

# ***Viewlogic Tutorials***

***PROcapture and PROsim  
Tutorial***

***X-BLOX Tutorial***

***Xilinx ABEL Tutorial***

***XACT-Performance and  
Timing Analyzer Tutorial***

**Σ XILINX**<sup>®</sup>, XACT, XC2064, XC3090, XC4005, and XC-DS501 are registered trademarks of Xilinx. All XC-prefix product designations, XACT-Floorplanner, XACT-Performance, XAPP, XAM, X-BLOX, X-BLOX plus, XChecker, XDM, XDS, XEPLD, XPP, XSI, BITA, Configurable Logic Cell, CLC, Dual Block, FastCLK, HardWire, LCA, Logic Cell, LogicProfessor, MicroVia, PLUSASM, SMARTswitch, UIM, VectorMaze, VersaBlock, VersaRing, and ZERO+ are trademarks of Xilinx. The Programmable Logic Company and The Programmable Gate Array Company are service marks of Xilinx.

IBM is a registered trademark and PC/AT, PC/XT, PS/2 and Micro Channel are trademarks of International Business Machines Corporation. DASH, Data I/O and FutureNet are registered trademarks and ABEL, ABEL-HDL and ABEL-PLA are trademarks of Data I/O Corporation. SimuCad and Silos are registered trademarks and P-Silos and P/C-Silos are trademarks of SimuCad Corporation. Microsoft is a registered trademark and MS-DOS is a trademark of Microsoft Corporation. Centronics is a registered trademark of Centronics Data Computer Corporation. PAL and PALASM are registered trademarks of Advanced Micro Devices, Inc. UNIX is a trademark of AT&T Technologies, Inc. CUPL, PROLINK, and MAKEPRG are trademarks of Logical Devices, Inc. Apollo and AEGIS are registered trademarks of Hewlett-Packard Corporation. Mentor and IDEA are registered trademarks and NETED, Design Architect, QuickSim, QuickSim II, and EXPAND are trademarks of Mentor Graphics, Inc. Sun is a registered trademark of Sun Microsystems, Inc. SCHEMA II+ and SCHEMA III are trademarks of Omaton Corporation. OrCAD is a registered trademark of OrCAD Systems Corporation. Viewlogic, Viewsim, and Viewdraw are registered trademarks of Viewlogic Systems, Inc. CASE Technology is a trademark of CASE Technology, a division of the Teradyne Electronic Design Automation Group. DECstation is a trademark of Digital Equipment Corporation. Synopsys is a registered trademark of Synopsys, Inc. Verilog is a registered trademark of Cadence Design Systems, Inc.

Xilinx does not assume any liability arising out of the application or use of any product described or shown herein; nor does it convey any license under its patents, copyrights, or maskwork rights or any rights of others. Xilinx reserves the right to make changes, at any time, in order to improve reliability, function or design and to supply the best product possible. Xilinx will not assume responsibility for the use of any circuitry described herein other than circuitry entirely embodied in its products. Xilinx devices and products are protected under one or more of the following U.S. Patents: 4,642,487; 4,695,740; 4,706,216; 4,713,557; 4,746,822; 4,750,155; 4,758,985; 4,820,937; 4,821,233; 4,835,418; 4,853,626; 4,855,619; 4,855,669; 4,902,910; 4,940,909; 4,967,107; 5,012,135; 5,023,606; 5,028,821; 5,047,710; 5,068,603; 5,140,193; 5,148,390; 5,155,432; 5,166,858; 5,224,056; 5,243,238; 5,245,277; 5,267,187; 5,291,079; 5,295,090; 5,302,866; 5,319,252; 5,319,254; 5,321,704; 5,329,174; 5,329,181; 5,331,220; 5,331,226; 5,332,929; 5,337,255; 5,343,406; 5,349,248; 5,349,249; 5,349,250; 5,349,691; 5,357,153; 5,360,747; 5,361,229; 5,362,999; 5,365,125; 5,367,207; 5,386,154; 5,394,104; 5,399,924; 5,399,925; 5,410,189; 5,410,194; 5,414,377; RE 34,363, RE 34,444, and RE 34,808. Other U.S. and foreign patents pending. Xilinx, Inc. does not represent that devices shown or products described herein are free from patent infringement or from any other third party right. Xilinx assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

Xilinx products are not intended for use in life support appliances, devices, or systems. Use of a Xilinx product in such applications without the written consent of the appropriate Xilinx officer is prohibited.

## Preface

---

### About This Manual

This manual is a series of tutorials illustrating how to use the PRO Series PROcapture, PROsim, and PROwave programs. It also shows you how to use these programs with X-BLOX, Xilinx ABEL, XACT-Performance, and the Timing Analyzer.

Before using this manual, you should be familiar with the operations that are common to all of Xilinx's 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 *Development System Reference Guide*.

Other publications that you can consult for related information are the *Viewlogic Interface Guide*, the *Xilinx ABEL User Guide*, the *X-BLOX Reference/User Guide*, the *Design Manager/Flow Engine Reference/User Guide*, the *Timing Analyzer Reference/User Guide*, and the *PRO Series* manuals from Viewlogic.

### Manual Contents

This manual covers the following topics.

- Chapter 1, "PROcapture and PROsim Tutorial," guides you through a typical design procedure from schematic entry to completion of a functioning device using Viewlogic's PROcapture schematic editor. It steps through both a functional simulation and a timing simulation using Viewlogic's PROsim and PROwave programs. It also describes how use the Xilinx Design Manager to implement the design.

- Chapter 2, “X-BLOX Tutorial,” shows you how to incorporate X-BLOX modules into your FPGA design. X-BLOX is an advanced library and a synthesis tool that allows you to shorten design entry time, increase design speed, and use a device more efficiently.
- Chapter 3, “Xilinx ABEL Tutorial,” shows you how to incorporate Xilinx ABEL modules into your FPGA design. Xilinx ABEL 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).
- Chapter 4, “XACT-Performance and Timing Analyzer Tutorial,” shows you how to use XACT-Performance and the Timing Analyzer on an FPGA design in the Viewlogic environment. XACT-Performance consists of a set of library primitives that allow timing requirements to be placed on a schematic. The implementation tools use this timing information during mapping, placing, and routing of the design. The Timing Analyzer is a Windows-based tool that performs a static timing analysis of a routed FPGA design.

## 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.
<b>Courier font</b> <b>bold</b>	Literal commands that you must enter in syntax statements are in bold Courier font.
<i>italic font</i>	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 1 PROcapture and PROsim Tutorial

Introduction .....	1-1
Devices .....	1-1
Length.....	1-1
Design Description.....	1-2
Getting Started.....	1-3
Required Software .....	1-3
Before Beginning the Tutorial .....	1-4
Installing the PRO Series Tutorial.....	1-5
Starting Xilinx PROflow.....	1-6
Defining the Calc Project .....	1-8
Creating the Calc Project.....	1-8
Obtaining Design Status.....	1-12
Selecting the CALC.1 Schematic .....	1-13
Navigating in PROcapture .....	1-17
Mouse Buttons.....	1-17
Function Keys .....	1-17
Starting PROcapture.....	1-18
Changing the PROcapture Window Colors .....	1-20
Moving Around the Screen .....	1-22
Panning .....	1-23
Zooming.....	1-23
Making Icons of Schematics and Symbols .....	1-27
PROcapture Command Summary .....	1-28
Creating Symbols.....	1-31
Creating the ANDBLK2 Symbol.....	1-31
Changing the Size of the Symbol .....	1-34
Creating a Symbol Box.....	1-34
Adding Pins .....	1-36
Adding Pin Labels.....	1-37
Adding Pin Attributes .....	1-39
Using the Add Object Attribute Command.....	1-40
Editing Pin Attributes .....	1-41
Adding Other PINTYPE Attributes.....	1-44

Changing Attribute Size .....	1-45
Controlling Attribute Visibility .....	1-47
Adding Symbol Text.....	1-48
Changing Symbol Text Size .....	1-50
Moving Text and Objects .....	1-50
Saving the ANDBLK2 Symbol .....	1-52
Creating the ORBLK2 Symbol .....	1-53
Viewing Symbols Simultaneously .....	1-55
Closing Symbol Windows .....	1-57
Creating Schematics .....	1-59
Opening a Schematic Sheet .....	1-59
Adding Components .....	1-61
Copying Components .....	1-65
Moving Components .....	1-68
Adding Nets .....	1-69
Adding Buses .....	1-72
Adding Labels .....	1-74
Saving the Schematic .....	1-77
Creating the ORBLK2 Schematic .....	1-78
Changing AND2 Components to OR2 Components.....	1-82
Saving the ORBLK2 Schematic .....	1-86
Completing the ALU Schematic .....	1-87
Making the CALC.1 Schematic Visible .....	1-87
Pushing into the ALU Symbol's Schematic.....	1-89
Placing the ANDBLK2 and ORBLK2 Symbols.....	1-90
Placing the FD4CE Component.....	1-94
Adding Nets, Buses, and Labels.....	1-96
Adding Symbol Labels .....	1-97
Saving the ALU.1 Schematic .....	1-99
Viewing the OSC_3K or OSC_7K Schematic.....	1-100
Exchanging Components.....	1-103
Controlling Layout from the Schematic .....	1-104
Adding the LOC Attribute .....	1-105
Adding Flags to Nets.....	1-108
Adding the FAST and SLOW Attributes.....	1-111
Using IOB Flip-Flops.....	1-114
Functional Simulation.....	1-116
Creating the Simulation Network .....	1-116
Adding Signals and Vectors to the Waveform .....	1-120
Adding Signals to the Waveform .....	1-120
Adding Vectors to the Waveform .....	1-121



---

Defining the Design Inputs.....	1-123
Defining a Clock .....	1-123
Defining Input Values .....	1-124
Simulating the Design Inputs .....	1-125
Invoking PROwave .....	1-127
Changing the Display Radix .....	1-129
Simulating the Calc Design.....	1-131
Loading 1111 to the ALU Register .....	1-132
Changing the Radices for PROcapture Display .....	1-134
Pushing 1111 to the STACK Register .....	1-136
Viewing the Waveforms.....	1-138
Re-Creating Previous Simulation.....	1-140
Implementing the Calc Design .....	1-140
Invoking the Design Manager .....	1-140
Creating the Calc Implementation Project and Its Initial Translation.....	1-142
Implementing the FPGA Design .....	1-152
Setting General FPGA Options .....	1-154
Setting Advanced FPGA Options .....	1-157
Invoking the Flow Engine .....	1-161
Implementing the EPLD Design.....	1-165
Setting Advanced EPLD Options.....	1-166
Invoking the Flow Engine .....	1-169
Timing Simulation .....	1-173
Creating the Simulation Network File .....	1-173
Invoking PROwave .....	1-177
Comparing the Functional and Timing Simulation Files .....	1-179
Zooming the Waveform Files.....	1-182
Obtaining a Transition Time .....	1-187
In the Calc.wfm File .....	1-188
In the Calct.wfm File .....	1-189
Obtaining a Delta Time.....	1-190
Downloading an FPGA Design .....	1-191
Using a Demonstration Board.....	1-192
Connecting the Cable for Download .....	1-192
FPGA Demonstration Board .....	1-194
XC3000A Demonstration Board .....	1-194
Downloading the Bitstream.....	1-195
Testing the Design.....	1-196

## Chapter 2 X-BLOX Tutorial

Before Beginning the Tutorial.....	2-1
Required Software .....	2-1
Preparing the Design .....	2-2
Modifying the Design.....	2-3
Adding X-BLOX Modules to CALC .....	2-3
Viewing the ALU_BLOX Schematic .....	2-3
Completing the ALU_BLOX Schematic .....	2-5
Understanding X-BLOX Buses.....	2-5
Using BUS_DEF Symbols .....	2-6
Completing the Bus Definition.....	2-7
Saving Your Changes .....	2-9
X-BLOX Symbol Library .....	2-9
X-BLOX Symbol Examples .....	2-9
X-BLOX Schematics .....	2-10
Functional Simulation.....	2-11
Creating the Simulation Schematic.....	2-11
Examining XSimMake Output .....	2-13
Performing a Functional Simulation .....	2-15
Implementing the Calc Design .....	2-18
Translating the Netlist .....	2-18
Examining XMake Netlist Translation Output .....	2-18
Creating a Routed Design.....	2-19
Examining the Flow Engine History File .....	2-20
Timing Simulation.....	2-22
Creating the Simulation Network .....	2-22
Examining XSimMake Output .....	2-22
Performing a Timing Simulation .....	2-23
Verifying CALC on the Demonstration Board.....	2-23
Further Reading .....	2-23

## Chapter 3 Xilinx ABEL Tutorial

Before Beginning the Tutorial.....	3-1
Required Software .....	3-1
Preparing the Design .....	3-2
Viewing Stat_abl.abl.....	3-3
Simulating Within Xilinx ABEL.....	3-9
Compiling STAT_ABL.ABL .....	3-10
Including STAT_ABL in the CALC Design .....	3-10
Creating a Symbol for STAT_ABL .....	3-11

Viewing the STAT_ABL Symbol .....	3-14
Viewing the STAT_ABL Schematic .....	3-14
Verifying the Symbol Type.....	3-15
Verifying the Symbol Attributes .....	3-16
Functional Simulation.....	3-17
Creating the Simulation Schematic.....	3-18
Examining XSimMake Output.....	3-19
Performing a Functional Simulation.....	3-21
Implementing the CALC Design.....	3-22
Translating the Netlist .....	3-22
Examining XMake Netlist Translation Output .....	3-23
Creating a Routed Design .....	3-24
Examining the Flow Engine History File .....	3-25
Timing Simulation .....	3-26
Creating the Simulation Netlist .....	3-26
Examining XSimMake Output.....	3-27
Performing a Timing Simulation.....	3-27
Verifying CALC on the Demonstration Board .....	3-28
Further Reading .....	3-28

## **Chapter 4 XACT-Performance and Timing Analyzer Tutorial**

Before Beginning the Tutorial .....	4-2
Required Software .....	4-2
Preparing the Design .....	4-2
Understanding XACT-Performance .....	4-3
Grouping Symbols with TNM Attributes .....	4-4
TNMs on Logic Primitives.....	4-4
TNMs on Higher-Level Macro Symbols.....	4-4
TNMs on Nets to Tag Flip-Flops.....	4-5
Grouping Symbols by Predefined Groups .....	4-5
Simplifying Symbol Grouping.....	4-5
Combining Groups with the TIMEGRP Symbol .....	4-6
Joining Two or More Groups into One.....	4-6
Using the EXCEPT Statement.....	4-6
Triggering on RISING or FALLING Clock Edges.....	4-7
Forming Groups by Output Net Name.....	4-7
Attaching Timing Specifications with the TIMESPEC Symbol .....	4-8
Deciding When to Use XACT-Performance .....	4-9
Setting Default Timing Requirements .....	4-10
Adding a TNM Attribute .....	4-10
Entering Default Timing Specifications .....	4-11

Adding Timing Constraints to Specific Paths .....	4-14
Defining TNM Groups .....	4-14
Defining the INFFS Group .....	4-14
Defining the STACKER Group (XC4000 Family Only) .....	4-15
Defining the STACKER Group (XC3000A Only) .....	4-16
Defining the ALUFF Group .....	4-17
Defining the CTLFF Group .....	4-17
Defining the STFF Group.....	4-18
Grouping Using TIMEGRP.....	4-18
Specifying TIMESPEC Constraints.....	4-20
Making a Final Check .....	4-21
Implementing the Calc Design .....	4-23
Translating the Netlist .....	4-24
Examining Translation Output.....	4-24
Creating a Routed Design.....	4-25
Examining the Implementation Output.....	4-26
Using the Timing Analyzer .....	4-27
Analyzing the Calc Design .....	4-29
Invoking the Timing Analyzer.....	4-29
Disabling False Paths .....	4-29
Resetting Path Filters.....	4-31
Displaying Current Settings .....	4-32
Generating a Performance Summary Report .....	4-32
Generating a Performance to TimeSpecs Report.....	4-33
Generating a Detailed Path Report.....	4-39
Reporting by Path Type .....	4-40
Reporting by Sources and Destinations .....	4-42
Using the Console Window.....	4-44
Creating Macros.....	4-44
Further Reading .....	4-45

# ***Viewlogic Tutorials***

***PROcapture and PROsim  
Tutorial***



# Chapter 1

## PROcapture and PROsim Tutorial

---

This tutorial guides you through a typical field-programmable gate array (FPGA) and erasable programmable logic device (EPLD) design procedure from schematic entry to completion of a functioning device. It uses PRO Series, Viewlogic's Windows-based toolset for design entry and simulation on personal computers (PCs). The tutorial uses a design called Calc, a 4-bit processor with a stack. In the first part of the tutorial, you use PROcapture, the PRO Series schematic entry tool, to create the schematics and symbols for the Calc design. Next, you use PROsim, the PRO Series simulator, to perform a functional simulation on it. In the third step, you use the Xilinx Design Manager to implement the design. Finally, you verify the design in PROsim using worst-case delays.

To install the tutorial, see the "Getting Started" section in this chapter.

### Introduction

This section provides you with some basic information about the tutorial: the devices to which it applies, approximately how long it will take to complete, and a description of the Calc design.

### Devices

The procedures described in this tutorial apply to both FPGAs and EPLDs; differences are noted where applicable. Although the tutorial describes how to create both FPGA and EPLD designs, all figures illustrate the FPGA version of Calc except where noted.

### Length

Performed without interruption, the tutorial takes approximately five or six hours to complete. If you need to stop the tutorial at any time,

be sure to save the work that you have done by selecting the **File** → **Save** command. Then exit PRO Series either by selecting **File** → **Exit** from the menu or by typing **quit** at the command line.

## Design Description

The processor in the Calc design 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 bar LEDs.

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 four-nibble storage device implemented with flip-flops in the device-independent design.
- **OSC\_3K, OSC\_7K**  
These modules are used in XC3000A and XC7000 designs, respectively. OSC\_3K generates a clock signal using the RC oscillator circuit on the FPGA (XC3000A/XC4000) and XC3000A demonstration boards. OSC\_7K is the equivalent oscillator block for EPLDs.
- **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\_TRU  
This module implements the connections to the 7-segment display on the XC3000A demonstration board.
- LED\_TRU  
The value at the top of the stack is displayed in binary on the LED bank of the XC3000A demonstration board.
- 7SEG\_INV  
This module implements the connections to the 7-segment display on the FPGA demonstration board.
- LED\_INV  
The value at the top of the stack is displayed in binary on the LED bank of the FPGA demonstration board.

## Getting Started

This section describes how to configure your PC to use the PRO Series tutorial, install the tutorial, start Xilinx PROflow, and set up the directories and initialization files for the Calc project.

## Required Software

This tutorial assumes that you are using the following versions of the development software:

- PRO Series release 6.0 or later
- Xilinx/Viewlogic Interface and Libraries: WIR2XNF V6.0.x and XNF2WIR V6.0.x or later
- XACTstep Development Software: DS-502 V6.0.x or later for FPGAs; DS-550 V6.0.x for EPLDs on PCs; DS-550 V5.2.x for EPLDs on workstations

**Note:** The instructions in this tutorial are written for the PRO Series user, and the PRO Series environment is shown in the figures of this document. However, you can use the Xilinx interface programs with

any current Viewlogic software, including Powerview V5.x or Workview PLUS V5.x. Workview 4.1.3a is also supported but does not run under the Windows environment.

## Before Beginning the Tutorial

Before beginning the tutorial, you must set up your PC to use the Viewlogic and XACTstep Development System software.

1. Verify that your system is properly configured. Consult the *Getting Started & Installation Guide* for instructions on setting up your machine to run the software.
2. Install one of the following sets of software. Each of these options includes Viewlogic PRO Series, the Xilinx/Viewlogic Interface and Libraries, and an XACTstep Development System.
  - Base (DS-VLS-BAS-PC1), Standard (DS-VLS-STD-PC1), or Extended (DS-VLS-EXT-PC1) Stand-Alone (/S) Package Solutions for Viewlogic
  - or
  - Viewlogic PROcapture Schematic Editor, Interface, and Libraries (DS-390); and/or PROsim Simulator (DS-290); and XACTstep Development System (DS-502) for FPGAs, and/or XACTstep Development System (DS-550) for EPLDs
  - or
  - Viewlogic PRO Series V6.0 or later
  - and
  - Viewlogic Interface and Libraries (DS-391) and XACTstep Development System (DS-502) for FPGAs and/or XACTstep Development System (DS-550) for EPLDs; or Base (DS-VL-BAS-PC1) or Standard (DS-VL-STD-PC1) Interface Package Solution for Viewlogic
3. Verify that the following variables are set in your autoexec.bat file. It is assumed that you have loaded the software noted in the previous step to the c:\proser and c:\xact directories on your PC. If the software has been installed in different areas, modify the following Set statements accordingly. See the *Getting Started & Installation Guide* for additional information on system setup.

- The PATH variable sets the overall executable search path. It must include the directories where the PRO Series and XACTstep Development System software have been installed. Use this syntax:

```
PATH=other_paths;c:\XACT;c:\PROSER;other_paths
```

**Note:** The PATH variable cannot include any previous version of either the XACTstep or Viewlogic software. Be sure to remove all paths to older software.

- The XACT variable is used by the XACTstep and PRO Series software to locate data files. It must include the directory where the XACTstep Development System resides and the directory that contains the \unified directory, where the Unified Libraries reside. Use the following syntax:

```
SET XACT=C:\XACT;C:\PROSER
```

**Note:** As with the PATH variable, you can set multiple paths using a semicolon (;) between the paths. In the syntax just given, the XACTstep software is located in c:\xact, and the \unified directory is located in c:\proser. Because both paths are needed, they have been concatenated into a single path using a semicolon.

- The WDIR variable sets the data file search path for the PRO Series software. It must include a directory to which you can write. Use this syntax:

```
SET WDIR=C:\PROSER\STANDARD
```

- The SYSPLT variable sets the PRO Series plotting directory. Use this syntax:

```
SET SYSPLT=C:\PROSER\STANDARD
```

## Installing the PRO Series Tutorial

The tutorial files are optionally installed when you install the Xilinx PRO Series software. This tutorial can be used with either XC3000A or XC7000 designs. If you have already installed the software but are not sure whether you specified the tutorial installation, check for the \calc3ka directory for XC3000A designs or the \calc7k directory for XC7000 designs under the c:\proser\tutorial\vwlogic\procalc directory. The \calc3ka directory contains the tutorial files needed to perform the tutorial for XC3000A designs, and the \calc7k directory

contains the files needed to perform the tutorial for XC7000 designs. It is recommended that you copy these files into another directory before performing the tutorial to preserve the original files.

For XC3000A designs, use the Windows File Manager to copy the \calc3ka directory to another directory such as c:\user\calc. For XC7000 designs, copy the \calc7k directory to this other directory.

**Note:** The rest of the tutorial refers to the c:\user\calc directory as the design directory.

## **Starting Xilinx PROflow**

This tutorial uses Xilinx PROflow to implement the Calc design. On the basis of information such as the design type, part type, family, and the schematic components used, PROflow determines which options are available and which programs it must run to process the design correctly. Every tool and process step is executed by or invoked from PROflow. In addition to managing the processing of your design, PROflow also seamlessly integrates all the tools needed to enter, implement, simulate, and download your design.

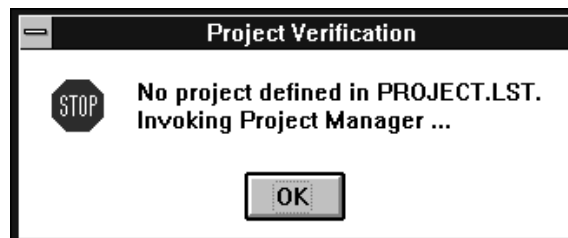
PROflow even handles design maintenance. When you initially select the tutorial design within PROflow, it creates a Viewlogic design project and defines the associated libraries.

1. To open Xilinx PROflow, double-click on the Xilinx PROflow icon, shown in Figure 1-1, in the Program Manager XACTstep program group.



**Figure 1-1 Xilinx PROflow Icon**

Selecting the Xilinx PROflow icon for the first time brings up a warning message, shown in Figure 1-2, stating that there is no project defined.



**Figure 1-2 Project Verification Warning Message**

2. Click on **OK**.

The message box closes, and the PRO Series Project Manager appears, as shown in Figure 1-3. The next section describes how to use it to create the Calc project.

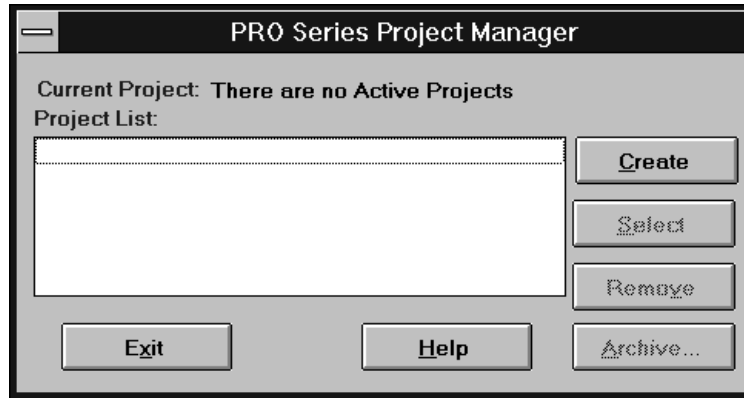


Figure 1-3 PRO Series Project Manager

## Defining the Calc Project

The Viewlogic tools use the concept of projects to keep track of designs. A project is a working directory that contains the sub-directories and data files for a given design. Projects can even contain several designs of the same type, for example, a single project containing several XC3000A designs; however, it is recommended that each project contain only one design. The project containing the design actively being processed is known as the current project.

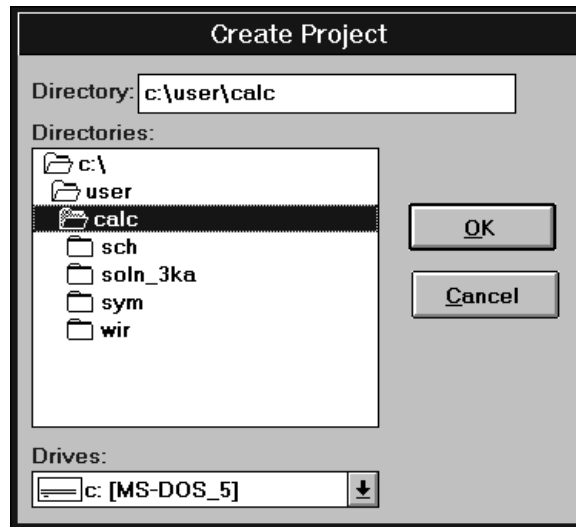
The definition of the Calc project involves three steps: creating a project in the c:\user\calc directory, generating the needed program initialization files and profiles, and selecting the CALC.1 schematic as the top-level schematic used when processing.

### Creating the Calc Project

The PRO Series Project Manager allows you to select, create, and remove projects. You must add the tutorial directory c:\user\calc to the Project List box and then select it.

1. Click on **Create** in the PRO Series Project Manager.

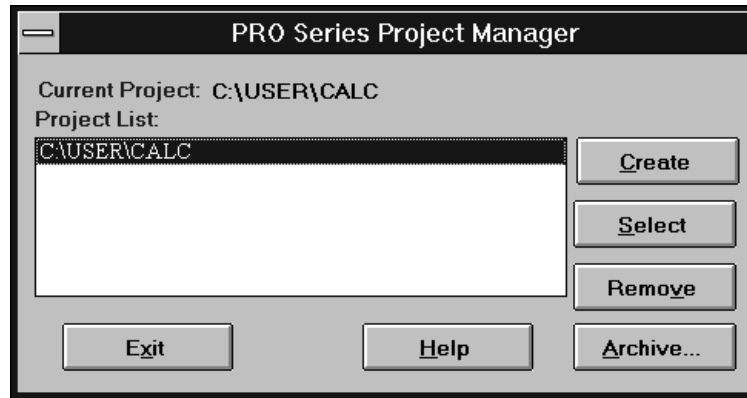
The Create Project dialog box appears, as shown in Figure 1-4.



**Figure 1-4 Create Project Dialog Box**

2. Select the c:\user\calc tutorial directory by double-clicking in the Directory list box until the field is correctly updated, as illustrated in Figure 1-4.
3. Click on **OK**.

The Create Project dialog box closes, and the PRO Series Project Manager opens showing the c:\user\calc directory in the Project List box. Figure 1-5 displays the updated Project Manager.



**Figure 1-5 Updated PRO Series Project Manager**

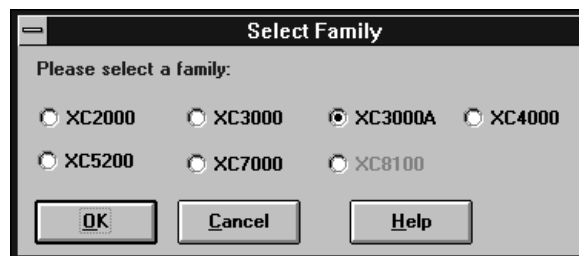
Now that you have created an entry for the tutorial directory, all that remains to do is to select it as the current project.

4. Click on **select**, or double-click on the project name.

When you select `c:\user\calc`, the PRO Series Project Manager automatically creates the necessary initialization files.

5. To close the PRO Series Project Manager, click on **Exit**.

When you close the PRO Series Project Manager, the Select Family dialog box appears, as shown in Figure 1-6.



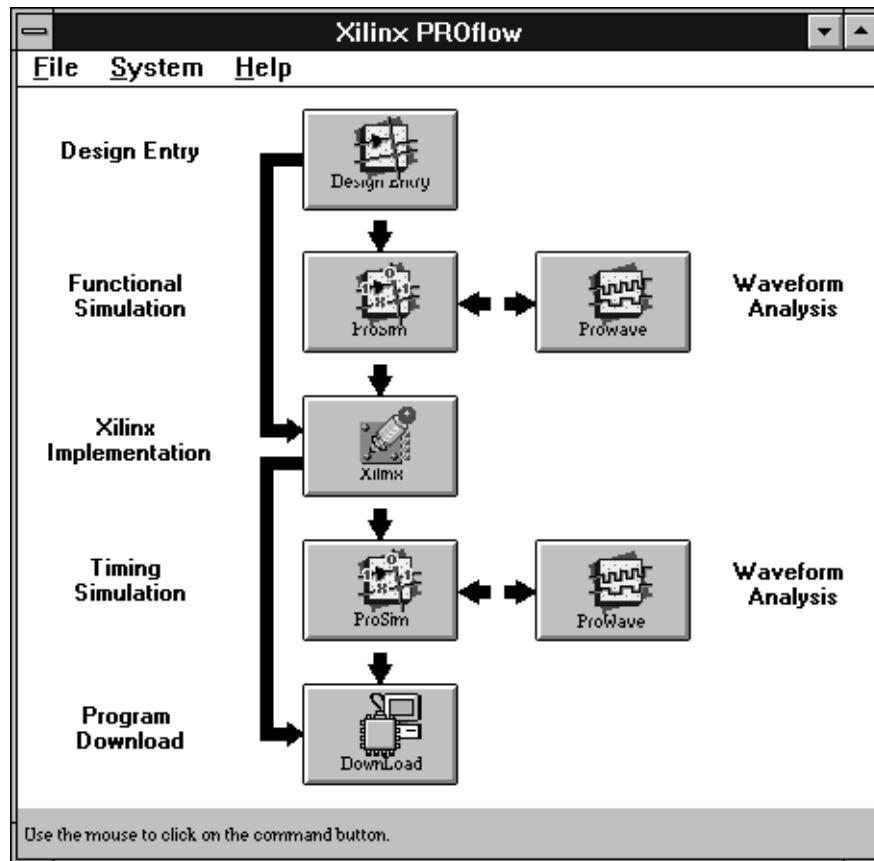
**Figure 1-6 Select Family Dialog Box**

6. Select **xc3000A** if you are performing the FPGA tutorial or **xc7000** if you are performing the EPLD tutorial.



7. Click on OK.

Xilinx PROflow comes up, as shown in Figure 1-7.



**Figure 1-7 Xilinx PROflow Window**

To guide you through processing your design, PROflow only allows you to enter stages in the design flow if the files that are needed for that step are present. For instance, if you were to click on the Xilinx Implementation icon, you would receive the message shown in Figure 1-8.



**Figure 1-8 Xilinx Implementation Warning Message**

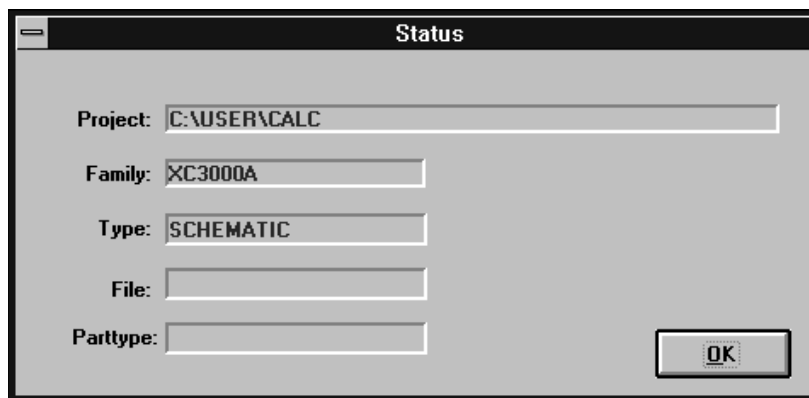
Because the design to be processed has not been specified, PROflow does not allow you to enter the Xilinx Implementation section.

### Obtaining Design Status

To see PROflow's initial design status, follow these steps.

1. Select the **File** → **Status** command.

This command displays the Status dialog box, shown in Figure 1-9, which displays the currently known information about the design. As you enter and process the Calc design, additional information appears in the remaining fields.



**Figure 1-9 Status Dialog Box**

2. Click on **OK**.

The Status dialog box closes.

### **Selecting the CALC.1 Schematic**

In the previous sections, you defined the Calc project and created the necessary initialization files. Now you must select the Calc design's top-level schematic, CALC.1, as the current design.

1. In PROflow, click on the Design Entry icon, shown in Figure 1-10.



**Figure 1-10 Design Entry Icon**

The Design Entry dialog box now appears, as shown in Figure 1-11.

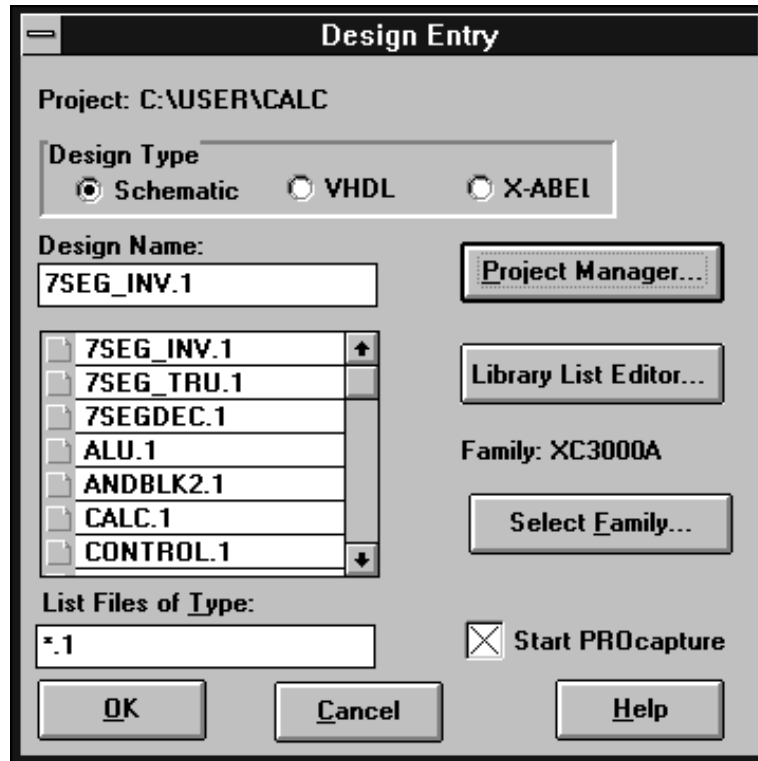


Figure 1-11 Design Entry Dialog Box

The project displayed at the top of the dialog box is `c:\user\calc`. The List Files of Type field displays the default filter, `*.1`.

Viewlogic allows a schematic to contain multiple sheets. The sheets are saved to the project directory's `sch` directory, where each sheet is a separate file. The extension of the file is the actual sheet number. If Calc's top-level schematic had two sheets, the `sch` directory would contain a `CALC.1` and a `CALC.2` file. The current string in the List Files of Type field in the Design Entry dialog box restricts the display to only the first sheet of each schematic. If you wanted to see all the second sheets, you could change the List Files of Type field to `*.2`. To see all sheets, you could enter `*.*` in the List Files of Type field.

**Note:** The default of the Design Type field is Schematic. You can also process VHDL designs if the PROsynthesis package is installed or XABEL designs if the XABEL package is installed. The Calc tutorial focuses on processing straight schematic designs.

There are two ways to select the top-level schematic, CALC.1: you can either type the name directly in the Design Name field, or click on the file in the design list box.

2. Click on the CALC.1 file in the design list box.

The CALC.1 file now appears in the Design Name field.

3. Click in the Start PROcapture check box to deselect it, as shown in Figure 1-12.



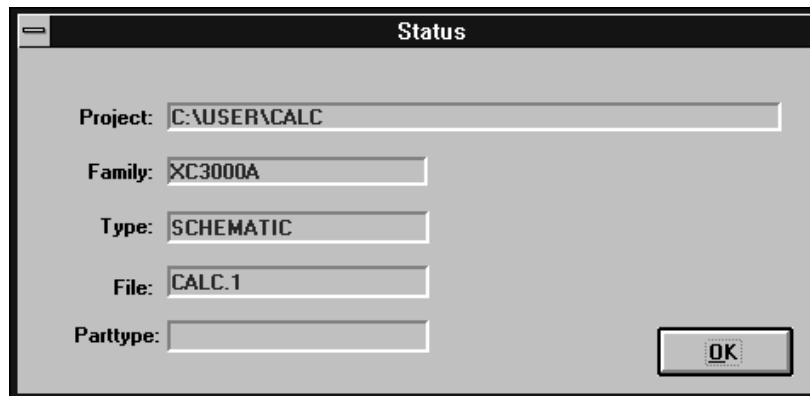
Figure 1-12 Disabling the Start PROcapture Check Box

Clicking on the Start PROcapture check box determines whether or not PROcapture is invoked when you click on the OK button. By default, Start PROcapture is selected. For now, you only want to select the design, so you must deselect the Start PROcapture check box.

4. Click on **OK**.

Because Start PROcapture is disabled, selecting OK closes the Design Entry dialog box and reactivates PROflow.

5. Select the **File** → **Status** command to display the Status dialog box, shown in Figure 1-13.



**Figure 1-13 Status Dialog Box**

Now the File field contains the selected design, CALC.1.

6. Click on **OK** to close the Status dialog box.

## **Navigating in PROcapture**

In creating a design, you typically use hierarchical levels to divide the design into more manageable sections. The top-level schematic for the Calc tutorial, CALC.1, has already been created for you. In this tutorial, you will add schematic information to the lower levels using the Viewlogic PROcapture program to create all schematics and symbols for a design. This section describes how to access PROcapture, adjust the colors of the PROcapture window, and move around the screen.

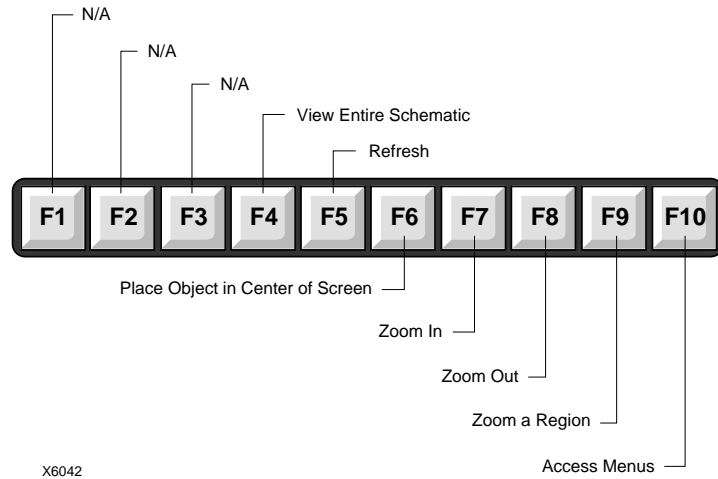
### **Mouse Buttons**

Mouse buttons perform the following functions in PRO Series:

- The left mouse button selects objects.
- The right mouse button cancels the current command mode. In addition, you use it to select multiple items; select the first item with the left mouse button and subsequent items with the right mouse button.

### **Function Keys**

The function keys in PRO Series are labeled F1, F2, F3, and so forth on your keyboard. They are assigned the functions shown in Figure 1-14.



**Figure 1-14 Default Function Keys**

## Starting PROcapture

To open the CALC.1 schematic in PROcapture, follow these steps.

1. In PROflow, click on the Design Entry icon, shown in Figure 1-15.



**Figure 1-15 Design Entry Icon**

2. As demonstrated in Figure 1-16, click in the box next to the Start PROcapture field.



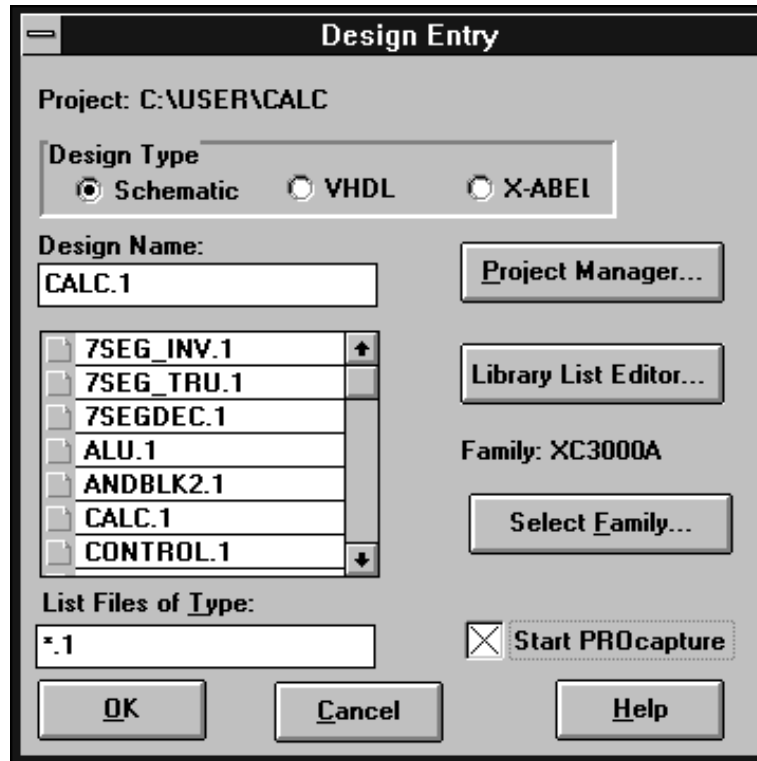


Figure 1-16 Selecting the Start PROcapture Check Box

3. Click on OK.

Because you enabled Start PROcapture, clicking on OK closes the Design Entry dialog box and invokes PROcapture on the selected CALC.1 schematic, as shown in Figure 1-17.

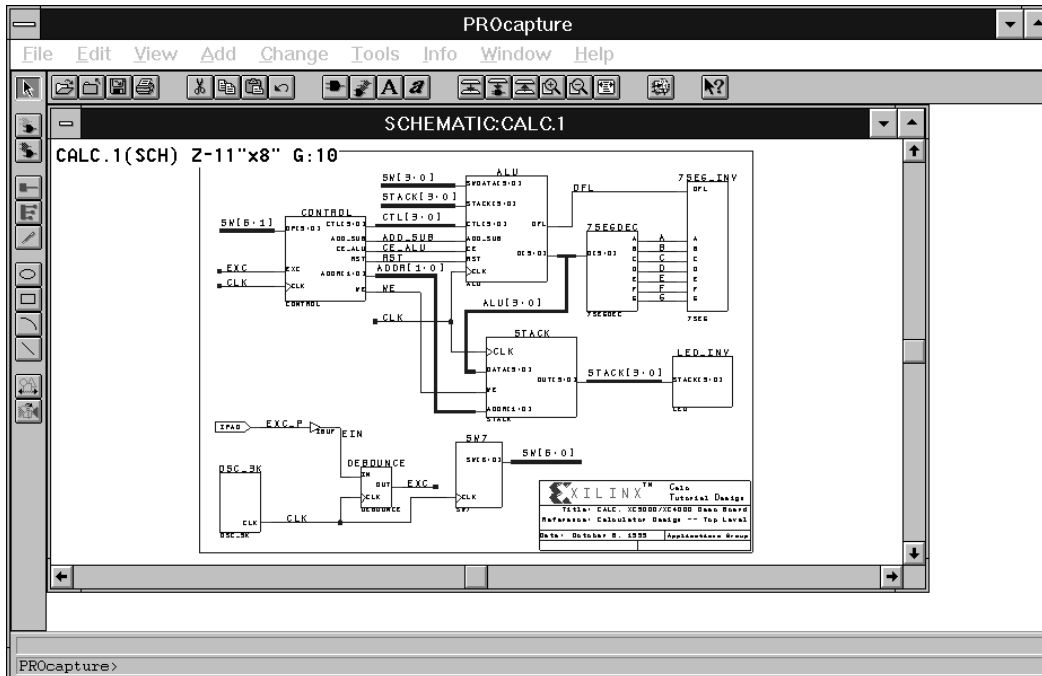


Figure 1-17 CALC.1 Schematic in PROcapture Window

## Changing the PROcapture Window Colors

You can change the PROcapture color settings so that viewing a schematic will be easier. If the background of your schematic is white and you want to change the color palette, proceed with the steps in this section; otherwise, skip to the next section.

1. To change the color palette, select **Change** → **PROcapture Colors**.

The PROcolor Manager dialog box appears, as shown in Figure 1-18.

