at location 3.30, 2.70 and type **a** to select **Add-NET**.

   The `Net Name?` prompt appears.

5. Type the name of the first input pin: **A0** ↵.

   Now the Net Type menu appears.

6. Select **Input**.

   The A0 input pin is displayed on the left side of the functional block.

7. To define the second input pin, position the cursor at 3.30, 2.80 and select **Add-Net**. Type **A1** ↵, and click on **Input**.

   These commands establish the A1 input pin for the block.

8. Similarly, add input pins for A2, A3, B0, B1, B2, and B3, in the locations shown in Figure 11-7

   You will define the output as a bus. Any pins may be defined either as buses or as individual signals. Which of these types of pins you use in your designs is a matter of personal choice.

9. To define the output pin, position the cursor at 4.40, 3.00 and select **Add-Net**. Type **O[3..0]** ↵, and select **Output**.

   These commands establish the O[3..0] output pin for the block.

10. If a pin is misplaced or incorrectly named, delete it by placing the cursor on the pin and selecting **Delete**. The erroneous pin is erased; use **Add-Net** to place the correct pin.

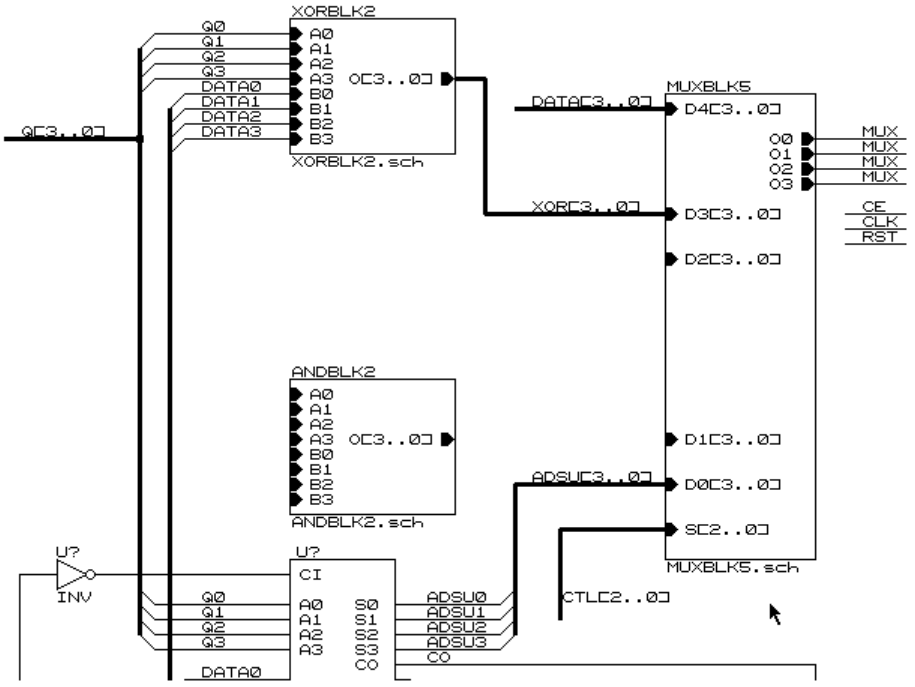


Figure 11-7 Completed ANDBLK2 Symbol in ALU Block

11. Once all pins have been correctly placed, the sheet symbol must be given a name. Select **Name**.

    The `Sheet Name? ?` prompt appears. The second question mark is actually the current name of this sheet symbol, as shown above the block.

12. Use the **Backspace** key to delete this default name and type **ANDBLK2**↵.

    This particular sheet symbol can have any name, but for the sake of simplicity it is given the same name as the schematic.

    If you do not specify a Sheet Name for this symbol, a name is assigned by the software, and the name may change if you make a change anywhere in the schematic. It is easier in the long run to make a habit of assigning a Sheet Name to each symbol as you enter it on the schematic.

    You must also specify the file that contains the actual drawing for ANDBLK2. This file name *must* be defined to be the same as the schematic name with the addition of an .sch extension.

13. To name this file, select **Filename**.

14. Use the Backspace key to erase the default file name given, and type **ANDBLK2.SCH** ↵.

    This step completes the definition of the ANDBLK2 sheet symbol.

15. Press the **Escape** key twice or click the right mouse button twice to escape from the current command lists.

Copying a Sheet Symbol

The other sheet symbol missing from the ALU schematic is called ORBLK2. You could create this symbol with the same commands that you used for ANDBLK2. However, since ORBLK2 is equivalent to ANDBLK2 in size and pin format, it is simpler to copy ANDBLK2 to another location, and then edit the symbol to create ORBLK2.

1. To copy ANDBLK2, move the cursor to the upper left corner of the sheet symbol just below the symbol name.
2. Select **Block** → **Save** → **Begin** → **End**.

This procedure stores a copy of the ANDBLK2 symbol in a temporary buffer for later placement.

3. To place this copy, select **Block → Get**.

   A highlighted outline of ANDBLK2 appears; as the cursor is moved, an outline of the sheet symbol is dragged with it.

Keyboard Shortcut: **Block → Save → Begin → End → Block → Get** corresponds to the Key Macro "F3." You can perform steps 2 and 3 by pressing the **F3** function key.

4. To place this new symbol in the proper location for ORBLK2, move the cursor up to location 3.30, 1.40 and select **Place**.

   A copy of ANDBLK2 appears in the new location. The block outline is still highlighted over this copy of ANDBLK2 , so you could move the cursor to another location and place a second copy. However, only one copy is needed for this schematic, so press **Escape** to exit the command.

5. Next, the names on this copy of ANDBLK2 must be modified to represent ORBLK2. Place the cursor inside the new block and select **Edit → Edit**.

   The command list used during pin creation appears on the prompt line. As before, the cursor is locked onto the sides of the symbol.

6. Correct the symbol name by selecting **Name**.

7. Change the name ANDBLK2to **ORBLK2** ↵.

   As before, you can make the Sheet Name any string, as long as it does not violate the Xilinx conventions. See the "Naming Conventions" section of the "OrCAD SDT Design Techniques" chapter in this manual for a discussion of legal names.

8. Now select **Filename**.

9. Change the file name to **ORBLK2.SCH** ↵.

   Since none of the pins or pin names need to be changed, this step completes the definition of the ORBLK2 sheet symbol.

10. Press **Escape** twice to exit the Edit Edit command.

Placing a Library Symbol

You have placed two schematic sheets, ANDBLK2 and ORBLK2. There is still one block missing from the ALU schematic: a 4-bit register to store the contents of the ALU. You can use a 4-bit register from the Xilinx library to implement this function. The Xilinx library supports basic primitive functions such as AND gates and OR gates, and also more sophisticated macros, including counters, registers, and many other standard MSI functions

1. Move the cursor to the upper right-hand corner of the ALU schematic.
2. To load the Xilinx macro symbol for a 4-bit register, select **Get**.

   The `Get?` prompt appears.

Keyboard Shortcut: **Get** is similar to the Key Macro "F1," which corresponds to **Get** ↵. The Get command followed by a carriage return displays a list of all libraries referenced by your sdt.cfg file. When you select one of the libraries, the contents of the library are displayed so that you can select an element for placement.

3. Type the Xilinx name for a 4-bit register with clock enable, **fd4ce** ↵.

   An outline of the library symbol appears attached to the cursor.
4. Position the symbol at location 7.60, .90 and select **Place**.

   This step places a copy of the FD4CE symbol at the specified location. After the first register has been placed, the symbol is still attached to the cursor, so that additional symbols can be placed on the drawing. However, you only need one placement of FD4CE.
5. Press **Escape** to exit the Get command.

Drawing Wires

The next step is to add the wires that connect the FD4CE symbol to the other signals on the schematic. Refer to Figure 11-8 for a drawing of the symbol with added wires.

1. To draw the wire connected to the D0 pin, first position the cursor at the D0 pin on FD4CE, and select **Place → Wire → Begin**.

   These commands establish the starting point of the wire.

Keyboard Shortcut: **`Place → Wire → Begin`** corresponds to the Key Macro "F2." You can perform step 1 by pressing the **F2** function key.

2. Now move the cursor to the left, to the pin on the MUXBLK5 symbol labeled "O0." A new wire is drawn behind the moving cursor. To establish this point as the endpoint of the wire, select **End**.

   The complete wire is now in place.

3. Similarly, add wires connecting the pins labeled D1, D2, and D3 with the MUXBLK5 symbol pins labeled O1, O2, and O3.

Note: If a wire is drawn so that it overlaps the pin of a symbol, the two nets *are not* connected. The wire must terminate at the endpoint of the symbol pin.

4. Add dangling wires to the CE, C, and CLR pins. Refer to Figure 11-8 for approximate locations. You connect these wires later on in the tutorial by placing labels on them that correspond to label names on wires elsewhere in the schematic.

5. Add wires to the Q0, Q1, Q2, and Q3 pins as shown in Figure 11-8. Besides making internal connections, these wires leave the ALU schematic sheet by way of elements called module ports that you add later on in the tutorial.

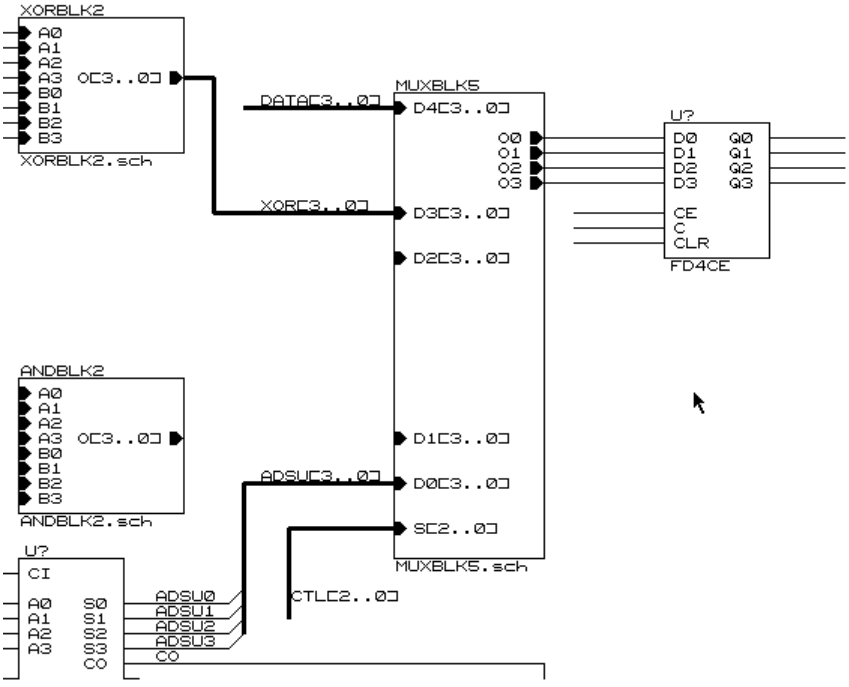


Figure 11-8 FD4CE with Wires in ALU Schematic

Drawing Buses

Move the cursor to the left until the ORBLK2 and ANDBLK2 sheet symbols that you previously added come into view. Now draw the buses connecting the bused output pins of ORBLK2 and ANDBLK2 with the MUXBLK5 input pins.

1. Position the cursor at the ORBLK2 output pin labelled O[3..0].
2. Select **Place → Bus → Begin**.

   These commands establish the starting point of the bus.

Keyboard Shortcut: **Place → Bus → Begin** corresponds to the Key Macro "F6." You can perform step 2 by pressing the **F6** function key.

3. Now move the cursor right to the MUXBLK5 pin labeled D2[3..0]. The new bus is drawn behind the moving cursor.
4. Select **End** to establish the endpoint of the bus.

   The complete bus is now in place.
5. Repeat steps 1 through 4 to make the bus connection between the ANDBLK2 O[3..0] pin and the D1[3..0] pin of the MUXBLK5 symbol.

Placing Bus Entry Elements

It is often convenient to join several related signals into a bus. To join signals into a bus, or to break individual signals from a bus, use a Bus Entry element.

The 4-bit buses used as input to the ANDBLK2 and ORBLK2 sheets are broken into individual elements before descending into the sheets, in order to teach you to handle both types of symbol pins.

Break out the Q3, Q2, Q1, and Q0 signals from the bus labeled Q[3..0]. Refer to Figure 11-9 while performing the commands in this section.

1. Select **Place → Entry (Bus)**.

   A "/" symbol appears at the cursor.
2. Move the cursor to location 2.40, 1.50.
3. Select **Place**.

   The Bus Entry element is placed so that one end rests on the bus labeled Q[3..0] and the other end is in line with the A0 pin on the ORBLK2 symbol.
4. Move the cursor down by one grid and select **Place** again to place the Entry element for the A1 pin.
5. Similarly, place Entry elements for the A2 and A3 pins of the register.
6. Move the cursor to location 2.60, 1.90 and select **Place**.
7. Moving down one grid at a time, place the other Bus Entry elements to allow connections between the B pins and the bus labeled DATA[3..0], as shown in Figure 11-9.
8. Similarly, add Entry elements for the ANDBLK2 inputs. As with

the ORBLK2 component, the Q[3..0] bus provides input to the A pins, and the DATA[3..0] bus drives the B pins.

9. Press **Escape** or click the right mouse button to exit the Place command.

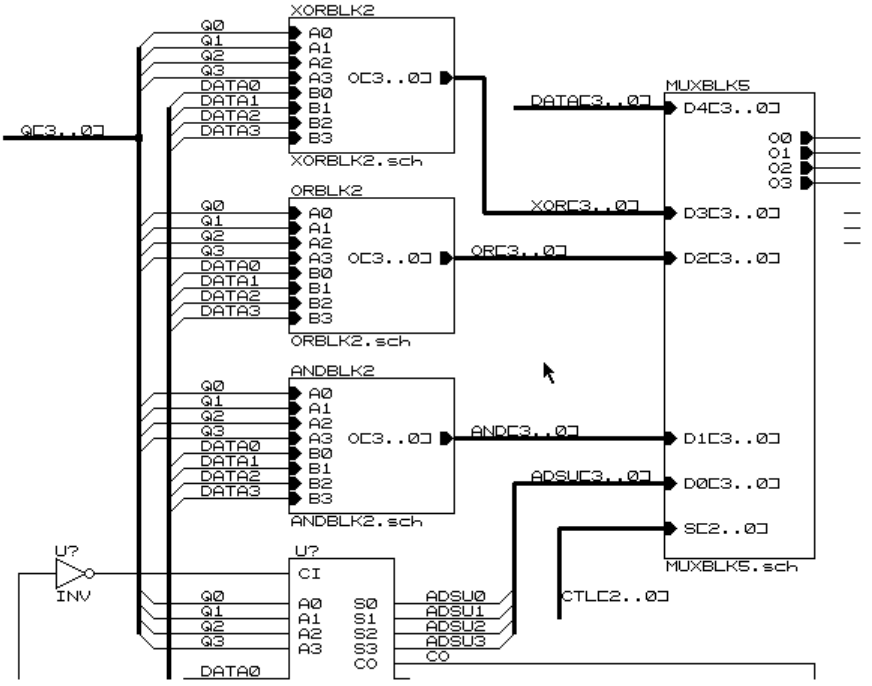


Figure 11-9 ANDBLK2 and ORBLK2 Connections and Labels

Completing Connections to ANDBLK2 and ORBLK2

The next step is to add the wires that connect the ANDBLK2 and ORBLK2 symbols with the bus entry elements, as shown in Figure 11-9.

1. Use the **Place → Wire → Begin → End** command to add connections between the bus entry elements and the pins on the ORBLK2 symbols.
2. Similarly, make the connections for the ANDBLK2 symbol.

Placing a Junction Symbol

If a bus or wire is drawn so that it overlaps or touches another bus or wire, the two signals *are not* connected. At any location where two buses or wires meet in a T-junction the connection must be indicated with a junction symbol.

Where the Q[3..0] bus enters the schematic, it branches in two directions in a T junction. However, as the schematic now stands, no connection actually exists between the buses.

Add a junction symbol to the Q[3..0] bus as shown in Figure 11-9.

1. Select **Place → Junction**.

Keyboard Shortcut: **Place → Junction** corresponds to the Key Macro "Ctrl-F2." You can perform step 1 by holding down the **Control** key and pressing the **F2** function key,

2. Position the cursor at the junction of the buses, at location 2.30, 1.00, and select **Place**.

   A blue square appears at the junction of the two buses, completing the connection.

3. Press **Escape** to exit from the `Place Junction` command.

Placing Labels

Assigning a name to every signal net is highly recommended, because it greatly simplifies debugging later in the design process. Unnamed nets are automatically given names that have no meaning to you. These semi-random names are used in all error and warning messages, as well as by the simulator.

Every bus must have an indexed label. Every net broken from a bus must be labeled with the name of the bus followed by the appropriate index.

Label all wires and buses that you added to the ALU schematic.

1. Select **Place → Label**.

   The `Label?` prompt appears.

Keyboard Shortcut: **Place → Label** corresponds to the Key Macro "F7." You can perform step 1 by pressing the **F7** function key.

2. Enter the first signal name, **OR[3..0]** ↵.

   This signal name appears attached to the cursor.

3. Move the label to the appropriate signal net, which is the bus output of the ORBLK2 symbol.

   The tip of the cursor arrow must be on the bus itself, ensuring that the link between the bus and its label is recognized.

4. When the label is properly positioned, select **Place**.

   The label is placed, and the `Label?` prompt reappears.

5. Type **AND[3..0]** ↵.

6. Move the label to the output of the ANDBLK2 block and select **Place**.

7. In response to the `Label?` prompt, type **Q**0↵.

8. Move the cursor so that it points to the wire connecting the Q bus entry with the A0 pin of the ORBLK2 symbol.

9. Select **Place**.

   The Q0 label is attached to the wire and the label at the cursor changes to Q1.

   The label names increment automatically, which means that after entering a label ending with a number, the next label defaults to the same prefix with the next consecutive number. If you do not want that label, simply press the **Escape** key or the right mouse button before entering the next label.

10. Move the cursor down to the next net and select **Place** to attach the Q1 label.

11. Repeat the above steps until you have labeled all inputs to the ORBLK2 and ANDBLK2 symbols, as shown in Figure 11-9.

12. Move the mouse to the upper right until the FD4CE library symbol appears on the screen.

13. Label all wires making connections to the FD4CE symbol, as shown in Figure 11-10.

14. When all signals have been named, exit the Place Label command by pressing **Escape**.

If a signal name is entered incorrectly, you can edit it by positioning the cursor to point to the label and selecting **Edit → Edit → Name**. Correct the name and press **Escape** to leave the Edit Label command.

Keyboard Shortcut: **Edit → Edit → Name** corresponds to the Key Macro "Alt-E." You can change the name of a label or a module port by holding down the **Alt** key and typing **E**.

If a signal name is misplaced, the easiest solution is to erase it using the Delete Object command, and place it again.

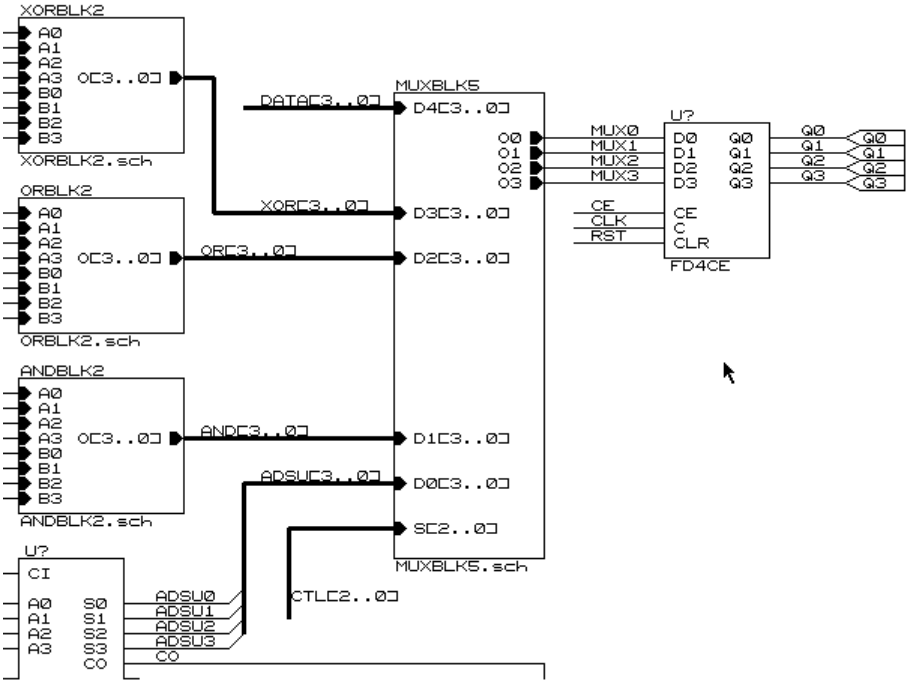


Figure 11-10 FD4CE Symbol with Labels and Module Ports

Placing Module Ports

In Figure 11-10, the symbols attached to the Q0, Q1, Q2, and Q3 signals are called module ports. They are necessary to link the signals

on the lower-level drawing to the pins on the upper-level sheet symbol. The name used for each module port must exactly match the name of the corresponding upper-level pin.

1. To place the first module port, select **Place → Module Port**.

   The `Module Port Name?` prompt appears.

Keyboard Shortcut: **Place → Module Port** corresponds to the Key Macro "F8." You can perform step 1 by pressing the **F8** function key.

2. Enter the first port name, **Q0** ↵.

   The Module Port Type menu appears.

3. Since this is an output signal, select **Output**.

   Now the module port appears attached to the cursor.

4. Move the port to the end of the Q0 signal net and select **Place**.

   The Q0 module port is established, and a new module port, Q1, appears at the cursor.

   As with label names, module port names increment automatically. If you do not want that module port name, simply press the **Escape** key or the right mouse button before entering the next module port name.

5. Place the remaining output module ports, Q1, Q2, and Q3, as shown in Figure 11-10.

6. Press **Escape** to exit from the Place Module Port command.

7. To edit a module port, use the **Edit → Edit** command, as described for signal names.

Although the Calc design uses only input and output module port types, an FPGA design can also use the bidirectional module port. Do not use any port type other than input, output, or bidirectional; it will not be understood by the translation software.

Naming Buses in OrCAD/SDT

Naming buses in OrCAD can be problematical because the naming conventions must be strictly followed. Some important points for using buses are listed here.

- The bus itself and each of its individual signals must be labeled. The bus and its component signals are linked through a common label prefix. If the individual nets are labeled D0 through D3, the bus itself must be labeled D[0..3].
- If signals without a common label prefix, such as control signals, are to be grouped on a single bus, additional labels with common prefixes must be attached to the individual signals. This technique is shown in Figure 11-11. Double labels are necessary where signals are joined together and where the bus is broken apart.
- A bus can be connected to a sheet symbol pin. The pin name must be a legal bus name, and it must exactly match both the module port name and the bus signal name on the lower-level drawing. The bus label in the upper-level drawing need not match the name at the lower level. There are several buses in the Calc design that have different names at different levels of hierarchy: one example is the SWDATA[3:0] bus in the ALU schematic, which is labeled SW[3..0] in the top-level CALC drawing.
- Bus pins are not allowed on library symbols.
- Remember, every bus on the schematic must be labeled, even if you have a module port attached to the bus and there is a legal bus name on the module port. Neglecting this rule leads to errors in the translation process.

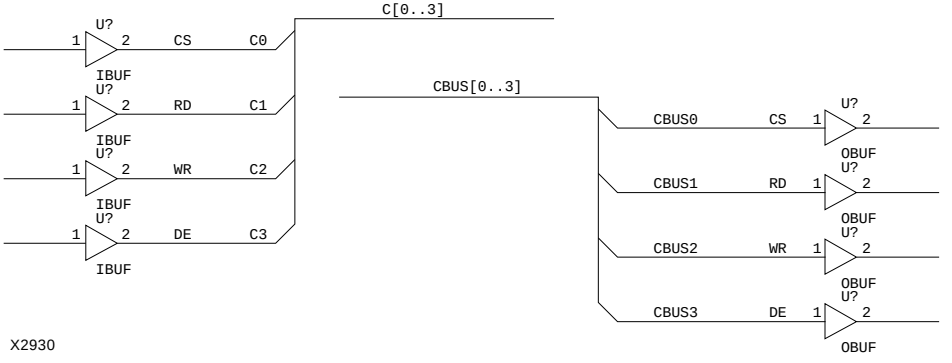


Figure 11-11 Double-Label Bus

Saving the ALU Drawing

The schematic for the ALU module is now complete and ready to be saved. The completed schematic for the ALU is shown in Figure 11-12.

1. To save the finished drawing, select **Quit → Update File**.

   This procedure stores the schematic under the name alu.sch, as the ALU sheet symbol specifies.

Keyboard Shortcut: **Quit → Update File** corresponds to the Key Macro "Shift-F6." You can perform step 1 by holding down the shift key and pressing the **F6** function key.

2. Press **Escape** to exit the Quit command.

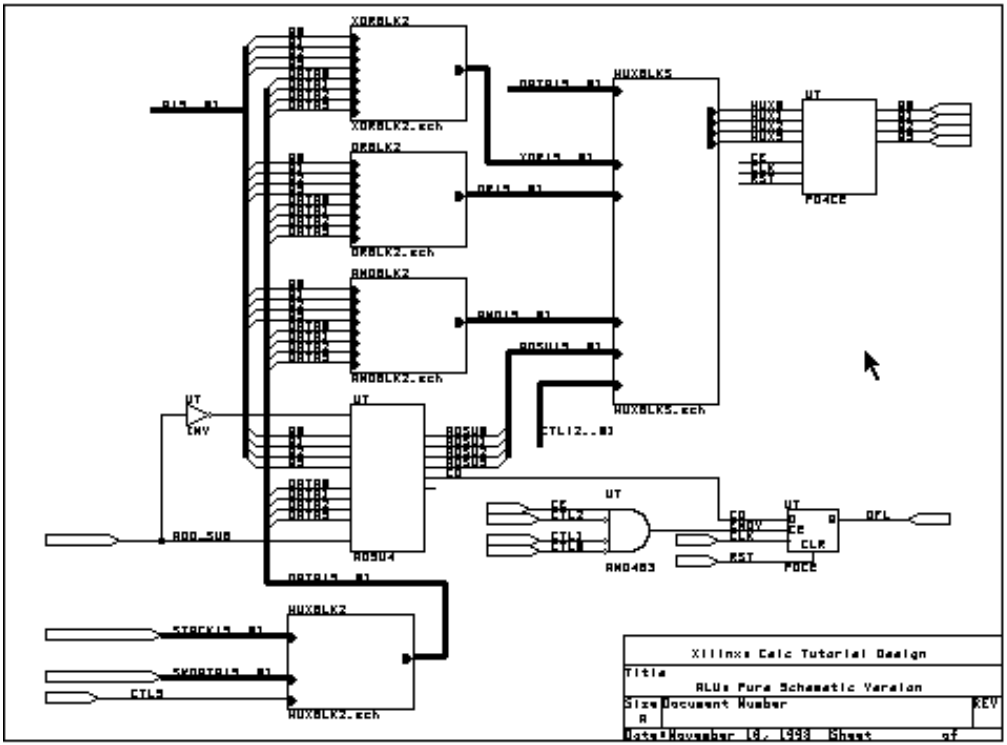


Figure 11-12 Completed Schematic for ALU

Creating the ANDBLK2 Schematic

In this section, you create a new schematic. There are currently no schematics for the ANDBLK2 and ORBLK2 blocks that you placed in the ALU. First, you create the ANDBLK2 schematic, which ANDs the bits of two 4-bit buses.

The schematic for ANDBLK2 is shown in Figure 11-13. Since the drawing for ORBLK2 is very similar to ANDBLK2, ANDBLK2 is created first and then edited to create ORBLK2.

Note: If you feel that you are already proficient in creating OrCAD schematics, you can copy the andblk2.sch file from any of the solutions directories, such as c:\xact\tutorial\calc\soln_3ka, into your project directory, and continue with the next section, "Creating the ORBLK2 Schematic."

Creating a New Schematic Sheet

Create a new worksheet for ANDBLK2.

1. Move the cursor into the ANDBLK2 sheet symbol on the ALU drawing and select **Quit → Enter Sheet → Enter**.

   The <<<New Worksheet>>> message appears briefly to indicate that a blank drawing page has been created for andblk2.sch.

2. Press **Escape**.

Placing Xilinx Library Primitives

Xilinx primitive gates are named according to their function and number of inputs, so a three-input AND gate is designated AND3. If there are inverted inputs on the gate, the letter B followed by the number of inverted inputs is appended to the name. For example, a two-input NAND gate with one inverted input is designated NAND2B1.

1. To see a list of all available Xilinx macros, select **Library → Directory.**

Keyboard Shortcut: **Library → Directory** corresponds to the Key Macro "Ctrl-F1." You can perform step 1 by holding down the **Control** key and pressing the **F1** function key.

2. Select **XC3000.LIB**, or **XC4000.LIB** if you are doing an XC4000 design.

   The `To?` prompt appears at the top of the screen.

3. Select **Screen** to send the output to the screen.

4. Select **More** and type any key to return to the schematic editor.

   Place the AND2 gates required to implement the ANDBLK2 function.

5. To load the primitive symbol for a two-input AND gate, select **Get**.

   The `Get?` prompt appears.

6. Type the Xilinx name for a two-input AND gate with no inverted inputs, **and2** ↵.

   The AND2 symbol appears attached to the cursor.

7. Position the symbol as shown in Figure 11-13 and select **Place**.

   This step places a copy of the AND2 symbol at the specified location. After the first AND2 gate has been placed, the symbol is still attached to the cursor, so that additional symbols can be placed on the drawing.

8. Press **Place** to place a second copy of the AND2 gate.

9. Press **Escape** or click the right mouse button to exit the Get command.

   You will use a different technique to place the remaining AND2 gates.

Copying Library Elements

Use the Block Save and Block Get commands to copy one of the symbols on the schematic. This command sequence seems awkward but is both quick and easy if you use the corresponding key macro. It works very well for copying any object under the cursor.

1. Place the cursor on top of either of the AND2 symbols.

2. Select **Block** → **Save** → **Begin** → **End** → **Block** → **Get**.

The AND2 symbol appears attached to the cursor.

3. Move the cursor to the correct location for the third symbol and select **Place**.
4. Move the cursor again and select **Place** to place the last AND2 gate.
5. Press **Escape** to exit the Get command.

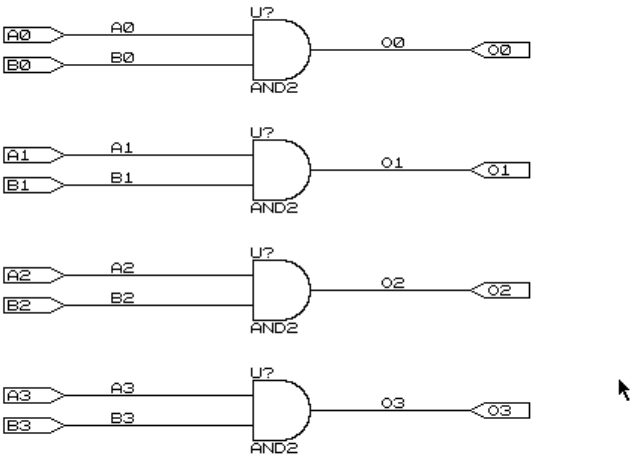


Figure 11-13 Schematic Drawing for ANDBLK2

Moving Library Elements

If you placed a symbol in the wrong location, you can move it:

1. Select **Block → Move**.

Keyboard Shortcut: **Block → Move** corresponds to the Key Macro "Ctrl-F5." You can perform step 1 by holding down the **Control** key and pressing the **F5** function key.

2. Position the cursor inside the symbol to be moved and select **Begin → End**.

This procedure attaches the symbol to the cursor so that it can be repositioned.

3. Select **Place** to establish the new location.

Experimenting with Wires and Buses

When learning to draw wires and buses in Draft, it can be helpful to experiment with the different methods.

Place Wire Begin (or the F2 function key) specifies the initial point of a wire. Place Bus Begin (or the F6 function key) specifies the initial point of a bus. Repeating the Begin key places a corner point, End places an end point on the bus or wire, and the right mouse button ends the command without adding any more segments.

1. Experiment with the Place Wire and Place Bus commands.
2. After experimenting with drawing wires and buses, erase the extra segments by placing the cursor on the bus or wire segment and selecting **Delete → Object**.

Keyboard Shortcut: **Delete → Object** corresponds to the Key Macro "F4." You can perform step 2 by pressing the **F4** function key.

3. For each additional object to be removed, position the cursor on the object and select **Delete**.
4. If Draft cannot determine which object is to be deleted, it presents an additional menu; select the appropriate object to complete the command.
5. When all unnecessary wires and buses have been removed, press **Escape** to exit from the Delete Object command.

Completing the ANDBLK2 Schematic

You have placed the components on the ANDBLK2 schematic. To complete the schematic, you need to add wires, labels, and module ports. This schematic does not require any junction symbols because there are no T connections where wires meet wires, or buses meet buses.

The completed schematic should look like the drawing in Figure 11-13.

1. Use the Place Wire command to add wires to the schematic as shown.
2. Use the Place Label command to add the following labels: A0, A1, A2, A3, B0, B1, B2, B3, O0, O1, O2, O3.
3. Add module ports to each labeled wire using the Place Module Port command.
4. Save the schematic using the Quit Update command.

Creating the ORBLK2 Schematic

Since the ORBLK2 schematic is very similar to the ANDBLK2 schematic, you can use the ANDBLK2 schematic as a template to create the ORBLK2 drawing.

Note: If you skipped the previous section, "Creating the ANDBLK2 Schematic," and copied the andblk2.sch file from a solutions directory, push into the ANDBLK2 schematic and continue with this section.

Exporting a Block

Using the Block Export command, you can save any rectangular section of an OrCAD schematic as a new drawing file. The circuit drawn for ANDBLK2 can be exported into a temporary file and then loaded back into the ORBLK2 drawing.

The difference between the Block Export command and the Block Save command, which you used in the "Copying Library Elements" section earlier in the tutorial, is that Block Export writes the information to a file, while Block Save places the information in a temporary buffer. Unless you plan to use the saved information immediately, it is safer to write it to a file. This file remains in your design directory until you delete it.

1. To save the ANDBLK2 drawing created above to a temporary file, select **Block → Export**.

Keyboard Shortcut: **Block → Export** corresponds to the Key Macro "Shift-F4." You can perform step 1 by holding down the **Shift** key and pressing the **F4** function key.

2. Position the cursor in the upper left corner of the schematic and select **Begin**.
3. Now move the cursor to the lower right corner of the drawing.

   The block definition window expands as the cursor is moved.
4. To establish this corner of the block, select **End**.

   The `Export Filename?` prompt appears.
5. Enter the name of the temporary file, **temp.tmp** ↵.

   The defined portion of the drawing is saved in the temp.tmp file.
6. Now move one level up in the hierarchy to the ALU schematic by selecting **Quit → Leave Sheet**.

   The ALU drawing returns.

Importing a Block to Create ORBLK2

You are ready to create the ORBLK2 schematic.

1. Select **Enter Sheet** from the Quit submenu on the screen.
2. Position the cursor inside the ORBLK2 sheet symbol, and select **Enter**.

   The `<<<New Worksheet>>>` message appears briefly, indicating that a blank drawing page has been created for ORBLK2.sch. The entire process of moving from ANDBLK2 to ALU to ORBLK2 was executed from within the Quit submenu; many OrCAD menus are structured this way to facilitate the rapid selection of related commands. It is often necessary to use the Escape key to exit from a command menu.
3. Press **Escape** to exit the Quit Enter Sheet command.
4. To begin the ORBLK2 schematic, load the drawing previously exported as temp.tmp by selecting **Block → Import**.

   The `File to import?` prompt appears.

Keyboard Shortcut: **Block → Import** corresponds to the Key Macro "Shift-F3." You can perform step 4 by holding down the **Shift** key and pressing the **F3** function key.

5. Type **temp.tmp** ↵.

6. Move the cursor to place the drawing near the center of the schematic sheet, and select **Place**.

   The imported drawing is loaded into the ORBLK2 schematic. The imported drawing remains highlighted, so you could place another copy of the import file, if you wished to do so.

7. Press **Escape** to exit the Block Import command.

Completing the ORBLK2 Schematic

You need to change the AND2 gates to OR2 gates, and the ORBLK2 schematic will be complete, as shown in Figure 11-14.

Use the Delete Block command to delete the AND2 gates.

1. Select **Delete → Block**.

Keyboard Shortcut: **Delete → Block** corresponds to the Key Macro "Ctrl-F4." You can perform step 1 by holding down the **Control** key and pressing the **F4** function key.

2. Position the cursor inside one of the AND2 symbols and select **Begin → End**.

   The symbol is deleted from the schematic.

3. Repeat steps 1 and 2 to delete the remaining AND2 components from the schematic.

4. Select **Get** and type **or2**↵.

5. Position the OR2 symbol to replace one of the AND2 gates and select **Place**.

6. Use **Place** to add the remaining components as in Figure 11-14.

7. Save the ORBLK2 schematic and return to the ALU schematic by selecting **Quit → Update File → Leave Sheet**.

8. Press **Escape** to exit the Quit command.

Keyboard Shortcut: **Quit → Update File → Leave Sheet** corresponds to the Key Macro "Alt-F8." You can perform steps 7 and 8 by holding down the **Alt** key and pressing the **F8** function key.

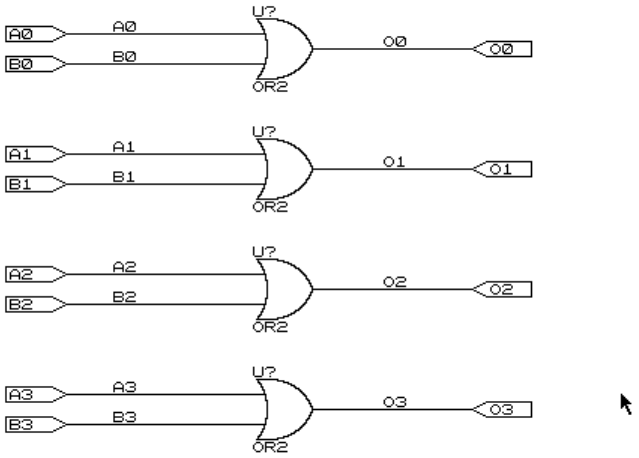


Figure 11-14 Schematic Drawing for ORBLK2

Saving a File to Another Name

This section is for your information only; no action is required.

If you wish to copy an entire schematic, rather than a block cut from a schematic, you can use the Quit Write to File command rather than the Block Save and Block Get, or Block Export and Block Import commands. The Quit Write to File command saves a schematic under a new name. For example, to save the ANDBLK2 schematic as ORBLK2, select **`Quit → Write to File`** from the ANDBLK2 schematic, and type **`orblk2.sch`** ↵.

Keyboard Shortcut: **`Quit → Write to File`** corresponds to the Key Macro "Shift-F7." You can save a schematic file to another name by holding down the **`Shift`** key and pressing the **`F7`** function key.

Alternatively, you can simply exit to DOS with the Quit Suspend to System command, and use the DOS Copy command to copy the file to the new name. After copying the file, return to the OrCAD environment by typing **`exit`** ↵.

Keyboard Shortcut: **Quit → Suspend to System** corresponds to the Key Macro "Shift-F9." You can open a DOS shell by holding down the **Shift** key and pressing the **F9** function key.

Exploring Xilinx Library Elements

The Xilinx libraries contain three types of elements. Primitives are basic logical elements such as the AND2 and OR2 gates that you previously placed in ANDBLK2 and ORBLK2. Soft macros are schematics made by combining primitives and sometimes other soft macros into a schematic. Relationally placed macros (RPMs) are soft macros that contain placement information and sometimes carry-logic elements. RPMs are currently only available in the XC4000 library.

All three types of library elements are placed on a schematic in exactly the same way. However, while soft macros and RPMs have associated schematics, primitives do not.

Viewing a Xilinx Soft Macro Schematic

Soft macro schematics are just schematics such as you might make for your own designs. In fact, you can load one of these schematics and use the Quit Write to File command (or just use DOS Copy) to save the schematic under another name, then edit this new schematic to customize it to your needs.

You cannot use the Quit Enter Sheet command to view the Xilinx library schematics, because SDT does not know where to look for them. However, you can load the schematic for any soft macro in the library by specifying the full path to the schematic supplied by Xilinx.

Load the FD4CE schematic.

1. Select **Quit → Initialize** from the menu.

   The `Abandon Hierarchy?` prompt appears.

2. Select **Yes**, since you saved all of your edits each time you changed levels of hierarchy.

   The `Load file?` prompt appears.

3. Type **c:\xact\xc3000\fd4ce.sch** ↵. If you are targeting this tutorial to an XC4000 design, load the schematic from the XC4000 library instead.

   The schematic for FD4CE replaces the ALU schematic on the screen.

4. If necessary, select **Zoom → Out** to view the entire schematic.

Keyboard Shortcut: **Zoom → Out** corresponds to the "PgUp" key or the Key Macro "Alt-F6." You can perform step 4 by pressing the **PgUp** key or by holding down the **Alt** key and pressing the **F6** function key.

As shown in Figure 11-15, FD4CE was created from four placements of FDCE.

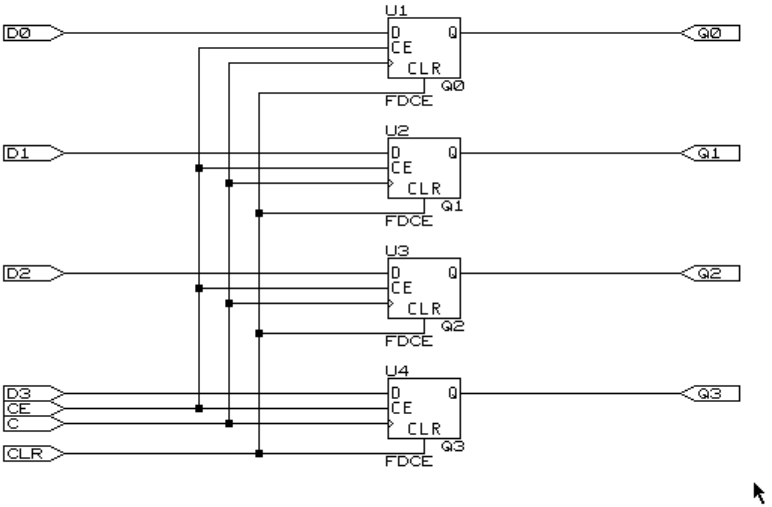


Figure 11-15 FD4CE Schematic From XC3000 Library

Viewing a Xilinx RPM (XC4000 Family Only)

The ALU contains a component from the Xilinx library called ADSU4, which is a 4-bit wide adder/subtracter. If your design is reading components from the XC4000 library, this schematic is implemented as a relationally placed macro (RPM). If not, ADSU4 is implemented with standard logic gates and multiplexers.

As with soft macros, RPM schematics are just schematics such as you might create for your own designs. You can save an RPM schematic under another name, then edit this new schematic to customize it to your needs.

If your design is targeted towards the XC4000 family, load the ADSU4 schematic. If you are targeting an XC3000 family part, you will not be able to bring up the XC4000 schematic successfully. Read through the text while looking at the schematic for the RPM depicted in Figure 11-16, but do not perform any of the commands in this section. Continue the tutorial with the next section, "Returning to the CALC Schematic."

1. Select **`Quit → Initialize`** from the menu.

   The `Load file?` prompt appears.

2. Type **`c:\xact\xc4000\adsu4.sch`** ↵.

   The schematic for ADSU4 replaces the FD4CE schematic on the screen.

3. Use the **`Zoom → In`** command to zoom into the upper portion of the schematic as shown in Figure 11-16.

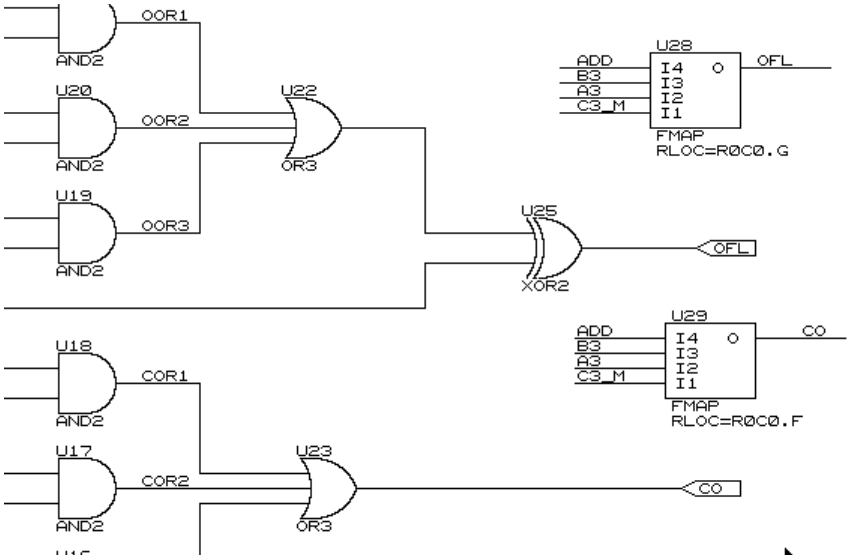


Figure 11-16 Zooming In on the ADSU4 RPM Schematic

Elements placed in the RPM schematic include CY4 components and FMAPs. The CY4 symbol allows schematic access to the fast carry logic that is one of the features of the XC4000 family devices. The FMAPs map logical functions to the function generators in the CLBs. Both CY4 symbols and FMAP symbols have RLOC attributes attached to them; RLOCs are attributes attached to the symbols that assign relative locations to the CLBs. You can use CY4 and other carry symbols, as well as FMAPs and other mapping components, in your own schematics. For a description of each of these components, see the *XACT Libraries Guide*.

Examine the attributes attached to the top FMAP component.

4. Place the mouse on the top FMAP component.
5. Select **Edit → Edit → Options_1 → Name**.

   The `OPTIONS_1? RLOC=R0C0.G` prompt appears at the top of the screen, indicating that this function is mapped to the G function generator of the upper-left corner CLB in the RPM. RPM origins are in the upper left-hand corner.

6. Press **Escape** three times to cancel the Edit command.
7. Use the mouse to pan around the schematic looking at the RLOCs.

   As you can see, logic is mapped to three different CLBs, designated as R0C0, R1C0, and R2C0. Therefore, this RPM uses three CLBs that are arranged in a column. Information as to the number of CLBs used and the shape of the logic block is readily available for each RPM in the *XACT Libraries Guide*.

Returning to the CALC Schematic

Reload the CALC schematic to investigate some other features of the Xilinx FPGA architecture.

1. Select **Quit → Initialize** from the menu.

   The `Load file?` prompt appears.
2. Type **calc** ↵.

   The CALC schematic appears on the screen.

Using the XC3000 Oscillator (XC3000 Family Only)

The FPGA (XC3000 / XC4000) and XC3000 demonstration boards have a built-in RC circuit for clock generation for the XC3000 family part. The OSC_3K block contains an oscillator that connects to that circuit. The frequency of the output varies with processing, so it is not suitable for many applications, but it is adequate for clocking a human interface.

If your design is targeted towards the XC3000 family, push into the OSC_3K schematic. If not, read through the text while looking at the schematic for OSC_3K depicted in Figure 11-17, but do not perform any of the commands in this section. Continue the tutorial with the next section, “Using the XC4000 Oscillator.”

1. Place the mouse on the OSC_3K symbol at the lower left corner of the Calc schematic.
2. Select **Quit → Enter Sheet → Enter** to push into the OSC_3K schematic.

   The OSC_3K schematic appears as shown in Figure 11-17.

   The output of the oscillator circuit in OSC_3K is routed through a

global clock buffer before being passed to the rest of the device. There are several reasons for using the global buffers. The global clock buffers drive dedicated routing resources that can reach any clock pin in the device very quickly with minimal skew. In addition, using the dedicated clock nets frees up programmable interconnect for use by other signals in the design.

The GCLK symbol at the right side of the OSC_3K schematic is the Xilinx primitive for the XC3000 global clock buffer. The equivalent primitive for the alternate clock buffer is named ACLK. The ACLK buffer is used in this design to drive the divide-down curcuit on the clock, although it is not really necessary.

In general, you should use at least one clock buffer (GCLK or ACLK) in every clocked XC3000 or XC2000 design. Use GCLK for the highest-priority clock net, that is, the largest fanout or fastest clock net, and ACLK for the second-highest-priority clock.

3. Select **Leave** to return to the CALC schematic.
4. Press the **Escape** key or click the right mouse button to exit the Edit command.

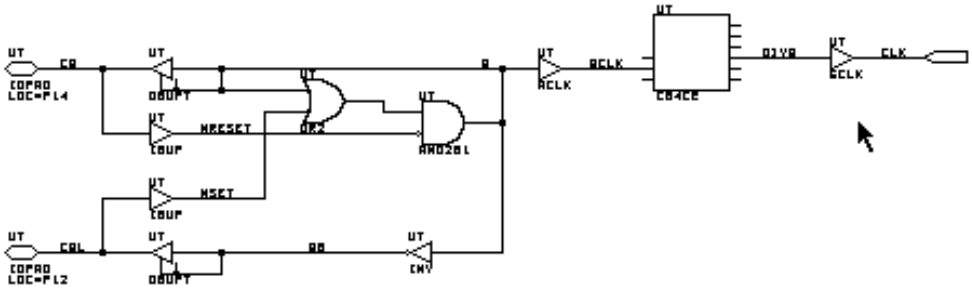


Figure 11-17 OSC_3K Schematic

Using the XC4000 Oscillator (XC4000 Family Only)

The XC4000 family devices include an on-chip oscillator, which makes it unnecessary to use an external RC circuit. Therefore, no such circuit is built into the demonstration boards for the XC4000 family parts. If your design is targeted to an XC4000 family device, you must exchange the OSC_3K component for one that calls the XC4000 family oscillator and the XC4000 family clock buffers.

If your design is targeted towards the XC4000 family, continue with the commands in this section. If not, read through the text while looking at the schematic for OSC_4K depicted in Figure 11-18, but do not perform any of the commands in this section. Continue the tutorial with the next section, "Inverting Output Display Signals."

1. Place the mouse on the OSC_3K symbol at the lower left corner of the Calc schematic.
2. Select **Edit → Edit → Filename**.

   The `Filename? OSC_3K.sch` prompt appears at the top of the screen.
3. Backspace to delete "ORC_3K.sch" and type **orc_4k.sch**↵ to change the name of the referenced schematic.
4. Press the **Escape** key twice or click the right mouse button twice to exit the Edit command.
5. Select **Quit → Update File → Enter Sheet → Enter** to push into the OSC_4K schematic.

   The OSC_4K schematic appears on the screen. The schematic is shown in Figure 11-18. The OSC_4K schematic contains a placement of the XC4000 library symbol called OSC4. This symbol provides access to an on-chip oscillator that generates clock frequencies of a nominal 8 MHz, 500 KHz, 16 KHz, 490 Hz, and 15 Hz. These frequencies are approximate, since they are process-dependent. The Calc design uses the 15-Hz output from this component when targeted for XC4000 family designs.

   The clock output from OSC4 is buffered through a BUFGS global clock buffer.

   Members of the XC4000 family have eight on-chip clock buffers — one BUFGP and one BUFGS in each corner of the device. The Calc

design uses a BUFGS because the clock is generated internally. BUFGP inputs must come from pads. Although an internally generated clock can be routed out through an output buffer and back in through a BUFGP, it is better to use a BUFGS for internal signals. Also, never use a BUFGP for any signal other than a clock. You can use a BUFGS for a clock enable or flip-flop reset signal, as long as you do it sparingly. See the *XACT Libraries Guide* and the *The Programmable Logic Data Book* for more information on global clock buffers for Xilinx devices.

6. Return to the top-level CALC schematic by selecting **Leave**.
7. Press the **Escape** key or click the right mouse button to exit the Edit command.

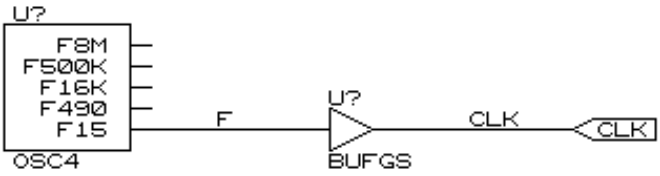


Figure 11-18 OSC_4K Schematic

Inverting Output Display Signals (XC3000 Demonstration Board Only)

The FPGA (XC3000/XC4000) demonstration board and the XC4000 demonstration board are both designed such that a Low signal on an output turns on the display element. Therefore the blocks controlling the displays, 7SEG_INV and LED_INV, both contain inverters before each output buffer. The CALC schematic is set up for this configuration as the default.

The XC3000 demonstration board is designed in such a way that a display element is turned on when the signal is High. In addition, two pin connections are reversed on this board, LDC and HDC. The inverters must be removed and the pin locations reversed if you plan to download to an XC3000 demonstration board. This board has only

a single FPGA socket and the socket contains an XC3000 family part in a PC68 package.

If you plan to download to an XC3000 demonstration board, continue with the commands in this section. If not, skip to the next section, "Controlling FPGA Layout from the Schematic."

To remove the output inverters and reverse the pin connections, simply replace 7SEG_INV and LED_INV with 7SEG_TRU and LED_TRU in the CALC schematic.

1. Place the mouse on the 7SEG_INV symbol.
2. Select **Edit → Edit → Filename**.

   The `Filename? 7SEG_INV.sch` prompt appears at the top of the screen.
3. Backspace to delete "7SEG_INV.sch" and type **7SEG_TRU.SCH**↵ to change the name of the referenced schematic.
4. Press the **Escape** key twice or click the right mouse button twice to exit the Edit command.
5. Repeat steps 1 through 4 to change the file name of the LED driver from LED_INV.sch to LED_TRU.SCH.

Controlling FPGA Layout from the Schematic

You can use attributes in the schematics to control the placement and routing of your design. One of these attributes enables you to specify the part type on your schematic.

Specifying the Part Type

You can specify the part type on the schematic, in XDM, or by controlling program defaults. The safest method is to specify it on the schematic, because that way the part type is documented; also, the translation software issues a warning if you accidentally override this part type designation using one of the other two methods.

To specify the part type on the schematic, add text to the top-level schematic.

1. Select **Place → Text**.

   The `Text?` prompt appears at the top of the screen.

Keyboard Shortcut: **Place → Text** corresponds to the Key Macro "Ctrl-F9." You can perform step 1 by holding down the **Control** key and pressing the **F9** function key.

2. Type **|parttype=** and the name of the part on your demo board. For example, if you are targeting the 3020APC68, type **|parttype=3020apc68-7** ↵.

Note: The first character you type *must* be a vertical bar. Without the vertical bar, the "parttype" attribute is not passed to the Xilinx software.

An outline of the text appears at the cursor.

3. Select **Larger** to increase the size of the text and make it more readable.
4. Move the cursor to the bottom right-hand corner of the schematic above the title box.
5. Select **Place** to place the text.
6. Click the right mouse button to cancel the Text command.

Other devices commonly found in Xilinx demonstration boards are the 3020pc68-50, 4003apc84-6, and 4003pc84-6. The speed grade can be included or omitted. If omitted, a default speed grade is used. Which speed grade you select affects the timing of the device, both in routing and during timing simulations.

Assigning Pin Locations

You can also assign pin locations to your pads or input/output buffers on the schematic. It is highly recommended that you let the automatic placing and routing programs, PPR and APR, define the pinout, because locking the I/Os can restrict the programs from placing and routing the design completely.

However, I/O pin numbers are assigned to the tutorial schematics so that the Calc design functions in the Xilinx demonstration boards. (Later in the tutorial you learn how to override these pin assignments when using an XC4000 family device.) Because the design is fairly simple, these pin assignments do not adversely affect the ability of PPR or APR to place and route the design completely.

Assign a pin location to the EXC_P pin on the CALC schematic.

1. Move the cursor onto the I/O pad symbol connected to EXC_P.
2. For an XC2000 or XC3000 family design, select **Edit → Edit → Loc, options → Name.** For an XC4000 family design, select **Edit → Edit → Options_1 → Name.**
3. At the `Loc, options?` or `Options_1?` prompt, type **LOC=P11** ↵.

   The location is displayed under the pad symbol.

Note: The Options_1 and Options_2 fields are interchangeable for XC4000 family designs. XC2000 and XC3000 family designs have only a single LOC,Options field.

4. Press the **Escape** key twice or click the right mouse button twice to exit the Edit command.

Valid pin locations vary depending on the package. PLCC package pins are designated with a P followed by the pin number, such as P17. Pin grid array (PGA) package pins use alphanumerics such as A12. *The Programmable Logic Data Book* lists the pinouts of each FPGA for each package that Xilinx supplies.

Adding Net Attributes

Suppose you want to make sure that a net in your design is not absorbed into a CLB, to facilitate simulation or for some other reason. You can attach an "Explicit" (or "External") flag to the net.

Net flags are placed on the schematic with the Get command, just like any other symbol.

1. Select **Get**.

   The `Get?` prompt appears at the top of the screen.

2. Type **x** ↵.
3. Move the cursor so that the new symbol is positioned above the net labeled "A." between 7SEGDEC and 7SEG_INV or 7SEG_TRU, as in Figure 11-19.

4. Select **Place** and click the right mouse button to terminate the command.
5. Use the Place Wire Begin End command to add a wire between the "X" flag and the net labelled "A".
6. Use the Place Junction Place command to add junction to the intersection of the two wires.

   The wire and the junction symbol *must* be used or the net flag is not considered connected. See Figure 11-19.

There are several other net attributes. For a description of available attributes, along with a discussion of when to use them and, more importantly, when not to use them, see the "Attributes and Constraints" section of the *XACT Libraries Guide.*

It is easy to overconstrain a design. Some constraints in a schematic may prevent the software from doing the best possible job. First try to route a design with no constraints at all, then go back and add constraints only if necessary.

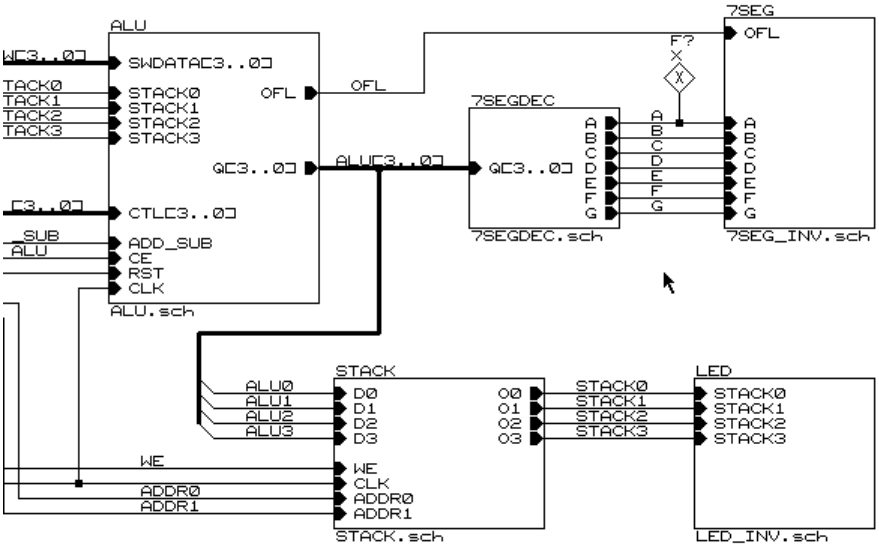


Figure 11-19 Net Flag on Net "A"

Designating FAST Pads

Output slew rate can be modified by assigning a FAST attribute to the pad or output buffer. The default slew rate is SLOW. "Fast" pads have different timing specifications from "slow," or slew-rate-limited, pads, and draw more current. Slew-rate-limited pads are used by default.

Add a FAST attribute to the OFL_P pad in the 7SEG_INV or 7SEG_TRU schematic under CALC.

1. Place the mouse on the 7SEG_INV or 7SEG_TRU symbol at the upper right corner of the Calc schematic.
2. Select **Quit → Update → Enter Sheet → Enter** to push into the 7SEG_INV or 7SEG_TRU schematic.
3. Click the right mouse button to exit the Quit Enter Sheet command.
4. Place the mouse on the topmost pad, labelled OFL_P.
5. For an XC2000 or XC3000 family design, select **Edit → Edit → Loc, options → Name**. For an XC4000 family design, select **Edit → Edit → Options_1 → Name**.
6. The `Loc, options? loc=P28` or `Options_1? loc=P28` prompt appears.
7. Add the FAST attribute to the existing attribute list by typing:

   `,fast` ↵.

   The leading comma is necessary to separate the two attributes.

   The Name field `loc=p11,fast` is displayed under the pad symbol, as shown in Figure 11-20.
8. Click the right mouse button twice to cancel the Edit command.
9. Select **Quit → Update → Leave Sheet** to save your changes and return to the CALC schematic.
10. Click the right mouse button to cancel the Quit command.

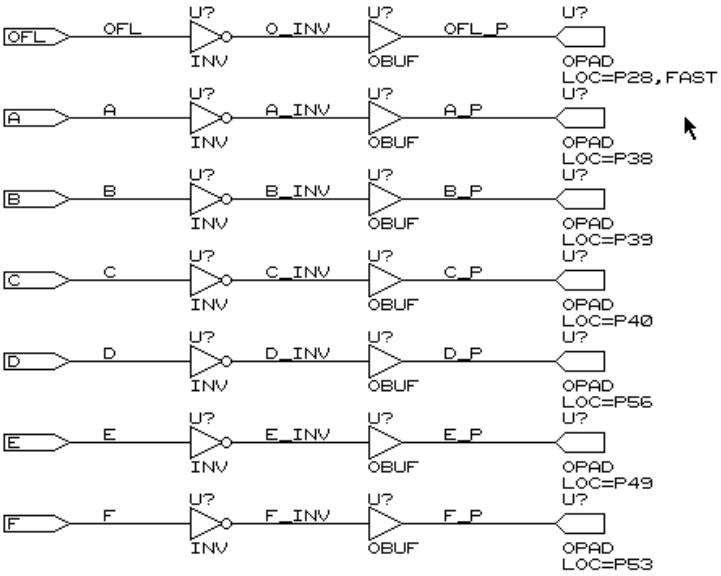


Figure 11-20 Designating a FAST Pad

Using the I/O Flip-Flops

Xilinx XC3000 family devices, XC4000 and XC4000A devices have two flip-flops in each IOB (I/O block). Each pad has an associated input flip-flop and output flip-flop. You can also configure input flip-flops as latches and can make output flip-flops 3-state. You access these I/O memory elements using special library components called IFD, ILD, OFD, and OFDT. For more information on these library elements, consult the *XACT Libraries Guide*.

Using the IOB flip-flops frees up internal CLB resources, which can be a help when designs are large in proportion to the device size or are difficult to route.

The SW7 schematic uses input flip-flops to latch data from the switches. Push into the SW7 schematic.

1. Place the mouse on the SW7 symbol at the bottom of the CALC schematic.
2. Select **Quit** → **Enter Sheet** → **Enter** to push into the SW7 schematic.

   The SW7 schematic appears as shown in Figure 11-21.
3. Pan around the schematic; observe that the IFD library element is used to access the input flip-flops.
4. Select **Leave** to return to the CALC schematic.
5. Click the right mouse button to exit the Quit command.

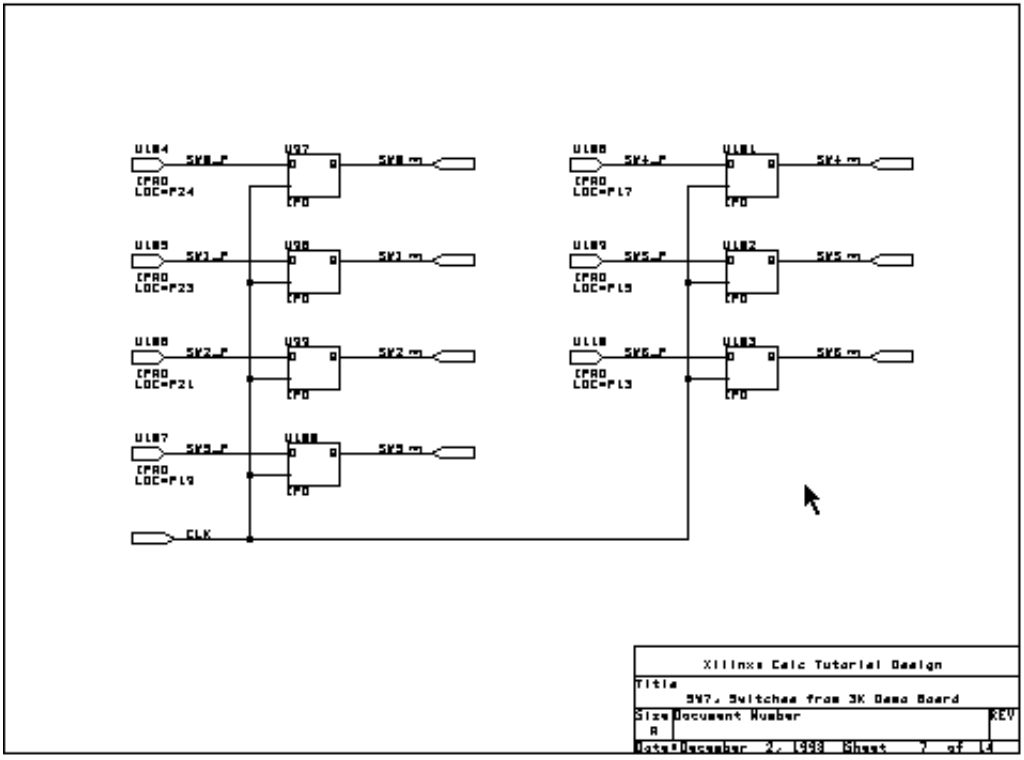


Figure 11-21 SW7 Schematic Using Input Flip-Flops

Editing the Design for the XC4000 Family

You have created or edited four schematic files, CALC, ALU, ANDBLK2, and ORBLK2. The rest of the schematics were copied to your disk during the tutorial installation. If you are targeting the design to any of the XC3000 family parts, the schematics are complete.

The design as it stands is perfectly suitable for use in an XC4000 family device. No changes need be made to adapt it further to the XC4000 family. However, XC4000 family devices have several advanced features that are not being used by this device-independent design. One of these features is the on-chip memory built into the XC4000 CLB. Another is the wide-edge decoder structures.

Device-Independent Stack Implementation

The device-independent stack is implemented as a set of registers and muxes. This implementation is independent of chip architecture and can be used for any Xilinx device.

View the stack as implemented in the device-independent design.

1. Place the mouse on the STACK symbol in the CALC schematic.
2. Select **Quit → Enter Sheet → Enter** to push into the STACK schematic.

   The schematic for STACK replaces the CALC schematic on the screen, as shown in Figure 11-21.

   There are 16 flip-flops in the block. Therefore, the most efficient implementation of this logic uses a minimum of eight CLBs.
3. Select **Leave** to return to the CALC schematic.
4. Click the right mouse button to exit the Quit command.

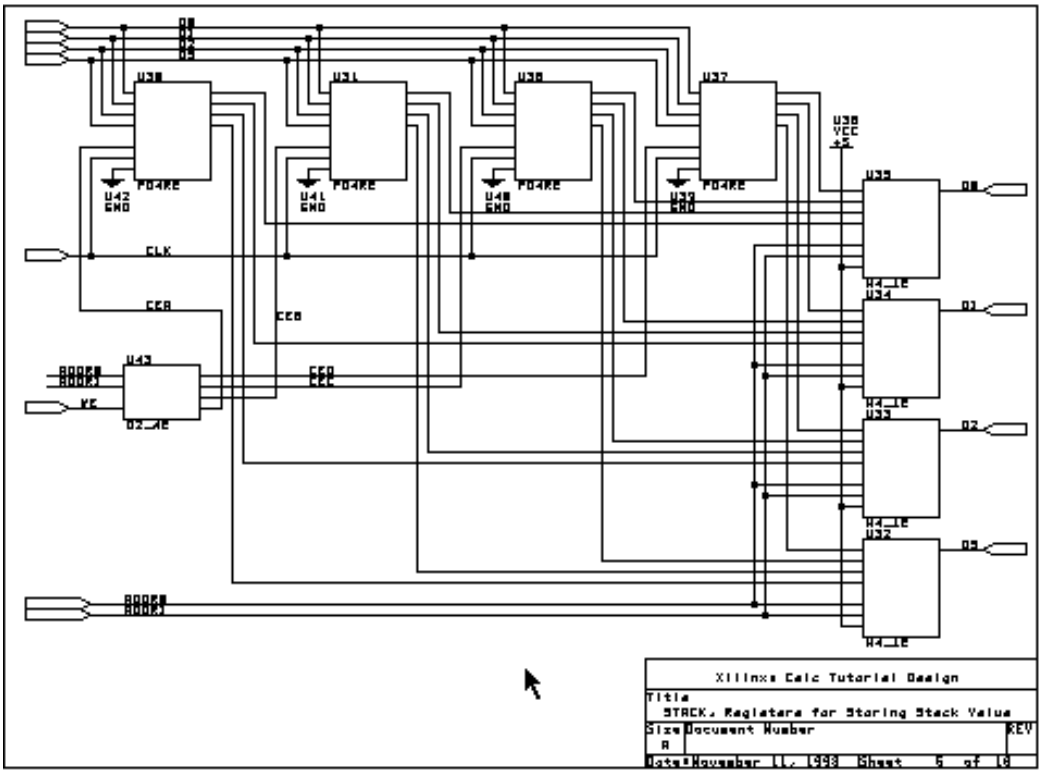


Figure 11-22 Device-Independent Stack Implementation

RAM Stack Implementation (XC4000 Family Only)

Using the on-chip static memory feature of the XC4000 family CLB, you can reduce the stack from eight CLBs to only two CLBs. If you are not doing an XC4000 family design, skip this section and continue with the “Device-Independent State Machine” section, following.

Replace the device independent stack with the XC4000 family RAM implementation.

1. Place the mouse on the STACK symbol.
2. Select **Edit → Edit → Filename**.

The `Filename? STACK.sch` prompt appears at the top of the screen.

3. Backspace to delete "STACK.sch" and type **`stack_4k.sch`**↵ to change the name of the referenced schematic.
4. Press the **`Escape`** key twice or click the right mouse button twice to exit the Edit command.
5. Select **`Quit`** → **`Update File`** → **`Enter`** **`Sheet`** → **`Enter`** to push into the STACK_4K schematic.

   The schematic for STACK_4K replaces the CALC schematic on the screen, as shown in Figure 11-23.

   The RAM stack is implemented by placing a single element from the XC4000 library. Although the stack is only 4x4, RAM and ROM are only available in 16x1 or 32x1 increments, so only one fourth of this memory is used. You could implement a stack four times as deep while still using only two CLBs. An equivalent flip-flop implementation would require 64 flip-flops or 32 CLBs.

   With the RAM implementation, a clock is not needed in the stack. However, a sheet symbol in OrCAD may not have a pin that is left unconnected, so the CLK net is used to drive a buffer and the output is left hanging. This buffer is later removed by the software when is it found to be unused. Alternatively, you could edit the CALC schematic to remove the CLK pin from the STACK_4K sheet symbol.
6. Select **`Leave`** to return to the CALC schematic.
7. Click the right mouse button to exit the Quit command.

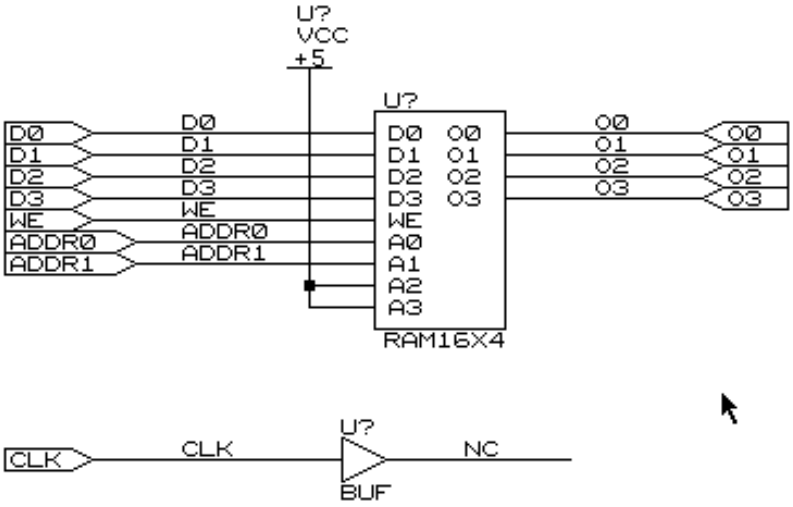


Figure 11-23 XC4000 Family RAM Stack Implementation

Device-Independent State Machine

The device-independent control logic is implemented as a set of flip-flops forming a three-state one-hot state machine, and some additional control signals made from logic gates and muxes. This implementation is independent of chip architecture and can be used for any Xilinx device.

View the state machine and control logic as implemented in the device-independent design.

1. Place the mouse on the CONTROL symbol in the CALC schematic.
2. Select **Quit** → **Enter Sheet** → **Enter** to push into the CONTROL schematic.

   The CONTROL schematic appears on the screen.
3. Place the mouse on the STATMACH symbol in the CONTROL schematic.
4. Select **Enter** to push into the STATMACH schematic.

5. Click the right mouse button to exit the Quit Enter Sheet command.

   The schematic for STATMACH replaces the CALC schematic on the screen, as shown in Figure 11-24.

   Observe that the signals INT1, INT2, RST, SEL_OP, and UP_DN are all just ANDs of opcode inputs and the EXC signal.

6. Select **`Quit → Leave Sheet → Leave Sheet`** to return to the CALC schematic.

7. Click the right mouse button to exit the Quit command.

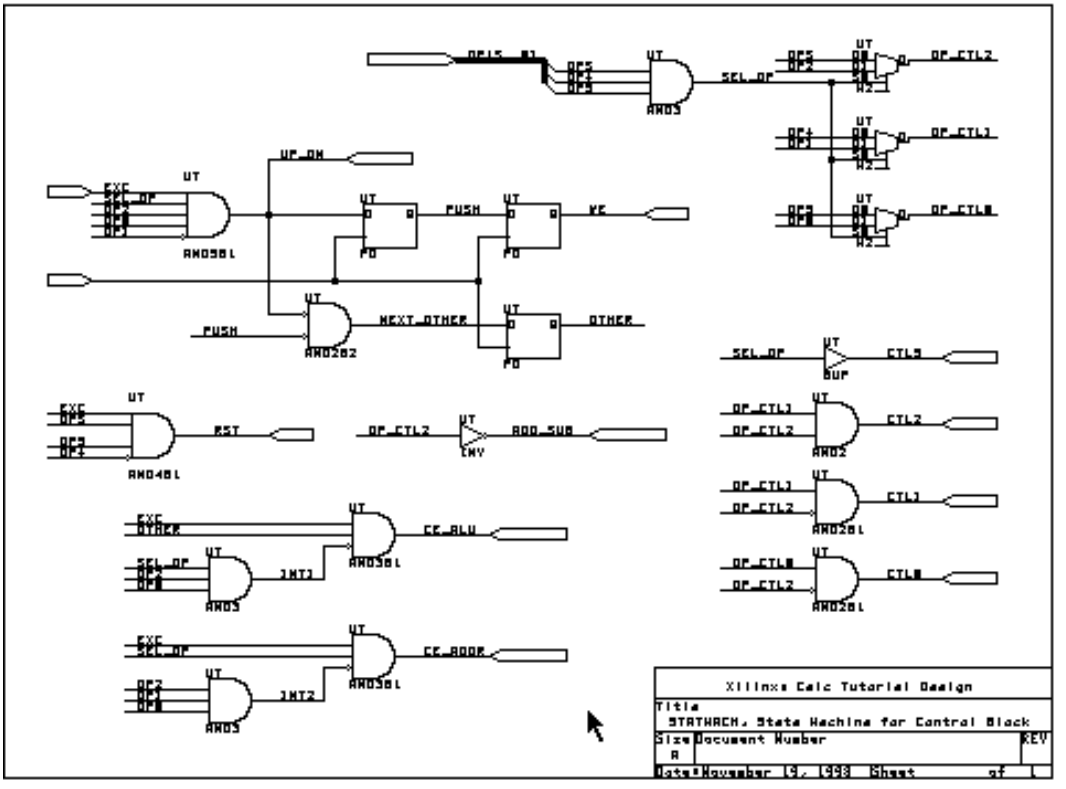


Figure 11-24 Device-Independent State Machine Implementation

State Machine with Wide-Edge Decoders (XC4000 Family Only)

XC4000 devices have four wide-edge decoder lines on each edge of the device. XC4000A devices have two decoder lines per edge. Each decoder can accept as input all pads on that side of the device and also internal nets.

For this design, all opcode pins are on the left side of the device in the 4003A and 4003 parts to which this tutorial is targeted. EXC is an internal net. You can therefore implement four of the five signals in the STATMACH schematic as wide-edge decoders. For the XC4003A device, two of the decoders must be implemented using decoders from adjacent edges. Therefore, this implementation is not very efficient for the 4003A device. However, it is included in this tutorial to demonstrate the use of wide-edge decoders in an XC4000 family design.

Normally, you should not use a wide-edge decoder to implement an AND gate unless there are at least ten inputs to the gate, since a CLB implementation is faster and routes more easily for an AND function with nine or fewer inputs. When using a wide-edge decoder, you should make sure that the pads associated with the inputs are all on one edge of the device to make the best use of this feature.

Although none of the decoders in this simple tutorial design have as many as ten inputs, the control logic in the STATMACH block is implemented using wide-edge decoders as an example. This schematic is called STATE_4K.

If you are not doing an XC4000 family design, skip this section and continue with the “Checking Schematics” section, following.

Replace the device independent state machine with the XC4000 family wide-edge decoder implementation. The state machine is placed in the CONTROL block.

1. Place the mouse on the CONTROL symbol.
2. Select **`Quit`** → **`Enter Sheet`** → **`Enter`** to push into the CONTROL schematic.

   The CONTROL schematic appears on the screen.

3. Click the right mouse button to exit the Quit Enter Sheet command.
4. Place the mouse on the STATMACH symbol.
5. Select **Edit → Edit → Filename**.

   The Filename? STATMACH.sch prompt appears at the top of the screen.
6. Backspace to delete "STATMACH.sch" and type **state_4k.sch**↵ to change the name of the referenced schematic.
7. Press the **Escape** key twice or click the right mouse button twice to exit the Edit command.
8. Select **Quit → Update File → Enter Sheet → Enter** to push into the STATE_4K schematic.

   The schematic for STATE_4K replaces the CONTROL schematic on the screen, as shown in Figure 11-23.
9. Move the mouse within the STATE_4K schematic and observe the new components that have been added to implement the wide-edge decoders.

   The decode logic is implemented by placing elements called DECODE4, DECODE8, and so forth. A PULLUP symbol is connected to the output. Tie unused inputs to a VCC symbol.
10. Select **Leave → Leave** to return to the CALC schematic.
11. Click the right mouse button to exit the Quit command.

See *The Programmable Logic Data Book* for additional information about wide-edge decoders in XC4000 family devices.

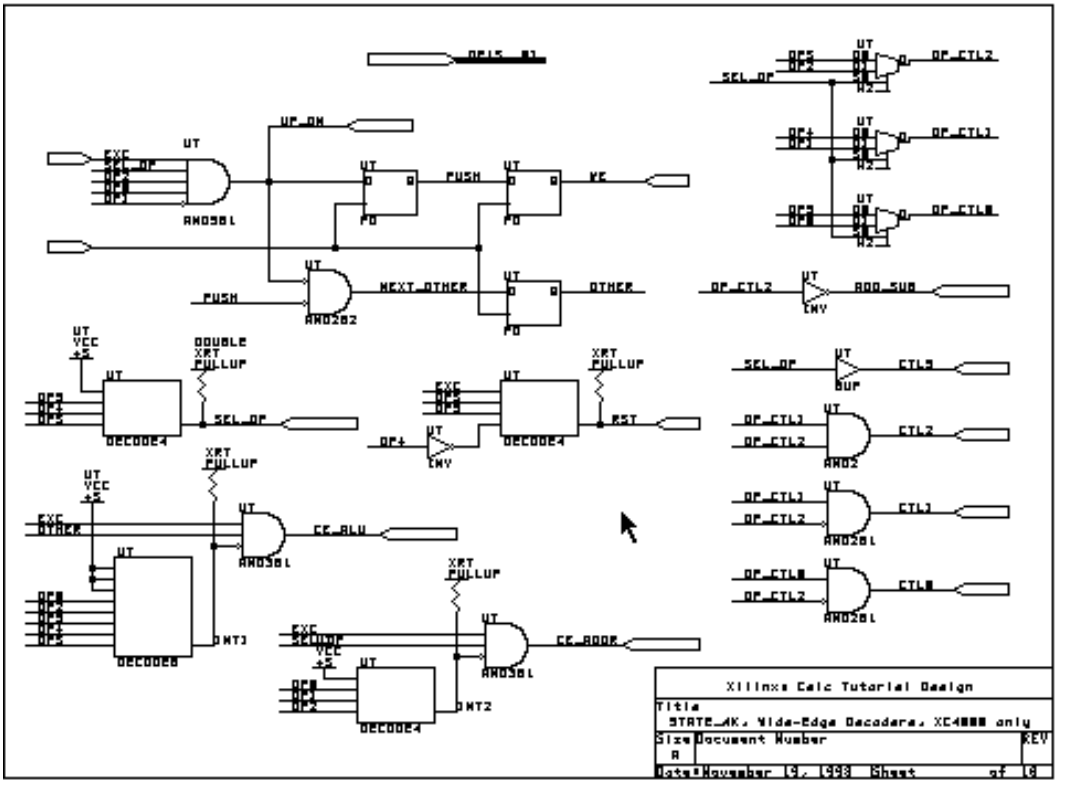


Figure 11-25 XC4000 State Machine with Wide-Edge Decoders

Checking Schematics

The schematic entry phase of the design process is now complete, but before beginning the implementation phase, you should check your drawing. The two main things that cause errors at this stage are schematic connections that look like they are there but are not, and naming problems, such as inconsistent names between nets and module ports, and names missing from buses.

- Check the wiring on the different schematic pages, making certain that all junction symbols are correctly placed and that all connections are formed properly. One way to do this is to position

the cursor over each block in turn and use the **Block Drag Begin End** command on each. Drag it a little way to make sure all the connections stay with it, then terminate the command by pressing the right mouse button. Wire-to-block connections in OrCAD do not actually make a connection unless they are positioned correctly. For example, a wire must end at the end of a block pin, not overlap it. Set Drag Buses to Yes to enable dragging buses as well as wires.

Keyboard Shortcut: **Block → Drag → Begin → End** corresponds to the "F5" function key. You can perform this command by pressing the **F5** function key.

- Make certain there is continuity between the pin names on upper-level sheet symbols and the corresponding module port names on lower-level drawings. If the names are not exactly the same, you cannot process your design without errors.
- Ensure that all labels are placed correctly. Remember that the wire should cross the bottom left corner of the attached label where the cursor point was attached.
- For a new design, check to make sure you have used one or more of the global clock buffers, where appropriate.

To check the schematics, move between the various drawings using the Enter Sheet Enter and Leave commands.

Exiting from SDT

When the schematics are correct, save the design and return to XDM.

1. Select **Quit → Update File → Abandon Edits**.

Keyboard Shortcut: **Quit → Update File → Abandon Edits** corresponds to the Key Macro "Shift-F10." You can perform step 1 by holding down the **Shift** key and pressing the **F10** function key.

2. Select **To Main → Execute → Exit ESP → Execute**.
3. Type any key to return to XDM.

The design entry phase is now complete.

Configuring XDM

The XACT Design Manager XMake program automates the translation portion of the Xilinx design flow, making the processing of a complex design as simple as running one program. Given the name of the top-level schematic drawing, XMake can find and process all lower-level drawings. It produces an LCA file that is placed and routed, as well as a bit file ready for downloading to an FPGA. It does this by creating and running a "makefile" — in this case, calc.mak — that calls a series of subprograms. You can also run these programs individually, either from XDM or at the command line, but it is simpler just to run the single program, XMake. See the "Command Summaries" section at the end of this chapter for an equivalent list of programs that you can run individually if you wish to do so. Each program is covered at length in the *XACT Reference Guide* or in the other chapters of this manual.

1. You should already be in XDM. If you are not in XDM, activate it by typing **xdm** ↵ from the DOS prompt.
2. Verify that the directory in XDM is set to \orcad\calc. The directory is displayed at the lower left corner of the XDM screen. The directory should be set to \orcad\calc because you were in the \orcad\calc directory when you brought up XDM.

   If the directory is not set correctly, click on **Directory:** and use the left mouse button to change the directory selection.
3. Click on **Family:** at the lower left corner of the screen.

   A menu of available families appears. If a device family does not show up in XDM, it means that your installed software does not support that family.
4. Click on the family that you are targeting in this tutorial.

   A list of available parts within the selected family appears on the screen. If a given part type does not show up in XDM, it means that your installed software does not support that device.

   Since the Calc design already has the part type specified on the schematic, you do not want to specify a part type in XDM.
5. If InDesign does not appear at the top of the list, scroll the list using the scrollbar until the top of the list is visible.

6. Click on **InDesign**.
7. From the menus at the top of the XDM executive screen, select **Profile → Options → XMake.**
8. Select **-V (Verbose message mode)** so XMake will keep you informed of its progress as it processes the design.
9. Select **Done → Done** to return to the XDM executive screen.
10. To save all of the changes that you just made to the XDM and XMake defaults, pull down the **Profile** menu and select **Save-profile → Yes**.

The Saveprofile command saves the recent changes to the xdm.pro file in your design directory. There are several PRO files associated with the Xilinx software. They are used to set default options. Xdm.pro sets options that are used by XMake, whether or not it is called from XDM, and by any other tools called from XDM. Other than XMake, if you run these commands directly from the DOS command line, bypassing XDM, they do not use these defaults; parameters must be typed at the command line or spelled out in a batch file that you create. These defaults are a good reason for using XDM to process your designs. To learn about running commands from the command line, look up each tool in the *XACT Reference Guide*.

Some default profiles are supplied with the Xilinx software, and some are created the first time that a tool is run. Default files are located in the \xact\data directory. If the appropriate PRO file exists in the design directory, it is used; if not, the software uses the one from the \xact\data directory. It is common practice to keep PRO files in the design directory to customize your environment. With this practice, you avoid having to change the program options each time the software is run. User PRO files should be saved to the design directory, not to the \xact\data directory. The files there should generally be retained as templates. They are overwritten each time that the software is updated or reinstalled.

Cleaning up the Design

From XDM, you can call an OrCAD utility, Cleanup, that checks your design for dangling nets, corrects any errors that it can fix, and reports any errors that it cannot correct.

1. Select **Translate**.

   The various translation programs supported by Xilinx appear on the Translate menu.

2. Select **CLEANUP**.

   XDM presents a list of all schematic files in your project directory.

3. Select **CALC.SCH** from the list.

   A list of all options to the Cleanup command is presented.

4. Select **Done**.

   The Cleanup command displays a list of warnings and errors.

   No warnings or errors should be displayed. If warnings or errors are reported, return to the SDT schematic editor and correct the problem. Rerun the Cleanup program to verify that the warnings or errors do not reoccur.

5. Press any key to return to XDM.

Additional Configuration (XC4000 Family Only)

XC4000 family users must make three more changes to the XDM configuration before processing the tutorial design. LOC attributes are used throughout the Calc schematics to specify pad locations that match the XC3000 sockets on the Xilinx demonstration boards. You could go back and edit the schematics to include the XC4000 pad locations. Instead, tell XMake to ignore schematic pad locations when running the Xilinx netlist checker, XNFPrep. Then supply the XC4000 pad locations using a constraints file.

If you are not doing an XC4000 family design, do not perform any of the commands in this section.

1. From the menus at the top of the XDM window, select **Profile → Options → XNFPREP.**

2. Select **ignore_xnf_locs=<type> → IO** to specify that pad constraints in the schematic be ignored.

   If this option is not selected, XNFPrep reports that the pad locations in the schematic are illegal for the selected part type, and XMake terminates.

3. Select **Done** to return to the Profile Options menu.
4. Select **PPR → -helpall**.
5. Select **cstfile=<file>**.

   A list of constraints files in the design directory appears.
6. Select **calc_4k.cst** to supply pad locations for the XC4000 sockets in the Xilinx demo boards.
7. Select **Done → Done** to return to the XDM executive menu.
8. Select **Profile → Saveprofile** to save the changes.

Using a constraints file, you can supply constraints information in a textual form rather than putting it on a schematic. Sometimes this method is more powerful than putting constraints on a schematic.Figure 11-26 shows the constraints file calc_4k.cst, that is supplied for this tutorial. Figure 11-26 shows a sample constraints file, not used here, which can help in routing some XC3000 family designs.

Specifying a constraints file is valid for any device being processed by the PPR program, including XC3000A, XC3000L, and all XC4000 family parts. All other devices are processed with the APR (Automatic Place and Route) tool. To specify that a constraints file be used with APR, make sure that the correct part is selected, then click on **Profile → Options → APR → -C <file>** and select the constraints file from the list presented.

Constraints file syntax is different for PPR than for APR. For additional information about constraints files, see the "PPR" and "APR" sections of the *XACT Reference Guide.*

```
#This is a schematic constraints file for use with the
# Calc Tutorial Design. It maps the I/O to the correct
# pins for 4K family parts on the Xilinx demo boards.
#When using this contraints file,you must tell XNFPrep
# to ignore pad locations on the schematic,as the
# locations on the schematic are for 3K family parts.
#Set the ignore_xnf_locs option for XNFPrep to a value
# of "IO."

place instance EXC_P:      P19;
place instance 7SEG/OFL_P: P41;
place instance 7SEG/A_P:   P49;
place instance 7SEG/B_P:   P48;
```

```
place instance 7SEG/C_P:   P47;
place instance 7SEG/D_P:   P46;
place instance 7SEG/E_P:   P45;
place instance 7SEG/F_P:   P50;
place instance 7SEG/G_P:   P51;
place instance SW7/SW6_P:  P20;
place instance SW7/SW5_P:  P23;
place instance SW7/SW4_P:  P24;
place instance SW7/SW3_P:  P25;
place instance SW7/SW2_P:  P26;
place instance SW7/SW1_P:  P27;
place instance SW7/SW0_P:  P28;
place instance LED/LED3_P: P57;
place instance LED/LED2_P: P58;
place instance LED/LED1_P: P59;
place instance LED/LED0_P: P60;
```

Figure 11-26 CALC_4K .CST File

```
;sample.cst, GOOD FOR 2K AND 3K ONLY, not needed for
; tutorial design
;Constraint file to prohibit columns A, D, and G. The
; technique of prohibiting every third or fourth
; column is very useful when routing designs that
; are less than two-thirds full, but are very routing-
; intensive.
;If prohibiting columns is necessary to ease routing
; congestion, always prohibit the leftmost column for
; best results. This eases congestion by the pads.
;
Prohibit Location AA AD AG ;
Prohibit Location BA BD BG ;
Prohibit Location CA CD CG ;
Prohibit Location DA DD DG ;
Prohibit Location EA ED EG ;
Prohibit Location FA FD FG ;
Prohibit Location GA GD GG ;
Prohibit Location HA HD HG ;
```

Figure 11-27 Sample .CST File, Not Used for This Design

Translating the Calc Design

Now that XDM is properly configured, you are ready to process the Calc design.

1. Select **Translate**.

   The various translation programs supported by Xilinx appear on the Translate menu.

2. Choose automatic translation by selecting **XMAKE**.

   A list of options is presented.

3. Since this design uses the default options preset in the xdm.pro file, simply select **Done**.

   A menu of design files in the current directory is displayed.

4. Identify the top-level drawing file by selecting **CALC.SCH**.

   The `Select target:` prompt appears. A menu offers alternatives for XMake. Each option goes further in the translation process than the one above it. You want to complete the entire translation process all the way to creating a bitstream.

5. Select **Make bitstream**.

   The XMake program processes all the necessary design files, displaying its progress on the screen. If the translation is successful, XMake issues this message:

   ```
   'calc.bit' has been made, check output in
   'calc.out'
   ```

6. Press any key to return to XDM.

Examining XMake Output Files

In addition to the routed LCA file and the bitstream BIT file, XMake generates three very useful output files.

- The OUT file, in this case calc.out, contains a copy of all text that is echoed to the screen. You should always review the OUT file after running XMake, *even if you did not see any warnings or error messages* while the design was being translated. If there are any messages reporting problems, you will save yourself a lot of time by catching the problem now, instead of later in the design process. The

OUT file shows every program run by XMake.

- The PRP file, in this case calc.prp, is the DRC report file generated by XNFPrep. If XNFPrep finds any errors or warnings, the OUT file directs you to examine this file. Besides explanation of errors and warnings, the PRP file contains a detailed list of all logic trimmed by XNFPrep and why it was unnecessary. This file can be a very useful debugging tool.
- The RPT file, in this case calc.rpt, is a formal report of the results of the placement and routing. It is generated by the place-and-route software, either PPR or APR, depending on the family. Check the RPT file to make sure there were no unrouted pins or nets.

Examine the OUT, PRP, and RPT files for the Calc design.

1. Pull down the **Utilities** menu and select **DOS**.

   This command exits XDM temporarily. You can also create a DOS shell by typing either **dos** ↵ or **sys** ↵ at the keyboard; watch for the letters to appear at the XDM command line at the bottom of the screen. From any of these DOS accesses, you can return to the suspended program by typing **exit** ↵.

2. Use any text editor or the DOS Type command to examine the calc.out file.
3. Use any text editor or the DOS Type command to examine the calc.prp file.
4. Use any text editor or the DOS Type command to examine the calc.rpt file.
5. When you have finished looking at the files, return to XDM by typing **exit** ↵ at the DOS prompt.

Checking for Warnings in the OUT and PRP Files

You should expect to see some warning messages in the calc.out and calc.prp files, but no errors. Errors are problems with the design that cause XMake to terminate. Warnings are simply notifications that there are unusual aspects to your design of which you should be aware. You may or may not choose to correct the reported situations.

The OUT file shows that XNFPrep issued one or two warnings, depending on the product family, and no errors. The OUT file directs you to the PRP file for explanations about the warnings.

If you examine the calc.prp file, you should see warnings similar to those shown in Figure 11-28. Which of these warnings appear depends on which product family you targeted. The PRP file includes an explanation of the warnings, none of which are a matter for concern for this design, since timing is not an issue.

```
---------------

XNFPREP: WARNING 4037:

These inverters could not be absorbed and each will be
implemented in a single function generator. This will
introduce additional delay and use resources inefficiently.
(Note that some of the symbols listed below may have been
reduced to inverters by earlier trimming.)

Inverter Name = DEBOUNCE/U64
Output Signal = DEBOUNCE/$U64_O

Inverter Name = CONTROL/U165/U1/U2/U1
Output Signal = CONTROL/U165/U1/MD

---------------

XNFPREP: WARNING 4082:

Double pullups were found on TBUF-driven longlines and/or
edge decoder longlines. Requiring two pullups would prevent
half-length longlines from being used, and design placement
and resource utilization would be adversely affected.

One pullup is being removed from each of these longlines. PPR
will activate both pullups if the signal is routed using a
complete longline.

---------------

XNFPREP: WARNING 4613:

Each of the following signals drives more than 20 inputs.

Signal Name
-----------
ADDR0
ALU3
ALU2
ALU1
ALU0

Although this is valid there is a possibility that the
routing delays of these signals will be unacceptable. If
these signals are critical to the timing of your design, you
may consider replicating the logic used to generate the
signals so that the loading (and hence the subsequent routing
delay) is reduced.
```

Figure 11-28 XNFPrep Warning Messages

The XMake OUT file also reports warnings and errors from programs other than XNFPrep. A partial listing of calc.out for the XC3020APC68 is shown in Figure 11-29.

For XC3000 family designs, PPR reports a warning that the Calc design contains a combinational loop. This loop is intentional: the OSC_3K schematic uses this logic loop, together with an external RC circuit, to generate the clock. Therefore you should ignore this warning if you compiled the Calc design to an XC3000 family part.

Any other errors or warnings besides those just discussed must be corrected before continuing. To find the source of unexpected errors, reload the design in SDT and compare the schematics to those in the figures in this chapter. After correcting the errors, save the changes, run the Cleanup program, and rerun XMake as previously described.

```
************************************************************
XMAKE: Execute command ‘annotate calc.sch ‘.
************************************************************

XMAKE: Execute command ‘inet calc.sch ‘.

************************************************************

XMAKE: Execute command ‘sdt2xnf calc.inf calc.xnf -D xnf ‘.

************************************************************

XMAKE: Execute command ‘xnfmerge -A -D xnf -D . -P 3020APC68-
7 xnf\calc.xnf calc.xff’.
Netlist written to file calc.xff

************************************************************

XMAKE: Execute command ‘xnfprep calc.xff calc.xtf
parttype=3020APC68-7’.

xnfprep: running design rule checker ...
xnfprep: checking XACT-Performance specifications ...
xnfprep: trimming unnecessary and redundant logic...
xnfprep: running design rule checker on trimmed design...
xnfprep: reverifying XACT-Performance specifications in
trimmed design ...

XNFPREP SUMMARY
---------------
0 Errors found
1 Warnings found
17 Unnecessary inverters and buffers removed
18 Unnecessary or disabled symbols removed
13 Sourceless or loadless signals removed
Refer to the calc.prp file for details.
```

```
************************************************************

XMAKE: Execute command 'xnfmap -P 3020APC68-7 calc.xtf
calc.map'.

DESIGN SUMMARY:
Part type=3020APC68-7
49 of 64 CLBs used
23 of 58 I/O pins used
0 of 6 internal IOBs used
0 of 16 internal three-state signals used (0 TBUFS used)
29 CLB flipflops used
Total number of WARNINGS generated during mapping = 0.
Total number of ERRORS generated during mapping = 0.

Writing design file calc.map...
Writing guide file calc.pgf...

************************************************************
XMAKE: Execute command 'ppr calc.map parttype=3020APC68-7'.

ppr: Routing signals...
ppr: Generating .LCA File...

ppr: Generating .BID Back Annotation File...
ppr: Making Report File...
Wrote report on the result without delay optimization to
calc.rpf.
ppr: Routing signals...

*** PPR: WARNING 6811:
This design has 1 purely combinational loop. Such loops
should be avoided. If at all possible, please modify the
design to eliminate all unclocked feedback paths.

A loop of 1 source-to-load connections:
FG4 FG_OSC_3K/Q (Net OSC_3K/Q) to first gate again.

No more unroutes
Therefore deleting result with 2 unroutes
Begin work on a 65.8ns path with 9 pins.
Design has 0 unroutes.

------------------------------------------------------------
Timing analysis summary
------------------------------------------------------------

Deadline Actual(*) label: [qualifier]
-------- --------- ------------------
<auto>    65.8ns   DEFAULT_FROM_FFS_TO_FFS=FROM:ffs:TO:ffs
<auto>    14.0ns   DEFAULT_FROM_PADS_TO_FFS=FROM:pads:TO:ffs
<auto>    52.8ns   DEFAULT_FROM_FFS_TO_PADS=FROM:ffs:TO:pads

(*) Note: please run xdelay to confirm the actual path delays
```

```
computed by PPR.

*** PPR: WARNING 7028:
The design uses an asynchronous input (RD/CLR and/or SD/PRE
pin) of one or more flip-flops. PPR can trace paths up to such
a pin, but it never considers paths into a flip-flop on such
a pin and on through the flip-flop output.

# of unrouted connections: 0.
ppr: Generating .LCA File...
ppr: Making Report File...

************************************************************
XMAKE: Execute command 'xdelay -D -W calc.lca'.
xdelay: writing calc.lca with delay information...

************************************************************

XMAKE: Execute command 'makebits -R2 -S0 -XB -YA calc.lca'.

XMAKE: 'calc.bit' has been made.
```

Figure 11-29 XMake Partial Output File, Calc.out

Checking the RPT File

The report file contains a great deal of detailed information about your routed design. The most useful information you will find in the RPT file is probably the pinout description, and detailed timing information in the case of PPR.

A portion of the report file for the XC3020APC68 Calc design is shown in Figure 11-30.

```
PPR RESULTS FOR DESIGN CALC

Design Statistics and Device Utilization

----------------------------------------
Design Summary from XNF File
Number of Signals: 133
Number of Pins: 497

Partitioned Design Utilization Using Part 3020APC68-7

                            No.Used Max Available % Used

--------------------------- ------- ----------- ------
Occupied CLBs                    50         64        78%
Packed CLBs                      42         64        65%
--------------------------- ------- ----------- ------
Bonded I/O Pins:                 22         64        35%
CLB Function Generators: (*)     70         128       54%
```

```
CLB Flip Flops:                 29          128       22%
IOB Input Flip Flops:           7           64        10%
IOB Output Flip Flops:          0           64        0%
3-State Buffers:                0           192       0%
3-State Longlines:              0           32        0%
--------------------------- ------- ----------- ------

(*) Each base F or FGM function counts as two

Routing Summary
Number of unrouted connections: 0

Chip Pinout Description
-----------------------

This chapter describes where your design's pins were placed in
terms of the package pins. This first list is sorted by
package pin location. The second list presents the same data
sorted by your design's pin names.

Package Pin Location Pin Name
-------------------- -----------------
P11 :                EXC_P
P12 :                OSC_3K/CQL
P13 :                SW7/SW6_P
P14 :                OSC_3K/CQ
.
.
P55 :                7SEG/G_P
P56 :                7SEG/D_P

This list describes where your design's pins are in terms of
the package pins; it is sorted by your design's pin name. The
list presented above has the same data sorted by package pin
location.

Package Pin Location Pin Name
-------------------- -----------------
P11 :                EXC_P
P29 :                LED/LED0_P
P30 :                LED/LED1_P
P31 :                LED/LED2_P
.
.
P55 :                7SEG/G_P
P53 :                7SEG/F_P
```

Figure 11-30 Partial Calc.rpt File

Examining Routed Designs with XDE

Note: The XACT Design Editor, XDE, is not available with the Base Development System. This section can be skipped without affecting the outcome of the tutorial; simply continue with the next section, "Verifying the Design Using a Demonstration Board."

The design process is now complete. Next, EditLCA, a subprogram of the XACT Design Editor (XDE), allows you to take a graphic look at the placed and routed design. XDE can be accessed from XDM or directly from DOS.

XDE and EditLCA provide several useful functions, such as the ability to hand-edit a routed design, insert probes while doing in-circuit verification, and the ability to do static timing analysis. For a much more complete tutorial on using XDE, see the "XDE Tutorial" chapter of the *XACT User Guide*.

Entering the Design Editor

1. To call up XDE from XDM, pull down the **PlaceRoute** menu and select **XDE**.

   An options menu appears. The default options are correct.

2. Select **Done**.

   The XDE Executive screen appears. If, instead, the message `Command 'xde' failed, rc=-1` appears, there is probably a semicolon at the end of your search path. Remove it and try again.

   At the bottom left corner of the screen, `Mode: Safe` is reported. This mode prevents you from making any changes to the LCA file that will change the functionality of the design. If you want to make changes to a design using EditLCA, you must change to "Expert" mode before loading the design.

3. Click on **Mode** with the left mouse button.

   A menu with a choice of Safe or Expert appears.

4. Click on **Cancel** to remain in Safe mode.

5. To load the design into the editor, select **Designs**.

   A pulldown menu appears.

6. To choose the input LCA file, select **Design**.

   A menu of available LCA files appears. In this case, there is only one LCA file in the directory, calc.lca, the routed file generated by XMake.

7. Select **CALC.LCA**.

   The name of the design file appears in the status area above the command line. When you select the design, the part type is automatically set to the part type in the selected LCA file.

8. To enter the Design Editor, pull down the **Programs** menu and select **EditLCA**.

   The design begins loading into the editor. Watch the status line above the command line to see what XDE is doing. The following messages appear.

   ```
   Loading die/package file...
   Loading design file...
   Building pip drawing information...
   Drawing screen...
   Timing nets...
   ```

   Now the editor appears, as shown in Figure 11-31. The editor is a graphic representation of the LCA file. Pan around the device by holding down the left mouse button and moving the mouse. A world view of the device appears in the lower right corner of the screen, and a red box shows the current location on the device.

There are two types of blocks shown by the editor: the IOBs appear around the periphery of the device, and the CLBs appear in the middle. Pan around the screen to see how the design was placed and routed in the device. Used blocks are highlighted, and the signal nets connecting them are shown as highlighted traces.

Looking at the routed design, observe how the global clock was laid out in the device and how the pin location constraints were carried through from the schematic level to the routed design.

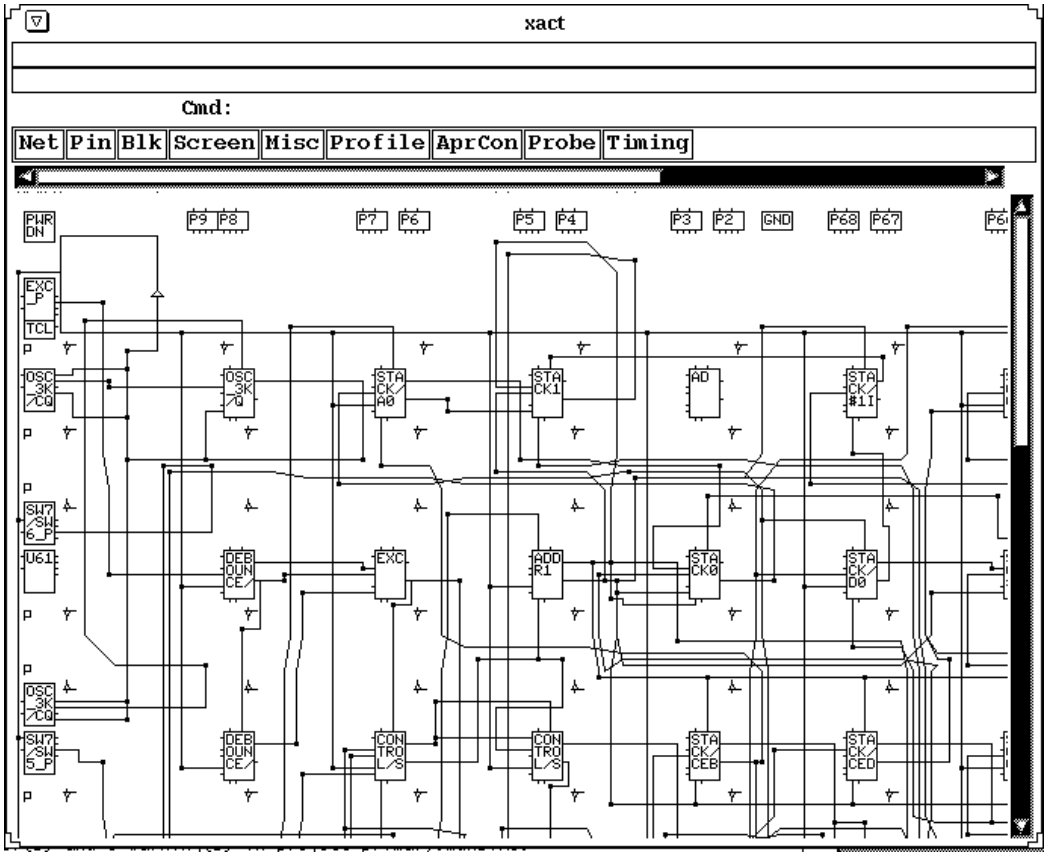


Figure 11-31 Portion of XDE EditLCA Screen, XC3020APC68

Finding a Block

You can locate the global clock buffers.

1. Pull down the **Screen** menu and select **Find**.
2. Now enter the name of the global clock buffer, **gclk** ↵ for an XC3000 family design or **bufgs_tl** ↵ for an XC4000 family design, to select the buffer in the top left corner.

   The cursor moves to the top left corner of the LCA editor. This is the location of the global clock buffer on XC3000 devices and one

of the global secondary clock buffers on XC4000 devices. This buffer may or may not be the one that PPR used to generate the clock signal in your XC4000 design.

3. Select **Done** to exit the Find command.

Highlighting a Net

The detailed operation of XDE is beyond the scope of this tutorial, but there are a couple of commands that are especially useful for examining this design. The Net Hilight command is used to highlight the path of a particular net, which makes that net easier to trace across the device.

1. Pull down the **Net** menu and select **Hilight**.

   A menu of colors appears.

2. Select a highlight color to use.

3. Type **clk** ↵.

   The clock net appears in the selected color.

4. Select **Done** from the top of the screen to finish the Net Hilight command.

Using Command Line Entry

XDE commands can also be entered directly from the keyboard without using the pulldown menus.

1. To remove the highlighting from the global clock net, type **unhilight clk** ↵ on the command line.

2. Leave EditLCA by typing **quit** ↵.

The XDE Executive menu returns.

Running the Design Rule Checker

XDE has a design rule check program, called DRC, which ensures that an LCA file is valid. If you make any edits in EditLCA, you should run the DRC program to make sure you have not introduced any invalid connections.

1. To run the DRC program, pull down the **Programs** menu and select **DRC**. A menu of options appears.
2. Since none of these options are necessary for this design, simply select **Done**.

   The screen switches into text mode, and messages describing what the DRC program is checking appear. No errors or warnings should occur.
3. Press any key to return to the XDE Executive Screen.
4. Leave XDE by typing **quit** ↵.
5. Press any key to return to XDM.

You can also run DRC from inside the EditLCA program.

The commands just given are only a few of the many useful commands available in the XACT Design Editor. For more details on these and all other XDE commands, consult the *XACT Reference Guide.*

Verifying the Design Using a Demonstration Board

Now that you have completed your design and run a design rule check, you are ready to download it to an FPGA. XMake has already generated a bitstream for the design.

There are three Xilinx demonstration boards in common use. Which board you have depends on what software you purchased and when you bought it. Your tutorial design should be targeted to a device on the board that you have available.

The three boards include one with both an XC3000 family socket and an XC4000 family socket, containing an XC3020APC68 and an XC4003APC84. This tutorial only uses one of the two parts; which part you use is up to you. This combination board is called the FPGA Demonstration Board. Another board, referred to as the XC3000 Demonstration Board, contains a single XC3020PC68, and the XC4000 Demonstration Board contains a single 4003PC84 or 4003APC84.

To load the configuration bitstream to the demonstration board, you need one of Xilinx' hardware cables. Xilinx makes two different hardware cables, the XChecker cable and the Download cable. Either cable works with any of the Xilinx demonstration boards.

Connecting the Cable for Download

Before initiating the physical downloading of the design into the FPGA on a Xilinx demonstration board, the board must be correctly hooked up to your PC.

There are several control and power pins that must be connected between the board and the cable. The bundles of leads supplied with the cables are labeled to make connecting the board to the cable a simple process. Additionally, a pair of power and ground pins must be connected to a regulated 5 volt power supply to supply power to the board and cable.

Connect one end of the cable to your demonstration board as described in Table 11-1 and Table 11-1. For the FPGA demonstration board, use the leftmost column of pins, which is missing the pin in the third position.

The other end of the cable must be plugged into the back of your machine. Attach the Download cable to a parallel port or the XChecker cable to a serial port.

The “XC3000 Design Demonstration Board”, “XC4000 Design Demonstration Board,” and “FPGA Demonstration Board” chapters of the *XACT Hardware and Peripherals Guide* discuss in detail the various demonstration boards and how to hook them up. Please refer to this manual for instructions, if necessary, then carefully verify the following.

- Verify that the hardware cable is correctly connected to both your system and the demonstration board. Connections from the cable to the demonstration boards are shown in Table 11-1 for XC3000 family devices and Table 11-1 for XC4000 family devices.
- Verify that the power supply is connected to the demonstration board *and is turned on.* The power connections for the demonstration boards are shown in Table 11-3.

Table 11-1 Demonstration Board Cable Connections For XC3000

XC3000 Board	Cable Label	FPGA Board	Cable Label
J1-1	V_{CC}	J1-1	V_{CC}
J1-2	Gnd	J1-3	Gnd
J1-3	No Connection	J1-5	No Connection
J1-4	CCLK	J1-7	CCLK
J1-5	D / P	J1-9	D / P
J1-6	DIN	J1-11	DIN

Table 11-2 Demonstration Board Cable Connections For XC4000

XC4000 Board	Cable Label	FPGA Board	Cable Label
J1-1	V_{CC}	J2-1	V_{CC}
J1-2	Gnd	J2-3	Gnd
J1-3	No Connection	J2-5	No Connection
J1-4	CCLK	J2-7	CCLK
J1-5	D / P	J2-9	D / P
J1-6	DIN	J2-11	DIN
XChecker Cable Only:		**XChecker Cable Only:**	
J1-7	PROG	J2-13	PROG
J1-8	INIT	J2-15	INIT
J1-9	RST	J2-17	RST

Table 11-3 Demonstration Board Power Connections

XC3000 Board		XC4000 Board		FPGA Board	
J3-1	+ 5 volts	J2-1	+5 volts	J9-1	+5 volts
J3-2	Gnd	J2-2	Gnd	J9-2	Gnd

FPGA (XC3000/XC4000) Demonstration Board

Make sure the FPGA demonstration board is set up for slave mode configuration. The configuration mode for the XC3000 family part is controlled by the SW1 bank of switches. The configuration mode for the XC4000 family part is controlled by the SW2 bank of switches. The switches should be set as shown in Table 11-4 and Table 11-5.

Table 11-4 FPGA Board Switch Positions for XC3000

Switch	Label	Setting
SW1-1 (left)	INP	Don't Care
SW1-2	MPE	Off
SW1-3	SPE	Off
SW1-4	M0	On
SW1-5	M1	On
SW1-6	M2	On
SW1-7	MCLK	Off
SW1-8 (right)	COUT	Off

Table 11-5 FPGA Board Switch Positions for XC4000

Switch	Label	Setting
SW2-1 (left)	PWR	Don't Care
SW2-2	MPE	Off
SW2-3	SPE	Off
SW2-4	M0	On
SW2-5	M1	On
SW2-6	M2	On
SW2-7	RST	Off
SW2-8 (right)	No Label	Off

XC4000 Demonstration Board

Make sure the XC4000 demonstration board is set up for slave mode configuration. The configuration mode is controlled by the SW4 bank of switches. The switches should be set as shown in Table 11-6.

Table 11-6 XC4000 Board Switch Positions

Switch	Label	Setting
SW4-7	PWR	Off (unless using battery)
SW4-6	MPE	Off
SW4-5	SPE	Off
SW4-4	M0	Off
SW4-3	M1	Off
SW4-2	M2	Off
SW4-1	RST	On
SW4-0	No Label	Don't Care

XC3000 Demonstration Board

If you have an XC3000 demonstration board that has been modified for use with a serial PROM, be sure it is not configured for use with an XC1736A or XC1765 Serial Configuration PROM. (If you have the demonstration board as shipped with Xilinx products, there is no serial PROM socket on the board.) Such a modified board contains a four-position DIP switch with a power switch and three switches controlling the programming mode. If this DIP switch is present, make sure that the switches are set for slave mode download. A serial PROM can be present on the board if this DIP switch is installed and set correctly. Information on modifying the demonstration board for use with a serial PROM is available in the "XC3000 Design Demonstration Board" chapter of the *XACT Hardware and Peripherals Guide*.

Downloading the Bitstream

Once the cable is connected to your PC, you are ready to download the bitstream.

1. Set all of the input switches High. This setting (SW3 switches High on the FPGA (XC3000/XC4000) board, SW5 switches High on the XC4000 boards, SW1 switches to "1" on the XC3000 board) selects the No-Op command.
2. In XDM, select the **Verify** menu.
3. Select **XCHECKER**.

   The XChecker software is used for either hardware cable.
4. Select **-port <name>** and the correct port.
5. Select **Done** and the input file name: **CALC.BIT**.

   Alternatively, you can run XChecker from the system prompt. Type:

   xchecker -port *portname* calc↵

   An example is the following:

   ```
   xchecker -port com2 calc↵
   ```

   Valid port names on a PC are COM1 or COM2.

   Once you have used XChecker and set the correct port, that information is saved in a file called xchecker.pro in your design directory, and you do not have to specify it each time.
6. If you are using the Download cable to program an XC4000 family part, press the PROG button. This step is not necessary if you are using the XChecker cable or using the Download cable to program an XC3000 family part.
7. Press the ↵ key.

   If the FPGA is successfully programmed, the following message appears:

   ```
   DONE signal went high.
   ```
8. Press any key to return to XDM.

   If the done signal does not go High, check the connections

between the cable and the demonstration board, power the board off and on, and try downloading again.

Note: The Download cable has limited functionality when used with XC4000 family parts and may report that Done went High even if you do not press the PROG button as in step 5, above. In this case the part is not reprogrammed. Download the bitstream again, this time pressing the PROG button prior to configuration. Cycling the power off and on before beginning the download has the same effect.

Testing the Design

As described in the "Design Description" section near the beginning of this tutorial, the Calc design is essentially a 4-bit processor with a stack, in other words, a calculator that uses reverse polish notation. There are three types of input that you must supply: an opcode, data, and an Execute command.

Each demonstration board has a row of eight rocker switches that provide input to the design (SW3 on the FPGA (XC3000/XC4000) board, SW5 on the XC4000 board, SW1 on the XC3000 board). The leftmost switch, labelled "1," is the Execute command,which is activated by toggling the switch twice. The next three switches (labelled 2-4) select the opcode. Opcode encoding is shown in Table 11-7. Use the rightmost four switches (labelled 5-8) to input data. When the extended instruction set is selected with opcode 111, these switches provide additional bits of opcode.

Note: The rocker switches on the XC3000 demonstration board are On when down, Off when up. Use the "0" and "1" labels on the board as your guide.