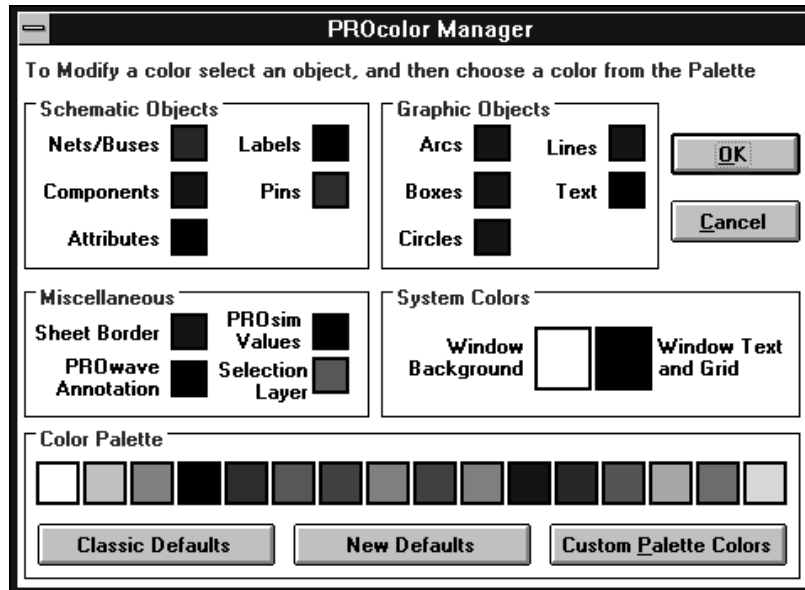


Figure 1-18 PROcolor Manager Dialog Box

2. Click on **Classic Defaults**.

This setting changes the color configuration for the various objects. The window background, window text, and grid toggle from white to black and vice versa.

3. Click on **OK** to close the PROcolor Manager dialog box and reactivate PROcapture.

An information dialog box comes up to inform you that the changes to the color palette will not take effect until you close all schematics and symbols.

4. Select **File** → **Close** to close the CALC.1 schematic.
5. Select **File** → **Open** to re-open the CALC.1 schematic.

The schematic should resemble the one shown in Figure 1-19. The colors onscreen are now changed.

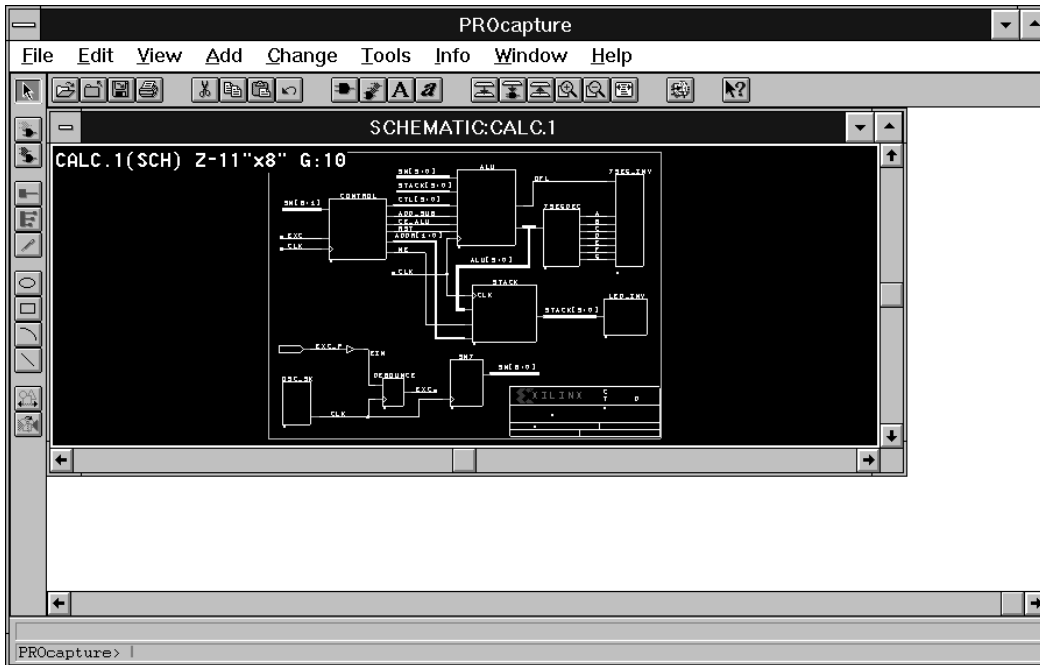


Figure 1-19 CALC.1 Schematic in PROcapture Window

## Moving Around the Screen

PROcapture works like any other Windows program; it allows you to view and work in several different designs. The concept of working in several documents at once is known as the Multiple Document Interface, or MDI. MDI enables you to bring up multiple schematic windows and arrange them in the workspace in any fashion.

You can also zoom and pan around a schematic in PROcapture. Zooming allows you to view an entire schematic or focus on a particular section. You can select the zoom commands from the View menu, from the toolbar, or by pressing the designated function key.

You can also make icons of schematics when the schematics are not needed. Making an icon of a schematic does not close the schematic; it merely turns the window into an icon. Later, when you need the schematic, you can double-click on the icon to re-display it.

Table 1-1 at the end of this chapter summarizes the PRO Series functions, toolbar icons, menu commands, command line shortcuts, and function keys used in the rest of the tutorial.

## Panning

You can familiarize yourself with the CALC.1 schematic by panning around the window. Panning is the process of obtaining a panoramic view of the screen by using one point as the center of the view.

To pan across the screen, first move the cursor to the location that will become the center of the new view, then press the **F6** function key. This step centers the edit area around the location of the cursor but does not move the cursor. Repeat moving the cursor and pressing F6 to move around the screen.

## Zooming

Zooming magnifies or shrinks the view onscreen. You can either view the entire schematic or focus on one portion of it. All of the zoom commands are dynamic, which means you can use them while you are in the middle of another command.

- Click on the up arrow, shown in Figure 1-20, in the upper right corner of the CALC.1 schematic window to fill the entire workspace with the CALC.1 schematic window. Figure 1-21 shows the expanded window.

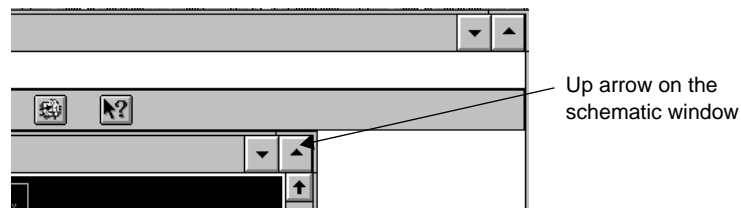
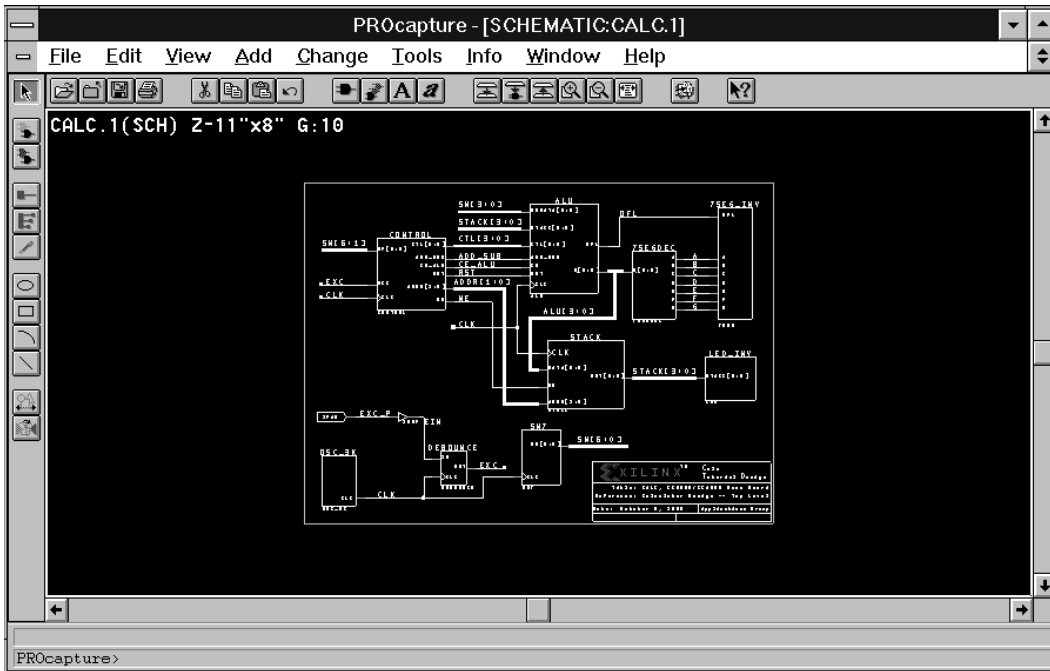


Figure 1-20 Up Arrow on the CALC.1 Schematic Window



**Figure 1-21 Expanded Schematic Window**

Notice that when the window is enlarged to fill the work space, the schematic does not fill the entire area of the new schematic window.

- To view the entire sheet — that is, to expand the schematic to fill the entire window — select the **View** → **Full** command.

*Keyboard Shortcut:* You can execute the View → Full command by pressing the **F4** function key.

*Toolbar Shortcut:* You can execute the View → Full command by clicking on the Full toolbar icon, shown in Figure 1-22.



**Figure 1-22 Full Toolbar Icon**

- To zoom in, or magnify the view of the design, select the **View → In** command.

*Keyboard Shortcut:* You can execute the View → In command by pressing the **F7** function key.

*Toolbar Shortcut:* You can execute the View → In command by clicking on the In toolbar icon, shown in Figure 1-23.



**Figure 1-23 In Toolbar Icon**

- To zoom out, or shrink the view of the design, select the **View → Out** command.

*Keyboard Shortcut:* You can execute the View → Out command by pressing the **F8** function key.

*Toolbar Shortcut:* You can execute the View → Out command by clicking on the Out toolbar icon, shown in Figure 1-24.



**Figure 1-24 Out Toolbar Icon**

- To zoom a particular region, select the **View → Region** command. Define the area to be zoomed by pressing and holding the left mouse button in the upper left corner of the area and the left mouse button in the lower right corner. Release the left mouse button.

*Keyboard Shortcut:* You can execute the View → Region command by pressing the **F9** function key. You define the upper left corner when you select the command and the lower right corner when you press the left mouse button.

Practice these functions by magnifying the ALU block in the center of the screen, as shown in Figure 1-25.

1. To place the ALU block in the center of the screen, point the mouse at the middle of the ALU block and press the **F6** function key.
2. Use the **View** → **In** and **View** → **Out** commands to magnify the ALU block until it fills the schematic. You can also use the **View** → **Region** command.

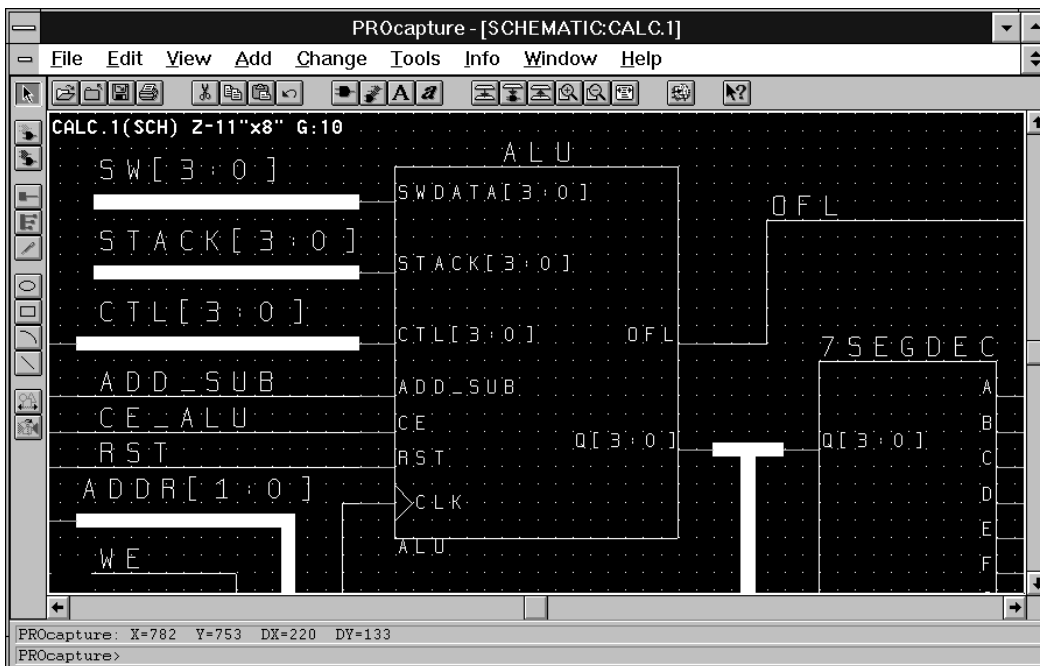


Figure 1-25 Zoomed ALU Block



## Making Icons of Schematics and Symbols

Use the following procedure to make an icon of the CALC.1 schematic window.

1. When the schematic window occupies the entire screen, the name of the schematic is shown in the PROcapture title bar. You can reduce the size of the CALC.1 schematic by pressing the up/down arrow button to the right of the Help menu.
2. Make an icon of the CALC.1 schematic window by pressing the down arrow button in the upper right of the CALC.1 schematic window. Figure 1-26 illustrates the resulting icon.

To re-open the CALC.1 schematic window, you can double-click on the icon; however, for this part of the tutorial, leave the CALC.1 schematic window as an icon.

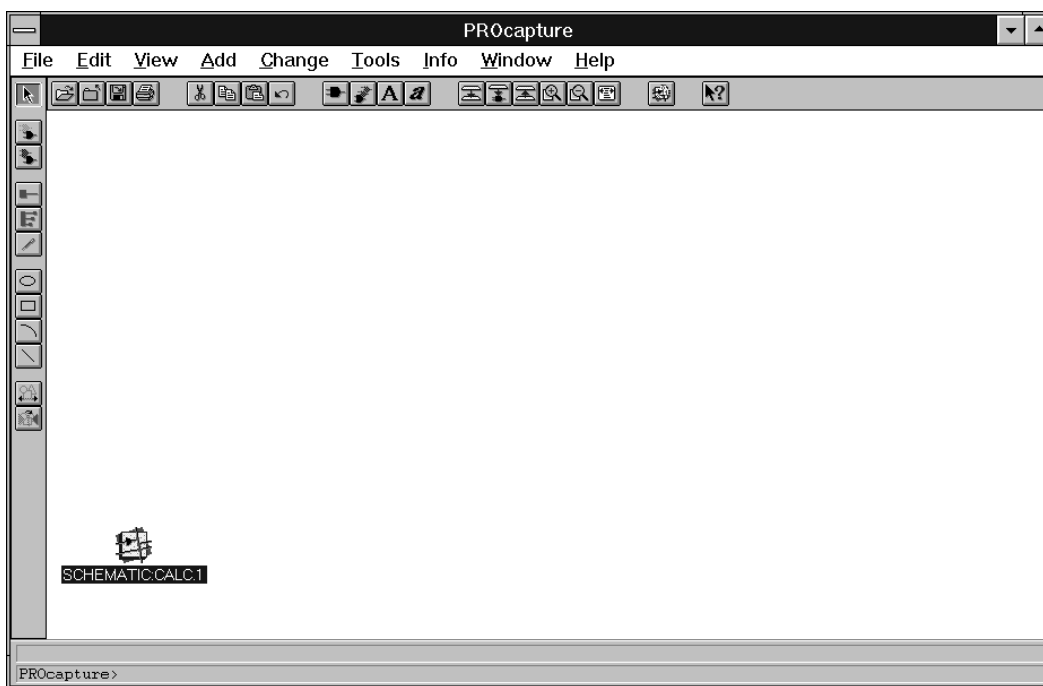
















Figure 1-26 CALC.1 Icon







## PROcapture Command Summary



The following table summarizes the commands used in this chapter and in the rest of the tutorial.

**Table 1-1 PROcapture Commands**

Description	Toolbar Icon	Menu Command	Function Key	Command Line Entry
Obtain context-sensitive help		Help → Help	None	<code>help.↵</code>
Cancel current command		None	None	Esc key
View entire schematic		View → Full	F4	<code>full.↵</code>
Zoom in		View → In	F7	<code>in.↵</code>
Zoom out		View → Out	F8	<code>out.↵</code>
Zoom region	None	View → Region	F9	<code>zoom.↵</code>
Place object in center of screen	None	None	F6	None
Refresh screen	None	View → Refresh	F5	<code>refresh.↵</code>
Push into schematic view		View → Push Into Schematic	None	<code>psc.↵</code>

Description	Toolbar Icon	Menu Command	Function Key	Command Line Entry
Push into symbol view		View → Push Into Symbol	None	<b>psy</b> ↵
Pop out of schematic or symbol		View → Pop	None	<b>pop</b> ↵
Open a file		File → Open	None	<b>sym</b> ↵ for symbols <b>sch</b> ↵ for schematics
Save a file	None	File → Save	None	<b>wri</b> ↵
Save a file as		File → Save As	None	<b>writeto</b> ↵
Close the active window		File → Close	None	<b>wcl</b> ↵
Add a box		Add → Box	None	<b>box</b> ↵
Add a symbol pin		Add → Pin	None	<b>pin</b> ↵
Add a label	None	Add → Object Label	None	<b>la</b> ↵
Change an attribute		Change → Object Attributes → Dialog	None	<b>at</b> ↵

Description	Toolbar Icon	Menu Command	Function Key	Command Line Entry
Add text		Add → Text	None	text.↓
Add a component		Add → Component	None	com.↓
Add a net		Add → Net	None	ne.↓
Add a bus		Add → Bus	None	bu.↓
Make attributes invisible	None	Change → Object Attributes → Visibility → All Attrs Off	None	ain.↓
Make attributes visible	None	Change → Object Attributes → Visibility → All Attrs On	None	avi.↓
Select components	None	Edit → Select	None	sco.↓
Move an object		Edit → Move	None	m.↓
Copy selected objects to the buffer		Edit → Copy	None	bcop.↓

Description	Toolbar Icon	Menu Command	Function Key	Command Line Entry
Paste objects from the buffer		Edit → Paste	None	bpa.↵
Copy selected components		None	None	cop.↵
Change the selected components	None	Change → Component	None	cc.↵

## Creating Symbols

To create a hierarchical module, you must create a symbol, which is a graphic representation of the schematic block. In the ALU block, three symbols are missing. You can find one symbol, FD4CE, in the Unified Libraries supplied by Xilinx. You must create the ANDBLK2 and ORBLK2 symbols yourself using the instructions in this section.

### Creating the ANDBLK2 Symbol

To create the ANDBLK2 symbol, follow these steps.

1. To open a new symbol window, select the **File** → **Open** command, which displays the File Open dialog box.

*Keyboard Shortcut:* You can execute the File → Open command for a symbol by typing **sym.↵** on the PROcapture command line, as in the following example:

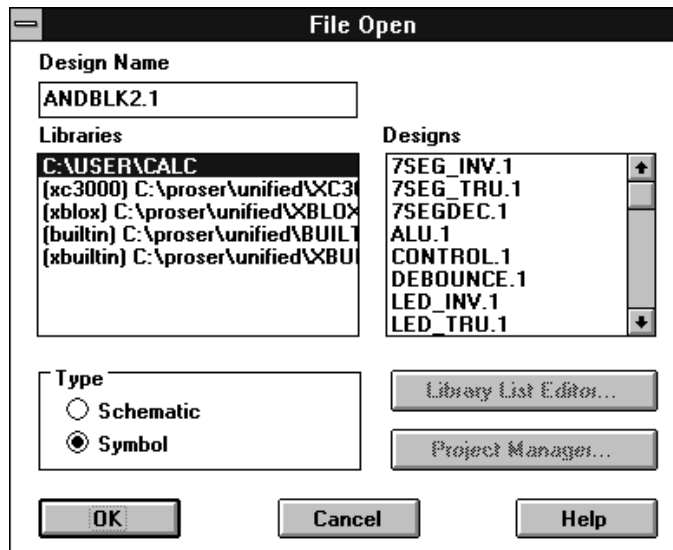
```
sym andblk2.↵
```

*Toolbar Shortcut:* You can execute the File → Open command by clicking on the Open toolbar icon, shown in Figure 1-27.



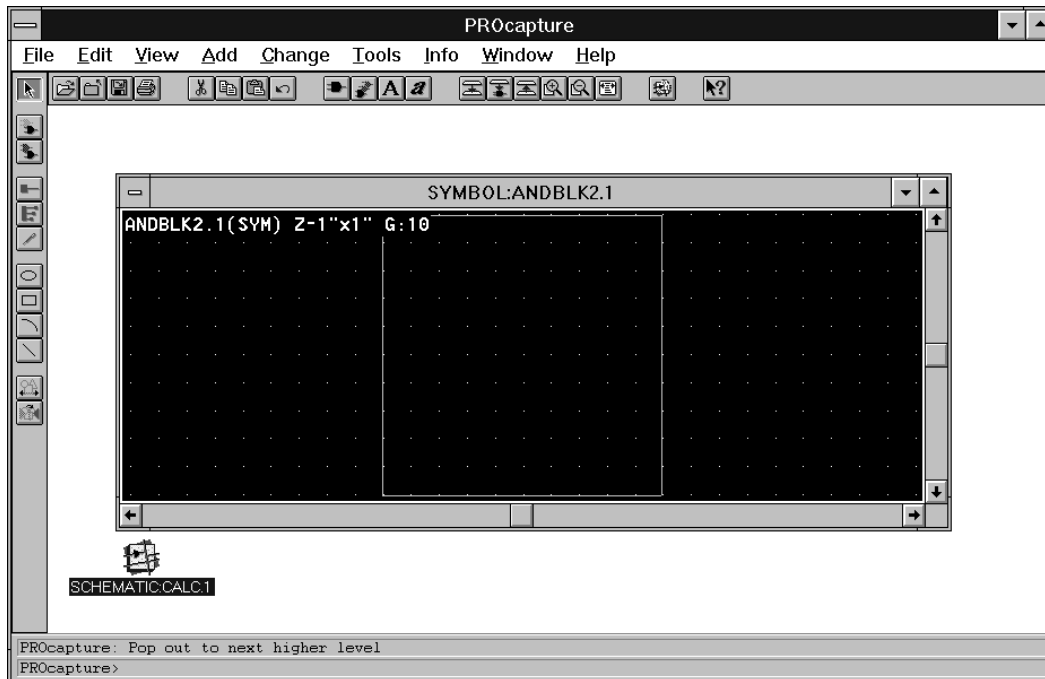
**Figure 1-27 Open Toolbar Icon**

2. In the Design Name field, type **ANDBLK2.1**.
3. Select the **Symbol** setting in the Type field, as shown in Figure 1-28. The default setting is Schematic.



**Figure 1-28 Selecting the Symbol Setting**

4. Click on **ok**.  
The File Open dialog box closes and the new symbol window opens, displayed in Figure 1-29.



**Figure 1-29 Symbol Window**

The ANDBLK2 symbol contains a box called a block sheet. It defines the perimeter of the symbol but does not show up on the screen when the symbol is placed in a design schematic. Only the elements that you add to the symbol are visible.

The initial size of the ANDBLK2 symbol, shown as the area defined by the block sheet, is 1 inch by 1 inch, or 100 x 100 grid units. For most symbols, it is necessary to enlarge or reduce this default size.

## Changing the Size of the Symbol

To change the size of the symbol, follow these instructions.

1. Select the **Change** → **Sheet Size** → **Z-WxH** command.

*Keyboard Shortcut:* You can execute the Change → Sheet Size → Z-WxH command by typing **zsi width height** on the PROcapture command line.

PROcapture prompts you first for the new block width at the PROcapture command line.

2. At the **Block width [100]** prompt, type **150**.

Now PROcapture prompts you for the new block height.

3. At the **Block height [100]** prompt, type **120**.

When you change the size of a symbol, the change is reflected in the text at the top of the symbol window. When you change the ANDBLK2 symbol, the text changes to the following:

```
ANDBLK2.1(SYM) Z-1.5"x1.2" G:10
```

The text tells you that the window is for a symbol called ANDBLK2.1, which has the dimensions of 1.5 inches by 1.2 inches. The G:10 notation indicates that the grid spacing is every tenth of an inch. You can change the grid spacing to any desired value, but 10 works well with the Unified Libraries because the pins on the various components are spaced by multiples of 10.

## Creating a Symbol Box

Most symbols have a visible frame or bounding box to which its pins are attached. You can create this box.

1. Select the **Add** → **Box** command.

The pointer changes to a crosshair.

*Keyboard Shortcut:* You can execute the Add → Box command by typing **box** on the PROcapture command line.

*Toolbar Shortcut:* You can execute the Add → Box command by clicking on the Box toolbar icon, shown in Figure 1-30.

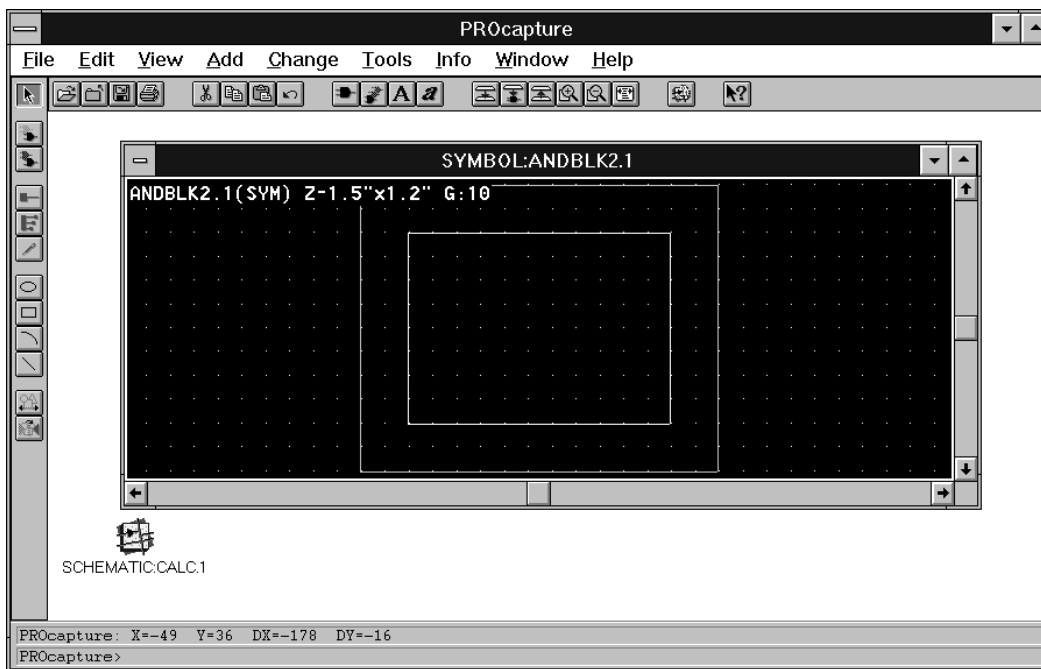




**Figure 1-30 Box Toolbar Icon**

2. Point the crosshair at the spot where you want to place the upper left corner of the box.
3. Holding down the left mouse button, drag the mouse to the spot where you want to place the lower right corner of the box.
4. Release the left mouse button.

Once the second point of the box is defined, the box is added to the symbol, as indicated in Figure 1-31.



**Figure 1-31 Creating a Symbol Box**

When you create a box, the status line gives the dimensions of the box. The Box toolbar icon remains selected after the box has been defined because PROcapture remains in Add Box mode. You can add multiple boxes in succession without having to re-invoke the Add → Box command.

5. To exit Add Box mode, press the Escape key, click the right mouse button, or click on the Clear toolbar icon, shown in Figure 1-32.



**Figure 1-32 Clear Toolbar Icon**

## Adding Pins

Once the symbol body is defined, the next step is to add pins to the symbol.

1. Select the **Add → Pin** command.

The pointer changes to a crosshair.

*Keyboard Shortcut:* You can execute the Add → Pin command by typing `pin↵` on the PROcapture command line.

*Toolbar Shortcut:* You can execute the Add → Pin command by clicking on the Pin toolbar icon, shown in Figure 1-33.

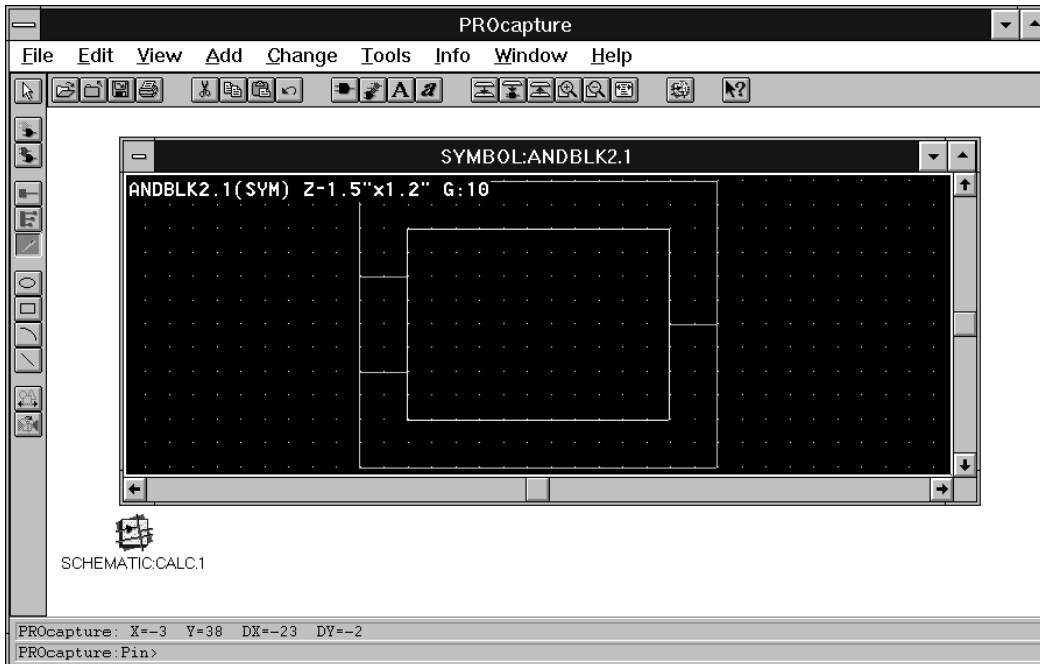


**Figure 1-33 Pin Toolbar Icon**

2. On the symbol body, point the crosshair at the spot where you want the pin to start.
3. Holding down the left mouse button, drag the mouse to the spot on the block sheet boundary where you want the pin to end.
4. Release the left mouse button.

- Repeat steps 2 through 4 to add the remaining two ANDBLK2 pins, as shown in Figure 1-34.

**Note:** You must begin the pin at the symbol body.



**Figure 1-34 Adding Pins to ANDBLK2**

When a pin is created, the status line gives the dimensions of the pin. The Pin toolbar icon remains depressed after the pin has been defined because PROcapture remains in Add Pin mode. You can add multiple pins in succession without having to re-invoke the Add → Pin command.

- To exit Add Pin mode, press the Escape key, click the right mouse button, or click on the Clear toolbar icon.

### Adding Pin Labels

Now you are ready to add labels to the pins on the symbol. These labels must exactly match the labels used for the same signals in the

corresponding schematic. For example, if there is a pin labeled “clock” on the symbol, there must be a net labeled “clock” in the symbol’s schematic.

1. Using the left mouse button, select the upper left pin.

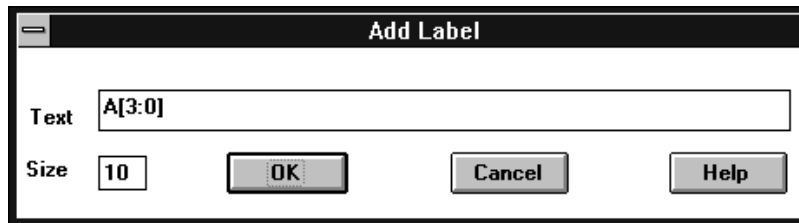
The pin color now changes.

2. Select the **Add** → **Object Label** command to bring up the Add Label dialog box.

*Mouse Shortcut:* You can execute the Add → Object Label command by double-clicking on the pin or component that is to be labeled.

*Keyboard Shortcut:* You can execute the Add → Object Label command by typing **l a** on the PROcapture command line once you select the desired pin.

3. Enter **A[3:0]** in the Text field of the dialog box, as illustrated in Figure 1-35.

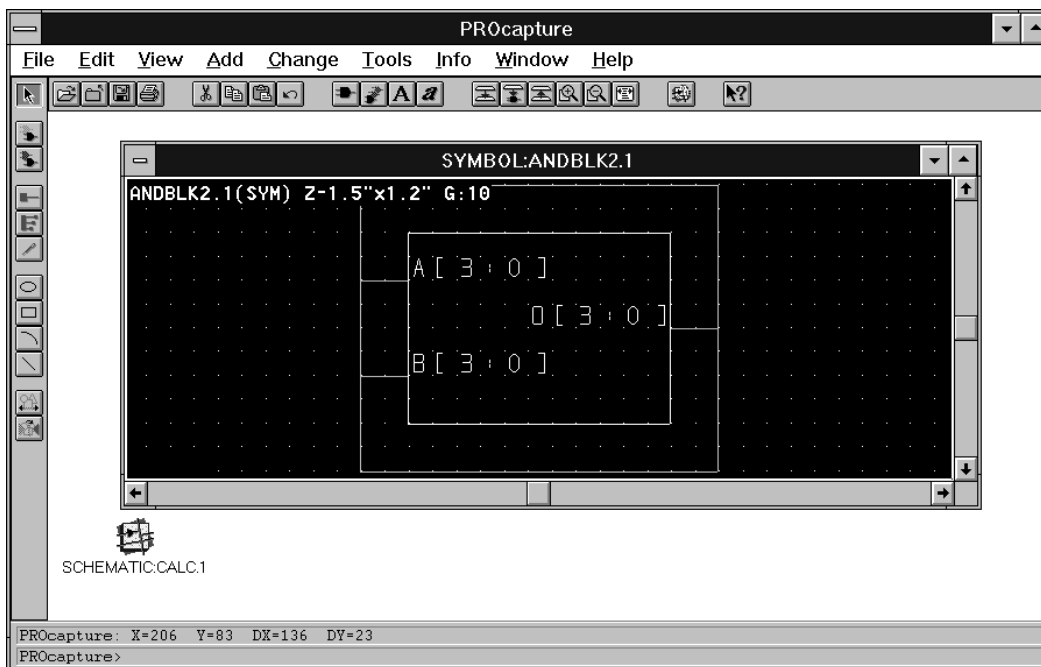


**Figure 1-35 Add Label Dialog Box**

4. Click on **OK**.

Selecting OK closes the Add Label dialog box and displays a PRO Series query box asking if the label should be expanded. This query box is only displayed when the label that you have entered has a width, for example, [3:0]. Because the label that has been entered will be connected to a bus in the symbol’s schematic, it should not be expanded into individual nets.

5. Click on **No**.  
The query box closes, and the symbol window is reactivated. An outline of the label, or bounding box, appears on the screen next to the pointer.
6. Point the mouse so that the label outline is in its desired location next to the upper left pin.
7. Click the left mouse button to place the label.
8. Repeat steps 1 through 7 to add labels to pins B[3:0] and O[3:0], as shown in Figure 1-36.



**Figure 1-36 Adding Pin Labels**

### Adding Pin Attributes

The next step in creating a symbol is to attach the PINTYPE attribute to each pin, giving the pins directionality. The most common values for the PINTYPE attribute are IN, OUT, and BI. PINTYPE attributes

were required in the older versions of the Viewlogic tools but are now optional except when there is no schematic for a symbol. (See the “Merging Non-Schematic-Based Modules” section of the “Design and Simulation Techniques” chapter of the *Viewlogic Interface Guide* for a discussion of this special case.)

Later in this tutorial, you will create an ANDBLK2 schematic, so the ANDBLK2 symbol does not require PINTYPE attributes. However, the tutorial now shows you how to assign PINTYPE attributes to the pins for cases in which you have no schematic for a symbol; it also offers you an opportunity to learn some useful Viewlogic commands.

### Using the Add Object Attribute Command

One way to add an attribute is to use the Add → Object Attribute command.

1. Using the left mouse button, select the upper left pin.  
Selecting the pin changes its color and displays the label's bounding box.
2. Select the **Add → Object Attribute** command.  
*Keyboard Shortcut:* You can execute the Add → Object Attribute command by typing **at**↵ on the PROcapture command line.
3. At the **Attribute Text String:** prompt, type **PINTYPE=IN**↵.  
When you enter the attribute, an outline of the attribute, or bounding box, appears on the screen next to the pointer.
4. Point the mouse so that the attribute outline is in its desired location next to the upper left pin.
5. Click the left mouse button to place the attribute, as shown in Figure 1-37.

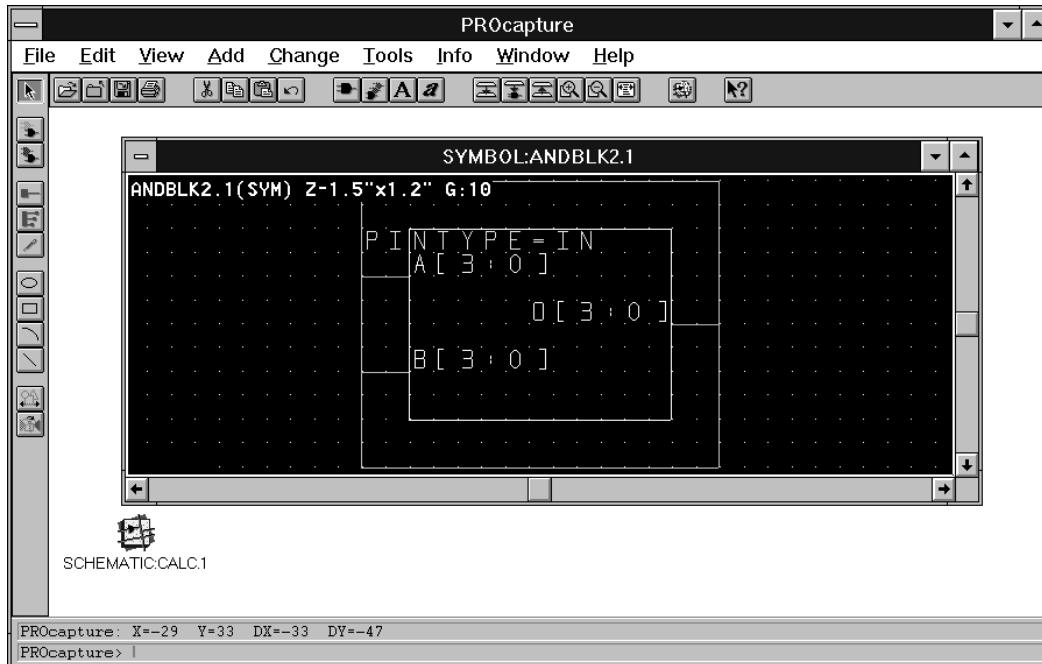


Figure 1-37 Adding Attributes

### Editing Pin Attributes

Editing pin attributes is another way to add an attribute.

1. Click the left mouse button on the lower left pin to select it.
2. Select the **Change** → **Object Attributes** → **Dialog** command.

*Keyboard Shortcut:* You can execute the **Change** → **Object Attributes** → **Dialog** command by typing **cat** on the PROcapture command line.

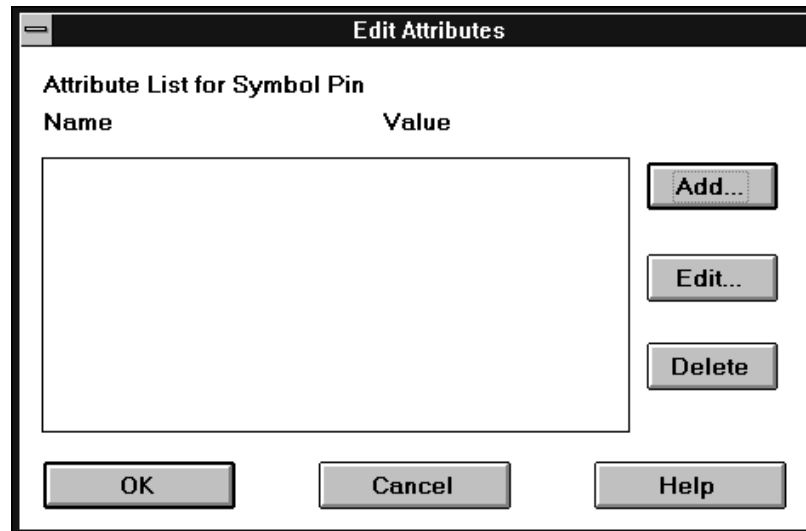
*Toolbar Shortcut:* You can execute the **Change** → **Object Attributes** → **Dialog** command by clicking on the Attribute toolbar icon, shown in Figure 1-38.



**Figure 1-38 Attribute Toolbar Icon**

*Mouse Shortcut:* You can execute the Change → Object Attributes → Dialog command by double-clicking on a labeled pin, net, or component.

The Edit Attributes dialog box appears, as illustrated in Figure 1-39. It is shown for a pin with no attributes attached.



**Figure 1-39 Edit Attributes Dialog Box**

3. Click on **Add**.

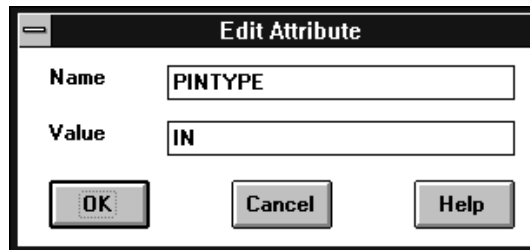
Selecting Add displays an empty Edit Attribute dialog box.

4. In the Name field, type **PINTYPE**.

5. In the Value field, type **IN**.

The completed Edit Attribute dialog box is shown in Figure 1-40.

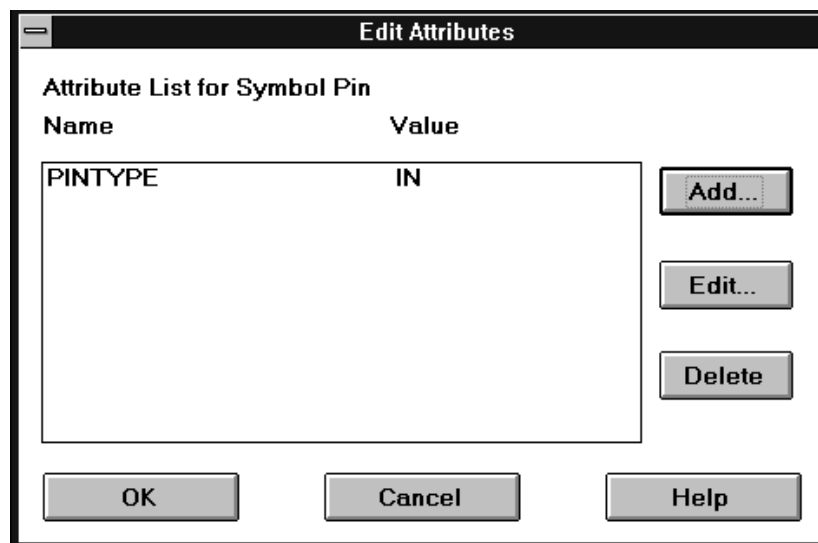




**Figure 1-40 Edit Attribute Dialog Box**

6. Click on **OK**.

Selecting OK closes the Edit Attribute dialog box and updates the Edit Attributes dialog box, shown in Figure 1-41.

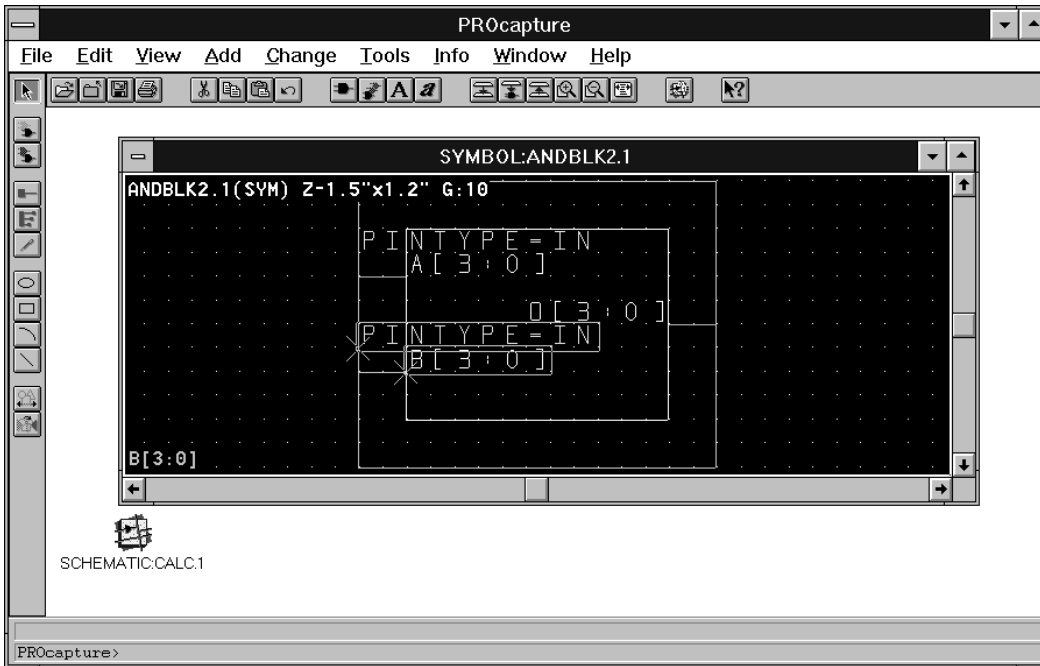


**Figure 1-41 Updated Edit Attributes Dialog Box**

7. Click on **OK**.

The Edit Attributes dialog box closes, and the ANDBLK2 symbol window is reactivated with the attribute added to the pin. Figure 1-42 illustrates this window.