Table 11-7 Processor Operations

2	3	4	5	6	7	8	Operation
0	0	0	*DATA*				ADD between switches and register
0	0	1	*DATA*				AND between switches and register
0	1	0	*DATA*				OR between switches and register
0	1	1	*DATA*				XOR between switches and register
1	0	0	*DATA*				SUB switch value from register
1	0	1	X	X	X	X	CLEAR register
1	1	0	*DATA*				LOAD register
1	1	1	0	0	0	X	ADD between stack and register
1	1	1	0	0	1	X	AND between stack and register
1	1	1	0	1	0	X	OR between stack and register
1	1	1	0	1	1	X	XOR between stack and register
1	1	1	1	0	0	X	SUB stack value from register
1	1	1	1	0	1	X	PUSH register value to stack
1	1	1	1	1	0	X	POP stack value to register
1	1	1	1	1	1	X	NOP

To perform an operation, simply set the data on the rightmost “nibble.” “On” is a one; “Off” is a zero. Look up the correct opcode for the operation you want to perform and set the three opcode switches to the correct value. Then toggle the leftmost Execute switch twice. If the switch is already On, switch it Off, wait a moment, and then return it to the On position.

The contents of the ALU register are displayed in hexadecimal on the 7-segment display. The top value in the stack is displayed in binary on the bank of LEDs.

1. Verify that the inital contents of both ALU and stack are all zeros. The decimal display says “0,” and the LED bar is all Off.

   Now put a 1 on the data switches and load the switch value to the ALU register. The op code is 110.

2. Set the seven rightmost switches to 110-0001.

3. Toggle the leftmost switch to execute the command.

   The decimal display shows the contents of the ALU register, which is now "1." The stack is still empty.

   Add 13 to the ALU register. The opcode is 000.

4. Set the seven rightmost switches to 000-1101.

5. Toggle the leftmost switch twice to execute the command.

   The decimal display shows the contents of the ALU register, which is now "E." The stack is still empty.

   Push the register value onto the stack. The opcode is 111, which is the extended opcode. The data must be set to 101*x*, where the *x* is a don't-care.

6. Set the seven rightmost switches to 111-1011.

7. Toggle the leftmost switch twice to execute the command.

   The decimal display still shows "E." The stack value is also "E," so the LED bar shows 1110 in the right-hand nibble.

   Perform an XOR operation between the switch value and the register. The opcode is 011. Set the data to all ones.

8. Set the seven rightmost switches to 011-1111.

9. Toggle the leftmost switch twice to execute the command.

   The decimal display changes to "1." The stack value on the LED display is still "E," 1110.

   Pop the value from the stack. The opcode is 111, which is the extended opcode. The data must be set to 110*x*, where the *x* is a don't-care.

10. Set the seven rightmost switches to 111-1101.

11. Toggle the leftmost switch twice to execute the command.

    The decimal display changes to "E." The stack value returns to "0".

    Clear the ALU register. The opcode is 101. The data is ignored.

12. Set the seven rightmost switches to 101-1101.

13. Toggle the leftmost switch twice to execute the command.

The decimal display changes to “0.” The stack value remains at “0”.

14. Try any other commands that you wish.

Making Incremental Design Changes

If you place and route your design, then decide to make a small design change, you do not have to start the place-and-route process over again from the beginning. You can use the existing design as a guide file, and the logic and routing that has not changed is retained from the previous version.

This is an advantage when you have a particular implementation with timing that meets your needs, or when time is a factor, since both APR and PPR place-and-route tools run significantly faster when most of the work is already done.

In this section of the tutorial, you make a small change to the schematic and reprocess the design using the Guide option.

Creating the Guide LCA File

The first step is to save the routed LCA file to another name. This file is used as the guide file.

1. Type **`sys`** ↵ to open a DOS shell.
2. From the DOS prompt type **`copy calc.lca gcalc.lca`** ↵.
3. Type **`exit`** ↵ to return to XDM.

Note: Alternatively, to run a DOS command from within XDM you can preface the command with the word “dos”. In this case you could type **`dos copy calc.lca gcalc.lca`** ↵.

The LCA file is not necessarily the only file that is used as input when using the Guide option. Incremental design with any member of the XC2000 or XC3000 families also requires a file called *filename*.pgf, as described in the “Configuring XMake for Incremental Design” section, following. If you complete an XC2000 or XC3000 family design and want to be able to use it as a guide file at a later time, it is best to save the PGF file, as well as the LCA file, although the PGF file can be recreated using the LCA2XNF program if necessary. There is no PGF file for XC4000 family designs.

Making an Incremental Schematic Change

Make a simple change to the Calc schematic that will be visible immediately on the demonstration board. Disconnect the overflow net and tie the signal high. When this net, labeled OFL, is high, the decimal point in the 7-segment display lights up to signal an overflow condition in the ALU. With the net tied high, the decimal point is on at all times.

Open OrCAD SDT and load the CALC schematic.

1. To start the OrCAD design environment from the XDM menu, click the left mouse button once on **DesignEntry**.
2. Select **ORCAD.**

   The ESP executive screen appears. Make sure that the words `CALC Design` appear at the top of the ESP screen. If they do not, you need to change the default project directory as described in the "Changing the Default Design" section near the beginning of the tutorial.
3. To enter SDT, select **Schematic Design Tools → Execute**.
4. Click on **DRAFT → Execute**.

   The CALC schematic appears on the screen.
5. Zoom in on the upper right-hand corner of the schematic.
6. Place the cursor over the vertical segment of the net connecting OFL to the ALU.
7. Select **Delete → Object → Delete**.

   The wire under the cursor disappears.
8. Click the right mouse button to exit the Delete command.

   The next step is to tie the OFL net High using the VCC component.

Note: For Xilinx designs, use the VCC and GND components from the Xilinx libraries to make connections to power and ground. These symbols are placed using the Get command just as with any other library symbols and must be connected with a wire. The wire *must* be used, or the symbol is not considered connected. Do not use the Place Power command in OrCAD.

9. Select **Get** and type **vcc** ↵.
10. Move the cursor so that the bottom of the VCC symbol makes a conection to the net that was formerly OFL, as shown in Figure 11-32.
11. Select **Place** and click the right mouse button to exit the Get command.

    The net is now tied high.

Note: It is important that any net connected to power or ground not be labeled with another name, or you will get errors during netlist translation. If you must put a label on a power or ground net, place a BUF symbol between the VCC or GND symbol and the net with the label.

12. Select **Quit → Update File → Abandon Edits** to save the changes and return to the ESP screen.
13. Select **To Main → Execute → Exit ESP → Execute**.
14. Type any key to return to XDM.

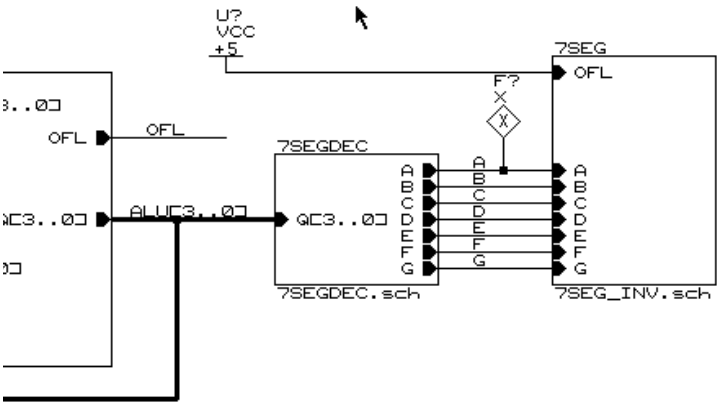


Figure 11-32 Adding a VCC Component

Configuring XMake for Incremental Design

Xilinx has two separate programs that perform placement and routing. PPR works for XC3000A, XC3000L, and all XC4000 family designs. APR works for XC3000, XC3100, and all XC2000 family designs.

If your tutorial design is targeted for a 3020APC68, 4003APC84, or 4003PC84 device, only commands related to PPR appear in the XDM menus. If it is targeted for a 3020PC84 device, only APR commands appear.

Read through the sequence of commands in this section and perform all that apply to your design.

PPR (XC3000A, XC3000L, XC4000 Family)

Tell PPR to use the guide file.

1. Select **`Profile → Options`**.

A list of programs appears on the screen.

2. Select **`PPR → -helpall`**.
3. Select **`guide=<file>`**.

   A list of LCA files in the design directory appears.

4. Select **`GCALC.LCA`**.
5. Select **`Done`** to return to the list of configurable programs.

APR (XC3000, XC3100, XC2000 Family)

Tell APR to use the guide file.

1. Select **`Profile → Options`**.

A list of programs appears on the screen.

2. Select **`APR → -G <file>`**.

   A list of LCA files in the design directory appears.

3. Select **`GCALC.LCA`**.
4. Select **`Done`** to return to the list of configurable programs.

XNFMap (XC3000 Family, XC2000 Family)

For all XC3000 and XC2000 family designs, an additional step is necessary. Logic mapping for these families is performed by a program called XNFMap. If the logic is not mapped into CLBs and IOBs in exactly the same way as in the guide design, PPR or APR cannot match up logic and signal names with the guide design. Therefore, you must select an option that tells XNFMap to use the existing mapping guide.

1. Select **XNFMAP → -K <file>.**

   XNFMap uses the file calc.pgf to guide mapping of the logic. The PGF file name is assumed to be the same as the input file name.

2. Select **Done** to return to the list of configurable programs.

This step is not necessary for XC4000 family designs, because for these parts PPR performs the mapping function as well as the placement and routing steps.

Return to XDM (All Families)

Select **Done** to return to XDM.

Translating the Incremental Design

Translate the guided Calc design in exactly the same way as you did the first time.

1. Select **Translate**.

   The various translation programs supported by Xilinx appear on the Translate menu.

2. Choose automatic translation by selecting **XMAKE**.

   A list of options is presented.

3. Since you already set all desired options in the xdm.pro file, simply select **Done**.

   A menu of design files in the current directory is displayed.

4. Identify the top-level drawing file by selecting **CALC.SCH**.

   The `Select target:` prompt appears.

5. Select **Make bitstream**.

   The XMake program processes all of the necessary design files, displaying its progress on the screen. If the translation is successful, XMake issues the following message:

   ```
   'calc.bit' has been made, check output in
   'calc.out'
   ```

6. Press any key to return to XDM.

Checking for Errors in the Calc.out File

The next step is to look at the OUT file created by XMake and check for errors. There should not be any errors or warnings that were not present in the first OUT file.

As always when checking the OUT file, the first priority is to verify that the placement and routing completed successfully and there were no unrouted nets.

XMake did not perform all of the commands this run, because most of the schematics did not change from last time. Also, XMake used different options when calling the various Xilinx programs to process the design.

For a description of the command flow followed by XMake for original and incremental translation, see the "Command Summaries" section at the end of this chapter. The command summaries are simplified versions of the program flow used by XMake but are equally valid.

Verifying the Change in the Demonstration Board

Verify that the change was performed by downloading the new bitstream to the demonstration board, as you did previously.

1. Set all of the input switches High. This setting (SW5 switches High on the FPGA (XC3000/XC4000) and XC4000 boards, SW1 switches to "1" on the XC3000 board) selects the No-Op command.
2. In XDM, select the **Verify** menu.
3. Select **XCHECKER**.

   The XChecker software is used for either hardware cable.

The port name is already set in the xchecker.pro file saved during the first download.

4. Select **Done** and the input file name: **CALC.BIT**.
5. If you are using the Download cable to program an XC4000 family part, press the PROG button. This step is not necessary if you are using the XChecker cable, or using the Download cable to program an XC3000 family part.
6. Press the ↵ key.

   If the FPGA is successfully programmed, the following message appears:

   ```
   DONE signal went high.
   ```

7. Press any key to return to XDM.

   If the Done signal does not go High, check the connections between the cable and the demonstration board, power the board off and on, and try downloading again.

The decimal point in the 7-segment display turns on, as directed by the incremental change you made to the schematic.

Leaving XDM

1. To leave XDM, select **quit** or type **quit** ↵. The following message appears:

   ```
   Current profile option changes will be lost.
   Do you really want to quit (Y/N)?
   ```

   You do not want to save the profile changes that set up XMake for incremental design.

2. Click on **Yes** in the menu or type **y** ↵ to quit without saving your profile changes.

You have completed the Xilinx SDT tutorial. At this point, you can, if you wish, continue on with the VST tutorial or skip to the special-topic tutorials also included in this manual.

Command Summaries

Although this tutorial uses XMake to process the Calc design, you do not have to use XMake if you do not wish to do so. You can also bypass XDM and run either XMake or each individual program from the system prompt.

This section details command sequences that you can use to perform all of the translations using XMake in this tutorial. The commands are written as you would type them at the system prompt or in a batch file. XMake executes some of these commands individually on each file in the design hierarchy, so that it does not rerun any program unnecessarily, but it is not necessary for you to do the same.

The command summaries assume that you have specified the part type in the schematic. If you prefer to specify the part type on the command line, call SDT2XNF with the -P option.

The Ignore_xnf_locs and Cstfile options are necessary for overwriting pad location constraints when compiling the Calc design for XC4000 family devices. You may or may not need to use them for your own designs.

Basic Translation for XC3000A and XC3000L Designs

cleanup calc.sch	(Not run by XMake)
annotate calc.sch	Update reference designators
inet calc.sch	Generate INF (OrCAD netlist) files
sdt2xnf calc	Translate INF files to XNF netlists
xnfmerge calc	Combine into one XNF file
xnfprep calc	Trim logic and check for errors
xnfmap calc	Map to CLBs and IOBs
ppr calc	Place and route
xdelay -d -w calc	Add delays to LCA file
makebits calc	Create bitstream

Basic Translation for XC4000 Family Designs

cleanup calc.sch	(Not run by XMake)
annotate calc.sch	Update reference designators
inet calc.sch	Generate INF (OrCAD netlist) files
sdt2xnf calc	Translate INF files to XNF netlists

xnfmerge calc	Combine into one XNF file
xnfprep calc ignore_xnf_locs=IO	Trim logic and check for errors
ppr calc cstfile=calc_4k	Map, place and route
xdelay -d -w calc	Add delays to LCA file
makebits calc	Create bitstream

Basic Translation for XC3000, XC3100, and XC2000 Family Designs

cleanup calc.sch	(Not run by XMake)
annotate calc.sch	Update reference designators
inet calc.sch	Generate INF (OrCAD netlist) files
sdt2xnf calc	Translate INF files to XNF netlists
xnfmerge calc	Combine into one XNF file
xnfprep calc	Check for errors
xnfmap calc	Map to CLBs and IOBs
map2lca calc	Convert to LCA file
apr -w calc calc	Place and route, add delays
makebits calc	Create bitstream

Incremental Translation for XC3000A and XC3000L Designs

Make schematic changes.

copy calc.lca gcalc.lca	(Not run by XMake)
cleanup calc.sch	(Not run by XMake)
annotate calc.sch	Update reference designators
inet calc.sch	Generate INF (OrCAD netlist) files
sdt2xnf calc	Translate INF files to XNF netlists
xnfmerge calc	Combine into one XNF file
xnfprep calc	Trim logic and check for errors
xnfmap -k calc	Map to CLBs and IOBs
ppr calc guide=gcalc	Place and route
xdelay -d -w calc	Add delays to LCA file
makebits calc	Create bitstream

Incremental Translation for XC4000 Family Designs

Make schematic changes.

copy calc.lca gcalc.lca	(Not run by XMake)
cleanup calc.sch	(Not run by XMake)
annotate calc.sch	Update reference designators
inet calc.sch	Generate INF (OrCAD netlist) files
sdt2xnf calc	Translate INF files to XNF netlists
xnfmerge calc	Combine into one XNF file
xnfprep calc ignore_xnf_locs=IO	Trim logic and check for errors
ppr calc guide=gcalc cstfile=calc_4k	Map, place and route
xdelay -d -w calc	Add delays to LCA file
makebits calc	Create bitstream

Incremental Translation for XC3000, XC3100, and XC2000 Family Designs

Make schematic changes.

copy calc.lca gcalc.lca	(Not run by XMake)
cleanup calc.sch	(Not run by XMake)
annotate calc.sch	Update reference designators
inet calc.sch	Generate INF (OrCAD netlist) files
sdt2xnf calc	Translate INF files to XNF netlists
xnfmerge calc	Combine into one XNF file
xnfprep calc	Check for errors
xnfmap -k calc	Map to CLBs and IOBs
map2lca calc	Convert to LCA file
apr -w -g gcalc calc calc	Place and route, add delays
makebits calc	Create bitstream

Annotate and Inet require that the SCH extensions be specified. For any design processed with PPR, you must run XDelay with the -w option if you want timing information back-annotated to the LCA file. You do not need to include this step for designs processed with APR, as APR places the delay information in the LCA file.

OrCAD Interface/ Tutorial Guide

VST Tutorial

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Chapter 12

VST Tutorial

This chapter steps through both a functional simulation and a timing simulation using OrCAD's VST simulator. The tutorial flow uses the Calc design created in the "SDT Tutorial" chapter. It also demonstrates how to use VST's Stimulus, Trace, and Breakpoint Editors and how to capture simulation output for later use as test vector input.

Required Software

This tutorial assumes that you are using the following versions of the development software:

- OrCAD / SDT — SDT 386+
- OrCAD / VST — VST 386+
- OrCAD / Xilinx Interface — Version 5.00 or later
- XACT Design Manager (XDM) — Version 5.00 or later

It is strongly recommended that you work through the SDT tutorial in the "SDT Tutorial" chapter before beginning this VST tutorial, even if you are familiar with OrCAD SDT. Information specific to using Xilinx parts is included in the documentation. In addition to schematic entry, the SDT tutorial steps through an entire place-and-route sequence, downloads the design to a demonstration board, and allows you to test the Calc processor in the board.

Directions given in this tutorial generally skip basic information already provided in the SDT tutorial: it is assumed that you have already read or performed that tutorial.

Before Beginning the Tutorial

Before starting the VST tutorial, perform the following steps.

Skipping the SDT Tutorial

If you chose to read through the SDT tutorial rather than actually perform the steps involved, you must verify that your PC is set up correctly to use OrCAD 386+ and the XACT Development System software. Then create and configure the design directory, and copy a completed set of schematics from any of the solutions directories supplied.

If you have already performed the SDT tutorial on your PC, skip to the next section, "XDraft and the Vst.cfg File."

1. Follow the instructions given in the "Before Beginning the Tutorial" section of the "SDT Tutorial" chapter for setting up your design environment.
2. The tutorial files are optionally installed when you install the Xilinx/OrCAD interface software. If you have already installed the software but are not sure whether you specified tutorial installation, check for the c:\xact\tutorial\orcad\calc directory. This directory contains the tutorial files.

Note: This tutorial assumes that your Xilinx and OrCAD software are loaded on the c: drive, but you can substitute any other drive. If your PC is configured with only one drive, it is not necessary to specify a drive. You can load the Xilinx software on one drive and the OrCAD software on another drive, but it is recommended that you load the OrCAD/Xilinx Interface on the drive with your Development System software rather than on the drive with the OrCAD software. If you load the interface to a drive other than the one containing the Development System, you create a second XACT directory containing your Xilinx libraries. If you must create a second XACT directory because of drive size limitations, see the "Partitioning Software Between Two Different Disks" section in the "Getting Started" chapter.

3. Create a new project called "Calc," as described in the "Creating the Project Directory" section of the "SDT Tutorial" chapter.
4. Copy a completed set of schematics from any one of the four

solutions directories provided. The four solutions directories are described in the “Solutions Directories” section of the “SDT Tutorial” chapter. From the DOS prompt, type:

cd \orcad\calc ↵
copy \xact\tutorial\orcad\calc*solution.*** ↵

where *solution* is either soln_3ka, soln_3k, soln_4ka, or soln_4k. The solutions schematics are targeted towards the 3020APC68-7, 3020PC68-50, 4003APC84-6, and 4003PC84-6 devices, respectively.

This procedure gives you a full set of completed schematics for the Calc design, with all supporting files.

Note: This tutorial assumes that your ORCADPROJ variable is set to c:\orcad\. You need not follow this convention.

5. Configure the design directory by typing **xdraft 3** ↵ or **xdraft 4** ↵, depending on whether you are targeting an XC3000 or an XC4000 device.

XDraft and the Vst.cfg File

Two important files in the design directory are sdt.cfg, the SDT configuration file, and vst.cfg, the VST configuration file. XDraft automatically edits these files to work with Xilinx designs.

XDraft makes the following changes to the vst.cfg file. The PLIB line is modified to point to the directory containing the Xilinx simulation model files. The name of this directory varies according to which family you specify when calling XDraft. The Prefix G is used for symbol instance names. A set of simulation macro files is provided using the MAC entry, and IMAC sets the initial macro.

A sample sdt.cfg file for the XC3000 family follows.

```
PDRV   = 'C:\ORCADESP\DRV\'
DD     = 'VGA640.DRV'
PLIB   = 'c:\xact\xc3000\'
PREFIX = 'G'
MAC    = 'c:\xact\VSTMAC.MAC'
IMAC   = 'F6'
```

See Table 12-1 for a listing of the macros defined in the vstmac.mac file. Several of these macros are referenced as “Keyboard Shortcuts” in the course of the tutorial.

Table 12-1 OrCAD/VST Key Macros

Function Key	Function
F1	Run Simulation → 100 (10.0 ns)
F2	Run Simulation → 1000 (100.0 ns)
F3	Run Simulation → 10000 (1000.0 ns)
F4	Verify → Enable Hold, Pulsewidth, and Setup Violation Checks → Escape
F5	Initialize → Yes → Set → Spool to Disk → Yes → Yes
F6	Zoom → Initial Display → 0
F7	Trace → Change View → 1 (0.1 ns / grid)
F8	Trace → Change View → 10 (1 ns / grid)
F9	Trace → Change View → 1000 (100ns / grid)

Completing VST Configuration

Although XDraft sets up the vst.cfg file for you, there is one more change that you must make manually using the OrCAD configuration procedure. You must change the simulation configuration so that the input file is CALC.VST rather than CALC.INF.

You must always simulate Xilinx designs from a VST file, never from an INF file. Xilinx does not provide INF models, because their functionality cannot correctly model the Xilinx libraries.

Enter the OrCAD environment and make sure you are editing files for the correct design directory.

1. From any directory, type **`orcad`** ↵. The OrCAD ESP design environment appears on the screen.
2. Make sure that the words `CALC Design` appear at the top of the screen. If you did not perform the SDT tutorial, the default design is probably set to TEMPLATE, instead of CALC. In that case, change the design to CALC as described in the “Selecting Calc as the Active Design” or “Changing the Default Design” sections of the “SDT Tutorial” chapter.

3. Select **Digital Simulation Tools → Execute** to enter the OrCAD simulation environment.
4. Select **Simulate → Local Configuration → Configure SIMULATE**.
5. Under File Options, click on the box next to Connectivity Database, and change the file name, such as CALC.INF, to **CALC.VST**↵.
6. Make sure that the Stimulus File is set to CALC.STM and the Trace File is set to CALC.TRC.
7. Click on **OK** to save the changes.

   This step writes the information to the vst.bcf file, the binary version of the vst.cfg file. If you edit the ASCII file vst.cfg using a text editor, a new vst.bcf is automatically generated the next time that you enter the simulator, and this change is lost. In general, it is better to configure the simulator using the OrCAD interface.
8. Exit from OrCAD by clicking twice on **To Main**, then twice on **Exit ESP**. ESP asks you whether you want to save the changes you made to the local configuration. Click on **Yes**.

Performing a Functional Simulation

There are six basic steps to the functional simulation of a Xilinx design in OrCAD. They are:

1. Place stimulus and trace data on the schematic to be simulated.
2. Create a functional simulation netlist with XSimMake, which performs the following steps:
   - Generates a Xilinx XNF netlist file for the design
   - Converts the XNF file to an OrCAD VST simulation file
   - Generates OrCAD stimulus (AST) and trace (ATR) files
3. Convert stimulus and trace files to binary format (STM and TRC).
4. Configure OrCAD VST for the particular design.
5. Add additional stimulus data using OrCAD's Stimulus Editor.
6. Perform the functional simulation.

You perform these steps on the Calc design in this tutorial.

Placing Stimulus and Trace Data on the Schematic

The first step in functionally simulating a design is to place the stimulus and trace data on the schematic to be simulated.

In this section, you place stimulus and trace data on the top-level CALC schematic. Additional stimulus data is already included in the SW7 schematic. You can place stimulus and trace data at any level of hierarchy.

Note: A complete schematic for CALC, with stimulus and trace data, is supplied in each solutions directory under the name calcsim.sch. However, it is recommended that you enter the stimulus and trace data using the procedure outlined in this section, unless you are already familiar with this process.

1. Start XDM from the DOS prompt. Type **XDM** ↵.
2. Start the OrCAD Schematic Editor by selecting **DesignEntry → ORCAD**.
3. To enter the Schematic Editor, select **Schematic Design Tools → Execute → Draft → Execute**.

   The top-level Calc schematic appears on the screen.

   There are two types of information to be added to the schematic. The first is stimulus information, which defines values for input nets, including clocks. The second is trace information, which tells the simulator which signals to monitor throughout the simulation.

   First, add the stimulus for the input clock. The oscillator output must be driven directly by the simulator, because there is no simulation model for the oscillator.

4. Select **Place → Stimulus**.

   The `Stimulus?` prompt appears at the top of the screen. If a menu appears instead, it means that you typed **S** to select Stimulus. Since the Sheet command occurs first on the menu, typing **S** is interpreted as Sheet. You must either type **I** or use the mouse to select **Stimulus** from the menu.

5. Type:

   `0:0 500:T 1000:G:500`↵

Note: Time units in VST are measured in tenths of nanoseconds. In other words, specifying a time interval of 500 means 50 nanoseconds.

6. Position the cursor somewhere on the net labeled CLK. Select **`Place`**.

   A small waveform marker (purple on color screens) appears above the net, as shown in Figure 12-1.

Note: Stimulus and trace data must always be placed with the cursor pointing directly at the net. The marker sits directly above the net to which the data is supplied. Additionally, make sure the point where you place the data is not on top of a symbol pin; the data marker must be at least one grid point from the end of the net segment, or it is ignored.

7. Press the **Escape** key.

   The stimulus just added defines the CLK signal to be low at time zero, toggle at time 500 (50.0 ns), then toggle again every 50.0 ns (according to the syntax, at time 1000, Goto 500). Thus, CLK is defined as a clock signal with a 50% duty cycle and a 100-ns period. Alternatively, you could obtain the same result by defining the stimulus as:

   `0:0 500:1 1000:G:0` or
   `0:0 500:T 1000:G:0`

8. Add the stimulus for EXC_P using the same technique as for CLK. Set the stimulus to:

   `0:1 1000:0 3000:1 6000:0 8000:1`

   Be careful to place the stimulus marker well away from the symbol pins at either end of the segment, as shown in Figure 12-1.

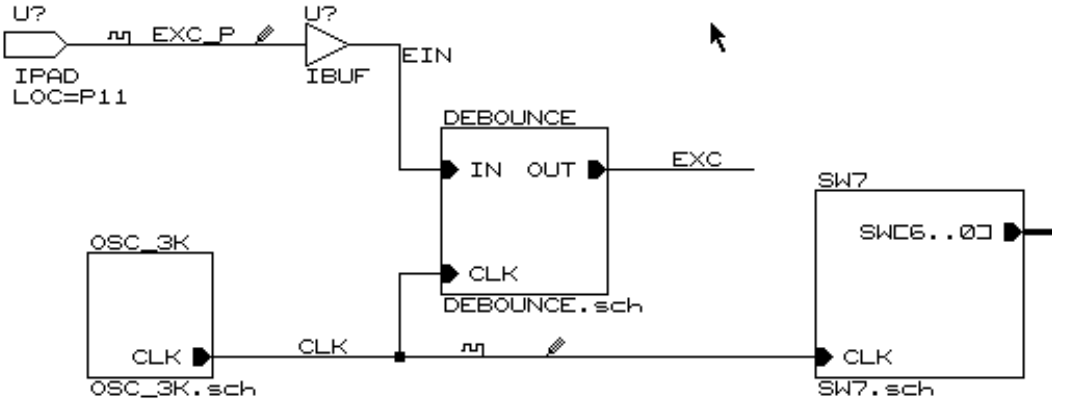


Figure 12-1 Stimulus and Trace Data for CLK and EXC_P

9. Add the trace data for the CLK node. Select **Place → Trace Name**.

   To select Trace Name you must type R or use the mouse; typing T selects the Text command.

   The `Trace Name?` prompt appears at the top of the screen.

10. Type **CLK**↵.

11. Position the cursor on the CLK net near the stimulus data. Select **Place**.

    The `Trace Name?` prompt reappears.

12. Type **EXC_P**↵.

13. Position the cursor on the EXC_P net as shown in Figure 12-1. Select **Place**.

14. Similarly, add trace data to the nets labeled WE, ALU0, ALU1, ALU2, ALU3, STACK0, STACK1, STACK2, STACK3.

    When you enter a trace name that ends with a number, the next trace name defaults to the same name plus one. To use the default

name, simply press ↵. If you do not want that name, backspace over the default name and enter the new name, or press the Escape key.

Note: You may not place trace or stimulus data on buses, only on wires.

15. Press the **Escape** key to exit the Place Trace command.

    Trace names should generally be the same as net names on the schematic to avoid confusion, but they do not have to be. They are the names that the simulator uses to display signal values. Trace names must never contain spaces.

    The complete CALC schematic, with stimulus and trace data, is shown in Figure 12-2.

16. Save the changes to the CALC schematic by selecting **`Quit` → `Update File` → `Abandon Edits`**.

17. Exit from OrCAD by clicking twice on **`To Main`** and twice on **`Exit ESP`**.

18. Press any key to return to the XDM environment.

It is not necessary to enter all stimulus and trace information from the schematic. In fact, you could enter all of this data from the simulator, but sometimes it is more convenient to place this type of information on the schematic.

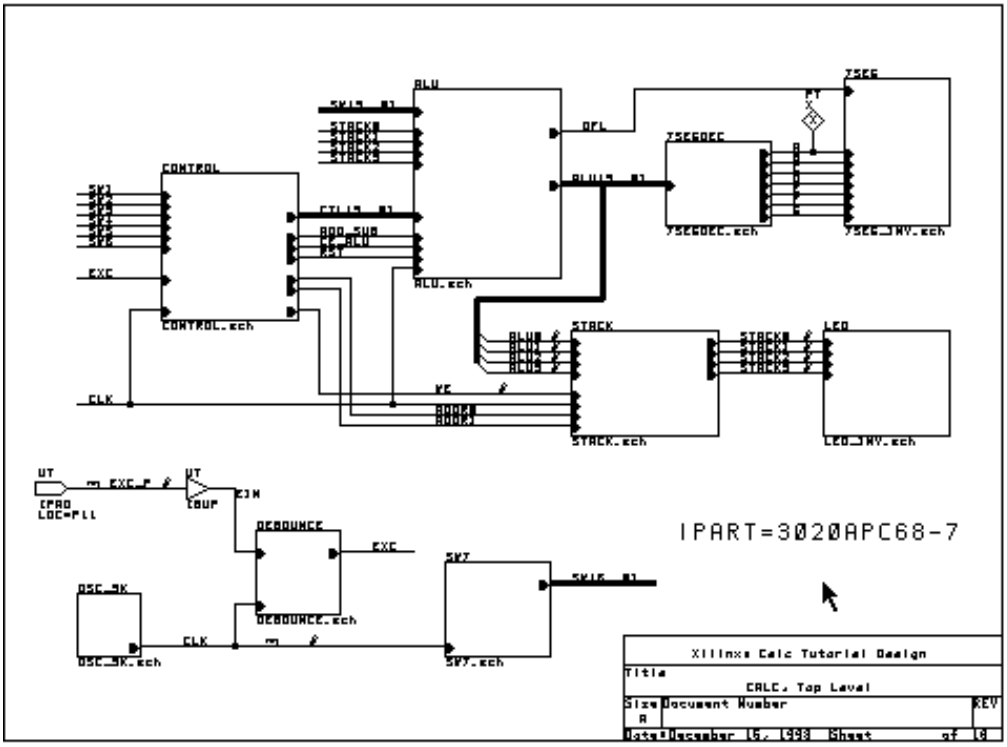


Figure 12-2 CALC Schematic with Stimulus and Trace Data

Creating a Functional Simulation Netlist with XSimMake

When completely placing and routing the Calc design in the SDT tutorial, you used the XMake program. To functionally simulate, it is not necessary to route the design unless CLB or IOB primitives are used; you need only translate it to XNF (Xilinx Netlist Format) and back again to incorporate the Xilinx simulation models. Therefore, instead of using XMake, you run a program called XSimMake.

The flow using XSimMake as described in this section works for simulating all designs created using either the Xilinx Unified

Libraries or the older v4.1 libraries, with or without X-BLOX, and including designs where some symbols do not have schematics but simply reference XNF netlist files by name. This flow specifically includes logic blocks created using Xilinx ABEL and MemGen. For more information on using these programs in Xilinx designs, see the "Xilinx ABEL Tutorial " chapter and the "Merging Design Files from Other Sources" section of the "FPGA Design Issues" chapter of this manual.

The commands in this section show you how to run XSimMake on your design. To learn more about the command sequence used by XSimMake, see the "Functional Command Summary" section of this chapter, and the other chapters of this manual.

Creating a Functional Simulation Netlist

Use XSimMake to create a functional netlist from XDM.

1. Select the **Verify** menu and click on **XSIMMAKE**.
2. Select **-F <flow name>**. If it is already highlighted, click the left mouse button twice to deselect it and select it again.

   A menu of supported command flows appears.
3. Click on **Orcad_Fpga_Func** to select the functional simulation flow.
4. Select **Done**.

   A list of all schematic sheets in the design directory appears.
5. Select **CALC.SCH** as the input schematic file.

   XSimMake routes text output to the screen, so that you can follow the conversion process and watch for error messages.
6. Press any key to return to XDM.

Examining the XSimMake Output File

XSimMake generates a very useful output file called xsimmake.out and places it in the project directory.

The xsimmake.out file contains a copy of all text that is echoed to the screen. You should always review this file after running XSimMake, *even if you did not see any warnings or error messages* while the design

was being translated. The xsimmake.out file shows every program run by XSimMake.

Examine the xsimmake.out file for functional simulation of the Calc design.

1. Pull down the **Utilities** menu and select **DOS**.
2. Use any text editor or the DOS Type command to examine the xsimmake.out file.

   Only one warning should be generated. XFind is a program run by XSimMake to determine what special blocks are used in the design, such as X-BLOX or Xilinx ABEL. The types of special blocks found determine the flow used by XSimMake to process the design. In this case, since there are no special blocks found, XFind reports a Warning 2: No special blocks found in calc.xff.
3. When you have finished looking at the file, return to XDM by typing **exit** ↵ at the DOS prompt.

A partial xsimmake.out file for functional translation of the Calc design is shown in Figure 12-3. The command sequence used by XSimMake varies greatly depending on what device is used, whether or not X-BLOX modules are included, and whether all blocks have schematics under them, or some reference XNF netlists as with designs containing Xilinx ABEL. XSimMake also calls the translation programs INF2XNF and XNF2INF directly rather than through the functionally equivalent programs SDT2XNF and XNF2VST. The sequence shown in Figure 12-3 is for the Calc design as it now stands, targeted for an XC3020APC68.

Note: XSimMake flows vary depending on the design. The flow used by XSimMake for your design may be slightly different from the flow shown in this tutorial.

```
XSIMMAKE COMMAND : creating directory savexnf
XSIMMAKE COMMAND : creating directory otherxnf
------------------------------------------------------------
XSIMMAKE COMMAND : annotate calc.sch
"OrCAD/SDT 386+ v1.10 H 17-SEP-93"
------------------------------------------------------------
XSIMMAKE COMMAND : inet calc.sch /t
"OrCAD/SDT 386+ v1.10 H 17-SEP-93"
------------------------------------------------------------
XSIMMAKE COMMAND : inf2xnf calc d= otherxnf
Executing 'inf2xnf calc d=otherxnf logfile=sdt2xnf.log' .....
------------------------------------------------------------
XSIMMAKE COMMAND : xnfmerge -y -d otherxnf otherxnf\calc.xnf
otherxnf\calc.xff
Netlist written to file otherxnf\calc.xff
------------------------------------------------------------
XSIMMAKE COMMAND : xfind otherxnf\calc.xff calc.xfw calc.xgs

LIST OF WARNINGS FOUND for XFIND
------------------------------
WARNING 2: No special blocks found in otherxnf\calc.xff
0 Errors and 1 Warnings occured during processing.
------------------------------------------------------------
XSIMMAKE COMMAND : xnfmerge -z -d otherxnf -d xnf -d .
otherxnf\calc.xnf otherxnf\calc.xff
------------------------------------------------------------
XSIMMAKE COMMAND : xnf2inf otherxnf\calc.xff calc.vst u=true

*****************************************
* YOU HAVE SPECIFIED UNIT DELAY TRANSLATION *
*****************************************

Writing the NRF file "calc.nrf" .....
Writing the VST file "calc.vst" .....
Writing the AST file "calc.ast" .....
Writing the ATR file "calc.atr" .....

************************************************************
* BEFORE BEGINNING YOUR SIMULATION, BE SURE TO EXPLICITLY *
* SPECIFY THE FILE EXTENSION AS 'VST', INSTEAD OF 'INF'.  *
* EXAMPLE:                                                 *
* simulate calc.vst [/t] [/a]                              *
* NOTE: IF NO EXTENSION SPECIFIED, OrCAD USES 'INF' AS    *
* DEFAULT.                                                 *
************************************************************
```

Figure 12-3 Partial Xsimmake.out File for Functional Translation

Files Created by XSimMake

Several simulation input files are created by the XSimMake functional flow. These files are:

- calc.vst — OrCAD simulation netlist
- calc.nrf — Signal name aliases. This file is useful in determining new names that may have been assigned during translation. (See Figure 12-4.)
- calc.ast — ASCII stimulus information from the schematic (See Figure 12-5.)
- calc.atr — ASCII trace information from the schematic (See Figure 12-6.)

```
# ALL XILINX SYMBOLS RENAMED TO UNIQUE REFERENCE DESIGNATOR
# NAMES

G1 =   U44
G2 =   CONTROL/STATMACH/U185
G3 =   CONTROL/STATMACH/U184
.
.
.
G414=  DEBOUNCE/U72-AND
G415=  OSC_3K/U51-AND
G416=  OSC_3K/U50-OR

# LONG XILINX SIGNAL NAMES RENAMED TO SHORT ORCAD NAMES

$1 =   CONTROL/STATMACH/SEL_OP
$2 =   CONTROL/CE_ADDR
$3 =   CONTROL/STATMACH/INT2
.
.
.
$253=  DEBOUNCE/U70/$U2_G
$254=  DEBOUNCE/U69/$U1_P
$255=  DEBOUNCE/U69/$U2_G
```

Figure 12-4 Partial Calc.nrf File, Signal Name Aliases

```
STIMULUS_SPECIFICATION

CONTEXT: .
NAME: CLK_1
INITIALSTATE: 0
L1
500: T
1000: GOTO L1
END

CONTEXT: .
NAME: EXC_P
INITIALSTATE: 1
1000: 0
3000: 1
6000: 0
8000: 1
END

CONTEXT: .
NAME: SW6_1
INITIALSTATE: 0
1000: 1
END

CONTEXT: .
NAME: SW5_1
INITIALSTATE: 0
1000: 1
END

CONTEXT: .
NAME: SW4_1
INITIALSTATE: 0
6000: 1
END

CONTEXT: .
NAME: SW3_1
INITIALSTATE: 0
1000: 1
END

CONTEXT: .
NAME: SW2_1
INITIALSTATE: 0
END

CONTEXT: .
NAME: SW1_1
INITIALSTATE: 0
1000: 1
END
```

```
CONTEXT: .
NAME: SW0_1
INITIALSTATE: 0
END
```

Figure 12-5 Calc.ast File, Stimulus Information

```
TRACE_SPECIFICATION

DISPLAYNAME: CLK
TYPE: Signal
TraceOn: ON
DisplayOn: ON
CONTEXT: .
TRACENAME: CLK_1

DISPLAYNAME: EXC_P
TYPE: Signal
TraceOn: ON
DisplayOn: ON
CONTEXT: .
TRACENAME: EXC_P

DISPLAYNAME: WE
TYPE: Signal
TraceOn: ON
DisplayOn: ON
CONTEXT: .
TRACENAME: WE_1

DISPLAYNAME: ALU0
TYPE: Signal
TraceOn: ON
DisplayOn: ON
CONTEXT: .
TRACENAME: ALU0_1

DISPLAYNAME: ALU1
TYPE: Signal
TraceOn: ON
DisplayOn: ON
CONTEXT: .
TRACENAME: ALU1_1

DISPLAYNAME: ALU2
TYPE: Signal
TraceOn: ON
DisplayOn: ON
CONTEXT: .
TRACENAME: ALU2_1

DISPLAYNAME: ALU3
```

```
TYPE: Signal
TraceOn: ON
DisplayOn: ON
CONTEXT: .
TRACENAME: ALU3_1

DISPLAYNAME: STACK0
TYPE: Signal
TraceOn: ON
DisplayOn: ON
CONTEXT: .
TRACENAME: STACK0_1

DISPLAYNAME: STACK1
TYPE: Signal
TraceOn: ON
DisplayOn: ON
CONTEXT: .
TRACENAME: STACK1_1

DISPLAYNAME: STACK2
TYPE: Signal
TraceOn: ON
DisplayOn: ON
CONTEXT: .
TRACENAME: STACK2_1

DISPLAYNAME: STACK3
TYPE: Signal
TraceOn: ON
DisplayOn: ON
CONTEXT: .
TRACENAME: STACK3_1
```

Figure 12-6 Calc.atr File, Trace Information

Converting Stimulus and Trace Files to Binary Format

You now have a netlist and two files, AST and ATR, containing stimulus and trace information. However, OrCAD VST needs this data in binary form. The OrCAD ASCTOVST program converts the ASCII files to binary files.

1. From XDM, pull down the **Verify** menu and select **ASCTOVST → CALC.AST → Done**.

   The calc.ast file is converted to binary format and written to calc.stm.

2. Press any key to return to XDM.
3. Once again, from the **Verify** menu select **ASCTOVST → CALC.ATR → Done**.

   Calc.atr is converted to binary format and written to calc.trc.
4. Press any key to return to XDM.

Note: ASCTOVST is a memory-intensive program. If it fails with the error message "Insufficient conventional memory for data buffers," terminate XDM by typing **exit**↵. Type **asctovst calc.ast**↵ at the DOS prompt to convert the stimulus file, and **asctovst calc.atr**↵ to convert the trace file. Type **xdm**↵ to return to XDM, and continue with the tutorial.

Configuring OrCAD VST for the Particular Design

You should already have configured the simulator for this design, as described in the "Before Starting the Tutorial" section earlier in this chapter. However, it is good design practice to verify these values before entering the simulator.

Performing these steps before each simulation eliminates the most common error made by Xilinx users simulating with OrCAD VST: simulating from the INF file rather than the VST file.

1. Select **DesignEntry → ORCAD** to enter the OrCAD design environment.
2. Select **Digital Simulation Tools → Execute → Simulate → Local Configuration → Configure SIMULATE**.

   The Configure Simulate screen appears.

Note: OrCAD's configuration programs are memory intensive. If you receive the message `Could not find the .EXE, or not enough memory to load \orcadexe\VST_CLC.EXE`, return to the XDM executive screen. Close XDM by typing **exit** ↵, type **orcad** ↵, and verify the configuration as described in this section. Type **xdm** ↵ to return to XDM, and continue with the tutorial.

3. Look in the File Options portion of the screen. The file names should read as follows:

```
Connectivity database  CALC.VST
Stimulus file          CALC.STM
Trace file             CALC.TRC
```

4. If the names are incorrect, make the necessary changes.
5. Click on **OK**.

Setting these values before entering the simulator is the easiest way to select the files. You must do this each time you start a new project or want to simulate from an input file of a different name.

Adding Stimulus Data Using OrCAD's Stimulus Editor

You have created input vectors by adding stimulus to the schematic. Another method is to add an input stimulus using the Stimulus Editor inside the simulator. This section describes this method of defining input for the global reset signal in your schematic.

XC2000/XC3000 Families Reset Signal

For XC2000 and XC3000 family devices, the global reset signal is called GR, for Global Reset. It is an active Low signal that resets all of the flip-flops in the design. GR comes up Low, in the active state, to initialize the device, just as it does in the actual FPGA. In this simulation you set this signal High at 1 ns.

You cannot place this stimulus on the schematic, because the GR signal is not accessible through the schematics; it is an integral part of the silicon and is not user-programmable. This dedicated net is connected to the Reset pad on each XC2000 or XC3000 family FPGA.

XC4000 Family Reset Signal

For XC4000 family devices, the corresponding signal is called GSR, for Global Set Reset. It is an active High signal that sets or resets each flip-flop in the device, Whether a flip-flop is set or reset depends on whether it is an FDR or an FDS flip-flop, or on the value of the INIT attribute attached to the flip-flop. Unless you deliberately use a set flip-flop, the flip-flop will reset, since this is the default configuration.

GSR comes up High, in the active state, to initialize the device, just as it does in the actual FPGA. In this simulation you set this signal Low at 1 ns.

You can place the GSR stimulus on the schematic if you are using the STARTUP symbol to gain access to the global set reset signal. If you want access to the Global Set Reset net from one of the pins of the XC4000 FPGA, place the STARTUP component in your schematic and attach an IPAD and IBUF to the GSR pin. This pad becomes an active-High Global Set Reset signal. You can also use an internally generated signal to drive the GSR pin of the STARTUP component. There is an active-High Global Three State signal (GTS) that you can access in the same way. See the *XACT Libraries Guide* for more information on the Startup symbol.

Accessing the Stimulus Editor

Enter the OrCAD simulator and bring up the stimulus editor.

1. Select **`Simulate → Execute`**.

   Occasionally an error message appears at this point in a simulation. If the following message appears:

   ```
   PAD in X3K_LIB not found. Delete Device or EXIT Simulator?
   ```

   it means that an INF file is being loaded into the simulator instead of a VST file. If this message appears, exit the simulator and change the Local Configuration under Simulate to select **`CALC.VST`** as the connectivity database.

2. Press ↵ to access the menu. (From now on, it is assumed that you know how to access the VST commands.)

3. Select **`Edit Stimulus → Yes`**.

   You are now in the stimulus editor. A list of existing stimulus data appears.

   The CLK signal has been renamed to CLK_1, and the SW signals have been similarly altered. The OrCAD software renames all internal signals to add the _1. The EXC_P signal, which comes from a pad, is not changed. The _1 signifies an internal signal on sheet number one.

Adding a New Stimulus

Perform the following commands to add the necessary stimulus to the appropriate global reset signal.

1. To add the new stimulus, select **Add**.
2. Move the cursor to the `Signal Name` field.
3. Select **Browse**.

   A list of signal names appears.
4. Type **g**.

   This step performs a search for names beginning with G. It demonstrates one advantage of using meaningful labels.
5. Select **GR** or **GSR**, depending on whether you are simulating an XC3000 or an XC4000 family design.
6. Move the cursor to `Initial Value` and select either **0** for GR or **1** for GSR.
7. Move the cursor to `End Stimulus` and select **Add**.

   You are prompted by `Time of Function?`.
8. Enter **10** ↵ to set a time of 1.0 nanoseconds.

   The `Function?` prompt appears.
9. Select either **1** for GR or **0** for GSR.
10. Return to the stimulus editor by selecting **Return**.
11. Save the new stimulus file and overwrite the old one by selecting **Write → Yes**.

    This step overwrites the old calc.stm file, which now includes the stimulus for the global reset signal. But the new stimulus file has not yet been read into the simulator. If you ran a simulation now, the global reset signal would not deactivate, and the flip-flop outputs would remain reset throughout the simulation.
12. To read the new stimulus file into the simulator, select **Use → Stimulus**.

    This command also exits the Stimulus Editor and returns you to the simulation screen.

This method allows you to apply a stimulus to a signal that does not exist on the schematic. It is also very useful for making changes quickly without starting over and reprocessing the entire design from the schematic level. On the other hand, stimuli created and saved in this way do not exist in an ASCII text format, so they are difficult to reproduce without the original binary STM file.

You can also directly edit the ASCII AST and ATR files, and rerun ASCTOVST to regenerate the binary STM and TRC files. This last method provides you with stimulus and trace information in a permanent, editable form.

Design Description

The Calc design consists of a four-bit processor with a stack. Inputs are a seven-bit bus, SW[6:0], which defines the opcode and data, and an Execute switch, EXC_P. Whenever EXC_P goes high, the processor reads the opcode and data and executes the defined command.

The ALU performs functions between an internal register and either the top of the stack or data read in from the external switches. Outputs include ALU[3:0], the current contents of the internal register, and STACK[3:0], the top value in the stack.

For a more detailed description of the Calc design, see the "Design Description" section of the "SDT Tutorial" chapter. Additional information, including a table of opcodes, appears in the "Testing the Design" section of the same chapter.

Performing the Functional Simulation

The waveform data may not all fit in RAM, so you can spool the data to disk as it is created.

1. Select **Set → Spool to Disk**.

   The `Spool Trace Data to Disk?` prompt appears.

2. Select **Yes → Yes** to spool the data to the spool.tdf file.

Keyboard Shortcut: **Set → Spool to Disk → Yes → Yes** corresponds to the "F5" Key Macro. You can perform steps1 and 2 by pressing the **F5** function key.

The default time scale is 10.0 ns per screen. Adjust the time scale to

1,000.0 ns per screen, which means increasing the scale by a factor of 100. (About 1,200 ns will actually be visible on the screen.)

3. Select **Trace → Change View**.

   The `Trace Delta Time?` prompt appears at the top of the screen.

4. Type **100** ↵.

   You are ready to simulate the design using your stimulus and trace data.

5. Select **Run Simulation**.

6. Type **12000** ↵ to simulate for 1,200 ns.

Keyboard Shortcut: **Run Simulation → 10000** corresponds to the "F3" Key Macro, and **Run Simulation → 1000** corresponds to the "F2" Key Macro. You can perform step 6 by typing the **F3** function key once and the **F2** function key twice.

The resulting waveforms are displayed as in Figure 12-7. Whenever EXC_P goes High, the next rising CLK edge executes the opcode selected by the SW switches (not displayed). First, the value 1010 is loaded into the ALU. Then the next command, a PUSH to the stack, writes 1010 into the stack. The PUSH command takes longer to complete because it requires extra states in the state machine.

Debugging the Functional Simulation

If your results do not match those displayed in Figure 12-7, first make sure that your traces are not simply displayed in a different order than those in the figure. If the output is actually incorrect, you may have one of the problems discussed in this section.

If all outputs are Low, it is probably because you did not set GR High or GSR Low at 1 ns to deactivate the global reset.

If all outputs are Unknown, you may not have run XDraft, or you may have run XDraft with the -S option, which does not configure the VST simulator. One of the tasks performed by XDraft is to set the prefix definition to 'G'. If the prefix definition 'G' is not defined as described in the "XDraft and the Vst.cfg File"section earlier in the tutorial, all outputs are always Unknown. Exit OrCAD, rerun XDraft as described in the "Skipping the SDT Tutorial" section earlier in the

tutorial, and try again.

If the circuit simulates, but you do not see the expected output, you may have made an error in the circuitry or while entering stimulus data. Return to the schematic editor, make the necessary corrections, and rerun XSimMake before simulating again. Alternatively, you can copy the simulation solution schematic, calcsim.sch, from the appropriate solutions directory. Either way, to see the changes to the stimulus or trace data, you must delete the existing AST and ATR files before rerunning XSimMake, then use ASCTOVST to translate the new stimulus and trace files. The XNF2VST program called by XSimMake does not overwrite existing AST and ATR files, because users often modify the stimulus and trace files created from the schematic.

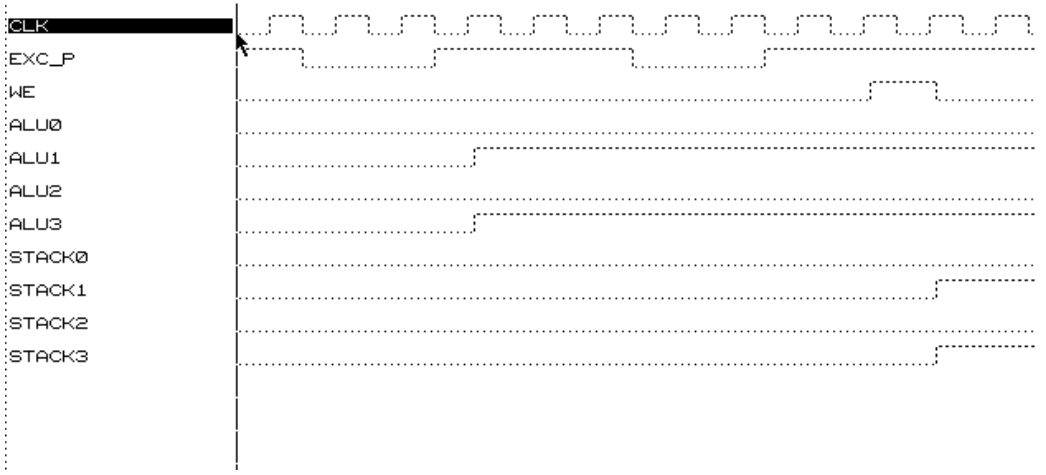


Figure 12-7 Results of Functional Simulation of Calc

Useful Simulation Commands

Experiment with the commands in the VST menus. Useful commands for checking simulation output are:

- Hardcopy — Print the waveforms (plotter must be properly configured)
- Initialize — Restart the simulation at time zero

- Place Marker — Measure times between two points
- Delete Marker — Remove measurement marker
- Quit — Leave the simulation environment
- Verify — Examine or short (force) signal values
- Zoom waveforms — Change the scale or time of displayed

See the *OrCAD Digital Simulation Tools Reference Guide* and *Users Guide* for more information about these commands.

Exiting the OrCAD Simulator

When you have finished testing the simulation commands, close the simulator and return to the XDM environment.

1. Select **`Quit → Abandon Simulation → Yes → To Main → Execute → Exit ESP → Execute`**.

   The following message may appear on the screen:

   Local configuration(s) have changed. Do you want to save the changes?

2. If this message appears, click on **`Yes`**. Otherwise, press any key to return to XDM.

Functional Command Summary

Although this tutorial uses XSimMake to process the Calc design, you do not have to use XSimMake if you do not wish to.

You can run any of the following programs from XDM, or bypass XDM and run them from the system prompt. The commands are listed below as you would type them at the system prompt or in a batch file.

The command summaries assume that you have specified the part type in the schematic. If you prefer to specify the part type on the command line, call XSimMake or SDT2XNF with the -P option and specify the part type.

To run XSimMake from the system prompt or a batch file, type:

```
xsimmake -f orcad_fpga_func calc
```

or

```
xsimmake -f off calc
```

The -f option specifies the flow.

Alternatively, you can run the individual programs from the system prompt or from a batch file. The command sequence varies depending on whether or not Xilinx ABEL files are included, whether X-BLOX elements are placed in the schematic, and so forth. XSimMake automatically uses the correct command sequence for each type of design. The following command sequence applies only to basic schematics without special Xilinx symbols such as X-BLOX or Xilinx ABEL symbols.

a**nnotate calc.sch**	Update reference designators
inet calc.sch	Generate INF (OrCAD netlist) files
sdt2xnf calc -d otherxnf	Translate INF files to XNF netlists
xnfmerge -d otherxnf -q otherxnf\calc.xnf otherxnf\calc.xff	Combine into one XFF file
xnf2vst -u otherxnf\calc.xff calc	Create VST netlist, AST, ATR files
asctovst calc.ast	Translate ASCII AST to binary STM (Not run by XSimmake)
asctovst calc.atr	Translate ASCII ATR to binary TRC (Not run by XSimmake)

Performing a Timing Simulation

The basic steps involved in the timing simulation of a Xilinx design are similar to those of the functional simulation flow. Instead of merely creating an XNF file, however, you must fully route the design using XMake before running XSimMake to create the VST netlist.

The basic steps of the timing simulation procedure are the following.

1. Place and route the design with XMake.
2. Create a timing simulation netlist with XSimMake, which performs the following steps:
   - Runs XDelay to backannotate delays to the routed LCA file
   - Generates a Xilinx XNF netlist file from the routed LCA file
   - Restores net names to match the original schematics

- Generates an OrCAD VST simulation file from the Xilinx netlist
- Generates OrCAD stimulus (AST) and trace (ATR) files if they do not already exist

3. Convert AST and ATR files to binary format (STM and TRC) if new files were created in step 2. This step is not necessary if you have already performed a functional simulation, as in this tutorial.
4. Configure OrCAD VST for timing simulation.
5. Perform the timing simulation.

Placing and Routing the Design with XMake

The first step in a timing simulation is to place and route the design. If you worked through the SDT Tutorial, you have already performed this step. Skip to the next section, "Creating a Timing Simulation Netlist with XSimMake."

Use XMake to create a routed LCA file.

1. In XDM, verify that the `Family` and `Part` displayed in the lower left-hand corner are correct for your design. These settings override the parttype setting in your design.
2. If the settings are incorrect, click on **`Family`** and select the correct family, then click on **`InDesign`**.
3. Pull down the **`Translate`** menu and select **`XMAKE`**.

   A list of options is presented. The default options are correct for this application.
4. Select **`Done`**.

   A menu of design files in the current directory is displayed.
5. Identify the top-level drawing file by selecting **`CALC.SCH`**.

   The `Select target:` prompt appears. A menu offers alternatives for XMake. Each option goes further in the translation process than the one above it. You want to create a placed and routed design for timing simulation, but you do not need to make a bitstream.
6. Select **`Make placed & routed design`**.

The XMake program processes all the necessary design files, displaying its progress on the screen. If the translation is successful, XMake issues this message:

```
XMAKE succeeded. Check 'calc.out' for warning
messages.
```

7. Press any key to return to XDM.

XMake produces a routed LCA file that you can use for timing simulations. The most useful report file generated is calc.out, which contains all error or warning messages generated by any subprogram of XMake, as well as much other useful information. See the "SDT Tutorial" chapter for more discussion of the OUT file.

Creating a Timing Simulation Netlist with XSimMake

The flow using XSimMake as described in this section works for simulating any routed LCA file. To learn more about the command sequence used by XSimMake, see the "Timing Command Summary" section of this chapter, and the other chapters of this manual.

Creating a Timing Simulation Netlist

Use XSimMake to create a timing netlist from XDM.

1. Select the **Verify** menu and click on **XSIMMAKE**.
2. Select **-F <flow name>**. If it is already highlighted, click the left mouse button twice to deselect it and select it again.

   A menu of supported command flows appears.
3. If it is not already highlighted, click on **Orcad_Fpga_Timing** to select the timing simulation flow.
4. Select **Done**.

   A list of all routed LCA files in the design directory appears.
5. Select **CALC.LCA** as the input schematic file.

   XSimMake routes text output to the screen, so that you can follow the conversion process and watch for error messages. A portion of the XSimMake output for the timing flow is displayed in Figure 12-3.

Note: XSimMake flows vary depending on the design. The flow used by XSimMake for your design may be slightly different from the flow shown in this tutorial.

No errors or warnings should be reported.