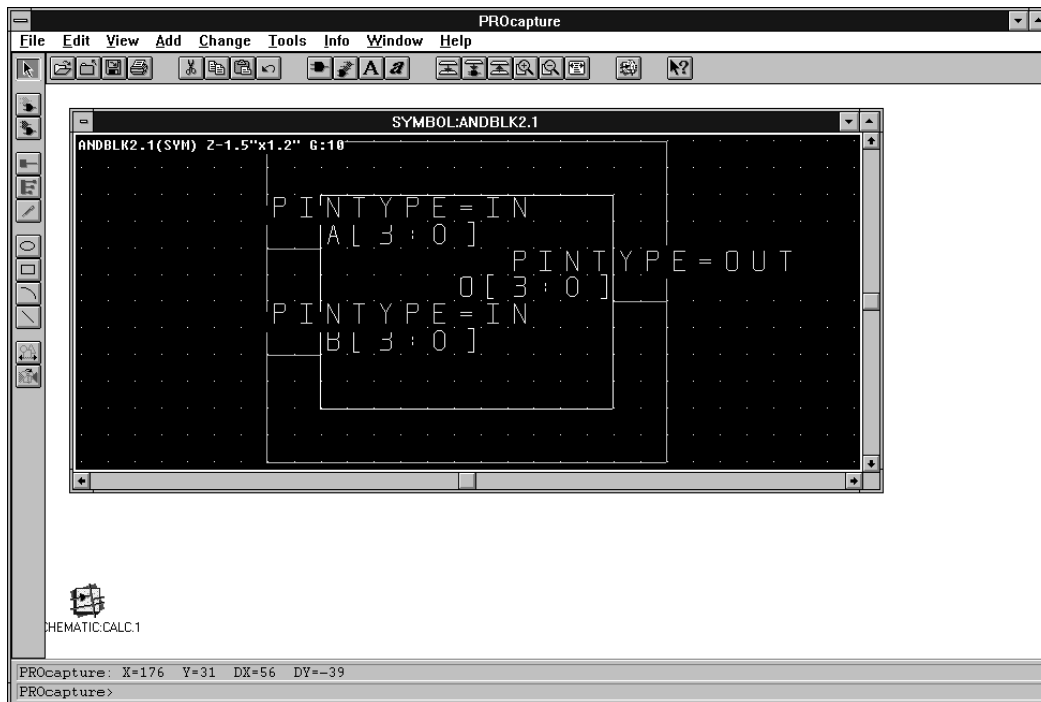
Notice that the attribute is also placed when the ANDBLK2 symbol is reactivated.



**Figure 1-42 ANDLBK2 Symbol Window with PINTYPE=IN Attribute**

### **Adding Other PINTYPE Attributes**

Either by using the Add → Object Attribute command or by editing the pin attributes, as described in the previous two sections, add a PINTYPE=OUT attribute to the O[3:0] pin. Figure 1-43 shows the desired results.



**Figure 1-43 ANDLBK2 Symbol Window with PINTYPE=OUT Attribute**

### Changing Attribute Size

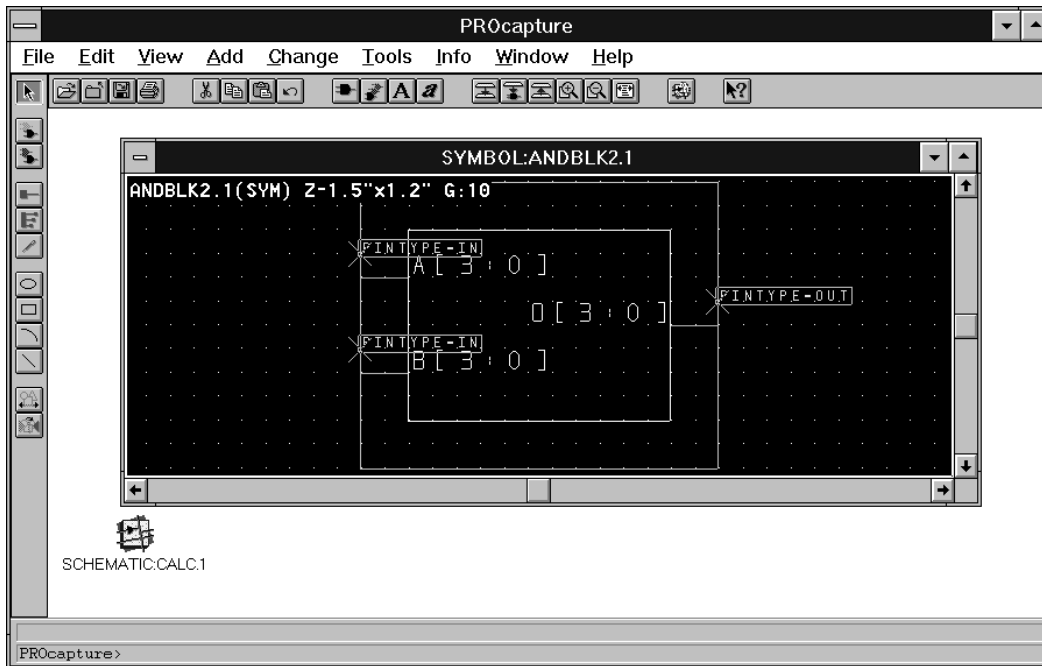
Adding the PINTYPE attributes has better defined the ANDBLK2 symbol, but in the process the symbol has become cluttered. To clean up the symbol, you can reduce the font size of the PINTYPE attributes.

1. Using the left mouse button, select the upper left pin's PINTYPE attribute.  
The attribute's bounding box is displayed.
2. Using the right mouse button, select the remaining two PINTYPE attributes.

The right mouse button allows you to select multiple objects.

3. At the PROcapture command line, type **size 5**. Alternatively, you can select **Change** → **Text** from the menu and change the text size to 5.

Figure 1-44 displays the reduced attribute text size.



**Figure 1-44 Reduced Attribute Text Size**

**Note:** The Size command affects all selected text. Any text created after the Size command is used also reflects the change.

You can, however, change PROcapture's text size variable back to 10.

1. Press the **Esc** key.  
PROcapture now returns the prompt to its default state.
2. Click the left mouse button on an empty area outside of the symbol body.

Clicking the left mouse button on an empty area deselects any selected objects.

3. Type `size 10`.↵.

When you specify the Size 10 command with nothing selected in the symbol window, a message in the status bar confirms the change by indicating that nothing was selected.

### Controlling Attribute Visibility

You can also make attributes invisible if you want to further improve the appearance of the symbol. Rendering the attributes invisible is for cosmetic reasons only; the attributes still affect the symbol.

1. Select the **Change → Object Attributes → Visibility → All Attrs Off** command.

*Keyboard Shortcut:* You can execute the Change → Object Attributes → Visibility → All Attrs Off command by typing `ain`↵ on the PROcapture command line.

*Keyboard Shortcut:* You can execute the Change → Object Attributes → Visibility → All Attrs On command by typing `avi`↵ on the PROcapture command line.

2. At the Attribute Text String [\*]: prompt, press ↵.

Pressing ↵ selects the default wildcard value, which turns all the symbol attributes invisible, as shown in Figure 1-45.

3. To redraw the screen, press **F5** or select **View → Refresh**.

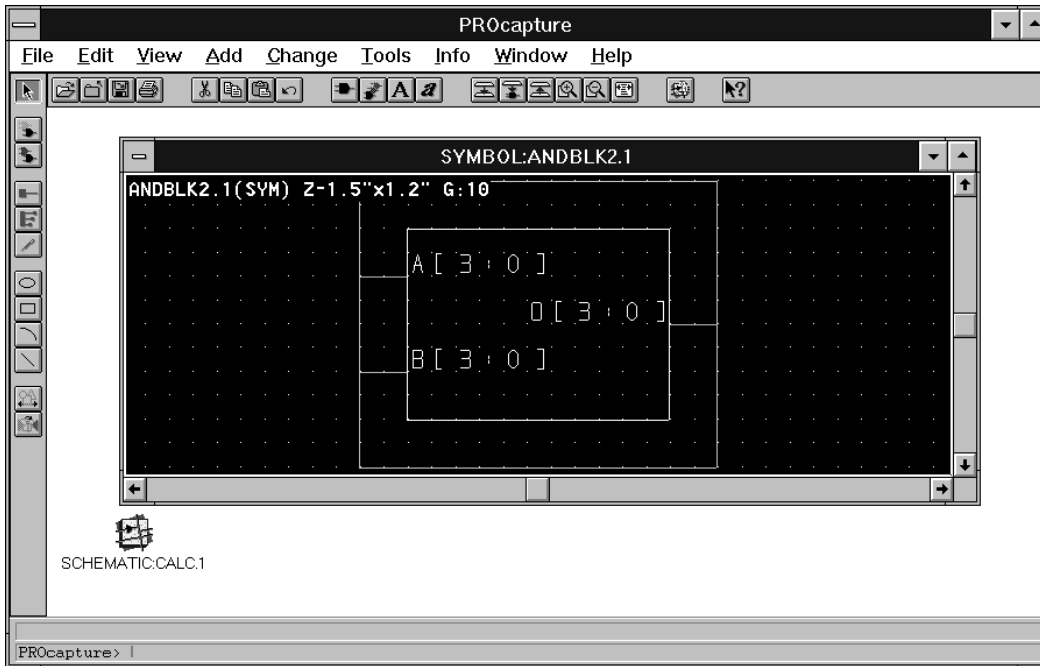


Figure 1-45 Invisible Attributes

## Adding Symbol Text

When a symbol is placed on a schematic, it may be difficult to distinguish it from another symbol. To distinguish one symbol from another, you can add text such as the symbol name to the symbol definition. Text is purely visual.

1. Select the **Add** → **Text** command to bring up the Add Text dialog box.

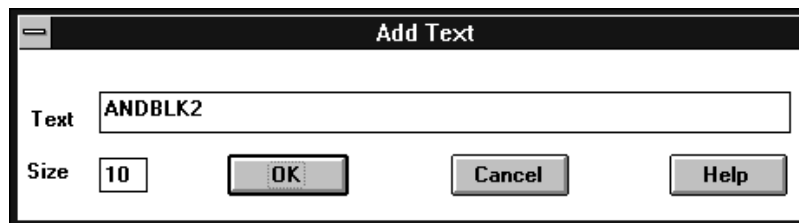
*Keyboard Shortcut:* You can execute the Add → Text command by typing `text` on the PROcapture command line.

*Toolbar Shortcut:* You can execute the Add → Text command by clicking on the Text toolbar icon, shown in Figure 1-46.



**Figure 1-46 Text Toolbar Icon**

2. In the Text field, type **ANDBLK2**, as illustrated in Figure 1-47.



**Figure 1-47 Add Text Dialog Box**

3. Click on **OK**.  
Selecting **OK** closes the **Add Text** dialog box and reactivates the symbol window. A bounding box is displayed next to the pointer.
4. Move the mouse so that the text bounding box is in the desired position; in this case, place it on top of the symbol box.
5. Click the left mouse button to place the text string. The results are depicted in Figure 1-48.

If you make a mistake while typing the text, or want to change the text after you have placed it on the symbol, double-click on the added text. The **Change Text** dialog box comes up so that you can edit the text.

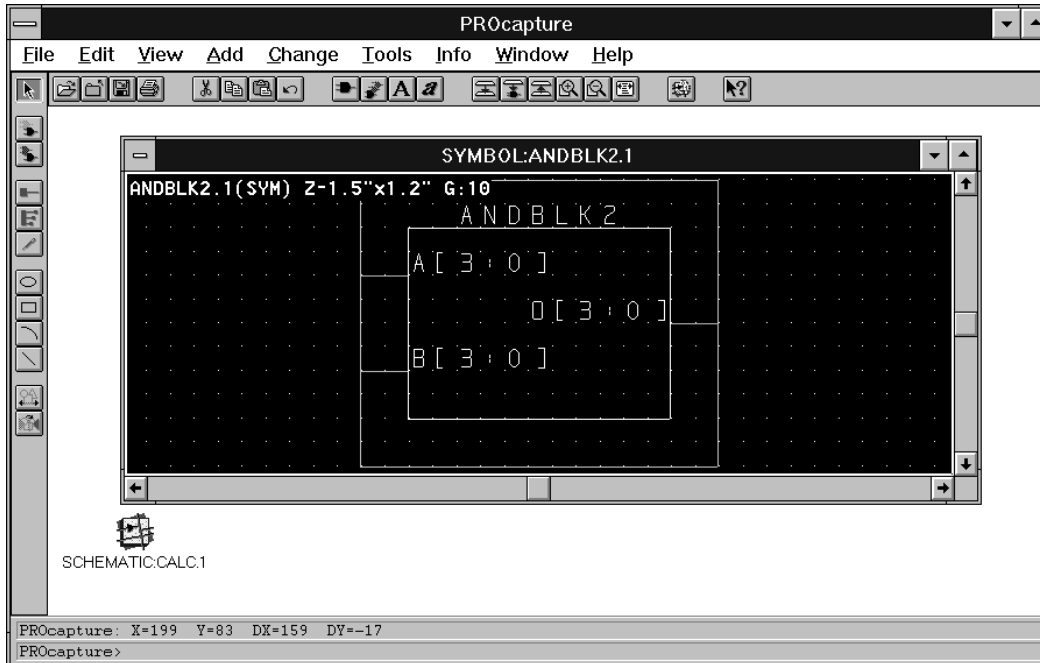


Figure 1-48 Symbol Text

### Changing Symbol Text Size

Depending on the size of the schematic sheet, the text that you have added to the symbol may not be visible at full zoom when the symbol is later placed in a schematic. It may be necessary to change the text size.

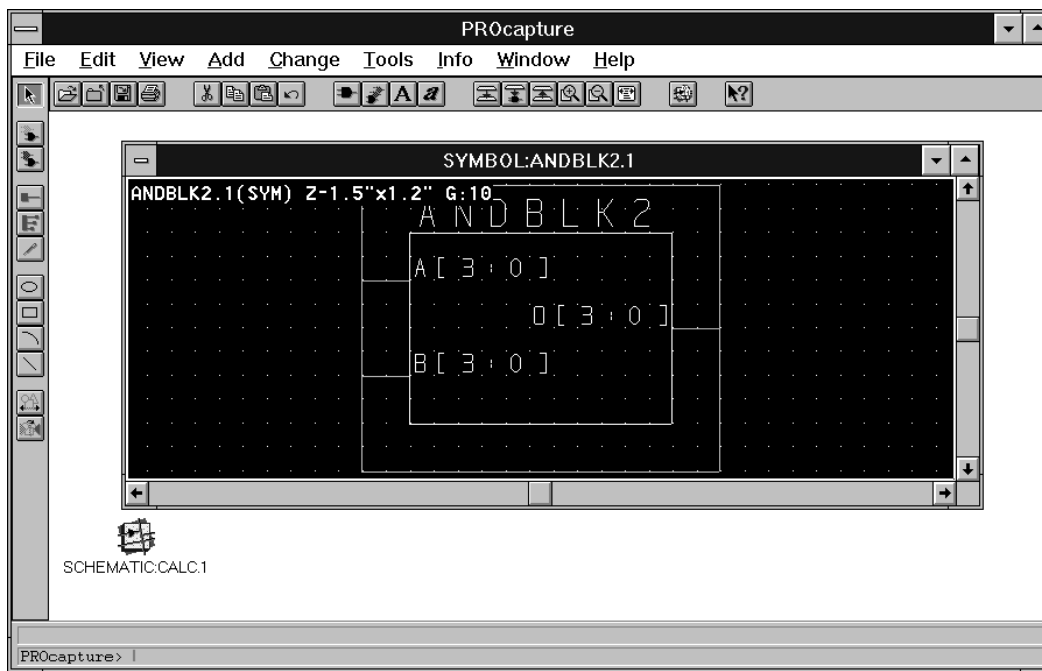
1. Double-click on the ANDBLK2 text and change the size to 15.
2. Click on **OK** to close the Change Text dialog box.

### Moving Text and Objects

Changing the size of the text causes it to be misaligned with the top of the symbol body. Like any other object, text can easily be moved. To reposition the text so that it is centered at the top of the symbol body, follow this procedure.



1. Using the left mouse button, select the text.  
When the text is selected, its bounding box is displayed.
2. Position the mouse inside the bounding box.
3. While holding the left mouse button down, drag the text to its new location. Figure 1-49 shows the repositioned text.



**Figure 1-49 Repositioned Text**

You can also move objects by selecting the objects to be moved, then using the **Edit** → **Move** command to move them to their desired location.

*Mouse Shortcut:* You can execute the Edit → Move command by selecting the component to be moved and dragging it to the new location.

*Keyboard Shortcut:* You can execute the Edit → Move command by typing **move** on the PROcapture command line.

*Toolbar Shortcut:* You can execute the Edit → Move command by clicking on the Move toolbar icon, shown in Figure 1-50.



**Figure 1-50 Move Toolbar Icon**

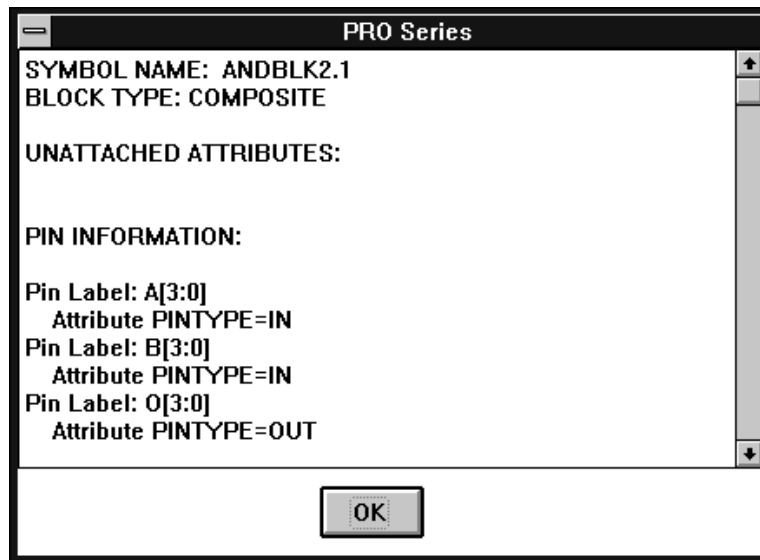
### Saving the ANDBLK2 Symbol

The ANDBLK2 symbol is now complete, and you can save it to the project directory's sym directory.

1. Select the **File** → **Save** command.

*Keyboard Shortcut:* You can execute the File → Save command by typing `wri` on the PROcapture command line.

This command generates a report, which is displayed in a PRO Series information dialog box. Figure 1-51 shows this report.



**Figure 1-51 PRO Series Report in Information Dialog Box**

2. To view the entire report, use the down arrow at the bottom of the scroll bar. Make sure that the information displayed is the same as that shown in Figure 1-51. If your output is not the same, correct the symbol to eliminate the differences and then save the symbol as noted earlier.

The report shows the name of the primary symbol, ANDBLK2. This name not only includes the given name of the symbol, "ANDBLK2," but also the name of the library in which the symbol resides. Because the library was created from scratch, it resides in the first writable library in the viewdraw.ini file. The ANDBLK2 symbol here is part of the primary, or project, directory, c:\user\calc.

3. Click on **OK**.

Selecting OK closes the information dialog box and reactivates the ANDBLK2 symbol window.

## Creating the ORBLK2 Symbol

The next step is to create the symbol for ORBLK2. Since ORBLK2 is similar to ANDBLK2, use the ANDBLK2 symbol as the starting point in creating the ORBLK2 symbol. Only a change to the text is needed.

1. Double-click on the ANDBLK2 text in the ANDBLK2.1 symbol window.
2. In the Change Text dialog box, change the text in the Text field to **ORBLK2**.
3. Click on **OK**.
4. Move the text so that it is centered above the symbol body.
5. Select the **File** → **Save As** command.

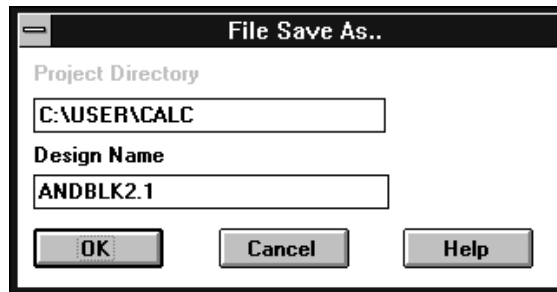
*Keyboard Shortcut:* You can execute the File → Save As command by typing **writeto** on the PROcapture command line.

*Toolbar Shortcut:* You can execute the File → Save As command by clicking on the Save As toolbar icon, shown in Figure 1-52.



**Figure 1-52 Save As Toolbar Icon**

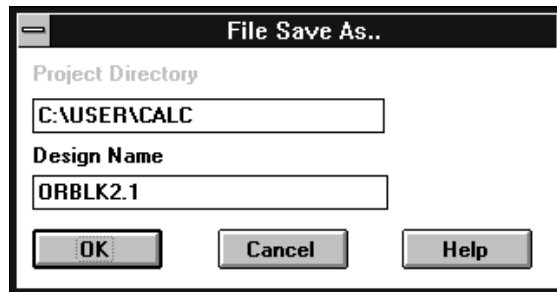
The File → Save As command displays the File Save As dialog box, displayed in Figure 1-53.



**Figure 1-53 File Save As Dialog Box**

All that you need to change is the name of the symbol. The library to which the symbol is written is still the primary library.

6. Change the name in the Design Name field to `ORBLK2.1`, as shown in Figure 1-54.

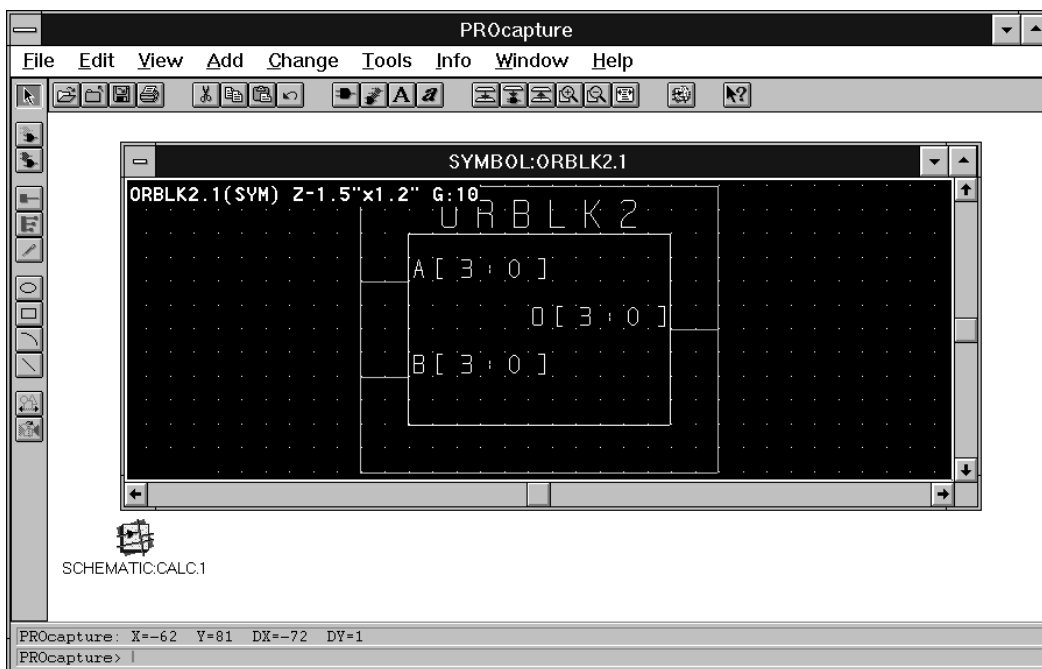


**Figure 1-54 Changing the Design Name**

- Click on **OK** to close the File Save As dialog box and save the design to the desired name.

Figure 1-55 displays the resulting ORBLK2 symbol.

The name in the title bar is now ORBLK2.1 instead of ANDBLK2.1. The File → Save As command changes the window to the current file name so that any further edits only affect the new symbol.



**Figure 1-55 ORBLK2 Symbol**

**Note:** You can also create the ORBLK2 symbol by typing `writeto orblk2.1` on the PROcapture command line.

## Viewing Symbols Simultaneously

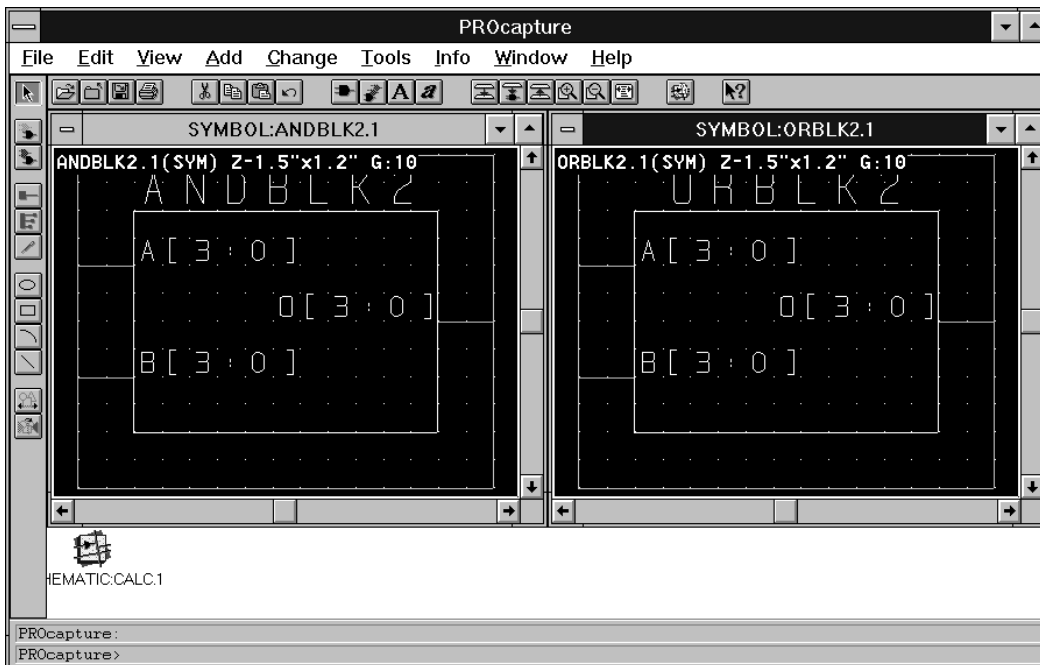
Using the MDI (Multiple Document Interface) feature, you can view the two new symbols side by side.

1. Click on **File** → **Open** to open the ANDBLK2.1 symbol.
2. Select the **Window** → **Tile** command.

The two symbols are now displayed side by side.

3. In each of the symbol windows, zoom to full view by clicking on the **Full** toolbar icon or selecting **View** → **Full**.

Figure 1-56 shows the two symbol windows side by side.



**Figure 1-56 Tiled ANDBLK2 and ORBLK2 Symbol Windows**

## Closing Symbol Windows

You can either make the ANDBLK2 and ORBLK2 symbol windows into icons like the CALC.1 schematic window, or close them. Because the symbols do not require further edits, close them with the following procedure.

1. Click on the ANDBLK2.1 symbol window's title bar to activate the window.
2. Select the **File** → **Close** command.

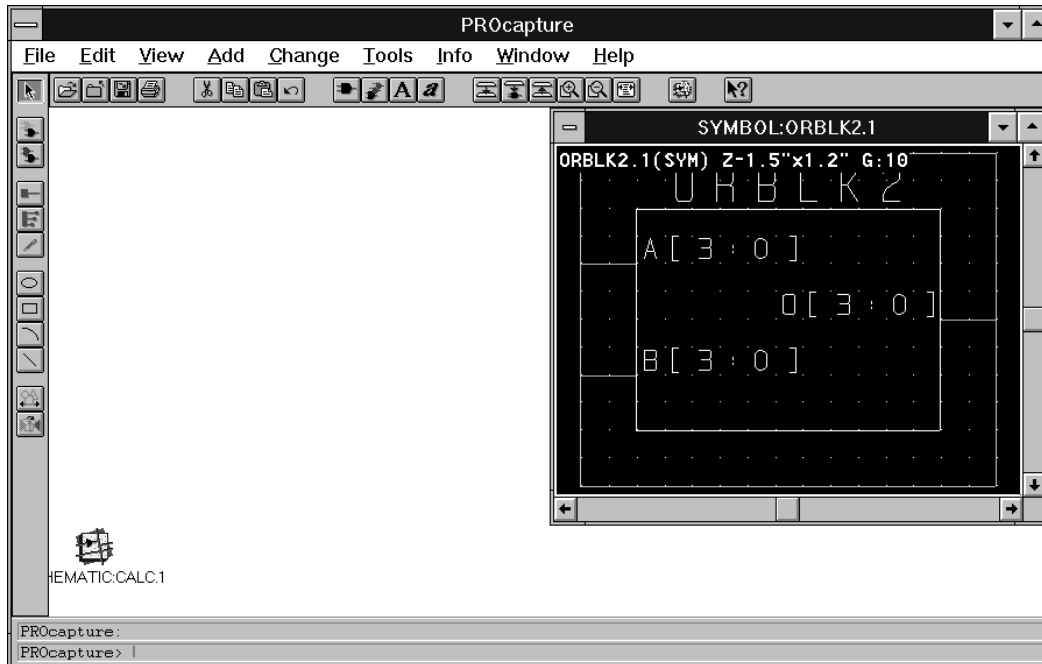
*Keyboard Shortcut:* You can execute the File → Close command by typing `wc1` on the PROcapture command line.

*Toolbar Shortcut:* You can execute the File → Close command by clicking on the Close toolbar icon, shown in Figure 1-57.



**Figure 1-57 Close Toolbar Icon**

The ANDBLK2 symbol window closes. The ORBLK2 window remains onscreen as in Figure 1-58.



**Figure 1-58 Closing the ANDBLK2 Symbol Window**

When the ANDBLK2 symbol window is closed, the ORBLK2 symbol window is activated.

3. To close the ORBLK2 symbol window, click on the Close toolbar icon.

The ORBLK2 symbol window now closes. Your screen should resemble the one in Figure 1-59.



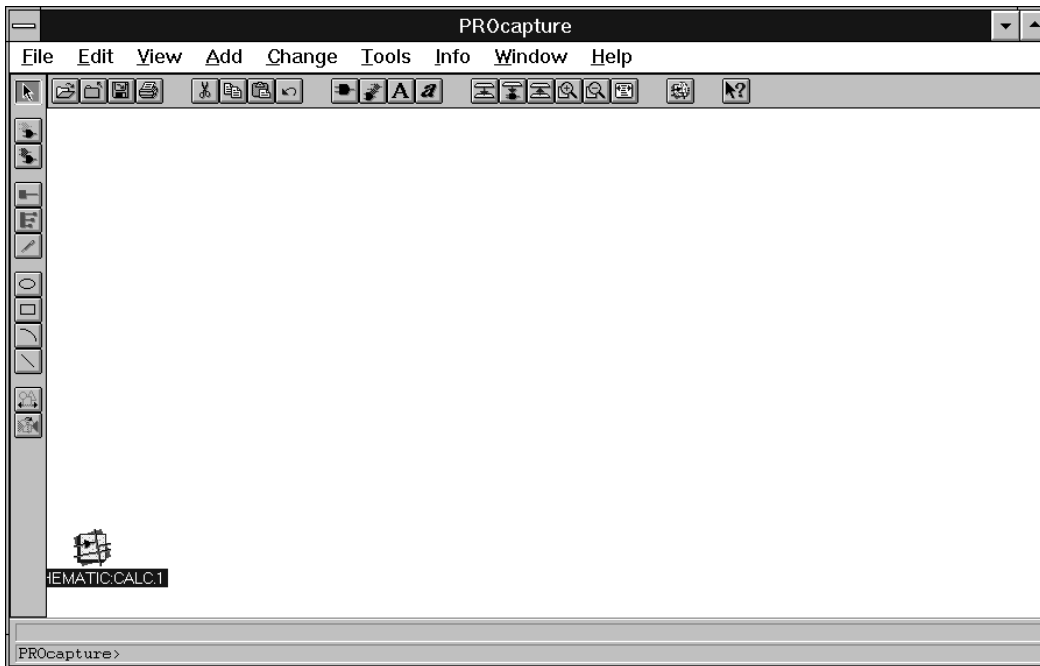


Figure 1-59 Closing the ORBLK2 Symbol Window

## Creating Schematics

You have created symbols for ANDBLK2 and ORBLK2. The next step is to create schematics for these blocks using the instructions in this section. You can then reference the schematics in a higher-level schematic by instantiating and placing the symbols.

### Opening a Schematic Sheet

To create the ANDBLK2 schematic, you must first open a new schematic sheet.

1. Open a new schematic window by clicking on **File** → **Open**.

The File Open dialog box appears.

*Keyboard Shortcut:* You can execute the File → Open command for a schematic by typing **sch.↓** on the PROcapture command line.

2. In the Design Name field, type `ANDBLK2.1`, as shown in Figure 1-60.

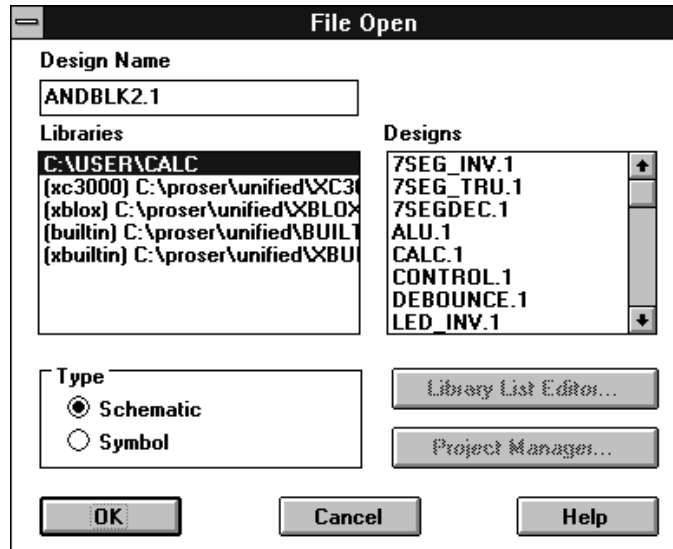
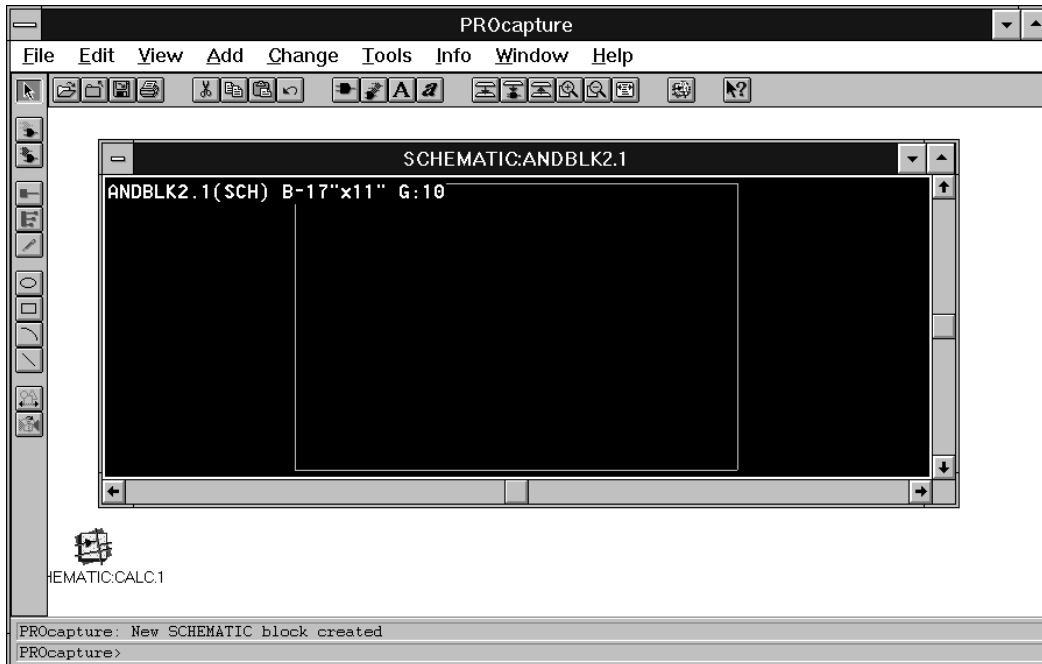


Figure 1-60 Specifying the Design Name

3. Click on `OK`.

The schematic window shown in Figure 1-61 now opens.

**Note:** You can also create the `ANDBLK2` schematic by typing `sch andb1k2.1` on the PROcapture command line.



**Figure 1-61 ANDBLK2.1 Schematic Window**

The file name and type followed by its size in inches and the grid spacing is displayed at the top of the opened window:

ANDBLK2.1(SCH) B-17"x11" G:10

4. Click on the up arrow in the upper right corner of the schematic window.

The schematic window expands to fill the work space.

5. Select the **View** → **Full** command to expand the schematic to fit the window.

## Adding Components

Now that you have a blank schematic window, you are ready to start adding components.

1. To add an AND2 component to the ANDBLK2 schematic window, select the **Add → Component** command.

*Mouse Shortcut:* You can execute the Add → Component command by double-clicking the left mouse button in an unused area of the schematic.

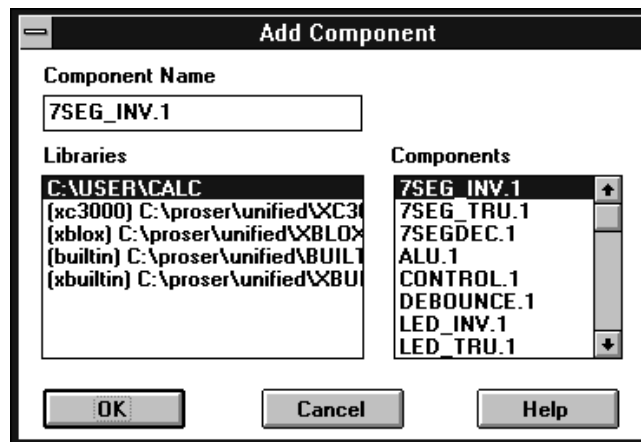
*Keyboard Shortcut:* You can execute the Add → Component command by typing **com** on the PROcapture command line.

*Toolbar Shortcut:* You can execute the Add → Component command by clicking on the Add toolbar icon, shown in Figure 1-62.



**Figure 1-62 Add Component Toolbar Icon**

The Add → Component command brings up the Add Component dialog box, shown in Figure 1-63.



**Figure 1-63 Add Component Dialog Box**

When you initially bring up the Add Component dialog box, PROcapture displays the first component in the selected library in

the Component Name field. You will add a two-input AND gate or AND2. You can find the AND2 component in either the XC3000 or XC7000 library. When you change the selected library in the Libraries list box, the available components of that library are displayed in the Components list box.

2. To view the available components in the XC3000 or XC7000 library, select the desired library in the Libraries list box.

The Components list box in the dialog box is now updated, as indicated in Figure 1-64.

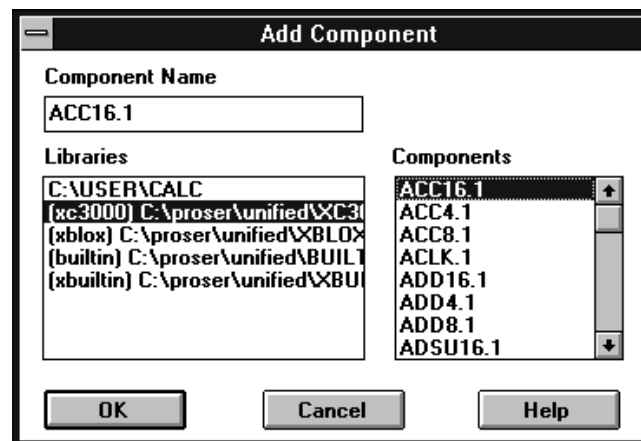


Figure 1-64 Updated Components List

3. Using the down arrow in the scroll bar, scroll down until the AND2.1 component is displayed.
4. Select the **AND2.1** component.

The component is now highlighted, and the Component Name field is updated, as shown in Figure 1-65.

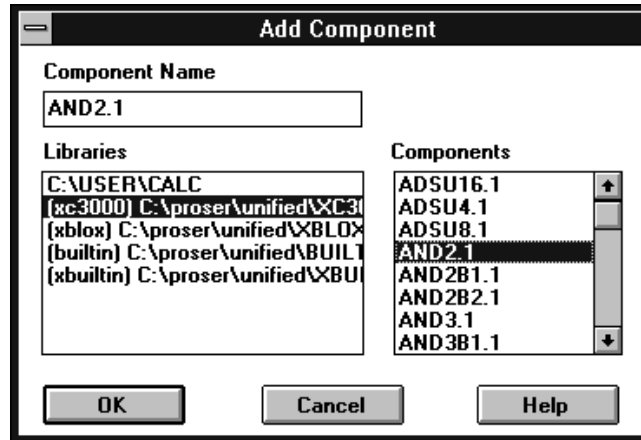


Figure 1-65 Updated Component Name

5. Click on **OK**.

The Add Component dialog box closes and the ANDBLK2.1 schematic window is reactivated. An outline of the component being added appears at the end of the pointer.

**Note:** You can also add a component like the AND2.1 component by typing `com comp_name` on the PROcapture command line.

6. To place the AND2 component, as shown in Figure 1-66, click the left mouse button on the desired location in the schematic window.

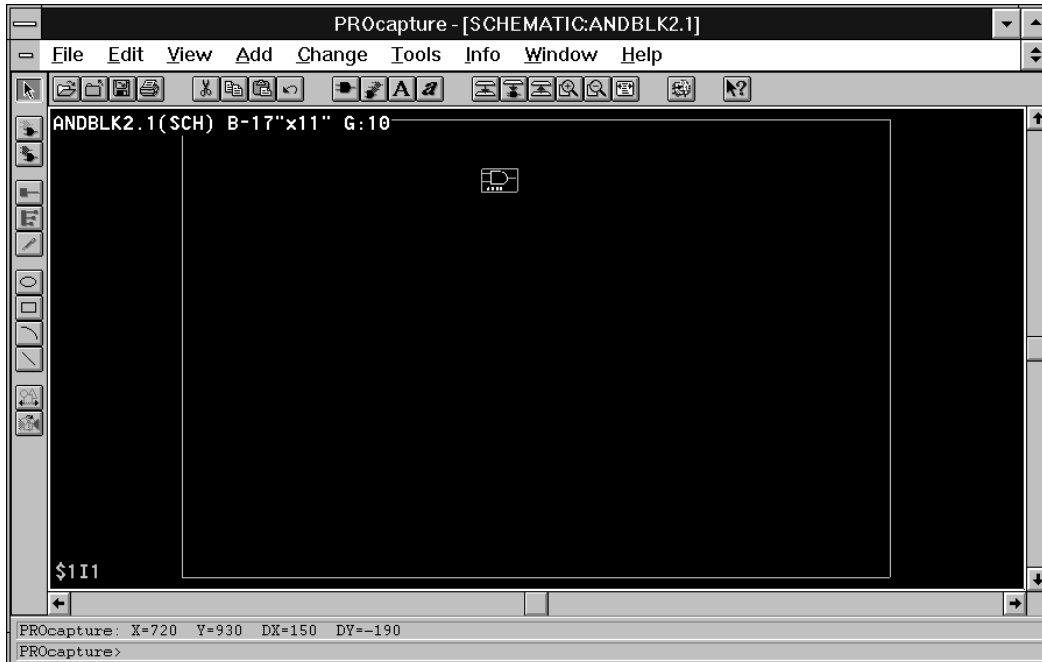


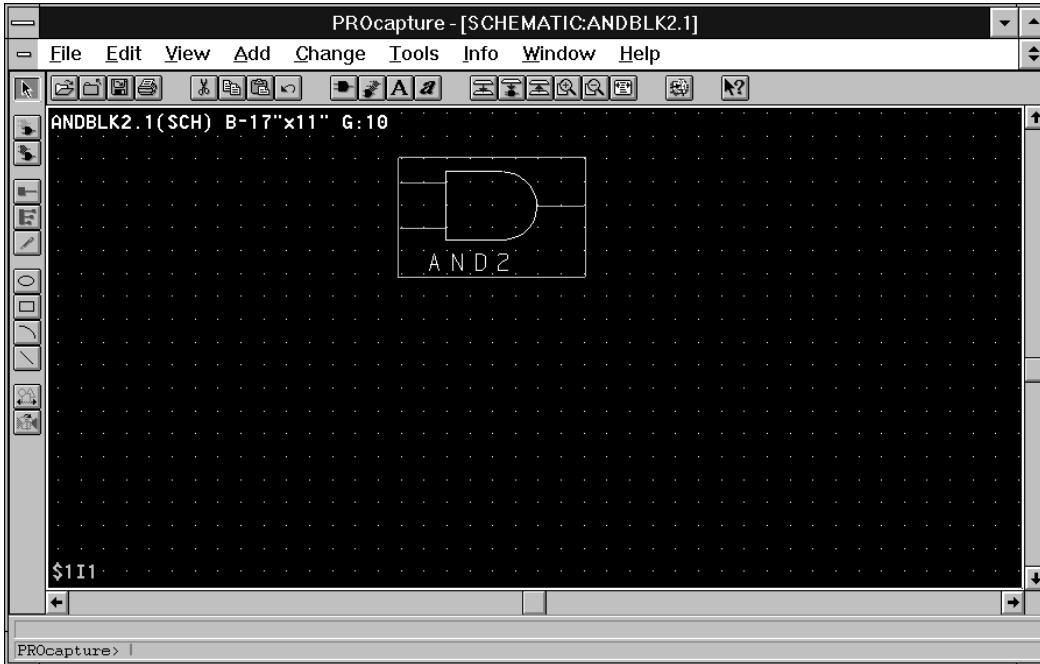
Figure 1-66 Placing the AND2 Component

## Copying Components

Now that there is an AND2 component placed on the schematic, you can use the Copy command to instantiate additional AND2 components.

1. To copy the AND2 component, zoom in to the area surrounding AND2.

If the AND2 component does not have a box surrounding it, as shown in Figure 1-67, select it by clicking the left mouse button.