6. Press any key to return to XDM.

```
------------------------------------------------------------
XSIMMAKE COMMAND : xdelay -w -d calc.lca
xdelay: No paths traced for design calc.lca...
Xilinx LCA xdelay Ver. 4.7.3 ended normally
------------------------------------------------------------
XSIMMAKE COMMAND : lca2xnf -g calc.lca calc.xnf
Simulation model written to file calc.xnf
------------------------------------------------------------
XSIMMAKE COMMAND : xnfba calc.xff calc.xnf


File xnfba.rpt created.
File xnfba.xnf created.
------------------------------------------------------------
XSIMMAKE COMMAND : xnf2inf xnfba.xnf calc

*********************************************
* YOU HAVE SPECIFIED TIMING DELAY TRANSLATION *
*********************************************

Writing the NRF file “calc.nrf” .....
Writing the VST file “calc.vst” .....
Writing the DBA file “calc.dba” .....

************************************************************
* BEFORE BEGINNING YOUR SIMULATION, BE SURE TO EXPLICITLY  *
* SPECIFY THE FILE EXTENSION AS ‘VST’, INSTEAD OF ‘INF’    *
* EXAMPLE:                                                 *
* simulate calc.vst [/t] [/a]                              *
* NOTE: IF NO EXTENSION SPECIFIED, OrCAD USES ‘INF’ AS     *
* DEFAULT.                                                 *
************************************************************
```

Figure 12-8 Partial Xsimmake.out File for Timing Translation

Files Created by XSimMake

Output files created by the XSimMake timing flow include the following.

- xsimmake.out Screen output from all programs run by XSimMake
- calc.vst OrCAD simulation netlist
- calc.dba Delay back-annotation file with timing information for each signal; must be present during timing simulation
- calc.ast ASCII stimulus information from the schematic (This file is not created if there is already an existing file, as in this tutorial.)
- calc.atr ASCII trace information from the schematic (This file is not created if there is already an existing file, as in this tutorial.)

Converting AST and ATR Files to Binary Format

Because you already have valid STM and TRC files from the functional simulation, you do not need to run this conversion before performing a timing simulation of the Calc design. For instructions on how to perform this conversion, see the "Converting Stimulus and Trace Files to Binary Format" section under "Performing a Functional Simulation," earlier in this tutorial.

Configuring OrCAD VST for the Particular Design

Verify that the simulator is correctly configured to simulate the input file, calc.vst. Then configure the simulator to use timing delays from the DBA file.

1. Select **`Design Entry → ORCAD`** to enter the OrCAD design environment.
2. Select **`Digital Simulation Tools → Execute → Simulate → Local Configuration → Configure SIMULATE`**.

   The Configure Simulate screen appears.

Note: OrCAD's configuration programs are memory intensive. If you receive the message `Could not find the .EXE, or not enough memory to load \orcadexe\VST_CLC.EXE`, return to the XDM executive screen. Close XDM by typing **`exit`** ↵, type **`orcad`**

↵, and complete the configuration as described in this section. Type **xdm** ↵ to return to XDM, and continue with the tutorial.

3. Under File Options, verify that the file names read as follows:

```
Connectivity database  CALC.VST
Stimulus file          CALC.STM
Trace file             CALC.TRC
```

4. If the names are incorrect, make the necessary changes.
5. Set the simulator to use timing delays from the DBA file by selecting **Use Delay Annotation.**
6. Return to the ESP screen by seleting **OK**.

Performing the Timing Simulation

You are ready to perform the timing simulation. Since you are using the same stimulus files as in the functional simulation, you do not need to use the Stimulus Editor to release the global reset signal.

1. Enter the simulator by selecting **Simulate → Execute**.
2. Select **Set → Spool to Disk**.

   The `Spool Trace Data to Disk?` prompt appears.
3. Select **Yes → Yes** to spool the data to the spool.tdf file.

   The default time scale is 10ns per screen. Adjust that to 1000ns per screen, which means increasing the scale by a factor of 100.
4. Select **Trace → Change View**.

   The `Trace Delta Time?` prompt appears at the top of the screen.
5. Type **100** ↵.
6. Select **Run Simulation**.
7. Type **12000** ↵ to simulate for 1,200 ns.

   The resulting waveforms are superficially the same as the functional waveforms displayed in Figure 12-7.

   However, this time the waveforms reflect the delays from the routed design.

8. Zoom in on the traces with **Zoom → Scale → 4**, and verify the delay between clock high and WE going high.

Timing Command Summary

Although this tutorial uses XSimMake to process the Calc design, you do not have to use XSimMake if you do not wish to.

You can run any of the following programs from XDM, or bypass XDM and run them from the system prompt. The commands are listed below as you would type them at the system prompt or in a batch file. The same command sequence is valid for any Xilinx FPGA design. The starting point is a placed and routed LCA file created by XMake.

To run XSimMake from the system prompt or a batch file, type:

```
xsimmake -f orcad_fpga_timing calc
```

or

```
xsimmake -f oft calc
```

The -f option specifies the flow.

Alternatively, you can run the individual programs from the system prompt or from a batch file.

xdelay -w -d calc.lca Add delays to LCA file

lca2xnf -g calc.lca calc.xnf
Translate LCA file to XNF netlist

xnfba calc.xff calc.xnf
Restore original net names to XNF file

xnf2vst xnfba.xnf calc Generate OrCAD simulation netlist

If you had no existing stimulus and trace files, new AST and TRC files were created by XNF2VST. Convert the files to binary format.

asctovst calc.ast Translate ASCII AST to binary STM (Not run by XSimMake)

asctovst calc.atr Translate ASCII ATR to binary TRC (Not run by XSimMake)

Using the OrCAD Trace Editor

Earlier in the tutorial you used the Stimulus Editor to add additional stimulus to the simulation. Similarly, you can use the Trace Editor to trace additional signals or buses.

1. Select **Trace → Trace Edit**.

   You are now in the Trace Editor, which displays the currently defined traces. You previously defined these in the schematic.

   Add a new signal to be traced. The new signal is the ALU register value in bus format.

2. Select **Add → Edit**.

   The cursor is in the Display Name field.

3. Type **ALUBUS** ↵.

4. Move the cursor to Type. Select **DecimalBus**.

5. Move the cursor to the Name column opposite Bit 0. Select **Browse** and type **a**.

   The list of signal names scrolls to the names starting with "A."

6. Use the cursor or the PgDn key to scroll through the signal names beginning with "A."

7. Select **ALU0_1 → Next → Next → Next**.

8. Exit the Trace Editor and overwrite the old TRC file by selecting **Return → Write**.

   The CALC.TRC? prompt appears.

9. Select **Yes**.

   This procedure overwrites the old calc.trc file. The new file includes a trace record for the ALU bus. This file is in binary format, so it cannot be viewed and is not very useful as documentation.

   As with the stimulus file created earlier, the new trace file is not seen by the current simulation session until it is explicitly read in.

10. To read the new trace file into the simulator, select **Use Trace**. This command also exits the Trace Editor and returns you to the simulation screen.

Turn back to Table 12-1 and review the function key macros defined in the vstmac.mac file.

11. Press **F5** to initialize the simulator and set Spool to Disk on.
12. Select **Trace → Change View** and type **100** ↵ to set the screen width to 10,000 ns.
13. Press **F3 → F2 → F2** to simulate for 1,200 nanoseconds.

The ALUBUS bus value is displayed as a decimal number. On color screens, bus outputs are displayed in red.

Using the OrCAD Breakpoint Editor

In addition to the Stimulus Editor and the Trace Editor, OrCAD VST has a Breakpoint Editor that you can set to stop the simulation automatically when specified signals reach specified values. As an example, this section shows you how to set a breakpoint to stop the simulation when ALUBUS changes to a value of decimal ten.

Inserting a Breakpoint

Define the conditions under which you want the simulation to stop.

1. Re-initialize the simulation by pressing the **F5** function key.
2. Select **Breakpoint → Breakpoint Edit**.

   The cursor is on breakpoint 1.
3. Select **Edit**.

   The cursor moves to the Status field.
4. Select **Enable**.
5. Move the cursor to Type and select **And**.
6. Move the cursor to `* Last Record *` and type **b** to select Browse.
7. Type **a**.
8. Scroll down with the mouse, the arrow keys, or the PgDn key and select **ALU0_1**.

9. Move the cursor down to `* Last Record *` again. Repeat the last few steps to add ALU1_1, ALU2_1 and ALU3_1 using **`Browse`** and **`a`**.
10. Place the cursor on ALU0_1 and select **`Edit`**.
11. Change the name to **`~ALU0_1`** ↵.

    This step sets the breakpoint to check for a Low on ALU0 instead of a High.
12. Place the cursor on ALU2_1 and select **`Edit`**.
13. Change the name to **`~ALU2_1`** ↵.
14. Select **`Return`** to exit the breakpoint editor.
15. Select **`Write`** to save the breakpoint file.

    The message `Write Breakpoint file?` appears at the top of the screen.
16. Type **`calc`** ↵.

    The `CALC.BRK?` prompt appears.
17. Select **`Yes`**.
18. Select **`Use`** to read the breakpoint file, calc.brk, into the simulator.

    `<B>` appears in the upper left-hand corner, indicating that a breakpoint is active.

    The simulator will stop when ALUBUS reaches the transition from any other value to a value of 1010, binary ten.

Resimulating the Design

Resimulate the design and watch for the breakpoint when the load to the ALU register becomes effective.

1. Press **`F3`** to simulate for 1000 ns, which is enough to get past the first breakpoint.

   The simulator stops at the selected breakpoint. The following message appears:

   ```
   <B> BREAKPOINT 1 encountered !!! Type any key to continue.
   ```

2. Type any key.

   <B> disappears.

   The breakpoint is disabled. At this point you could re-enable the breakpoint with the Breakpoint Enable Breakpoint command to break the next time the ALU register value changed to ten.

Creating Tabular Output

OrCAD VST offers a semi-automated way to create test vectors directly from simulation files. The first step in creating test vectors is to capture trace output in tabular format. You can then modify this data using any text editor and use it as an input to future simulation.

1. Re-initialize the simulation by selecting **F5**.
2. Save the values of each traced node whenever one of them changes. Select **Set → Print on Change → Yes**.

   The default file name offered is prtonchg.tvs; however, in this example you call the output calc.tvs.

3. Select **No**.

   You are prompted to enter a file name.

4. Type **calc.tvs** ↵.
5. Select **Yes**.
6. Simulate for 1,200 ns by selecting **Run Simulation** and typing **12000** ↵. Alternatively, you can press **F3 –> F2 –> F2**.

   Tabular output is written to the calc.tvs file as the design is simulated.

7. Exit the simulator by selecting **Quit –> Abandon Edits –> Yes**.
8. Move the cursor to the blue bar at the top of the ESP screen. Click the left mouse button.

   A menu appears.

9. Select **Suspend to System**.

Use any text editor or the DOS Type command to look at the tabular output in the calc.tvs file. The results should be the same as those in Figure 12-9.

You can modify the TVS file using any text editor or the VST Test Vector Editor. It can be used as input stimulus in future simulations. See the *OrCAD Digital Simulation Tools User Guide* for detailed instructions.

You have completed the functional and timing simulation of a simple Xilinx design. In the process, you learned how to configure the OrCAD environment to work with Xilinx designs, how to process a schematic for functional simulation, and how to perform a timing simulation on a placed and routed design. You were then introduced to some basic features of the OrCAD VST simulator. Information on the simulator is available in the OrCAD manuals; issues specific to Xilinx designs are covered in the other chapters of this manual.

```
;           . . . . . . . . . . .
;           C E W A A A A S S S S
;           L X E L L L L T T T T
;           K C _ U U U U A A A A
;           _ _ 1 0 1 2 3 C C C C
;           1 P _ _ _ _ K K K K
;           1 1 1 1 0 1 2 3
;           _ _ _ _
;           1 1 1 1
;

0000000000  0 1 0 0 0 0 0 0 0 0 0
0000000500  1 1 0 0 0 0 0 0 0 0 0
0000001000  0 0 0 0 0 0 0 0 0 0 0
0000001500  1 0 0 0 0 0 0 0 0 0 0
0000002000  0 0 0 0 0 0 0 0 0 0 0
0000002500  1 0 0 0 0 0 0 0 0 0 0
0000003000  0 1 0 0 0 0 0 0 0 0 0
0000003500  1 1 0 0 0 0 0 0 0 0 0
0000003572  1 1 0 0 1 0 1 0 0 0 0
0000004000  0 1 0 0 1 0 1 0 0 0 0
0000004500  1 1 0 0 1 0 1 0 0 0 0
0000005000  0 1 0 0 1 0 1 0 0 0 0
0000005500  1 1 0 0 1 0 1 0 0 0 0
0000006000  0 0 0 0 1 0 1 0 0 0 0
0000006500  1 0 0 0 1 0 1 0 0 0 0
0000007000  0 0 0 0 1 0 1 0 0 0 0
0000007500  1 0 0 0 1 0 1 0 0 0 0
0000008000  0 1 0 0 1 0 1 0 0 0 0
0000008500  1 1 0 0 1 0 1 0 0 0 0
```

```
0000009000  0 1 0 0 1 0 1 0 0 0 0
0000009500  1 1 0 0 1 0 1 0 0 0 0
0000009572  1 1 1 0 1 0 1 0 0 0 0
0000010000  0 1 1 0 1 0 1 0 0 0 0
0000010500  1 1 1 0 1 0 1 0 0 0 0
0000010572  1 1 0 0 1 0 1 0 0 0 0
0000010688  1 1 0 0 1 0 1 0 0 0 1
0000010695  1 1 0 0 1 0 1 0 1 0 1
0000011000  0 1 0 0 1 0 1 0 1 0 1
0000011500  1 1 0 0 1 0 1 0 1 0 1
0000012000  0 1 0 0 1 0 1 0 1 0 1
END
```

Figure 12-9 Calc.tvs, Tabular Output of Timing Simulation

OrCAD Interface/ Tutorial Guide

X-BLOX Tutorial

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Chapter 13

X-BLOX Tutorial

X-BLOX consists of an advanced library and a synthesis tool that allow you to take advantage of the built-in expert knowledge and the special features of Xilinx XC3000A, XC3000L, XC3100A, and all XC4000 FPGAs. X-BLOX cannot be used on XC3000 designs. By using X-BLOX, you can significantly shorten design entry time, increase design speed, and use the device more efficiently.

This chapter gives a practical example using X-BLOX within the OrCAD design environment. It is not intended to fully explain all of the functionality found within X-BLOX. Please refer to the "Further Reading" section at the end of this tutorial for a list of sources from which to obtain more information.

Before Beginning the Tutorial

This section of the tutorial assumes that you are already familiar with the material in the “SDT Tutorial” and “VST Tutorial” chapters of this manual. If not, please review those chapters before continuing.

Required Software

This tutorial assumes that you are using the following versions of the development software:

- OrCAD / SDT — SDT 386+
- OrCAD / VST — VST 386+
- OrCAD / Xilinx Interface — version 5.00 or later
- XACT Design Manager (XDM) — Version 5.00 or later
- X-BLOX program and libraries, which allow you to use and synthesize X-BLOX library components — Version 5.00 or later.

If you have Xilinx software on CD-ROM, you should have at least temporary access to all of the above software using the temporary licensing available on the programmable key, provided that the temporary licensing has not already been exhausted.

Preparing the Design

If you chose to read through the SDT tutorial rather than actually perform the steps involved, you must verify that your PC is set up correctly to use OrCAD 386+, the XACT Development System software, and X-BLOX. Then create and configure the design directory, and copy a completed set of schematics from any of the solutions directories supplied.

If you have already performed the SDT tutorial on your PC, skip to the next section, "Modifying the Design."

1. Follow the instructions given in the "Before Beginning the Tutorial" section of the "SDT Tutorial" chapter for setting up your design environment.
2. The tutorial files are optionally installed when you install the Xilinx/OrCAD interface software. If you have already installed the software but are not sure whether you specified tutorial installation, check for the c:\xact\tutorial\orcad\calc directory. This directory contains the tutorial files.
3. Create a new project called "Calc," as described in the "Creating the Project Directory" section of the "SDT Tutorial" chapter.
4. Full solutions for the SDT tutorial are supplied in the solutions directories located in \xact\tutorial\orcad\calc. One of the following file directories must be copied to the directory where you will be performing the tutorial.

   ...\soln_3ka — Solution files for XC3020APC68
   ...\soln_4ka — Solution files for XC4003APC84
   ...\soln_4k — Solution files for XC4003PC84

   For example, if you are targeting the tutorial for an XC3020APC68 device, perform the following sequence of commands.

   cd \orcad\calc ↵
   copy \xact\tutorial\orcad\calc*solution.*** ↵

   where *solution* is either soln_3ka, soln_4ka, or soln_4k. The

solutions schematics are targeted towards the 3020APC68-7, 4003APC84-6, and 4003PC84-6 devices, respectively.

Note: This tutorial assumes that your ORCADPROJ variable is set to c:\orcad\. You need not follow this convention.

This procedure gives you a full set of completed schematics for the Calc design, with all supporting files.

5. Configure the design directory by typing **`xdraft 3`**↵ or **`xdraft 4`**↵, depending on whether you are targeting an XC3000A or an XC4000 device.

Note: All of the screen outputs refer to the processing of the 3000A solutions design. Other devices have slightly different outputs.

Modifying the Design

In the Calc design, the ALU block performs many bus-oriented arithmetic logic functions and is ideally suited for implementation using X-BLOX. For more information on the function of the Calc design, refer to the "Design Description" section of the "SDT Tutorial" chapter.

Adding X-BLOX-Based Module to CALC

An X-BLOX-based replacement for the ALU instance on the CALC schematic is inserted in this section. The replacement block is called ALU_BLOX, which is functionally equivalent to ALU, except that ALU_BLOX is implemented using X-BLOX components.

Replace the existing ALU block with the X-BLOX version.

1. Open the top-level CALC schematic in SDT.
2. Place the cursor on the ALU block.
3. Select **`Edit`** → **`Edit`** → **`Filename`**.

   The `Filename? ALU.sch` prompt appears.
4. Backspace over the current name and type **`ALU_BLOX.sch`**↵.
5. Press the **`Escape`** key twice to exit the Edit command.

This procedure replaces the original ALU file reference with ALU_BLOX. The change is reflected by the name ALU_BLOX.sch

appearing at the bottom of the symbol.

Note: OrCAD is case insensitive, so you do not need to follow the case conventions used in this tutorial.

Viewing the ALU_BLOX Schematic

Now save the change to the CALC schematic, and view the schematic for ALU_BLOX by pushing into the ALU_BLOX symbol.

1. Place the cursor on the ALU_BLOX symbol in the CALC schematic.
2. Select **`Quit → Update File → Enter Sheet → Enter`**.

   The schematic for ALU_BLOX appears, as shown in Figure 13-1.
3. Press the **`Escape`** key to exit the Quit Enter Sheet command.

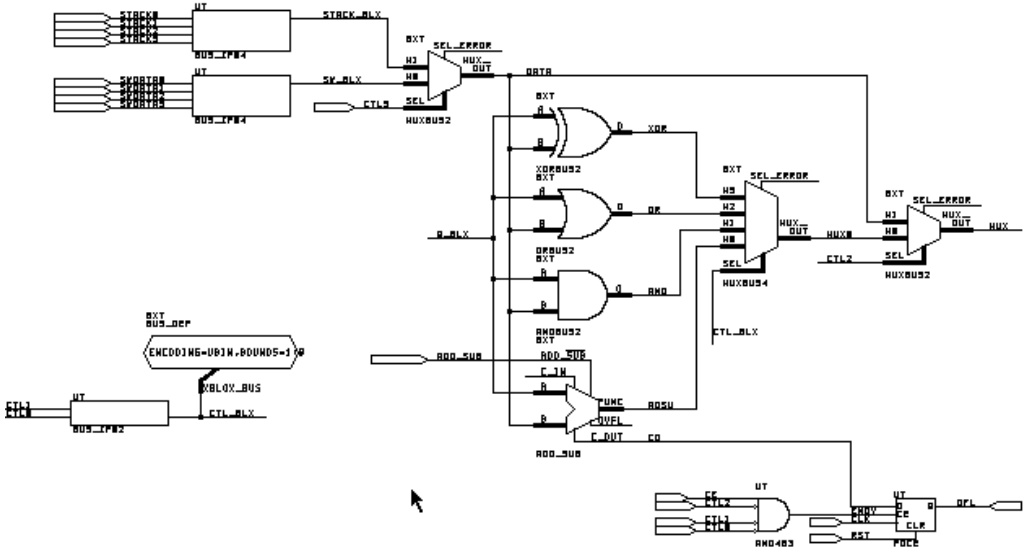


Figure 13-1 Incomplete ALU_BLOX Schematic

Completing the ALU_BLOX Schematic

The ALU-BLOX schematic on your screen is missing some key X-BLOX elements that you add by performing the commands in this section.

Complete the ALU_BLOX schematic, using Figure 13-2 and the following steps as a guide.

1. Select **Get** and type **data_reg**↵.
2. Select **Place** to place the DATA_REG symbol in the space at the lower left corner of the schematic.
3. Use the Place Wire command to add the MUX, RST, CE, and CLK nets, as shown in Figure 13-2. Be sure to label the nets as shown, using the Place Label command.
4. Use the Get command to place the X-BLOX BUS_IF04 component to the right of DATA_REG as shown in the figure.
5. Using the Place Wire command, add a wire connecting the Q_OUT pin of the DATA_REG symbol to the XBLOX_BUS pin of the BUS_IF04 symbol.

Note: Never use the Place Bus command to add an X-BLOX bus. Use the Place Wire command. For a discussion of X-BLOX buses and how they differ from OrCAD buses, see the next section, "Understanding X-BLOX Buses."

6. Label the wire Q_BLX.
7. Place a BUS_DEF component above and between DATA_REG and BUS_IF04, as shown in the figure.
8. Connect the BUS_DEF symbol to the Q_BLX net with a wire segment, and add a junction with the Place Junction command to make the T-connection.
9. Attach dangling wires to the BUS_IF04 pins E00, E01, E02, and E03, and label them Q0, Q1, Q2, and Q3.
10. Add input module ports to RST, CE, and CLK using the Place Module Port command.
11. Add output module ports to Q0, Q1, Q2, and Q3.

At this point, the ALU_BLOX schematic is almost complete.

However, you must still add X-BLOX-specific attributes to complete the schematic.

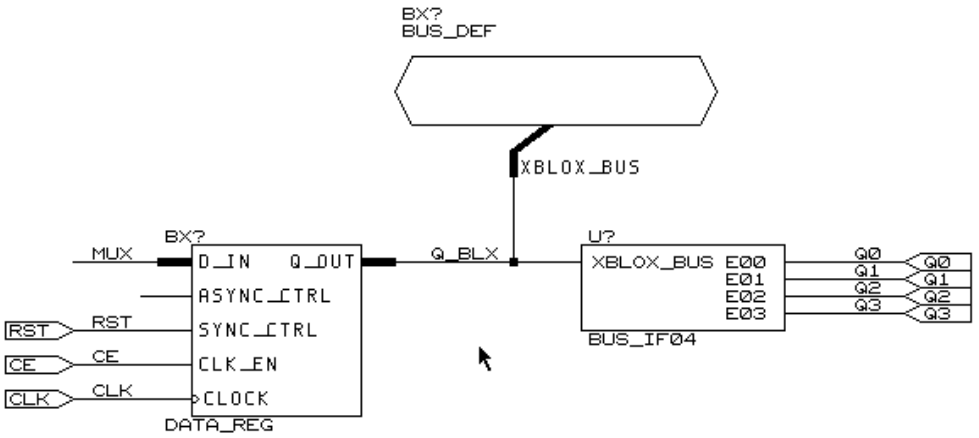


Figure 13-2 Adding X-BLOX Components and Buses

Understanding X-BLOX Buses

In Figure 13-1 and Figure 13-2, the rectangular BUS_IF02 and BUS_IF04 boxes connecting OrCAD nets to X-BLOX buses are bus interfaces. They interface X-BLOX buses with OrCAD nets.

An X-BLOX bus is not the same as a bus normally used in OrCAD. It does not have a defined width, and therefore you must use a wire to represent an X-BLOX bus on your schematic. The width of an X-BLOX bus is not defined by the name attached to the bus. In fact, X-BLOX buses must never be given indexed names such as Q_BLX[3:0], because the bus pins on X-BLOX symbols do not have indexed names. For example, the output pin of DATA_REG on the ALU_BLOX schematic has the unindexed name Q_OUT. All X-BLOX symbols have unindexed bus pins so that the same symbol can be used in any design, regardless of the width of the buses in the design.

If an indexed label is attached to a wire representing an X-BLOX bus, INET flags the label as an error. If an OrCAD bus is used instead of a wire, SDT2XNF does not see the connection. Therefore, BUS_IF symbols are needed as interfaces between X-BLOX buses and OrCAD buses.

The specific BUS_IF symbol required depends upon the width of the bus being interfaced. No interface is necessary for individual nets that connect to X-BLOX symbols, such as CTL2 and CTL3 on the two MUXBUS2 symbols.

Using BUS_DEF Symbols

Where are the X-BLOX bus widths defined? Attached to two buses in the schematic are BUS_DEFs, or bus definition symbols. By adding attributes to these symbols, you can define the properties of the entire data path attached to the BUS_DEF, not just those of the bus to which the BUS_DEF is directly connected. That is why the ALU_BLOX schematic requires only two BUS_DEF elements: one for the 4-bit data path through the ALU, and one for the 2-bit control signal path.

The BOUNDS attribute is placed on a BUS_DEF to define the width of the bus attached to the BUS_DEF, as shown in Figure 13-3. In this case, the data path has a width of four bits, giving "3:0" for the value of BOUNDS.

The ENCODING attribute specifies the type of data being propagated on the data path. The possible choices are UBIN (unsigned binary), BIT (same as UBIN), TWO_COMP (two's complement), or ONE_HOT (one-of-n). The choice of ENCODING value affects the functionality of every symbol on the data path. The ADD_SUB block in the ALU_BLOX schematic, for example, is implemented as an unsigned binary adder / subtracter. If you designated a data type of TWO_COMP, the macro would be implemented as a two's- complement adder / subtracter, with a different implementation of the OFL output.

Only some ENCODING types are appropriate for a given data path. For example, it does not make sense to give the ADD_SUB data path ONE_HOT, or one-of-n, encoding. On the other hand, on the control path for the multiplexer, the other BUS_DEF in the schematic, ONE_HOT encoding would be suitable. If the control lines attached to the multiplexer are encoded as ONE_HOT, you must define ENCODING accordingly. In that case, the choice of ENCODING completely alters the implementation of the multiplexer.

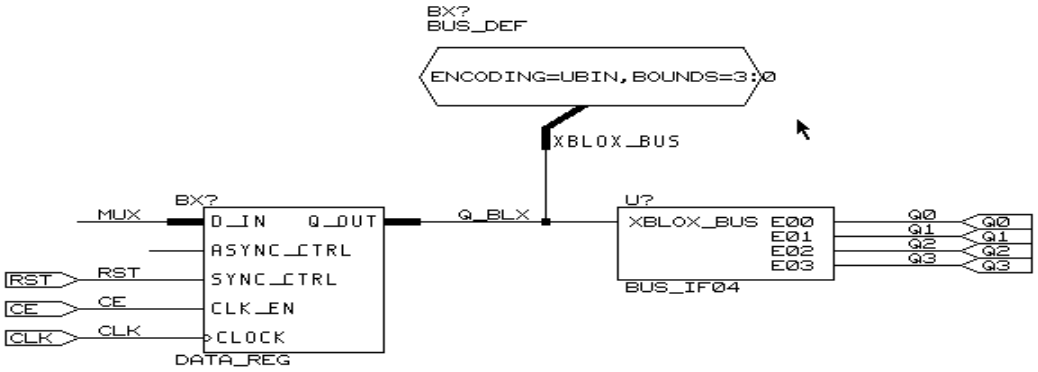


Figure 13-3 Adding ENCODING and BOUNDS Attributes

Completing the Bus Definition

The definition of the ALU data path has not yet been set. Add the following properties to the BUS_DEF symbol attached to the bus named Q_BLX on the bottom of the sheet. Figure 13-3 shows the resulting BUS_DEF symbol with attributes defined.

1. Place the cursor over the BUS_DEF symbol connected to the Q_BLX bus, on the lower portion of the page.
2. Select **Edit → Edit → LOC,OPTIONS → Name** for an XC3000A design, or **Edit → Edit → OPTIONS_1 → Name** for an XC4000 family design.
3. Type **ENCODING=UBIN,BOUNDS=3:0**↵.

   The text appears above the BUS_DEF symbol.
4. Press the **Escape** key to exit the Edit menu.
5. If you wish, you can move the text inside the symbol as shown in Figure 13-3, by selecting **Edit → Edit → LOC,OPTIONS** or **OPTIONS_1 → Location**, moving the cursor inside the BUS_DEF symbol, and selecting **Place**. The placement of the text has no effect on the function of the attributes.
6. Press the **Escape** key if necessary until all menus are cleared from the screen.

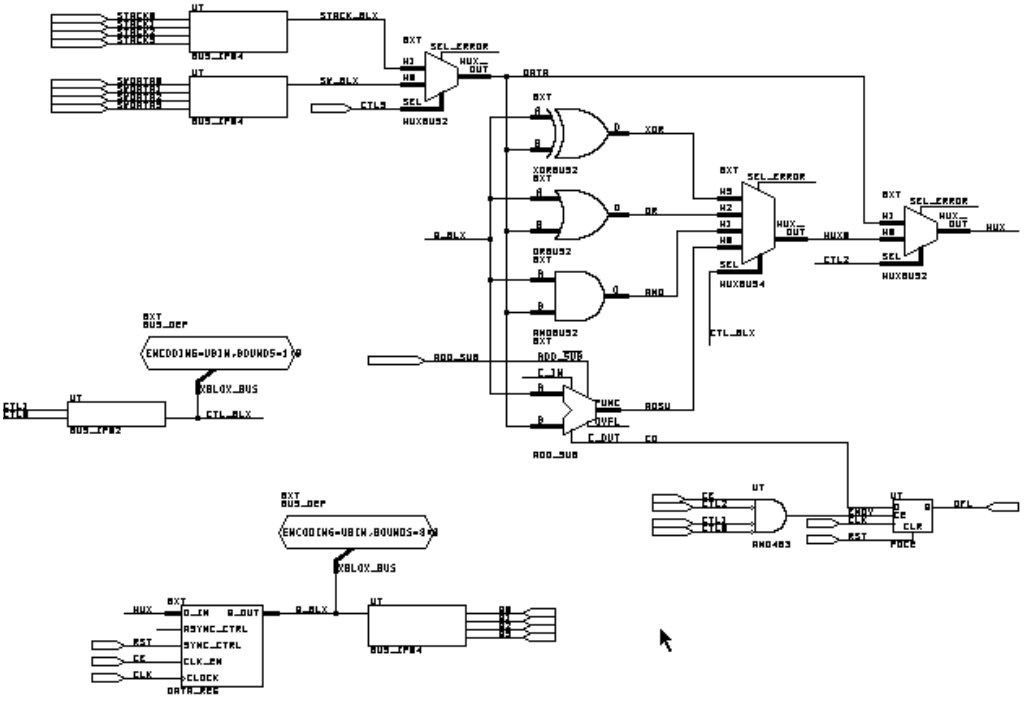


Figure 13-4 Complete ALU_BLOX Schematic

Saving Your Changes

Save the changes you made to the ALU_BLOX schematic before continuing with the tutorial.

1. Select **`Quit`** → **`Update File`** → **`Abandon Edits`** to save your changes and exit SDT.

   The `Abandon Hierarchy?` prompt appears, because you are closing SDT from a schematic that is not at the top level.

2. Select **`Yes`** to return to XDM or the DOS prompt.

X-BLOX Symbol Library

The X-BLOX library contains elements that simplify the design process by providing bus-oriented versions of logic, register, and multiplexing functions. By placing different attributes on X-BLOX symbols, you can customize them for a specific application. Also, the X-BLOX software implements macros differently depending on which pins are used on the symbol. This flexibility allows a wide range of different functions to be implemented using the small set of parts found in the X-BLOX library.

X-BLOX Symbol Examples

The following are examples of how attributes and pin usage affect the implementation of the X-BLOX macros in ALU_BLOX. You may wish to refer to the *X-BLOX User Guide* during this discussion.

- DATA_REG

  DATA_REG in this design has two attributes that can be set to alter its implementation, SYNC_VAL and ASYNC_VAL. These attributes define the value that is loaded in the data register when it is synchronously or asynchronously reset using the SYNC_CTRL and ASYNC_CTRL pins, respectively. In this example, the data register must reset to zero in either case, so both values are undefined and thus default to zero. The SYNC_CTRL pin is connected, specifying a synchronous reset register.

- ADD_SUB

  The ADD_SUB component used in ALU_BLOX is implemented as an adder/subtracter, because the ADD_SUB pin is connected. Since the C_IN pin is unconnected, the block defaults to the proper values for normal adding and subtracting. The implementation of the ADD_SUB macro is greatly affected by the definition of its data path and the pins connected to it.

- ANDBUS2, ORBUS2, XORBUS2, MUXBUSx

  The other X-BLOX symbols on the schematic are implemented the same way as those used in the original ALU design. Their considerable advantage, however, is that you do not need to create any special schematic and symbol for them, much reducing the time necessary to enter the design. The MUXBUSx symbols are

affected by the ENCODING value of their attached buses.

The bused logic symbols, such as ANDBUS2 and ORBUS2, have one very useful attribute that affects their implementation, the INVMASK attribute. By changing INVMASK, you can invert the inputs to the symbol. For example, in order to invert input bit zero on the upper bus connection to the ANDBUS, all that is necessary is to select the ANDBUS and set the value for the INVMASK attribute to 2#0001#. The "1" in the string represents the inversion of bit zero, the "2" indicates that the INVMASK value is specified in binary, with the total number of bits on the bus equal to four. All of the INVMASKs in ALU_BLOX are undefined and thus all default to a value of zero, indicating no bit inversions.

X-BLOX Schematics

X-BLOX macros have a unique ability to adapt to any bus width and to be implemented differently depending on data path encoding and pin usage, so no single schematic can represent the functionality of an X-BLOX macro. The schematic page underneath each X-BLOX macro is "filled in" by the X-BLOX synthesis program, which is run by the two Xilinx program processors XMake and XSimMake. When you first create a design using X-BLOX, no information is available even to functionally simulate the design. You must exit OrCAD to prepare a design containing X-BLOX symbols for simulation with VST.

Functional Simulation

The XSimMake program allows you to easily simulate designs containing X-BLOX components. It coordinates the program execution flow necessary for functional or timing simulation. XSimMake is similar to XMake, except that it produces a schematic that can be simulated instead of producing a bitstream.

Note: For more detailed information on XSimMake, refer to the “Creating a Functional Simulation Netlist with XSimMake” section of the “VST Tutorial” chapter.

Creating the Functional Simulation Netlist

Run XSimMake to generate a netlist that you can functionally simulate.

1. Enter XDM by typing **xdm**↵, if you are not already in XDM.
2. Select **Verify → XSIMMAKE**.
3. Select the **-F** option.

   A menu of possible flows appears.
4. Select the **Orcad_Fpga_Func** flow option.
5. Select **Done**.
6. Choose **CALC.SCH** from the list of schematic files shown.

   XSimMake now executes.
7. After you review the XSimMake output, press any key to return to XDM.

Note: If XSimMake returns errors, check the xsimmake.out, calc.prp, and calc.blx files for details. A completed version of ALU_BLOX is included in each of the solutions directories, with the name BLOXSOLN. If problems cannot be resolved, replace ALU_BLOX with BLOXSOLN on the CALC schematic, save, and try again.

Text similar to the following appears in the XDM window.

Note: XSimMake flows vary depending on the design. The flow used by XSimMake for your design may be slightly different from the flow shown in this tutorial.

```
XSIMMAKE COMMAND : creating directory savexnf
XSIMMAKE COMMAND : creating directory otherxnf
XSIMMAKE COMMAND : annotate calc.sch
XSIMMAKE COMMAND : inet calc.sch /t
XSIMMAKE COMMAND : inf2xnf calc d= otherxnf
XSIMMAKE COMMAND : xnfmerge -y -d otherxnf otherxnf\calc.xnf
otherxnf\calc.xff
XSIMMAKE COMMAND : xfind otherxnf\calc.xff calc.xfw calc.xgs
READING XFW FILE : calc.xfw
XSIMMAKE COMMAND : xnfmerge -z -d otherxnf -d xnf -d .
otherxnf\calc.xnf otherxnf\calc.xff
XSIMMAKE COMMAND : xnfprep otherxnf\calc.xff
otherxnf\calc.xtg drc_only=true
XSIMMAKE COMMAND : xblox otherxnf\calc.xtg otherxnf\calc.xg
archopt=false mergeio=false
XSIMMAKE COMMAND : xnf2inf otherxnf\calc.xg calc.vst u=true
0 Errors and 0 Warnings occurred during processing.
```

Examining XSimMake Output

An explanation of the XSimMake functional flow output seen in XDM follows.

```
XSIMMAKE COMMAND : creating directory savexnf
XSIMMAKE COMMAND : creating directory otherxnf
```

First, XSimMake creates new directories for storage of intermediate files.

```
XSIMMAKE COMMAND : annotate calc.sch
XSIMMAKE COMMAND : inet calc.sch /t
```

XSimMake then runs the OrCAD conversion programs to create an updated OrCAD netlist.

```
XSIMMAKE COMMAND : inf2xnf calc d= otherxnf
```

Next, XSimMake runs INF2XNF to convert the OrCAD INF files to standard Xilinx Netlist Format (XNF) files. INF2XNF is the engine of the SDT2XNF program used by XMake and in the manual flow.

```
XSIMMAKE COMMAND : xnfmerge -y -d otherxnf otherxnf\calc.xnf
otherxnf\calc.xff
```

XMFMerge combines all of the schematic XNF files into a single XNF format file with an .xff extension.

```
XSIMMAKE COMMAND : xfind otherxnf\calc.xnf calc.xfw calc.xgs
```

XFind reads the XNF file to determine what types of symbols the netlist contains. In this case, it discovers X-BLOX symbols and modifies program execution accordingly by generating a file called calc.xfw.

```
READING XFW FILE : calc.xfw
XSIMMAKE COMMAND : xnfmerge -z -d otherxnf -d xnf -d .
otherxnf\calc.xnf otherxnf\calc.xff
XSIMMAKE COMMAND : xnfprep otherxnf\calc.xff
otherxnf\calc.xtg drc_only=true
```

XSimMake reads the calc.xfw file produced by XFind. The calc.xfw file instructs XSimMake to run XNFMerge and XNFPrep in order to prepare the netlist for use as input to X-BLOX. XNFMerge flattens the hierarchical netlists into a single, completely flattened XFF file, while XNFPrep verifies that the XFF file is correct.

```
XSIMMAKE COMMAND : xblox otherxnf\calc.xtg otherxnf\calc.xg
archopt=false mergeio=false
```

X-BLOX implements the X-BLOX modules and optimizes the design.

```
XSIMMAKE COMMAND : xnf2inf otherxnf\calc.xg calc.vst u=true
```

XSimMake then runs XNF2INF to generate a simulation file from the X-BLOX output for use in the VST simulator. XNF2INF is the engine of the XNF2VST program used in the manual flow.

Stimulus and Trace Files

You now have a simulatable VST netlist. In the "VST Tutorial" chapter, you learned how to add stimulus and trace information to your schematic, and how to use the stimulus and trace editors within VST. However, since this tutorial assumes you already know how to create stimulus and trace data, Xilinx has supplied both ASCII and binary stimulus and trace files that you can use to simulate your design. The files are located in the solutions directories discussed in the "Preparing the Design" section of this tutorial. The binary files are are called calcsim.stm and calcsim.trc, respectively.

1. If you copied one of the solutions directories in order to perform this tutorial, the stimulus and trace files are already present in your working directory.
2. If the files do not exist in your working directory, copy them by typing the following commands at the DOS prompt, from your working directory:

   copy \xact\tutorial\orcad\calc*solution.stm** ↵
   copy \xact\tutorial\orcad\calc*solution.trc** ↵

   where *solution* is either soln_3ka, soln_4ka, or soln_4k.

Configuring OrCAD VST for the Calc Design

You must specify the simulation input files before entering VST.

Performing these steps before each simulation eliminates the most common error made by Xilinx users simulating with OrCAD VST: simulating from the INF file rather than the VST file.

1. Select **DesignEntry → ORCAD** to enter the OrCAD design environment.

2. Select **Digital Simulation Tools → Execute → Simulate → Local Configuration → Configure SIMULATE**.

   The Configure Simulate screen appears.

Note: OrCAD's configuration programs are memory intensive. If you receive the message `Could not find the .EXE, or not enough memory to load \orcadexe\VST_CLC.EXE`, return to the XDM executive screen. Close XDM by typing **exit** ↵, type **orcad** ↵, and verify the configuration as described in this section. After completing the commands in this section, type **xdm** ↵ to return to XDM, and continue with the "Performing a Functional Simulation" section, following.

3. Look in the File Options portion of the screen. The file names appear similar to the following:

```
Connectivity database  CALC.VST
Stimulus file          CALC.STM
Trace file             CALC.TRC
```

4. Change the file names to the following:

```
Connectivity database  CALC.VST
Stimulus file          CALCSIM.STM
Trace file             CALCSIM.TRC
```

5. Click on **OK**.

Setting these values before entering the simulator is the easiest way to select the files. You must do this each time you start a new project or want to change the name of one of your input files.

Performing a Functional Simulation

You are ready to enter the OrCAD simulator, VST.

1. Select **Simulate → Execute**.

   The VST waveform display appears on the screen. At the left are listed the nodes from the trace file. No waveforms are visible, since you have not yet run a simulation.

   The waveform data may not all fit in RAM, so you can spool the data to disk as it is created.

2. Select **Set → Spool to Disk**.

   The Spool Trace Data to Disk? prompt appears.

3. Select **Yes → Yes** to spool the data to the spool.tdf file.

   The default time scale is 10.0 ns per screen. Adjust the time scale to 1,000.0 ns per screen, which means increasing the scale by a factor of 100. (About 1,200 ns will actually be visible on the screen.)

4. Select **Trace → Change View**.

   The Trace Delta Time? prompt appears at the top of the screen.

5. Type **100** ↵.

   You are ready to simulate the design using your stimulus and trace data.

6. Select **Run Simulation**.

   The Simulation Length? prompt appears.

7. Type **12000** ↵ to simulate for 1,200 ns.

The resulting waveforms are displayed as in Figure 13-5. Whenever EXC_P goes high, the next rising CLK edge executes the opcode selected by the SW switches (not displayed). First, the value 1010 is loaded into the ALU. Then the next command, a PUSH to the stack, writes 1010 into the stack. The PUSH command takes longer to complete because it requires extra states in the state machine.

The output of this simulation run is identical to the output of the functional simulation run on the original non-X-BLOX Calc design, in the “VST Tutorial” chapter.

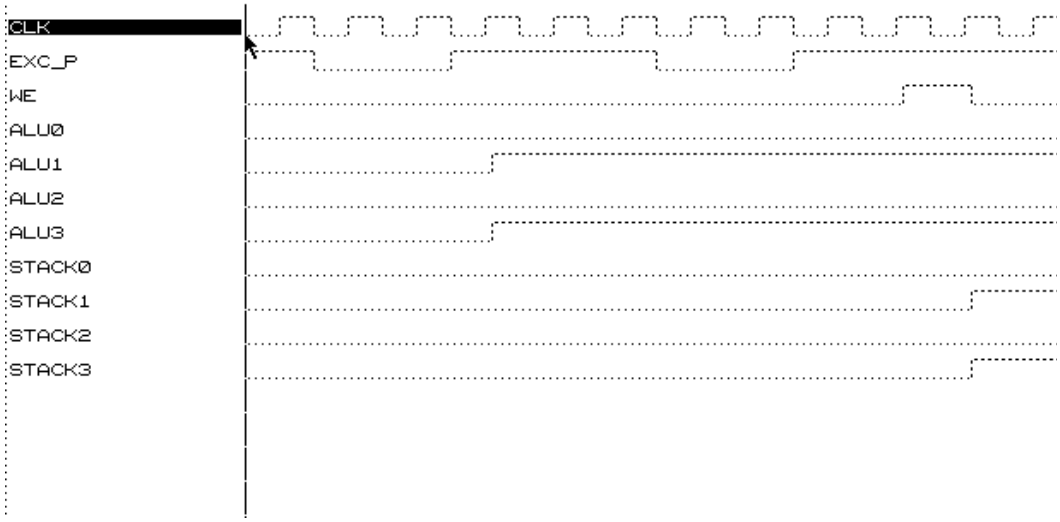


Figure 13-5 Simulation Output for X-BLOX Design

Implementing the Calc Design

The translation of designs containing X-BLOX components is similar to the translation of other designs. You can use XMake to generate files for timing simulation, or to make a bitstream for programming actual devices, just as you use it for non-X-BLOX designs. In the process, XMake runs the X-BLOX program, which synthesizes the X-BLOX macros into standard logic gates.

For detailed information on XMake and the translation process, refer to the "Configuring XDM and XMake" and subsequent sections in the "SDT Tutorial" chapter and to the *XACT Reference Guide*.

Creating a Routed Design

Run XMake to generate files for use in timing simulation and device programming.

1. Quit OrCAD, saving your changes to the VST configuration, and re-enter XDM.
2. Select **Translate → XMAKE**.

3. Using the default options, choose the **CALC.SCH** file as input.
4. Choose **Make bitstream** as the target.

   XMake translates, maps, places, and routes the design.
5. After you review the XSimMake output, press any key to return to XDM.

Examining XMake Output

XMake produces a screen output similar to the following.

```
XMAKE: Generating makefile 'calc.mak' ...
XMAKE: Profile used is the current XDM settings.
XMAKE: Execute command 'annotate calc.sch'.
XMAKE: Execute command 'inet calc.sch'.
XMAKE: Execute command 'sdt2xnf calc.inf calc.xnf -D xnf'.
XMAKE: Set the part type to '3020APC68-7 from 'xnf\calc.xnf'.
XMAKE: Running with the following options: (none)
>>> XDELAY is run always with '-D' and '-W' options by XMAKE.
XMAKE: Makefile saved in 'calc.mak'.
XMAKE: Making 'calc.bit' ...
XMAKE: Execute command 'xnfmerge -A -D xnf -D . -P 3020APC68-
7 xnf\calc.xnf calc.xff'.
XMAKE: Execute command 'xnfprep calc.xff calc.xtg
parttype=3020APC68-7'.
XMAKE: Execute command 'xblox calc.xtg calc.xg
parttype=3020APC68-7'.
XMAKE: Execute command 'xnfprep calc.xg calc.xtf
parttype=3020APC68-7'.
XMAKE: Execute command 'xnfmap -P 3020APC68-7 calc.xtf
calc.map'.
XMAKE: Execute command 'ppr calc.map parttype=3020APC68-7'.
XMAKE: Execute command 'xdelay -D -W calc.lca'.
XMAKE: Execute command 'makebits -R2 -S0 -XB -YA calc.lca'.
XMAKE: 'calc.bit' has been made. Check output in 'calc.out'.
```

An explanation of the XDM output follows.

```
XMAKE: Generating makefile 'calc.mak' ...
XMAKE: Profile used is the current XDM settings.
XMAKE: Execute command 'annotate calc.sch'.
XMAKE: Execute command 'inet calc.sch'.
XMAKE: Execute command 'sdt2xnf calc.inf calc.xnf -D xnf'.
```

XMake runs a series of commands that convert the OrCAD SCH files to Xilinx XNF files.

```
XMAKE: Set the part type to '3020APC68-7 from 'xnf\calc.xnf'.
XMAKE: Running with the following options: (none)
>>> XDELAY is run always with '-D' and '-W' options by XMAKE.
XMAKE: Makefile saved in 'calc.mak'.
```

XMake always creates a file with a .mak extension that contains a list of the commands used to process the design.

```
XMAKE: Making 'calc.bit' ...
XMAKE: Execute command 'xnfmerge -A -D xnf -D . -P 3020APC68-
7 xnf\calc.xnf calc.xff'.
```

XNFMerge flattens the hierarchical XNF files into a single netlist, which is written out as an XFF file.

```
XMAKE: Execute command 'xnfprep calc.xff calc.xtg
parttype=3020APC68-7'.
```

XNFPrep is run to verify that the flattened XFF file is correct. It creates a report that is stored in the file calc.prp. If you choose Stop to Review DRC as the target for XMake, it stops at this point so that you can review this report. The output of XNFPrep is specified as calc.xtg.

```
XMAKE: Execute command 'xblox calc.xtg calc.xg
parttype=3020APC68-7'.
```

X-BLOX is run to synthesize the X-BLOX symbols in the design into standard logic.

```
XMAKE: Execute command 'xnfprep calc.xg calc.xtf
parttype=3020APC68-7'.
```

XNFPrep must once again be run to verify that the logic produced by X-BLOX is correct. In this case, the output file name is specified as calc.xtf.

```
XMAKE: Execute command 'xnfmap -P 3020APC68-7 calc.xtf
calc.map'.
```

XNFMap maps the logic found in the calc.xtf file into sections that will fit within XC3000A CLBs. For XC4000 designs, this step is handled by PPR.

```
XMAKE: Execute command 'ppr calc.map parttype=3020APC68-7'.
```

XMake runs PPR to place the mapped logic and route the interconnections. The output is a Logic Cell Array (LCA) file, which is a description of the design as it will actually be configured on the chip. For XC3000 designs, this step is performed by APR.

```
XMAKE: Execute command 'xdelay -D -W calc.lca'
```

XDelay writes delay information into the LCA file.

```
XMAKE: Execute command 'makebits -R2 -S0 -XB -YA calc.lca'
```

Since Make Bitstream was the chosen target, XMake runs MakeBits in order to create a bitstream that can be downloaded to the part.

```
XMAKE: 'calc.bit' has been made. Check output in 'calc.out'
```

The entire output of XMake is stored in the calc.out file.

Verifying CALC on the Demonstration Board

At this point, a BIT file has been created that can be downloaded to the appropriate demonstration board to verify the validity of the design. If you are unfamiliar with this process, please refer to the "Verifying the Design Using a Demonstration Board" section of the "SDT Tutorial" chapter for more information.

Timing Simulation

You have already performed many of the steps necessary for timing simulation. The calc.lca file created in the previous design implementation section contains the timing information for the design. All that is necessary is to back-annotate the timing and netlist information from the LCA file to VST. This task is accomplished with the aid of XSimMake.

Creating the Simulation Netlist

Run XSimMake to generate a netlist that can be used for timing simulation.

1. Select **`Verify → XSIMMAKE`**.
2. Select the **`-F`** option.

   A menu of possible flows appears.
3. Choose the **`Orcad_Fpga_Timing`** option and select **`Done`**.
4. Choose **`CALC.LCA`** from the list of LCA files shown.
5. After you review the XSimMake output, press any key to return to XDM.

As XSimMake runs, text similar to the following appears in the XDM window.

Note: XSimMake flows vary depending on the design. The flow used by XSimMake for your design may be slightly different from the flow shown in this tutorial.

```
XSIMMAKE COMMAND : xdelay -w -d calc.lca
XSIMMAKE COMMAND : lca2xnf -g calc.lca calc.xnf
XSIMMAKE COMMAND : xnfba calc.xg calc.xnf
XSIMMAKE COMMAND : xnf2inf xnfba.xnf calc
XSIMMAKE COMMAND : deleted file xnfba.xnf
0 Errors and 0 Warnings occurred during processing.
```

Examining XSimMake Output

An explanation of the XSimMake output for the timing flow follows.

```
XSIMMAKE COMMAND : xdelay -w -d calc.lca
```

XSimMake runs XDelay on the design, although it may already have been run by XMake, to ensure that timing information is included in the LCA file.

```
XSIMMAKE COMMAND : lca2xnf -g calc.lca calc.xnf
```

LCA2XNF converts the LCA file, which contains the delay information, back to an XNF file.

```
XSIMMAKE COMMAND : xnfba calc.xg calc.xnf
```

When logic is optimized by the place and route tools, although functionally equivalent, it may not exactly reflect the logic as seen on the schematic. XNFBA reads the XNF file produced by LCA2XNF and the XG file produced by X-BLOX, and attempts to rewrite the netlist so that it looks like the logic described on the schematic. However, it still reflects the timing information found in the back-annotated netlist.

```
XSIMMAKE COMMAND : xnf2inf xnfba.xnf calc
XSIMMAKE COMMAND : deleted file xnfba.xnf
```

The above steps generate simulation input files from the back-annotated netlist, then delete the intermediate XNF file created by XNFBA.

Configuring OrCAD VST for Timing Simulation

Verify that the simulator is correctly configured to simulate the input file, calc.vst. Then configure the simulator to use timing delays from the DBA file.

1. Select **`Design Entry`** → **`ORCAD`** to enter the OrCAD design environment.
2. Select **`Digital Simulation Tools`** → **`Execute`** → **`Simulate`** → **`Local Configuration`** → **`Configure SIMULATE`**.

   The Configure Simulate screen appears.

Note: OrCAD's configuration programs are memory intensive. If you receive the message `Could not find the .EXE, or not enough memory to load \orcadexe\VST_CLC.EXE`, return to the XDM executive screen. Close XDM by typing **`exit`** ↵, type **`orcad`** ↵, and modify the configuration as described in this section. After completing the commands in this section, type **`xdm`** ↵ to return to XDM, and continue with the "Performing a Timing Simulation" section, following.

3. Under File Options, verify that the file names read as follows:

```
Connectivity database   CALC.VST
Stimulus file           CALCSIM.STM
Trace file              CALCSIM.TRC
```

4. If the names are incorrect, make the necessary changes.
5. Set the simulator to use timing delays from the DBA file by selecting **`Use Delay Annotation.`**
6. Return to the ESP screen by selecting **`OK`**.

Performing a Timing Simulation

You are ready to perform the timing simulation.

1. Enter the simulator by selecting **`Simulate`** → **`Execute`**.
2. Select **`Set`** → **`Spool to Disk`**.

   The `Spool Trace Data to Disk?` prompt appears.
3. Select **`Yes`** → **`Yes`** to spool the data to the spool.tdf file.

The default time scale is 10 ns per screen. Adjust that to 1,000 ns per screen, which means increasing the scale by a factor of 100.

4. Select **`Trace → Change View`**.

   The `Trace Delta Time?` prompt appears at the top of the screen.

5. Type **`100`** ↵.

6. Select **`Run Simulation`**.

   The `Simulation Length?` prompt appears.

7. Type **`12000`** ↵ to simulate for 1,200 ns.

The output of this simulation run is nearly identical to the output of the timing simulation run on the original non-X-BLOX Calc design. The timing differs slightly, because different place and route runs produce different net delays.

Command Summaries

Implementation and simulation of designs containing X-BLOX modules is significantly more complex than that of standard schematics. It is strongly recommended that you use XMake and XSimMake when processing your X-BLOX designs rather than using a manual flow or creating your own batch files. For this reason, X-BLOX command summaries are not given.

Further Reading

Before beginning an X-BLOX design, you should read the descriptions of the X-BLOX macros found in the *XACT Libraries Guide* in order to understand the abilities and limitations of each macro. You should also review the section on the X-BLOX program found in the *X-BLOX User Guide*.

OrCAD Interface/ Tutorial Guide

Xilinx ABEL Tutorial

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Chapter 14

Xilinx ABEL Tutorial

The Xilinx ABEL software package enables you to define logic in terms of text-based Boolean equations, truth tables, and state machine descriptions using the ABEL Hardware Description Language (HDL). These logic blocks can then be included as part of a larger design, allowing logic defined by both graphical and text-based entry to exist within the same design.

This chapter gives a practical example of using Xilinx ABEL within the OrCAD design environment. It is not intended to fully explain all of the functionality found within Xilinx ABEL. Please refer to the "Further Reading" section at the end of this tutorial for a list of sources from which to obtain more information.

Before Beginning the Tutorial

This section of the tutorial assumes that you are already familiar with the material in the “SDT Tutorial” and “VST Tutorial” chapters of this manual. If not, please review those chapters before continuing.

Required Software

This tutorial assumes that you are using the following versions of the development software:

- OrCAD / SDT — SDT 386+
- OrCAD / VST — VST 386+
- OrCAD / Xilinx Interface — Version 5.00 or later
- XACT Design Manager (XDM) — Version 5.00 or later
- Xilinx ABEL, the Xilinx text-based entry tool that uses Data I / O's ABEL HDL language to enter logic descriptions

If you have Xilinx software on CD-ROM, you should have at least temporary access to all of the above software using the temporary licensing available on the programmable key, provided that the temporary licensing has not already been exhausted.

Preparing the Design

If you chose to read through the SDT tutorial rather than actually perform the steps involved, you must verify that your PC is set up correctly to use OrCAD 386+, the XACT Development System software, and X-BLOX. Then create and configure the design directory, and copy a completed set of schematics from any of the solutions directories supplied.

If you have already performed the SDT tutorial on your PC, skip to the next section, "Viewing Stat_abl.abl."

1. Follow the instructions given in the "Before Beginning the Tutorial" section of the "SDT Tutorial" chapter for setting up your design environment.
2. The tutorial files are optionally installed when you install the Xilinx/OrCAD interface software. If you have already installed the software but are not sure whether you specified tutorial installation, check for the c:\xact\tutorial\orcad\calc directory. This directory contains the tutorial files.
3. Create a new project called "Calc," as described in the "Creating the Project Directory" section of the "SDT Tutorial" chapter.
4. Full solutions for the SDT tutorial are supplied in the solutions directories located in \xact\tutorial\orcad\calc. One of the following must be copied to the directory where you will be performing the tutorial.

   ...\soln_3k — Solution files for XC3020PC68
   ...\soln_3ka — Solution files for XC3020APC68
   ...\soln_4ka — Solution files for XC4003APC84
   ...\soln_4k — Solution files for XC4003PC84

   For example, if you are targeting the tutorial for an XC3020APC68 device, perform the following sequence of commands on a PC.

   ```
   cd \orcad\calc↵
   copy \xact\tutorial\orcad\calc\solution\*.*↵
   ```

where *solution* is either soln_3ka, soln_3k, soln_4ka, or soln_4k. The solutions schematics are targeted towards the 3020APC68-7, 3020PC68-70, 3020PC68-4003APC84-6, and 4003PC84-6 devices, respectively.

Note: This tutorial assumes that your ORCADPROJ variable is set to c:\orcad\. You need not follow this convention.

This procedure gives you a full set of completed schematics for the Calc design, with all supporting files.

5. Configure the design directory by typing **`xdraft 3`**↵ or **`xdraft 4`**↵, depending on whether you are targeting an XC3000A or an XC4000 device.

Note: All of the screen outputs refer to the processing of the 3000A solutions design. Other devices have slightly different outputs.

Viewing Stat_abl.abl

A Xilinx ABEL-based block called STAT_ABL is created in this section to replace the STATMACH state machine that resides within the CONTROL block on the CALC schematic. The Xilinx ABEL code for STAT_ABL is functionally identical to the schematic for STAT-MACH, so this substitution in no way alters the function of the CALC design.

Note: If you are not already familiar with the CALC design, read the discussion found in the "Design Description" section of the "SDT Tutorial" chapter.

Stat_abl.abl is the name of the Xilinx ABEL HDL file from which a logic description for the STAT_ABL block is generated.

Enter the Xilinx ABEL environment and view the stat_abl.abl source code.

1. In XDM select **`Design Entry`** → **`XABEL`** → **`STAT_ABL.ABL`**.
2. The file displayed in Figure 14-1 appears in the text window of Xilinx ABEL.

```
module stat_abl

title 'State machine for Calc design'
"This state machine has 3 states which control the functions
"of the ALU and the stack. The states are as follows:
"      SPUSH  -- increment stack pointer
"      SWE    -- write value into stack
"      SOTHER -- do neither (initial state)

"This is a one-hot state machine, which means that only
"one of the states is active at any given time.  This method
"is particularly suited for use with Xilinx ABEL and Xilinx
"FPGAs, which are rich in flip-flop resources.
"This file also generates control signals from equations.
"For an equivalent schematic, see statmach.1.

declarations
"inputs
          OP5, OP4, OP3, OP2, OP1, OP0, EXC        pin;
"clock
          CLK                                      pin;
"outputs
          CTL3, CTL2, CTL1, CTL0                   pin;
          UP_DN, WE, RST, ADD_SUB, CE_ALU, CE_ADDR pin;

"state diagram declarations and assignments
          XABELSM        state_register istype 'reg_d';
          SPUSH, SWE, SOTHER                    state;

"vector definitions
          OP  = [OP5,OP4,OP3,OP2,OP1,OP0];
          HOP = [OP5,OP4,OP3];
          CTL = [CTL3,CTL2,CTL1,CTL0];

"declare internal nodes
          SEL_OP, OP_CTL2, OP_CTL1, OP_CTL0      node;

"node declarations for simulation only, can't use state names
"in simulation vectors
          PUSH, OTHER                            node;

"define clock & don't-care values for test vectors
          C, X = .C., .X.;

Xilinx property 'initialstate SOTHER';
```

```
equations
    XABELSM.CLK     = CLK;
    RST             = (HOP == ^b101) & EXC;
    ADD_SUB         = !OP_CTL2;
    SEL_OP          = (HOP == ^b111);
    CE_ALU          = !(SEL_OP & OP2 & OP0) & SOTHER & EXC;
    CE_ADDR         = !(OP2 & OP1 & OP0) & SEL_OP & EXC;
    OP_CTL2         = (OP5 & !SEL_OP) # (OP2 & SEL_OP);
    OP_CTL1         = (OP4 & !SEL_OP) # (OP1 & SEL_OP);
    OP_CTL0         = (OP3 & !SEL_OP) # (OP0 & SEL_OP);
    CTL3            = SEL_OP;
    CTL2            = OP_CTL2 & OP_CTL1;
    CTL1            = OP_CTL1 & !OP_CTL2;
    CTL0            = !OP_CTL2 & OP_CTL0;
    UP_DN           = OP2 & !OP1 & OP0 & SEL_OP & EXC;
    PUSH            = SPUSH;
    WE              = SWE;
    OTHER           = SOTHER;

"always optimize out don't-cares
@DCSET

state_diagram XABELSM
        state SPUSH:  goto                              SWE;
        state SWE:    goto                            SOTHER;
        state SOTHER: if                  (UP_DN) then SPUSH
                      else                             SOTHER;

test_vectors
"begin in initial state, each line is one clock cycle

([CLK EXC  OP ]->[ PUSH WE OTHER  ADD_SUB  RST  CE_ALU  CE_ADD RCTL])

"quick check to test the state machine
[ C, 0,   X ]->[  0,  0,  1,    X,     X,    X,     X,    X  ];
[ C, 1, ^h3F]->[  0,  0,  1,    X,     X,    X,     X,    X  ];
[ C, 0,   X ]->[  0,  0,  1,    X,     X,    X,     X,    X  ];
[ C, 1, ^h3D]->[  1,  0,  0,    X,     X,    X,     X,    X  ];
[ C, 0,   X ]->[  0,  1,  0,    X,     X,    X,     X,    X  ];
[ C, 0,   X ]->[  0,  0,  1,    X,     X,    X,     X,    X  ];
[ C, 1, ^h38]->[  0,  0,  1,    X,     X,    X,     X,    X  ];

"test the control logic, EXC low
[ C, 0, ^h0 ]->[  0,  0,  1,    1,     0,    0,     0,   ^h0 ];
[ C, 0, ^h8 ]->[  0,  0,  1,    1,     0,    0,     0,   ^h1 ];
[ C, 0, ^h10]->[  0,  0,  1,    1,     0,    0,     0,   ^h2 ];
[ C, 0, ^h18]->[  0,  0,  1,    1,     0,    0,     0,   ^h3 ];
[ C, 0, ^h20]->[  0,  0,  1,    0,     0,    0,     0,   ^h0 ];
[ C, 0, ^h28]->[  0,  0,  1,    0,     0,    0,     0,   ^h0 ];
[ C, 0, ^h30]->[  0,  0,  1,    0,     0,    0,     0,   ^h4 ];
```

```
"extended instruction set
[ C, 0, ^h38]->[  0,  0,  1,    1,     0,    0,     0,   ^h8 ];
[ C, 0, ^h39]->[  0,  0,  1,    1,     0,    0,     0,   ^h9 ];
[ C, 0, ^h3A]->[  0,  0,  1,    1,     0,    0,     0,   ^hA ];
[ C, 0, ^h3B]->[  0,  0,  1,    1,     0,    0,     0,   ^hB ];
[ C, 0, ^h3C]->[  0,  0,  1,    0,     0,    0,     0,   ^h8 ];
[ C, 0, ^h3D]->[  0,  0,  1,    0,     0,    0,     0,   ^h8 ];
[ C, 0, ^h3E]->[  0,  0,  1,    0,     0,    0,     0,   ^hC ];
[ C, 0, ^h3F]->[  0,  0,  1,    0,     0,    0,     0,   ^hC ];

"test the control logic, EXC high
[ C, 1, ^h0 ]->[  0,  0,  1,    1,     0,    1,     0,   ^h0 ];
[ C, 1, ^h8 ]->[  0,  0,  1,    1,     0,    1,     0,   ^h1 ];
[ C, 1, ^h10]->[  0,  0,  1,    1,     0,    1,     0,   ^h2 ];
[ C, 1, ^h18]->[  0,  0,  1,    1,     0,    1,     0,   ^h3 ];
[ C, 1, ^h20]->[  0,  0,  1,    0,     0,    1,     0,   ^h0 ];
[ C, 1, ^h28]->[  0,  0,  1,    0,     1,    1,     0,   ^h0 ];
[ C, 1, ^h30]->[  0,  0,  1,    0,     0,    1,     0,   ^h4 ];

"extended instruction set
[ C, 1, ^h38]->[  0,  0, 1,    1,      0,    1,     1,   ^h8 ];
[ C, 1, ^h39]->[  0,  0, 1,    1,      0,    1,     1,   ^h9 ];
[ C, 1, ^h3A]->[  0,  0, 1,    1,      0,    1,     1,   ^hA ];
[ C, 1, ^h3B]->[  0,  0, 1,    1,      0,    1,     1,   ^hB ];
[ C, 1, ^h3C]->[  0,  0, 1,    0,      0,    1,     1,   ^h8 ];
[ C, 1, ^h3D]->[  1,  0, 0,    0,      0,    0,     1,   ^h8 ];

"insert two clocks to return to initial state
[ C, 0, ^h3D]->[  0,  1, 0,    0,      0,    0,     0,   ^h8 ];
[ C, 0, ^h3D]->[  0,  0, 1,    0,      0,    0,     0,   ^h8 ];
[ C, 1, ^h3E]->[  0,  0, 1,    0,      0,    1,     1,   ^hC ];
[ C, 1, ^h3F]->[  0,  0, 1,    0,      0,    0,     0,   ^hC ];

end stat_abl
```

Figure 14-1 Stat_abl.abl File

A breakdown of the contents of the Xilinx ABEL file follows.

```
module stat_abl
```

The Module statement specifies the beginning of the Xilinx ABEL module.

```
title 'State machine for Calc design'
```

The Title statement, while not necessary, is added as a header for the intermediate files created by Xilinx ABEL.

```
"This state machine has 3 states which control the functions
"of the ALU and the stack.  The states are as follows:
"      SPUSH  -- increment stack pointer
"      SWE    -- write value into stack
"      SOTHER -- do neither (initial state)

"This is a one-hot state machine, which means that only
"one of the states is active at any given time. This method
"is particularly suited for use with Xilinx ABEL and Xilinx
"FPGAs, which are rich in flip-flop resources.
"This file also generates control signals from equations.
"For an equivalent schematic, see statmach.1.
```

Any text preceded by double quotation marks, as in the example just given, is interpreted as comment text.

```
declarations
"inputs
          OP5, OP4, OP3, OP2, OP1, OP0, EXC          pin;
"clock
          CLK                                        pin;
"outputs
          CTL3, CTL2, CTL1, CTL0                     pin;
          UP_DN, WE, RST, ADD_SUB, CE_ALU, CE_ADDR   pin;
```

The Pin statements in the declaration define the pinout of the Xilinx ABEL module. Pins must be either inputs or outputs; bidirectional pins are not allowed.

```
"state diagram declarations and assignments
          XABELSM         state_register istype  'reg_d';
          SPUSH, SWE, SOTHER                      state;
```

The State_register keyword declares a symbolic state machine. The State keyword declares states that appear in a symbolic state machine. Istype ‘reg_d’ declares that the state machine will be implemented using D flip-flops. State_register must be used in conjunction with State.

```
"vector definitions
          OP  = [OP5,OP4,OP3,OP2,OP1,OP0];
          HOP = [OP5,OP4,OP3];
          CTL = [CTL3,CTL2,CTL1,CTL0];
```

Vector definitions define bus vectors within Xilinx ABEL; these vectors can be used during simulation in the Xilinx ABEL environment.

```
"declare internal nodes
          SEL_OP, OP_CTL2, OP_CTL1, OP_CTL0       node;
```

These nodes are declared for use as variables in intermediate equations.

```
"node declarations for simulation only, can't use state names
"in simulation vectors
          PUSH, OTHER                                node;
```

The Xilinx ABEL simulator does not allow the use of symbolic state names — that is, state names used in the definition of a state machine — in test vectors, so these two "dummy" nodes were created. They mirror SPUSH and SOTHER for use in the simulation test vectors found at the end of the file.

```
"define clock & don't-care values for test vectors
          C, X = .C., .X.;
```

This definition allows the default clock and don't-care syntax (.C. and .X.) to be replaced by a simpler one without periods (C and X) so that the simulation vectors are easier to read.

```
Xilinx property 'initialstate SOTHER';
```

The Xilinx Property statement defines the initial power-up state of the state machine as the SOTHER state. This state and others are defined in a later section of the file.

```
equations

          XABELSM.CLK= CLK;
          RST        = (HOP == ^b101) & EXC;
          ADD_SUB    = !OP_CTL2;
          SEL_OP     = (HOP == ^b111);
          CE_ALU     = !(SEL_OP & OP2 & OP0) & SOTHER & EXC;
          CE_ADDR    = !(OP2 & OP1 & OP0) & SEL_OP & EXC;
          OP_CTL2    = (OP5 & !SEL_OP) # (OP2 & SEL_OP);
          OP_CTL1    = (OP4 & !SEL_OP) # (OP1 & SEL_OP);
          OP_CTL0    = (OP3 & !SEL_OP) # (OP0 & SEL_OP);
          CTL3       = SEL_OP;
          CTL2       = OP_CTL2 & OP_CTL1;
          CTL1       = OP_CTL1 & !OP_CTL2;
          CTL0       = !OP_CTL2 & OP_CTL0;
          UP_DN      = OP2 & !OP1 & OP0 & SEL_OP & EXC
          PUSH       = SPUSH;
          WE         = SWE;
          OTHER      = SOTHER;
```

The Equations statement defines the internal logic of the module. Each equation is synthesized into combinatorial logic.

```
"always optimize out don't-cares
@DCSET
```

The @DCSET statement instructs Xilinx ABEL to optimize don't-cares in the same way that Karnaugh maps are used to minimize a logic function.

```
state_diagram XABELSM
         state SPUSH:  goto                              SWE;
         state SWE:    goto                            SOTHER;
         state SOTHER: if                    (UP_DN) then SPUSH
                       else                               SOTHER;
```

The State_diagram statement defines under what circumstances state transitions occur. In this case, the SPUSH state is always followed by SWE, SWE is always followed by SOTHER, and SOTHER is followed by SPUSH if the UP_DN signal is High. Otherwise, the state machine remains in the SOTHER state.

```
test_vectors
```

Test_vectors specifies the beginning of a section containing test vectors. The test vectors define sets of inputs and expected outputs.

```
"begin in initial state, each line is one clock cycle

([CLK EXC  OP ]->[PUSH  WE OTHER  ADD_SUB  RST  CE_ALU  CE_ADD RCTL])
```

This line defines the set of inputs as the vectors CLK, EXC, and OP. Output names are then specified, for which expected values are specified in the following lines.

```
"quick check to test the state machine
[ C, 0,   X ]->[  0,  0,  1,    X,     X,    X,     X,    X  ];
[ C, 1, ^h3F]->[  0,  0,  1,    X,     X,    X,     X,    X  ];
[ C, 0,   X ]->[  0,  0,  1,    X,     X,    X,     X,    X  ];
[ C, 1, ^h3D]->[  1,  0,  0,    X,     X,    X,     X,    X  ];
[ C, 0,   X ]->[  0,  1,  0,    X,     X,    X,     X,    X  ];
                              .
                              .
                              .
"insert two clocks to return to initial state
[ C, 0, ^h3D]->[  0,  1, 0,    0,      0,    0,     0,   ^h8 ];
[ C, 0, ^h3D]->[  0,  0, 1,    0,      0,    0,     0,   ^h8 ];
[ C, 1, ^h3E]->[  0,  0, 1,    0,      0,    1,     1,   ^hC ];
[ C, 1, ^h3F]->[  0,  0, 1,    0,     0,    0,     0,   ^hC ];
```

Simulation begins with the state machine in the initial power-up state. Each successive line steps forward by one clock cycle. The input values to the left of the arrow are applied to the current state; the

resulting outputs are displayed to the right of the arrow. The "^h" before some values tells the simulator that the vectors are specified in hexadecimal.

```
end stat_abl
```

The End statement specifies the end of the Xilinx ABEL module.

Simulating Within Xilinx ABEL

The Xilinx ABEL simulator is now used to verify the STAT_ABL design, using the test vectors described above as input.

1. Hold down the **Alt** key and type **c** to select the Compile menu.
2. Select **Simulate Equations**.

Note: Xilinx ABEL running from XDM may run out of memory on some designs. If your system runs out of memory at any time during this tutorial, exit Xilinx ABEL by selecting File → Exit, quit XDM, and reboot your machine to free the memory. Restart Xilinx ABEL directly from the DOS prompt by typing xabel ↵. Return to step 1 and continue with the tutorial.

Xilinx ABEL prepares the test vectors for simulation, then simulates them. It reports that 39 of 39 test vectors simulated correctly. This result means that as each of the test vector inputs was executed, the output of the state machine corresponded exactly to the expected values entered in the test vectors.

Note: If errors occur, you may have inadvertently modified the Xilinx ABEL source code. Recopy stat_abl.abl from the appropriate solutions directory and try again.

Compiling STAT_ABL.ABL

The Xilinx ABEL file could be compiled within Xilinx ABEL using the Compile → Xilinx FPGA Netlist command. Instead, perform this step from within XDM, using a program called ABL2XNF. It is necessary at this point to compile the netlist in order to ensure its validity. ABL2XNF also generates an XSF file that is used to create an OrCAD symbol for the Xilinx ABEL design, and an XAS simulation netlist file.

1. Select **File → Exit** and return to XDM.
2. Within XDM, make sure the Family, Part, and Directory options are set correctly, then choose **Translate → ABL2XNF → STAT_ABL.ABL.**
3. Choose **Done** to select the default options.

   ABL2XNF compiles the ABL file into an XNF netlist.

Note: If errors occur, be sure your path and XACT environment variable are set correctly. If errors persist, re-copy the stat_abl.abl file from the installation area.

Including STAT_ABL in the CALC Design

You are ready to create a symbol for the Xilinx ABEL block and place it in your schematic.

Creating a Symbol for STAT_ABL

You must create a special symbol so that you can include the Xilinx ABEL module on the CONTROL schematic.

Creating a Command File with SymGen

The SymGen program automates the creation of symbols for Xilinx ABEL modules. It uses as input an XSF file created by ABL2XNF. The XSF file contains the pinout for the symbol. SymGen uses this file to generate a command file for the OrCAD library editor.

1. Within XDM, select **Design Entry → SYMGEN**.
2. Select the **-o** option to generate an OrCAD symbol, then select **Done**.
3. Choose **STAT_ABL.XSF** from the menu of available input files. It should be the only file on the menu.

   SymGen creates a command file called stat_abl.cmd and places it in the design directory.

Creating the Library Symbol

The next step is to create the library symbol for the STAT_ABL block by invoking the command file from inside the library editor.

1. Enter the OrCAD environment by selecting **DesignEntry → ORCAD** from the XDM menus.
2. Select **Schematic Design Tools → Execute → Edit Library → Execute**.

   The `Read Library?` prompt appears at the top of the screen.
3. Type **.\stat_abl.lib**↵.

   The ".\" at the head of the library name ensures that the new library is placed in the design directory. The Xilinx / OrCAD interface software looks for user-created libraries in the current design directory.

Note: You can invoke the OrCAD library editor from the DOS prompt, by typing libedit .\stat_abl.lib↵.

4. Select **Import** and type **stat_abl.cmd**↵ to invoke the command file.

   The symbol editor creates a symbol, as shown in Figure 14-2. The new symbol has now been drawn but not saved into the library.
5. Select **Library → Update Current** to save the symbol to memory.
6. Select **Quit → Update File** to save the library to disk.

   The stat_abl.lib file is written to the current design directory.
7. Select **Abandon Edits** to exit the library editor.

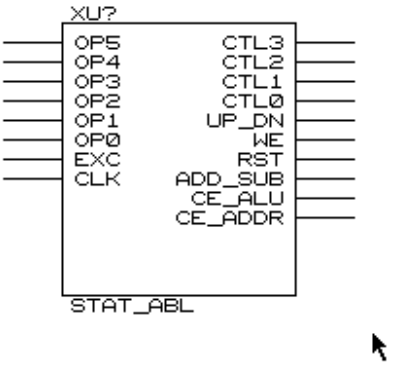


Figure 14-2 STAT_ABL Symbol

Adding the Library to Your Search Path

You must add the new library to your search path.

1. Select **`Draft → Configure Schematic Tools`**.
2. Pan down to find the list of Available Libraries.
3. Click on **`.\STAT_ABL.LIB → Insert`**.

   The new library appears in the Configured Libraries list at right.
4. Pan back to the top and click on **`OK`**.

Note: Alternatively, you can add the new library to your search path by using any text editor to edit the sdt.cfg file in your design directory. Add the line: LIB = '.\STAT_ABL.LIB' below the other library definitions.

Adding STAT_ABL to the CONTROL Schematic

Now that you have generated the symbol and the XNF file for STAT_ABL, substitute the symbol for the schematic-based STATMACH state machine.

1. Open the CALC schematic in an SDT window.
2. Place the cursor on the CONTROL instance on the CALC schematic.
3. Select **`Quit → Enter Sheet → Enter → Escape`**.

   The schematic for CONTROL is displayed.
4. Place the cursor on the upper left-hand corner of the STATMACH block.
5. *Without moving the cursor*, use the keyboard keys to select **`Delete → Block → Begin → End`**.

   The STATMACH block disappears.

Note: If the nets connected to the STATMACH block disappear also, then the cursor must have moved during the delete process. Quit without saving your changes and reload the schematic.

Place the new STAT_ABL block on the schematic.

6. Select **`Get`** and type s**`tat_abl`**↵.
7. Move the cursor to the correct location and select **`Place`**.

   This procedure replaces the original STATMACH block with the functionally equivalent Xilinx ABEL module called STAT_ABL.

Adding Symbol Attributes

The STAT_ABL symbol is different from the other symbols in the CALC schematic, because its logic is described in an XNF file, generated earlier using Xilinx ABEL, instead of in a graphical representation using parts from the OrCAD libraries.

You must tell the Xilinx programs where to find the logic description for the STAT_ABL block by attaching the FILE attribute to the symbol placement. The FILE attribute specifies the name of the ABL file containing the logic description. Do not specify an extension when including the file name on the symbol.

Additionally, XSimMake looks for an attribute that defines the symbol as representing a Xilinx ABEL netlist, since the FILE attribute can be used to designate an XNF file from any source. This attribute is DEF=XABEL.

Add the FILE and DEF attributes to the STAT_ABL symbol placement.

1. Position the cursor on the STAT_ABL symbol.
2. Select **Edit → Edit → LOC,OPTIONS → Name** for an XC3000 family design, or **Edit → Edit → OPTIONS1 → Name** for an XC4000 family design.
3. Type **FILE=STAT_ABL,DEF=XABEL**↵.
4. Select **Quit → Update File → Abandon Edits → Yes** to save your schematic and leave the SDT schematic editor.

The CONTROL schematic with the STAT_ABL module is shown in Figure 14-3.

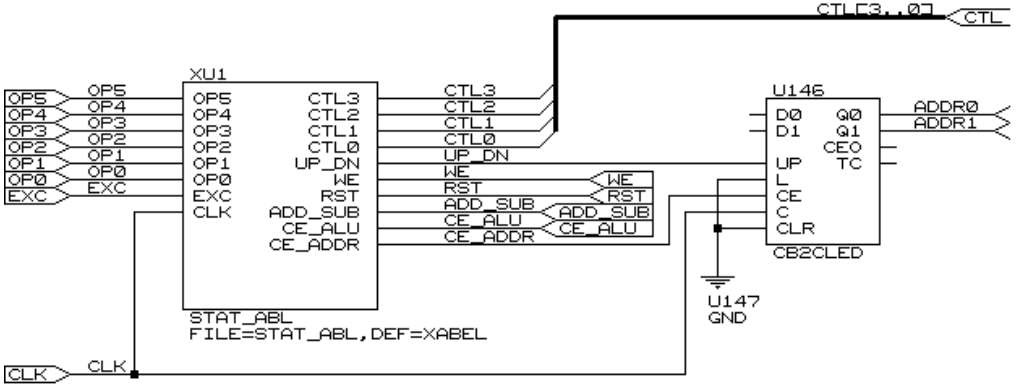


Figure 14-3 STAT_ABL Symbol in CONTROL Schematic

Functional Simulation

The XSimMake program allows you to easily simulate designs containing Xilinx ABEL components. It coordinates the program execution flow necessary for functional or timing simulation. XSimMake is similar to XMake, except that it produces a simulation netlist instead of a bitstream.

Note: For more detailed information on XSimMake, refer to the "Creating a Functional Simulation Netlist with XSimMake" section of the "VST Tutorial" chapter.

Creating the Functional Simulation Netlist

Run XSimMake to generate a netlist that you can functionally simulate.

1. Return to XDM.
2. Select **Verify → XSIMMAKE**.
3. Select the **-F** option.

   A menu of possible flows appears.
4. Select the **Orcad_Fpga_Func** flow option.
5. Select **Done**.
6. Choose **CALC.SCH** from the list of schematic files shown.

   XSimMake now executes.
7. After you review the XSimMake output, press any key to return to XDM.

As XSimMake runs, text similar to the following appears in the XDM window.

Note: XSimMake flows vary depending on the design. The flow used by XSimMake for your design may be slightly different from the flow shown in this tutorial.

```
XSIMMAKE COMMAND : creating directory savexnf
XSIMMAKE COMMAND : creating directory otherxnf
XSIMMAKE COMMAND : annotate calc.sch
XSIMMAKE COMMAND : inet calc.sch /t
XSIMMAKE COMMAND : inf2xnf calc d= otherxnf
XSIMMAKE COMMAND : xnfmerge -y -d otherxnf otherxnf\calc.xnf
```

```
otherxnf\calc.xff
XSIMMAKE COMMAND : xfind otherxnf\calc.xff calc.xfw calc.xgs
READING XFW FILE : calc.xfw
XSIMMAKE COMMAND : copied file xnf\stat_abl.xas to
savexnf\stat_abl.xnf
XSIMMAKE COMMAND : copied file savexnf\stat_abl.xnf to
otherxnf\stat_abl.xnf
XSIMMAKE COMMAND : xnfmerge -z -d otherxnf -d xnf -d .
otherxnf\calc.xnf otherxnf\calc.xff
XSIMMAKE COMMAND : xnf2inf otherxnf\calc.xff calc.vst u=true
0 Errors and 0 Warnings occured during processing.
```

Examining XSimMake Output

An explanation of the XSimMake functional flow output seen in XDM follows. While it is not necessary to know anything about the inner workings of XSimMake, it sometimes gives useful perspective to have some idea of how the preparation of the design is accomplished. However, none of the information in this section is necessary for everyday usage of XSimMake.

```
XSIMMAKE COMMAND : creating directory savexnf
XSIMMAKE COMMAND : creating directory otherxnf
```

First, XSimMake creates new directories for storage of intermediate files.

```
XSIMMAKE COMMAND : annotate calc.sch
XSIMMAKE COMMAND : inet calc.sch /t
```

XSimMake then runs the OrCAD conversion programs to create an updated OrCAD netlist.

```
XSIMMAKE COMMAND : inf2xnf calc d= otherxnf
```

Next, XSimMake runs INF2XNF to convert the OrCAD INF files to standard Xilinx Netlist Format (XNF) files. INF2XNF is the engine of the SDT2XNF program used by XMake and in the manual flow.

```
XSIMMAKE COMMAND : xnfmerge -y -d otherxnf otherxnf\calc.xnf
otherxnf\calc.xff
```

XMFMerge combines all of the schematic XNF files into a single XNF format file with an .xff extension.

```
XSIMMAKE COMMAND : xfind otherxnf\calc.xff calc.xfw calc.xgs
```

XFind reads the XNF file to determine what types of symbols the netlist contains. In this case, it discovers the Xilinx ABEL symbol

STAT_ABL in the netlist and modifies program execution accordingly by generating a file called calc.xfw.

```
READING XFW FILE : calc.xfw
```

The calc.xfw file created in the previous step is now read by XSimMake. In this case, the file changes the program flow so that the simulation netlist for the Xilinx ABEL file is used.

```
XSIMMAKE COMMAND : copied file xnf\stat_abl.xas to
savexnf\stat_abl.xnf
XSIMMAKE COMMAND : copied file savexnf\stat_abl.xnf to
otherxnf\stat_abl.xnf
```

XSimMake copies the simulation netlist XAS file created by ABL2XNF into the simulation XNF directory, otherxnf. This step ensures that the simulation netlist for STAT_ABL is used when creating the simulation netlist for CALC. If you have not run ABL2XNF already, XSimMake runs it for you.

```
XSIMMAKE COMMAND : xnfmerge -z -d otherxnf -d xnf -d .
otherxnf\calc.xnf otherxnf\calc.xff
```

Next, XNFMerge is run to merge the Xilinx ABEL simulation XNF file into the design.

```
XSIMMAKE COMMAND : xnf2inf otherxnf\calc.xff calc.vst u=true
```

XSimMake then runs XNF2INF to generate a simulation file from the merged XNF format (XFF) file for use in the VST simulator. XNF2INF is the engine of the XNF2VST program used in the manual flow.

Stimulus and Trace Files

You now have a simulatable VST netlist. In the “VST Tutorial” chapter, you learned how to add stimulus and trace information to your schematic, and how to use the stimulus and trace editors within VST. However, since this tutorial assumes you already know how to create stimulus and trace data, Xilinx has supplied both ASCII and binary stimulus and trace files that you can use to simulate your design. The files are located in the solutions directories discussed in the “Preparing the Design” section of this tutorial. The binary files are called calcsim.stm and calcsim.trc, respectively.

1. If you copied one of the solutions directories in order to perform this tutorial, the stimulus and trace files are already present in your working directory.
2. If the files do not exist in your working directory, copy them by typing the following commands at the DOS prompt, from your working directory:

   copy \xact\tutorial\orcad\calc*solution.stm** ↵
   copy \xact\tutorial\orcad\calc*solution.trc** ↵

   where *solution* is either soln_3ka, soln_3k, soln_4ka, or soln_4k.

Configuring OrCAD VST for the Calc Design

You must specify the simulation input files before entering VST.

Performing these steps before each simulation eliminates the most common error made by Xilinx users simulating with OrCAD VST: simulating from the INF file rather than the VST file.

1. Select **DesignEntry → ORCAD** to enter the OrCAD design environment.
2. Select **Digital Simulation Tools → Execute → Simulate → Local Configuration → Configure SIMULATE**.

   The Configure Simulate screen appears.

Note: OrCAD's configuration programs are memory intensive. If you receive the message `Could not find the .EXE, or not enough memory to load \orcadexe\VST_CLC.EXE`, return to the XDM executive screen. Close XDM by typing **exit** ↵, type **orcad** ↵, and verify the configuration as described in this section. After completing the commands in this section, type **xdm** ↵ to return to XDM, and continue with the "Performing a Functional Simulation" section, following.

3. Look in the File Options portion of the screen. The file names appear similar to the following:

```
Connectivity database  CALC.INF
Stimulus file          CALC.STM
Trace file             CALC.TRC
```

4. Change the file names to the following:

```
Connectivity database   CALC.VST
Stimulus file           CALCSIM.STM
Trace file              CALCSIM.TRC
```

5. Click on **OK**.

Setting these values before entering the simulator is the easiest way to select the files. You must do this each time you start a new project or want to change the name of one of your input files.

Performing a Functional Simulation

You are ready to enter the OrCAD simulator, VST.

1. Select **Simulate → Execute**.

   The VST waveform display appears on the screen. At the left are listed the nodes from the trace file. No waveforms are visible, since you have not yet run a simulation.

   The waveform data may not all fit in RAM, so you can spool the data to disk as it is created.

2. Select **Set → Spool to Disk**.

   The `Spool Trace Data to Disk?` prompt appears.

3. Select **Yes → Yes** to spool the data to the spool.tdf file.

   The default time scale is 10.0 ns per screen. Adjust the time scale to 1,000.0 ns per screen, which means increasing the scale by a factor of 100. (About 1,200 ns will actually be visible on the screen.)

4. Select **Trace → Change View**.

   The `Trace Delta Time?` prompt appears at the top of the screen.

5. Type **100** ↵.

   You are ready to simulate the design using your stimulus and trace data.

6. Select **Run Simulation**.

   The `Simulation Length?` prompt appears.

7. Type **12000** ↵ to simulate for 1,200 ns.

The resulting waveforms are displayed as in Figure 14-4. Whenever EXC_P goes high, the next rising CLK edge executes the opcode selected by the SW switches (not displayed). First, the value 1010 is loaded into the ALU. Then the next command, a PUSH to the stack, writes 1010 into the stack. The PUSH command takes longer to complete because it requires extra states in the state machine.

The output of this simulation run is identical to the output of the functional simulation run on the original non-Xilinx ABEL Calc design, in the “VST Tutorial” chapter.

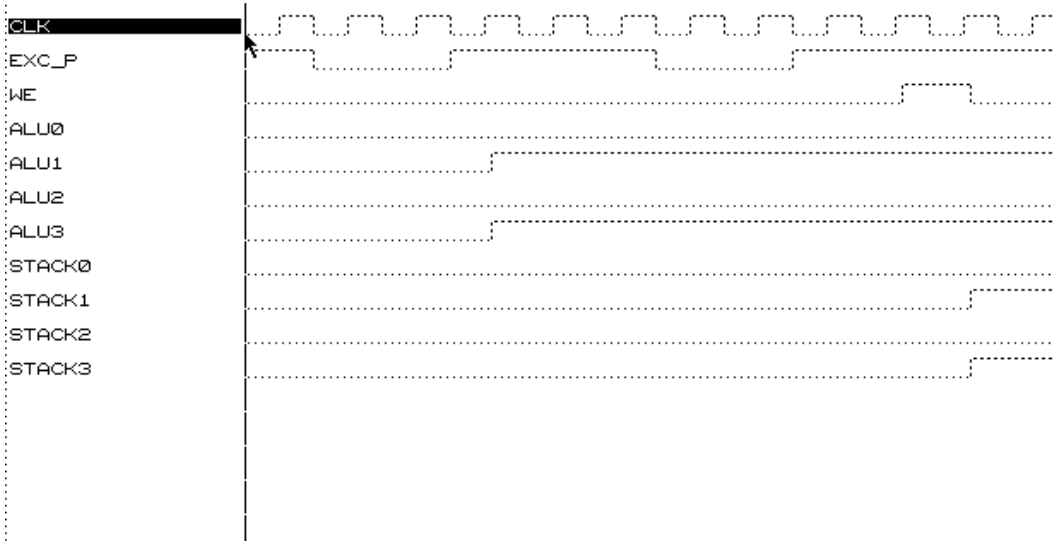


Figure 14-4 Simulation Output for Xilinx ABEL Design

Implementing the CALC Design

The translation of designs containing Xilinx ABEL blocks is similar to the translation of other designs. You can use XMake to generate files for timing simulation or a bitstream for programming actual devices, just as you use it for designs that do not contain Xilinx ABEL blocks. When the translation programs find the FILE attribute on the STAT_ABL symbol, the logic described in the stat_abl.xnf file, created earlier using Xilinx ABEL, is simply merged with the top-level XNF

file created from CALC before mapping, placing, and routing is begun.

For detailed information on XMake and the translation process, refer to the ”Configuring XDM and XMake” and subsequent sections in the “SDT Tutorial” chapter and to the *XACT Reference Guide.*

Creating a Routed Design

1. Quit OrCAD, saving your changes to the VST configuration, and re-enter XDM.
2. Select **`Translate → XMAKE`**.
3. Using the default options, choose the **`CALC.SCH`** file as input.
4. Choose **`Make bitstream`** as the target.

   XMake translates, maps, places, and routes the design, then creates a bit file that you can use to program a device.
5. After you review the XMake output, press any key to return to XDM.

Examining XMake Output

Xmake produces a screen output similar to the following.

```
XMAKE: Generating makefile 'calc.mak' ...
XMAKE: Profile used is the current XDM settings
XMAKE: Execute command 'annotate calc.sch'.
XMAKE: Execute command 'inet calc.sch'.
XMAKE: Execute command 'sdt2xnf calc.inf calc.xnf -D xnf'.
XMAKE: Set the part type to 3020APC68-7 from xnf\calc.xnf'.
XMAKE: Running with the following options: (none)
>>> XDELAY is run always with '-D' and '-W' options by XMAKE.
XMAKE: Makefile saved in 'calc.mak'.
XMAKE: Making 'calc.bit'...
XMAKE: Execute command 'xnfmerge -A -D xnf -D . -P3020APC68-7
xnf\calc.xnf calc.xff'.
XMAKE: Execute command 'xnfprep calc.xff calc.xtf
parttype=3020APC68-7'.
XMAKE: Execute command 'xnfmap -P 3020APC68-7 calc.xtf
calc.map'.
XMAKE: Execute command 'ppr calc.map parttype=3020APC68-7'.
XMAKE: Execute command 'xdelay -D -W calc.lca'
XMAKE: Execute command 'makebits -R2 -S0 -XB -YA calc.lca'
XMAKE: 'calc.bit' has been made. Check output in 'calc.out'.
```

The following is an explanation of the XMake output seen within XDM.

```
XMAKE: Generating makefile 'calc.mak' ...
```

XMake always generates a MAK file that can be used to finish an XMake run that has been halted in the middle. For example, if you chose the Stop To Review DRC option in XMake, you might determine that the DRC output was acceptable, and continue compiling the design by selecting the MAK file as the input to XMake instead of the design file. The MAK file contains a listing of the commands that were executed by XMake, so it is also useful for those who like to create their own custom command scripts.

```
XMAKE: Profile used is the current XDM settings
XMAKE: Execute command 'annotate calc.sch'.
XMAKE: Execute command 'inet calc.sch'.
XMAKE: Execute command 'sdt2xnf calc.inf calc.xnf -D xnf'.
```

XMake runs a series of commands that convert the OrCAD SCH files to Xilinx XNF files.

```
XMAKE: Set the part type to 3020APC68-7 from xnf\calc.xnf
XMAKE: Running with the following options: (none)
>>> XDELAY is run always with '-D' and '-W' options by XMAKE.
XMAKE: Makefile saved in 'calc.mak'.
XMAKE: Making 'calc.bit'...
```

If you have not already done so, XMake runs ABL2XNF for you at this point in the flow to generate an XNF netlist from your Xilinx ABEL input file.

```
XMAKE: Execute command 'xnfmerge -A -D xnf -D . -P3020APC68-7
xnf\calc.xnf calc.xff'.
```

Next XMake runs XNFMerge to flatten the hierarchy of the design and produce a single XFF file that contains a netlist for the entire design. This step merges the XNF file produced by ABL2XNF with the rest of the design.

```
XMAKE: Execute command 'xnfprep calc.xff calc.xtf
parttype=3020APC68-7'.
```

XNFPrep is run to check the validity of the netlist that XNFMerge produces. Many schematic errors are caught at this phase.

```
XMAKE: Execute command 'xnfmap -P 3020APC68-7 calc.xtf
calc.map'.
XMAKE: Execute command 'ppr calc.map parttype=3020APC68-7'.
```

XMake partitions XC3000A designs with XNFMAP before running PPR to place and route the design. Depending on the part for which the design is targeted, programs executed may include XNFMAP for XC3000/XC3000A mapping, APR for XC3000 placing and routing, and/or PPR for XC3000A/XC4000 mapping, placing, and routing.

```
XMAKE: Execute command 'xdelay -D -W calc.lca'
```

XMake runs XDelay to write the proper delay values in the LCA file produced by PPR.

```
XMAKE: Execute command 'makebits -R2 -S0 -XB -YA calc.lca'
```

From the LCA file, MakeBits produces a bitstream that can be used to program a device. The bitstream is placed in a file with a .bit extension.

```
XMAKE: 'calc.bit' has been made. Check output in 'calc.out'.
```

A copy of the entire output of the XMake run is always placed in a file with the same name as the top-level design and an .out extension.

Verifying CALC on the Demonstration Board

XMake created a BIT file that you can download to the appropriate demonstration board to verify the validity of the design. If you are unfamiliar with this process, please refer to the "Verifying the Design Using a Demonstration Board" section of the "SDT Tutorial" chapter for more information.

Timing Simulation

You have already performed many of the steps necessary for timing simulation. The calc.lca file created in the previous design implementation section contains the timing information for the design. All that is necessary is to back-annotate the timing and netlist information from the LCA file to VST. This task is accomplished with the aid of XSimMake.

Creating the Simulation Netlist

Run XSimMake to generate a netlist that can be used for timing simulation.

1. From the XDM menus, select **Verify → XSIMMAKE**.

2. Select the **-F** option.

   A menu of possible flows appears.

3. Choose the **Orcad_Fpga_Timing** option and select **Done**.

4. Choose **CALC.LCA** from the list of design files shown.

5. After you review the XSimMake output, press any key to return to XDM.

As XSimMake runs, text similar to the following appears in the XDM window.

Note: XSimMake flows vary depending on the design. The flow used by XSimMake for your design may be slightly different from the flow shown in this tutorial.

```
XSIMMAKE COMMAND : xdelay -w -d calc.lca
XSIMMAKE COMMAND : lca2xnf -g calc.lca calc.xnf
XSIMMAKE COMMAND : xnfba calc.xff calc.xnf
XSIMMAKE COMMAND : xnf2inf xnfba.xnf calc
XSIMMAKE COMMAND : deleted file xnfba.xnf
0 Errors and 0 Warnings occurred during processing.
```

Examining XSimMake Output

An explanation of the XSimMake output for the timing flow follows. None of the following information is necessary in order to use XSimMake. It is provided to give you insight into the back-annotation process.

```
XSIMMAKE COMMAND : xdelay -w -d calc.lca
```

XDelay is a tool that can be used to perform static timing analysis on a design. In this case, it writes the delay information into the LCA file produced by XMake. Once the delay information is in the LCA file, it can be back-annotated and used in simulation.

```
XSIMMAKE COMMAND : lca2xnf -g calc.lca calc.xnf
```

XSimMake then runs LCA2XNF to convert the LCA description of the design, including the timing information, back into an XNF file.

```
XSIMMAKE COMMAND : xnfba calc.xff calc.xnf
```

XNFBA reads the original (pre-routed) calc.xff netlist and the post-routed calc.xnf netlist generated in the previous step by LCA2XNF and writes out a new netlist. It writes this netlist so that as many

instance and net names match the original schematic as possible. In the process of placing and routing the design, some of this information is lost and must be replaced by XNFBA.

```
XSIMMAKE COMMAND : xnf2inf xnfba.xnf calc
XSIMMAKE COMMAND : deleted file xnfba.xnf
```

The above steps generate simulation input files from the back-annotated netlist, then delete the intermediate XNF file created by XNFBA.

Configuring OrCAD VST for Timing Simulation

Verify that the simulator is correctly configured to simulate the input file, calc.vst. Then configure the simulator to use timing delays from the DBA file.

1. Select **`Design Entry → ORCAD`** to enter the OrCAD design environment.
2. Select **`Digital Simulation Tools → Execute → Simulate → Local Configuration → Configure SIMULATE`**.

   The Configure Simulate screen appears.

Note: OrCAD's configuration programs are memory intensive. If you receive the message `Could not find the .EXE, or not enough memory to load \orcadexe\VST_CLC.EXE`, return to the XDM executive screen. Close XDM by typing **`exit`** ↵, type **`orcad`** ↵, and modify the configuration as described in this section. After completing the commands in this section, type **`xdm`** ↵ to return to XDM, and continue with the "Performing a Timing Simulation" section, following.

3. Under File Options, verify that the file names read as follows:

   ```
   Connectivity database  CALC.VST
   Stimulus file          CALCSIM.STM
   Trace file             CALCSIM.TRC
   ```

4. If the names are incorrect, make the necessary changes.
5. Set the simulator to use timing delays from the DBA file by selecting **`Use Delay Annotation.`**
6. Return to the ESP screen by selecting **`OK`**.

Performing a Timing Simulation

You are ready to perform the timing simulation.

1. Enter the simulator by selecting **`Simulate → Execute`**.
2. Select **`Set → Spool to Disk`**.

   The `Spool Trace Data to Disk?` prompt appears.
3. Select **`Yes → Yes`** to spool the data to the spool.tdf file.

   The default time scale is 10 ns per screen. Adjust that to 1,000 ns per screen, which means increasing the scale by a factor of 100.
4. Select **`Trace → Change View`**.

   The `Trace Delta Time?` prompt appears at the top of the screen.
5. Type **`100`** ↵.
6. Select **`Run Simulation`**.

   The `Simulation Length?` prompt appears.
7. Type **`12000`** ↵ to simulate for 1,200 ns.

The output of this simulation run is similar to the output of the timing simulation run on the original (non-Xilinx ABEL) Calc design. The timing differs slightly, because different place and route runs produce different net delays.

Further Reading

This tutorial shows you the basic functions involved in including Xilinx ABEL-based components in an OrCAD SDT design. Before attempting to build your own Xilinx ABEL design, please review both the *Xilinx ABEL User Guide* and the *Xilinx ABEL Software Design Reference Manual*.

OrCAD Interface/ Tutorial Guide

XACT-Performance and XDelay Tutorial

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Chapter 15

XACT-Performance and XDelay Tutorial

The specification of exact timing requirements on schematics has become a necessity as FPGAs have become larger and designs consequently more complex. The term XACT-Performance refers to the method used by the Xilinx software to describe these timing requirements. XACT-Performance consists of a set of library primitives that allow timing requirements to be placed on a schematic, along with built-in functionality within the PPR program that allows PPR to use this timing information during mapping, placement, and routing of the design.

XDelay is the companion tool that allows you to obtain exact timing information about the routed design created by PPR. Whenever using XACT-Performance, you should verify the path timing using the XDelay program. In order to reduce run time, XACT-Performance does not use the highest possible level of accuracy in computing delays. XDelay reports completely accurate worst-case delays for all Xilinx FPGAs. Differences between the two reports are minor, but when they occur, use the XDelay output as the definitive source for timing information.

Note: Since APR does not interpret XACT-Performance specifications, only XC3000A / L, XC3100A, and all XC4000 family designs can take advantage of the features described in this tutorial. XACT-Performance does not function on XC3000, XC3100, or XC2000 family designs.

The intent of this tutorial is to give a practical example of using XACT-Performance and XDelay within the OrCAD design environment. It is not intended to fully explain all of the functionality found within XACT-Performance or XDelay. Please refer to the “Further Reading” section at the end of this tutorial for a list of sources from which to obtain more information.

Before Beginning the Tutorial

This section of the tutorial assumes that you are already familiar with the material in the "SDT Tutorial" and "VST Tutorial" chapters of this manual. If not, please review those chapters before continuing.

Required Software

You should have access to the following software:

- OrCAD / SDT — SDT 386+
- OrCAD / Xilinx Interface — version 5.00 or later
- XACT Design Manager (XDM) — Version 5.00 or later

If you have Xilinx software on CD-ROM, you should have at least temporary access to all of the above software (with the exception of SDT) using the temporary licensing available on the programmable key, provided that the temporary licensing has not already been exhausted.

Preparing the Design

If you chose to read through the SDT tutorial rather than actually perform the steps involved, you must verify that your PC is set up correctly to use OrCAD 386+ and the XACT Development System software. Then create and configure the design directory, and copy a completed set of schematics from any of the solutions directories supplied.

If you have already performed the SDT tutorial on your PC, skip to the next section, "Understanding XACT-Performance."

1. Follow the instructions given in the "Before Beginning the Tutorial" section of the "SDT Tutorial" chapter for setting up your design environment.
2. The tutorial files are optionally installed when you install the Xilinx / OrCAD interface software. If you have already installed the software but are not sure whether you specified tutorial installation, check for the c:\xact\tutorial\orcad\calc directory. This directory contains the tutorial files.

3. Create a new project called “Calc,” as described in the “Creating the Project Directory” section of the “SDT Tutorial” chapter.

4. Full solutions for the SDT tutorial are supplied in the solutions directories located in \xact\tutorial\orcad\calc. One of the following file directories must be copied to the directory where you will be performing the tutorial.

   ...\soln_3ka — Solution files for XC3020APC68
   ...\soln_4ka — Solution files for XC4003APC84
   ...\soln_4k — Solution files for XC4003PC84

   For example, if you are targeting the tutorial for an XC3020APC68 device, perform the following sequence of commands.

   `cd \orcad\calc`↵
   `copy \xact\tutorial\orcad\calc\`*`solution`*`\*.*`↵

   where *solution* is either soln_3ka, soln_4ka, or soln_4k. The solutions schematics are targeted towards the 3020APC68-7, 4003APC84-6, and 4003PC84-6 devices, respectively.

Note: This tutorial assumes that your ORCADPROJ variable is set to c:\orcad\. You need not follow this convention.

   This procedure gives you a full set of completed schematics for the Calc design, with all supporting files.

5. Configure the design directory by typing **`xdraft 3`**↵ or **`xdraft 4`**↵, depending on whether you are targeting an XC3000A or an XC4000 family device.

Note: All of the screen outputs refer to the processing of the 3000A solutions design. If you wish to reproduce the software output as referenced in this tutorial, copy the XC3020APC68 solutions design. Other devices have slightly different outputs.

Understanding XACT-Performance

When discussing the timing requirements of a design, it is simple to describe a requirement in such terms as “this path must get from the source to this load in a certain amount of time.” XACT-Performance uses a similar from:to type of syntax. Symbols are grouped into sets, and these sets are then used as endpoints for timing specification. Timing requirements are defined as the maximum acceptable delays

from the sources in one defined set, through intermediate combinatorial logic, to the associated loads in another set.

The three steps for adding timing specifications to a schematic are as follows:

1. Add TNM attributes to symbols on your schematic to group them into sets. This step is not necessary if you are using only predefined sets.
2. Add a TIMEGRP text string and define new timegroups based on the sets defined in step 1. These timegroups combine existing sets into additional, more complex, sets. This step is optional.
3. Add a TIMESPEC text string and specify the timing requirements for the sets defined in steps 1 and 2.

Grouping Symbols with TNM Attributes

The most basic and flexible way of defining these sets is through the addition of TNM (Timing NaMe) attributes to symbols on a schematic. By giving two or more symbols TNM attributes with identical values, these symbols become part of the same set, which you can reference in a from:to statement.

TNM attributes can be applied to library symbols with the Edit Edit Options Name command. You cannot add TNM attributes to sheet symbols: you must either create corresponding library symbols or use one of the other TIMEGRP techniques.

TNMs on Logic Primitives

TNMs are applicable to four types of primitives: flip-flops, RAMs, I/O pads, and IOB latches. A set may not contain more than one of these types of symbols, with the exception of flip-flops and IOB latches, which may be included in the same set. TNMs on other primitives, such as OR gates, are invalid.

The syntax of the TNM attribute is as follows:

`TNM=`*identifier*

where *identifier* is replaced with the name of the set. The name can be any ASCII string using only the characters A-Z, a-z, _, and 0-9.

TNMs on Higher-Level Macro Symbols

You can also place TNM attributes on library macro symbols containing one or more of the logic primitives just discussed. The TNM attribute is passed down through the hierarchy and placed on the logic primitives below.

If the macro contains primitives of more than one type, you must specify the types of primitives inside the macro to which the TNM attribute applies. For example, a macro may contain RAMs and flip-flops. If you place a TNM on this macro, you must specify it as applying to either the RAMs or the flip-flops. Since RAMs and flip-flops cannot be in the same set, if neither set is specified XMake fails while running XNFMerge, and issues a message explaining the error.

The syntax for applying TNM attributes to macros is as follows. You can specify one or more of the primitive types.

TNM=FFS:*identifier*;RAMS:*identifier*;LATCHES:*identifier*;
PADS:*identifier*

In this case, each instance of *identifier* is replaced by a unique set name, with the exception of FFS and LATCHES, which can be in the same set if desired.

TNMs on Nets, to Tag Flip-Flops

The TNM attribute can also be placed on nets, by using the Get command to place the TNM symbol, attaching it to your net using the Place Wire and Place Junction commands, then using Edit Edit Value Name on the TNM symbol to add the TNM=*identifier* attribute. The software traces the signal to all load pins on the net and forward through combinatorial logic, and applies the TNM to any flip-flops reached. This spreading of TNM specifications to load pins is known as forward tracing.

For this purpose, if RAMs are encountered while tracing forward to load pins, they are seen as transparent. This means that if a flip-flop is sourced by the output of a RAM, and a TNM property is attached to the write enable of the RAM, the flip-flop becomes part of the set.

Grouping Symbols by Predefined Sets

In many cases, it makes sense to apply a timing requirement to all associated symbols of a certain type. For example, a given flip-flop output may have a clock-to-setup timing requirement that applies to all other flip-flops driven by the flip-flop output.

In order to simplify the grouping procedure in such cases, Xilinx provides four predefined sets. These sets are FFS (flip-flops), PADS (I/O pads), RAMS (XC4000 family RAM elements), and LATCHES (IOB latches). Instead of placing a TNM attribute on each symbol, you can reference the entire set in a from:to statement by taking advantage of the predefined sets.

In the flip-flop example just discussed, you can use the from:to syntax to specify that the timing requirement be applied from the source flip-flop to the predefined set FFS.

Simplifying Symbol Grouping

The simplest way to group symbols is to use the basic syntax, TNM=*identifier*, on primitives. The other methods are shortcuts that enable you to quickly define sets that are related in some way, such as instances within the same library macro, flip-flops driven by a common net, and so forth.

Combining Sets: TIMEGRP

Once sets are defined using TNM attributes, it can be useful to define new sets in terms of the existing sets. You may wish to join two or more sets into one, define a set of all symbols not already included in another set, or designate a set of flip-flops triggered by a given clock edge. You can also use TIMEGRP to designate a set by the output net names of the primitive symbols.

To create these new sets, add the TIMEGRP text string to your schematic, then add a text string for each new attribute. The name of the attribute is the new set name. The value of the attribute is the set definition.

The format of the text strings must be as follows:

```
|TIMEGRP
|group1=set_definition1
|group2=set_definition2
```

The pipe characters must be vertically aligned, with the group definitions on succeeding lines below the TIMEGRP text. Do not place any of this text in the same horizontal row as PARTTYPE or TIMESPEC text strings. Do not place the pipe characters in the same vertical column as pipe characters from PARTTYPE or TIMEGRP text.

You can place TIMEGRP text strings at any level of your hierarchy. There is no limit to the number of group specifications.

Joining Two or More Sets into One

You can define a new set as the combination of two or more existing sets using the following syntax:

```
|new_set=set1:set2[...:setn]
```

Using the EXCEPT Statement

A set defined using TNM attributes may account for all but a few of the flip-flops in a design. One way to apply timing specifications to the rest of the flip-flops is to create a new set that consists of all flip-flops not already in the first set. You can create the new set by defining an attribute that contains an EXCEPT statement. Use the following syntax:

```
|new_set=set1:EXCEPT:set2
```

where *set1* is replaced by one of the predefined sets (FFS, PADS, RAMS, or LATCHES) or by the name of a user-defined set. *Set2* is replaced by the name of a user-defined set.

For example, in the situation just discussed, assume that the set defined using TNMs is called FFGRP1, and the new set name is FFGRP2. You can create the set of all flip-flops not in the set FFGRP1 by adding the following text string below the TIMEGRP text:

```
|FFGRP2=FFS:EXCEPT:FFGRP1
```

Triggering on RISING or FALLING Clock Edges

You can also use TIMEGRP symbol attributes to make subsets of flip-flops that are triggered by a certain clock edge. Use the following syntax:

|*new_set*=RISING:*set*

|*new_set*=FALLING:*set*

The new set consists of all symbols within *set* that are clocked by the specified clock edge.

Forming Sets by Output Net Name

You can define a new set as the set of all primitives with output net names starting with a certain string. (BLKNMs or HBLKNMs are used for PADS, if you added these attributes; otherwise the external net name is used.) The full hierarchical net name is used. This technique is particularly useful when you wish to form a set, for example, of all flip-flops placed within a given sheet symbol. *Provided the output net is not renamed at a higher level of the hierarchy*, the hierarchical output net name starts with the sheet name of the symbol.

Specify the set of all primitives with output net names beginning with *name* using the following syntax:

|*new_set*=*class*(*name)**

where *class* is one of FFS, RAMS, LATCHES, or PADS. This designation defines a new set called *new_set*, which consists of all blocks in the designated class with output net names starting with the string *name*.

Note: This TIMEGRP capability must be used with caution. If your design contains unrelated nets with names beginning with the same string, they may be included in your time group or cause errors in XNFMerge or XNFPrep. If you attempt to apply the attribute to all blocks in a given sheet symbol, but the outputs of some flip-flops are renamed at a higher level of hierarchy, they are not included in the set.

Attaching Timing Specifications: TIMESPEC

Once you have defined appropriate sets by attaching TNM attributes to symbols, and, optionally, by combining these sets using TIMEGRP text strings, the next step is to add the timing specifications to the schematic.

First place a TIMESPEC text string on the schematic, then add the from:to timing requirements in the form of attributes in the following lines. The name of the attribute is TS followed by a unique identifier. The value of the attribute defines the paths to which the specification applies, and the timing requirement that applies to the paths.

The format of the text strings must be as follows:

|TIMESPEC
|TS*name1*=FROM:*set1*:TO:*set2*=*time1*
|TS*name2*=FROM:*set3*:TO:*set4*=*time2*

The *set* references are replaced with the appropriate set names, as defined by using TNMs and TIMEGRP attributes or by using the pre-defined sets. *Time* specifies the timing requirement in microseconds (US), nanoseconds (NS), kilohertz (KHZ), or megahertz (MHZ). If no units are specified, *time* is assumed to be in nanoseconds.

The pipe characters must be vertically aligned, with the attribute definitions on succeeding lines below the TIMESPEC text. Do not place any of this text in the same horizontal row as PARTTYPE or TIMEGRP text strings. Do not place the pipe characters in the same vertical column as pipe characters from PARTTYPE or TIMEGRP text.

For example, to specify that the pad-to-setup path delay between all pads and all flip-flops should be no greater than 40ns, place the following text anywhere in your schematic:

```
|TIMESPEC
|TS01=FROM:PADS:TO:FFS=40NS
```

Note: FROM:PADS:TO:FFS is not exactly equivalent to pad-to-setup, since the PADS group includes not just data pads but also clock pads. Therefore, FROM:PADS:TO:FFS includes both pad-to-setup and pad-to-clock specifications. This inclusion is normally an advantage; however, if desired, you can use the EXCEPT syntax to eliminate the pad to clock paths. For example, to create a source group equivalent to the

set referenced by pad-to-setup, use the TIMEGRP symbol to define a group such as PADS:EXCEPT:pads_sourcing_clocks.

You can place TIMESPEC text strings at any level of your hierarchy. There is no limit to the number of attribute specifications.

Deciding When to Use XACT-Performance

The ideal approach to using XACT-Performance is to route your design once without any timing constraints. PPR by default controls path timing, using reasonable default values it calculates based on your design. If the resulting LCA file meets your timing requirements, your design is complete.

If not, the PPR log file (ppr.log) gives values that can be achieved for FFS:TO:FFS, PADS:TO:FFS, and FFS:TO:PADS timing. Use these values to help determine reasonable default timing requirements as described in the following section, “Setting Default Timing Requirements.”

After this run, check the new log file. If PPR is unable to meet the default timing for all paths, it reports the paths for which the default is not met. If critical paths in your design are not fast enough to meet your specifications, it is time to consider adding more specific constraints, as described in the “Adding Timing Constraints to Specific Paths” section.

Clearly, tightening the default specifications for the entire design is unlikely to help PPR speed up the critical paths. Instead, consider a tighter specification on the most critical paths, combined with a looser specification for unimportant paths. You can even assign a value of IGNORE to some classes of paths, which is a very effective technique because it clearly tells PPR that it can sacrifice timing on the unimportant paths in order to improve timing on the important ones. To use the IGNORE value, you must use the following syntax:

|TS*name*=*class*:IGNORE

where *class* is one of the following special sets: DC2S, DC2P, DP2S, or DP2P.

You may want to skip the first step and start by setting reasonable default timing requirements.

If XACT-Performance and PPR are unable to achieve the speed

needed for your application, you may have simply reached the limits of the hardware and / or software. You can increase the speed of the hardware by using a part with a faster speed grade. The tutorial designs, being designed to work with the Xilinx Demonstration Boards, use the slowest available speed grades.

Consider speeding up your design by making changes to the logic to use the Xilinx FPGA architectures to better advantage. For example, try reducing the number of logic levels between flip-flops in critical paths. Xilinx FPGA architectures are rich in flip-flops, so pipelining is a good approach. Alternatively, you can often increase the speed of your design by using floorplanning: planning the placement of your logic to simplify the data flow and locking down symbol locations using CLBMAPs, FMAPs, HMAPs, and LOC attributes. See the *XACT Libraries Guide*, the *XACT Reference Guide*, and the other chapters of this manual for more information on how to floorplan your design.