**Figure 1-67 Outlined AND2 Component**

2. Select the Copy toolbar icon, shown in Figure 1-68.



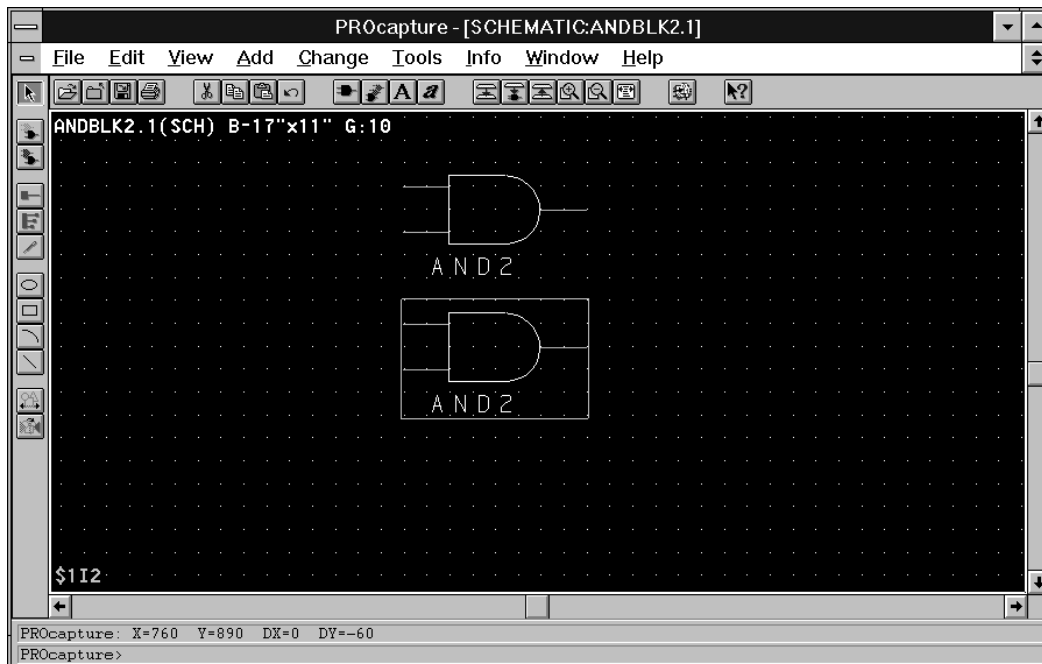
**Figure 1-68 Copy Toolbar Icon**

*Keyboard Shortcut:* You can execute the Copy command by typing `cop-l` on the PROcapture command line.

Selecting the Copy toolbar icon changes the pointer to a crosshair and enters Copy mode, which is reflected in the PROcapture command line. If the pointer does not change to a crosshair, you have not selected the item to copy.



3. While holding down the left mouse button in the schematic window, drag the mouse to the desired location of the new AND2 component.
4. Release the left mouse button to place the copied AND2 component, as Figure 1-69 illustrates.



**Figure 1-69 Copying AND2 Component**

Instantiate the remaining two AND2 components by using either the Add or Copy commands to create the schematic shown in Figure 1-70.

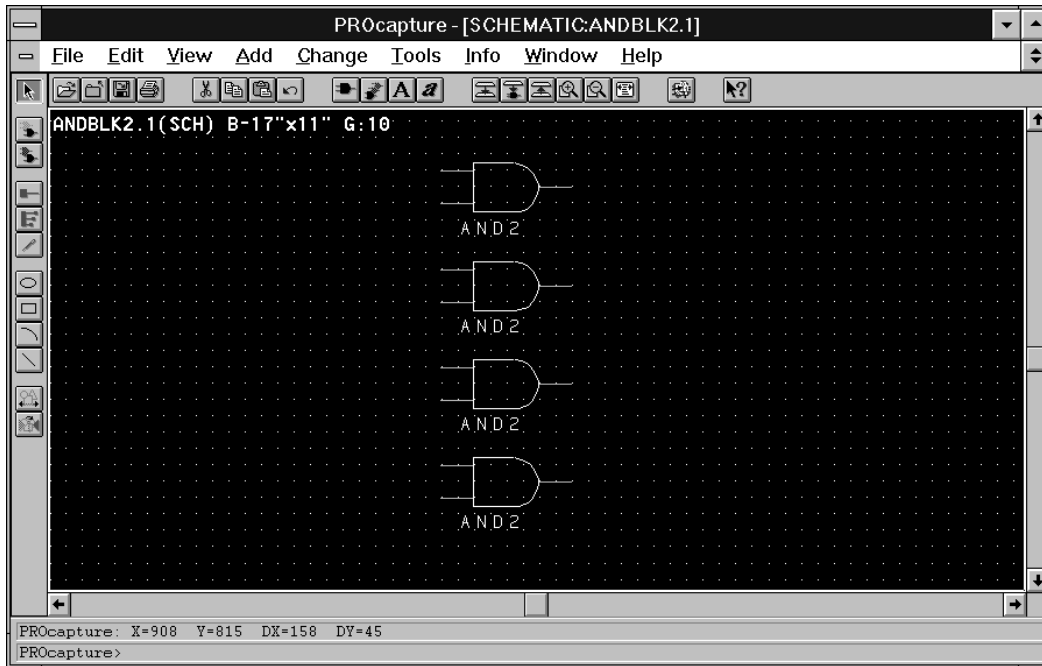


Figure 1-70 Instantiated AND2 Components

## Moving Components

If you make a mistake while placing a component, you can easily move the component to a new location.

1. Select the component to be moved.
2. After selecting the first component, use the right mouse button to select any additional components to be moved.
3. Select the **Edit** → **Move** command.

*Mouse Shortcut:* You can execute the Edit → Move command by selecting the component to be moved and dragging it to the new location.

*Keyboard Shortcut:* You can execute the Edit → Move command by typing **m** on the PROcapture command line.

*Toolbar Shortcut:* You can execute the Edit → Move command by clicking on the Move toolbar icon, shown in Figure 1-71.



**Figure 1-71 Move Toolbar Icon**

Selecting the Edit → Move command changes the pointer to a crosshair and enters Move mode, which is reflected in the PROcapture command line.

4. While holding the left mouse button down, drag the selected component to its new location.

## Adding Nets

Nets and buses establish connectivity between pins on the same hierarchical level of a design. However, it is not always necessary to physically connect nets on the schematic. If two dangling nets or buses within a single schematic are labeled with the same name, they are considered electrically connected. Labeling unconnected nets with the same name is sometimes a useful technique that can make schematics easier to read, especially when dealing with clocks. However, you must keep track of all signal names and make sure they match exactly.

Use the following procedure to add a net.

1. Select the **Add → Net** command.

*Keyboard Shortcut:* You can execute the Add → Net command by typing **ne** on the PROcapture command line.

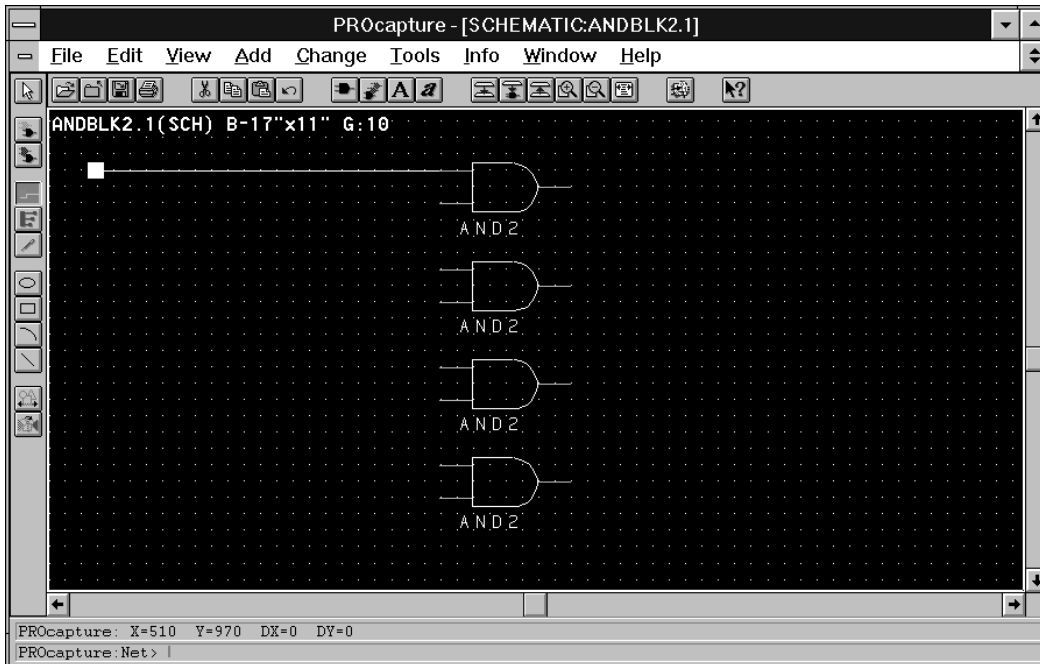
*Toolbar Shortcut:* You can execute the Add → Net command by clicking on the Net toolbar icon, shown in Figure 1-72.



**Figure 1-72 Net Toolbar Icon**

Selecting Add → Net changes the pointer to a crosshair and enters Add Net mode, which is reflected in the PROcapture command line.

2. Point the crosshair at the top left pin on the first AND2 component.
3. Click the left mouse button on the pin.
4. Move the mouse to the desired end point of the net and click the left mouse button.
5. Click the right mouse button to complete the net, as shown in Figure 1-73.

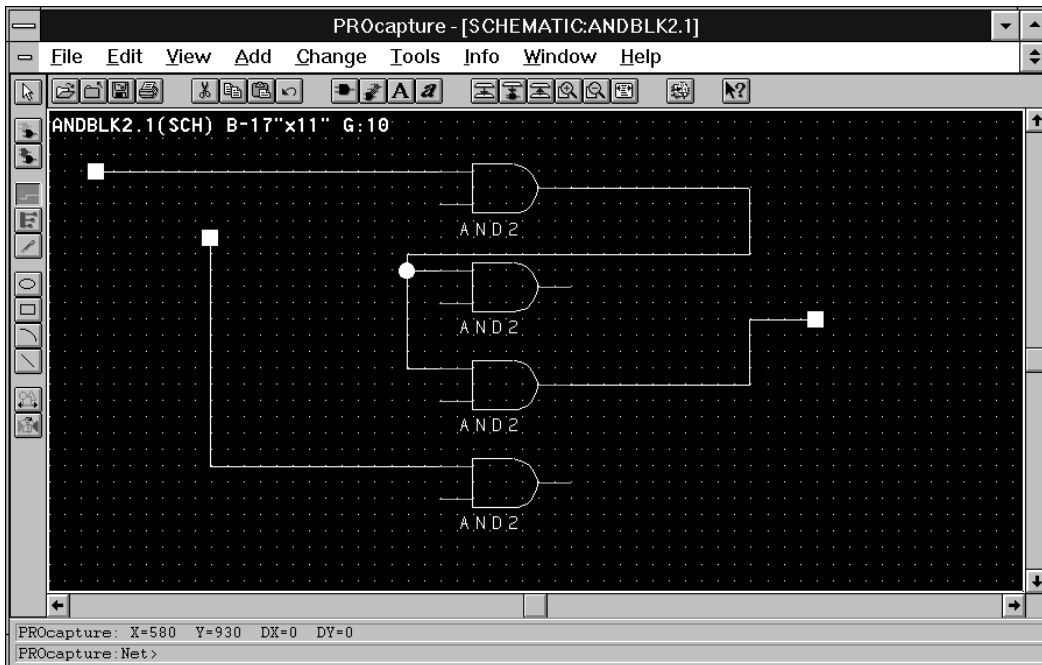


**Figure 1-73 Dangling Net**

The net created here is a dangling net. You can also add nets to connect components. Often when you use a net to connect two components, the net must change direction, or pivot, to reach its destination. To create a pivot point when adding a net, click the left

mouse button. The net continues from that location to either the next pivot point or the destination.

Using the left mouse button to begin the net and add pivot points, experiment by adding various nets. End each net at the last pivot point by clicking the right mouse button on open space to create a dangling net. Connect nets to other pins or nets by clicking the left mouse button on the desired location. For example, add the appropriate nets to create the schematic shown in Figure 1-74.



**Figure 1-74 Completed Net**

After you have familiarized yourself with the various mouse and toolbar commands used to add nets, delete all the nets from the schematic, leaving only the AND2 components. Delete the nets by selecting them and pressing the Delete key or typing `del` on the command line.

## Adding Buses

Sometimes it is convenient to draw a set of signals as a bus rather than as several separate wires. It is not necessary to connect a bus physically with the nets that make up the bus. There are several schematics in the Calc design that simply place a short bus segment on the schematic and label it, so that a bus pin can be used on the symbol.

To add a bus, follow these steps.

1. Select the **Add** → **Bus** command.

*Keyboard Shortcut:* You can execute the Add → Bus command by typing **bu** on the PROcapture command line.

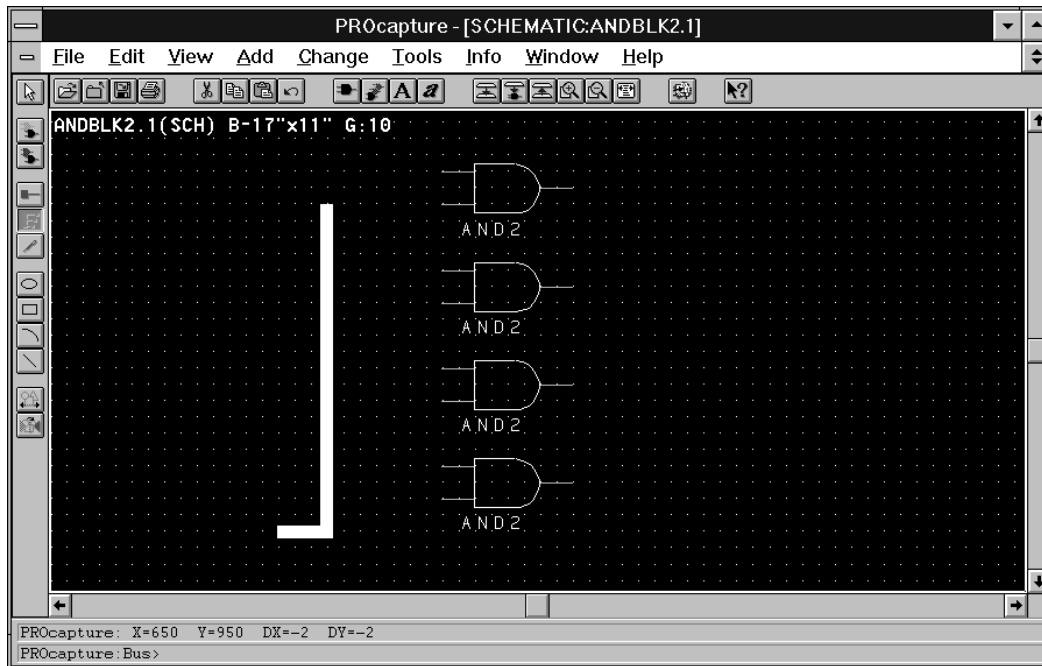
*Toolbar Shortcut:* You can execute the Add → Bus command by clicking on the Bus toolbar icon, as shown in Figure 1-75.



**Figure 1-75 Bus Toolbar Icon**

Selecting Add → Bus changes the pointer to a crosshair and enters Add Bus mode. The mode is reflected in the PROcapture command line.

2. Point the crosshair at the lower left corner of the schematic.
3. Click the left mouse button to begin the bus.
4. To add the final pivot point, move the mouse to the right, click the left mouse button, move the mouse up, and click the left mouse button.
5. Click the right mouse button to complete the bus, which is shown in Figure 1-76.



**Figure 1-76 Completed Bus**

6. Add the remaining buses and finish off the connections by connecting the AND2 components to the buses using nets, as shown in Figure 1-77.

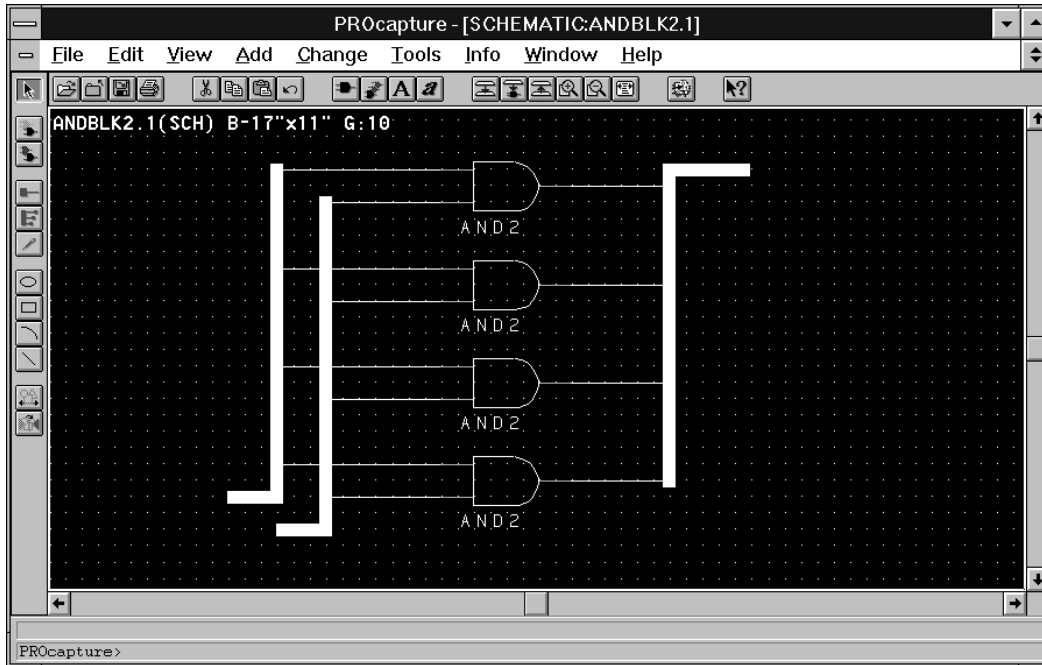


Figure 1-77 AND2 Components Connected to Buses

## Adding Labels

The next step is to add labels to the nets and buses. Labeling is the process of identifying a net or a component by assigning a text string to it. It is strongly recommended that you label all nets on the schematic to make debugging and simulation easier.

1. Using the left mouse button, select the upper net attached to the upper AND2 component.
2. Select the **Add → Object Label** command.

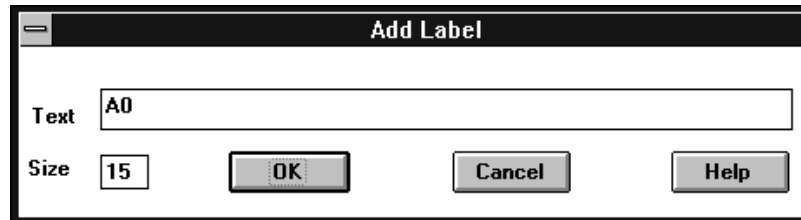
*Mouse Shortcut:* You can execute the Add → Object Label command by double-clicking on the net to which you want to add the label.

*Keyboard Shortcut:* You can execute the Add → Object Label command by typing **la** on the PROcapture command line.



The Add Label dialog box appears.

3. Type **A0** in the Text field of the dialog box, as indicated in Figure 1-78.

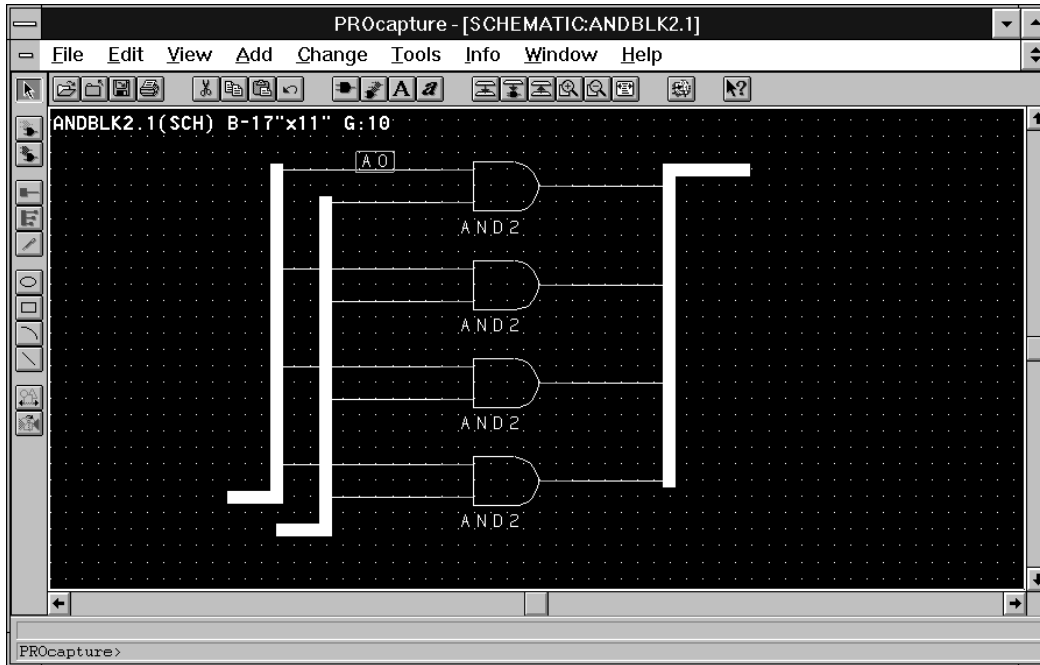


**Figure 1-78 Text in the Add Label Dialog Box**

4. Click on **OK**.

The Add Label dialog box closes, and the ANDBLK2 schematic window is reactivated. An outline of the label, or bounding box, appears on the screen next to the pointer.

5. Point the mouse so that the label outline is in its desired location above the upper net.
6. Click the left mouse button to place the label. Figure 1-79 shows the labeled net.



**Figure 1-79 Labeled Net**

7. If you mislabel a net or bus, you do not have to delete the label. To change the label, select it and choose **Change** → **Text**, or double-click on the text to bring up the Change Text dialog box.

All buses and nets going into a bus must be labeled. Some examples of legal bus names are given following.

BUS LABEL	DESCRIPTION
Q[0:7]	8-bit bus, signals Q0 (MSB) through Q7 (LSB)
Q[7:0]	8-bit bus, signals Q7 (MSB) through Q0 (LSB)
Q[7:0], SET, CLK	10-bit bus, signals Q7 through Q0, SET and CLK
A[7:0], B[7:0]	16-bit bus, signals A7 through A0, B7 through B0
DATA[0:7:2]	4-bit bus, signals DATA0, DATA2, DATA4, and DATA6
DATA[0:F/H]	16-bit bus, specified in hexadecimal (you can also specify a bus in binary, octal, or decimal)

Complete the ANDBLK2 schematic by adding labels to the remaining buses and nets, as shown in Figure 1-80. You can add labels to buses and nets by double-clicking on them.

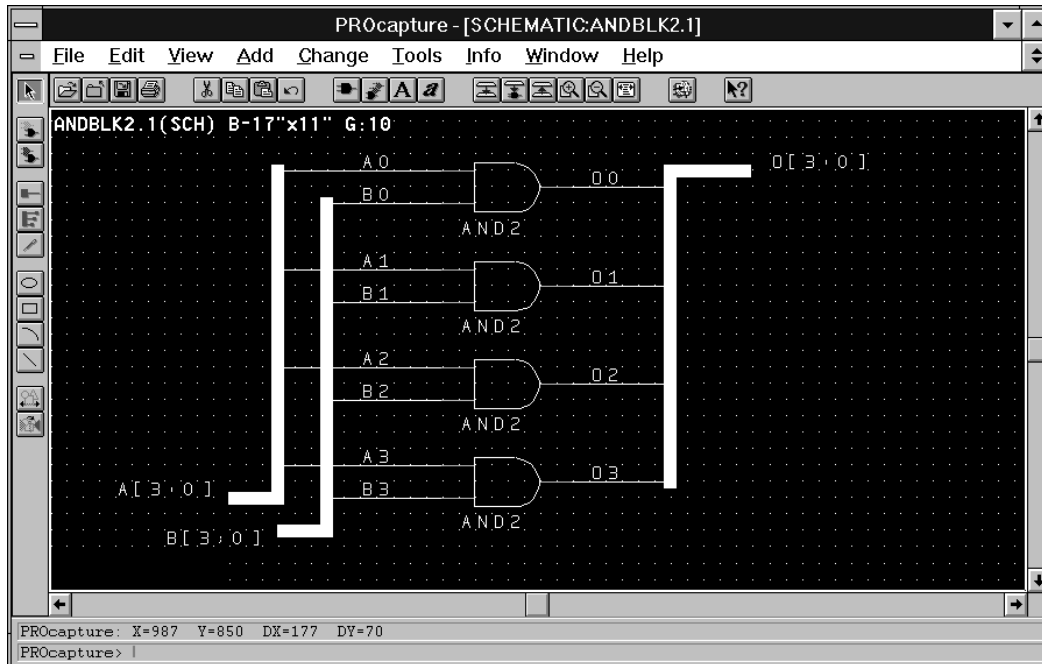


Figure 1-80 Completed ANDBLK2 Schematic

**Note:** Do not confuse the letter “O” with the number zero (0). You can see the contrast between these two characters in the O0 net name label in Figure 1-80.

## Saving the Schematic

The ANDBLK2.1 schematic is now complete and must be saved.

To save the schematic, select the **File** → **Save** command.

Selecting File → Save checks the schematic for any errors and then saves the schematic. If the following message is not displayed in the status window and a dialog box appears, correct the schematic accordingly and re-save when finished.

0 error(s) and 0 warning(s) in project  
primary:ANDBLK2.1

## Creating the ORBLK2 Schematic

The ORBLK2.1 schematic is very similar to the ANDBLK2.1 schematic. Rather than create it by adding components and drawing nets, you can copy the ANDBLK2.1 schematic into the copy buffer and then paste it into the new ORBLK2.1 schematic window.

1. Select the **File** → **Open** command to open a new schematic with the name ORBLK2.1.

The new ORBLK2 window opens on top of the ANDBLK2 window, as Figure 1-81 demonstrates.

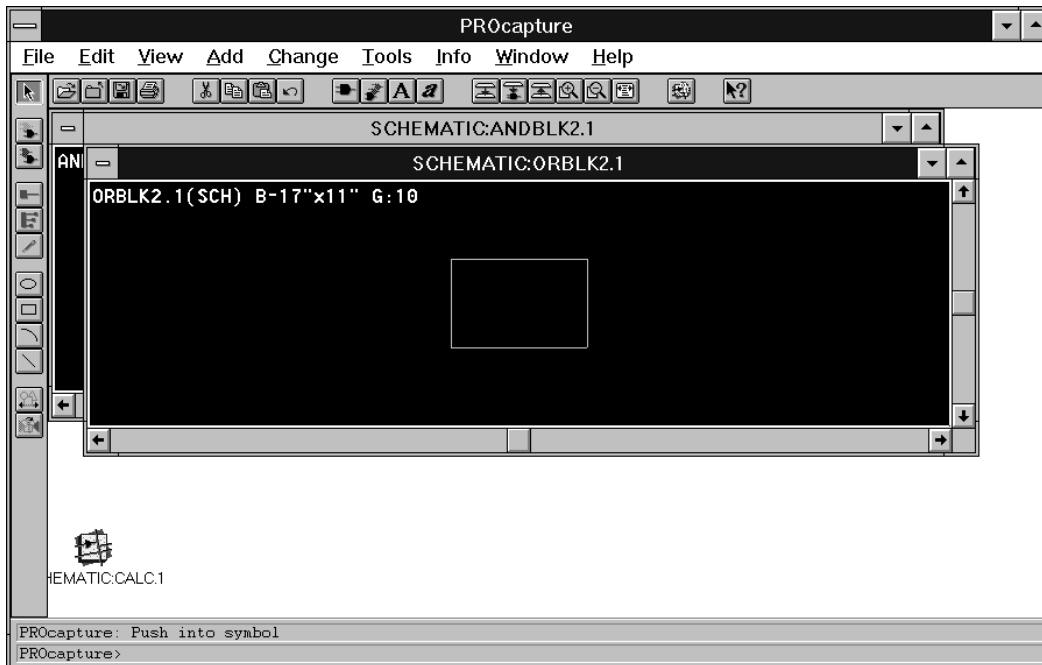
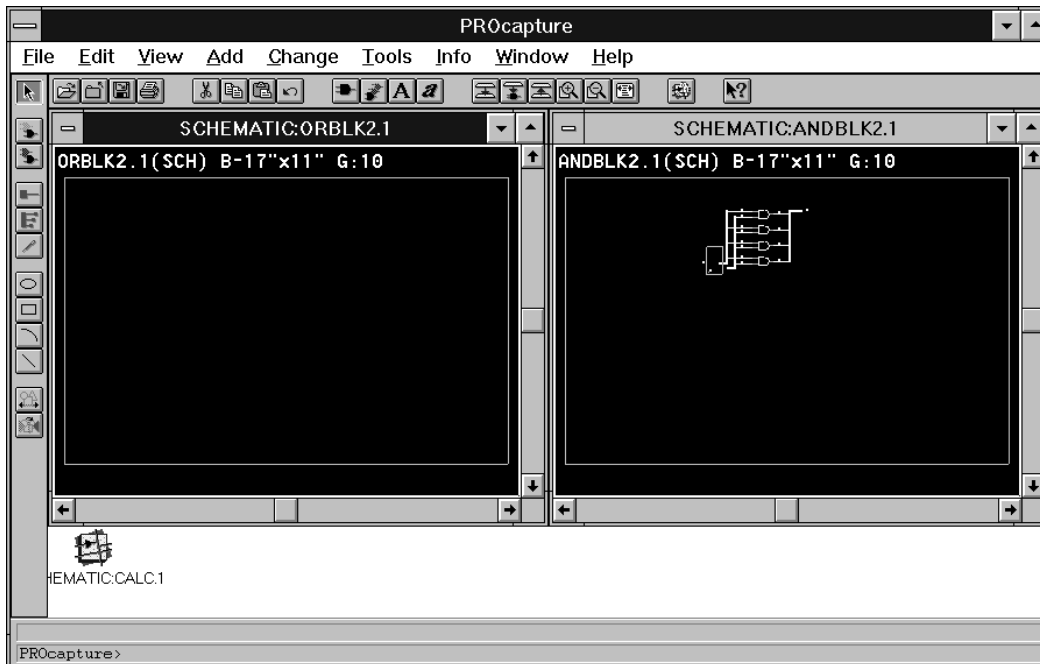


Figure 1-81 Copying ANDBLK2 Schematic to ORBLK2

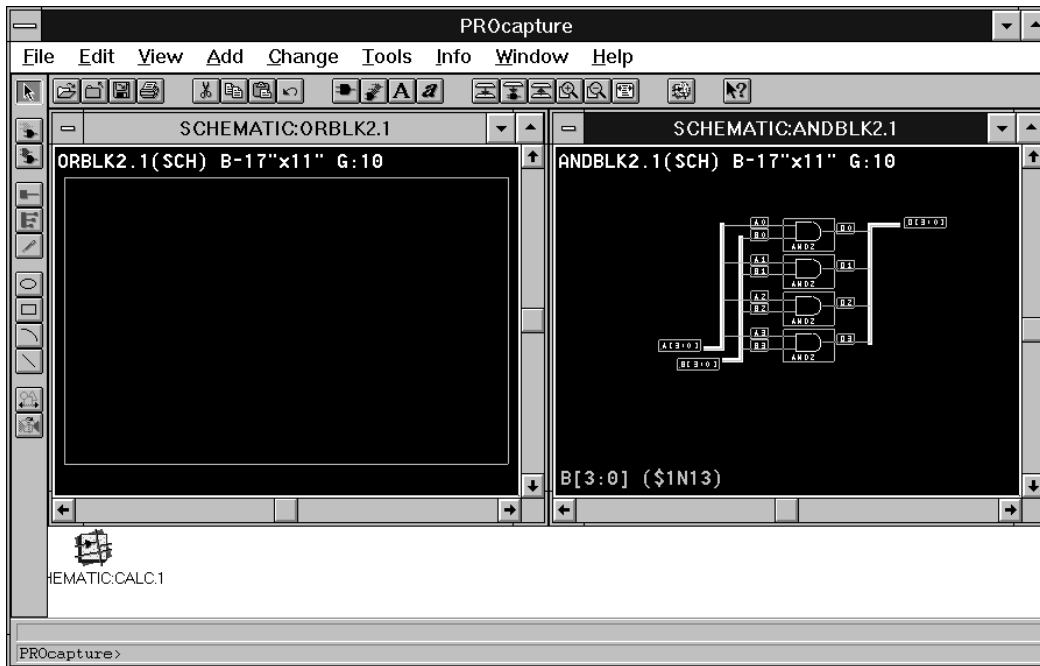
2. Select the **Window** → **Tile** command.

This command displays the open windows side by side in the workspace, as shown in Figure 1-82.



**Figure 1-82 Tiled Schematic Windows**

3. Click on the title bar of the ANDBLK2.1 schematic window to set it as the current window.
4. Point the mouse above and to the left of all the components, nets, and buses in the ANDBLK2.1 schematic.
5. Holding down the left mouse button, drag the mouse below and to the right of all the components, nets, and buses in the ANDBLK2.1 schematic.
6. Release the left mouse button to select any component, net, or bus in the area of the drag. Figure 1-83 displays the selected components.



**Figure 1-83 Selecting Components, Nets, and Buses**

7. Select the **Edit** → **Copy** command.

*Keyboard Shortcut:* You can execute the Edit → Copy command by typing **bcop** on the PROcapture command line.

*Toolbar Shortcut:* You can execute the Edit → Copy command by clicking on the Edit Copy toolbar icon, shown in Figure 1-84.



**Figure 1-84 Edit Copy Toolbar Icon**

8. Click on the title bar of the ORBLK2.1 schematic window to set it as the current window.

9. Select the **Edit** → **Paste** command.

*Keyboard Shortcut:* You can execute the Edit → Paste command by typing **bpa** on the PROcapture command line.

*Toolbar Shortcut:* You can execute the Edit → Paste command by clicking on the Paste toolbar icon, shown in Figure 1-85.



**Figure 1-85 Paste Toolbar Icon**

Selecting Edit → Paste displays a bounding box of the components, nets, and buses that are being pasted on the screen next to the pointer.

10. Point the mouse to position the bounding box.
11. Click the left mouse button to place the components, nets, and buses. Figure 1-86 shows the copied components.
12. To inspect the two schematics, select each and zoom in to the area where the components have been placed.

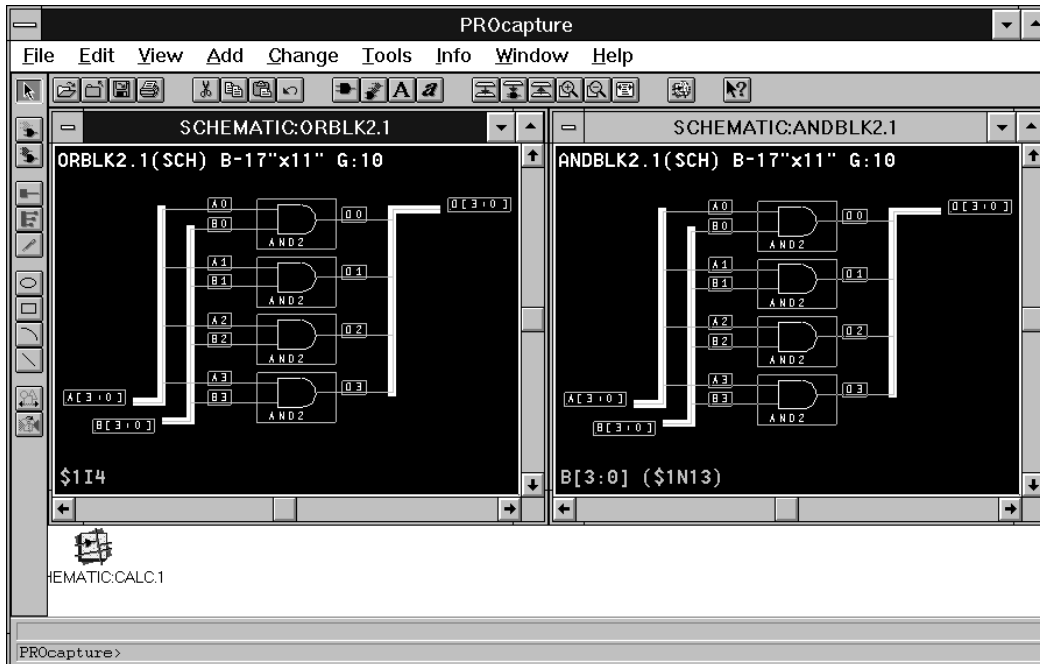


Figure 1-86 Copying Components, Nets, and Buses

## Changing AND2 Components to OR2 Components

All that remains to be done is to change the AND2 components to OR2 components.

1. Make sure that the ORBLK2 window is highlighted, and select the **Edit** → **Select** → **Component** command.

*Keyboard Shortcut:* You can execute the Edit → Select command by typing `sco` followed by a carriage return on the PROcapture command line. Here is an example:

```
sco and2
```

The Select Component dialog box appears, as indicated in Figure 1-87. It displays all the components in the active schematic. By default, xc3000:AND2.1 is selected in the Component List field.



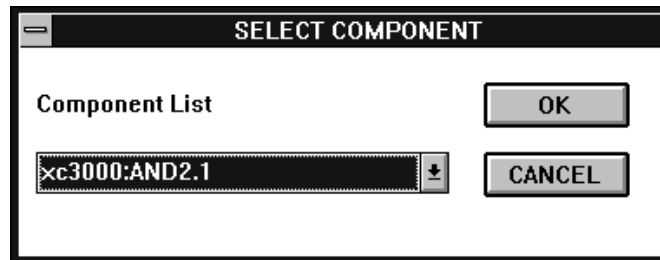


Figure 1-87 Select Component Dialog Box

2. Click on OK.

The Select Component dialog box closes, and the ORBLK2.1 schematic is reactivated with all the AND2.1 components selected, as you can see in Figure 1-88.

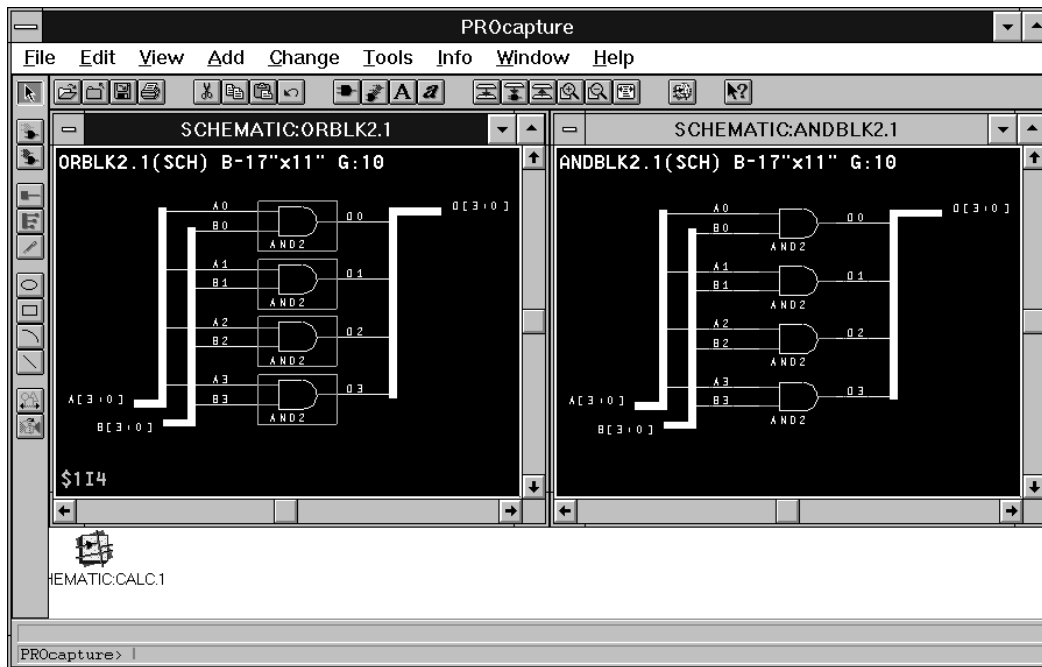


Figure 1-88 Reactivated ORBLK2.1 Schematic

3. Select the **Change** → **Component** command.

*Keyboard Shortcut:* You can execute the Change → Component command by typing `cc ↵` on the PROcapture command line. Here is an example:

```
cc or2 ↵
```

The Change → Component command brings up the Change Component dialog box.

4. Select the XC3000 library for an FPGA design, as shown in Figure 1-89, or the XC7000 library for an EPLD design.

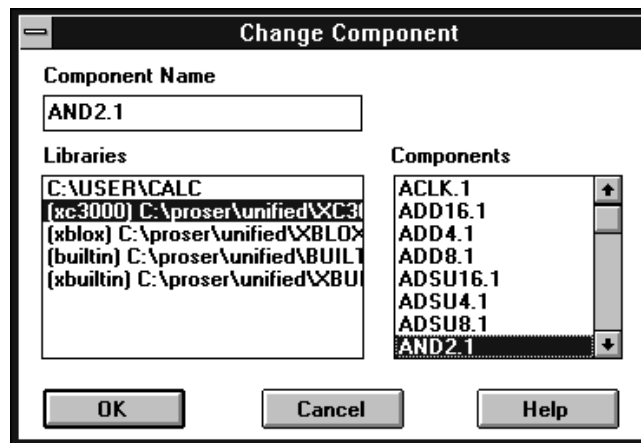


Figure 1-89 Change Component Dialog Box

5. Using the down arrow in the scroll bar, scroll down until the OR2.1 component is displayed.
6. Select the OR2.1 component.

The component is highlighted, and the Component Name field is updated, as indicated in Figure 1-90.

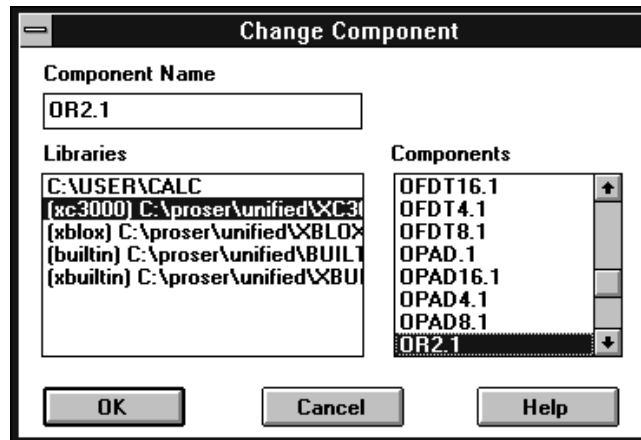


Figure 1-90 Updated Component Name

7. Click on ok.

Selecting OK closes the Change Component dialog box and re-activates the updated ORBLK2.1 schematic, as shown in Figure 1-91.

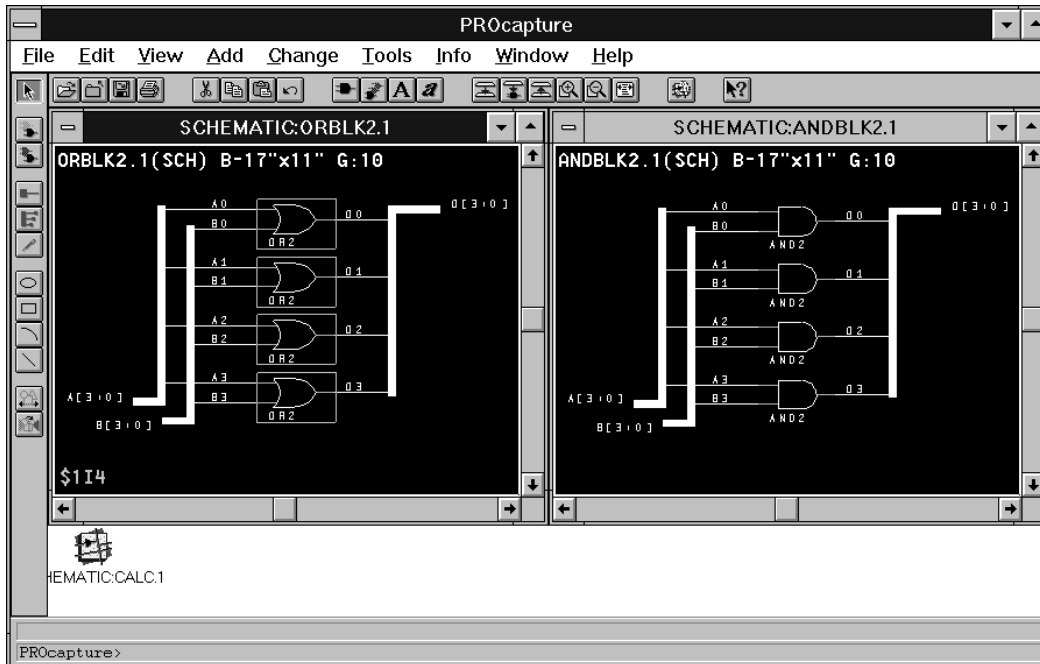


Figure 1-91 Completed ORBLK2.1 Schematic

## Saving the ORBLK2 Schematic

With the AND2 components changed to OR2 components, you can now save the ORBLK2.1 schematic.

1. Select **File** → **Save**, which checks the schematic for any errors and then saves the schematic.

If the following message is not displayed in the status window and a dialog box appears, correct the schematic accordingly and re-save when finished:

```
0 error(s) and 0 warning(s) in project
primary:ORBLK2.1
```

2. Close the ANDBLK2.1 and ORBLK2.1 schematic windows.

## Completing the ALU Schematic

So far you have created the ANDBLK2 and ORBLK2 symbols as well as their underlying schematics. The next step is to instantiate these user-created symbols in the schematic for the ALU block along with the Xilinx-created macro FD4CE to complete the ALU. This section explains this procedure.

### Making the CALC.1 Schematic Visible

First, you must make the CALC.1 schematic visible.

1. Double-click on the SCHEMATIC:CALC.1 icon, shown in Figure 1-92.



Figure 1-92 SCHEMATIC:CALC.1 Icon

The CALC.1 schematic is now displayed.

2. If the CALC.1 schematic does not fill the workspace, click on the up arrow in the upper right corner, as illustrated in Figure 1-93. The CALC.1 schematic now fills the entire workspace.

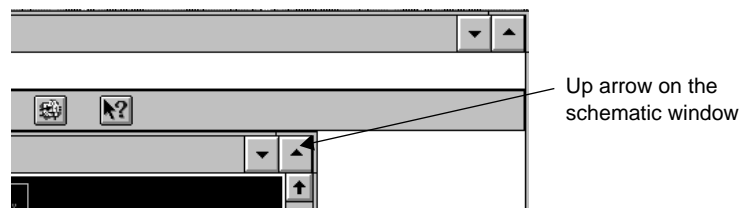
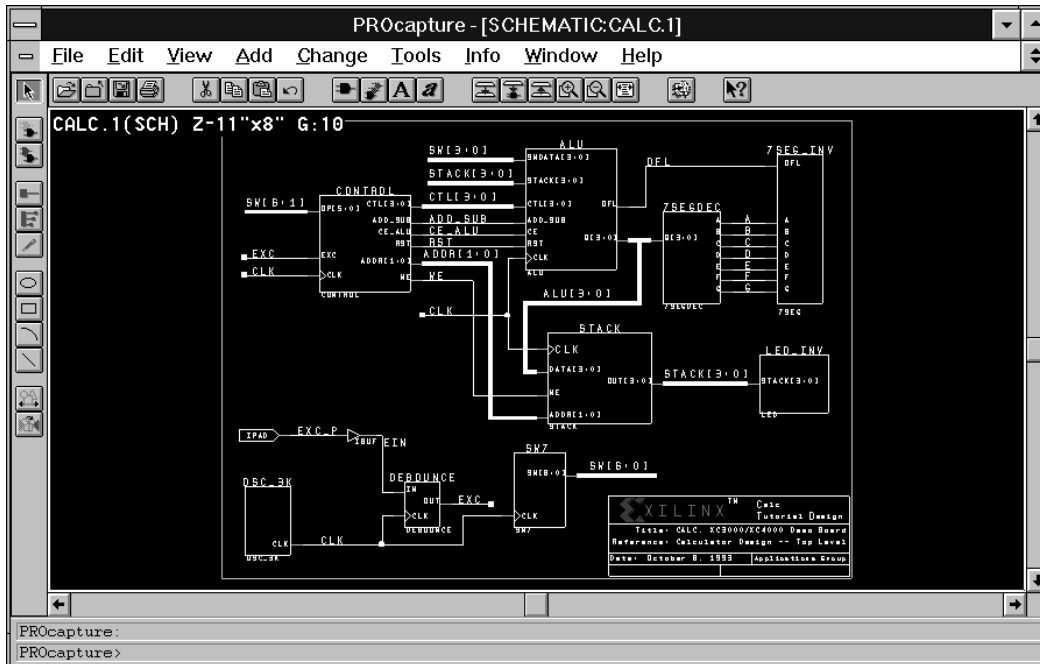


Figure 1-93 Up Arrow on the Schematic Window

3. Press the **F4** key to zoom to full view. Your screen should look like the one pictured in Figure 1-94.



**Figure 1-94 Full CALC.1 Schematic**

The ALU block that must be completed appears in the top middle of the CALC.1 schematic. There are two ways that you can bring up the the ALU.1 schematic for editing.

- You can open the ALU.1 schematic as a separate window using the **File** → **Open** command.
- You can “push into” the ALU symbol on the CALC.1 schematic. Pushing into a symbol’s schematic pushes the schematic into the active window’s “stack.” The next section gives instructions on this procedure.

Each window in the workspace actually displays the schematic or symbol on the top of its display stack. The CALC.1 (SCH) text at the top of the schematic indicates the schematic’s place in the stack.

## Pushing into the ALU Symbol's Schematic

To push into the ALU symbol's underlying schematic, follow this procedure.

1. Select the ALU symbol with the left mouse button.

A bounding box appears around the symbol when it is selected.

2. Select the **View → Push Into Schematic** command.

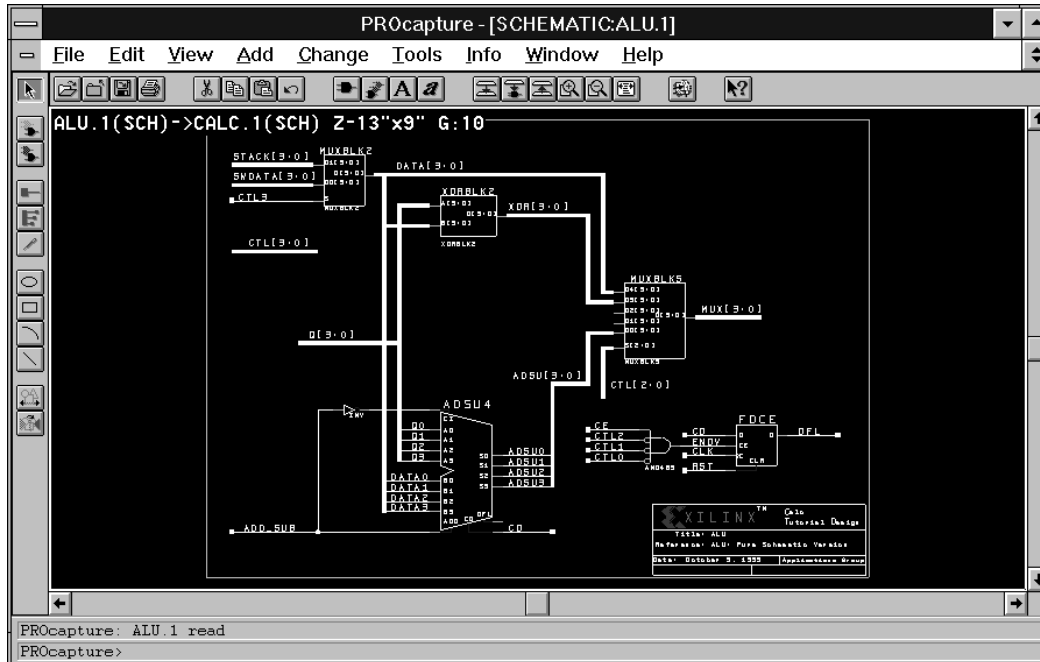
*Keyboard Shortcut:* You can execute the View → Push Into Schematic command by typing **psc** on the PROcapture command line.

*Toolbar Shortcut:* You can execute the View → Push Into Schematic command by clicking on the Push Into Schematic toolbar icon, shown in Figure 1-95.



**Figure 1-95 Push Into Schematic Toolbar Icon**

The View → Push Into Schematic command pushes the ALU schematic onto the top of the active window's display stack and displays the ALU.1 schematic, shown in Figure 1-96.



**Figure 1-96 ALU.1 Schematic**

The text at the top of the schematic window now shows that ALU.1 (SCH) is currently on the top of the display stack with CALC.1 (SCH) after it.

In addition, the name of the PROcapture window has now changed from PROcapture - [SCHEMATIC:CALC.1] to PROcapture - [SCHEMATIC:ALU.1]. This change tells you which schematic you are currently editing.

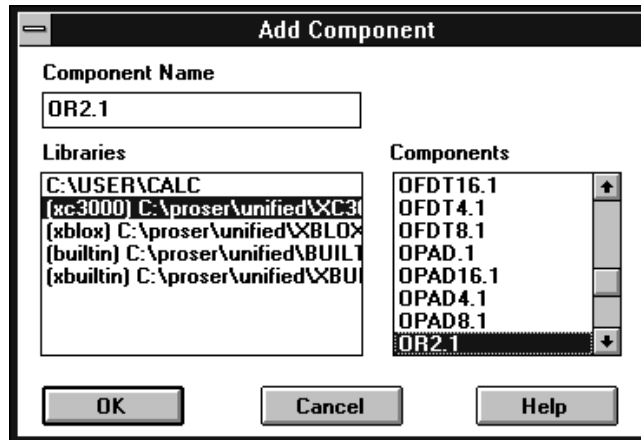
## Placing the ANDBLK2 and ORBLK2 Symbols

Now that the ALU.1 schematic is open and can be edited, you are ready to place the ANDBLK2 and ORBLK2 symbols on the schematic. Place them exactly as you did the AND2 gate from the Xilinx XC3000 library when you created the ANDBLK2.1 schematic.



1. Use the **F9** function key to zoom into the empty area near the center of the schematic.
2. Select the **Add → Component** command.

The Add Component dialog box appears, as shown in Figure 1-97.



**Figure 1-97 Add Component Dialog Box**

3. Select the **C:\USER\CALC** entry in the Libraries list box.

The Components list box displays the components available in the primary library, as Figure 1-98 shows.

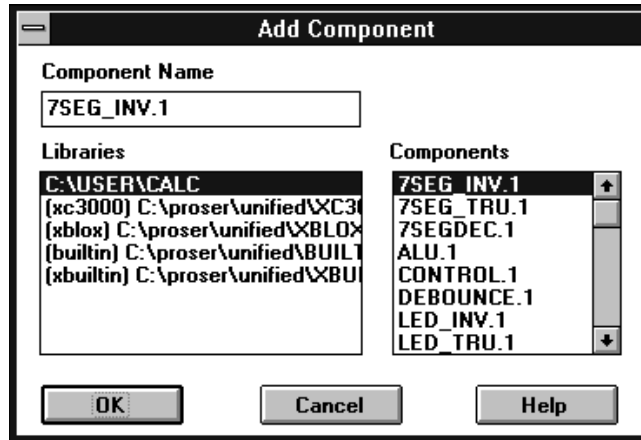


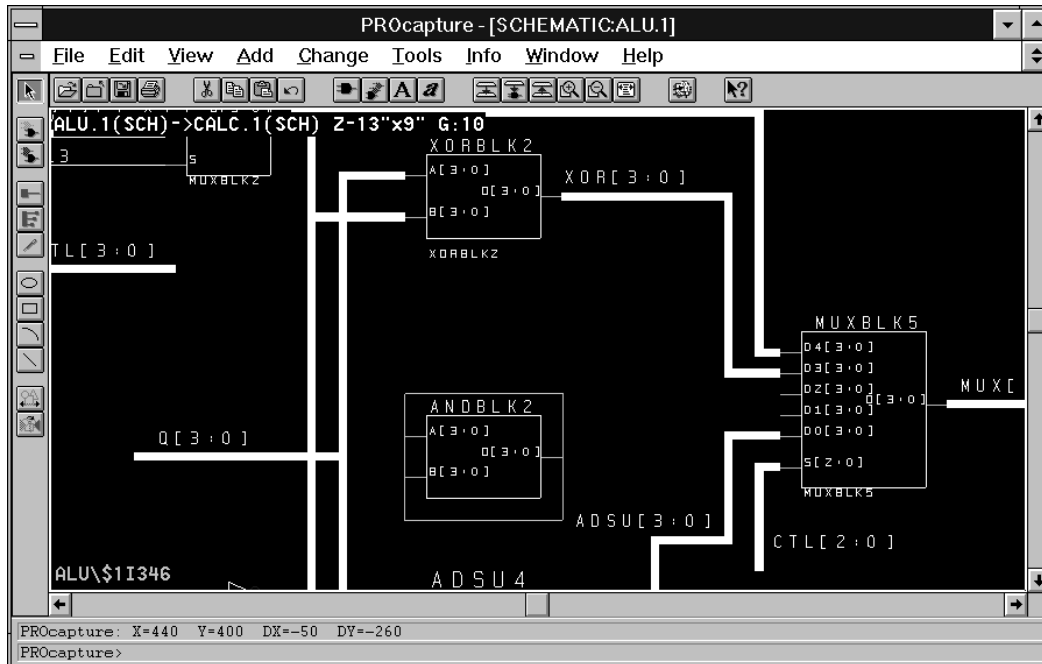
Figure 1-98 Selecting the Library

4. Select the **ANDBLK2.1** component in the Components list box.
5. Click on **OK**.

The Add Component dialog box closes and the ALU.1 schematic window is reactivated. An outline of the component being added appears at the end of the pointer.

**Note:** You can also add the ANDBLK2 component by typing `comp andblk2.1` on the PROcapture command line.

6. To place the ANDBLK2 component, as shown in Figure 1-99, click the left mouse button on the desired location in the schematic window.



**Figure 1-99 Placing the ANDBLK2 Component**

7. Double-click the left mouse button in an unused area of the schematic to bring up the Add Component dialog box to add the ORBLK2 symbol. Place it as shown in Figure 1-100.

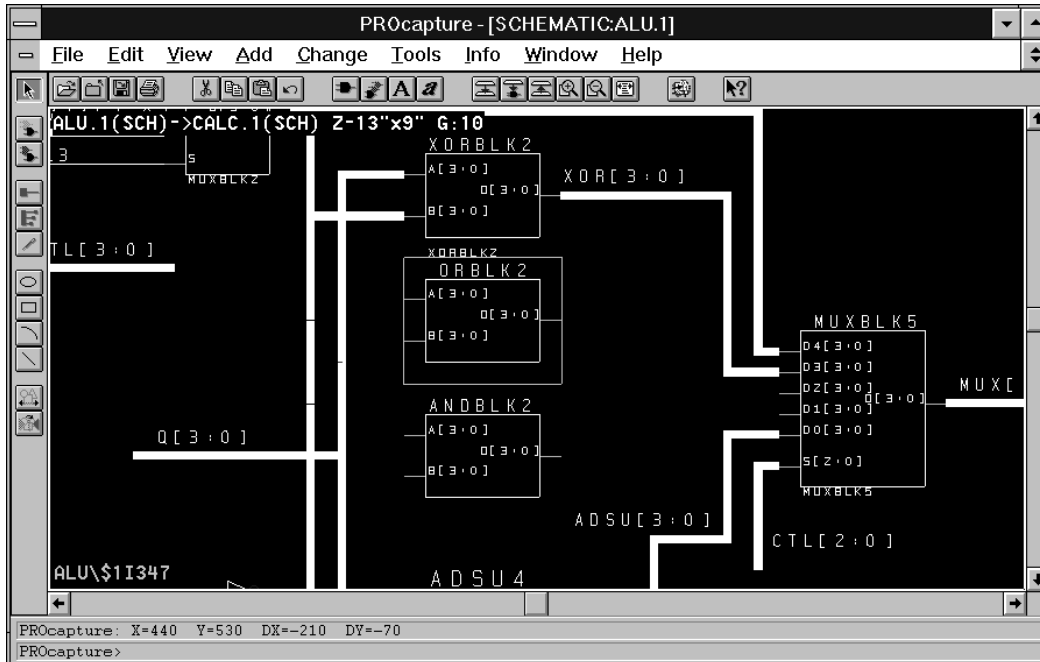


Figure 1-100 Placing the ORBLK2 Component

## Placing the FD4CE Component

The other component missing from the ALU.1 schematic is the FD4CE component, a set of four flip-flops with clock enable, from the Xilinx XC3000 component library. This component can also be found in the XC4000 and XC7000 libraries so that retargeting is as simple as changing the alias of the component on the schematic.

Follow this procedure to add the FD4CE component to the ALU.1 schematic.

1. Use the **F4** function key to zoom to full view and then use the **F9** function key to zoom into the open area in the upper right corner of the ALU.1 schematic.

2. Double-click in an open area with the left mouse button.

The Add Component dialog box comes up.

3. Select the FD4CE.1 component after selecting the XC3000 or XC7000 library in the Libraries list box.
4. Click on **OK**.

Selecting OK closes the Add Component dialog box and re-activates the ALU.1 schematic window. An outline of the component being added appears at the end of the pointer.

**Note:** You can also add the FD4CE.1 component by typing `comp fd4ce.1` on the PROcapture command line.

5. To place the FD4CE.1 component, as shown in Figure 1-101, click the left mouse button on the desired location in the schematic window.

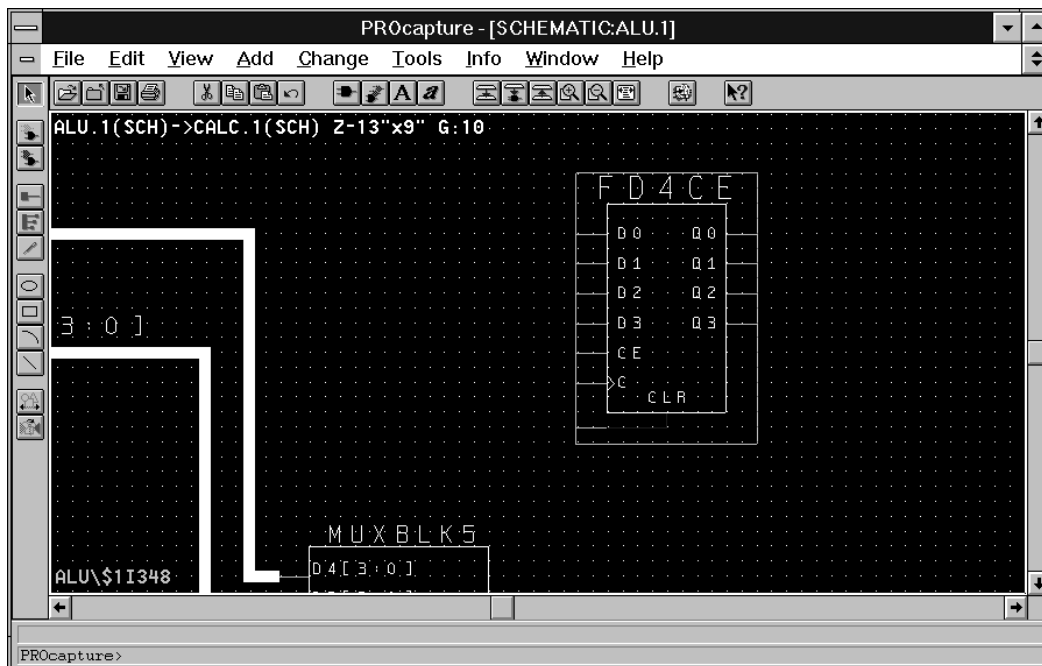


Figure 1-101 Placing the FD4CE Component

## Adding Nets, Buses, and Labels

All the missing components have now been added to the ALU.1 schematic. The next step is to add the missing nets, buses, and labels.

1. Add the necessary nets, buses, and labels to complete the connections for the FD4CE component, as shown in Figure 1-102.

Inputs to the FD4CE component are MUX[3:0], CE, CLK, and RST.

Outputs of the FD4CE component are Q[3:0].

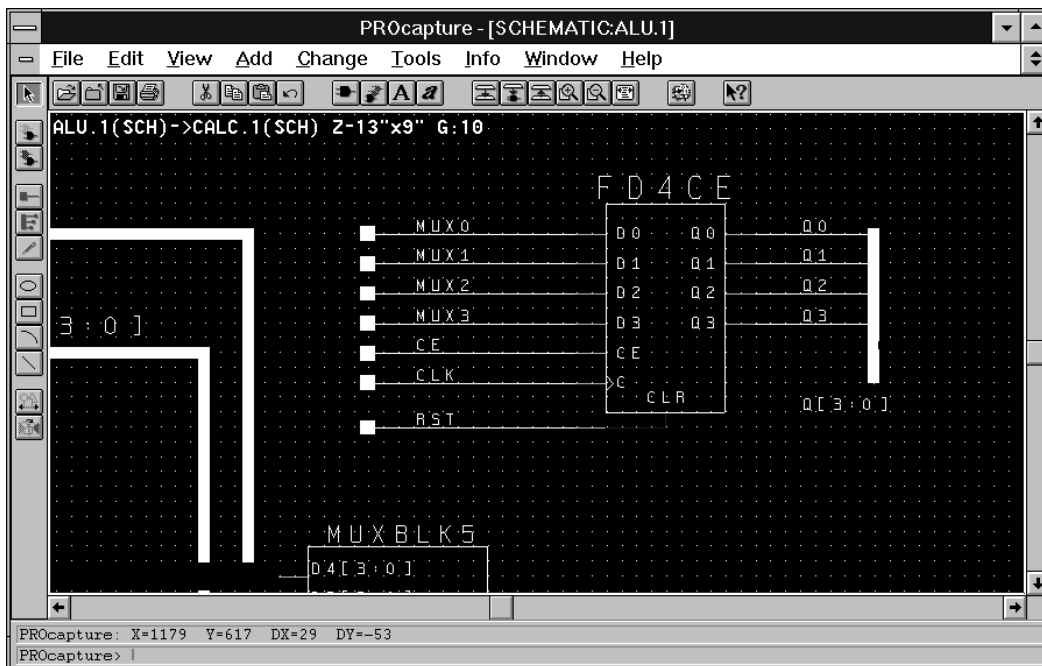


Figure 1-102 Completed FD4CE Connections

2. Add the necessary nets, buses, and labels to complete the connections for the ANDBLK2 and ORBLK2 components, as shown in Figure 1-103.

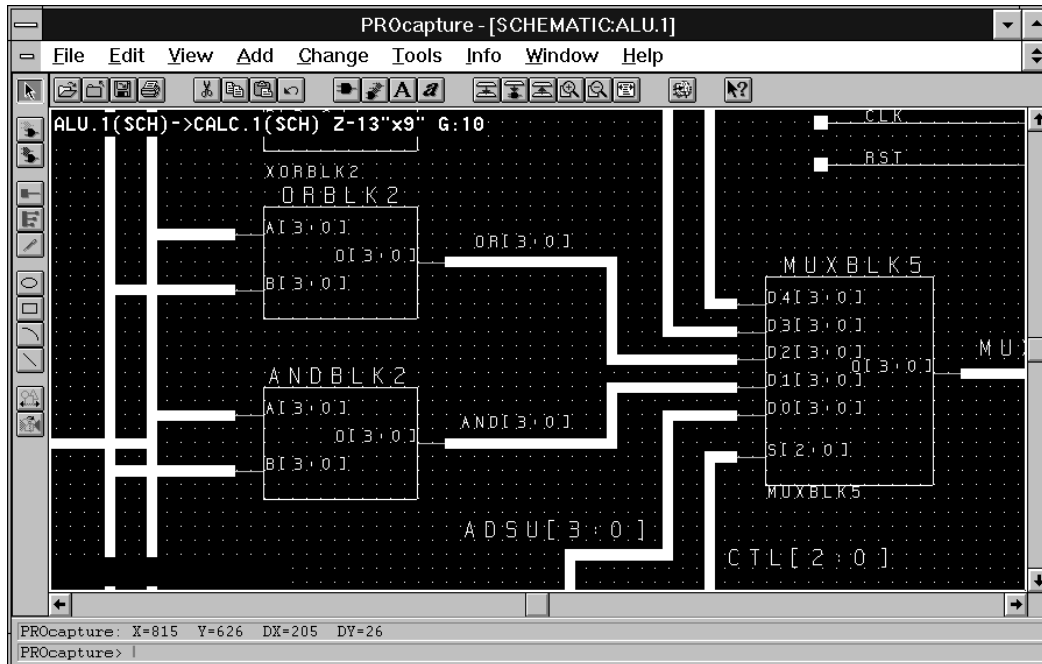


Figure 1-103 Completed ANDBLK2 and ORBLK2 Connections

### Adding Symbol Labels

Using the left mouse button, select the ANDBLK2 symbol.

Notice the text in the lower left corner of the ALU.1 schematic window, as shown in Figure 1-100: ALU\ \$1I347. This is the label or instance name of the currently selected ANDBLK2 symbol. Labels or instance names follow this format:

*block\_label\block\_label\default\_label*

In the case of the ALU\ \$1I347 label, "ALU" is the instance name of the hierarchy that contains the ANDBLK2 symbol; this hierarchy only has one level. The "\$1I347" notation is the default instance name given to the instantiated ANDBLK2 symbol itself. The instance name for your ANDBLK2 symbol may be slightly different.

It is a good idea to define the instance name of each symbol placed on a schematic to make debugging your design easier. Error and warning messages often refer to component labels, and the labels also appear in the simulation netlists. Placing a label on ANDBLK2 replaces this default name with the label string. The name currently on the symbol is just text and is ignored by the software.

In the ALU.1 schematic, there are already labels on the MUXBLK2, XORBLK2, and MUXBLK5 blocks. You can choose not to label the blocks that are library components and just treat them as black boxes.

1. Using the left mouse button, select the ANDBLK2 symbol.
2. Select the **Add** → **Object Label** command.
3. Enter **ANDBLK2** in the Text field of the Add Label dialog box.
4. Click on **OK**.
5. Point the mouse to the desired location and click the left mouse button to place the label.
6. Repeat steps 1 through 5 to add an ORBLK2 label to the ORBLK2 symbol.

Figure 1-104 illustrates the labeled ANDBLK2 and ORBLK2 blocks.



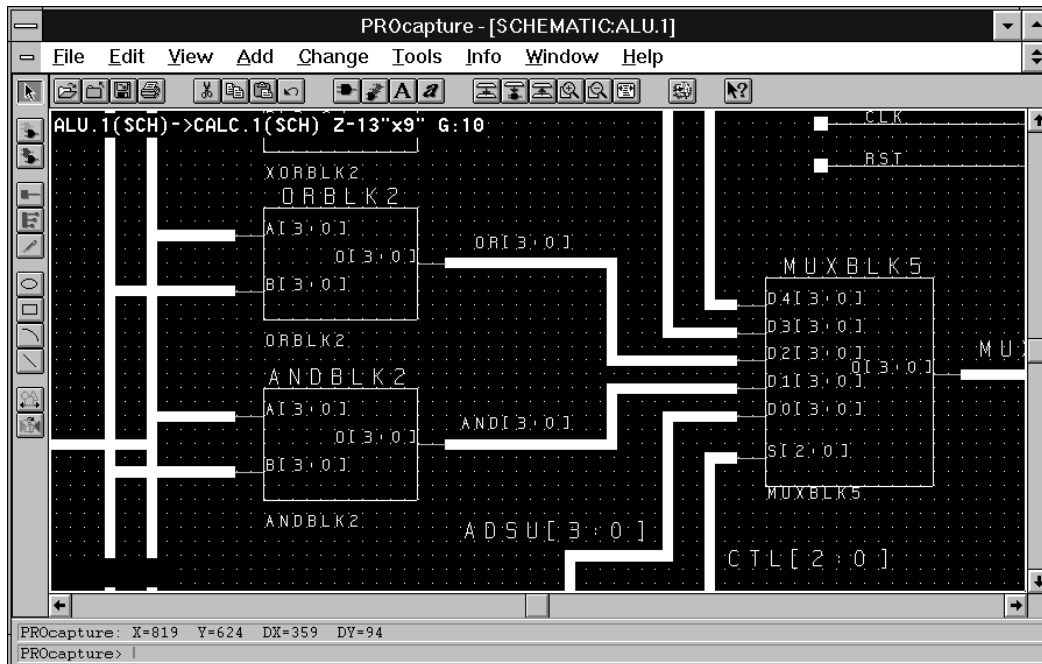


Figure 1-104 Labeled ANDBLK2 and ORBLK2 Blocks

## Saving the ALU.1 Schematic

Use the **File** → **save** command to save the ALU.1 schematic.

The ALU.1 schematic is now complete. You can explore the various components by using the **View** → **Push Into Schematic** and **View** → **Pop** commands.

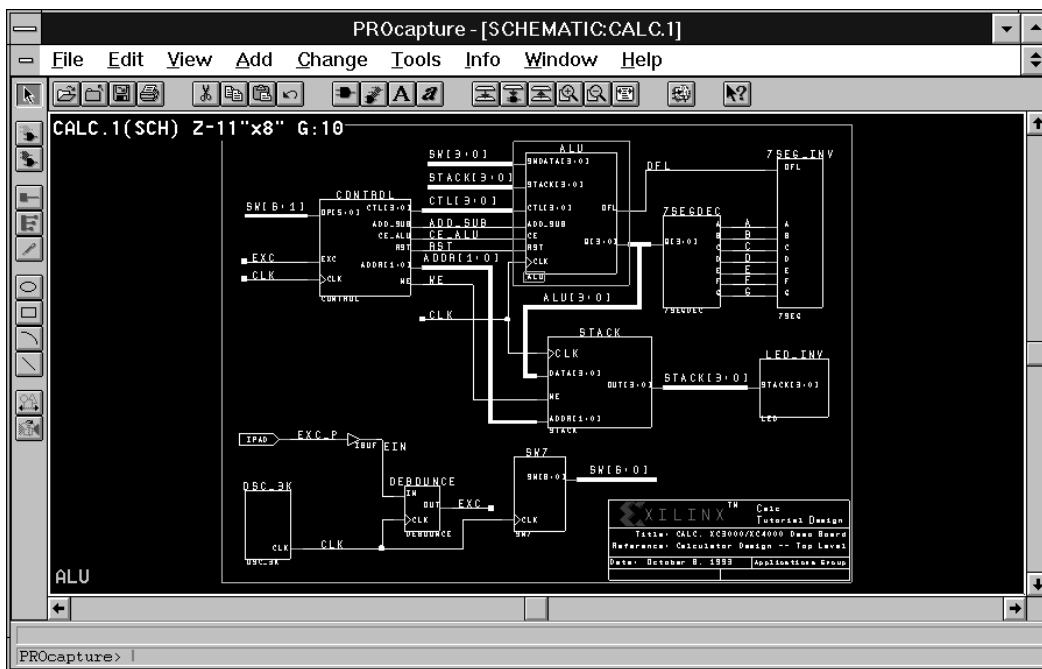
**Keyboard Shortcut:** You can execute the **View** → **Pop** command by typing **pop** on the PROcapture command line.

**Toolbar Shortcut:** You can execute the **View** → **Pop** command by clicking on the Pop Schematic toolbar icon, shown in Figure 1-105.



**Figure 1-105 Pop Schematic Toolbar Icon**

Once you have become familiar with the ALU.1 schematic and its components, pop out of the ALU.1 and return to the CALC.1 schematic, shown in Figure 1-106.



**Figure 1-106 CALC.1 Schematic**

## Viewing the OSC\_3K or OSC\_7K Schematic

The FPGA and XC3000A demonstration boards have a built-in RC circuit for clock generation for the XC3000 family devices. The OSC\_3K block contains an oscillator that connects to that circuit. The frequency of the output varies from device to device because of

process differences, so it is not suitable for many applications, but it is adequate for clocking a human interface. The EPLD (XC7000) version of the CALC design has an equivalent oscillator block called OSC\_7K.

To view the OSC\_3K or OSC\_7K schematic, push into the OSC\_3K or OSC\_7K symbol, respectively, in the lower left corner of the CALC.1 schematic.

Figure 1-107 displays the OSC\_3K schematic.

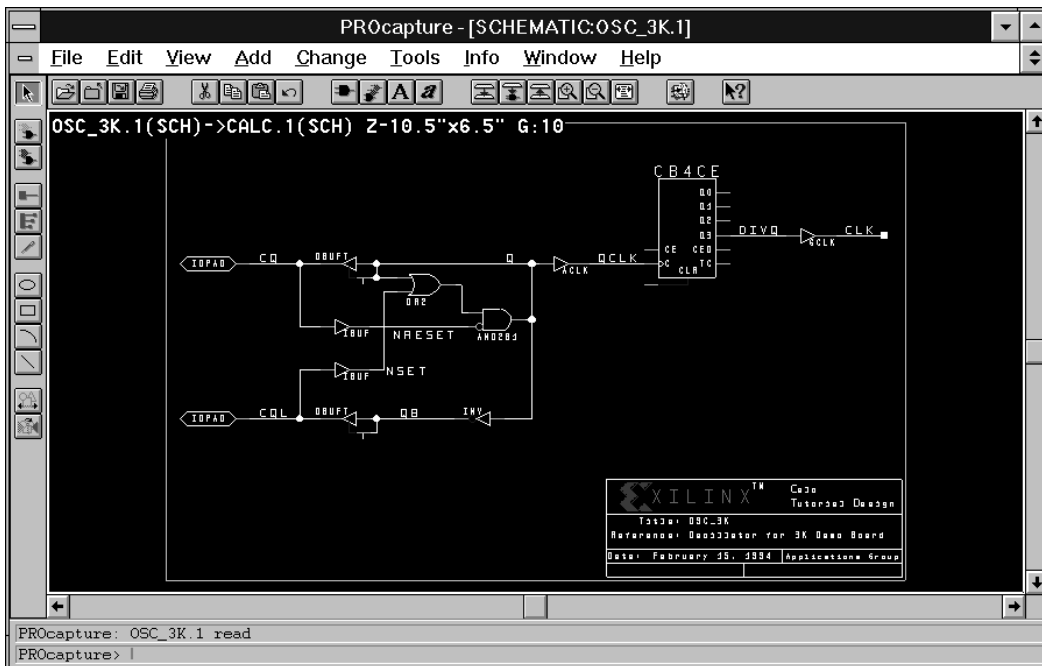


Figure 1-107 OSC\_3K Schematic

Figure 1-108 represents the OSC\_7K schematic.

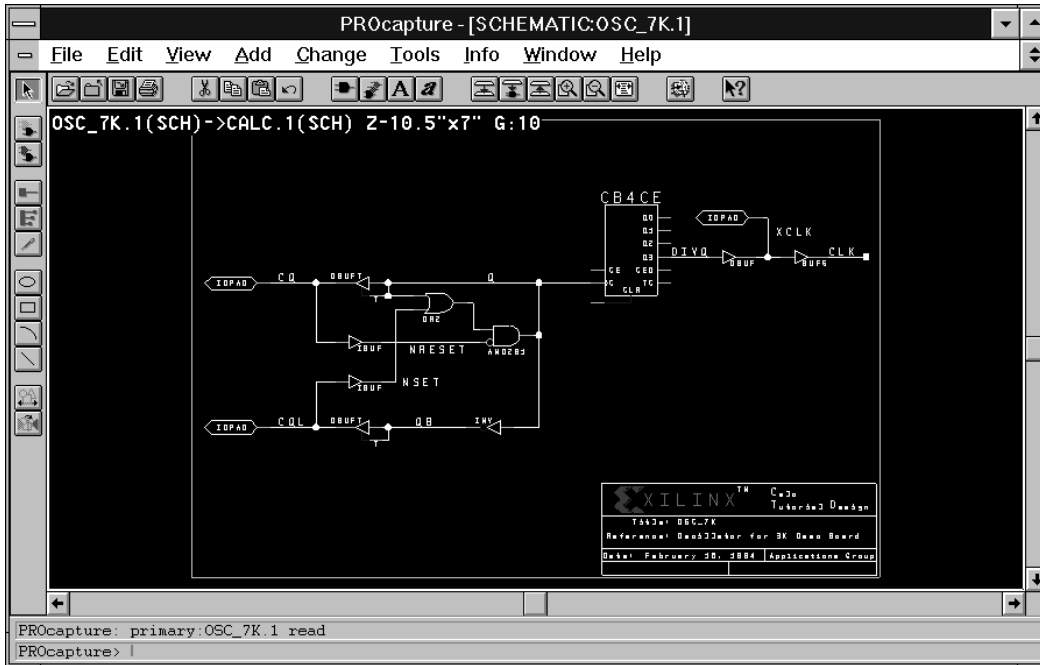


Figure 1-108 OSC\_7K Schematic

The output of the oscillator circuit in OSC\_3K or OSC\_7K 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 with minimal delay and very low skew. In addition, using the dedicated clock nets frees up programmable interconnect for use by other signals in the design.

In the OSC\_3K schematic, the GCLK symbol at the right is the Xilinx primitive for the XC3000 primary 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 circuit on the clock, although it is not really necessary.

In the OSC\_7K schematic, the BUFG symbol at the right is the Xilinx primitive for the XC7000 global clock buffer. The signal is routed through an I/O pad before being passed to the global clock buffer so that the clock signal is visible during timing simulation. This step is necessary because the EPLD fitter eliminates many internal signals during logic optimization.

In general, you should use at least one clock buffer (GCLK or ACLK) in every clocked XC2000 or XC3000 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. For clocked XC7000 designs, you should use at least one BUFG clock buffer.

Select the **View** → **Pop** command to return to the CALC.1 schematic.

## **Exchanging Components**

The FPGA demonstration board is designed so 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.1 schematic is set up for this configuration as the default.

The XC3000A demonstration board is designed in such a way that the display elements are turned on when the applied 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 XC3000A demonstration board. This board has only a single FPGA socket, and the socket contains an XC3000 family part in a PC68 package.

If your target board is the XC3000A demonstration board, use the following steps to exchange components. If you are targeting an EPLD, skip to step 3.

1. Using the left mouse button, select the 7SEG\_INV symbol.
2. At the PROcapture command line, type `cc 7SEG_TRU`.

3. Using the left mouse button, select the LED\_INV symbol.
4. At the PROcapture command line, type `cc LED_TRU`.

Figure 1-109 shows the swapping of these components.

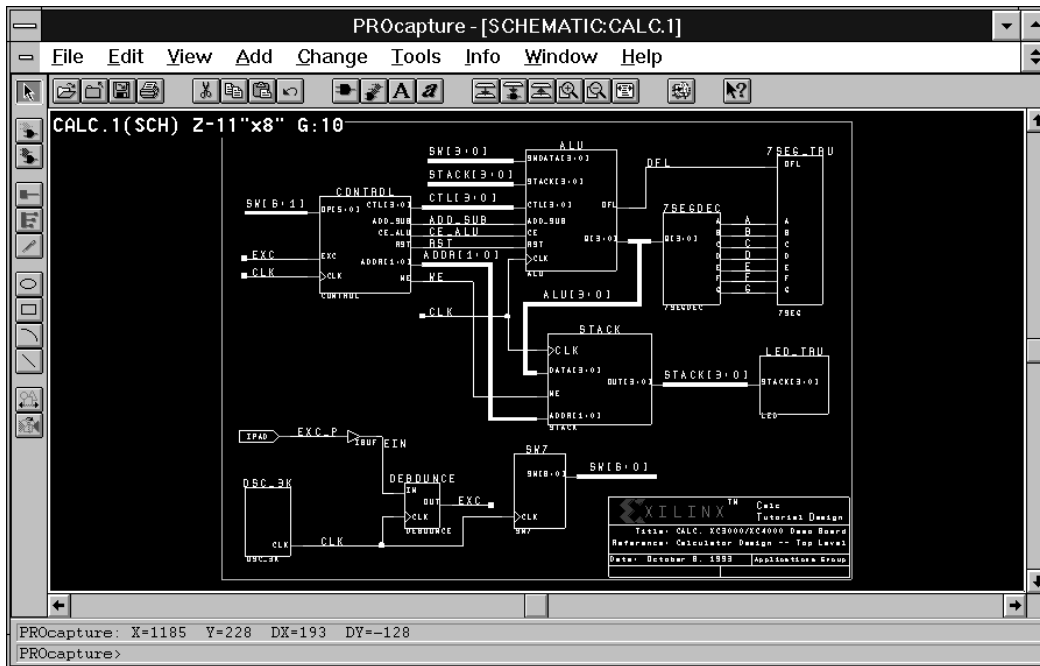


Figure 1-109 Swapped Components

## Controlling Layout from the Schematic

You can use attributes in the schematics to control the placement and routing of your FPGA design or the fitting of your EPLD design. Some attributes control the speed of the inputs and outputs, and others control the actual placement of the pins in the design.

The following procedures add the needed attributes to the Calc design with the first attributes defining the pin placement of the I/Os. It is highly recommended that you let the automatic placing and routing programs, PPR and APR for FPGAs, or the automatic fitting program for EPLDs, define the pinout. Locking the I/Os can

prevent the programs from placing and routing, or fitting, the design completely or can reduce design performance.

However, I/O pin numbers are assigned to the tutorial schematics so that the Calc design functions in the Xilinx demonstration boards. Because the design is fairly simple, these pin assignments do not adversely affect the ability of the software to place and route the design completely. The attribute used to lock down an I/O pin to a particular pad on the device is the LOC attribute.

### Adding the LOC Attribute

The following procedure adds a LOC attribute to the EXC\_P signal.

1. Using the left mouse button, double-click on the IPAD symbol connected to EXC\_P.

Double-clicking on the IPAD symbol displays the Edit Attributes dialog box, shown in Figure 1-110.

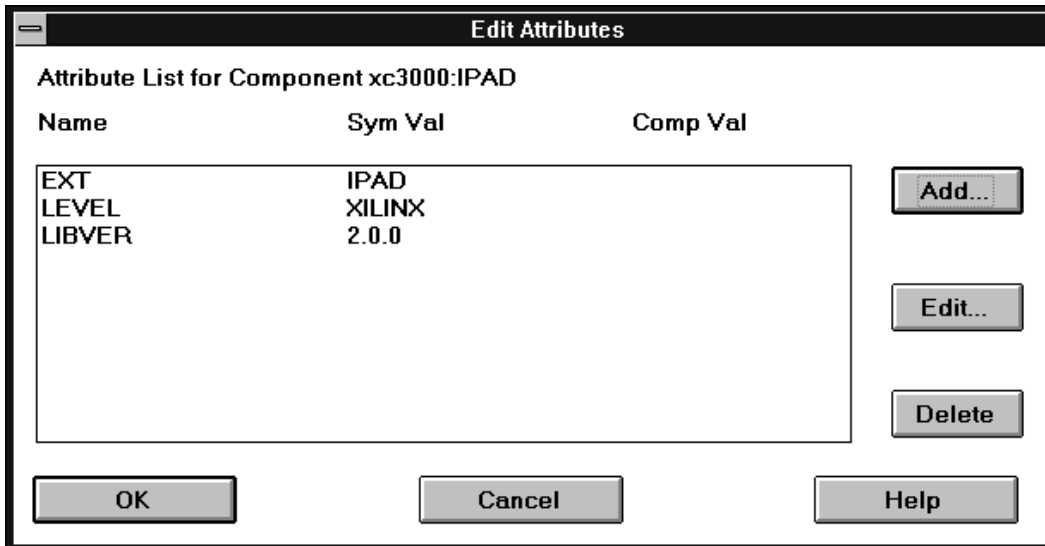
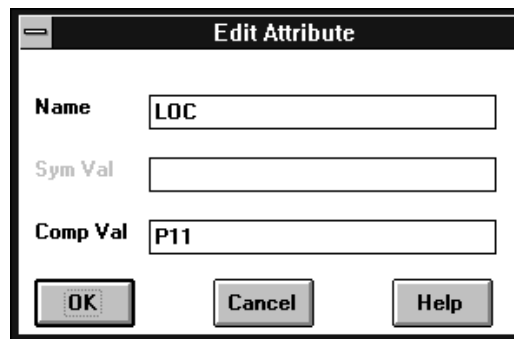


Figure 1-110 Edit Attributes Dialog Box

The attributes list box already contains the EXT, LEVEL, and LIBVER attributes assigned their default values. These attributes have been added by Xilinx and cannot be removed.

2. Click on **Add**.

This command brings up the Edit Attribute dialog box, shown in Figure 1-111.



**Figure 1-111 Edit Attribute Dialog Box**

3. In the Name field, type **LOC**.
4. In the Comp Val field, type **P11** for either the XC3000A version or the XC7000 version.
5. Click on **OK**.

Selecting OK closes the Edit Attribute dialog box and updates the Edit Attributes dialog box, as Figure 1-112 illustrates.

The value of the added LOC attribute is in the Comp Val column, because the attribute was added to the actual IPAD instance on the schematic, not to the symbol in the library. The three initial attributes are defined on the symbol in the library and appear on every instantiation of the IPAD, whereas the LOC attribute that was just added only appears on the explicitly selected IPAD.



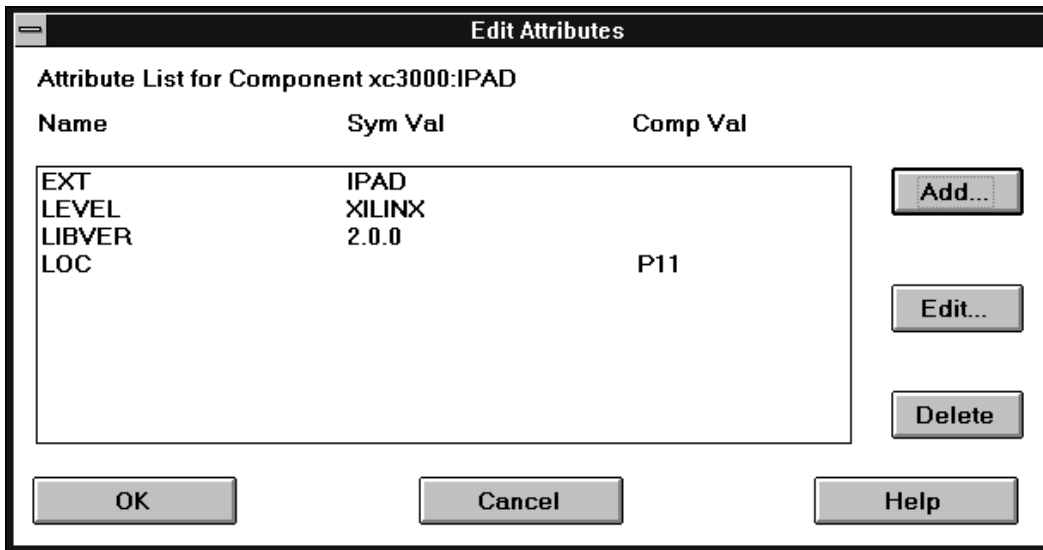
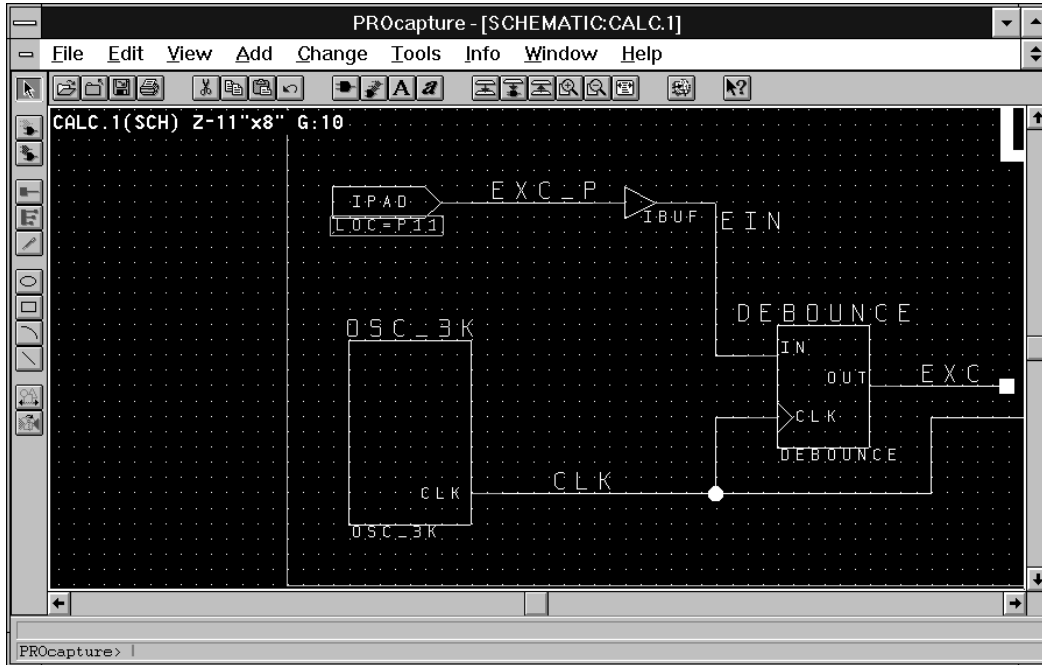


Figure 1-112 Updated Edit Attributes Dialog Box

6. Click on OK.

The Edit Attributes dialog box now closes. In the updated CALC.1 schematic window, the LOC=P11 attribute is placed under the selected IPAD component, as shown in Figure 1-113.