Setting Default Timing Requirements

In this tutorial, you add XACT-Performance attributes to the Calc schematics but do not otherwise change the design. Many of the TIMESPEC constraints used in the following tutorial are not actually necessary for this or similar applications. They are used here to illustrate different usages of XACT-Performance that you may find useful in other designs.

Note: For more information on the Calc design, refer to the discussion in the “Design Description” section of the “SDT Tutorial” chapter.

Adding a TNM Attribute

First, use a TNM attribute to define a group of flip-flops. The group is used in the next section to specify the clock to pad default timing requirement.

1. Open OrCAD SDT 386+ and load the CALC top-level schematic.
2. Select **Get** from the menu.

   The `Get?` prompt appears at the top of the screen.
3. Type **tnm**↵.

4. Move the cursor so the TNM symbol is above the net labeled CLK.
5. Select **Place** to place the symbol on the schematic, and press **Escape** to terminate the command.
6. Use the Place Wire and Place Junction commands to attach the TNM symbol to the CLK net, as shown in Figure 15-1.
7. Place the cursor over the TNM symbol and select **Edit → Edit → Part Value → Name**.

   The `Value? TNM` prompt appears.

8. Type **=FFGRP**↵ and press **Escape** twice to terminate the command.

   The part value TNM=FFGRP is displayed above the TNM symbol.

You have defined a set with the name FFGRP that consists of all flip-flops driven by the clock net CLK.

Note: Because there is only one clock in the Calc design, the FFGRP set in this case is the same as the predefined set FFS, which includes every flip-flop in the design. However, the set is defined in order to demonstrate the effects of attaching TNMs to clock signals, a very common technique in XACT-Performance.

Entering Default Timing Specifications

Next, set the default timing specifications for the clock. For the tutorial design, assume that the clock speed is 500kHz. This clock speed is quite slow, so the place and route software has no problem meeting the timing requirements.

1. Use the Place Text command to place the following text on the schematic, as shown in Figure 15-1.

   ```
   |TIMESPEC
   |TS01=FROM:FFS:TO:FFS=500KHZ
   |TS02=FROM:PADS:TO:FFS=1MHZ
   |TS03=FROM:FFGRP:TO:PADS=1000NS
   ```

Warning: If you have INET v1.10 or 1.10 H, you must perform an additional step each time you add text to the schematic. This version of INET discards the first line of each group of pipe text, after the first group of text encountered on the schematic. For each placement of text on the schematic, *including the part type already present*, you must

place an additional text line consisting of a single pipe character above the first line of the text, with the pipe characters vertically aligned. INET discards this pipe character and correctly reads the text in the next line. INET v1.08 did not display this behavior and it should be fixed in any release subsequent to v1.10.

2. Select **Quit** → **Update File** → **Escape** to save your changes to the CALC schematic.

The new attribute TS01 specifies that all clock-to-setup paths must have timing such that they can be driven by a 500kHz clock.

Pad-to-setup and clock-to-pad path delays are typically half of the clock-to-setup requirement. For the Calc design, they must be driven by a 1-MHZ clock. TS02 specifies that all pad-to-setup path delays have timing such that they can be driven by a 1-MHz clock.

Making use of the FFGRP set, which in this case is equivalent to the FFS set, TS03 specifies that the clock-to-pad timing for the design be a maximum of 1000 ns. The two timing specifications of 1000NS and 1MHz are interchangeable. An equivalent specification for TS03 is |TS03=FROM:FFS:TO:PADS=1MHZ.

Figure 15-1 shows a portion of the CALC schematic with the TNM attribute and TIMESPEC text strings.

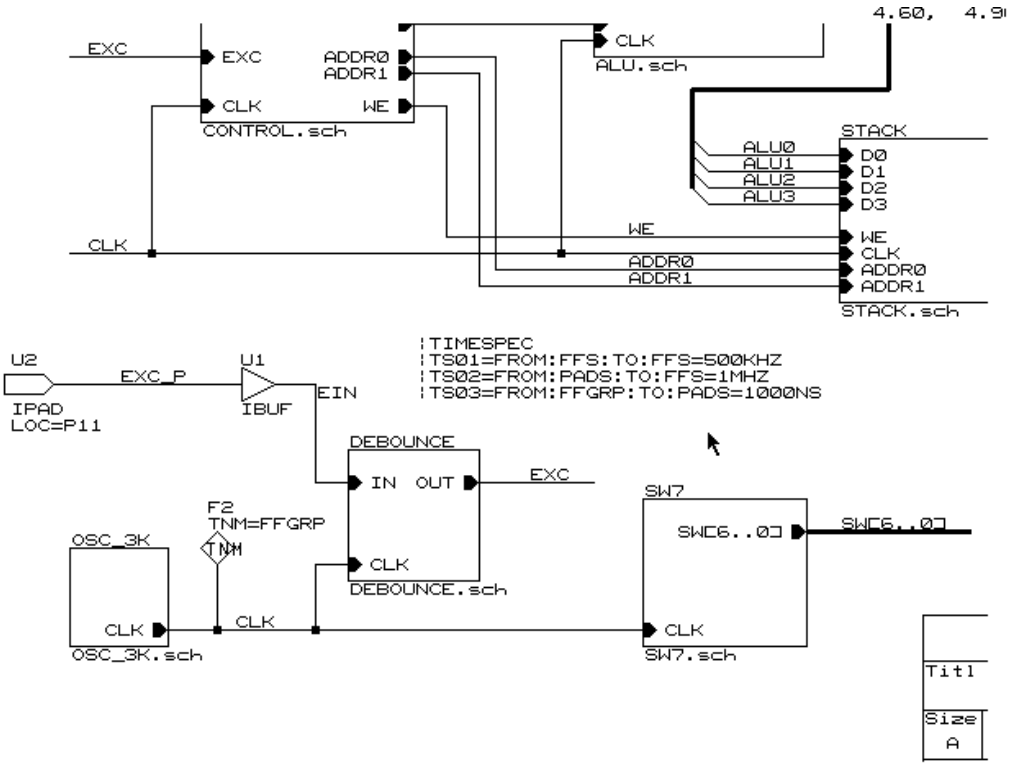


Figure 15-1 Default Timing Constraints on CALC Schematic

Adding Timing Constraints to Specific Paths

In the previous section, you applied default timing specifications to the Calc tutorial design. In real applications, the above specifications usually supply enough guidance to allow PPR to meet the timing requirements. However, in this section the tutorial continues with more specific path timing constraints to illustrate the application of XACT-Performance to more specific groups of paths in a design.

Defining TNM Groups

First, define sets to be used as endpoints for timing specification. You can use these sets to apply timing requirements from one set to another or from one set to the same set.

Defining the ALUFF Set

The ALUFF set includes all flip-flops from the ALU sheet symbol. Define the set by adding the TNM attribute to the CE pin of the ALU. This pin drives all flip-flops in the ALU and no other flip-flops.

1. Select **Get** from the menu.

   The `Get?` prompt appears at the top of the screen.

2. Type **tnm**↵.

3. Move the cursor so the TNM symbol is in the open area above the CE_ALU signal, as shown in Figure 15-2.

4. Select **Place** to place the symbol on the schematic, and press **Escape** to terminate the command.

5. Use the Place Wire and Place Junction commands to attach the TNM symbol to the CE_ALU net.

6. Place the cursor over the TNM symbol and select **Edit → Edit → Part Value → Name**.

   The `Value? TNM` prompt appears.

7. Type **=ALUFF**↵ and press **Escape** twice to terminate the command.

   The part value TNM=ALUFF is displayed above the TNM symbol.

You have defined a set named ALUFF that consists of all flip-flops driven by the clock enable signal CE_ALU.

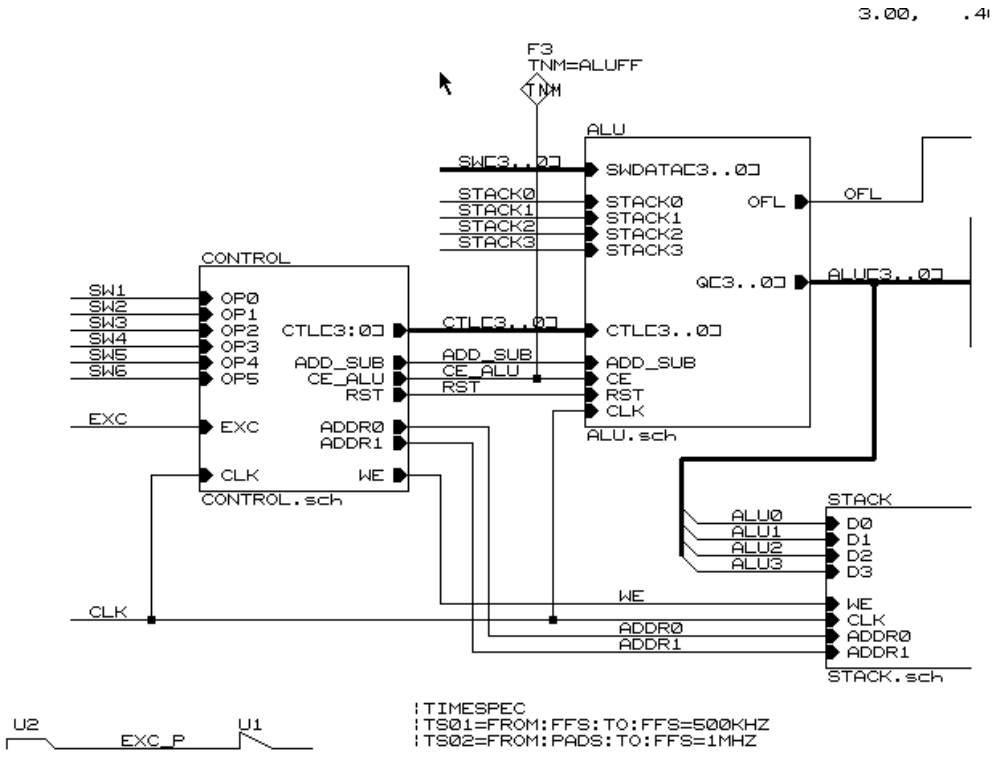


Figure 15-2 Defining the ALUFF Set

Defining the CTL_ADR_FF Set

The CTL_ADR_FF set includes all flip-flops from the CB2CLED counter macro in the CONTROL sheet. To define the set, push into the CONTROL sheet. The counter is a library macro, to which you can attach a TNM attribute.

1. Place the cursor over the CONTROL symbol in the top level CALC schematic.

2. Select **Quit → Update File → Enter Sheet → Enter → Escape** to save your changes and push into the CONTROL schematic.
3. Place the cursor over the CB2CLED symbol.
4. For an XC3000A device, select **Edit → Edit → LOC,OPTIONS → Name**. For an XC4000 family device, select **Edit → Edit → OPTIONS_1 → Name**.
5. Type **TNM=FFS:CTL_ADR_FF**↵ and press **Escape** twice to terminate the command.

   The attribute TNM=FFS:CTL_ADR_FF is displayed below the CB2CLED symbol, as shown in Figure 15-3.

Note: Whenever a library macro contains more than one class of primitives, you must specify the class to which the attribute applies. In this case, since the counter contains no RAMS, PADS, or LATCHES, the class specification is optional.

You have defined a set named CTL_ADR_FF that contains all flip-flops in the CB2CLED macro.

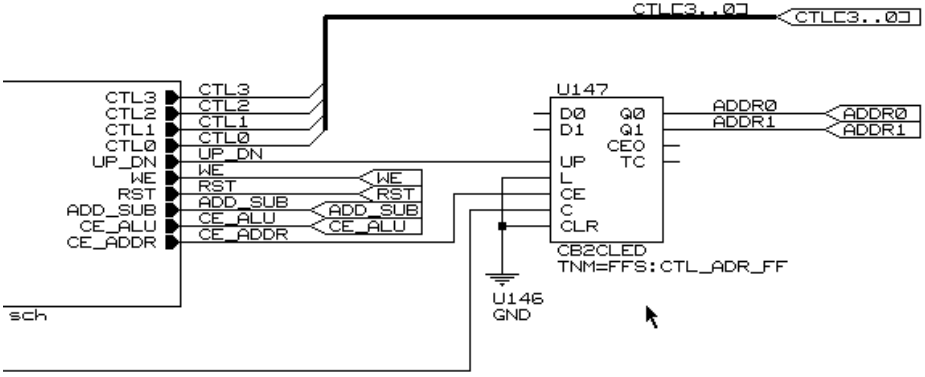


Figure 15-3 TNM Attribute in CONTROL Schematic

Defining the STFF Set

The STFF set includes all flip-flops in the state machine. Define the set by adding the TNM attribute to each FD in the schematic. The completed schematic for STATMACH, the XC3020A version of the state machine, is shown in Figure 15-4.

Note: If you already performed the Xilinx ABEL tutorial on this design, the state machine consists of a single library symbol called STAT_ABL. Use the technique described in the previous section, "Defining the CTL_ADR_FF Set," to add the TNM=STFF attribute to the symbol, rather than following the steps in this section.

1. Place the cursor over the STATMACH (XC3000A) or STATE_4K (XC4000 family) symbol in the top-level CALC schematic.
2. Select **`Quit → Update File → Enter Sheet → Enter → Escape`** to save your changes and push into the state machine schematic.
3. Placing the cursor over each FD symbol in turn, select **`Edit → Edit → Options`** (LOC,OPTIONS or OPTIONS_1) **`→ Name`**, and type TNM=STFF↵.

   The attribute and value appear below each FD as shown in Figure 15-4.
4. Select **`Quit → Update File → Leave Sheet → Leave Sheet → Escape`** to save your changes and return to the CALC schematic.

This attribute definition places each of the FDs in the state machine into a new timing set called STFF.

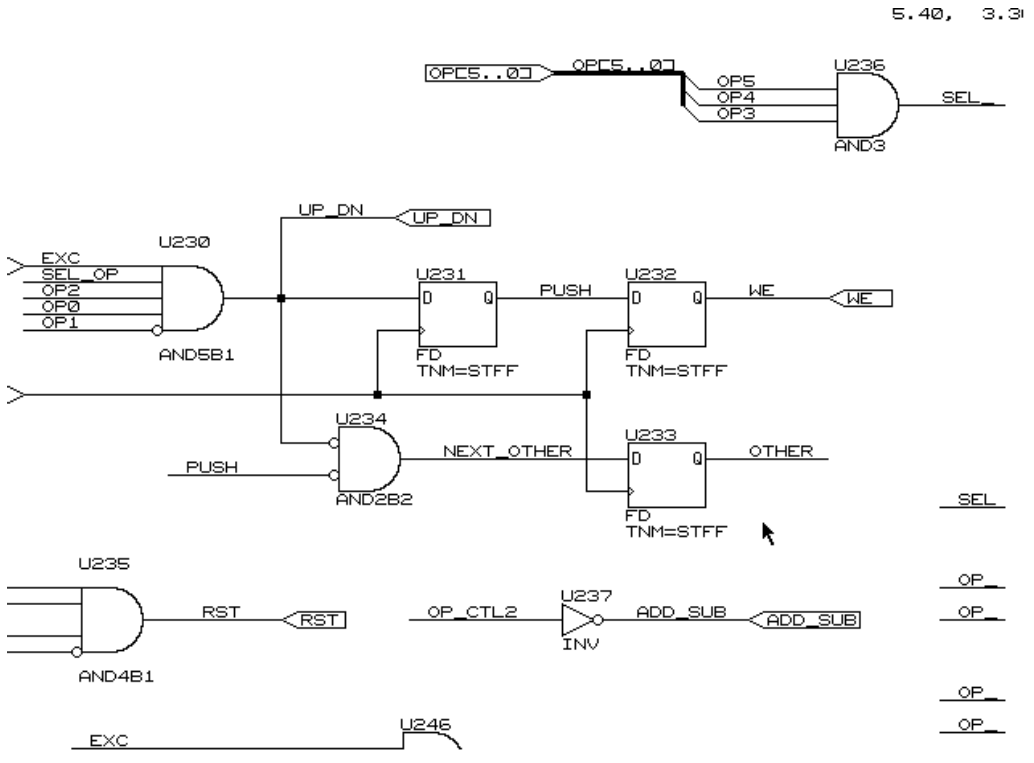


Figure 15-4 TNM Attributes in State Machine (XC3020A Shown)

Defining the INFFS Set

The INFFS set includes all input flip-flops from the SW7 macro. Define the set by adding the TNM attribute to each IFD in the sheet. The completed schematic is shown in Figure 15-5.

1. Place the cursor over the SW7 symbol in the top-level CALC schematic.
2. Select **Quit → Enter Sheet → Enter → Escape** to push into the SW7 schematic.

3. Placing the cursor over each IFD symbol in turn, select **Edit** → **Edit** → **Options** (LOC,OPTIONS or OPTIONS_1) → **Name**, and type TNM=INFFS↵.

   The attribute and value appear below each INFF as shown in Figure 15-5.

4. Select **Quit** → **Update File** → **Leave Sheet** → **Escape** to save your changes and return to the CALC schematic.

This attribute definition places each of the IFDs (IOB flip-flops) in the SW7 macro into a new timing set called INFFS.

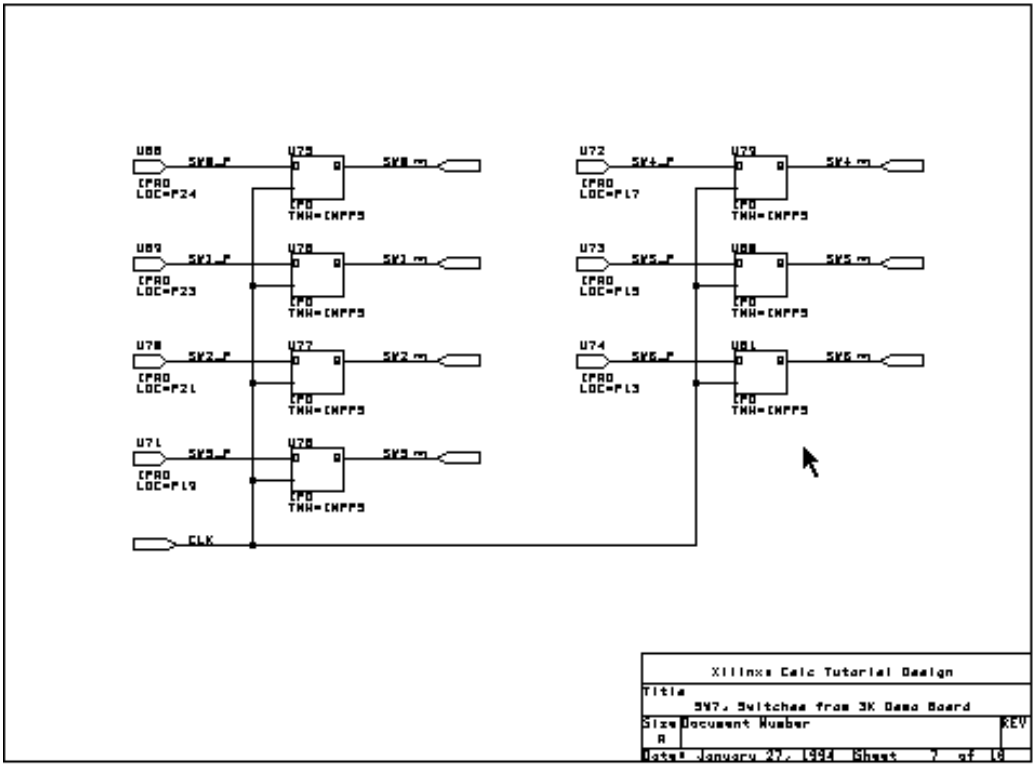


Figure 15-5 Defining the Set INFFS (XC3020A Shown)

Defining Sets with TIMEGRP

Applying TNM attributes to form sets is very useful for tagging library symbols. Alternatively, you could have defined the INFFS set with a single TIMEGRP statement, using the same technique you use to define the next two sets.

Defining the LEDPADS Set

The next set, LEDPADS, represents all symbols of the PADS type that begin with the character string "LED." In this case, the pad symbols themselves do not have assigned BLKNM attributes, so XACT-Performance uses the full hierarchical names from the attached nets. The pads in the LED block are therefore named LED/LED0_P, LED/LED1_P, and so forth.

1. Select **Place → Text.**

   The Text? prompt appears.

2. Place the following text in the upper left corner of the CALC schematic, as shown in Figure 15-6.

   |TIMEGRP
   |LEDPADS=PADS(LED*)

This attribute defines the new set named LEDPADS to contain all I/O pads with external net names starting with the string "LED," which in this case is all pads within the macro with sheet name LED.

Defining the STACKER Set (XC4000 Family Only)

The Calc design for the XC4000 family contains a stack implemented using on-chip RAM elements. In this section, these RAM elements are grouped into a set called STACKER. If your design is an XC3000A design, skip this section and continue with the next section, "Defining the STACKER Set (XC3000A Only)."

1. Select **Place → Text.**

   The Text? prompt appears.

2. Place the following text directly below the previous TIMEGRP text, similar to Figure 15-6. (Figure 15-6 shows the TIMEGRP text for the 3020APC68.) Make sure that the pipe characters are lined up vertically.

   |STACKER=RAMS(STACK*)

This attribute defines the new set named STACKER to contain all RAM elements with output net names starting with the string "STACK," which in this case is all RAM elements in the STACK _4K macro.

Defining the STACKER Set (XC3000A Only)

The Calc design for the XC3000A implements the stack using flip-flops. In this section these flip-flops are grouped into a set called STACKER. If your design is an XC4000 family design, skip this section and continue with the next section, "Combining Existing Sets with TIMEGRP."

1. Select **`Place → Text.`**

   The `Text?` prompt appears.

2. Place the following text directly below the previous TIMEGRP text, as shown in Figure 15-6. Make sure that the pipe characters are lined up vertically.

   |STACKER=FFS(STACK*)

This attribute defines a new set named STACKER that contains all flip-flops with output net names starting with the string "STACK," which in this case is all flip-flops in the STACK macro.

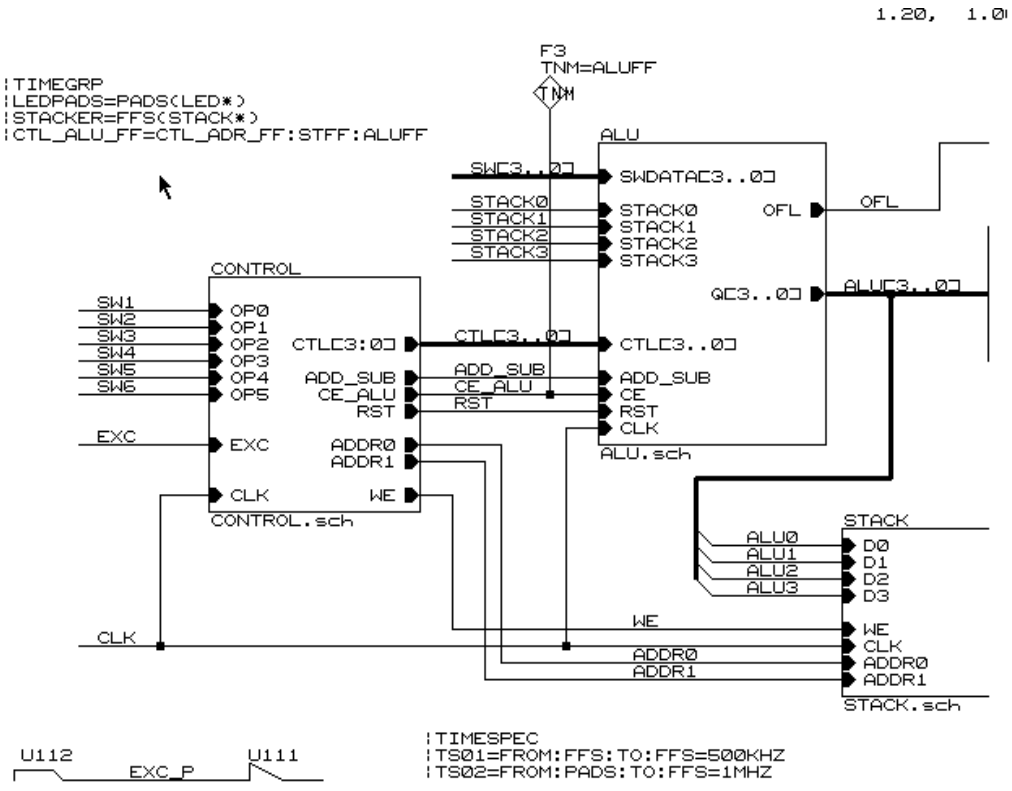


Figure 15-6 TIMEGRP Attributes on CALC Schematic

Combining Existing Sets with TIMEGRP

In addition to the TNM sets, you can use the TIMEGRP text string to define new sets in terms of existing sets or predefined symbol types (FFS, RAMS, PADS, LATCHES).

Use the TIMEGRP text string to create the CTL_ALU_FF set.

1. Use the Place Text command to place the following text directly below the previous TIMEGRP text, as shown in Figure 15-6. Make sure that the pipe characters are lined up vertically.

   |CTL_ALU_FF=CTL_ADR_FF:STFF:ALUFF

2. Select **Quit → Update File → Escape** to save your changes to the CALC schematic.

This attribute defines a new set named CTL_ALU_FF that combines three TNM sets, CTL_ADR_FF, STFF, and ALUFF. This set includes all flip-flops in any of these sets.

Specifying TIMESPEC Constraints

After completing all set definitions, specify the timing constraints. Use the defined sets from TNM and TIMEGRP and the predefined sets FFS, RAMS, PADS, and LATCHES as endpoints of the timing paths.

Since there is limited space below the TIMESPEC text string already on the top-level schematic, add an additional TIMESPEC text string in the lower left-hand corner of the CALC schematic, as shown in Figure 15-9.

1. Use the Place Text command to place the following text on the schematic.

   |TIMESPEC
   |TS04=FROM:INFFS:TO:FFS=80NS
   |TS05=FROM:CTL_ADR_FF:TO:ALUFF=50
   |TS06=FROM:CTL_ALU_FF:TO:STACKER=30
   |TS07=FROM:STACKER:TO:LEDPADS=50
   |TS08=FROM:ALUFF:TO:PADS=45

2. Select **Quit → Update File → Escape** to save your changes to the CALC schematic.

The constraints that you have specified are the following.

- The TS04 timing attribute specifies that the clock-to-setup timing from the set of flip-flops named INFFS to all flip flops (FFS) of the design be no more than 80 ns.
- TS05 specifies the clock-to-setup timing from the TNM set CTL_ADR_FF to the set of flip-flops named ALUFF to be 50 ns.
- TS06 specifies the clock-to-setup timing from the TIMEGRP CTL_ALU_FF to the set STACKER to be 30 ns.

- TS07 specifies the maximum path delays from the time the STACKER data becomes valid, plus any combinatorial delays, to the TIMEGRP LEDPADS, to be 50 ns.
- TS08 specifies the clock-to-pad timing from the TNM set ALUFF to all the pads in the design to be 45 ns.

Making a Final Check

Check to make sure that the TIMEGRP and TIMESPEC specifications look like the ones in Figure 15-7 and Figure 15-8. The order of the attributes is unimportant.

The completed CALC schematic for the 3020APC68 is shown in Figure 15-9.

Warning: If you have INET v1.10 or 1.10 H, you must perform an additional step each time you add text to the schematic. This version of INET discards the first line of each group of pipe text, after the first group of text encountered on the schematic. For each placement of text on the schematic, *including the part type already present*, you must place an additional text line consisting of a single pipe character above the first line of the text, with the pipe characters vertically aligned. INET discards this pipe character and correctly reads the text in the next line. INET v1.08 did not display this behavior and it should be fixed in any release subsequent to v1.10.

All desired XACT-Performance specifications have been entered on the schematic. The next step is to implement the design using XMake and verify the results in the calc.out and ppr.log files.

	TIMEGRP
|	LEDPADS=PADS(LED*)
|	STACKER=FFS(STACK*) **or** | STACKER=RAMS(STACK*)
|	CTL_ALU_FF=CTL_ADR_FF:STFF:ALUFF

Figure 15-7 TIMEGRP Text Definitions

	TIMESPEC
|	TS01=FROM:FFS:TO:FFS=500KHZ
|	TS02=FROM:PADS:TO:FFS=1MHZ
|	TS03=FROM:FFGRP:TO:PADS=1000NS
|	TS04=FROM:INFFS:TO:FFS=80NS
|	TS05=FROM:CTL_ADR_FF:TO:ALUFF=50
|	TS06=FROM:CTL_ALU_FF:TO:STACKER=30
|	TS07=FROM:STACKER:TO:LEDPADS=50
|	TS08=FROM:ALUFF:TO:PADS=45

Figure 15-8 TIMESPEC Text Specifications

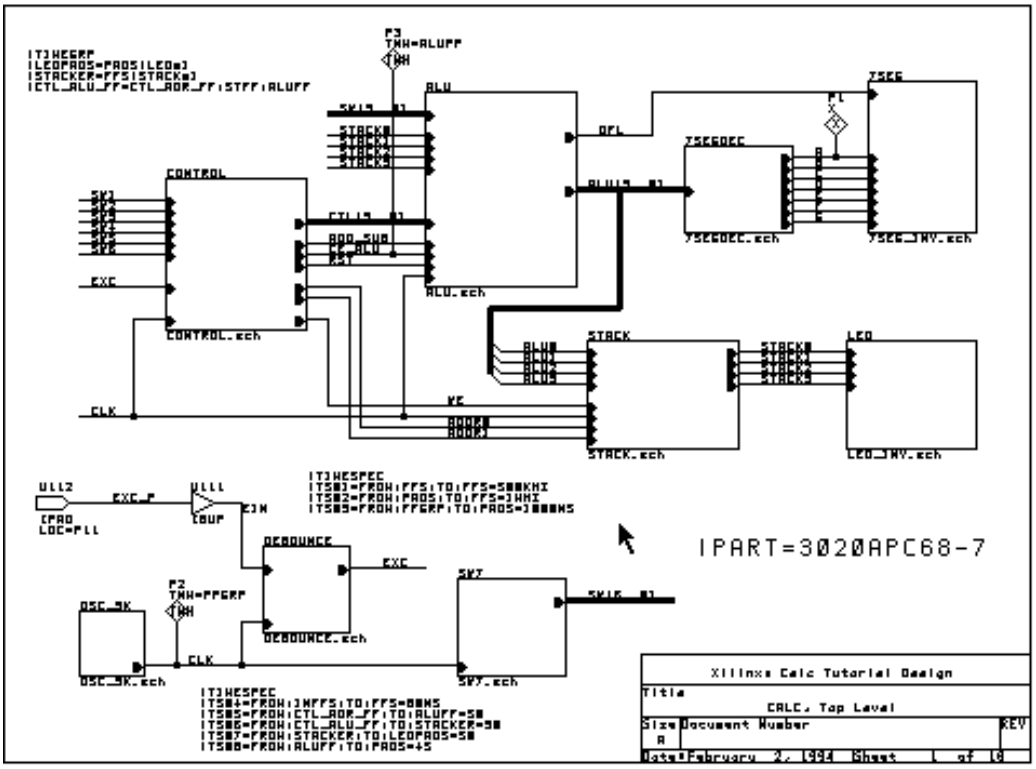


Figure 15-9 CALC Schematic with TNM, TIMEGRP, and TIMESPECs

Cleaning up the Design

From XDM you can call an OrCAD utility, Cleanup, that checks your design for dangling nets, corrects any errors that it can fix, and reports any errors that it cannot correct.

1. Quit OrCAD and enter XDM.
2. Select **`Translate`**.

   The various translation programs supported by Xilinx appear on the Translate menu.

3. Select **CLEANUP**.

   XDM presents a list of all schematic files in your project directory.

4. Select **CALC.SCH** from the list.

   A list of options to the Cleanup command is presented.

5. Select **Done**.

   The Cleanup command displays a list of warnings and errors.

   No warnings or errors should be displayed. If warnings or errors are reported, return to the SDT schematic editor and correct the problem. Rerun the Cleanup program to verify that the warnings or errors do not reoccur.

6. Press any key to return to XDM.

Implementing the Calc Design

The translation of designs containing XACT-Performance attributes is exactly the same as the translation of other designs. In fact, even if you do not specify any XACT-Performance attributes, PPR by default controls path timing. PPR assigns reasonable default values and attempts to meet the self-imposed requirements.

If you apply XACT-Performance attributes to your schematics, PPR detects these specifications and, wherever they apply, uses them instead of calculating default values. In each phase of the implementation, which includes mapping, placement, and routing, PPR takes the XACT-Performance attributes into account. If it is unable to meet a given specification, it issues a warning to the PPR log file, relaxes the requirement, and continues.

Note: You can tell PPR to terminate when it encounters an XACT-Performance specification it cannot meet, by setting stop_on_miss=true.

For more information on PPR, see the “PPR” section of the *XACT Reference Guide*. For detailed information on XMake and the translation process, refer to the “Configuring XDM and XMake” and subsequent sections in the “SDT Tutorial” chapter and to the *XACT Reference Guide*.

Creating a Routed Design

Run XMake to generate a routed LCA file.

1. From the XDM menus, select **Translate → XMAKE**.
2. Using the default options, choose **CALC.SCH** as input.
3. Choose **Make Bitstream** as the target.

XMake translates, maps, places, and routes the design.

Examining XMake Output

XMake produces a screen output similar to the following.

```
XMAKE: Generating makefile 'calc.mak' ...
XMAKE: Profile used is the current XDM settings.
XMAKE: Execute command 'annotate calc.sch'.
XMAKE: Execute command 'inet calc.sch'.
XMAKE: Execute command 'sdt2xnf calc.inf calc.xnf -D xnf'.
XMAKE: Set the part type to '3020APC68-7 from 'xnf\calc.xnf'.
XMAKE: Running with the following options: (none)
>>> XDELAY is run always with '-D' and '-W' options by XMAKE.
XMAKE: Makefile saved in 'calc.mak'.
XMAKE: Making 'calc.bit' ...
XMAKE: Execute command 'xnfmerge -A -D xnf -D . -P 3020APC68-
7 xnf\calc.xnf calc.xff'.
XMAKE: Execute command 'xnfprep calc.xff calc.xtf
parttype=3020APC68-7'.
XMAKE: Execute command 'xnfmap -P 3020PC68-7 calc.xtf
calc.map'.
XMAKE: Execute command 'ppr calc.map parttype=3020APC68-7'.
XMAKE: Execute command 'xdelay -D -W calc.lca'.
XMAKE: Execute command 'makebits -R2 -S0 -XB -YA calc.lca'.
XMAKE: 'calc.bit' has been made. Check output in 'calc.out'
```

If your XACT-Performance specifications have any syntax errors, they are flagged by XNFPrep. XNFPrep is a tool that performs a series of checks on the XFF file, looking for illegal conditions of any sort, and then trims unused logic from the netlist. If XNFPrep detects an illegal XACT-Performance specification, XMake terminates. The XMake output as recorded in the calc.out file prompts you to check the calc.prp file for errors detected by XNFPrep. The PRP file includes a list of all errors and warnings issued by XNFPrep. In this case the error message displays the attribute containing the syntax error and shows the correct syntax for the attribute.

If XNFPrep detects syntax errors in your design, locate the errors by checking the calc.prp file, correct the errors in the schematics, and rerun XMake.

If the calc.out file contains an INF2XNF warning message that an unknown primitive or macro 'TNM' was found, it usually means that the TNM symbol is present in your schematic but is not correctly connected to a net. Verify that you used a wire to connect the symbol to the net and placed the junction correctly.

Always check the *design*.out file after running XMake.

Examining the PPR Log File

After XMake completes, view the ppr.log file produced by PPR.

Figure 15-10 shows portions of the ppr.log file from the soln_3ka solutions directory design. The timing numbers reported vary depending on the target device. Additionally, unless you specify identical input parameters, each PPR run produces a slightly different result.

Note: Once you have a routed design that meets your timing needs, you can make changes to your design while retaining the timing characteristics of the unmodified logic. Use the incremental design procedure discussed in the "Making Incremental Design Changes" section of the "SDT Tutorial" chapter.

Warnings in the PPR Log File

There are several warnings in the PPR log file in Figure 15-10.

Warning 6811 refers to the oscillator loop in the XC3000A design. Since this loop is deliberate, this warning can safely be ignored.

Warning 7030 tells you how many paths are not controlled by timing specifications and how many paths are controlled by more than one timing specification.

Warnings 10601 and 10609 inform you that there was a block name duplication in the design, and a new name was assigned to one of the blocks. Since you did not assign any block names manually, this is not a matter of concern.

Warning 7015 reports that the timing specification TS05 does not meet the deadline. That particular constraint was purposely specified so that it would fail in order to illustrate how PPR displays a failure to

meet an XACT-Performance specification in the log file. The log file tells you which XACT-Performance specification failed and also gives the timing it was able to achieve for the associated set of paths. Since PPR checks timing both during placement and routing, this warning appears twice. Ignore the timing reported in the warning messages, since these are "best estimates" computed before the placement and routing is complete. The data in the timing analysis summary near the end of the log file is more accurate, although for the most accurate results you should run XDelay as shown in the latter half of this tutorial.

The rest of the XACT-Performance constraints all meet the timing requirements.

Warning 7028 is a reminder that PPR does not trace timing through asynchronous set / reset control signals. Since XDelay does trace these paths by default, to compare PPR and XDelay results you must disable this tracing in XDelay. This procedure is discussed in the "Using the FlagBlk Option" section in the XDelay portion of this tutorial.

Warning 10604 occurs whenever PPR saves a new LCA file and there is already an existing LCA file. Since PPR routes the design more than once, these warnings occur in virtually every ppr.log file and can safely be ignored.

Timing Analysis Summary

The tabular timing analysis summary near the end of the ppr.log file shows all XACT-Performance specifications in the design. For each specification, it reports both the timing requirement and the resulting timing for the worst-case path. Any missed specification, such as TS05 in this example, is flagged.

```
ppr 5.0.0 -- Xilinx Automatic CAE Tools
Copyright (c) 1994 Xilinx Inc. All Rights Reserved.

ppr: Reading input design data...
ppr: Placing logic...

*** PPR: WARNING 6811:

This design has 1 purely combinational loop. Such loops
should be avoided. If at all possible, please modify the
design to eliminate all unclocked feedback paths.

A loop of 1 source-to-load connections:
FG4 FG_OSC_3K/Q (Net OSC_3K/Q) to first gate again.

*** PPR: WARNING 7030:

The design has 0 path end-point pairs that are not controlled
by any timing specification and 178 that are controlled by
more than one timing specification. The end-point pairs are
listed in file calc.tsi. Note that there may be multiple
paths between any listed pair of end points. To limit the
number of paths reported in each category, set PPR paramenter
show_tsi_paths = <value>.

TS05: FROM_TO 50.0 ns;/ [best possible, 51.0 ns] = 0.98

*** PPR: WARNING 7015:
At least 1 of 5 path endpoints will miss spec. Continuing
anyway ..

ppr: Routing signals...

*** PPR: WARNING 10601:

The block name ‘WE’ associated with the instance ‘CONTROL/
STATMACH/U250/U3’ has already been used to name another
block. Another block name will be created for this block.

*** PPR: WARNING 10609:

The new block name ‘CONTROL/STATMACH/OTHER’ in assigned to
the instance ‘CONTROL/STATMACH/U250/U3’.

Design has 0 unroutes.

ppr: Generating .LCA File...

*** PPR: WARNING 10604:
An lca file already exists. The old lca file will be saved as
‘calc.lcb’.

ppr: Routing signals...

TS05: FROM_TO 50.0 ns;/ [best possible, 64.4 ns] = 0.78

*** PPR: WARNING 7015:
```

```
At least 4 of 5 path endpoints will miss spec. Continuing
anyway ..

Design has 0 unroutes.

------------------------------------------------------------
Timing analysis summary
------------------------------------------------------------

     Deadline     Actual(*)  Specification
     --------     ---------  -------------

     1000.0ns     48.6ns     TSO3=FROM:FFGRP:TO:PADS
     45.0ns       40.1ns     TS08=FROM:ALUFF:TO:PADS
     50.0ns       43.6ns     TS07=FROM:STACKER:TO:LEDPADS
     30.0ns       24.4ns     TS06=FROM:CTL_ALU_FF:TO:STACKER
(*)  50.0ns       79.2ns     TS05=FROM:CTL_ADR_FF:TO:ALUFF
     80.0ns       66.3ns     TS04=FROM:INFFS:TO:FFS
     1000.0ns     29.4ns     TS02=FROM:PADS:TO:FFS
     2000.0ns     79.2ns     TS01=FROM:FFS:TO:FFS

(*) Note: the actual path delays computed by PPR indicate
that 1 of 8 timing specifications you provided was not met.
To confirm this result, please run xdelay.

*** PPR: WARNING 7028:
The design has flip-flops with asynchronous set/reset
controls (PRE/SD or CLR/RD pins). When PPR analyzes design
timing, it does not trace paths through the asynchronous set/
reset input and on through the Q output.

# of unrouted connections: 0.

ppr: Generating .LCA File...

*** PPR: WARNING 10604:
An lca file already exists. The old lca file will be saved as
'calc.lcb'.

ppr: Making Report File...

- ppr @ 1994/01/16 08:55:22 [00:15:57]
= ---- @ 1994/01/16 00:17:33 [00:15:49]

+ ppr required [3896.727] kbytes of dynamic/allocated memory
```

Figure 15-10 Partial PPR Log File for an XC3020APC68-7 Design

Using XDelay, the Timing Analysis Program

The next step is to verify the timing of your routed design using XDelay.

XDelay is a static timing analysis tool that reports the worst-case timing delays of a routed FPGA design. XDelay has three operating modes:

- Analyze mode quickly shows the maximum clock speed of a design. It reports the worst-case timing paths for each of four typical design path types: pad-to-setup, clock-to-setup, clock-to-pad, and pad-to-pad.
- XDelay-TimeSpec mode verifies which XACT-Performance constraints are met and reports all missed paths in detail.
- XDelay mode provides detailed path timing information according to the selected options and offers insight as to which paths in the design are the most critical. This information helps you determine where to make modifications to meet the design timing requirements.

XDelay has many command options. The following options are the most commonly used. Many of them are demonstrated in this tutorial.

The following options apply to all three operating modes:

- The Flagblk option flags certain blocks to which the path-delay calculator gives special consideration.
- Query / Clear / Save / Read Template are commands to view, clear, save, and load a template file containing XDelay options.

The FailedSpec and SelectSpec options apply to the XDelay-TimeSpec mode:

- The FailedSpec option reports path delays that did not meet an XACT-Performance timing specification. It considers only the specifications selected using the SelectSpec option.
- The SelectSpec option allows you to select from a list of all defined XACT-Performance specifications. The number of delay paths reported for each selected specification is controlled by the TSMaxpaths option.

The following options apply to the XDelay and Analyze modes:

- The ClockToSetup option constrains reporting to paths that start at clocked outputs, such as flip-flop outputs, and end at clocked inputs, such as flip-flop data inputs. Reported delays include the setup requirement on flip-flops.
- The ClockToPad option constrains reporting to paths that start at clocked outputs, such as flip-flop outputs, and end at output pads.
- The PadToSetup option constrains reporting to paths that start at input pads and end at clocked inputs, such as flip-flop data inputs. Reported delays include the setup requirement on flip-flops.
- The PadToPad option constrains reporting to all paths that start at input pads and end at output pads, with only combinatorial logic elements in the path. (This option is not demonstrated, because the only unclocked paths in the tutorial design are in the clock oscillator in the XC3000A.)
- The FromFF and ToFF options together allow reporting on specific paths by selecting the flip-flops at the endpoints of the paths.

For more information on using XDelay, refer to the "XDelay" section of the *XACT Reference Guide*.

Analyzing the Calc Design

This section analyzes the results of the XACT-Performance design created earlier in this tutorial. The routed LCA file input to XDelay contains actual timing delays as well as XACT-Performance specifications. XDelay analyzes this information and shows different types of delay paths according to the options you select. Selected options are stored as an XDelay template file with an .xtm extension.

This section demonstrates a typical XDelay analysis command sequence.

Note: The sample XDelay output in this tutorial is from a single Calc LCA file, targeted to the XC3020APC68-7. Your results will vary.

Invoking XDelay

You can invoke XDelay from the operating system prompt, from the Verify menu in XDM, or from the EditLCA program under the XACT Design Editor (XDE), which is not available in the Base Development System.

1. Invoke XDelay from the operating system prompt by typing **`xdelay`**↵.

   The XDelay graphic environment appears on the screen.

2. Load the CALC design into memory by selecting **`Design`** → **`Design`** → **`CALC.LCA`**.

Using the Flagblk Option

In the Calc design, there are a number of flip-flops with asynchronous reset signals. You want to ignore the paths through these asynchronous inputs during timing analysis. (Normally a designer is unconcerned with asynchronous reset paths when considering clock-to-setup requirements.) The Flagblk option is useful for specifying that this type of path be ignored.

PPR does not trace paths through the Set and Reset (SD/RD) pins of flip-flops, so the timing results in the timing analysis summary of the ppr.log file do not include any paths through the SD/RD pins.

In order to verify the timing results from the ppr.log file, you must disable tracing paths through the SD/RD pins. As a result, XDelay does not trace through the above-mentioned paths when calculating path delays, which permits a valid comparison of the timing results from the PPR log file and the XDelay output.

Disabling Paths Through SD/RD Pins of Flip-Flops

Disable these paths with the Flagblk option as follows.

1. From the menu, select **`Timing`** → **`Flagblk`** → **`CLB_Disable_SR_Q`**.

   A menu appears, displaying a list of all CLB blocks in the design.

2. Type **`*`**↵ to select all blocks.

A prompt appears, asking whether you are sure you want to select all of the blocks.

3. Select **Yes → Done**.

   This command disallows paths from a CLB Asynchronous Set or Reset input to the Q output of the flip-flop.

Displaying Current Options

The Query Template command displays the current settings of all options. You can save a template to a file using the Save Template command, load a customized template using the Read Template command, or clear the current template with the Clear Template command.

1. Select **Timing → QueryTemplate** to view the current XDelay options.
2. After viewing the template listing, press any key to return to the XDelay executive screen.

The current template appears on the screen. The template includes all restrictions you applied to each CLB. A partial template for an example XC3000A design is shown in Figure 15-11.

```
XDelay -NoSourceClock
XDelay -NoDestClock
XDelay -NoIgnorenet
XDelay -NoNetfilter
XDelay -NoBreakLoop

Flagblk CLB_Disable_SR_Q ALU/DATA0
Flagblk CLB_Disable_SR_Q STACK2
.
.
Flagblk CLB_Disable_SR_Q ALU/ENOV
Flagblk CLB_Disable_SR_Q CTL2
```

Figure 15-11 Partial Template for an XC3020APC86-7 Design

Using Analyze Mode

Now that you have restricted XDelay to considering only clocked paths, perform a quick analysis to determine the worst-case timing for the Calc design.

By default, XDelay output is written to the screen. Normally, you want to keep a copy of the analysis results. Also, writing to a file is much faster than reporting to the screen, since there is no need to funnel information through a graphical interface.

1. Select **Misc → Report** to specify that you want the results written to a file.

   You are prompted to enter a file name.

2. Type ↵ to accept the default file name, calc.xrp.

3. Select **Analyze → Done**.

For the example 3020APC68 design, XDelay reports the same combinatorial logic loop detected by PPR. Since this loop is deliberately included to create an oscillator, ignore this message.

Examining Analyze Output

Examine the Analyze output file.

1. Press any key to return to the XDelay graphic screen.

2. Use any text editor to examine the XDelay report file, calc.xrp.

A partial report file for the XC3020APC68-7 example design is shown in Figure 15-12. The last line of the file shows that the design will operate at approximately 12.7 megahertz under worst-case conditions.

The XRP file lists the worst case delays for each of pad-to-pad, pad-to-setup, clock-to-pad, and clock-to-setup. It also provides an estimate of the minimum clock period and maximum clock speed for the input design.

```
Warning: Combinational logic loop(s) have been detected.
These may cause subtle design problems, and may yield some
inaccurate path delays. For a detailed enumeration of these
loops, use the “DRC -Inform” command from within XDE/EditLCA.

XDelay Report File:

Worst case Pad to Pad path delay   : 37.8ns (1 block level)
 Pad “OSC_3K/CQ” (P14) to Pad “OSC_3K/CQL” (P12.T)

Clock net “CLK” path delays:

Pad to Setup                       : 14.0ns (0 block levels)
 (Includes an external input margin of 0.0ns.)
```

```
 Pad to Input FF Setup, Pad “SW7/SW0_P” (P24).
 Target InFF drives output net “SW0”

Clock to Pad                        : 48.3ns (2 block levels)
 (Includes an external output margin of 0.0ns.)
 Clock to Q, net “ADDR0” to Pad “LED/LED1_P” (P30.O)

Clock to Setup (same edge)          : 78.6ns (6 block levels)
 Clock to Q, net “ADDR0” to FF Setup (D) at “ALU2.A”
 Target FFX drives output net “ALU2”

Minimum Clock Period                : 78.6ns
Estimated Maximum Clock Speed       : 12.7Mhz
```

Figure 15-12 Analyze Output for an XC3020APC68-7 Design

Using XDelay-TimeSpec Mode

Use the XDelay-TimeSpec mode to evaluate the timing of your design with respect to the XACT-Performance attributes that you added to your schematic.

The FailedSpec and SelectSpec options are particularly useful for evaluating XACT-Performance results. FailedSpec reports all path delays that do not meet timing specifications, and SelectSpec allows you to select which XACT-Performance specifications you wish to be considered.

Note: Used without the FailedSpec option, the SelectSpec option reports the worst paths for each XACT-Performance specification.

As in the “Using the Analyze Mode” section, create a written report file.

1. Select **Misc → Report**.
2. Type **calcts.xrp**↵ to distinguish the new output file from the previous one.
3. Select **XDelay-TimeSpec**.

   A menu appears displaying available delay options. The default options are highlighted.
4. Select **ClearOptions** to remove any delay options you may have previously selected.

   The Flagblk options you set earlier in the tutorial are not cleared, because they are template options rather than delay options.

5. Select the **-FailedSpec** option.
6. The **SelectSpec** option is already on by default, but select it anyway.

   A menu of defined XACT-Performance attributes appears. You can select any or all of the timing specifications on this list. By default, all XACT-Performance specifications are already selected.
7. Select **Cancel** to accept the list with all entries selected.

   Next, set the maximum number of paths shown for each failed XACT-Performance constraint to be three.
8. Select **-TSMaxpaths**.
9. Type **3**↵.

Note: If you do not set the TSMaxpaths option, the report file lists delays for every path controlled by each XACT-Performance specification in your design. This may cause XDelay to run out of memory; if not, it produces a very large output file.

10. Select **Done** to initiate the timing analysis.

As before, for the example 3020APC68 design XDelay reports the same combinatorial logic loop detected by PPR. Since this loop is deliberately included to create an oscillator, ignore this message.

Examining XDelay-TimeSpec Output

The XDelay-TimeSpec output file contains a great deal of information.

1. Press any key to return to the XDelay graphic screen.
2. Use any text editor to examine the XDelay report file, calcts.xrp.

A portion of the resulting report file for the XC3020APC68-7 example design is shown in Figure 15-13.

The first section of the XRP file lists all XACT-Performance specifications applied to your design. If you do not specify any paths that fall into a given default path type, FFS:TO:FFS, PADS:TO:FFS, or FFS:TO:PADS, that default specification is set to “auto,” which means that PPR assigns some reasonable value as the timing specification.

The output file then lists the contents of each time group that you defined using TNM attributes and TIMEGRP symbol attributes. This section can be useful in verifying your time group definitions.

For example, in Figure 15-13, the ALUFF time group contains the four ALU outputs and the OFL flip-flop output. Therefore, the group contains all of the flip-flops in the ALU, and no other flip-flops, just as expected.

The worst path delay is then reported for each XACT-Performance specification.

For TS05, which missed the target timing, the XRP file includes a detailed listing of the three slowest paths. You can use this listing to examine your critical paths and determine why each path is not routable with the current timing requirement, then take steps to remedy the situation.

For example, for the failed path shown in Figure 15-13, the longest delay on the path is an 8.9 ns delay between the Y output of CLB FE and the C input of CLB HH. (The block name of CLB FE is ADDR1, since the block is named after the X output, but the signal you are tracing is ADDR0.) Since these CLBs are some distance from each other, the net delay is significant. Compare this net delay to the net delay listed further down the path, between the X output of CD and the B input of CLB DD; the delay between these adjacent CLBs is only 1.9 ns. You might be able to speed up this path by using floorplanning techniques to place the logic within a smaller area.

A comparison of the results of the FailedSpec output in Figure 15-13 with the PPR log file in Figure 15-10 shows that the PPR and XDelay results vary by only a few tenths of nanoseconds. When there is a discrepancy between the two, the XDelay results are correct.

For example, the last XACT-Performance specification listing in Figure 15-13 is for TS08, FROM:ALUFF:TO:PADS=45. The FailedSpec output shows that the worst path delay is 40.0 ns. The ppr.log file in Figure 15-10 shows the worst path delay to be 40.1 ns.

```
XDelay: calc.lca (3020APC68-7), xdelay 5.0.0, Mon Jan 17
11:15:36 1994

Warning: Combinational logic loop(s) have been detected.
These may cause subtle design problems, and may yield some
inaccurate path delays. For a detailed enumeration of these
loops, use the “DRC -Inform” command from within XDE/EditLCA.

XDelay Report File:

Xdelay path report options:

TimeSpec ‘TSO3’ from group ‘FFGRP’ to group ‘PADS’ is
1000.0ns.
TimeSpec ‘TS01’ from group ‘FFS’ to group ‘FFS’ is 2000.0ns.
TimeSpec ‘TS02’ from group ‘PADS’ to group ‘FFS’ is 1000.0ns.
TimeSpec ‘TS04’ from group ‘INFFS’ to group ‘FFS’ is 80.0ns.
TimeSpec ‘TS05’ from group ‘CTL_ADR_FF’ to group ‘ALUFF’ is
50.0ns.
TimeSpec ‘TS06’ from group ‘CTL_ALU_FF’ to group ‘STACKER’ is
30.0ns.
TimeSpec ‘TS07’ from group ‘STACKER’ to group ‘LEDPADS’ is
50.0ns.
TimeSpec ‘TS08’ from group ‘ALUFF’ to group ‘PADS’ is 45.0ns.

TimeGroup ‘ALUFF’ contains these Flip-Flop output nets:
ALU0 ALU1 ALU2 ALU3 OFL

TimeGroup ‘CTL_ADR_FF’ contains these Flip-Flop output nets:
ADDR0 ADDR1

TimeGroup ‘CTL_ALU_FF’ contains these Flip-Flop output nets:
ADDR0 ALU0 ALU2 CONTROL/STATMACH/OTHER OFL
ADDR1 ALU1 ALU3 CONTROL/STATMACH/PUSH WE
.
.
.
TimeGroup ‘STACKER’ contains these Flip-Flop output nets:
STACK/A0 STACK/A3 STACK/B2 STACK/C1 STACK/D0 STACK/D3
STACK/A1 STACK/B0 STACK/B3 STACK/C2 STACK/D1
STACK/A2_1 STACK/B1 STACK/C0 STACK/C3 STACK/D2

Only paths that do not meet the selected TimeSpecs will be
reported.
Output will be sorted by decreasing path delays.
A maximum of 3 paths will be reported for each TimeSpec.

------------------------------------------------------------

TimeSpec ‘TSO3’ summary:
From TimeGroup ‘FFGRP’
To TimeGroup ‘PADS’

TimeSpec limit is          : 1000.0ns (Spec speed = 1.0MHz)
```

```
Worst path delay is        : 48.3ns   (Real speed = 20.7MHz)

TimeSpec passes by : 951.7ns

List of delay paths that fail the TimeSpec:
There are no paths that fail the TimeSpec.

------------------------------------------------------------

TimeSpec 'TS01' summary:
From TimeGroup 'FFS'
To TimeGroup 'FFS'

TimeSpec limit is          : 2000.0ns (Spec speed = 0.5MHz)
Worst path delay is        : 78.6ns   (Real speed = 12.7MHz)

TimeSpec passes by : 1921.4ns

List of delay paths that fail the TimeSpec:
There are no paths that fail the TimeSpec.

------------------------------------------------------------

TimeSpec 'TS02' summary:
From TimeGroup 'PADS'
To TimeGroup 'FFS'

TimeSpec limit is          : 1000.0ns (Spec speed = 1.0MHz)
Worst path delay is        : 29.4ns   (Real speed = 34.0MHz)

TimeSpec passes by : 934.6ns

List of delay paths that fail the TimeSpec:
There are no paths that fail the TimeSpec.

------------------------------------------------------------

TimeSpec 'TS04' summary:
From TimeGroup 'INFFS'
To TimeGroup 'FFS'

TimeSpec limit is          : 80.0ns   (Spec speed = 12.5MHz)
Worst path delay is        : 65.4ns   (Real speed = 15.3MHz)

TimeSpec passes by : 14.6ns

List of delay paths that fail the TimeSpec:
There are no paths that fail the TimeSpec.

------------------------------------------------------------

TimeSpec 'TS05' summary: *** TimeSpec FAILED! ***
From TimeGroup 'CTL_ADR_FF'
To TimeGroup 'ALUFF'

TimeSpec limit is          : 50.0ns   (Spec speed = 20.0MHz)
Worst path delay is        : 78.6ns   (Real speed = 12.7MHz)
```

```
*** TimeSpec FAILS by : 28.6ns ***

List of delay paths that fail the TimeSpec:

Logical Path                                Delay   Cumulative
------------                                -----   ----------

Source clock net : “CLK” (Rising edge)
From: Blk ADDR1          CLOCK to FE.Y    : 4.5ns ( 4.5ns)
Thru: Net ADDR0               to HH.C     : 8.9ns ( 13.4ns)
Thru: Blk STACK/$1I15/M01     to HH.X     : 5.1ns ( 18.5ns)
Thru: Net STACK/$1I15/M01     to HF.C     : 3.1ns ( 21.6ns)
Thru: Blk STACK0              to HF.X     : 5.6ns ( 27.2ns)
Thru: Net STACK0              to HB.A     : 4.7ns ( 31.9ns)
Thru: Blk ALU/DATA0           to HB.X     : 5.1ns ( 37.0ns)
Thru: Net ALU/DATA0           to ED.D     : 6.2ns ( 43.2ns)
Thru: Blk ALU/$1I308/C0       to ED.X     : 5.1ns ( 48.3ns)
Thru: Net ALU/$1I308/C0       to BE.E     : 4.5ns ( 52.8ns)
Thru: Blk ALU/$1I308/C1       to BE.X     : 5.1ns ( 57.9ns)
Thru: Net ALU/$1I308/C1       to CD.A     : 2.5ns ( 60.4ns)
Thru: Blk ALU/$1I308/C2       to CD.X     : 5.1ns ( 65.5ns)
Thru: Net ALU/$1I308/C2       to DD.B     : 1.9ns ( 67.4ns)
Thru: Blk LU/MUXBLK5/$1I5/M01 to DD.X     : 5.6ns ( 73.0ns)
Thru: Net LU/MUXBLK5/$1I5/M01 to DE.B     : 0.6ns ( 73.6ns)
To: FF Setup (D), Blk ALU3                : 5.0ns ( 78.6ns)

Target FFX drives output net “ALU3”
Dest clock net : “CLK” (Rising edge)
Clock delay to Source clock pin : 2.8 ns
Clock delay to Dest clock pin : 2.8 ns
Clock net “CLK”, delta clock delay [skew] : 0.0 ns

.
. (TWO MORE FAILED PATHS OMITTED)
.

------------------------------------------------------------

TimeSpec ‘TS06’ summary:
From TimeGroup ‘CTL_ALU_FF’
To TimeGroup ‘STACKER’

TimeSpec limit is          : 30.0ns   (Spec speed = 33.3MHz)
Worst path delay is        : 24.0ns   (Real speed = 41.8MHz)

TimeSpec passes by : 6.0ns

List of delay paths that fail the TimeSpec:
There are no paths that fail the TimeSpec.

------------------------------------------------------------

TimeSpec ‘TS07’ summary:
From TimeGroup ‘STACKER’
```

```
To TimeGroup 'LEDPADS'

TimeSpec limit is          : 50.0ns   (Spec speed = 20.0MHz)
Worst path delay is        : 43.4ns   (Real speed = 23.0MHz)

TimeSpec passes by : 6.6ns

List of delay paths that fail the TimeSpec:
There are no paths that fail the TimeSpec.

------------------------------------------------------------

TimeSpec 'TS08' summary:
From TimeGroup 'ALUFF'
To TimeGroup 'PADS'

TimeSpec limit is          : 45.0ns   (Spec speed = 22.2MHz)
Worst path delay is        : 40.0ns   (Real speed = 25.0MHz)

TimeSpec passes by : 5.0ns

List of delay paths that fail the TimeSpec:
There are no paths that fail the TimeSpec.

------------------------------------------------------------

Paths not used in TimeSpecs :

There are no paths in this section!

------------------------------------------------------------
```

Figure 15-13 XDelay-TimeSpec Output for XC3020APC68-7

Using XDelay Mode

In XDelay mode, you can generate reports either based on general path types or analyzing designated paths from specific sources to specific destinations.

Reporting by Path Type

In order to simplify analysis of designs without XACT-Performance specifications, XDelay has four options that restrict reporting to paths of four common types. These options are ClockToSetup, ClockToPad, PadToSetup, and PadToPad. You can select one or more of these options.

The number of paths reported depends on the value of the Maxpaths option.

Use the ClockToSetup option to report the single slowest path between two flip-flops clocked by the same edge of the clock.

1. Select **Misc → Report**.

   You are prompted to enter the name of the report file.

2. Type **calcc2s.xrp**↵.

3. Select **XDelay**.

   A menu appears displaying available delay options.

4. Select **ClearOptions** to remove any delay options you may have previously selected.

   The Flagblk options you set earlier in the tutorial are not cleared, because they are template options rather than delay options.

5. Select **-ClockToSetup**.

6. Select **-Maxpaths** and type **1**↵.

   Setting Maxpaths=1 directs XDelay to report only the worst clock-to-setup path in the design.

Note: If you do not set the Maxpaths option, the report file lists delays for every path in your design. This may cause XDelay to run out of memory; if not, it produces a very large output file.

7. Select **Done** and press any key to return to the XDelay graphic screen.

8. Use any text editor to examine the XDelay report file, calcc2s.xrp.

A portion of an example calcc2s.xrp file is shown in Figure 15-14.

The only paths reported are those between flip-flops: in other words, the paths that fall into the ClockToSetup or FROM:FFS:TO:FFS category. Since Maxpaths was set to one, only the worst-case clock-to-setup path is reported.

```
Output will be sorted by decreasing path delays.
Report file may include Clock To Setup paths.
A maximum of 1 path will be reported.

------------------------------------------------------------

Logical Path                                  Delay Cumulative
------------                                  ----- ----------

Source clock net : “CLK” (Rising edge)
From: Blk ADDR1      CLOCK   to FE.Y      : 4.5ns ( 4.5ns)
Thru: Net ADDR0              to HH.C      : 8.9ns ( 13.4ns)
Thru: Blk STACK/$1I15/M01    to HH.X      : 5.1ns ( 18.5ns)
Thru: Net STACK/$1I15/M01    to HF.C      : 3.1ns ( 21.6ns)
Thru: Blk STACK0             to HF.X      : 5.6ns ( 27.2ns)
Thru: Net STACK0             to HB.A      : 4.7ns ( 31.9ns)
Thru: Blk ALU/DATA0          to HB.X      : 5.1ns ( 37.0ns)
Thru: Net ALU/DATA0          to ED.D      : 6.2ns ( 43.2ns)
Thru: Blk ALU/$1I308/C0      to ED.X      : 5.1ns ( 48.3ns)
Thru: Net ALU/$1I308/C0      to BE.E      : 4.5ns ( 52.8ns)
Thru: Blk ALU/$1I308/C1      to BE.X      : 5.1ns ( 57.9ns)
Thru: Net ALU/$1I308/C1      to CD.A      : 2.5ns ( 60.4ns)
Thru: Blk ALU/$1I308/C2      to CD.X      : 5.1ns ( 65.5ns)
Thru: Net ALU/$1I308/C2      to DD.B      : 1.9ns ( 67.4ns)
Thru: Blk LU/MUXBLK5/$1I5/M01 to DD.X     : 5.6ns ( 73.0ns)
Thru: Net LU/MUXBLK5/$1I5/M01 to DE.B     : 0.6ns ( 73.6ns)
To: FF Setup (D), Blk ALU3                : 5.0ns ( 78.6ns)
Target FFX drives output net “ALU3”

Dest clock net : “CLK” (Rising edge)
Clock delay to Source clock pin : 2.8 ns
Clock delay to Dest clock pin : 2.8 ns
Clock net “CLK”, delta clock delay [skew] : 0.0 ns

------------------------------------------------------------
```

Figure 15-14 ClockToSetup Output for XC3020APC68-7 Design

Specifying Source and Destination

For a report of a specific path, use the FromFF and ToFF options in the XDelay mode. For instance, suppose you are concerned about the path delay between Delay1 and Delay2 in the debounce circuit of the Calc design. (These nets are the outputs of consecutive flip-flops. XDelay can not report path delays that span more than one flip-flop.)

Follow the steps below to get a detailed report of the delays on this path.

1. Select **Misc → Report**.

You are prompted to enter the name of the report file.

2. Type **dpath.xrp**↵.
3. Select **XDelay → -ClearOptions**.

   You are ready to enter the endpoints of the desired path.

4. Select **-FromFF**.

   A list of all flip-flops in the design appears on the screen. The flip-flops are identified by the name of the output net.

5. Select the source flip-flop by the name of the output net, **DEBOUNCE/DELAY1**.
6. Select **Done** to accept the list of source flip-flops.

   Alternatively, you could select additional flip-flops to get reports on more than one path.

7. Select **-ToFF**.
8. Select the destination flip-flop by the name of the output net, **DEBOUNCE/DELAY2**↵.
9. Select **Done**.

   As with the FromFF option, entering more names would enable you to obtain reports of multiple paths.

10. Select **Done** to initiate the analysis.

Note: In the EditLCA design editor, you can select the FromFF and ToFF flip-flops with the mouse. Alternatively, you can type the output net names or select the flip-flops from the menu.

11. Use any text editor to examine the XDelay report file, dpath.xrp.

The dpath.xrp file details a single path, the path between the Delay1 flip-flop and the Delay2 flip-flop, as shown in Figure 15-15. From the rising clock edge on the Delay1 flip-flop to the setup on the input pin of the Delay2 flip-flop, there is a maximum delay of 11.9 ns, under worst-case conditions. There is no detectable clock skew between the two flip-flops.

The From, FromIOB, FromAll, To, ToIOB, and ToAll options are all similar in usage to the FromFF and ToFF options demonstrated in this section. Refer to the "XDelay" section of the *XACT Reference Guide* for more details.

Note: When selecting nets in EditLCA for the From and To options, place the cursor on a pin, such as .X, .YQ, .F1, or .G2, that is connected to the desired net. Make sure the correct net name appears in the status window. Click the left mouse button to add the net to the list of selected nets.

```
.
.
.
From FF "DEBOUNCE/DELAY1"
To FF "DEBOUNCE/DELAY2"
Output will be sorted by decreasing path delays.

------------------------------------------------------------

Logical Path                                Delay   Cumulative
------------                                -----   ----------

Source clock net : "CLK" (Rising edge)
From:  Blk BOUNCE/DELAY1 CLOCK  to BA.X     : 4.5ns ( 4.5ns)
Thru:  Net DEBOUNCE/DELAY1      to BA.DI    : 3.4ns ( 7.9ns)
To:    FF Setup (D), Blk DEBOUNCE/DELAY1    : 4.0ns ( 11.9ns)
Target FFY drives output net "DEBOUNCE/DELAY2"

Dest clock net : "CLK" (Rising edge)
Clock delay to Source clock pin : 2.7 ns
Clock delay to Dest clock pin : 2.7 ns
Clock net "CLK", delta clock delay [skew] : 0.0 ns

------------------------------------------------------------
```

Figure 15-15 Dpath.xrp File for an XC3020APC68-7 Design

Further Reading

Before using XACT-Performance for your own designs, you should read the "XACT-Performance Utility" section of the *XACT Reference Guide*. More information on XDelay can be found in the "XDelay" section of the same manual.

OrCAD Interface/ Tutorial Guide

XEPLD Tutorial

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Chapter 16

XEPLD Tutorial

Welcome to the Xilinx EPLD OrCAD tutorial. This tutorial gives you a basic understanding of the XEPLD development software and some hands-on experience so you can start your first design as quickly as possible. This tutorial is not meant as a substitute for the *XEPLD Reference Guide* or your OrCAD manuals. For a summary of all commands used in this tutorial, refer to Appendix D.

Tutorial Guidelines

The tutorial consists of six sessions. Each session begins with an overview and summary of the interactive steps required. If you are not familiar with OrCAD, it takes about six hours to complete the entire tutorial. If you are familiar with OrCAD and you wish to skip the schematic capture and simulation sessions, you can finish in about an hour. Xilinx supplies example files, including a schematic of the tutorial design.

To end your XEPLD session at any time, select Quit from the XDM (Xilinx Design Manager) menu. To quit OrCAD and return to XDM at any time, go back to the main menu and select the Exit ESP command.

To remove an OrCAD menu from the screen without executing a command, or to cancel an OrCAD command before it is executed, press the Escape (Esc) key or click on the right mouse button. The Draft menu also has an Undo command for undoing deletions.

Tutorial Design

In this tutorial, you design the receiver section of a Universal Asynchronous Receiver Transmitter (UART). This circuit converts a serial data stream to parallel bytes and provides handshaking and error detection signals to the host system. Figure 16-1 illustrates the functionality of this design.

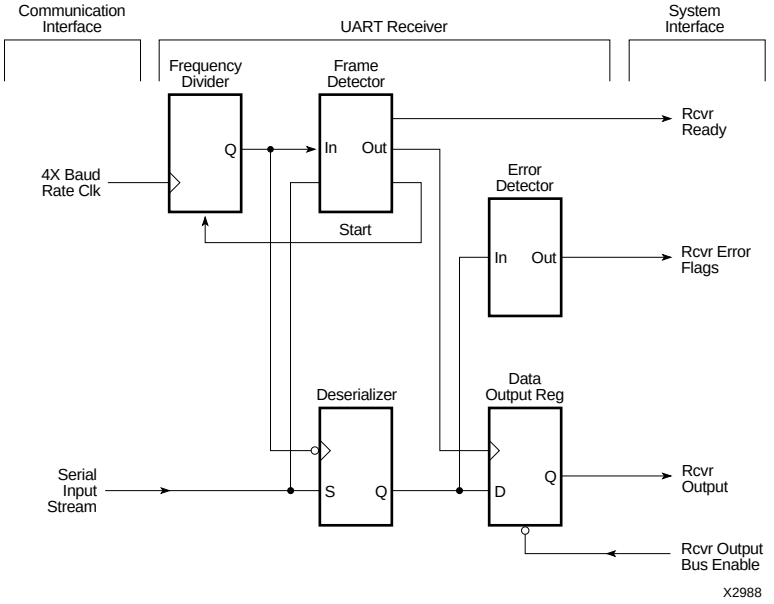


Figure 16-1 UART Receiver Functional Logic Diagram

The example design functions as follows:

1. A serial to parallel shift register (Deserializer) converts the serial stream to parallel data, which is then latched in a register (Data Output Reg).
2. A simple state machine (Frame Detector) controls the receiver. Once the start bit is detected, the counter (Frequency Divider) begins to count sequentially, clocked by the 4X Baud Rate Clk.

3. The host is notified with the ready signal (Rcvr Ready) and reads the data by asserting the Rcvr Output Bus Enable signal.
4. The output of the counter is decoded to generate the control signals for the shift register, data latch, and error detection circuits. The following signals are generated:
   - Parity Error is generated if a byte parity is odd.
   - Framing Error is generated if any of the stop bits are low.
   - Overrun Error is generated if new data is ready to be latched into the output register before the CPU reads the previous data.

   Figure 16-2 shows the format of the serial input stream.

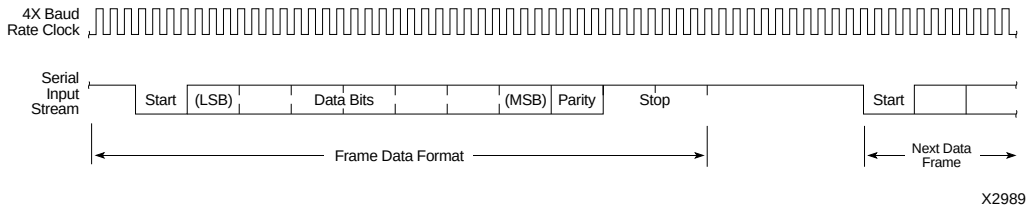


Figure 16-2 UART Serial Input Waveform

Tutorial Files

All the files used in the UART tutorial are installed into a sample design directory named tutorial\orcad\uart beneath your designated XACT software directory. The default path is c:\xact\tutorial\orcad\uart. The design in the uart directory contains a PAL symbol and its associated equation file. A second directory, c:\xact\tutorial\orcad\uarttop, contains an alternative implementation based purely on schematic library symbols. The file names in these directories are listed in Table 16-1. These directories also contain some of the resulting report files.

Table 16-1 Tutorial Files

Directory xact\tutorial\orcad\uart	
uart.sch	Schematic
rcvr.pld	PLD Equations
uart.stm	Simulation Stimulus File
uart.trc	Simulation Trace File
Directory xact\tutorial\orcad\uarttop	
uarttop.sch	Schematic
rcvrsub.sch	Schematic
uarttop.stm	Simulation Stimulus File
uarttop.trc	Simulation Trace File

The tutorial files are optionally installed during the installation of the OrCADInterface and Libraries software. If these files do not exist, refer to the library installation instructions in the *XACT Installation Guide* for directions.

Overview of the Sessions

The six sessions cover the following topics:

Session 1: Using the XEPLD Software

Session 2: Drawing the Design in Draft

Session 3: Defining PLD Equations

Session 4: Fitting the Design

Session 5: Simulating the Design

Session 6: Functionally Simulating a Purely Schematic Design

If you are unfamiliar with OrCAD, you should work through the entire tutorial.

If you are already familiar with OrCAD and want to focus on learning the XEPLD software, perform Sessions 1, 3, and 4 of this tutorial. Also review Steps 1 through 4 of Session 2 to view the design and Step 1 of Session 5 to see how XDM interfaces with OrCAD VST.

Session 1: Using the XEPLD Software

Session 1 concentrates on the XEPLD environment. The following operations are introduced in this session:

Step 1: Preparing the System

Step 2: Starting XDM

Step 3: Selecting Menu Items in XDM

Step 4: Configuring the XEPLD Environment

Step 1: Preparing the System

From the XDM menu system, you can access the XEPLD software. Before you start XDM, you must have installed the XEPLD software and the OrCAD libraries. Refer to the installation instructions in the *XACT Installation Guide* for information on how to install these products.

In addition, verify the following DOS environment configurations:

1. The XACT software directory containing XDM and the OrCAD software directory must be included in your path.

   You must set the OrCAD variables and the XACT variable before performing any other preparatory steps.

2. All OrCAD environment variables must be set according to OrCAD requirements.

   Take note of the directory to which the ORCADPROJ variable points, because this is the directory in which you store your design directories, including the directory for the tutorial design.

Note: This tutorial assumes that your ORCADPROJ variable is set to c:\orcad\. You need not follow this convention.

3. The XACT variable must be set according to XACT installation requirements.

Note: This tutorial assumes that your XACT variable is set to c:\xact. You need not follow this convention.

4. To specify a text editor to be used under XDM, set the EDITOR environment variable.

Step 2: Starting XDM

1. Invoke XDM by typing **xdm**↵ at the DOS prompt.

   The XDM menu is displayed.

You can access your preferred development tools for schematic capture, text editing, and programmer control through the XDM menu interface (Figure 16-3). For detailed information regarding the menu items, refer to the "XACT Design Manager" chapter of the *XEPLD Reference Guide*.

Note: To end your XDM session at any time, select Quit from the menu.

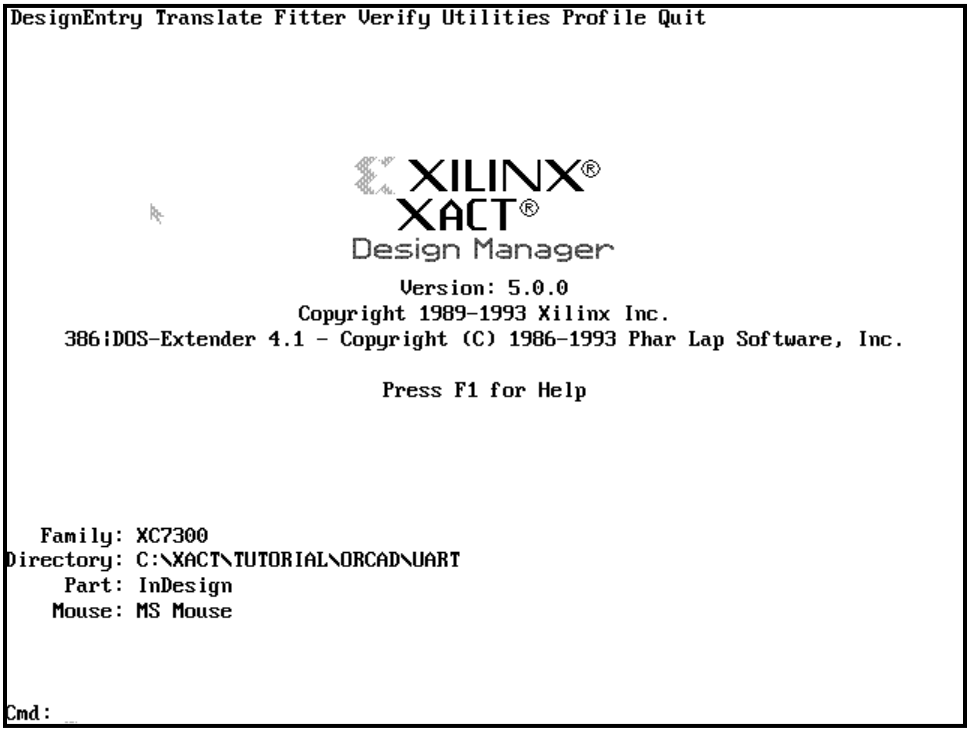


Figure 16-3 XDM Menu

Step 3: Selecting Menu Items in XDM

You can use either the mouse or the command line to select menu items.

Using the Mouse

You can use the mouse to open the menu and click on the command. The commands in this tutorial assume that the mouse is set to the default button configuration.

- Left button: Select
- Middle button: Done
- Right button: Cancel

You can also use the arrow keys to move up and down in the menus and the Enter key to select a menu item.

Typing Commands

Alternatively, you can type the command and any specified options from XDM. Your entry appears on the XDM command line, indicated by the `Cmd:` prompt at the bottom of the screen

Accessing DOS

To open a menu, position the cursor on the appropriate menu item and click on the left mouse button. You can also cancel the last command selected by pointing anywhere on the screen except on the menu titles and pressing the right mouse button. For example:

1. Move the mouse cursor over the **`Utilities`** menu item and click the left mouse button. Then move the mouse over the **`DOS`** command and click the left mouse button. This invokes the DOS command.
2. XDM opens up a DOS shell and lets you execute DOS commands or run utility programs. Type **`exit`**↵ at the DOS prompt to return to the XDM menu.

Each command has a few characters highlighted. The highlighted characters indicate the command abbreviations that you can also use to invoke the commands at the command line.

Responding to XDM Prompts and Menus

If you select a command and do not get the expected result, check the following:

- Some commands prompt you at the command line for more information, so you should always be aware of when the command line prompt changes. For example, when you select Utilities → Directory, the `Directory:` prompt appears at the command line, and you must type a directory name if none of the menu choices correspond to the desired directory.
- Other commands display a submenu with their own set of commands above the submenu. If you select an item on a submenu and nothing happens, select one of the commands above the menu, for example, Done.

Step 4: Configuring the XEPLD Environment

First, select the family and part type in XDM.

1. Select **Family** to specify the desired Xilinx device.
2. Select **XC7300** to bring up the menu for the EPLD devices. A submenu of parts appears.
3. Select **InDesign**.

You can customize the XDM interface by defining function keys using the Keydef command. For example, to invoke an external text editor with the push of a button, program one of the function keys using the Profile Keydef command. To use the DOS EDIT editor, enter the following on the XDM command line:

```
Cmd: keydef f2 dos edit\↵
```

F2 is now programmed to invoke EDIT.

The backslash (\) causes XDM to prompt you to finish entering the command when you press the defined key. You can use the backslash with any command that requires a variable argument at the end of its syntax. For example, if you define the F2 key as shown above and then press F2, you can answer the prompt with a file name as follows:

```
Cmd: dos edit uart.pin↵
```

You use a text editor with XDM primarily for preparing Boolean equation (behavioral entry) files for the PLUSASM equation assembler, and for viewing the reports generated by XEPLD. PLUSASM requires that text files contain only ASCII characters. If you use a word processor, it must save files as unformatted ASCII text.

Note: Do not change the file extensions assigned by XDM and OrCAD, as these applications do not recognize or process files with incorrect or unfamiliar extensions.

Session 2: Drawing the Design in Draft

In this session, you learn how to draw OrCAD schematics using Draft, OrCAD's schematic design entry package. Draft contains a large repertoire of features. To effectively use the software, you only need to master a few basic operations covered in this session.

Step 1: Creating a New Design

Step 2: Opening and Viewing the Design

Step 3: Changing the Zoom Level

Step 4: Creating a New Schematic

Step 5: Entering and Arranging Components

Step 6: Creating Wires

Step 7: Adding Junctions

Step 8: Labeling Components

Step 9: Labeling Wires

Step 10: Assigning Attributes

Step 11: Finishing the Drawing

Step 12: Assigning Signals to Specific Pins

Step 13: Saving the Design

Step 14: Exiting OrCAD

Step 1: Creating a New Design

Before beginning a new design, you must create a new design directory.

Creating the Design Directory

To create a design directory, follow these steps:

1. Select **DesignEntry → ORCAD**.

   The OrCAD ESP main menu appears.

2. Select **Design Management Tools → Execute**.

A dialog box appears with directories listed on the left, design files listed on the right, and a group of buttons at the bottom.

3. The TEMPLATE directory is normally highlighted. If it is not highlighted, select the **TEMPLATE** directory from the list on the left.
4. Click on the **Create Design** button.
5. The Create Design dialog box appears. Click on the **New Design Name** box.
6. When a small white square appears in the box, type **uart**↵.
7. Click on the **OK** button. The new UART directory is selected in the directory listing on the left.

Copying the Design Files

The complete UART design provided with the Xilinx OrCAD library is installed under the \xact directory. Copy the files into your new design directory as follows:

1. While in the Design Management Tools dialog box, click on **Suspend to System**.

   A DOS shell opens with the new design directory as your current directory.

2. Copy the entire contents of the tutorial directory, \xact\tutorial\orcad\uart, into your new design directory, \orcad\uart.

   copy c:\xact\tutorial\orcad\uart↵

Configuring the UART Directory

Before beginning a Xilinx EPLD design, you must use the XDraft program to configure the design directory to access the \xact\xc7000 libraries.

1. At the DOS prompt, type **xdraft 7**↵.
2. Type **exit**↵ to return to OrCAD.
3. Click on the **OK** button to return to the main menu.

The dialog box disappears and you are back in the main menu.

Step 2: Opening and Viewing the Design

Open the UART schematic in Draft, the OrCAD schematic editor.

Opening the Design

1. Select **`Schematic Design Tools → Execute`**.
2. Select **`Draft → Execute`**.

The UART schematic appears, as shown in Figure 16-4.

Note: If the UART schematic does not appear, select Quit → Initialize and type uart↵ to open the schematic.

Examine the UART schematic. Note the PLD library symbol, the other standard components, such as latches and shift registers, and the input and output ports. The PLD library symbol is especially significant because it requires that you use an equation file to define its internal logic. Session 3 explains how to use equation files in your Xilinx EPLD designs.

Note: To quit schematic entry and return to the main menu at any time, select Quit → Abandon Edits.

Using the Mouse in OrCAD

In OrCAD, the mouse buttons perform the following functions:

- Left Button: Select
- Right Button: Cancel (same as Escape key)

Selecting from the SDT Menus

The commands for editing the schematic are organized in a series of menus. You can select a menu command in one of two ways:

- Click the left mouse button until the command you want appears on a menu, then select the command.

  The placement of the cursor on the schematic is not affected when you select a command. The cursor freezes and the mouse only affects command selection.

- Press the key that corresponds to the first letter of the command name. If you know the name of the desired command, this is often the easier method.

To cancel a command and return to the previous menu, click the right mouse button or press the Escape (Esc) key.

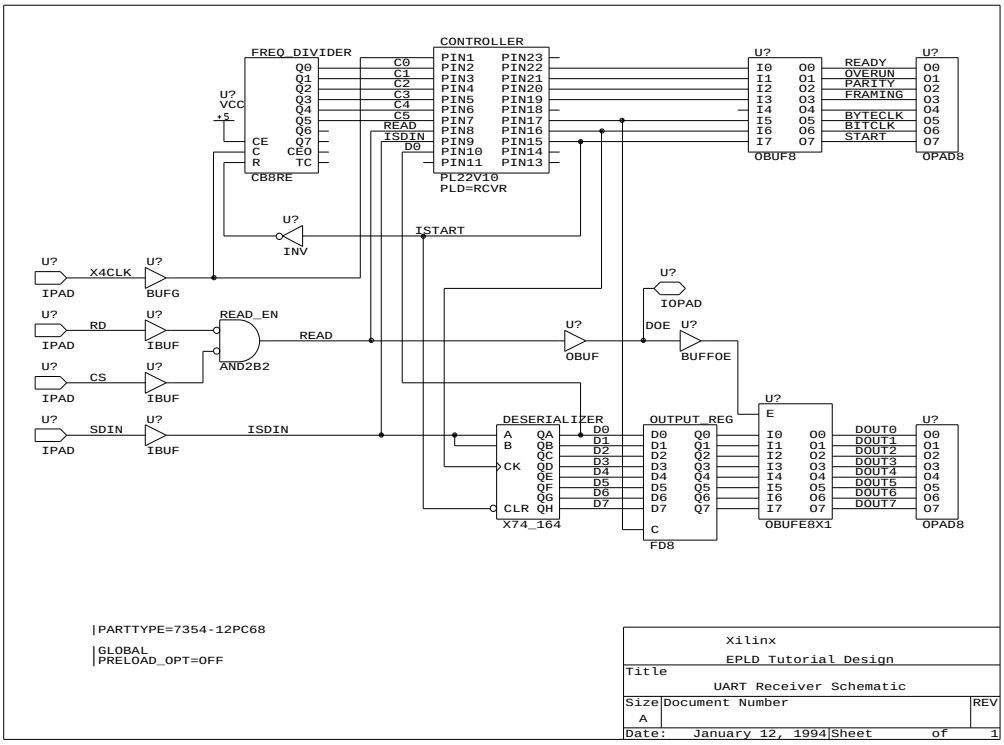


Figure 16-4 The Complete UART Schematic

Step 3: Changing the Zoom Level

Draft provides several Zoom commands that change the magnification level of the design.

1. Select the Zoom command either by clicking with the left mouse button to display the menu and selecting **Zoom**, or simply by typing **z**.

The following commands are displayed:

- Center — Centers the design at the current magnification level, which is listed in parentheses.
- In — Displays the design at the next larger magnification level, listed in parentheses, to show more detail.
- Out — Displays the design at the next smaller magnification level, listed in parentheses, to show more of the design.
- Select — Allows you to select a magnification level. The choices, from largest (most detail) to smallest (widest area), are 1, 2, 5, 10, and 20.

The design is displayed at zoom level 2 when you first view it.

2. Select **`In`**.

   The part of the schematic under the cursor is displayed in more detail. Some of the labels that were not previously readable are now readable.

   If the design is too large to fit on the screen at the current level of magnification, you can view another part of the design by moving the mouse cursor off the edge of the screen.

3. View the entire design by panning with the mouse.

Step 4: Creating a New Schematic

Create a blank OrCAD schematic.

Skipping Schematic Entry

If you are already familiar with OrCAD SDT and do not wish to draw the schematic, perform these steps:

1. Select **`Quit`** → **`Abandon Edits`** to close the UART schematic and return to the Schematic Design Tools menu.
2. Exit OrCAD and skip to Session 3.

Creating a New Schematic

To continue with the schematic capture session of this tutorial, perform these steps:

1. Click with the left mouse button to display the menu and then select **Quit → Initialize**, or simply type **q i**.

   This command exits the current schematic and prompts you for a new schematic.

2. Type **uart2**↵ in response to the prompt:

   A blank schematic appears, with the message <<<New Worksheet>>> in the top left corner. This schematic is named uart2.sch.

Step 5: Entering and Arranging Components

Enter components in the schematic and arrange them as shown in Figure 16-5.

Entering Components

To place IBUF components on the schematic, follow these steps:

1. Click with the left mouse button and select **Get**, or simply type **g**.

   The Get? prompt appears.

2. Type **ibuf**↵.

   The image of an IBUF component appears.

Note: Alternatively, you can browse a list of all components available from the XC7000 library. Press the Enter key in response to the Get? prompt and select XC7000.LIB from the menu. Move the mouse through the list of component names and select IBUF.

3. Move the mouse until the component image is in the lower left corner of the screen.

4. Select the **Place** command from the Get menu.

   The component is placed, and a second image appears on top of it.

5. Move this second image an inch above the first.

6. Select **Place** again.

   A third image appears.

7. Move the image an inch above the second IBUF.

8. Select **Place** again.

A fourth image appears.

9. Click the right mouse button or press the **Escape** key to cancel the Get command.

   The fourth image and the Get menu both disappear.

Arranging Components

After adding components, you must be able to move them. To move a component, follow these steps:

1. Select **Block → Move** from the menu.
2. Put the cursor near the top IBUF component.
3. Select the **Begin** command to initiate the selection of objects to be moved.
4. Move the cursor.

   The first location of the cursor corresponds to one corner of the selection frame box, and the current location is the opposite corner.

5. Move the corner of the selection frame box until the box traces the location of the top IBUF component.

Note: When you select a component or a group of components, any component that is partially in the box is included in the selection.

6. Select **End**.

   An image of the component appears.

7. Move the mouse to move the component image.
8. When the image is where you want it, select the **Place** command.

   The component disappears from its original location and appears where you placed the image.

Deleting Components

You must also be able to delete components. To delete a component, follow these steps:

1. Select **Delete → Object**.

2. Move the cursor over the top IBUF component, the one you moved in the last exercise.
3. Select the **Delete** submenu command.

   The component disappears.
4. Click the right mouse button or press **Escape** to end the Delete command.

Note: If you delete something by mistake, click the right mouse button or press Escape, then use the Delete Undo command.

Placing Rotated Components

Occasionally, you might need to rotate components. To place a rotated component, follow these steps:

1. Select **Get** and type **inv**↵ at the prompt.
2. Select **Rotate** twice on the Get menu.

   The INV component is now backwards.
3. Select **Place** on the Get menu to place the inverter as shown in Figure 16-5.
4. Click the right mouse button or press **Escape** to exit the Get command.

Entering Additional Components

Place the remaining components on the schematic.

1. Use the Get command to add the following components: PL22V10, CB8RE, BUFG, IBUF, IPAD, and AND2B2.

   The BUFG component represents a FastCLK input.

   Leave some space between the IPADs and the IBUFs. In Step 9, "Labeling Wires," you will add a label to each wire to name the EPLD device pins.
2. Use the Get command to add the VCC symbol. Do not use the Add Power command to add this symbol.

When you have finished arranging the components, the screen should look like Figure 16-5. If necessary, move the components using the Block Move command.

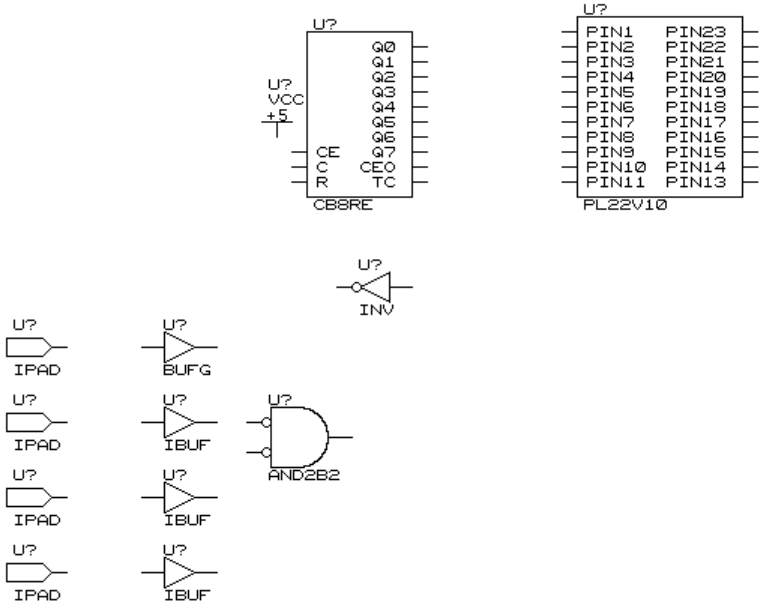


Figure 16-5 Adding Components

Step 6: Creating Wires

Connect your symbols by adding wires to your schematic.

Drawing a Wire

To create a signal between the Q0 output pin of the CB8RE and pin 2 of the PL22V10, follow these steps:

1. Select **Place → Wire**.
2. Move the cursor to the end of the Q0 pin of the CB8RE.
3. Select **Begin**.
4. Move the cursor to pin 2 of the PL22V10.
5. Select **End**.

Warning: When drawing a wire, be careful to begin and end the wire on the exact grid point where a component pin ends, otherwise OrCAD does not consider the wire to be connected.

To save time, you can use the Repeat command to draw wires that are parallel to and the same length as a wire you just drew. You might need to use the Set Repeat Parameters command to set your X Repeat step to 0 and your Y Repeat step to +1.

6. Select **Repeat** five times.

   Verify that pins Q0 through Q5 of the CB8RE now have wires going to pins 2 through 7 of the PL22V10.

Moving a Block

If the wires you just drew are not straight, you can use the Block Drag command to move one of the components while maintaining its connectivity .

To move the PL22V10 and maintain its connectivity, follow these steps:

1. Select **Block → Drag**.
2. Move the cursor to a point in or around the PL22V10.
3. Select **Begin**.
4. Move the box corner to another point in or around the PL22V10 so that no other component is in the box.
5. Select **End**.
6. Move the PL22V10 symbol to various locations.

   The wires change angle, stretch, and contract.

7. Move the PL22V10 symbol until the connecting wires are straight.
8. Select **Place**.

Drawing Wires Using Shortcuts

If you are drawing wires continuously and do not want to select the Place Wire command for each wire, you can use a few shortcuts.

To draw the wires between the IBUF components and the AND2B2 component, follow these steps:

1. Select **Place → Wire**.
2. Put the cursor on the output of the upper IBUF component and select **Begin**.
3. Move the cursor to the upper input of the AND2B2.
4. Select **New**.
5. Move the cursor to the output of the lower IBUF and select **Begin**.
6. Move the cursor so that it is even with the lower input of the AND2B2 and midway between the IBUFs and the AND2B2.
7. Select **Begin** again.
8. Move the cursor to the lower input of the AND2B2 and select **New**.
9. Continue to add wires until your design looks like Figure 16-6.
10. Click the right mouse button or press the **Escape** key to stop adding wires.

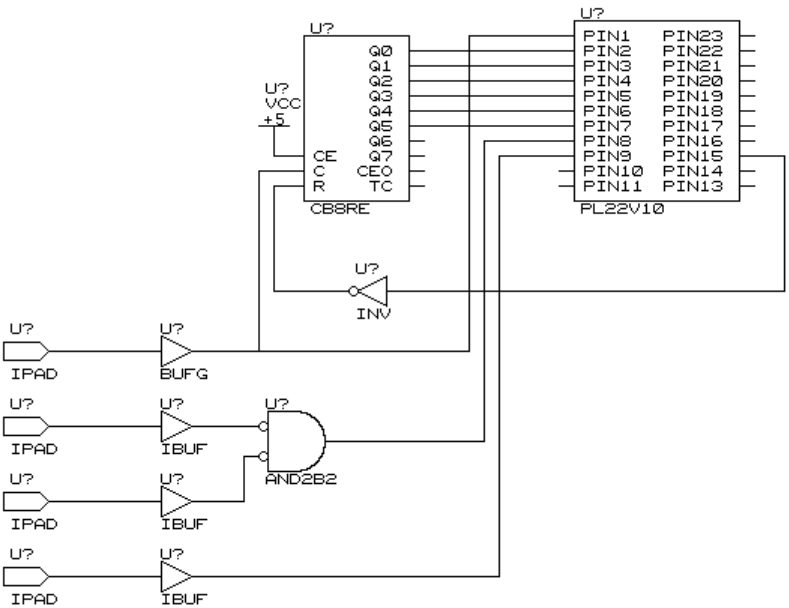


Figure 16-6 Adding Wires

Step 7: Adding Junctions

To add junctions where wires branch, follow these steps:

1. Select **Place → Junction**.
2. Move the cursor over the point where the wires meet.
3. Select **Place**.
4. Click the right mouse button or press the **Escape** key.

   Your design should look like Figure 16-7.

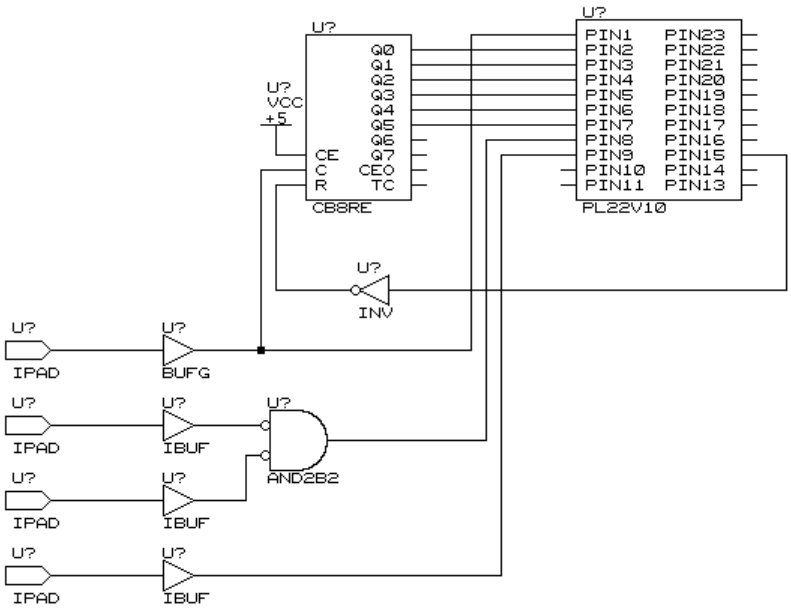


Figure 16-7 Adding Junctions

Step 8: Labeling Components

The reports generated by the XEPLD Fitter refer to all your on-chip signals using names based on your component labels, also called reference designators, and component output pin names.

For example, the macrocell logic and the signal produced by the least-significant bit of the CB8RE counter is referred to in the reports using the name FREQ_DIVIDER:Q0. Assigning your own unique labels makes it much easier to identify elements of your design in the reports.

Labeling the PL22V10 Component

To label the PL22V10 component, follow the steps outlined below:

1. Place the cursor on the PL22V10 component.
2. Select **Edit → Edit → Reference → Name**.
3. Backspace over the U? that follows the Reference? prompt.
4. Type **Controller**↵.
5. Click the right mouse button or press **Escape**.

You are still in the Edit menu.

Labeling the CB8RE Component

To label the CB8RE component, follow the steps outlined below:

1. Move the cursor to the CB8RE component.
2. Select **Edit → Reference → Name**.
3. Backspace over the U? that follows the Reference? prompt.
4. Type **Freq_Divider**↵.
5. Click the right mouse button or press **Escape**.

You are still in the Edit menu.

Labeling the AND2B2 Component

To label the AND2B2 component, follow the steps outlined below:

1. Move the cursor to the AND2B2 component.
2. Select **Edit → Reference → Name**.
3. Backspace over the U? that follows the Reference? prompt.
4. Type **Read_En**↵.
5. Click the right mouse button or press **Escape**.

When you are finished, your design should look like Figure 16-8.

6. Press the right mouse button or press **Escape** when you have reviewed the schematic.

Note: Use only alphanumeric characters and the underscore "_" character in labels.

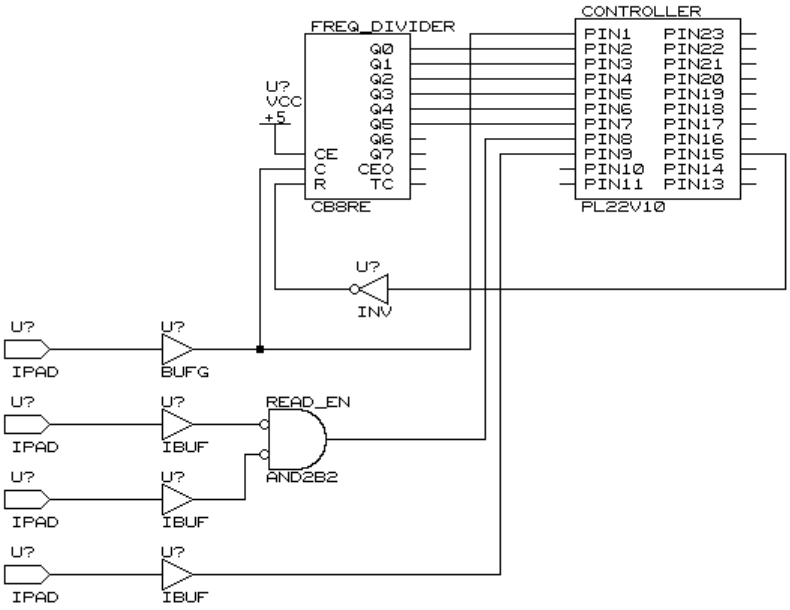


Figure 16-8 Adding Component Labels

Step 9: Labeling Wires

Names assigned to EPLD device pins in the Pinlist report and names on all signals used during simulation are taken from the labels you place on wires in your schematic. For example, the label X4CLK that you place on the wire between the IPAD and BUFG is listed as a pin name in the Pinlist report produced by the XEPLD Fitter. You will also apply your clock waveform to the X4CLK signal during functional and timing simulation.

To label the output wire of the AND2B2 component, follow these steps:

1. Select **Place → Label**.

   The Label? prompt appears.

2. Type **READ**↵.

   The label appears, attached to the cursor.

3. Move the cursor so that the tip touches the wire connected to the output of the AND2B2 component. Center the label on the segment just to the right of the component.

4. Select **Place**.

   The Label? prompt reappears.

5. Repeat steps 2 through 4 and place labels C0 through C5, X4CLK, RD, CS, SDIN, START, ISTART, and ISDIN as shown in Figure 16-9.

6. Click the right mouse button or press **Escape** to stop adding labels.

Note: Use only alphanumeric characters and the underscore "_" character in labels.

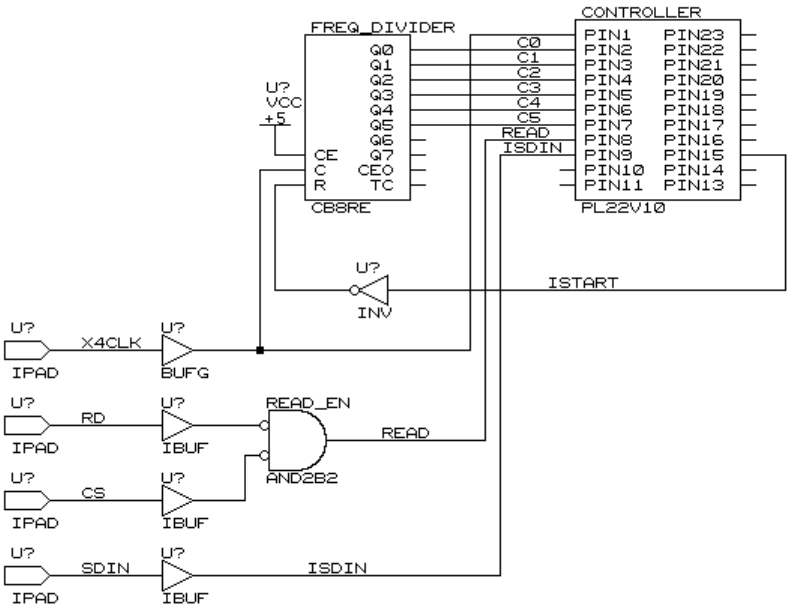


Figure 16-9 Adding Wire Labels

Step 10: Assigning Attributes

The PLD=*filename* component attribute must be attached to each PLD in your schematic. This attribute identifies the bitmap file used to define the logic of the PLD. The bitmap file is prepared by the PLUSASM assembler. The preparation of the logic equations for the PLD is covered in Session 3.

Adding the PLD Attribute

To add the PLD attribute to the PL22V10 component, follow these steps:

1. Move the cursor over the PL22V10 component.
2. Select **`Edit → Edit → 1st Part Field → Name`**.

   The `1st Part Field?` prompt appears.

Note: If the first part field is used for something else at your site, pick any of the other seven part fields. The XEPLD Fitter checks all part fields and extracts all the XEPLD attributes that it finds.

3. Type **PLD=RCVR**↵.
4. Click the right mouse button twice or press **Escape** twice to exit the Edit command.

Adding the PARTTYPE Attribute

To add the PARTTYPE attribute to the PL22V10 component, follow these steps:

1. Select **Place** → **Text**.

   The `Text?` prompt appears.

2. Type **|PARTTYPE=7354-12PC68**↵.
3. Using the mouse, move the text to the blank area near the lower left corner of the design border. Refer to Figure 16-10.
4. Select **Place** to place the text.

Adding Global Attributes

You also need to assign the PRELOAD_OPT global attribute.

1. Use the Place Text command to add the following text to your schematic.

   |GLOBAL
   |PRELOAD_OPT=OFF

2. Press the right mouse button to exit the Place Text command.

   The initial pipe characters (|) must be aligned, with the GLOBAL text string first.

   The text strings for the attributes should appear as follows:

   |GLOBAL
   |PRELOAD_OPT=OFF

Warning: If you have INET v1.10 or 1.10 H, you must perform an additional step each time you add text to the schematic. This version of INET discards the first line of each group of pipe text after the first group of text encountered on the schematic. For each placement of text on the schematic, you must place an additional text line consisting of a single pipe character above the first line of the text, with the pipe characters vertically aligned. INET discards this pipe character and correctly reads the text in the next line. INET v1.08 did not display this behavior, and the problem should be fixed in any release subsequent to v1.10.

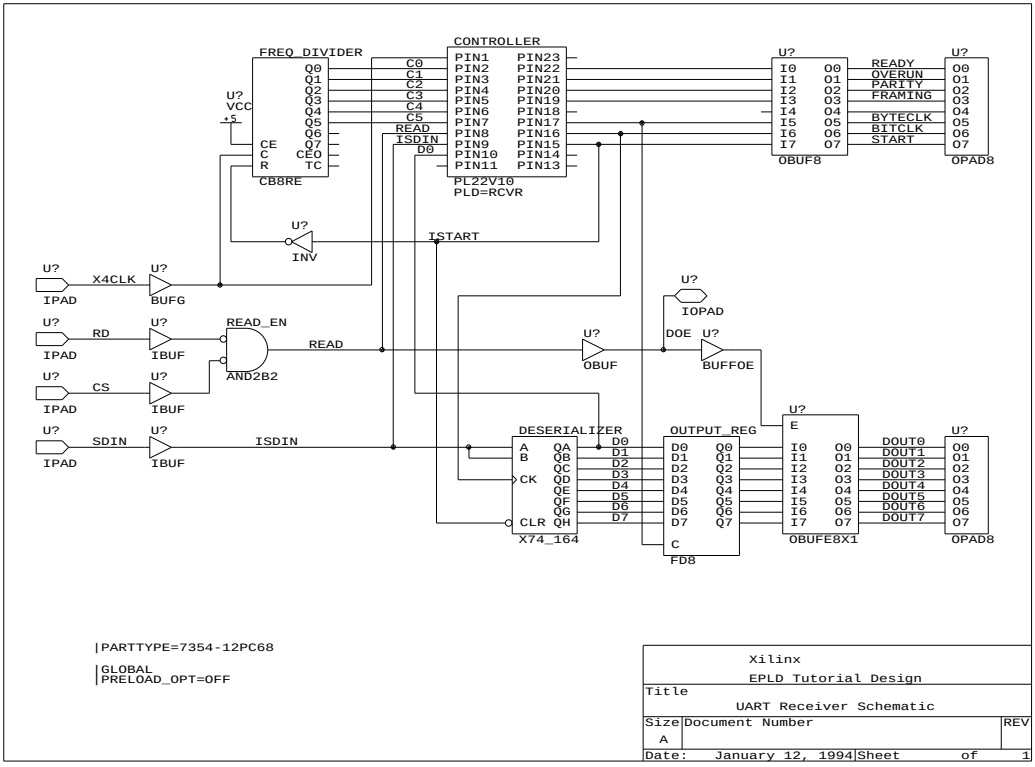


Figure 16-10 Completed UART Schematic

Step 11: Finishing the Drawing

You now know enough Draft commands to finish the schematic on your own. Figure 16-10 shows the completed UART receiver schematic.

1. Add the remaining components, wires, and labels as shown in Figure 16-10.
2. Double-check labels assigned to components and wires in the schematic. Do not confuse characters in labels such as "DO" and "D0". Conflicting labels on wires cause errors during design compilation.

When you finish the drawing, your design still has a "U?" above each unlabeled component. If you do not assign labels to the components, the OrCAD software automatically labels them as U1, U2, …, U*x* as it annotates the schematic.

Step 12: Assigning Signals to Specific Pins

You can assign signals to specific EPLD pins rather than letting the software assign them. To do pin assignment, you must be aware of the architecture of the target EPLD device so that you do not, for example, assign an output to an input pin.

You assign pins to input and output signals by assigning the LOC=*pin_number* attribute to IPAD and OPAD components. For example, you can assign a pin number to the SDIN signal to place it in a convenient position on the chip.

To assign the SDIN signal to pin 18, follow these steps:

1. Move the cursor over the IPAD component to which the SDIN signal is connected.
2. Select **`Edit → Edit → 2nd Part Field → Name`**.

   The `2nd Part Field?` prompt appears.

Note: If the second part field is used for something else at your site, pick any of the other seven part fields. The XEPLD Fitter looks at all of them and extracts all the XEPLD attributes that it finds.

3. Type **`LOC=P18`**↵.

4. Click the right mouse button twice or press **Escape** twice to exit the Edit command.
5. Assign any other signals to pins that you wish. Consult *The Programmable Logic Data Book* for a map of the pins on the device.

The LOC attribute assigns the SDIN signal to a specific pin of the targeted XC7236 EPLD device.

The XEPLD software automatically allocates the remaining signals.

Note: If a pin number is entirely numeric, for example pin 18, you must begin the pin number with a "P." If the pin number begins with a letter, for example pin E7, the "P" is not needed. You can assign pins to individual IPAD, OPAD, and IOPAD symbols, but not to the multibit OPAD8 symbols in this schematic.

Step 13: Saving the Design

Save your schematic if you intend to use your design rather than the one provided by Xilinx for the remainder of this tutorial.

1. To use the schematic you created for the rest of this tutorial, select **Quit → Write to File** and type **uart**↵ to save your design as UART rather than as UART2.
2. Alternatively, you can select **Quit → Abandon Edits → Yes** to exit and use the Xilinx-supplied UART file for the rest of the tutorial.

Note: You cannot simulate EPLD designs that contain PLD symbols, such as the PL22V10 symbol, until they are mapped by the XEPLD software. The PLD library components do not have any models associated with them to support pre-mapping functional simulation.

Step 14: Exiting OrCAD

Exit OrCAD and return to XDM.

1. Select **To Main → Execute** from the Schematic Design Tools menu.
2. Select **Exit ESP → Execute**.
3. Press any key to return to XDM.

   The XDM screen reappears.

You have now completed this session and are ready to proceed to the next section of the tutorial, where you define the function of the PL22V10 PLD component used in the schematic.

Session 3: Defining PLD Equations

You can define the function of the PLD in a schematic either directly in the PLUSASM equation language or through a third-party PLD compiler. As an example, a Xilinx ABEL file that produces a PLD file equivalent to rcvr.pld is shown in Figure 16-12. This session demonstrates the creation of a simple PLD equation file using PLUSASM.

You can use the PLUSASM language to define the function of a PLD in a schematic in terms of Boolean equations. In this session you learn how to develop the equation file, rcvr.pld, for the PL22V10 PLD used in the UART schematic. The following functions are introduced in this session:

Step 1: Defining Declaration Statements

Step 2: Creating Boolean Equations

Step 1: Defining Declaration Statements

Display the contents of the rcvr.pld text file.

1. Click on the XDM **Directory** field or select **Utilities → Directory**.
2. Type **c:\orcad\uart**↵ at the command prompt or select the UART project directory from the displayed directory list.
3. Select **Done**.

   The Directory field in XDM now displays the full path to the UART design directory.
4. Similarly, ensure that the Family and Part fields are set to XC7300 and InDesign.
5. Select **Utilities → Browse** or **Utilities → Edit**.
6. Select **RCVR.PLD** from the list of files in your design directory.
7. Examine the declarations section of the PLD file.

The declarations section of the RCVR Boolean equation file is shown in Figure 16-11.

The declarations section contains design and signal identification information. The PLUSASM keywords TITLE, AUTHOR, COMPANY, and DATE identify the design. The keyword CHIP identifies the PLD equation file name RCVR and PLD type PL22V10. The last two lines in the declarations section form a sequential list of signal names and polarities for each pin on the PL22V10.

Step 2: Creating Boolean Equations

Boolean equations are defined in the equations section of the PLD file.

1. Examine the equations section of the rcvr.pld file.

   The equations section of the RCVR equation file is shown in Figure 16-11. The keyword EQUATIONS identifies the beginning of this section. Following the EQUATIONS keyword are the Boolean equations written for each output signal used in the PL22V10.

2. Exit from your editor and return to the XDM executive screen.

For more detailed information regarding the syntax of PLUSASM equations and keywords, refer to the PLUSASM Language Reference in the *XEPLD Reference Manual*.

```
TITLE        UART Receiver Controller
AUTHOR       Applications
COMPANY      Xilinx
DATE         February 1993

CHIP    RCVR    PL22V10

; 1      2   3   4   5   6   7   8     9     10  11  12
  x4clk  c0  c1  c2  c3  c4  c5  read  sdin  d0  nc  gnd

; 13  14  15    16     17      18  19      20     21     22    23  24
  nc  nc  start bitclk byteclk par framing parity overun ready nc  vcc

EQUATIONS

/start := /start * sdin
        + c5*/c4*c3*/c2*/c1*c0
                                ; start goes high when sdin goes low
                                ; and stays high until count=41.

bitclk := /c0 * /c1 * start
                                ; bitclk pulses every 4 clock cycles
                                ; to strobe deserializer.
```

Figure 16-11 Rcvr.pld File

```
byteclk := c5*/c4*/c3*/c2*c1*/c0 * /ready
                                ; byteclk strobes output_reg at count=34
                                ; only if ready not still active.

overun := c5*/c4*/c3*/c2*c1*/c0 * ready
        + overun * /read
                                ; overrun error at count=34 if ready still
                                ; active; stays on until read.

par := par * /sdin * bitclk * start
     + /par * sdin * bitclk * start
     + par * /bitclk * start
                                ; accumulate parity on sdin on each bitclk;
                                ; reset while start=0.

parity := c5*/c4*/c3*/c2*c1*/c0 * par
        + parity * /read
                                ; parity error at count=34 if par odd (1);
                                ; stays on until read.

framing := c5*/c4*c3*/c2*/c1*/c0 * /sdin
```

Figure 16-11Rcvr.pld File (continued)

```
module rcvr
title 'Control Logic and Error Detector for UART Receiver Design
       Xilinx EPLD Applications, Feb. 93'

        rcvr  device  'p22v10';

" Inputs
  x4clk              pin 1;           " External clock (4x baud rate)
  c0,c1,c2,c3,c4,c5  pin 2,3,4,5,6,7; " State counter outputs (from cntr6)
  read               pin 8;           " Read enable (from cntr6,active-high)
  sdin               pin 9;           " Serial data input (external)
  d0                 pin 10;          " Shift register LSB output

" Outputs
  start              pin 15 istype 'reg'; " Start bit detector
  bitclk             pin 16 istype 'reg'; " Bit clock (to shifter)
  byteclk            pin 17 istype 'reg'; " Output data register clock
  par                pin 18 istype 'reg'; " Parity accumulator
  framing            pin 19 istype 'reg'; " Framing error output (external)
  parity             pin 20 istype 'reg'; " Parity error output (external)
  overun             pin 21 istype 'reg'; " Overrun error output (external)
  ready              pin 22 istype 'reg'; " Receiver ready output (external)

" Variables
  count = [c5..c0]; " c5 is MSB

Equations

!start := !start & sdin           " Start goes high when sdin goes low;
        # (count == 41);          " start stays high until count=41.
start.clk = x4clk;

bitclk := !c0 & !c1 & start;      " Bitclk pulses every 4 cycles.
bitclk.clk = x4clk;
```

Figure 16-12 Rcvr.abl File

```
ready := (count == 41) & !parity & !framing & !overun
       # ready & !read;           " Ready goes high at count=41 if no errors
ready.clk = x4clk;                " and stays high until register read.

byteclk := (count == 34) & !ready; " Strobe data register at count=34
byteclk.clk = x4clk;               " only if ready not still active.

overun := (count == 34) & ready  " Overrun error at count=34 if ready still on;
       # overun & !read;         " overun stays on until register read.
overun.clk = x4clk;

par := (par $ sdin) & bitclk & start
     # par & !bitclk & start;       " Accumulate parity of sdin on each bitclk;
par.clk = x4clk;                    " reset while start=0.

parity := (count == 34) & par     " Parity error at count=34 if par odd (1);
      # parity & !read;           " parity stays on until register read.
parity.clk = x4clk;

framing := (count == 40) & (!sdin # !d0) " Framing error at count=40 if either
      # framing & !read;                 " stop bit low;
framing.clk = x4clk;                    " framing stays on until register read.

end
```

Figure 16-12 Rcvr.abl File (continued)

Session 4: Fitting the Design

Session 4 focuses on the XEPLD Fitter. The following tasks are explained in this session.