**Figure 1-113 CALC.1 Schematic with LOC Attribute**

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 alphanumeric designations such as A12. *The Programmable Logic Data Book* lists the pinouts of each device for each package that Xilinx supplies.

## Adding Flags to Nets

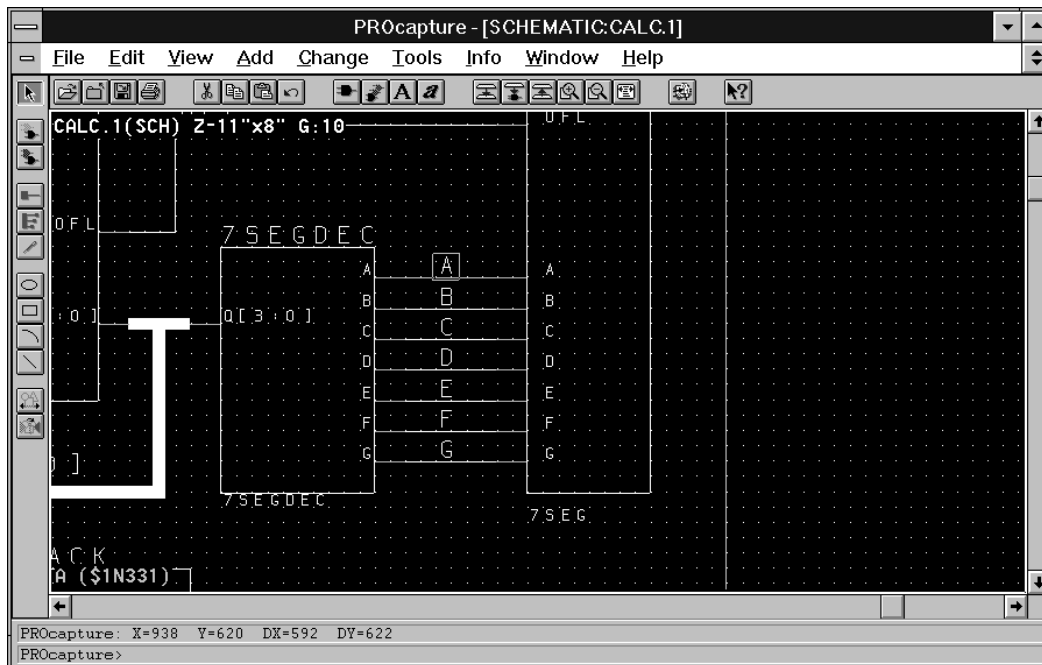
Suppose that you want to make sure that a net in your FPGA design is not absorbed into a CLB. You can attach an "Explicit" or "External" (X) flag to the net. A flag is merely an attribute with a name but no value.

**Note:** The X flag is not valid for EPLD designs. However, EPLD designs can use F and H flags to map logic to Fast Function Blocks (F) or High-Density Function Blocks (H). If you are using the XC7000 version of Calc, you can follow this part of the tutorial and apply the F flag instead of the X flag.

Use the following procedure to add the X flag to the A net feeding the 7SEG\_INV or 7SEG\_TRU component.

1. Using the left mouse button, select the net.

The net's color changes, and the label's bounding box is displayed, as shown in Figure 1-114.

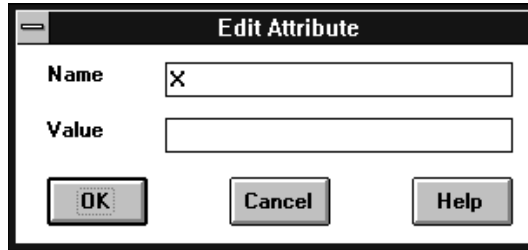


**Figure 1-114 Selected Net**

2. At the PROcapture command line, type `cat`.  
The Edit Attributes dialog box appears.
3. In the Edit Attributes dialog box, click on **Add**.

The Edit Attribute dialog box appears.

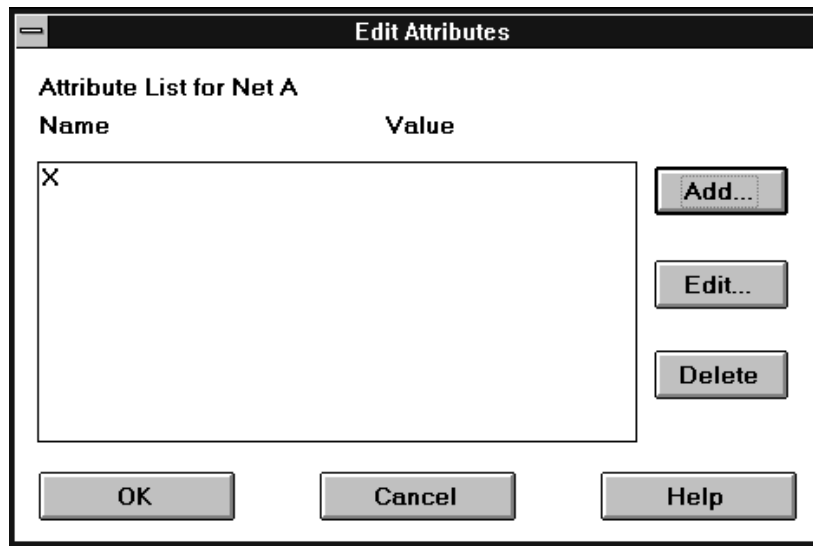
4. In the Name field, type **x** for FPGAs, as shown in Figure 1-115, or **ⓧ** for EPLDs.



**Figure 1-115 Adding the X Flag**

5. Click on **OK**.

Selecting OK closes the Edit Attribute dialog box and updates the Edit Attributes dialog box, which now displays the X flag, as Figure 1-116 indicates.

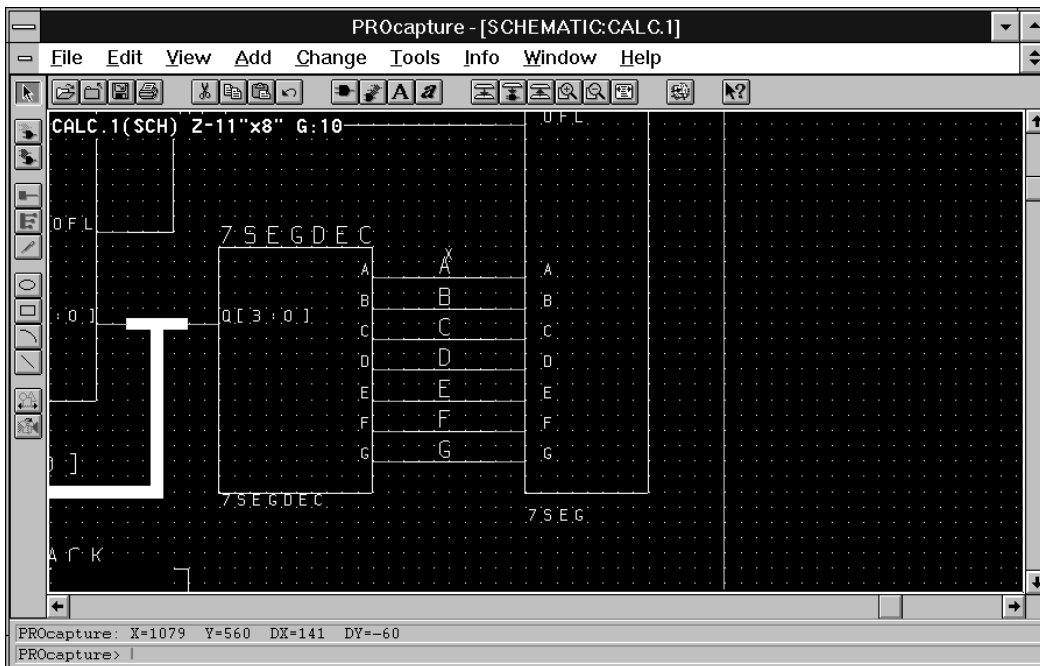


**Figure 1-116 Updated Edit Attributes Dialog Box**

6. Click on **OK**.

The Edit Attributes dialog box closes, and the CALC.1 schematic window is updated. The X flag is placed on the schematic next to the label, as shown in Figure 1-117.

**Note:** The X flag will be the same size as the last text added.



**Figure 1-117 Placing the X Flag**

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, Constraints, and Carry Logic” chapter of the *Libraries Guide*.

## Adding the FAST and SLOW Attributes

It is easy to over-constrain 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. Some attributes do not constrain the design but merely allow you to control how the FPGA is used in the final design. Two such attributes are the FAST and SLOW attributes used to control the transition time of device output pins.

The FAST and SLOW attributes modify the output slew rate. They are applied to the OPAD or output buffer. The default slew rate is SLOW. Pads programmed as “fast” have different timing specifications from “slow,” or slew-rate-limited, pads and draw more current.

**Note:** The FAST and SLOW attributes apply to the XC73144 EPLD device, but not to any other EPLD devices. Because the XC7000 version of the Calc design targets an XC73108 device, this section is not applicable to EPLDs.

To add the FAST attribute to the OFL\_P pad in the 7SEG\_INV or 7SEG\_TRU schematic in the Calc design, follow these instructions.

1. Using **View** → **Push Into Schematic**, push into the 7SEG\_INV or 7SEG\_TRU component.
2. Using the left mouse button, double-click on the OPAD component attached to the net labeled “OFL\_P.”
3. In the Edit Attributes dialog box, click on **Add**.
4. In the Name field of the Edit Attribute dialog box, type **FAST**, as shown in Figure 1-118.

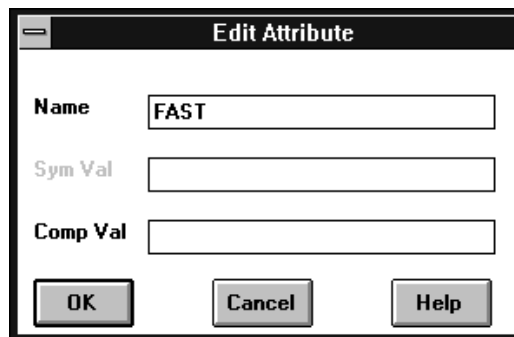
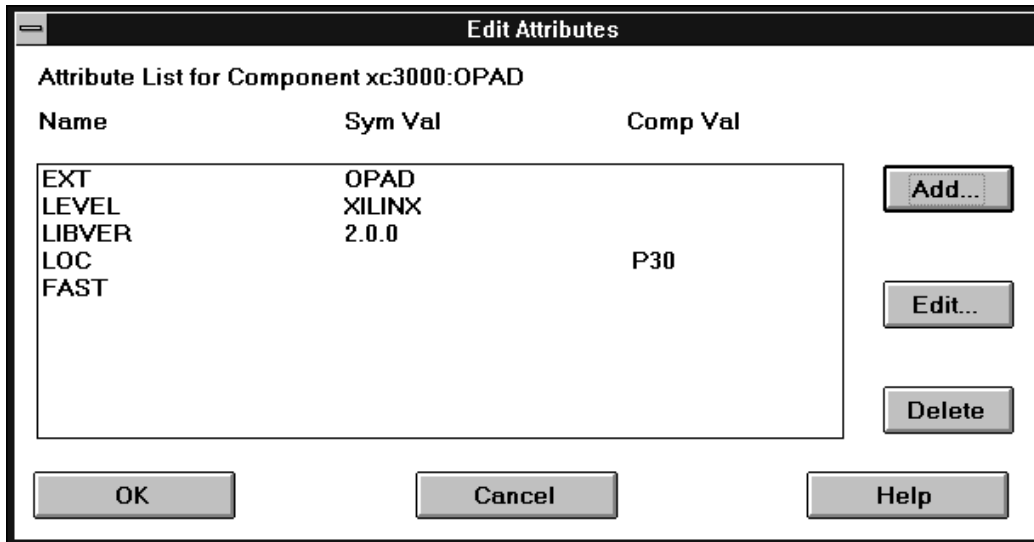


Figure 1-118 Adding the FAST Attribute

5. Click on **OK**.

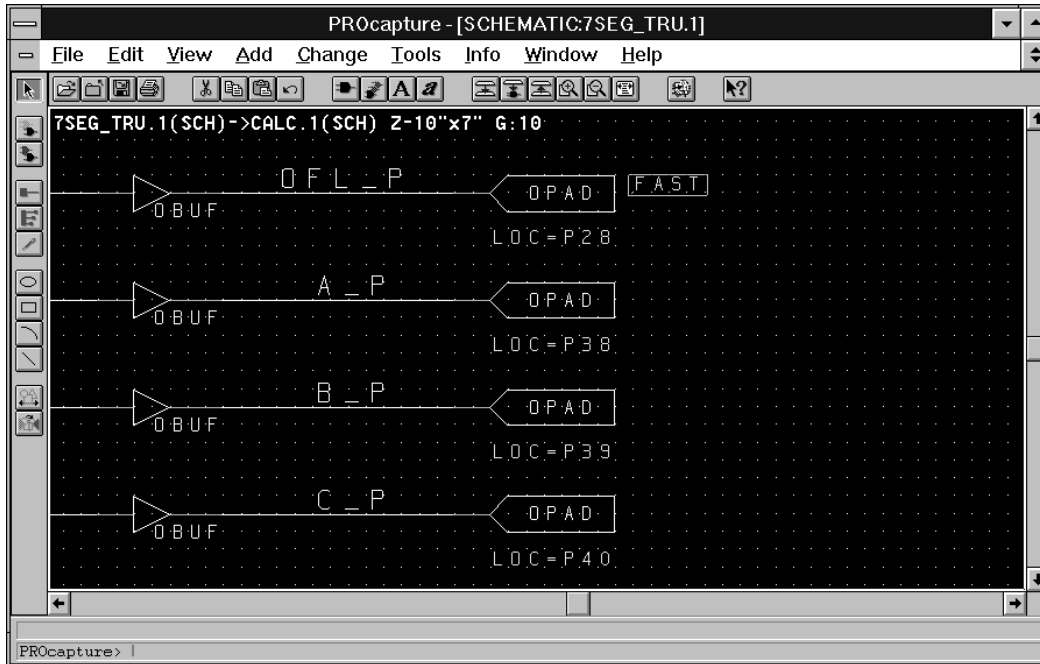
The Edit Attribute dialog box closes, and the updated Edit Attributes dialog box appears, as Figure 1-119 shows.



**Figure 1-119 Updated Edit Attributes Dialog Box**

6. Click on **OK**.

The Edit Attributes dialog box closes, and the current schematic is updated. The FAST attribute is placed on the schematic next to the OPAD, and you can move it to clean up the schematic, as pictured in Figure 1-120.



**Figure 1-120 Placing the FAST Attribute**

7. Save the changes made to the schematic by selecting the **File** → **Save** command.
8. Using the **View** → **Pop** command, return to the CALC.1 schematic.

See *The Programmable Logic Data Book* for timing specifications for the various types of pads.

## Using IOB Flip-Flops

The Xilinx XC3000, XC4000, and XC4000A devices have two registers in each IOB (I/O block). Each pad has an associated input register and output register. You can configure an input register as a flip-flop or as a latch. Output registers can only be flip-flops but can optionally have a tristate control on their output. You can access these I/O registers by using special library components called IFD, ILD, OFD,



and OFDT. The XC7000 devices have a flip-flop or latch at each input, accessible through an IFD, ILD, or IFDX1 component. For more information on these elements, consult the *Xilinx Libraries Guide*.

Using the IOB registers frees up internal resources, which can help when designs are large in proportion to the device size or, in the case of some FPGA designs, are difficult to route. The SW7 component's underlying schematic uses the input flip-flops to store the data from the switches. You can push into the SW7 schematic to view where and how these special components are used.

1. Using the **View** → **Push Into Schematic** command, push into the SW7 component's underlying schematic. This schematic appears in Figure 1-121.

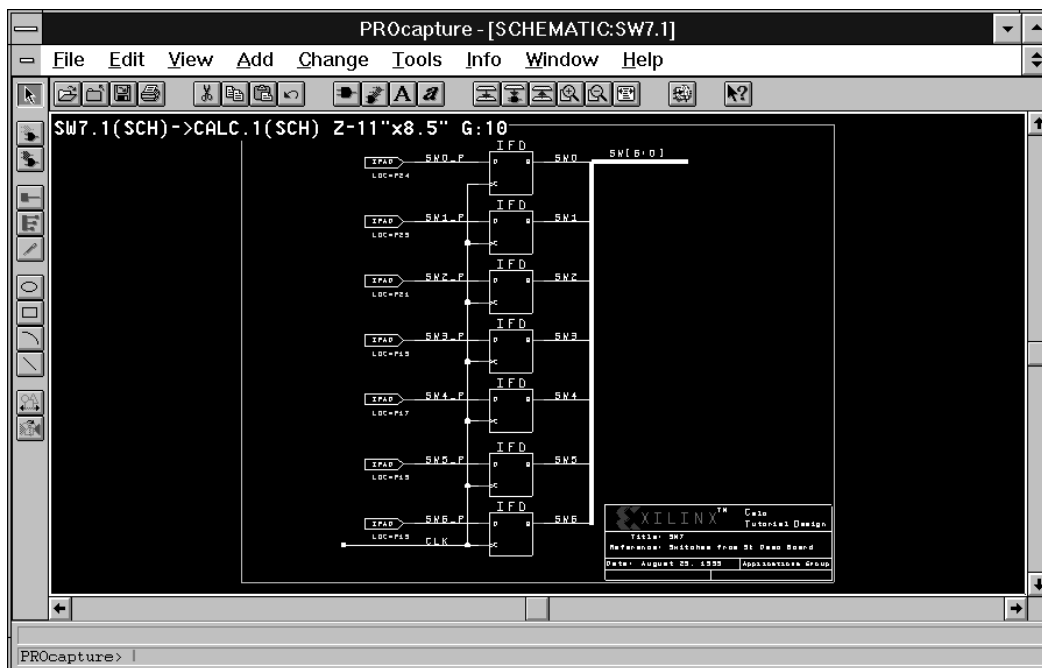


Figure 1-121 SW7 Schematic

2. Using the **View** → **Pop** command, return to the CALC.1 schematic.
3. Save the changes made to the CALC.1 schematic.

## Functional Simulation

Now that the Calc design is complete, you can functionally verify it by simulating it in PROsim. PROflow controls the flow of the tools to prepare the simulation network for the design and invokes PROsim. You can also invoke PROwave, the Viewlogic waveform viewer, from PROflow to view the simulation signals in a waveform format while simulating.

### Creating the Simulation Network

The first step in the functional simulation process is to create the simulation network, calc.vsm, which is loaded into PROsim to simulate the Calc design.

1. If you previously closed PROcapture, re-invoke it by clicking on the PROflow Design Entry icon and then clicking on **OK**.
2. Return to PROflow by making an icon of PROcapture and selecting PROflow with the left mouse button.
3. Click on the PROflow Functional Simulation PROsim icon, shown in Figure 1-122:



**Figure 1-122 PROsim Icon**

The Functional Simulation dialog box appears, pictured in Figure 1-123.

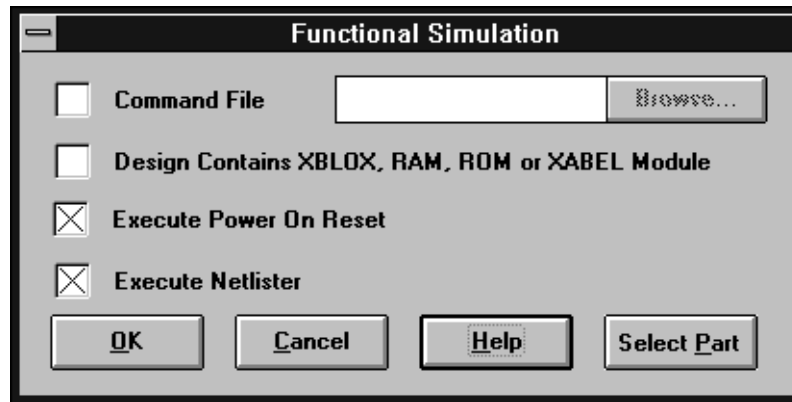


Figure 1-123 Functional Simulation Dialog Box

Notice that the Execute Power On Reset option is selected. When PROflow invokes the simulator, it executes the commands necessary to simulate the startup sequence when the global reset is released. The global reset signal used for XC2000, XC3000, and XC3000A devices is GR. A similar net, GSR, is used for the XC4000 family. The XC7000 family uses the PRLD global net as the power-on-reset signal.

The Command File and the Design Contains XBLOX, RAM, ROM or XABEL Module options are not selected. Command files are discussed at the end of this chapter. The latter option is only used if the design being processed contains X-BLOX, RAM, ROM, or Xilinx ABEL modules. If the design uses any of these special components, PROflow runs the XSimMake program to create the simulation network. For straight schematic designs such as Calc, PROflow runs the VSM program instead to create the simulation network. For more information on how to use XSimMake, see the “Functional Simulation” chapter of the *Viewlogic Interface Guide*.

4. Click on **OK**.

Because the Execute Netlister option is set, PROflow now runs the VSM program to create the calc.vsm simulation network. When VSM completes, PROflow invokes Notepad, a Windows program that lets you view and edit files, to display the VSM log file. A partial log file is shown following.

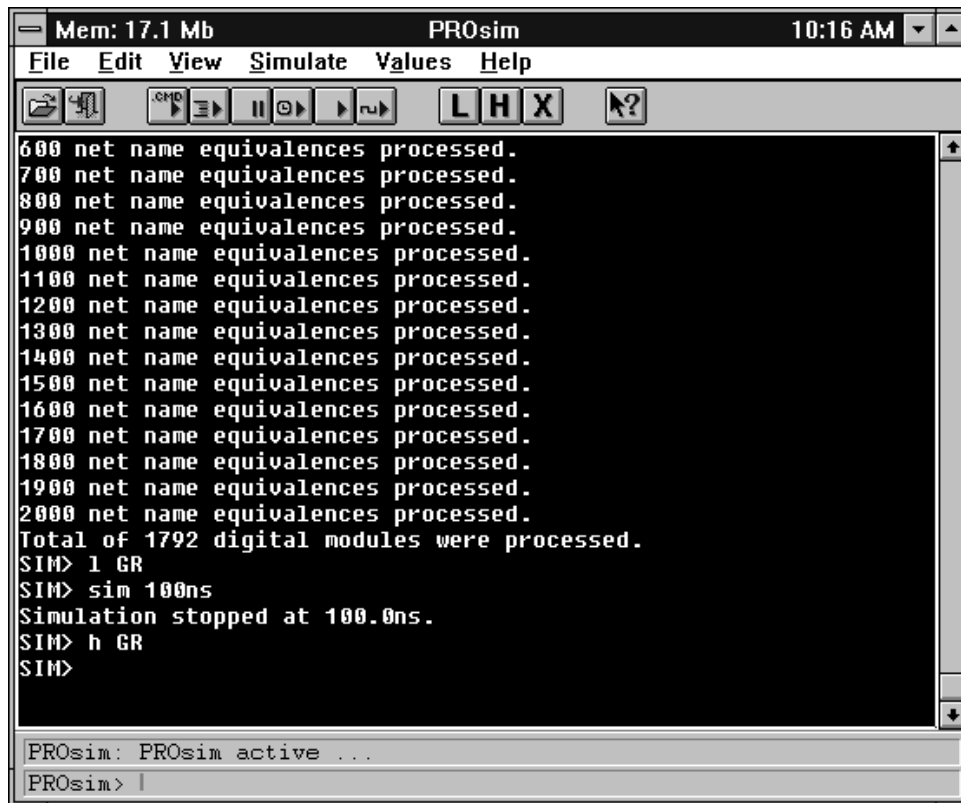
```
Completed file calc.vsm
1792 module(s), 1660 net(s), 2024 net
equivalence(s)
0 error(s) and 0 warning(s) in file calc.vsm
```

5. Verify that no errors or warnings were generated.

If VSM did not complete successfully, bring up PROcapture and correct the schematic that generated the error or warning message, then re-invoke VSM using the PROsim icon in PROflow as you did previously.

6. Select the **File** → **Exit** command to close Notepad.

Once you close Notepad, PROflow invokes PROsim and loads the calc.vsm simulation network. Figure 1-124 displays the processing messages that appear on the screen while the simulation network is being loaded and PROflow is simulating the startup sequence.



**Figure 1-124 Reading in the Simulation Network File**

After PROsim processes the net name equivalents, it forces the global reset net Low for 100 ns and then forces it High. This operation is required for simulation; it simulates the device's startup sequence when the device resets its registers to zero. The device on the board always resets. If these steps are not performed, the simulation output is not correct; all flip-flop outputs remain in an indeterminate state, even though valid inputs and clocks are applied. PROflow directs PROsim to perform these steps automatically when the Execute Power On Reset option is set in the Functional Simulation dialog box.

## Adding Signals and Vectors to the Waveform

Now that you have loaded the simulation netlist into PROsim, you can specify the signals to be displayed in PROwave and define the design's initial inputs.

Following is the list of signals to add to the display list contained in the calc.wfm file:

CLK (FPGAs only)	The system clock
OSC_7K\XCLK (EPLDs only)	The system clock
EXC_P	The execute signal
SW7\SW[6:0]_P	The opcode switches
ALU[3:0]	The ALU flip-flop outputs
STACK[3:0] (FPGAs only)	The top of the stack
LED\LED[3:0]_P (EPLDs only)	The top of the stack

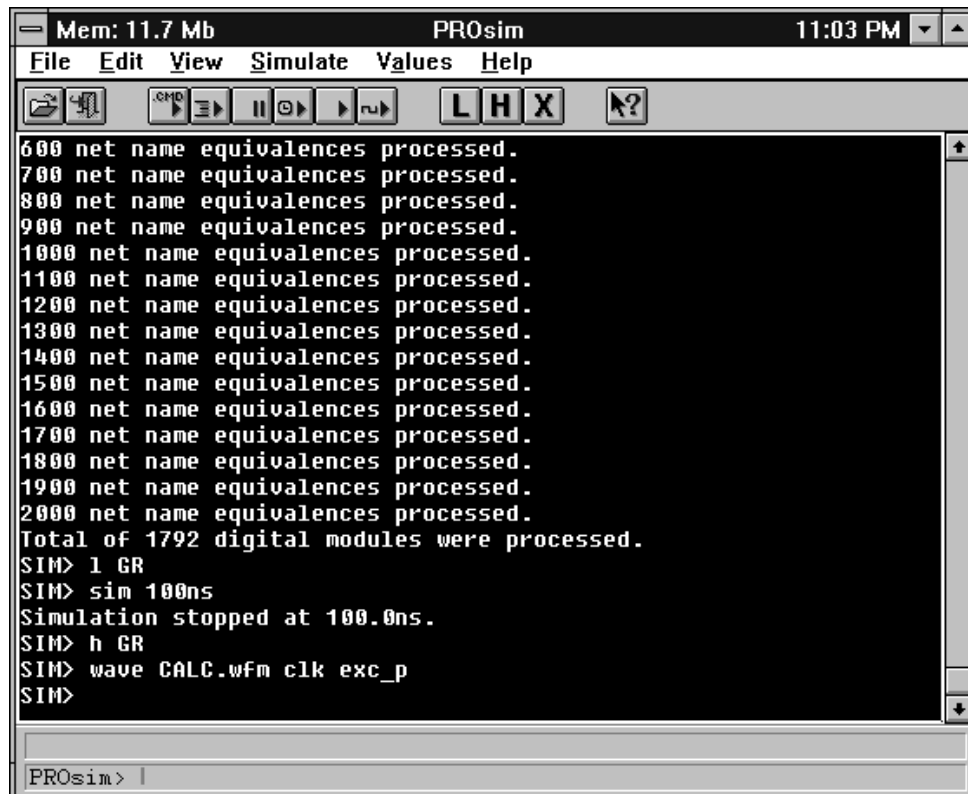
**Note:** Because the XEPLD optimization software eliminates many internal signals, the XC7000 version of this simulation uses the LED\_P vector, or bus, instead of the STACK vector.

### Adding Signals to the Waveform

To add the CLK or OSC\_7K\XCLK and EXC\_P signals to the waveform, follow these steps.

1. At the PROsim> prompt, type **s\_wave**.  
PROsim prompts for the PROwave data stream file name, giving the default name of calc.wfm.
2. Press **↵** to accept the default name.
3. At the Node(s)/Vector(s)> prompt, type **clk exc\_p** for FPGAs or **osc\_7k\xclk exc\_p** for EPLDs.

Once you have entered the signals to be added, PROsim displays the issued command, `wave CALC.wfm clk exc_p` for FPGAs, as shown in Figure 1-125, or `wave CALC.wfm osc_7k\xclk exc_p` for EPLDs.



The screenshot shows the PROsim software interface. The title bar indicates 'Mem: 11.7 Mb', 'PROsim', and '11:03 PM'. The menu bar includes 'File', 'Edit', 'View', 'Simulate', 'Values', and 'Help'. The toolbar contains various simulation control icons and the letters 'LHX'. The main window displays the following text:

```
600 net name equivalences processed.
700 net name equivalences processed.
800 net name equivalences processed.
900 net name equivalences processed.
1000 net name equivalences processed.
1100 net name equivalences processed.
1200 net name equivalences processed.
1300 net name equivalences processed.
1400 net name equivalences processed.
1500 net name equivalences processed.
1600 net name equivalences processed.
1700 net name equivalences processed.
1800 net name equivalences processed.
1900 net name equivalences processed.
2000 net name equivalences processed.
Total of 1792 digital modules were processed.
SIM> l GR
SIM> sim 100ns
Simulation stopped at 100.0ns.
SIM> h GR
SIM> wave CALC.wfm clk exc_p
SIM>
```

The status bar at the bottom shows 'PROsim> |'.

Figure 1-125 Adding the CLK and EXC\_P Signals to the Waveform

### Adding Vectors to the Waveform

The remaining signals are best defined as buses in the waveform display. To do so, you can create vectors representing the signals of the buses to be grouped together in the waveform display.

Follow this procedure to create vectors for SW7\SW[6:0]\_P, ALU[3:0], and STACK[3:0] (for FPGAs) or LED\LED[3:0]\_P (for EPLDs).

1. At the PROsim> prompt, type `vector SW SW7\SW[6:0]_p`.  
PROsim displays the command in the command log display window.
2. Repeat step 1 to create the ALU and STACK vectors, as shown in Figure 1-126.  
Instead of creating a STACK vector for EPLDs, create the LED\_P vector by typing `vector LED_P led\led[3:0]_p`.

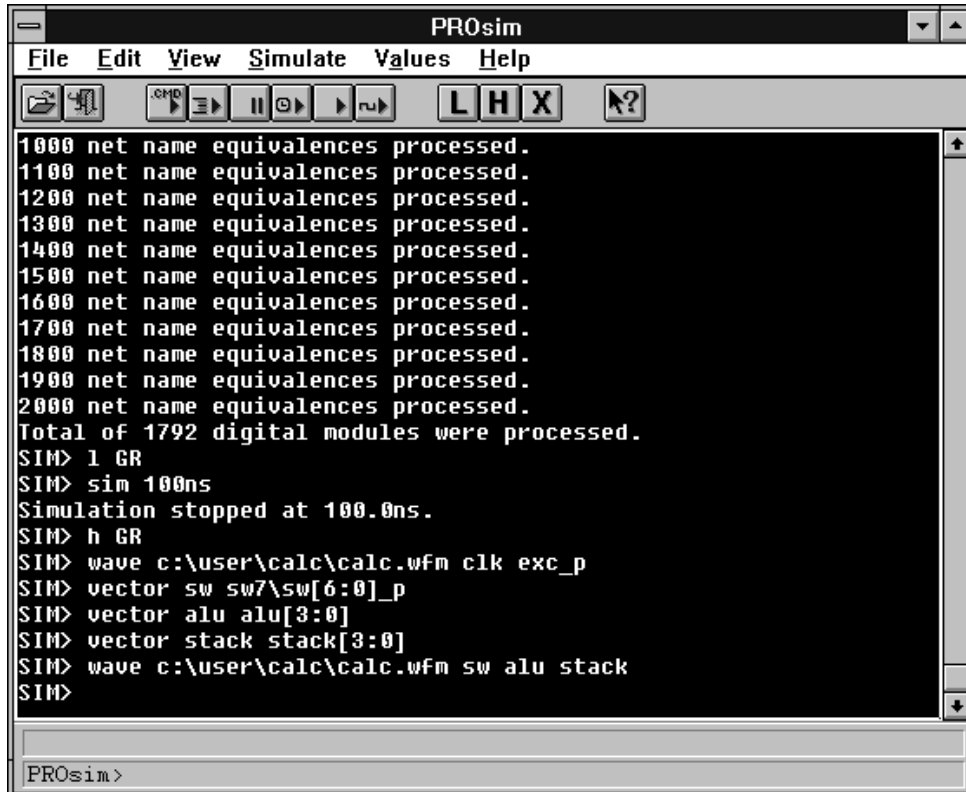


Figure 1-126 Adding the SW, ALU, and STACK Vectors to the Waveform



3. To add SW, ALU, and STACK to the FPGA waveform, type `s_wave calc.wfm SW ALU STACK`. To add SW, ALU, and LED\_P to the EPLD waveform, type `s_wave calc.wfm SW ALU LED_P`.

## Defining the Design Inputs

After you define the signals to be viewed in PROwave, you must set up the design inputs and give them values to simulate. You can define two types of stimuli: clocks and design inputs. Most clocks can be set up using the periodic waveform command `clock`.

### Defining a Clock

To define the clock as a periodic waveform, follow these steps.

1. At the PROsim> prompt, type `clock clk 0 1` for FPGAs or `clock osc_7k\clock 0 1` for EPLDs.

This step tells the simulator to give the clock net, CLK or XCLK, a 50% duty cycle with a starting value of 0.

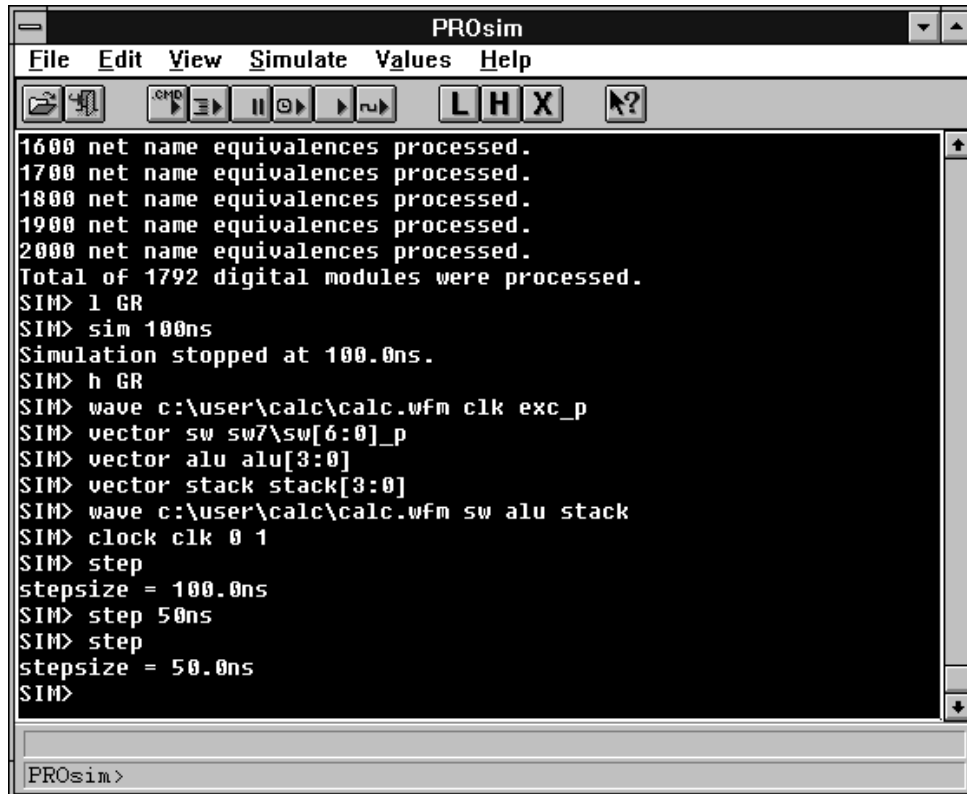
2. At the PROsim> prompt, type `step`.

PROsim returns the default step size value of 100 ns. The step size is the length in nanoseconds of one step of the clock definition. Therefore, the period of the CLK is 200 ns. For example, if you defined a new clock by typing the following, it would have a 50% duty cycle with a period of 400 ns:

```
clock newclk 0 0 1 1
```

3. To change the step size, type `step 50ns`.
4. To verify that the change has been made, type `step`.

This sequence of steps is reflected in Figure 1-127.



```
PROsim
File Edit View Simulate Values Help
[Icons: .CMD, Run, Stop, Step, Step Back, Step Forward, L, H, X, Help]
1600 net name equivalences processed.
1700 net name equivalences processed.
1800 net name equivalences processed.
1900 net name equivalences processed.
2000 net name equivalences processed.
Total of 1792 digital modules were processed.
SIM> 1 GR
SIM> sim 100ns
Simulation stopped at 100.0ns.
SIM> h GR
SIM> wave c:\user\calc\calc.wfm clk exc_p
SIM> vector sw sw7\sw[6:0]_p
SIM> vector alu alu[3:0]
SIM> vector stack stack[3:0]
SIM> wave c:\user\calc\calc.wfm sw alu stack
SIM> clock clk 0 1
SIM> step
stepsize = 100.0ns
SIM> step 50ns
SIM> step
stepsize = 50.0ns
SIM>
```

Figure 1-127 Defining the Clock Step Size

### Defining Input Values

Now that CLK or XCLK has been defined as a periodic waveform, you can set the EXC\_P and SW signals to their initial values.

1. To set the EXC\_P signal initially High, type **h EXC\_P** at the PROsim> prompt.
2. To set the opcode vector SW initially to the NOP instruction, type **assign SW 1111111** at the PROsim> prompt.

You can use the Assign command to force values to each signal of a bus (vector).

3. Re-display PROcapture by making an icon of PROsim and double-clicking on the PROcapture icon.

Now that PROsim is running, values are annotated to the CALC.1 schematic, as shown in Figure 1-128. All the flip-flop outputs and signals derived exclusively from flip-flop outputs have known values because the global reset line was asserted and simulated. The X values appear because the assignments of the user inputs and the clock have only been defined but not yet simulated.

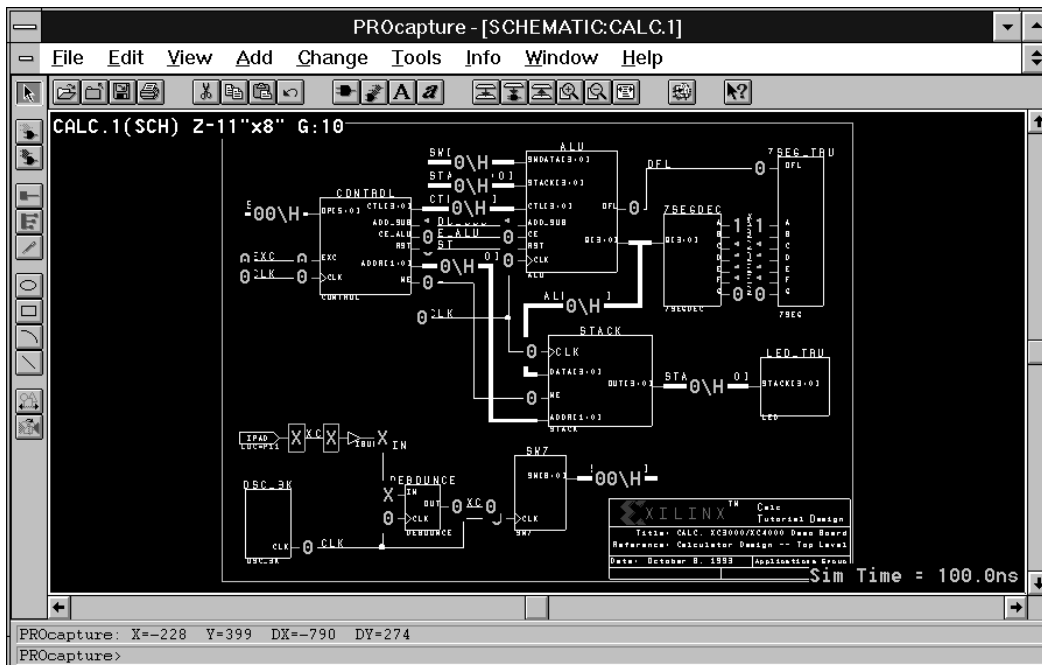


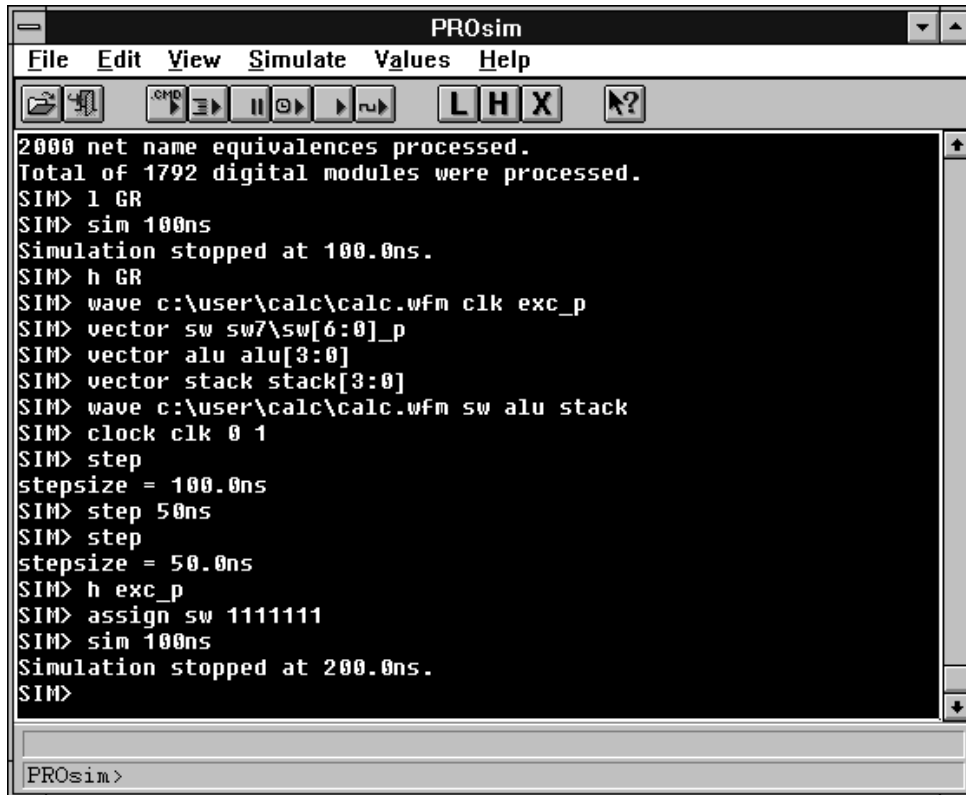
Figure 1-128 CALC.1 Schematic Annotated with Startup Values

### Simulating the Design Inputs

The next step is to simulate the defined user inputs.

1. Return to PROsim and type `sim 100ns`.

PROsim responds with the time at which the simulation stopped, as Figure 1-129 indicates.



**Figure 1-129 Initial Values Simulated**

2. Activate PROcapture by making an icon of PROsim and double-clicking on the PROcapture icon.

Now all the signals in the design display known values, as shown in Figure 1-130. The values displayed in boxes are those forced by the assignment commands entered in PROsim.

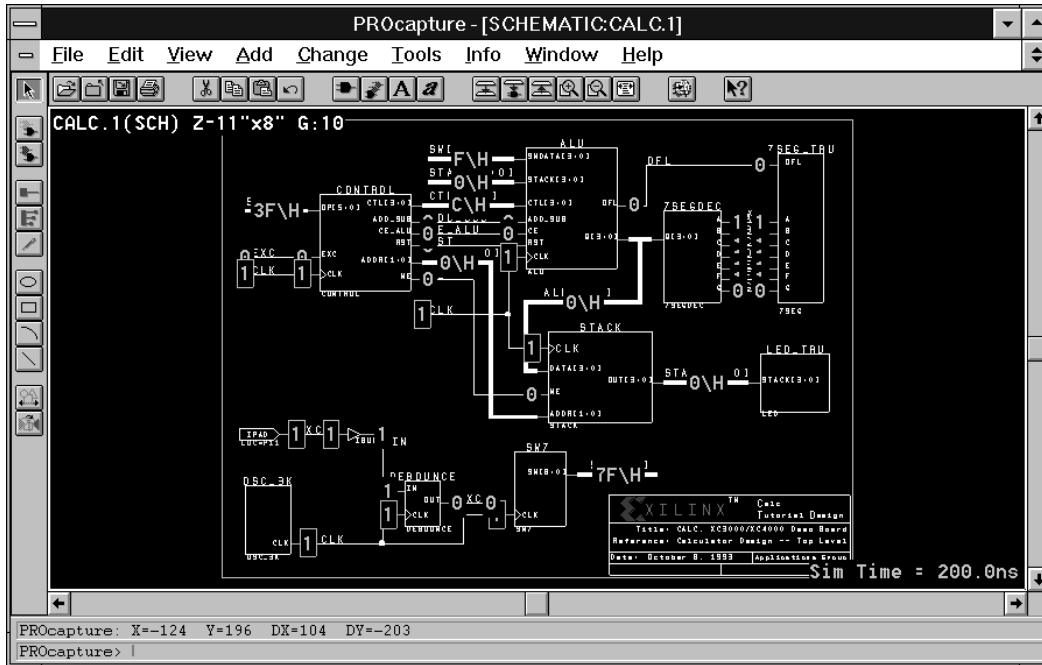


Figure 1-130 CALC.1 Schematic Annotated with Initial Values

## Invoking PROwave

You can view any of the known nets in the design while simulating by simply looking at the design schematics. Viewing the schematics can be very helpful when simulating, but another way to view a large number of signals is to open a waveform window in PROwave.

1. Activate PROflow by making an icon of PROcapture and selecting PROflow with the left mouse button.
2. Click on the Functional Simulation PROwave icon, shown in Figure 1-131.



Figure 1-131 PROwave Icon

Selecting the PROwave icon displays PROwave and brings up the Open dialog box, shown in Figure 1-132.

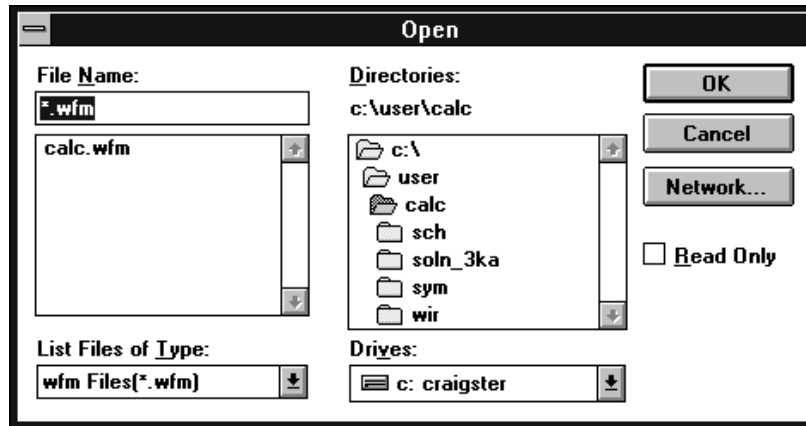


Figure 1-132 Open Dialog Box

3. In the File Name list box, select the `calc.wfm` waveform display file.
4. Click on `OK`.

The Open dialog box closes, and the `calc.wfm` waveform display file opens, as pictured in Figure 1-133.

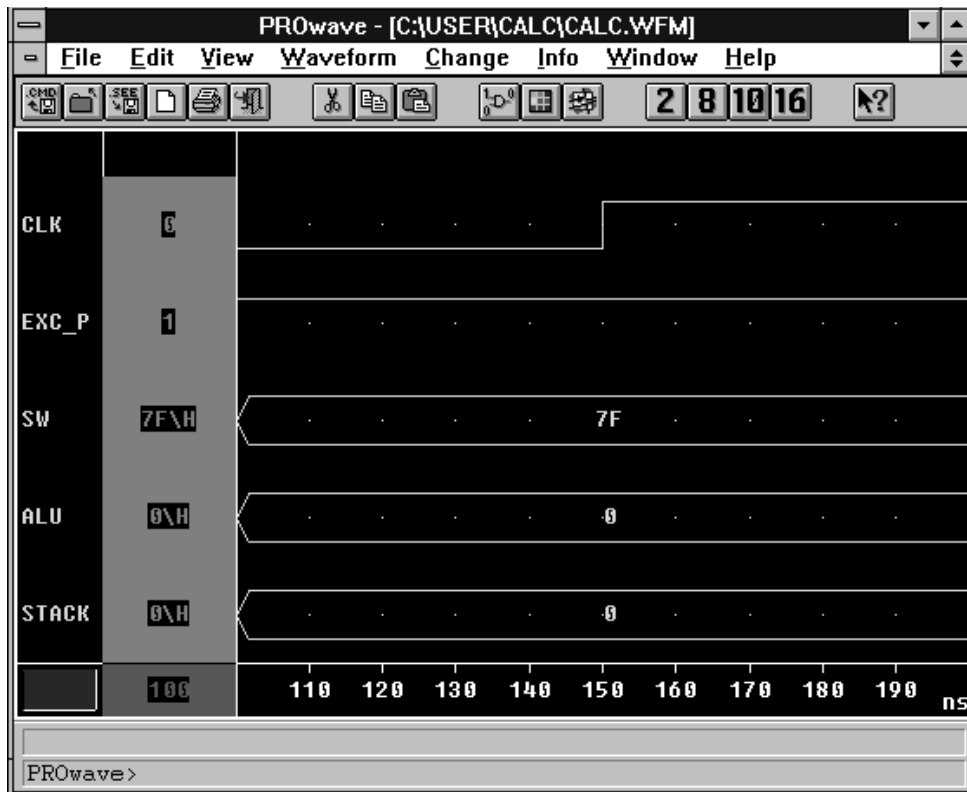


Figure 1-133 Calc.wfm Waveform Display File and Initial Values

### Changing the Display Radix

The default display radix for the grouped signals is hexadecimal. You may want to change the display radix to an alternative base.

Follow this procedure to change the display radix of the ALU and SW groups.

1. Point the mouse at the ALU group text in the list of signals on the left side of the waveform window and select it by clicking the left mouse button.
2. Select the **Waveform** → **Set Radix** → **Decimal Radix** command to change the ALU vector display to decimal.
3. Deselect the ALU group by clicking the left mouse button on the ALU group text.
4. Select the SW group by clicking the left mouse button on the SW group text.
5. Select the binary radix toolbar icon, shown in Figure 1-134, to set the SW group's display radix to binary.



**Figure 1-134 Binary Radix Toolbar Icon**

In Figure 1-135, the SW group's display radix is set to binary, the ALU group's display radix is set to decimal, and the STACK group's display radix is set to hexadecimal.



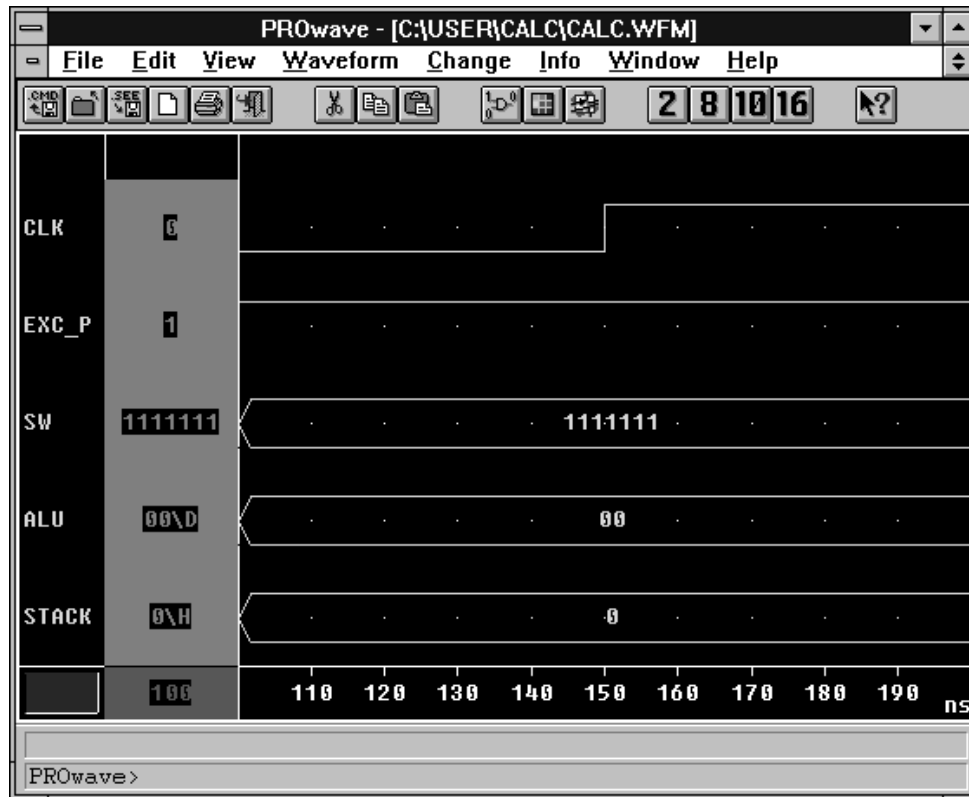


Figure 1-135 Binary, Decimal, and Hexadecimal Radices

## Simulating the Calc Design

With the desired signals added to the waveform display and set to the appropriate radix, you can now simulate the Calc design to verify its functionality. To this end, you will use PROsim to simulate two functions, loading the ALU register and pushing the STACK register.

The Calc design consists of a 4-bit processor with a stack. Inputs are a 7-bit bus, SW7\SW[6:0]\_P, which defines the opcode and data, and an Execute switch, EXC\_P. Whenever EXC\_P toggles High to Low and then Low to 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 of the stack.

Table 1-2 shows the valid opcodes and their functions.

**Table 1-2 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				Subtract 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	Subtract 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

### Loading 1111 to the ALU Register

Follow these instructions to load 1111 to the ALU register.

1. Activate PROsim by making an icon of PROwave and double-clicking on the PROsim icon.
2. At the PROsim> prompt, type `assign sw 1101111↓`.

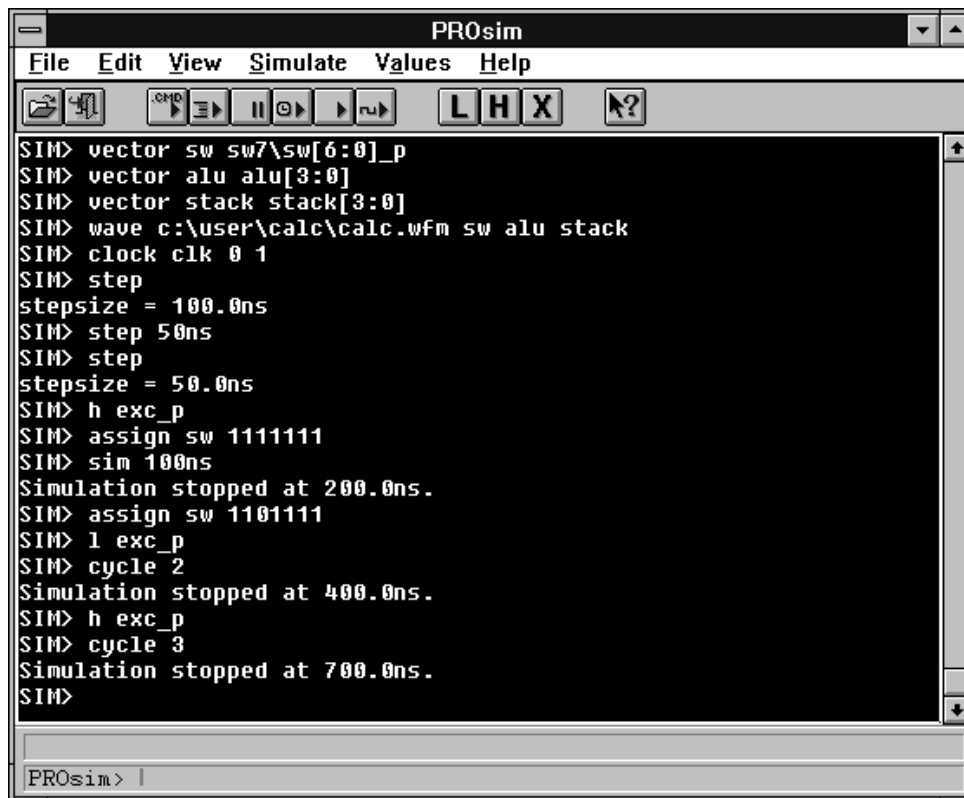
This step sets the switches to perform the load register operation 100 with the value 1111.

3. To bring EXC\_P Low, type `l exc_p`.
4. To simulate for two clock cycles, type `cycle 2`.

The assignments made in steps 2 and 3 are now applied to the simulation network, which is simulated for a total time of two clock cycles, or 200 ns. This simulation loads the opcode into Calc's internal logic.

5. To bring EXC\_P High, type `h exc_p`.
6. To simulate for three clock cycles, type `cycle 3`.

Figure 1-136 reflects the commands that you have just entered.



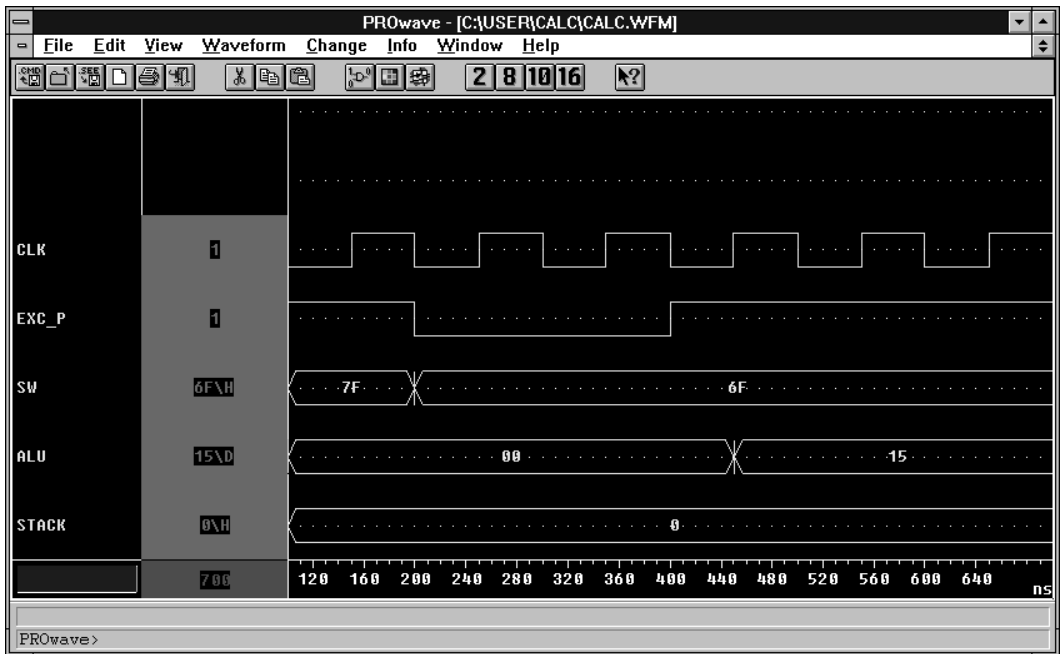
```
PROsim
File Edit View Simulate Values Help
[Icons] [CMD] [Run] [Pause] [Stop] [Next] [L] [H] [X] [Help]
SIM> vector sw sw7\sw[6:0]_p
SIM> vector alu alu[3:0]
SIM> vector stack stack[3:0]
SIM> wave c:\user\calc\calc.wfm sw alu stack
SIM> clock clk 0 1
SIM> step
stepsize = 100.0ns
SIM> step 50ns
SIM> step
stepsize = 50.0ns
SIM> h exc_p
SIM> assign sw 111111
SIM> sim 100ns
Simulation stopped at 200.0ns.
SIM> assign sw 110111
SIM> l exc_p
SIM> cycle 2
Simulation stopped at 400.0ns.
SIM> h exc_p
SIM> cycle 3
Simulation stopped at 700.0ns.
SIM>
```

Figure 1-136 Loading the ALU Register

7. Activate PROwave by making an icon of PROsim and double-clicking on the PROwave icon.

Notice that the ALU group's radix has reverted back to hexadecimal.

8. Re-set the ALU radix to decimal, as shown in Figure 1-137.



**Figure 1-137 Calc.wfm Waveform Display and Simulated Load Operation**

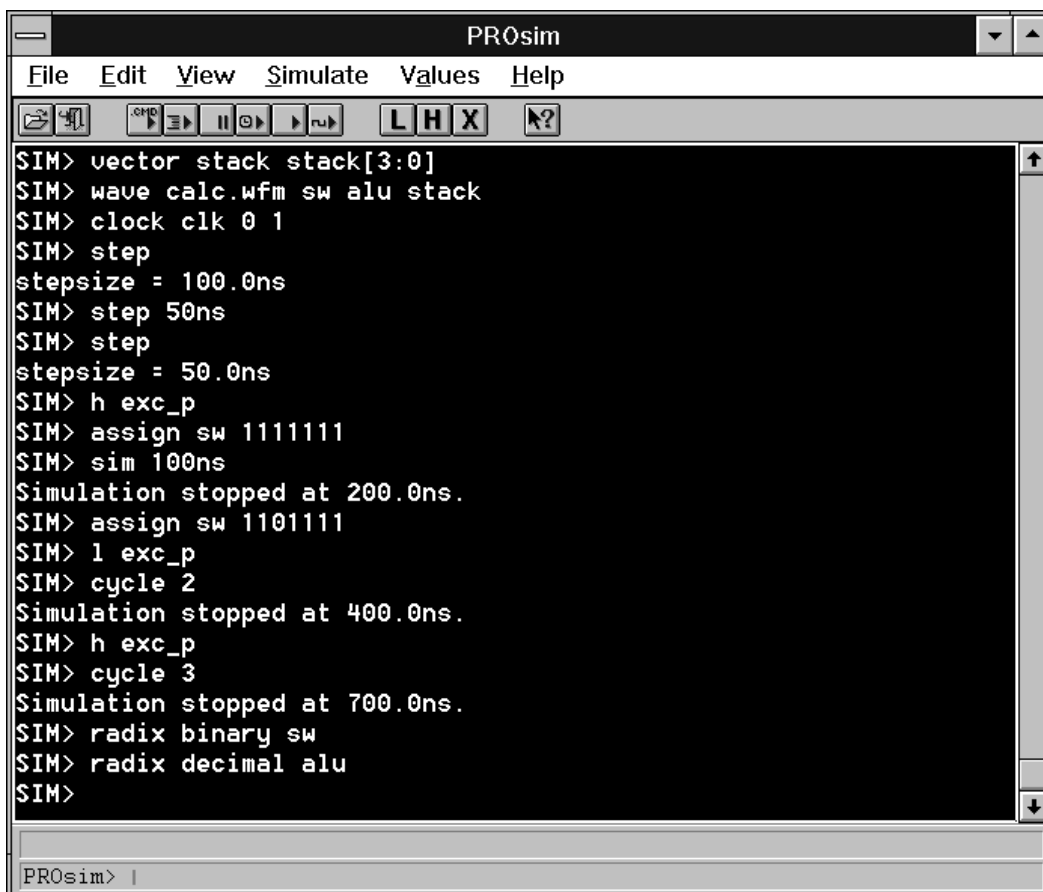
### Changing the Radices for PROcapture Display

You can also change the radix of the SW[6:0] bus and the ALU vector displayed in PROcapture by setting the radix in PROsim so that it is written to the calc.wfm waveform display file.

1. Activate PROsim by making an icon of the PROwave window and double-clicking on the PROsim icon.

2. To set the radix of the SW bus in the calc.wfm file to binary, type **radix binary SW** at the PROsim> prompt.
3. To set the radix of the ALU group in the calc.wfm file to decimal, type **radix decimal ALU** at the PROsim> prompt.

These two commands should be reflected on your screen, as in Figure 1-138.



```
PROsim
File Edit View Simulate Values Help
[Icons] [L] [H] [X] [?]
SIM> vector stack stack[3:0]
SIM> wave calc.wfm sw alu stack
SIM> clock clk 0 1
SIM> step
stepsize = 100.0ns
SIM> step 50ns
SIM> step
stepsize = 50.0ns
SIM> h exc_p
SIM> assign sw 1111111
SIM> sim 100ns
Simulation stopped at 200.0ns.
SIM> assign sw 1101111
SIM> l exc_p
SIM> cycle 2
Simulation stopped at 400.0ns.
SIM> h exc_p
SIM> cycle 3
Simulation stopped at 700.0ns.
SIM> radix binary sw
SIM> radix decimal alu
SIM>
```

Figure 1-138 Changing SW and ALU Radices

4. Activate PROcapture by making an icon of the PROsim window and double-clicking on the PROcapture icon.

As Figure 1-139 indicates, the display radices in PROcapture have been changed for the SW[6:0] bus and the ALU vector.

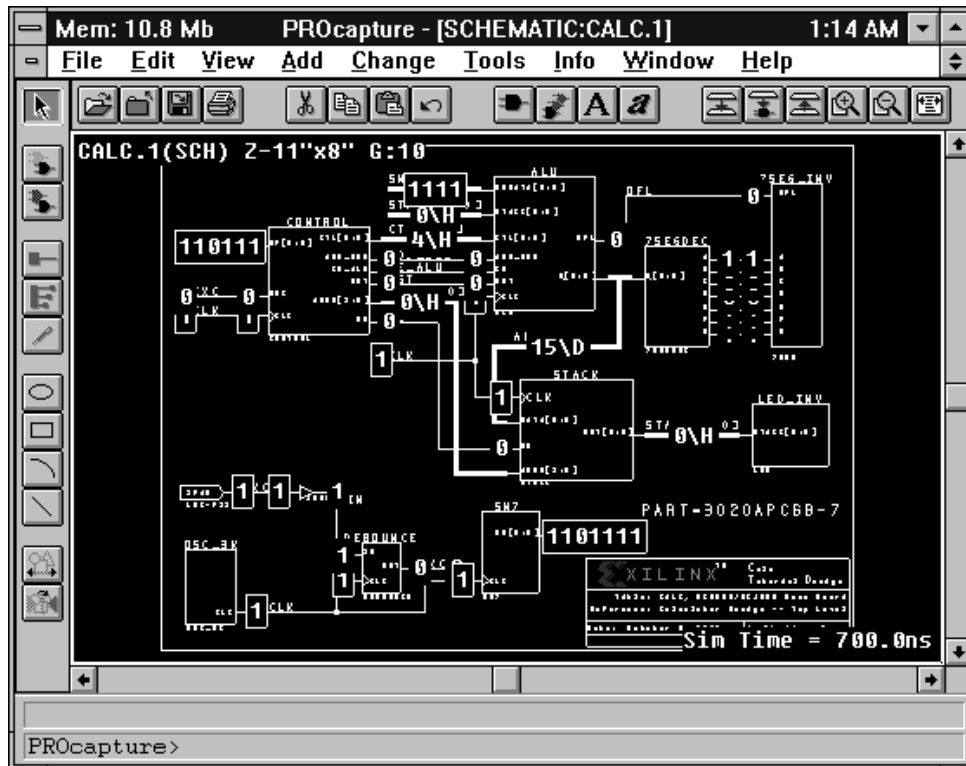


Figure 1-139 Changed SW and ALU Display Radices in PROcapture

### Pushing 1111 to the STACK Register

To load the ALU value 1111 to the STACK register, follow these steps.

1. Activate PROsim by making an icon of the PROcapture window and double-clicking on the PROsim icon.

2. At the PROsim> prompt, type **assign sw 1111011**↵.

This step sets the switches to perform the push register operation 111101. (According to Table 1-2, the last digit is a don't-care.)

3. To bring EXC\_P Low, type **l exc\_p**↵.
4. To simulate for two clock cycles, type **cycle 2**↵.

The assignments made in steps 2 and 3 are now applied to the simulation network, which is simulated for the total time of two clock cycles, or 200 ns. This simulation loads the opcode into Calc's internal logic.

5. To bring EXC\_P High, type **h exc\_p**↵.
6. To simulate for four clock cycles, type **cycle 4**↵.

**Note:** A push operation takes an extra state in the state machine, so it needs an extra clock cycle to execute.

Figure 1-140 reflects the commands that you have just entered.

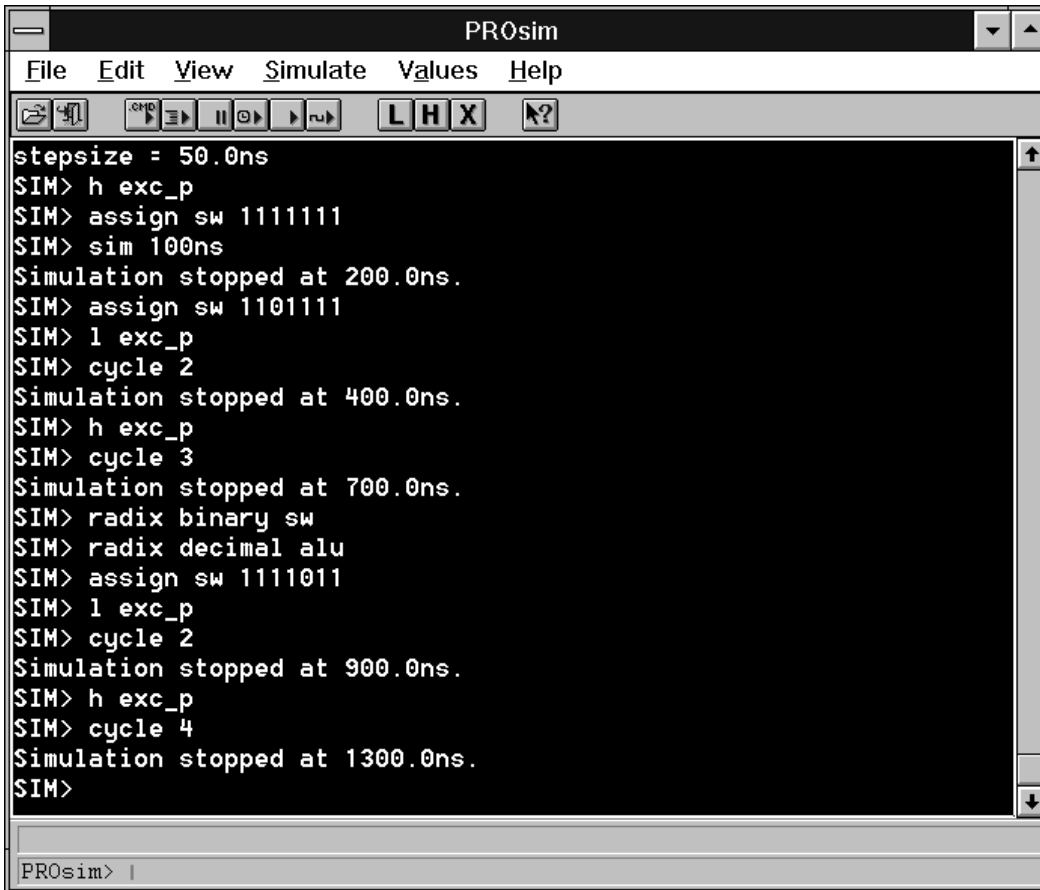


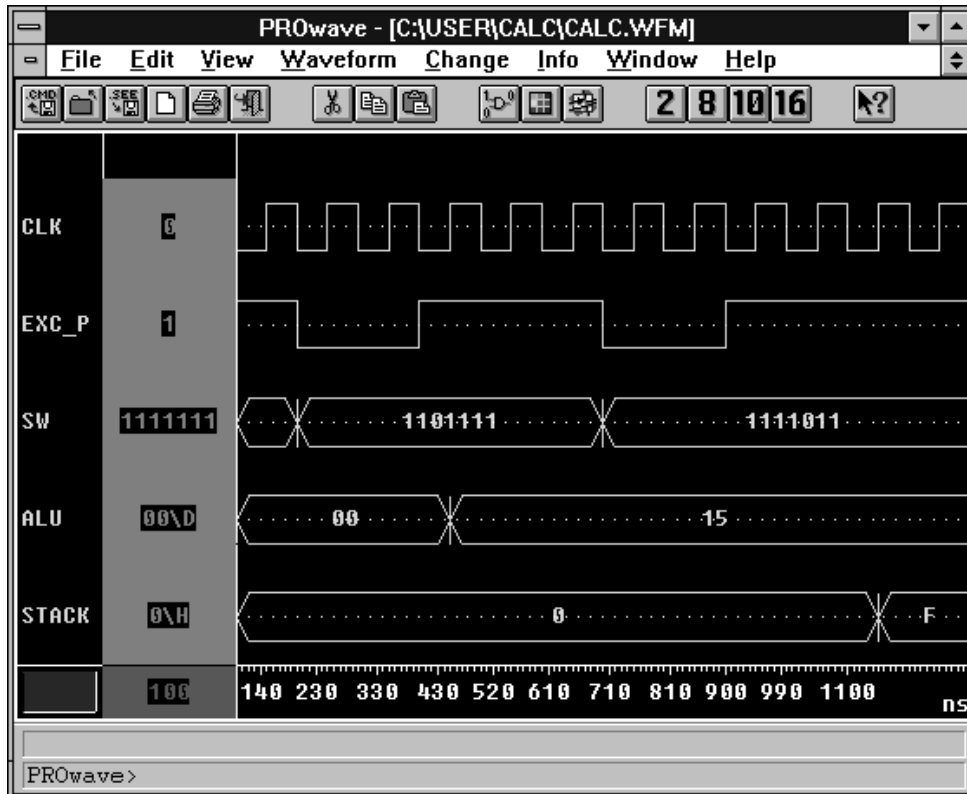
Figure 1-140 Pushing the Register Value to the STACK

### Viewing the Waveforms

You have simulated two processor commands, which is enough to show some interesting waveforms.

Activate PROwave by making an icon of the PROsim window and double-clicking on the PROwave icon, changing the radices if you wish. Figure 1-141 displays the waveforms resulting from the commands executed so far.





**Figure 1-141 Calc.wfm Waveform Display and the Simulated Load and Push Operations**

You can further verify the Calc design by using the other processor opcodes and viewing the values in both the waveform viewer and the schematic. If you wish to experiment with the other operations, use the File → Save command to save the current waveforms to the calc.see file. The calc.see file will be used later in the tutorial to perform timing simulation comparisons.

## Re-Creating Previous Simulation

During simulation, PROsim creates a log file that contains a history of every command issued in that session, as well as the PROsim status of the commands. You can edit this log file and save it to a command file that you can then use to re-create the previous simulation.

1. Once you have finished simulating, close PROsim.
2. Invoke Notepad and open the file in c:\user\calc\viewsim.log.
3. Save the file to a *filename.cmd* file.

## Implementing the Calc Design

With the Calc design functionally verified, it is time to implement the design. Clicking on the Xilinx Implementation icon opens the Xilinx Design Manager, which implements Calc. It creates the files needed to simulate timing and to download.

## Invoking the Design Manager

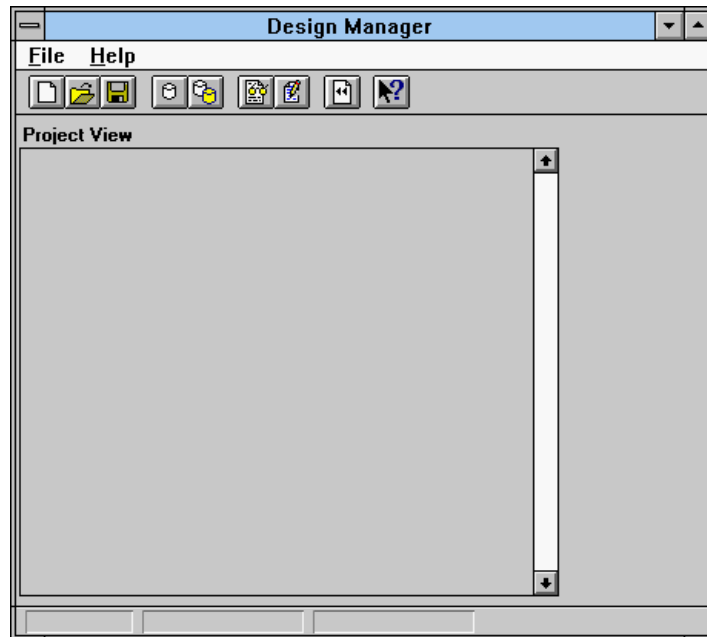
You can invoke the Design Manager from PROflow or from the Program Manager XACTstep program group.

1. Click on the Xilinx Implementation icon in PROflow, shown in Figure 1-142.



**Figure 1-142 Xilinx Implementation Icon**

The Design Manager window appears, as shown in Figure 1-143.



**Figure 1-143 Design Manager Window**

On the top of the window is the title bar, which identifies the tool as the Design Manager. Beneath the title bar is the menu bar on which the pull-down menus appear. The full set of available menus does not appear until you load an implementation project into the Design Manager. The toolbar, which is located below the menu bar, displays icons that perform the same functions as the most commonly used menu commands. The Project View section of the window contains a graphical representation of the versions and revisions of the design. Finally, the status bar at the bottom of the window displays the family, part number, version number, and revision number of your design when a project is loaded.

## Creating the Calc Implementation Project and Its Initial Translation

Before you can begin implementing the design, you must create the Xilinx project, which is a different process from creating the Viewlogic project. The Xilinx project directories contain version and revision information for multiple runs of your design through the Xilinx implementation tools.

Follow these steps to create the Calc implementation project.

1. Select the **File** → **New Project** command.

The New Project dialog box appears, as shown in Figure 1-144.

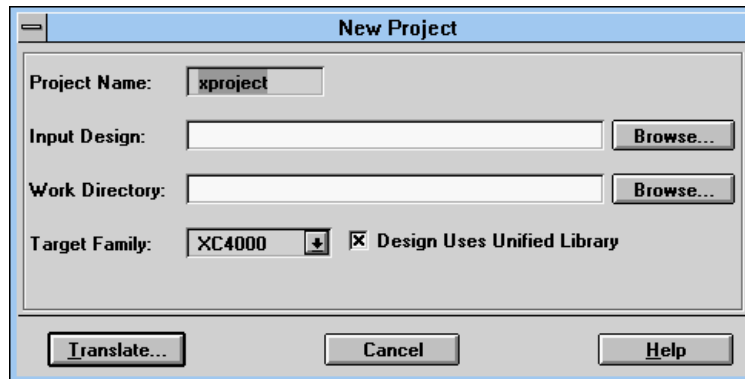


Figure 1-144 New Project Dialog Box

2. To specify the CALC.1 top-level schematic file as the input design, click on **Browse**.

The Open dialog box appears, as shown in Figure 1-145.

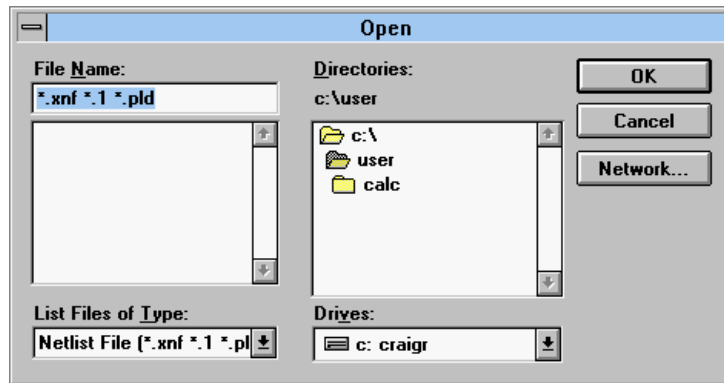


Figure 1-145 Open Dialog Box

3. Using the Directories list box, select the user\calc\sch directory, and select the CALC.1 schematic file, as Figure 1-146 demonstrates.

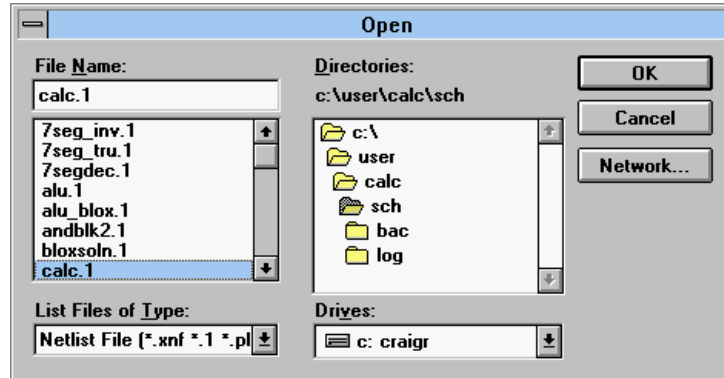
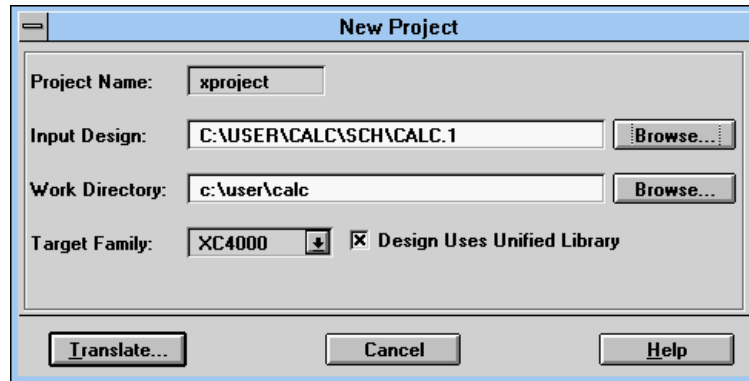


Figure 1-146 Schematic File Selected in Open Dialog Box

4. Click on OK.

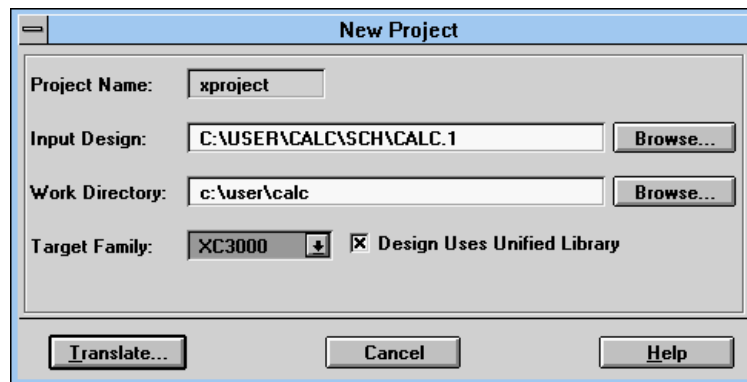
The Open dialog box now closes, and the New Project dialog box is updated with the selected file, as illustrated in Figure 1-147.



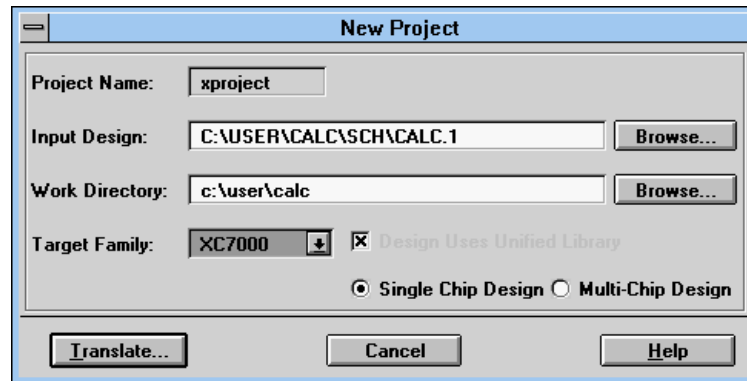
**Figure 1-147 Updated New Project Dialog Box**

The Work Directory field now displays the default value of c:\user\calc. It reflects the name of the directory in which the Design Manager places the project directory. The project directory, by default xproject, is where the version and revision data is stored.

5. In the Target Family field, select the family in which the device will be implemented, either XC3000, as shown in Figure 1-148, or XC7000, as shown in Figure 1-149.



**Figure 1-148 Selecting the XC3000 Family**



**Figure 1-149 Selecting the XC7000 Family**

You can partition a multi-chip XC7000 design with the Design Manager. Because the Calc design is targeted for the 73108-7PC84 part, select the **Single Chip Design** option. For more information on partitioning EPLD designs, see the *Design Manager/Flow Engine Reference/User Guide*.

6. Click on **Translate**.

The Translate Options dialog box appears, as shown in Figure 1-150.



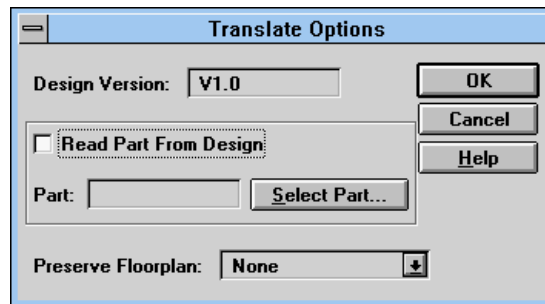
**Figure 1-150 Translate Options Dialog Box**

In this dialog box, you can name the design version being translated and specify the design part type. The Design Version

field displays the version number of the design. For the first translation, V1.0 appears by default. With subsequent translations of the same design, the version number automatically increments. For information on the Preserve Floorplan option, see the *Design Manager/Flow Engine Reference/User Guide*.

7. To specify the Calc part type, deselect the **Read Part From Design** check box.

The Select Part button and the Part field are now activated. You can specify the part type by typing it in the Part field or by clicking on the Select Part button.



**Figure 1-151 Activated Part Fields on Translate Options Dialog Box**

8. Click on **select Part**.

This button displays the Part Selector dialog box. This dialog box resembles the illustration in Figure 1-152 for the XC3000 family, or the one in Figure 1-153 for the XC7000 family.



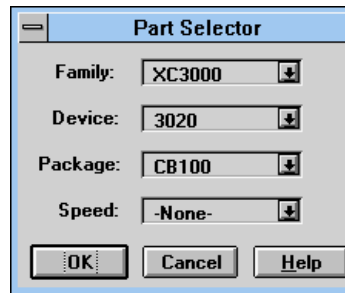


Figure 1-152 Part Selector Dialog Box for XC3000 Family

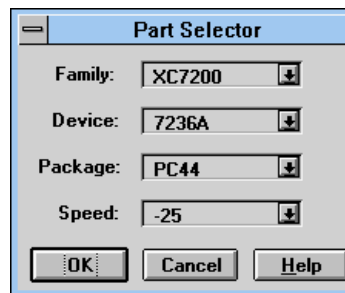


Figure 1-153 Part Selector Dialog Box for XC7000 Family

In the Part Selector dialog box, the Device field reflects only the parts for the family selected in the Family field. Similarly, the Package field displays only those packages suitable for the device selected. The Speed field displays only those speed grades appropriate for the part and package selected.

- Using the Part Selector pull-down list boxes, select the 3020APC68-7 part for the XC3000A family, as shown in Figure 1-154, or the 73108-7PC84 part for the XC7000 family, as shown in Figure 1-155.

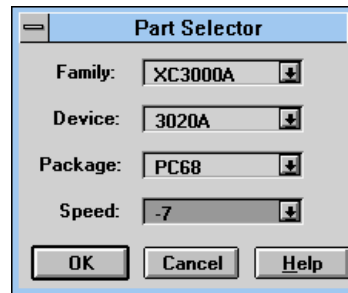


Figure 1-154 Selecting an XC3000A Part

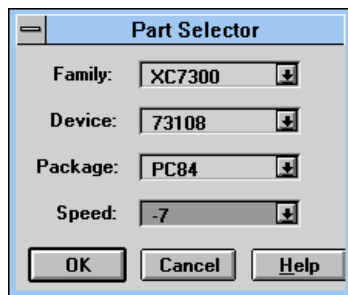


Figure 1-155 Selecting an XC7300 Part

10. Click on **OK** to accept the Part Selector dialog box.

The Translate Options dialog box is now updated with the selected parts. Figure 1-156 and Figure 1-157 display this dialog box for the XC3000A and the XC7000 families, respectively.



Figure 1-156 Translate Options Dialog Box with XC3000A Part

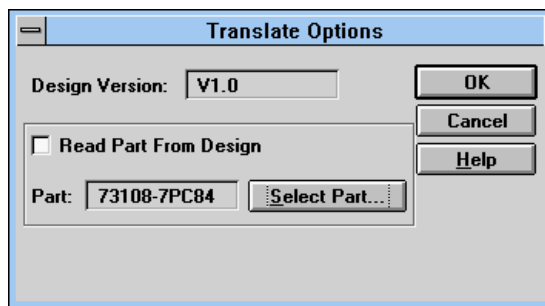
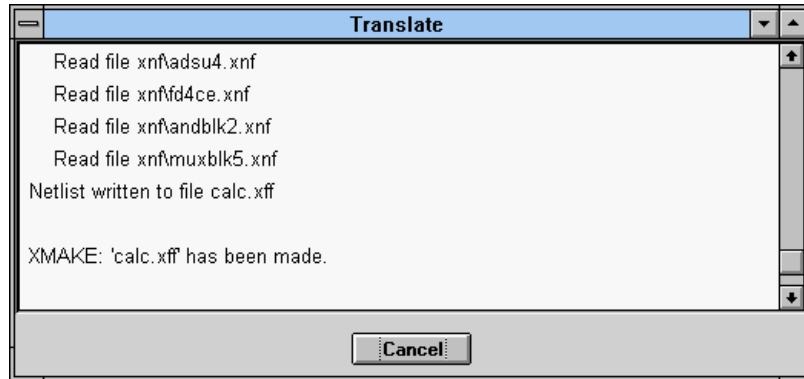


Figure 1-157 Translate Options Dialog Box with XC7000 Part

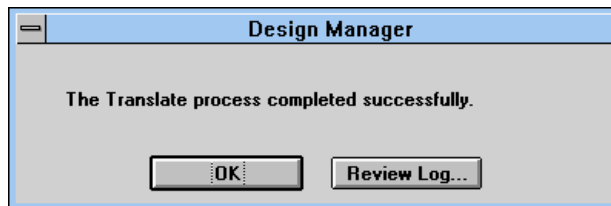
11. To translate the design, click on **OK** in the Translate Options dialog box.