Step 1: Checking the Design

Step 2: Invoking the Fitter

Step 3: Viewing the Reports

Step 4: Saving Pin Assignments

Step 5: Creating the Programming File

Step 1: Checking the Design

Check the following items before fitting your design:

1. Your directory must be set to the UART design directory in the XDM main screen. The default path is c:\orcad\uart.
2. Just below the Directory field, the Part field must be set to InDesign.

   After you have drawn a new schematic or made significant changes, check for drawing errors by running the Cleanup utility.
3. Select **Translate → CLEANUP**, select the **UART.SCH** file, then select **Done**.
4. If you also want to perform electrical rule checking (ERC), first annotate your design by selecting **Translate → ANNOTATE → UART.SCH → Done**.
5. Run the rule checker by selecting **Translate → INET → UART.SCH → /W (Also perform ERC) → Done**.

Note: Alternatively, to write the ERC messages to a report file called uart.erc, type the following on the XDM command line: **dos inet uart.sch uart.erc /w**↵.

The ERC report shows several warnings about single node nets and inputs with no driving source. These warnings are due to signals deliberately left unconnected in the UART schematic and can safely be ignored.

Step 2: Invoking the Fitter

You can create a netlist from the schematic, process the PLD file, fit the design, and optionally create an Intel Hex programming file using a single command.

Implementing the Design Automatically

Implement your design automatically using the XEMake program.

1. Select **Translate → XEMAKE**.
2. Do not select any of the options; select **Done** from the options prompt.
3. Select **UART.SCH** from the displayed list.
4. When asked to select a target, select **Make design data base**.

XEMake invokes the XDM commands listed in the next section.

Implementing the Design Manually

Alternatively, you can invoke the individual commands to control each step in the fitting process. You do not need to select any options for any of the commands.

The commands in this section are included for your information only; you do not need to perform a manual translation.

1. If you have not yet annotated your schematic, select **Translate → ANNOTATE → UART.SCH → Done**.

   The Annotate program assigns unique name reference designators to each symbol in your schematic.

2. To create an OrCAD netlist, select **Translate → INET → UART.SCH**, turn off the **/W (Also perform ERC)** option if it is on, then select **Done**.

   The INET program creates an OrCAD netlist file for each schematic in your design.

3. Select **Translate → SDT2XNF → UART.INF**. The default output file is uart.xnf; select **Done** to accept this output file name. By default, XNF files are written to the xnf subdirectory using the -d option; select **Done** to accept this output directory.

SDT2XNF generates a Xilinx netlist file for each INF file created by INET.

4. Merge all schematics and macros in your design by selecting **Translate → XNFMERGE**. By default, XNF files are read from the "xnf" subdirectory using the -D option; select **Done** to accept this directory. Select **XNF\UART.XNF** from the list of file names displayed. By default, the output file name is uart.xff; select **Done** to accept this output file name.

   XNFMerge creates a single netlist file containing the complete design.

5. Assemble your RCVR equation file by selecting **Translate → PLUSASM**. Select **RCVR.PLD** from the displayed list of input files.

   The `Assembling rcvr.pld` message appears on the screen. After PLUSASM has run, the screen displays where the output log files are stored.

   PLUSASM displays error or warning messages to the screen during processing. If you encounter any errors or warnings, review the contents of the rcvr.lga log file to understand their context.

   The result of the equation assembler is a VMH file stored in the "clib" subdirectory. If you make changes only in the schematic, you do not have to assemble your equations again before fitting.

6. Invoke the XEPLD Fitter by selecting **Fitter → FITNET**. Select **UART.XFF** from the list of file names displayed.

   If you saved pin allocation information from a prior fitting in a VMF file using the PINSAVE command, the FITNET -f option appears in the FITNET menu. The -f option instructs the Fitter to use an existing pinout.

7. To allow FITNET to automatically assign pin locations, select **Done**.

   The XEPLD Fitter is composed of several submodules. As the Fitter processing proceeds, a message is displayed on the screen indicating which submodule is running. The FITNET modules produce a database file (uart.vmh). From this database, a programming file can be produced to program the device. Simulation models are also produced from this database file.

If the Fitter encounters errors, it displays them on the screen and stores them in a file called uart.err for future reference. If errors are encountered, you can press Ctrl-C at any time to stop the execution and look at the error and warning logs.

Three warnings are displayed for the UART design. One port is removed because it has no logic connection, and two Freq_Divider outputs are removed because they do not drive anything. Ignore these warnings, since they are expected for this design.

Alternative Ways to Process PLDs

Read this section to learn about other design entry methods for XEPLD designs.

XEPLD can read 20V8 or 22V10 PLD designs in the form of JEDEC standard programming maps produced by any third-party PLD compiler. Select Translate → JED2PLD to read the JEDEC file and automatically process it using PLUSASM. If you choose this method, you do not need to also execute the PLUSASM command. The result is a bitmap file. Refer to the Behavioral Entry section of the *XEPLD Reference Guide* for details on JEDEC input.

Some third-party PLD compilers produce PALASM-compatible Boolean equation output files. Most PALASM equation syntax can be read directly by the PLUSASM assembler. For example, the logic for the PLD used in the UART design can be implemented using ABEL syntax, as shown in Figure 16-12, and compiled by Xilinx ABEL or ABEL to produce a formatted PLD equation file. You can read the resulting rcvr.pld file directly using the Translate → PLUSASM command.

Step 3: Viewing the Reports

The following report files are produced by the FITNET command:

uart.err	Fitter Error report
rcvr.err	Assembler Error report
rcvr.lga	Assembler Log report
uart.res	Resource report
uart.map	Mapping report
uart.pin	Pinlist report
uart.lgc	Logic Optimizer report
uart.par	Partitioner report

View the reports that the Fitter generates using the Utilities Browse or Utilities Edit command.

1. View the Fitter Error and Assembler Log reports if error or warning messages appeared on the screen during fitting.

   PLUSASM stores errors and warnings in a report file called rcvr.err. PLUSASM also produces a detailed report of its results in an assembler log file named rcvr.lga.

2. View the Resource report, uart.res.

   The Resource report lists the amount of resources that were used to implement the design. This report contains the total number of macrocell and Function Block resources and I/O pins used on the target device. These totals are subtracted from the total resources of the device to give the amount of remaining resources available to the designer.

3. View the Mapping report, uart.map.

   The Mapping report lists each Function Block in the device and details which component outputs were mapped to macrocells of each Function Block. The Mapping report is used primarily for design debugging and to assist manual mapping.

4. View the Pinlist report, uart.pin.

   The Pinlist report provides you with chip pin placement information. For each pin on the package, the Pinlist report indicates the operation of the pin as used in the design and the

name of the signal in your design appearing on the pin. The signal names in the Pinlist report are the labels you placed on the wires connected to pad symbols in your schematic.

5. View the Logic Optimizer report, uart.lgc.

   The Logic Optimizer report lists which inputs have been collapsed into their fanouts and which outputs have been optimized.

6. View the Partitioner report, uart.par.

   The Partitioner report provides cross-reference tables showing the allocation of all Function Block resources. This report provides detailed information for optimizing your design and manipulating chip resources.

Note: When you are using schematic capture, XEPLD uses the names based on component reference designators passed to it from OrCAD. Understanding these reports with OrCAD-assigned labels can be very difficult. For this reason, you should provide unique reference designators to components.

Step 4: Saving Pin Assignments

Save the pin allocation information into a Pinsave file, uart.vmf, after a successful fitting of your design.

1. Select **`Translate → PINSAVE`**.
2. Select **`UART.VMH`** from the list of file names displayed.

   This command preserves the saved pinout. In most cases, if you make a design change or add more logic, you can re-use this pinout assignment using the data saved in the VMF file.

   A message similar to the following is displayed on your screen:
   `Writing pin allocation in c:\ORCAD\UART.VMF`.

   If you set the -f (Pin-freezing) option of the FITNET command to On, the Fitter assigns listed pins to the locations indicated in the Pinsave file before mapping any other logic or new pins. This allows you to assign pins to the same positions with each iteration of your design. The -f option is Off by default. Selecting -f repeatedly before you select Done toggles the -f option On and Off. The On or Off setting of this option is displayed in a status

line at the bottom of the XDM screen just above the command line. The -f option only appears in the menu if one or more VMF files exist in the project directory.

Step 5: Creating the Programming File

Since you did not select the "Make Intelhex bitmap" target when you used the XEMake command, you must create a programming file.

1. Select **Verify → MAKEPRG**.
2. Select **UART.VMH** from the list of file names displayed.
3. Enter a user signature string to be written into an inactive EPROM area in the chip for identification purposes by typing **uart**↵.

   MAKEPRG creates a programming file, uart.prg.

If you installed the Xilinx HW120 programmer, PROLINK appears under the XDM Verify menu. PROLINK is the control and interface software used to download the programming file to the programmer. Refer to the HW120 documentation for instructions.

You can also create a JEDEC programming file, required by other third-party programmers, by selecting Verify → MAKEJED → UART.VMH.

Session 5: Simulating the Design

This session shows how to simulate the UART design. The following tasks are explained in this session:

Step 1: Creating a Simulation Netlist

Step 2: Preparing Input Vectors

Step 3: Running the Simulation

Step 4: Viewing Simulation Results

Step 5: Correcting Vector Errors

Step 6: Adding a Signal to the Waveform Display

Step 1: Creating a Simulation Netlist

After fitting your design, you can create an OrCAD VST file for timing simulation using a single command.

Creating a Timing Simulation Netlist Automatically

Translate your design automatically using the XSimMake program.

1. Select **Verify → XSIMMAKE**.
2. Select the **-F** option and choose **Orcad_Epld_Timing** as the flow to run, then select **Done**.

   A list of available input files is displayed.
3. Select **UART.VMH** from the displayed list.

XSimMake invokes the XDM commands listed in the next section.

Creating a Timing Simulation Netlist Manually

Alternatively, you can invoke the individual commands to control each step in the translation process. You do not need to select any options for any of the commands.

The commands in this section are included for your information only; you do not need to perform a manual translation.

1. Select **Verify → VMH2XNF → UART.VMH**.
2. Designate an output file name by typing **simuart.xnf**↵.
3. Select **Done**.

   This step creates a new XNF file that contains an image of the EPLD device and its timing parameters.
4. Select **Verify → XNF2VST → SIMUART.XNF → New File**.
5. Designate the output file name by typing **uart.vst**↵.
6. Select **Done**.

   This step creates a model, expressed as an OrCAD VST file, of an EPLD device containing the UART design.

If you are already familiar with OrCAD's simulator, you do not need to complete the rest of the tutorial.

Step 2: Preparing Input Vectors

After you have prepared the VST file as described in the last step, you are ready to enter OrCAD and run the simulation.

Entering the OrCAD Simulator

Use the XDM menus to access the OrCAD simulation software.

1. Select **DesignEntry → ORCAD**.
2. Select **Design Management Tools → Execute** from the main menu.
3. In the directory listing on the left, select the **UART** directory and click on the **OK** button.
4. Select **Digital Simulation Tools → Execute** from the main menu.

Configuring the OrCAD Simulator

Before simulating for the first time, you must configure your UART design directory for simulation.

1. Select **Simulate**.
2. Select **Local Configuration → Configure SIMULATE**.

Note: OrCAD's configuration programs are memory intensive. If you receive the message `Could not find the .EXE, or not enough memory to load \orcadexe\VST_CLC.EXE`, return to the XDM executive screen. Close XDM by typing **exit** ↵, type **orcad** ↵, and continue the configuration as described in this section. Type **xdm** ↵ to return to XDM, re-enter OrCAD, and continue with the tutorial.

3. Under File Options, click on the **Connectivity database** field, type **UART.VST** in the box, and press **Enter**.

   The File Options box appears as shown in Figure 16-13.
4. Make any necessary corrections to the file names in the box.
5. Under the Processing Options section, click on **Use Delay Annotation**.
6. Click on the **OK** button.

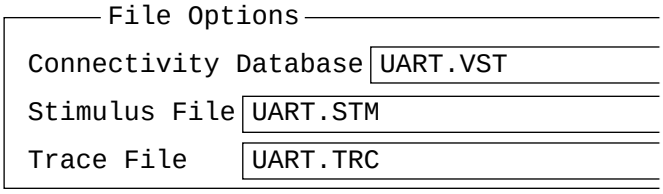


Figure 16-13 File Options in the Configure Simulate Screen

Using the Stimulus Editor

OrCAD provides a stimulus editor that you can use to provide input stimulus for your simulation.

1. Select **Simulate → Execute** to begin your simulation session.

   A waveform display appears. The display is blank except for a list of signals on the left.

   Two setup files determine which signals are on this list and the waveforms of the inputs. To save you time following this tutorial, we have provided these two setup files for you.

2. To view uart.stm, the stimulus setup file, select **Edit Stimulus**.

   The `Editing Stimulus will INITIALIZE simulator. Continue?` prompt appears.

3. Select **Yes.**

   The Stimulus Editor screen of the uart.stm file appears as shown in Figure 16-14.

4. Use the mouse or arrow keys to move the selection to SDIN.

   SDIN is the input data stream.

5. Select **Edit**.

6. The Stimulus Detail Editor screen appears as shown in Figure 16-15.

   This screen determines the times at which the input stimulus changes value. You will edit this screen later in the tutorial.

```
                                    STIMULUS EDITOR
Test Vectors : Disabled
   Signal Context || Signal Name
1. .X4CLK
2. .CS
3. .RD
4. .SDIN
5. .PRLD
6. * Last Record *
```

Figure 16-14 Uart.stm File: Stimulus Editor

```
                                STIMULUS DETAIL EDITOR
Context      : .
Signal Name  : SDIN

Initial Value:  1

    Time                        Function
    3500                              0
    7500                              1
    15500                             0
    19500                             1
    31500                             0
    35500                             1
    59500                             0
    63500                             1
    75500                             0
    87500                             1
    95500                             0
    99500                             1
    End Stimulus
```

Figure 16-15 Uart.stm File: Stimulus Detail Editor Showing Data Input

7. Select **Return** to return to the Stimulus Editor screen.
8. Move the mouse or use the arrow keys to move the selection to X4CLK, the clock input.
9. Select **Edit**.

   The Stimulus Detail Editor screen appears as shown in Figure 16-16.

   The JMP 1500 function in the last line allows you to define a repeating waveform without specifying the specific times at which the input value changes.
10. Select **Return** to return to the Stimulus Editor screen.
11. Select **Use Stimulus** to return to the waveform screen and use the contents of the stimulus file.

```
                              STIMULUS DETAIL EDITOR
Context      : .
Signal Name  : X4CLK

Initial Value:  1

    Time                      Function
    1500                        0
    2000                        1
    2500                      JMP 1500
    End Stimulus
```

Figure 16-16 Uart.stm File: Stimulus Detail Editor Showing Clock Input

Using the Trace Editor

OrCAD VST also includes a trace editor that you can use to provide a list of signals to be monitored during simulation.

1. Select **Trace → Trace Edit** to view uart.trc, the trace setup file.

   The uart.trc file is shown in Figure 16-17. This file lists the signals that are displayed in the waveform display and determines their

display characteristics. The signal names displayed are aliases, which you can change using the Trace Editor.

TRACE EDITOR

	Display Name	Type	Trace	Display
1.	PRLD	Signal	ON	ON
2.	X4CLK	Signal	ON	ON
3.	RD	Signal	ON	ON
4.	CS	Signal	ON	ON
5.	S.DATA.IN	Signal	ON	ON
6.	START	Signal	ON	ON
7.	BITCLK	Signal	ON	ON
8.	BYTECLK	Signal	ON	ON
9.	READY	Signal	ON	ON
10.	OVERRUN	Signal	ON	ON
11.	PARITY	Signal	ON	ON
12.	FRAMING	Signal	ON	ON
13.	D0	BinBus	ON	ON
14.	D1	BinBus	ON	ON
15.	D2	BinBus	ON	ON
16.	D3	BinBus	ON	ON
17.	D4	BinBus	ON	ON
18.	D5	BinBus	ON	ON
19.	D6	BinBus	ON	ON
20.	D7	BinBus	ON	ON
21.	* Last Record *			

Figure 16-17 Uart.trc File

2. To view the information for the S.DATA.IN signal, select the **S.DATA.IN** signal, then select **Edit**.

   The screen shown in Figure 16-18 is displayed.

   S.DATA.IN is an alias, or display name, that you can edit. The actual signal name, SDIN, is listed in the Signal Name field.

   You can use the Browse command to choose signals to add to the Trace file. You will add a new signal to the Trace file later in this tutorial.

3. Select **Return** to return to the main screen of the Trace file.
4. Select **Use Trace** to return to the waveform screen and use the defined set of signals.

```
                    TRACE EDITOR

Display Name : S.DATA.IN
Type         : Signal
Trace        : ON
Display      : ON
Context      : .

Signal Name  : SDIN
```

Figure 16-18 Uart.trc File Showing Detail for One Signal

Step 3: Running the Simulation

The waveform screen only shows 100 time intervals on its X-axis at the bottom of the graph. Each time interval is 0.1 nanoseconds.

1. To view more of the simulation at one time, select **Trace → Change View**.

   The `Trace Delta Time?` prompt appears.

2. Type **1100**↵ at the prompt.

   The waveform graph now shows 11,000 nanoseconds.

   You are now ready to run the simulation.

3. Select **Run Simulation**.

   The `Simulation length?` prompt appears.

4. Type **110000**↵ at the prompt to simulate for 11,000 ns.

   The simulation waveforms appear on the screen as shown in Figure 16-19.

Step 4: Viewing Simulation Results

Figure 16-19 shows the waveforms after simulation is complete.

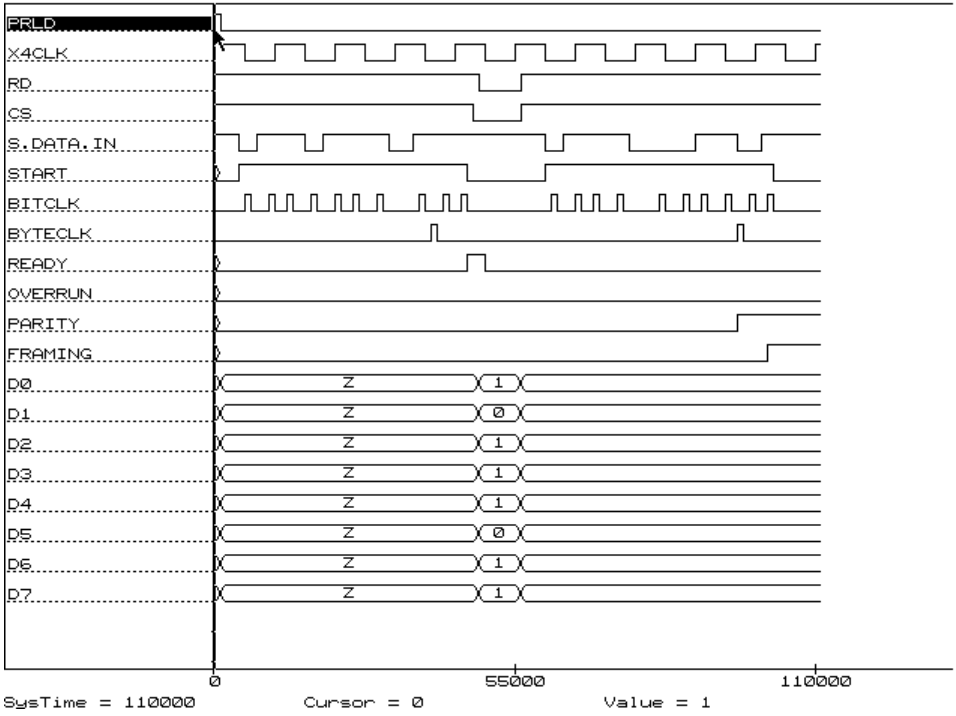


Figure 16-19 UART Simulation Results

Note: This scale is not detailed enough to show all value changes for BITCLK or for X4CLK, which changes four times as often as the BITCLK signal; therefore, only random samples of these waveforms are displayed. Zoom in with the Trace Change View command to see the actual waveforms.

If only a portion of the waveforms is displayed, you can view a part of the waveforms not currently visible by moving the mouse off the edge of the screen.

To rerun a simulation, select Initialize → Yes → Run Simulation.

At about 9,600 nanoseconds, a parity error occurs, and at 10,200 nanoseconds a framing error occurs. These errors were deliberately introduced to test whether the design could catch them.

In the next step, you alter the uart.stm file to correct the parity error. You then modify the uart.trc file to add another signal to the waveform display.

Step 5: Correcting Vector Errors

Edit the values in the uart.stm file to correct the parity error.

Identifying the Errors

The simulation waveforms show that a parity error occurred at approximately 9,600 ns.

1. Look at the first series of bits in the S.DATA.IN stream of the simulation.

   The UART processed these bits without errors. The BITCLK signal identifies the time at which these bits arrive in the S.DATA.IN stream. The first bit is the start bit, and its value is 0. The parity is even, so there are supposed to be an even number of 1s in each group of 7 data bits. The first group contains six 1s. The last two bits are the stop bits, which both have values of 1.

Note: Zoom in with the Trace Change View command to see all transitions on the BITCLK waveform.

2. Compare the first series of bits to the second series, which includes a 0 start bit, five 1s in the data bits, and stop bit values of 0 and 1.

Editing the Stimulus

Edit the uart.stm file to eliminate the parity error. You must change the value of one bit. A good choice is the bit that corresponds to the rising edge of BITCLK at 8200.0 ns.

1. Select **Edit Stimulus**.

   The `Editing Stimulus will INITIALIZE simulator. Continue?` prompt appears.

2. Select **Yes**.

   The Stimulus Editor screen appears.

3. Move the selection to the SDIN signal and select **Edit**.

   The Stimulus Detail Editor appears.

4. Move the selection to any of the values in the Time column and select **Add**.

   The `Time of Function?` prompt appears.

5. Type **80500**↵.

   The `Function?` prompt appears.

6. Select **1** to insert a rising edge.

7. Repeat this procedure to return SDIN to 0 at time 83500.

8. Select **Return → Use Stimulus** to return to the waveform graph.

9. Select **Run Simulation** and type **110000**↵ to simulate for 11,000 ns.

   The Parity error has disappeared, but the Framing error remains, as shown in Figure 16-20. You can eliminate the framing error as well, if you wish, by deleting the Stimulus Detail Editor entry that sets the S.DATA.IN value to 0 at time 95500.

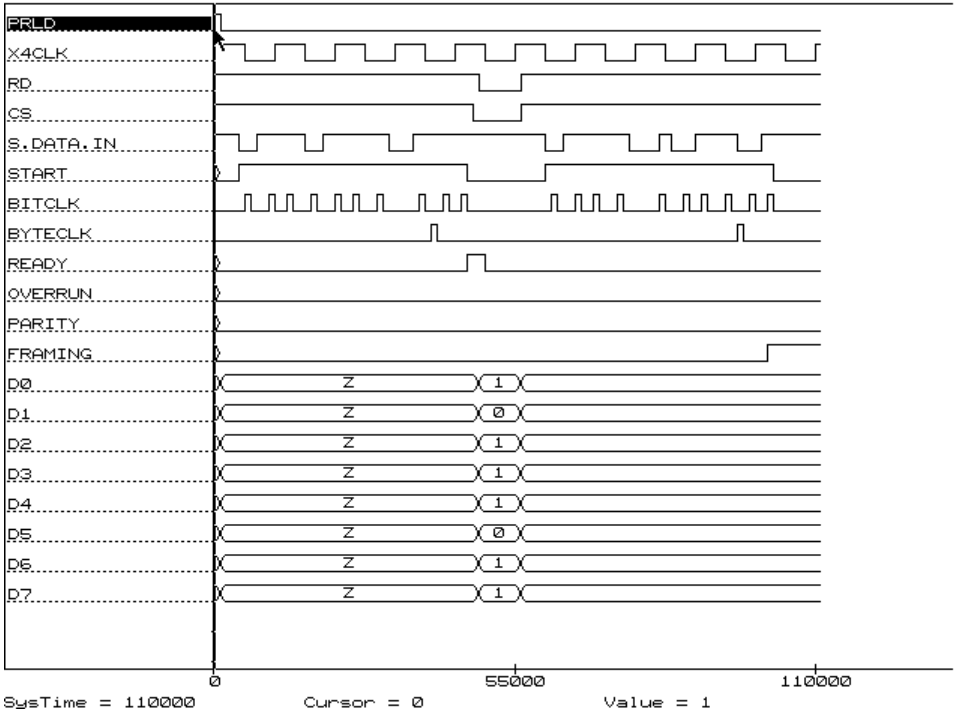


Figure 16-20 Elimination of the Parity Error

Step 6: Adding a Signal to the Waveform Display

You can add a signal to the waveform display. To add the C0 signal, follow these steps:

1. Select **Trace → Trace Edit**.
2. Select **Add**.

   The Trace Editor screen appears as shown in Figure 16-21. The Display Name field is automatically selected.

```
                                TRACE EDITOR

Display Name :
Type         : Signal
Trace        : ON
Display      : ON
Context      : .

Signal Name  :
```

Figure 16-21 Uart.trc File: Trace Editor

3. Select **Edit** and type **C0**↵.
4. Move the selection to the Signal Name field.
5. Select **Browse**.

   All the signals in the design are listed.
6. Use the PageDown key to scroll down the list until you see the C0_1 signal.

Note: Alternatively, you can type **c** to tab to the first signal name beginning with C.

   The internal wire originally labeled C0 appears as C0_1 in the Trace Editor.
7. Select the **C0_1** signal.
8. Select **Return → Use Trace** to return to the waveform screen.
9. Select **Initialize → Yes**.
10. Select **Run Simulation** and type **110000**↵.

    The simulation waveforms appear as shown in Figure 16-22.
11. After examining the simulation output, select **Quit → Abandon Simulation → Yes** to return to the Digital Simulation Tools screen.
12. Select **To Main → Execute** to return to the OrCAD ESP screen.

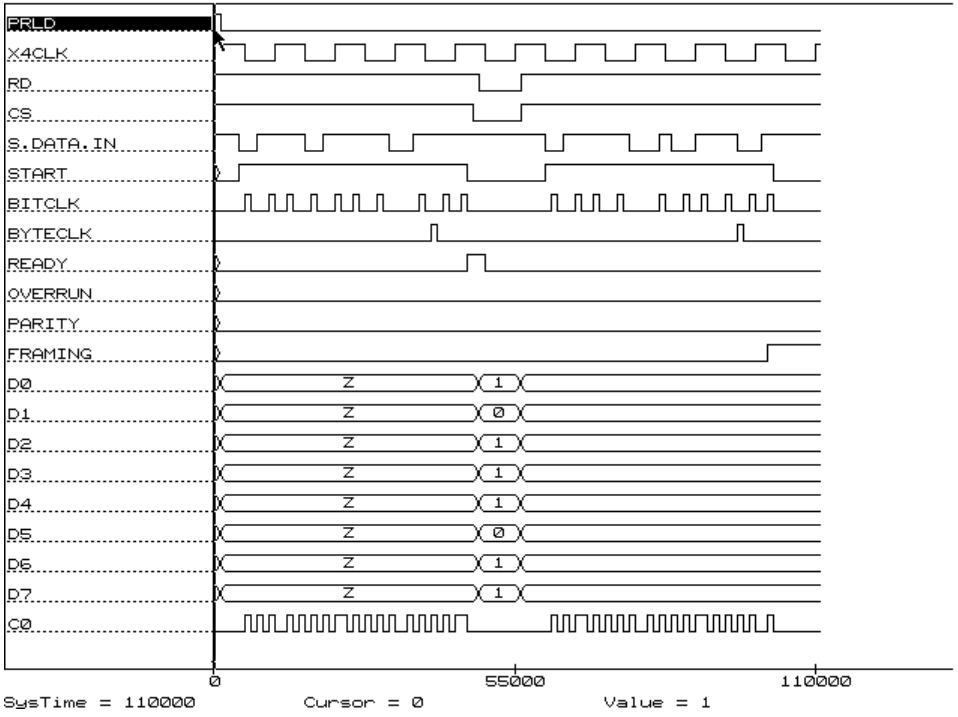


Figure 16-22 Addition of the C0 Signal

Session 6: Functionally Simulating a Purely Schematic Design

This session uses a version of the UART design in which the PL22V10 has been replaced with a lower-level schematic, which represents the RCVR function and has the same logic as the rcvr.pld file. This session also outlines the steps for functional simulation, which you can only perform for a purely schematic design.

Step 1: Copying the UART Design

Step 2: Creating a Custom Sheet Symbol

Step 3: Creating the Lower-Level Schematic

Step 4: Performing a Functional Simulation

Step 5: Exiting OrCAD and XDM

Step 1: Copying the UART Design

Create a new design directory for the purely schematic UART design.

Creating the Uarttop Design Directory

Use the Design Management Tools to create the new design directory.

1. Select **DesignEntry → ORCAD** from the XDM menus if you are not already in the OrCAD ESP environment.
2. Select **Design Management Tools → Execute**.

   A dialog box with directories listed on the left, design files listed on the right, and a group of buttons at the bottom appears.
3. Select the **UART** directory from the list on the left, if it is not already selected.
4. Click on the **Copy Design** button.

   The Copy Design dialog box appears.
5. Click on the **Destination design** box.
6. When a small white square appears in the box, type **uarttop**↵.
7. Click on the **OK** button.

   The new design directory is created.
8. Click on **Cancel** to return to the Design Management Tools screen.
9. Select **UARTTOP** from the list of directories on the left to designate the new design directory.

Copying the Design Files

The completed Uarttop design provided with the Xilinx OrCAD library is installed under the XACT directory. Copy it into your new uarttop design directory so you can view it, as follows:

1. While in the Design Management Tools dialog box, click on **Suspend to System** to open a DOS shell with the uarttop design directory as your current directory.

   Copy the entire contents of the installed uarttop tutorial directory, c:\xact\tutorial\orcad\uarttop, into your new tutorial directory, \orcad\uarttop.

2. Type **copy c:\xact\tutorial\orcad\uarttop**↵.

   Also copy the uart.sch file from the uart design directory.

3. Type **copy ..\uart\uart.sch**↵.

4. Type **exit**↵ to return to OrCAD.

5. Click on the **OK** button to return to the ESP menu.

Step 2: Creating a Custom Sheet Symbol

You can create your own sheet symbols, symbols that represent lower-level schematics. Under these sheet symbols, you can build other schematics. Creating a design with multiple levels allows you to focus on specific parts of the design rather than trying to understand the entire design all at once.

You can either use the uarttop.sch file provided or create your own by creating a custom sheet in the uart.sch file. To create a custom sheet, follow these steps:

1. Select **Schematic Design Tools** → **Execute** → **Draft** → **Execute** to open the schematic editor, Draft.

2. Select **Quit** → **Initialize** and type **uart**↵ to edit the UART schematic.

3. Delete the PL22V10 component by selecting **Delete** → **Object**, moving the cursor over the PL22V10 component, and selecting the **Delete** submenu command.

4. Click the right mouse button or press the **Escape** key to exit the Delete command.

5. Select **Place** → **Sheet**. Use the mouse and the **Begin** and **End** menu selections to draw a sheet whose edges touch all the nets that were connected to the PL22V10. Make sure the sheet extends

at least one grid unit higher than the highest net and one grid unit lower than the lowest net.

Note: You can stop and then resume editing the sheet symbol if you are interrupted while following this tutorial. To resume editing, place the cursor on the sheet symbol and select **Edit → Edit**.

6. Select the **Filename** command from the Place Sheet menu, and type **rcvrsub.sch**↵ when prompted for a file name.
7. Select the **Name** command from the Place Sheet menu, and type **Controller**↵ when prompted for a sheet name.
8. Select the **Add-Net** command. Type **X4CLK**↵ when prompted for a name, and select **Input** for the net type. Place this input even with the top net on the left side of the sheet.
9. Repeat the last step for the following nets, in order, from top to bottom on the left side of the sheet: C0 - C5, READ, SDIN, and D0. Refer to Figure 16-23 as a guide.

   Align these inputs with the input nets. If you make a mistake, delete the input and add it again. You cannot move an input pin once you have created it.
10. Select the **Add-Net** command. Type **READY**↵ when prompted for a name, and select **Output** for the net type. Place this output even with the top net on the right side of the sheet.
11. Repeat the last step for the following nets, in order, from top to bottom on the right side of the sheet: OVERUN, PARITY, FRAMING, BYTECLK, BITCLK, and START. Refer to Figure 16-23 as a guide.

    Align these outputs with the output nets. If you make a mistake, delete the output and add it again. You cannot move an output pin once you have created it.
12. Select **Quit → Write to File** and type **uarttop.sch**↵ when prompted for a file name.

You have created the RCVRSUB sheet symbol. The UARTTOP schematic appears as shown in Figure 16-23.

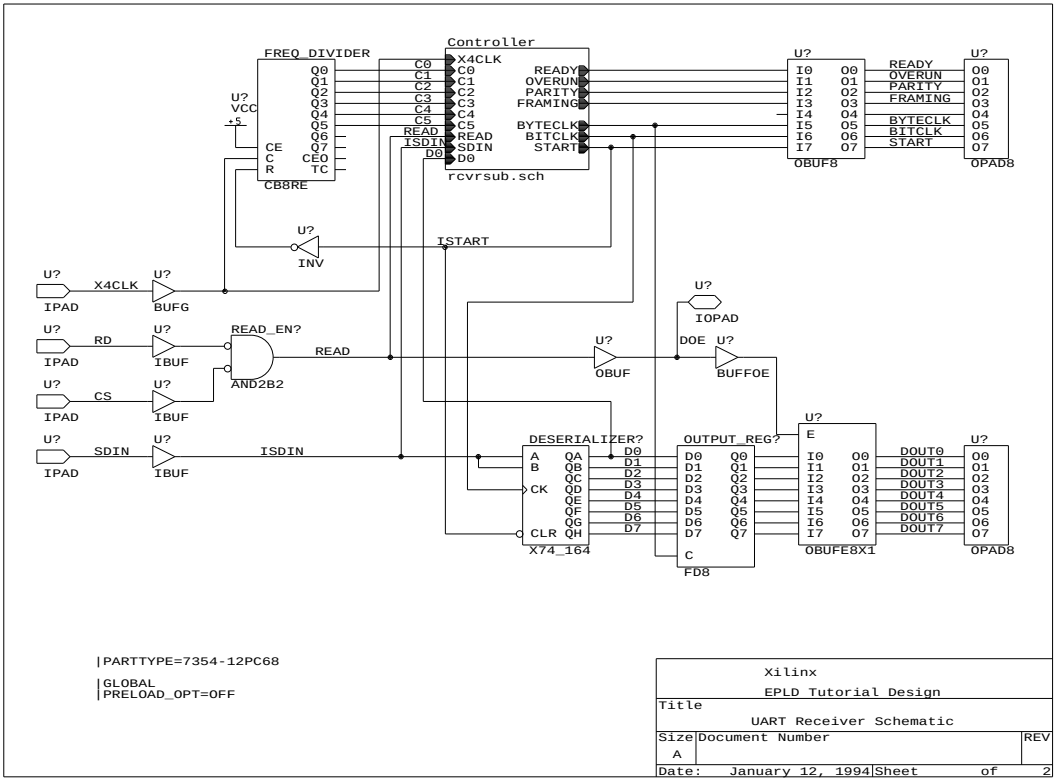


Figure 16-23 UART Schematic with RCVRSUB Sheet Symbol

Step 3: Creating the Lower-Level Schematic

Next, you create the lower-level schematic. To save you time in this tutorial, this file is provided for you.

To view the RCVRSUB schematic, follow these steps:

1. Place the cursor on top of the RCVRSUB sheet symbol.
2. Select **Quit → Enter Sheet → Enter**.

   A schematic sheet named rcvrsub.sch opens, as shown in Figure 16-24 below.

   The only components it contains are INV, OR2, NOR2, OR3, FD, and AND*n*, where *n* is a number between 2 and 9 that indicates the number of inputs.

   The module ports, which match inputs and outputs in the lower-level schematic to the pins in the sheet, were placed using the Place Module Port Place command. This command prompts you for the name of the module port, which must match a pin name in the sheet.
3. Select **Quit → Leave Sheet** to return to the UART schematic.
4. Select **Abandon Edits** to return to the Schematic Design Tools screen.

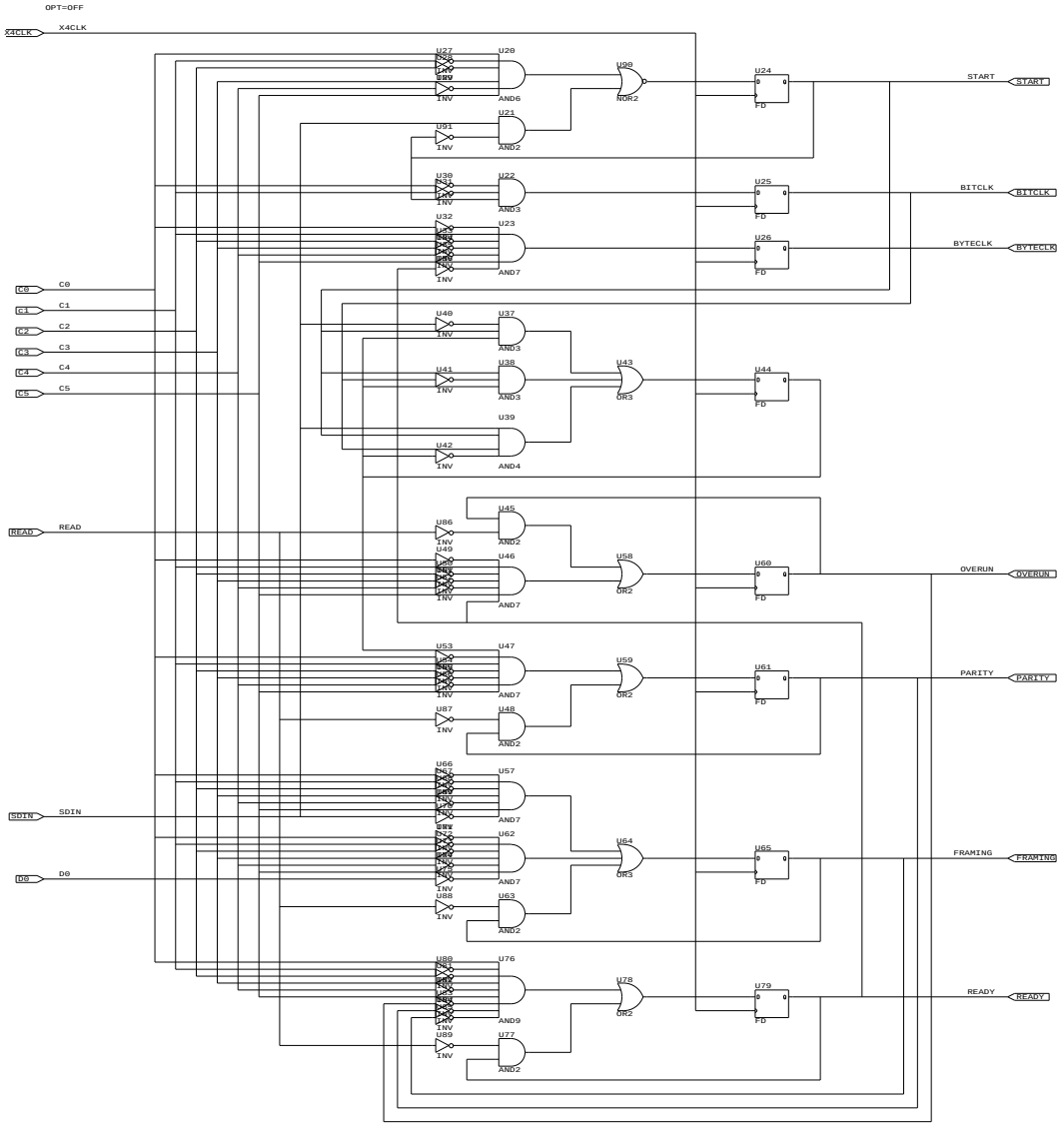


Figure 16-24 RCVRSUB Schematic

Step 4: Performing a Functional Simulation

To perform a functional simulation in OrCAD, follow the steps below.

1. Select **To Main → Execute → Exit ESP → Execute** to exit OrCAD and return to XDM.
2. Click on the **Directory** field in XDM and select **C:\ORCAD\UARTTOP → Done**.
3. Ensure that the Family and Part fields in XDM are set to XC7300 and InDesign.
4. Select **Verify → XSIMMAKE → -F**.
5. Select **Orcad_Epld_Func** as the flow to run, then select **Done**.
6. Select **UARTTOP.SCH** from the displayed list.
7. Re-enter OrCAD and run the simulation as described in Session 5, Steps 2 through 6. Be sure to configure the simulator not to use Delay Annotation, or an error will occur when you enter the simulator.

Step 5: Exiting OrCAD and XDM

Return to the DOS prompt as follows:

1. To exit VST, select **Quit → Abandon Simulation → Yes**.
2. To exit OrCAD, select **To Main → Execute**, then select **Exit ESP → Execute**.
3. To end your XDM session, select **Quit** from the XDM menu.

You have now completed the XEPLD OrCAD Tutorial.

OrCAD Interface/ Tutorial Guide

Program Options

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Appendix A

Program Options

This Appendix contains all the program options available for XDraft, SDT2XNF, and XNF2VST.

XDraft

Usage: xdraft *number* [-*options*]↵

where *number* represents the architecture of the family being targeted: 2, 3, 4, or 7.

Options

-l — Old library option

-s — SDT configuration only

-v — VST configuration only

-x — XACT path

SDT2XNF

Usage: sdt2xnf *inffile*[.inf] [*xnffile*[.xnf]] [-*options*]↵

inffile— Input INF file

xnffile — Output XNF file

Options

-d — Output directory for XNF files

-p *parttype* — Part type used

-s *path* — Search path for Xilinx-defined INF files

-u *path* — Search path for user-defined INF files

XNF2VST

Usage: xnf2vst *xnffile*[.xnf] [*vstfile*[.vst]] [-*options*]↵

xnffile — Input XNF file

vstfile — Output VST file

Options

-r — Read existing Name Reference File (NRF) file

-u — Unit delay option

-w — Overwrite existing AST and ATR files

-x *path* — XACT path

Note: For more information, enter the program name at the DOS prompt and press Enter.

OrCAD Interface/ Tutorial Guide

Error Messages

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Appendix B

Error Messages

XDRAFT (XCFG)

This section lists the error messages for XDraft.

`ERROR 001: The command-line options S and V cannot be both true at the same time.`

`ERROR 002: The XACT environment variable is not set.`

The variable needs to point to the directory where the XACT software is installed. Typically, it should be **`SET XACT=c:\xact`**.

`ERROR 003: The XACT path` *path* `is not set up properly or the Xilinx DS35 OrCAD interface is not installed in the XACT path.`

The XACT variable should be pointing to a valid location, typically the XACT directory.

`ERROR 004: Invalid directory path` *dirpath* `is detected.`

`ERROR 005: A` *command* `syntax error was detected at linenum` *linenum* `of the file` *filename*`.`

Your sdt.cfg or vst.cfg file is corrupted or contains syntax that is not recognized by XDraft. Check the content of these files.

SDT2XNF (INF2XNF)

This section lists the error messages for SDT2XNF.

`ERROR 001: An internal program error occurred, code` *code*.

Your sdt.cfg file might be corrupted. You can rectify this situation by copying the sdt.cfg file from the template directory and rerunning XDraft.

`ERROR 002: The root INF file was not specified from the command line.`

`ERROR 003: An invalid file extension was specified with the file` *filename*.

SDT2XNF should be invoked on INF files.

`ERROR 004: A non-existent file` *filename* `was specified.`

`ERROR 005: A non-readable file` *filename* `was specified.`

Check the "read" permission of the specified INF file.

`ERROR 006: A non-writable file` *filename* `was specified.`

There might exist an output file that does not have any write permission.

`ERROR 007: A non-existent directory` *dirname* `was specified.`

If the SDT2XNF command is invoked with the "-s" option, make sure that the specified directory exists.

`ERROR 008: A` *command* `syntax error was detected at line number` *linenum* `of the INF file` *filename*.

`ERROR 009: An invalid part type` *parttype* `was specified in the root INF file` *filename*.

You might have specified a part type that does not exist. Check the part type and try again.

ERROR 010: Invalid part type *parttype* was specified from the command line.

ERROR 011: A syntax error in the part type specification *parttypespec* was detected.

Make sure you have specified a part type in your schematic. Examples of proper syntax include |PARTTYPE=4005PC84 or |PART=4005PC84-6.

ERROR 012: Failed to add the user record *username* of value *uservalue*.

ERROR 013: The XACT environment variable is not set.

The variable needs to point to the directory where the XACT software is installed. Typically, it should be **SET XACT=c:\xact**.

ERROR 014: The Xilinx DS35 OrCAD interface is not installed or the XACT path is not set up properly.

Make sure the XACT environment variable is pointing to the proper directory. The default is where your XACT software is installed.

ERROR 015: No default Xilinx OrCAD family library was found under the path *xactpath*.

ERROR 016: Failed to open the file *filename* for reading.

Make sure all the schematic files have “read” permissions.

ERROR 017: Failed to create the xnf cell *xnfcellname*.

ERROR 018: Failed to get the next INF command at line number *linenum* in the INF file *filename*.

Your INF file might be corrupted. Try running INET with option /t on your schematic, then try SDT2XNF again.

ERROR 019: Failed to add the signal *signame* at line number *linenum* in the INF file *filename*.

ERROR 020: Failed to find the signal *signame*.

ERROR 021: Failed to add the symbol *symname* of type *symtype* at linenum *linenum* in the INF file *filename*.

ERROR 022: Failed to add parameter *paramname* to the symbol *symname* at line number *linenum* in the INF file *filename*.

ERROR 023: Failed to add user parameter *userparamname* to the symbol *symname* at line number *linenum* in the INF file *filename.*

ERROR 024: Failed to add the loc specification *loc_spec* on the symbol *symname*.

ERROR 025: Failed to add the signal *signame* to the XNF network structure at line number *linenum* in the INF file *filename*.

ERROR 026: Failed to add the external I/O for the signal *signame* of type *type* to the XNF file at line number *linenum* in the INF file *filename*.

This situation arises when there might be more than one source to a net. Check your schematic and try again.

ERROR 027: Program internal parsing error at line mumber *linenum*.

ERROR 028: An invalid INF command line was detected at line number *linenum* of the file *filename*.

The INF file contains characters that are not understood by SDT2XNF. You either need to correct this problem or generate an INF file again by running the INET program.

ERROR 029: No source was found on the joint statement at line number *linenum* in the INF file *filename*.

ERROR 030: Multiple sources were found on the joint statement at line number *linenum* in the INF file *filename*.

ERROR 031: The symbol *symname* was not found in the XNF file when processing the command *command*.

Your INF file is missing an instance of a symbol or it might be corrupted. Regenerate the INF file by running the INET program.

ERROR 032: Failed to add pin *pinname* of type *pintype* on the signal *signame* to the symbol *symname*.

There might be a duplicate reference designator and a duplicate net in the INF file or the INF file might be corrupt. First check the schematic file (SCH) and remove the duplicate symbol and net if they are present. Regenerate the INF file by running the INET program.

XNF2VST (XNF2INF)

This section lists the error messages for XNF2VST.

ERROR 001: The XNF file was not specified from the command line.

ERROR 002: An invalid file extension was detected with the file *filename*. The valid file extensions are ".xnf", ".xff", ".xg".

ERROR 003: A non-existent file *filename* was detected.

ERROR 004: A non-readable file *filename* wasx detected. Check the "read" permission of your input filename.

ERROR 005: A non-writable file *filename* was detected. Check the "write" permission of the output filename.

ERROR 006: Failed to create the XNF data cell *xnfcellname*. This can be an indication of a symbol error.

If your design includes setup / hold timing information, check to make sure that the input file includes that timing.

ERROR 007: No part type was specified in the XNF file *filename*.

ERROR 008: An invalid part type *parttype* was specified in the XNF file *filename*.

ERROR 009: The XACT environment variable is not set.

The variable needs to point to the directory where the XACT software is installed. Typically, it should be **SET XACT=c:\xact**.

ERROR 010: The Xilinx DS35 OrCAD interface is not installed or the XACT path is not set up properly.

Make sure the XACT environment variable is pointing to the proper directory. The default is where your XACT software is installed.

ERROR 011: An invalid directory path *dirpath* was detected.

ERROR 012: A duplicate symbol *symname* of type *symtypename* was detected.

ERROR 013: Failed to create the symbol *symname* of type *symtypename*.

ERROR 014: Failed to create the signal *signame*.

ERROR 015: Failed to create the pin *pinname* of type *pintype* on the signal *signame* of the symbol *symname*.

ERROR 016: Failed to find the input pin *inpinname* or the clock pin *clkpinname* for the setup record on the symbol *symname* of type *symtypename*.

ERROR 017: Failed to create setup on the symbol *symname* of type *symtypename*.

ERROR 018: Failed to create the external name *extname* of type *exttype*.

ERROR 019: Failed to create a parameter name *paramname* with the value *paramvalue* on the symbol *symname* of type *symtypename*.

ERROR 020: Failed to create a parameter name *paramname* with the value *paramvalue* on the pin *pinname*.

ERROR 021: Failed to create a parameter name *paramname* with the value *paramvalue* on the signal *signame*.

ERROR 022: A pin *pinname* is invalid on the symbol *symname* of type *symtypename*.

Your input file might be corrupted. Re-create it and then run XNF2VST again.

ERROR 023: A *command* syntax error was detected at line number *linenum* of the NRF file *filename*.

Your NRF file might be corrupted or missing an argument. If there is a variable $*number=signalname*, make sure that *signalname* exists.

ERROR 024: A syntax error was detected in the stimulus specification *stmspec* for signal *signame*.

Check the stimulus data on your schematic. It might contain illegal syntax.

ERROR 025: Sym *symname* of type *symtypename* has only one input connected.

ERROR 026: There is no initial value for the symbol *symname* of type *symtypename*.

You must define an "INIT" value for your ROM on the schematic.

ERROR 027: There is an illegal initial value *initvalue* for the symbol *symname* of type *symtypename*.

The "INIT" value on your ROM might be too long for its size.

ERROR 028: There is an illegal hex value *initvalue* for the symbol *symname* of type *symtypename*.

Check the syntax of the "INIT" value on the ROM part of the schematic.

```
ERROR 029: Some symbols have the LIBVER parameter
and some do not. A LIBVER parameter indicates
that the symbol is from the Unified Libraries.
```

Your design contains symbols from both the Unified Libraries and the older libraries. You cannot mix the two symbol types in the same design.

OrCAD Interface/ Tutorial Guide

Warning Messages

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Appendix C

Warning Messages

XDRAFT (XCFG)

This section lists the earning messages for XDraft.

WARNING 001: The Xilinx family was not specified from the command line.

WARNING 002: An invalid Xilinx device family was specified from the command line. The valid families are 2 (XC2000), 3 (XC3000), 4 (XC4000), and 7 (7000).

WARNING 003: Could not find the OrCAD configuration file *filename* in the current directory.

WARNING 004: The OrCAD configuration file *filename* is not readable.

Check the "read" permission on the configuration file.

WARNING 005: Failed to open the file *filename*.

Check the permission of the configuration files or the directory where these files are located.

WARNING 006: No SDT configuration is performed.

WARNING 007: No VST configuration is performed.

WARNING 008: Failed to create the new SDT configuration file sdt.cfg. The new SDT configuration file is called xcfg.sdt.

Check the permissions on the sdt.cfg file.

WARNING 009: Failed to create the new VST configuration file vst.cfg. The new VST configuration file is called xcfg.vst.

Check the permissions on the vst.cfg file.

SDT2XNF (INF2XNF)

This section lists the warning messages for SDT2XNF.

WARNING 001: The part type was either not specified on the command line or in the schematic or it conflicts with the SDT library used in the design, the default part type *defaultpart* is being used.

WARNING 002: Syntax error in parameter specification; it will be ignored.

WARNING 003: An unknown primitive or macro *symbolname* was found. If this is a user-created macro, make sure that a corresponding INF file exists.

The INF file might be corrupted. Try to regenerate the INF file.

WARNING 004: You might have mixed the old and new SDT libraries together, which is not allowed. If you ignore this warning, the output of this program is unpredictable.

The SDT configuration file should contain only one type of library, with only one "PLIB" statement.

WARNING 005: Illegal parameter specification *param_spec* at line number *linenum* in the INF file *filename*.

WARNING 006: Failed to add loc parameter *locparam* to the external I/O.

Check the syntax of the parameters you specified on the designated external record.

WARNING 007: Failed to add parameter *paramname* of value *paramvalue* to the external I/O.

WARNING 008: Failed to add user parameter *paramname* of value *paramvalue* to the external I/O.

WARNING 009: Failed to add the parameter *paramname* to pin *pinname* of type *pintype* on symbol *symname*.

WARNING 010: The INF vector statement is not supported; therefore, it will be ignored.

WARNING 011: All pins on the symbol *symname* of type *symtypename* are unconnected and the symbol will be removed.

You might have inserted a component in the schematic with no connections to it. If there is no connection to a component, the symbol is ignored. Even if the component appears to be connected, it may not be. Run Cleanup to adjust wire endpoints.

WARNING 012: The stimulus specified on the bus *busname* is not supported; therefore, it will be ignored.

WARNING 013: The stimulus specified on the pin *pinname* of symbol *symname* will be ignored. Please make sure that the net to which the stimulus was attached is labeled.

WARNING 014: The parttype was not specified either from the command line or in the root INF file.

WARNING 015: The trace specified on the bus *busname* is not supported; therefore, it will be ignored.

WARNING 016: The trace specified on the pin *pinname* of symbol *symname* will be ignored. Make sure that the net to which the trace was attached is labeled.

XNF2VST (XNF2INF)

This section lists the warning messages for XNF2VST.

WARNING 001: The GTS pin is not connected on the 4K STARTUP symbol.

WARNING 002: The GSR pin is not connected on the 4K STARTUP symbol.

WARNING 003: Failed to find the signal name *signame* in the NRF file.

Either check your NRF file again for the proper syntax, or try executing the XNF2VST command without the "-r" parameter.

WARNING 004: Failed to find the symbol name *symname*.

WARNING 005: The name *name* is longer than *length* characters at line number *linenum* of the file *filename*.

WARNING 006: The pin *pinname* on symbol *symname* of type *symtype* is not connected. It is assumed to be inactive.

WARNING 007: No setup or hold information was found for the pin *pinname* on the symbol *symname* of type *symtype*.

WARNING 008: Timing delay translation was specified, but no delay information was found. The DBA file is invalid for timing simulation.

WARNING 009: A non-simulatable symbol *symname* of type *symtypename* was found. The netlist might be invalid for simulation.

OrCAD Interface/ Tutorial Guide

OrCAD XEPLD
Demonstration Procedure

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Appendix D

OrCAD XEPLD Demonstration Procedure

This appendix summarizes the steps needed to run through the entire EPLD tutorial as a demonstration. These steps are described fully in the XEPLD tutorial chapter. This demonstration procedure assumes that you have completely installed and configured the XEPLD implementation software (DS550 or DS502), the OrCAD Library and Interface (DS35), and the OrCAD SDT, VST, and ESP tool sets.

Entering XDM and OrCAD

1. Type **xdm** ↵ at the DOS prompt.
2. In XDM, select **Family** → **XC7300** → **InDesign**.
3. Select **DesignEntry** → **ORCAD**.

   The OrCAD ESP menu appears.
4. Select **Design Management Tools** → **Execute**.

Configuring the Design Directory

Execute these commands the first time that you perform the tutorial:

1. Select **Create Design** → **New Design Name**, type **uart**↵, and select **OK**.
2. Select **Suspend to System**.
3. To copy the tutorial files from the XACT installation area, type the following at the DOS prompt:

   copy \xact\tutorial\orcad\uart↵

   Adjust the directory path in the copy command if you installed the OrCAD interface to a directory other than \xact.

4. Type **xdraft 7**↵ to configure the XC7000 library.
5. Type **exit**↵ to return to OrCAD.

Examining the UART Schematic

1. Select the **UART** directory and click on the **OK** button.
2. Select **Schematic Design Tools** → **Execute**.
3. Select **Draft** → **Execute**.

   The UART schematic appears; examine it as you wish.
4. Select **Quit** → **Abandon Edits** to close the UART schematic.
5. To exit OrCAD, select **To Main** → **Execute**, then select **Exit ESP** → **Execute**.

Examining the PLD File

1. Select **Directory** → **C:\ORCAD\UART** → **Done**.

   You might need to traverse several directories to find \orcad\uart.
2. To display the contents of the rcvr.pld file, select **Utilities** → **Browse** → **RCVR.PLD**.

   Examine the contents of the rcvr.pld file
3. Exit the browser by selecting **File** → **Exit**.

Implementing the Design

1. Select **Translate** → **XEMAKE** → **Done.**
2. Select **UART.SCH** from the list.
3. Select the **Make design data base** target.

   The schematic is read and the design is mapped into an XC7354 device.
4. Select **Verify** → **MAKEPRG** → **UART.VMH** to create a programming file.
5. Enter **uart**↵ as the signature.

Creating a Simulation Netlist

1. Select **Verify → XSIMMAKE**.
2. Select **-F → Orcad_Epld_Timing → Done**.
3. Select **UART.VMH** from the list.

   A timing simulation model is generated.
4. Select **DesignEntry → ORCAD** on the XDM menu.
5. Select **Design Management Tools → Execute**.
6. Select the **UART** directory and click on the **OK** button.
7. Click on **Digital Simulation Tools → Execute**.

Configuring the Simulator

Execute these commands the first time you perform the simulation:

1. Select **Simulate → Local Configuration → Configure SIMULATE**.
2. Click on the **Connectivity database** field, type **uart.vst↵** in the box.
3. Ensure that the stimulus and trace files are set to uart.stm and uart.trc.
4. Click on the **Use Delay Annotation** button.
5. Click on **OK**.

Simulating the Design

1. Select **Simulate → Execute** to begin your simulation session.
2. Select **Trace → Change View**; enter **1100↵** as the Trace Delta Time.
3. Select **Run Simulation**; enter **110000↵** as the Simulation Length.

   Simulation waveforms appear.
4. Select **Quit → Abandon Simulation → Yes** to return to the Digital Simulation Tools menu.

5. To exit OrCAD, select **To Main → Execute**, then select **Exit ESP → Execute**.
6. To exit XDM, select **Quit** from the menu.

OrCAD Interface/ Tutorial Guide

Index

OrCAD Interface/Tutorial Guide — 0401409 01 *Printed in U.S.A.*

Index

E

M

N

R

S

T

U

V

W

X

Z