The Translate status window opens, as shown in Figure 1-158. This window displays the processing of the design so that you can monitor the progress of the translation.



**Figure 1-158 Translate Status Window**

The Design Manager uses XMake for FPGAs and XEMake for EPLDs to translate the input design to a flattened file. These programs first translate the input design files to XNF files, then merge the XNF files into one flattened XNF file called *design.xff*. When they successfully complete the translation, the Design Manager displays the message box shown in Figure 1-159.



**Figure 1-159 Design Manager Translation Message Box**

This message box allows you to review the results of the translation, which are placed in a log file. Clicking on Review Log displays the log file in a text editor.

12. Click on **OK**.

The message box closes, and the Design Manager is updated with the new Calc implementation project. Figure 1-160 shows how the Design Manager is updated for an FPGA design, and Figure 1-161 shows how it is updated for an EPLD design.

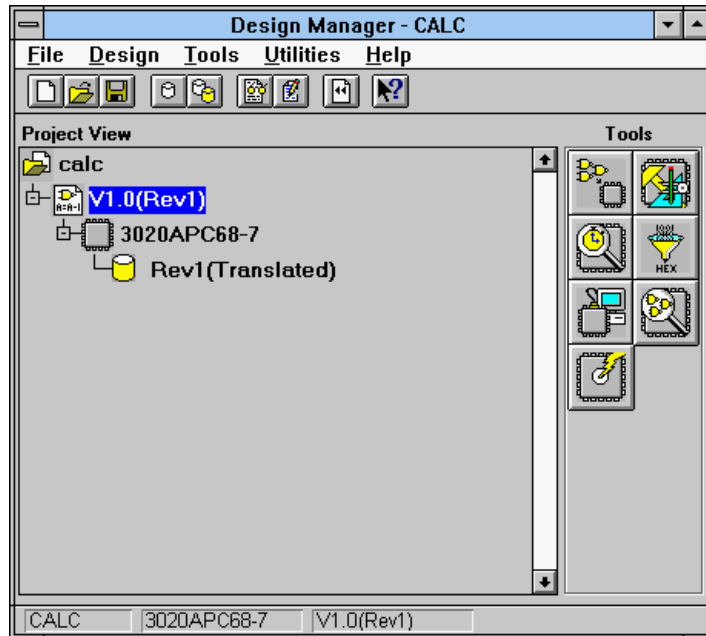
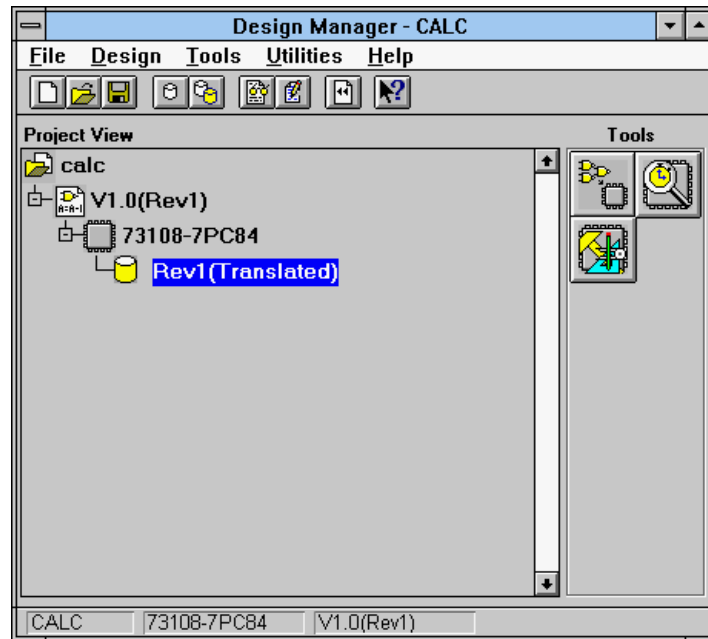


Figure 1-160 FPGA Design in Updated Design Manager Window



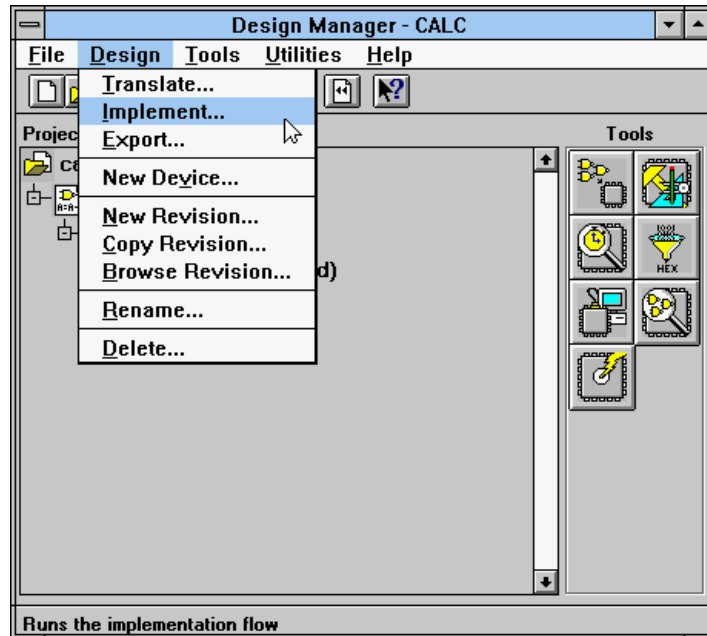
**Figure 1-161 EPLD Design in Updated Design Manager Window**

In the Project View section of the window, the new project contains a single version with an implementation revision. The status of the revision is “Translated.” The Tools section of the window contains the icons of the Xilinx tools available for the new project. The menu bar is also updated to show all the available menus. If you are performing the FPGA tutorial, proceed with the next section. If you are performing the EPLD tutorial, skip to the “Implementing the EPLD Design” section later in this chapter.

## Implementing the FPGA Design

The design is now translated and ready to be implemented. The Design Manager uses the Flow Engine to implement a design. For FPGA designs, the Flow Engine creates a configuration bitstream, *design.bit*, as well as timing simulation data. It automatically exports this data to the Viewlogic project directory for use by PROflow.

1. To set the implementation options and invoke the Flow Engine, select **Design** → **Implement** in the Design Manager, as shown in Figure 1-162.



**Figure 1-162 Selecting the Implement Command**

The Design Implementation dialog box opens, as indicated in Figure 1-163.

In this dialog box, you can select control files, create option templates, and specify optional targets. In the Constraints File field, you can select any constraints file. The Guide Design field displays only the revisions containing guide data. For more information on control files, see the *Design Manager/Flow Engine Reference/User Guide*.

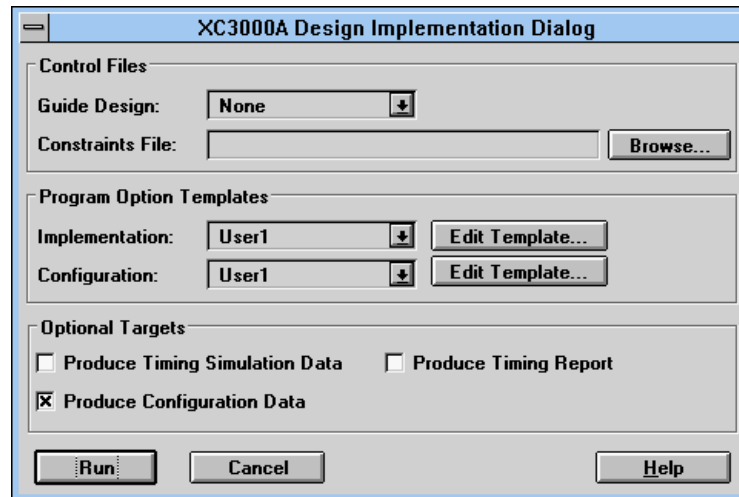


Figure 1-163 Design Implementation Dialog Box

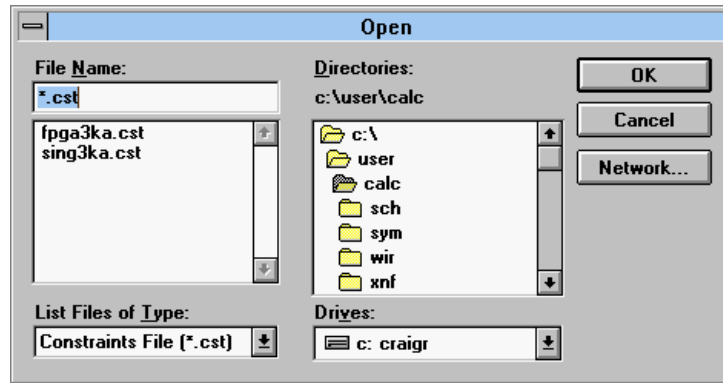
### Setting General FPGA Options

The general FPGA options are all the options on the Design Implementation dialog box except for those in the Program Option Templates field.

1. To select the constraints file, click on **Browse**.

The Open dialog box displays, as shown in Figure 1-164.





**Figure 1-164 Open Dialog Box**

The Open dialog box is initially displayed with the \*.cst file filter, so only constraints files are listed in the File Name list box.

2. Select the constraints file appropriate for the demonstration board that you will be using to test the implemented design, as noted in the example in Figure 1-165.

**Note:** Use the fpga3ka.cst constraints file if you are targeting the dual-device demonstration board. You can use the sing3ka.cst constraints file if you are targeting the single-device demonstration board.

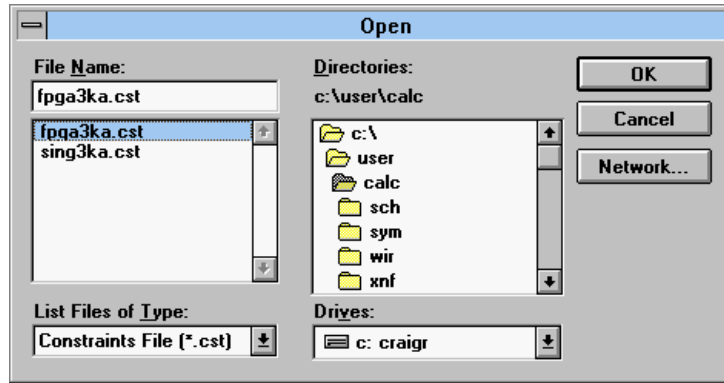


Figure 1-165 Selecting the Constraints File

3. Click on **OK**.

As shown in Figure 1-166, the Design Implementation dialog box is now updated with the selected constraints file.

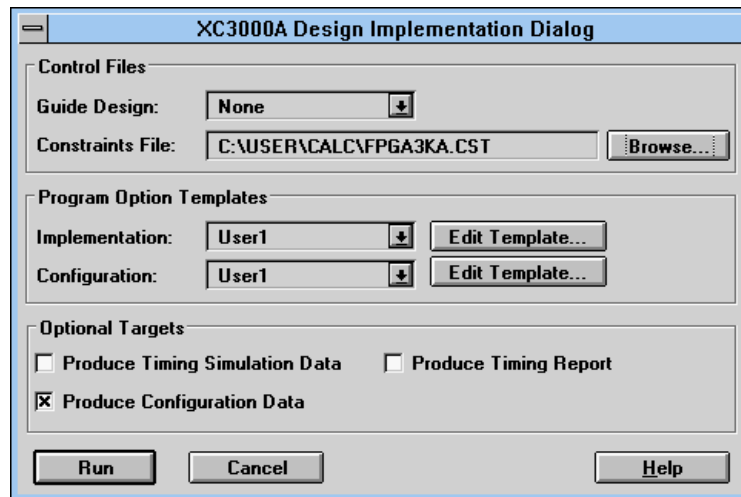
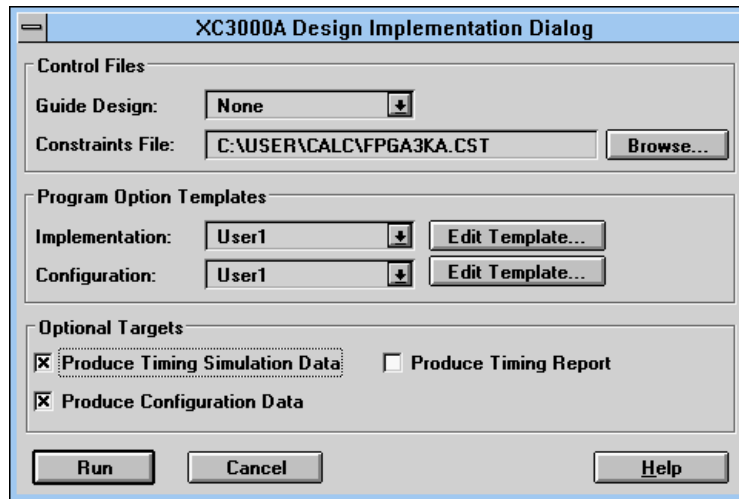


Figure 1-166 Updated Design Implementation Dialog Box

4. To produce timing simulation data, select the **Produce Timing Simulation Data** check box, as Figure 1-167 demonstrates.



**Figure 1-167 Selecting the Target**

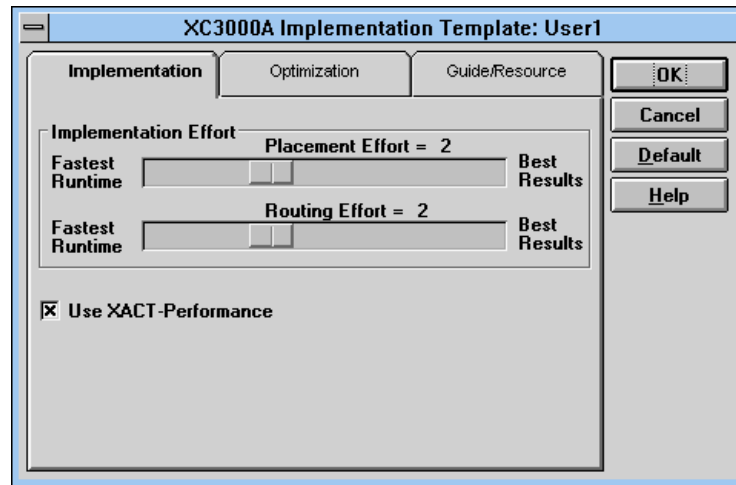
At this point, you are ready to run the Flow Engine to create the configuration and timing simulation data. The options in the Program Option Templates part of the Design Implementation dialog box, discussed in the next section, allow you to control how this data is produced.

### Setting Advanced FPGA Options

The advanced options are found in the Program Option Templates section of the Design Implementation dialog box.

1. Click on the **Edit Template** button in the Implementation field.

The Implementation Template dialog box appears, as shown in Figure 1-168.



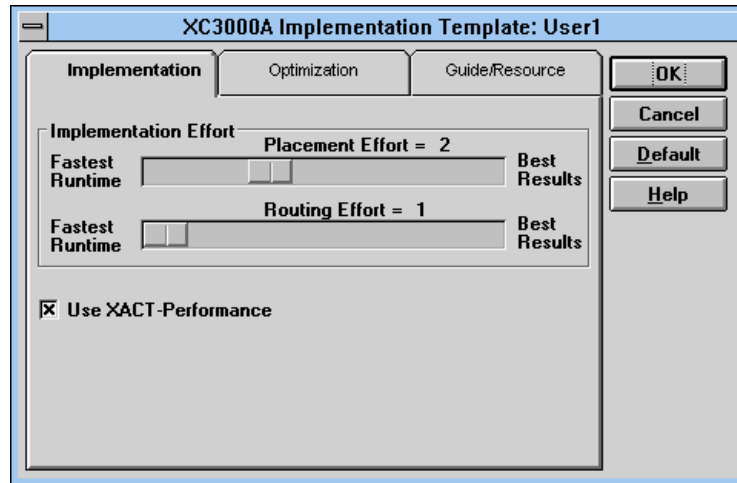
**Figure 1-168 Implementation Template Dialog Box**

The Implementation Template is divided into three tabs, each allowing you to specify implementation options.

- The Implementation tab specifies placement and routing effort. It also lets you decide if XACT-Performance should be used.
- The Optimization tab controls the types of optimization applied to the design during implementation.
- The Guide/Resource tab determines which resources will be used when you use a previous implementation to guide a design.

For more information on these options, consult the *Design Manager/Flow Engine Reference/User Guide*.

2. While holding the left mouse button down on the routing slide bar, set the routing effort to 1, as shown in Figure 1-169.



**Figure 1-169 Setting the Routing Effort**

3. Select the optimization tab by clicking the left mouse button on the title.

The Optimization tab is displayed in the Implementation Template dialog box, as shown in Figure 1-170.

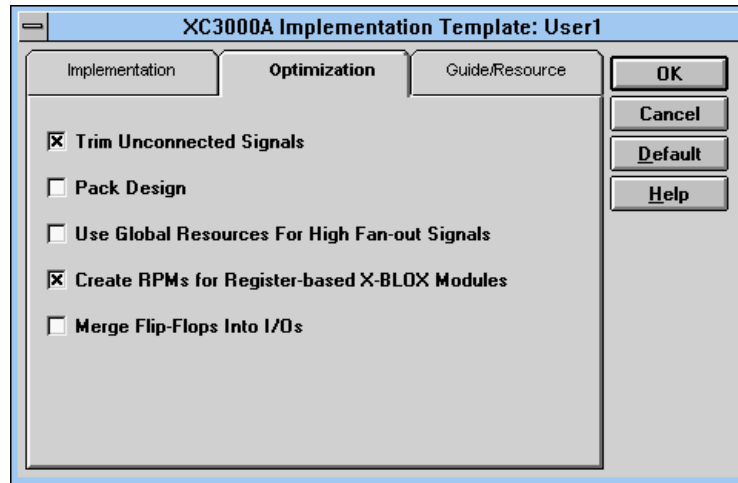


Figure 1-170 Optimization Tab of the Implementation Template

4. To specify that the design be implemented in the fewest number of CLBs, select the **Pack Design** check box, as Figure 1-171 indicates.

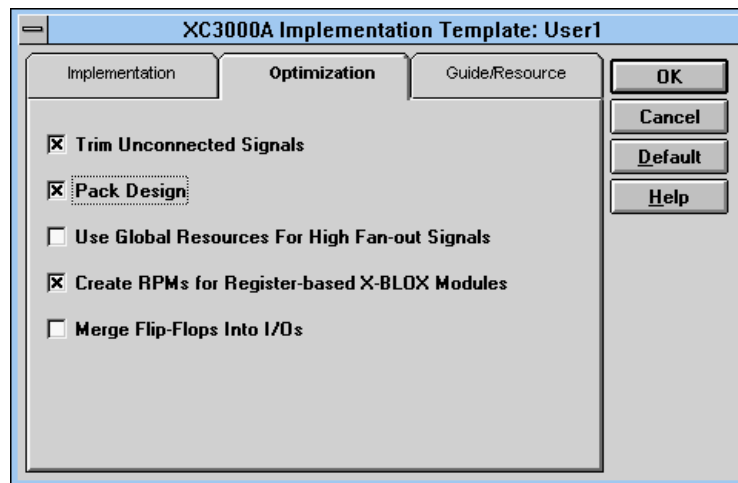
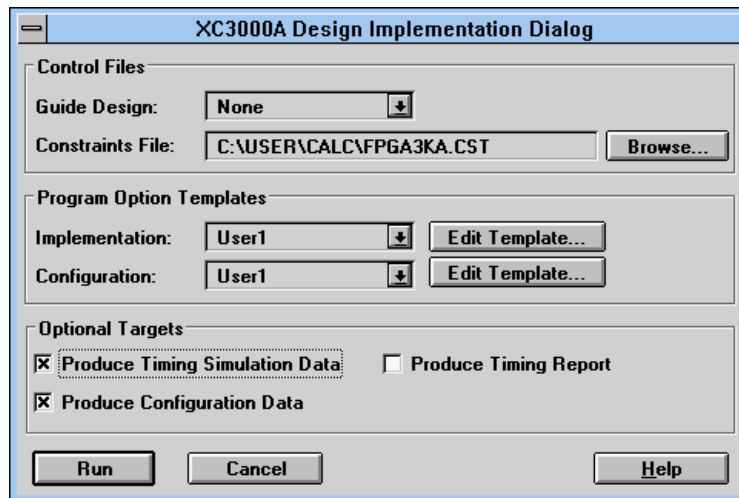


Figure 1-171 Selecting the Pack Design Option

5. Set any other options, if desired.
6. Click on **OK** to close the Implementation Template dialog box.

All the options on the Design Implementation dialog box should now be set. Figure 1-172 displays the resulting dialog box.



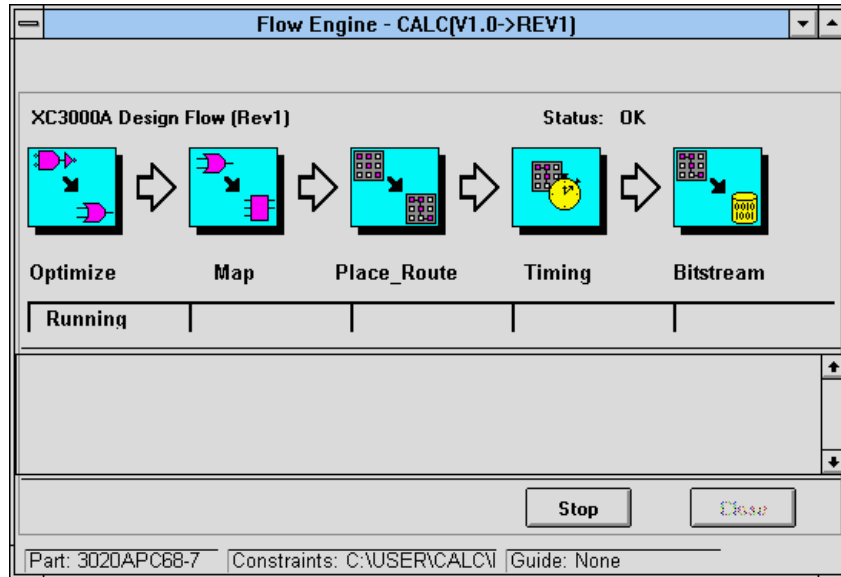
**Figure 1-172 Design Implementation Dialog Box with All Options Set**

## Invoking the Flow Engine

Now that all the options are set, you can invoke the Flow Engine.

1. Click on **Run** in the Design Implementation dialog box.

This command closes the dialog box and opens the Flow Engine, which handles the processing of the design from optimization to the generation of the bitstream. As each step is completed, the Flow Engine window is updated with the processing status. The first step is the optimization step, which is performed by the XNFPrep program. The Flow Engine window shown in Figure 1-173 displays the optimization step.

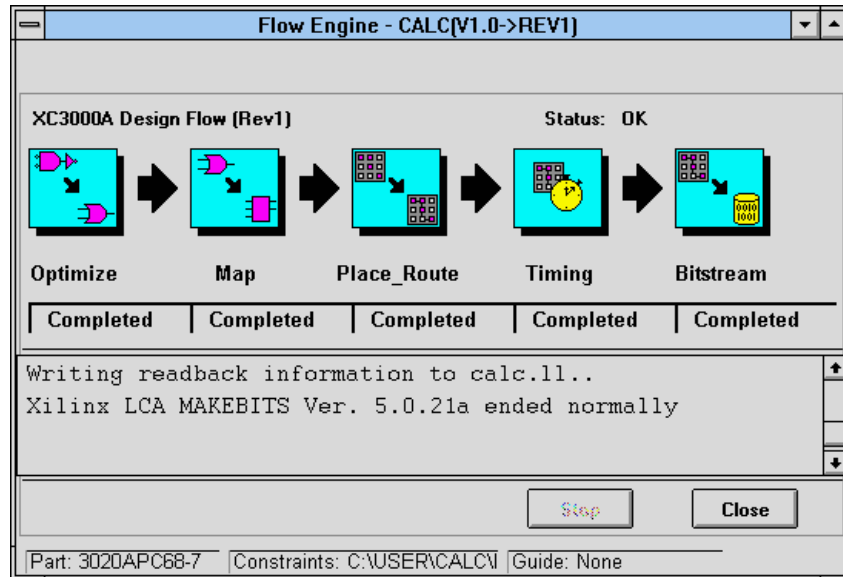


**Figure 1-173 Flow Engine Window During Optimization Step**

Once optimization is completed, the Flow Engine places and routes the design. The place and route stage creates the layout and routes the interconnect. Finally, the Flow Engine creates back-annotated timing data and generates a bitstream. The completion of these processes produces the data necessary for PROflow to create a timing simulation network and to download the design.

Figure 1-174 shows the Flow Engine as it completes processing.

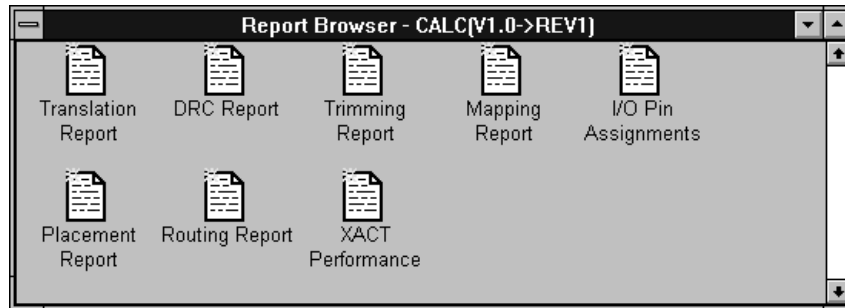




**Figure 1-174 Flow Engine Window on Completion of Processing**

When you click on the Run button in the Design Implementation dialog box, the Report Browser is also displayed. During processing, the Report Browser is updated with reports as they are created. Figure 1-175 illustrates the Report Browser once processing is complete.

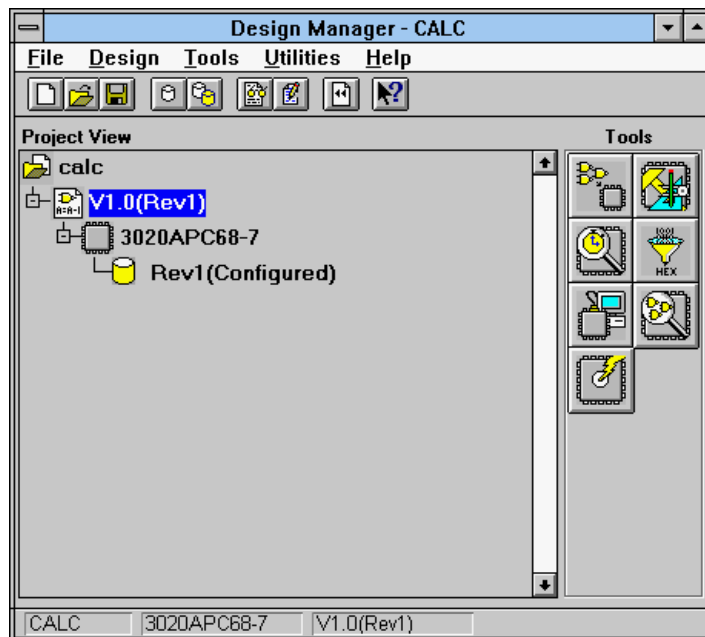
To view any of the reports, double-click on the appropriate icon in the Report Browser.



**Figure 1-175 Report Browser**

2. Close the Flow Engine and the Report Browser.

The Design Manager is updated, as shown in Figure 1-176.



**Figure 1-176 Updated Design Manager Window**

## Implementing the EPLD Design

As with FPGAs, the Design Manager uses the Flow Engine to implement an EPLD design. For EPLD designs, the Flow Engine creates configuration data, *design.prg*, as well as timing simulation data. It automatically exports this data to the Viewlogic project directory for use by PROflow.

1. To set the implementation options and invoke the Flow Engine, select **Design** → **Implement** in the Design Manager, as shown in Figure 1-177.

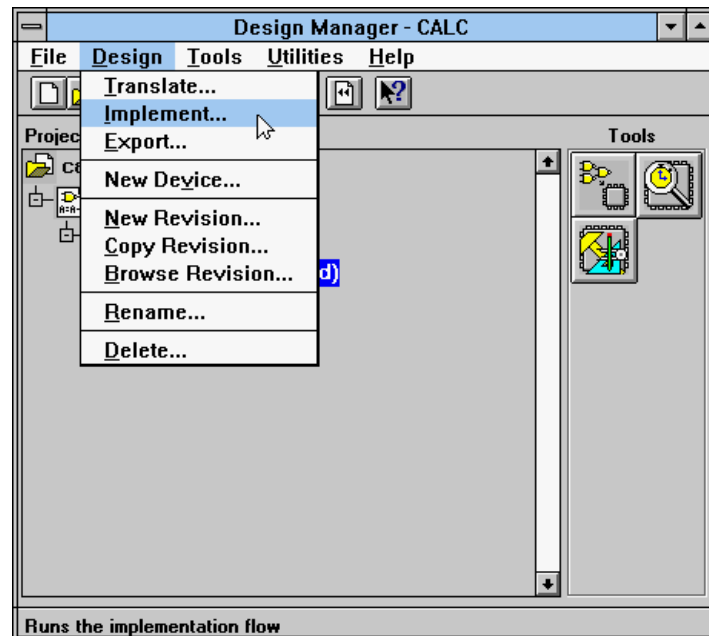
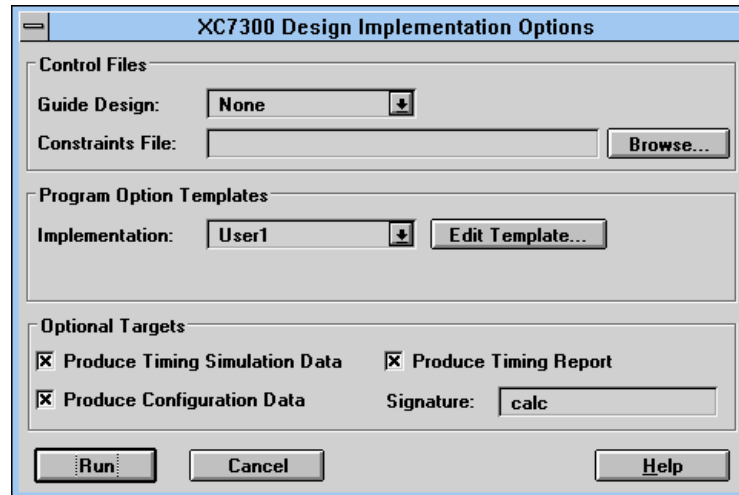


Figure 1-177 Selecting the Implement Command

The Design Implementation dialog box opens, as indicated in Figure 1-178.



**Figure 1-178 Design Implementation Dialog Box**

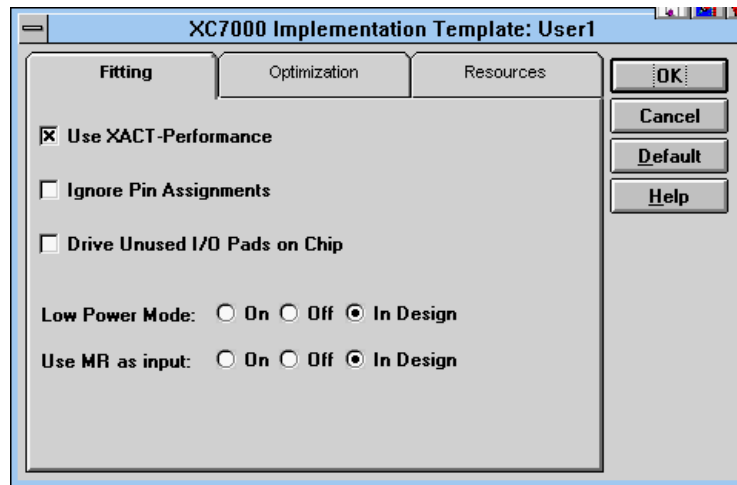
In this dialog box, you can select control files, create option templates, and specify optional targets. In the Constraints File field, you can select any constraints file. The Guide Design field displays only the revisions containing guide data. For more information on control files, see the *Design Manager/Flow Engine Reference/User Guide*. By default, all the optional targets are selected.

### Setting Advanced EPLD Options

The advanced options are found in the Program Option Templates section of the dialog box.

1. Click on the **Edit Template** button in the Implementation field.

The Implementation Template dialog box appears, as shown in Figure 1-179.



**Figure 1-179 Implementation Template Dialog Box**

The Implementation Template is divided into three tabs, each allowing you to specify implementation options.

- The Fitting tab allows you to use XACT-Performance, ignore pin assignments, and specify that unused I/O pads be driven.
- The Optimization tab controls the types of optimization applied to the design during implementation.
- The Resource tab determines which resources will be applied when you use a previous implementation to guide a design.

For more information on these options, consult the *Design Manager/Flow Engine Reference/User Guide*.

2. Select the optimization tab by clicking the left mouse button on the title.

The Optimization tab is displayed in the Implementation Template dialog box, as indicated in Figure 1-180.

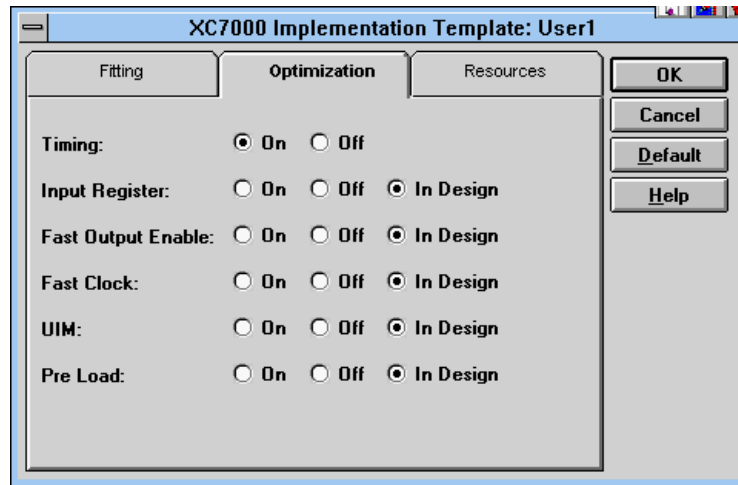


Figure 1-180 Optimization Tab of the Implementation Template

3. To use the fast clock capability of the EPLD device, set the **Fast Clock** option to On, as Figure 1-181 demonstrates.

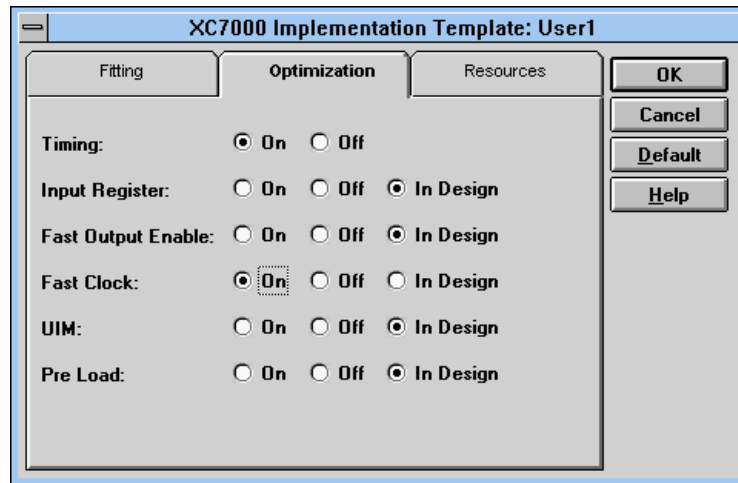
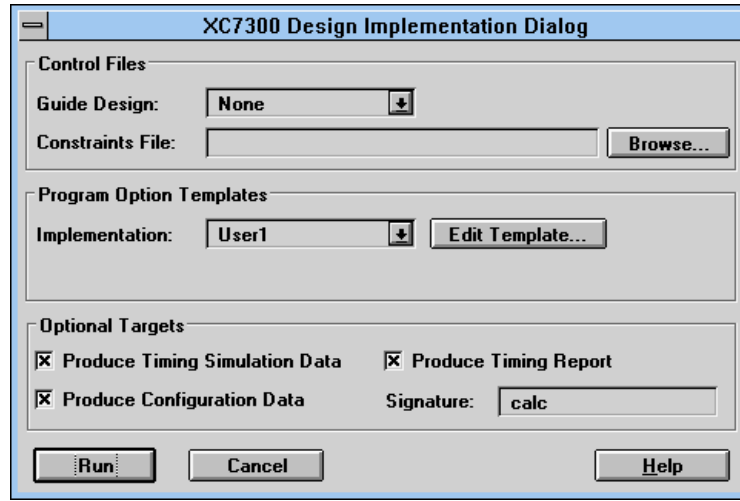


Figure 1-181 Setting the Fast Clock Option to On

- Click on **OK** to close the Implementation Template dialog box.  
All the options on the Design Implementation dialog box should now be set. Figure 1-182 displays the resulting dialog box.



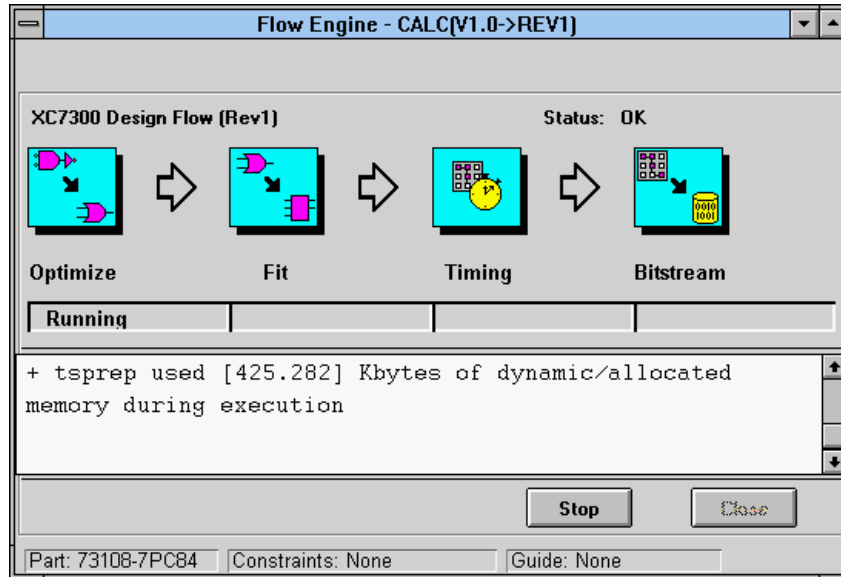
**Figure 1-182 Design Implementation Dialog Box with All Options Set**

## Invoking the Flow Engine

Now that all the options are set, you can invoke the Flow Engine.

- Click on **Run** in the Design Implementation dialog box.

This command closes the dialog box and opens the Flow Engine, which handles the processing of the design from optimization to the generation of the program data. As each step is completed, the Flow Engine window is updated with the processing status. The first step is the optimization step, which is performed by the XNFPrep program. The Flow Engine window shown in Figure 1-183 displays the optimization step.

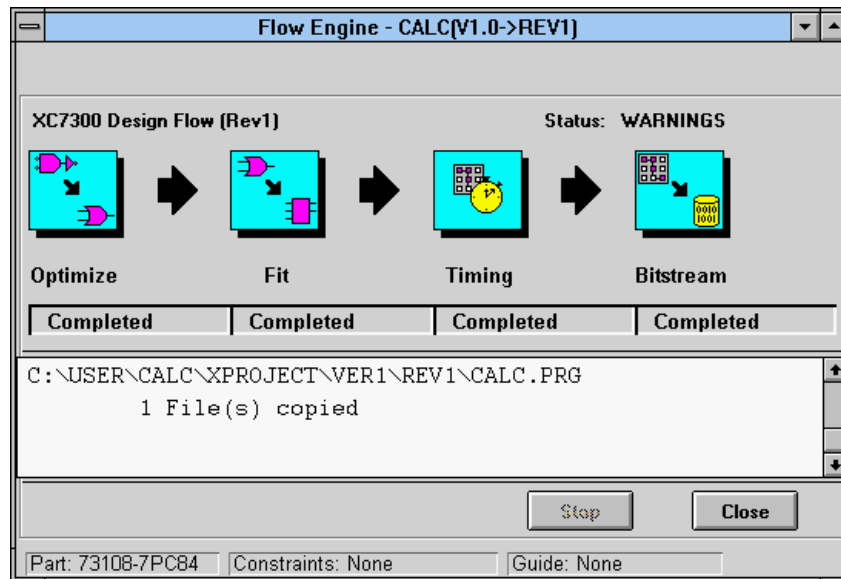


**Figure 1-183 Flow Engine Window During Optimization Step**

Once optimization is completed, the Flow Engine fits the design. Finally, the Flow Engine creates back-annotated timing data and program data. The completion of these processes produces the data necessary for PROflow to create a simulate timing network.

Figure 1-184 shows the Flow Engine as it completes processing.

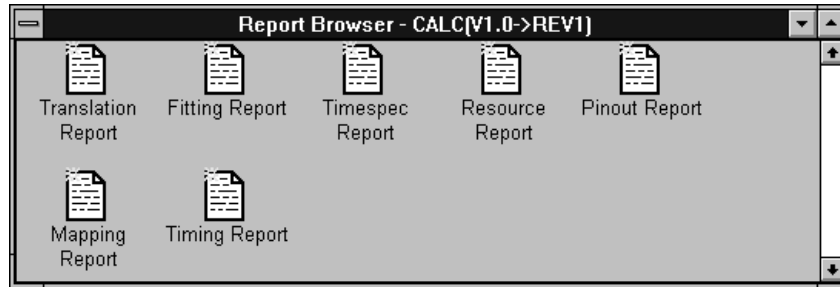




**Figure 1-184 Flow Engine Window on Completion of Processing**

When you click on the Run button in the Design Implementation dialog box, the Report Browser is also displayed. During processing, the Report Browser is updated with reports as they are created. Figure 1-185 illustrates the Report Browser once processing is complete.

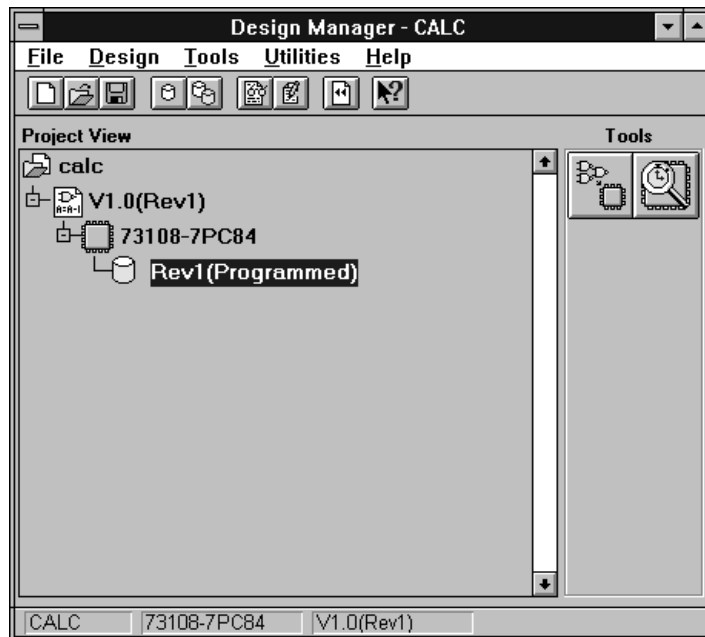
To view any of the reports, double-click on the appropriate icon in the Report Browser.



**Figure 1-185 Report Browser**

2. Close the Flow Engine and the Report Browser.

The Design Manager is updated, as shown in Figure 1-186.



**Figure 1-186 Updated Design Manager Window**

## Timing Simulation

With the FPGA design placed and routed, or the EPLD design fitted, the next step is to verify the timing by using PROsim. PROflow automates this task so that all you need is a timing simulation network to represent the routed FPGA or fitted EPLD.

### Creating the Simulation Network File

The first step in the timing simulation process is to create the simulation network, calc.vsm, which is loaded into PROsim to simulate the Calc design.

1. Click on the Timing Simulation PROsim icon, shown in Figure 1-187. It is located below the Xilinx Implementation icon.

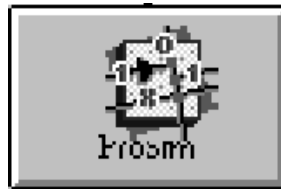


Figure 1-187 Timing Simulation PROsim Icon

The Timing Simulation dialog box appears, as shown in Figure 1-188.

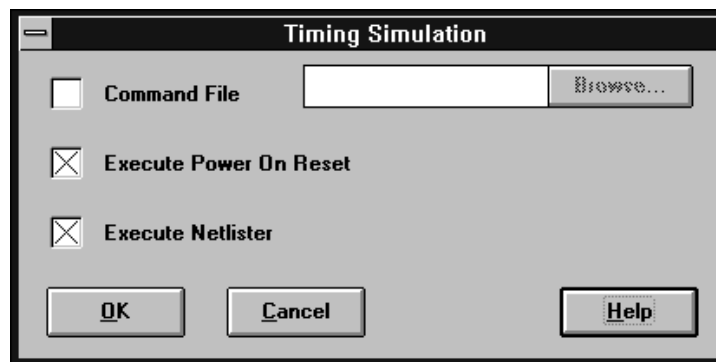
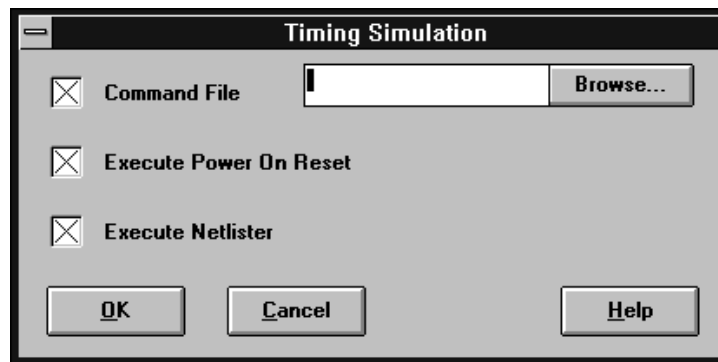


Figure 1-188 Timing Simulation Dialog Box

The Execute Power On Reset option is set by default. A timing simulation command file has already been prepared for you. It issues exactly the same commands as those for functional simulation.

2. Click on the empty **Command File** check box.

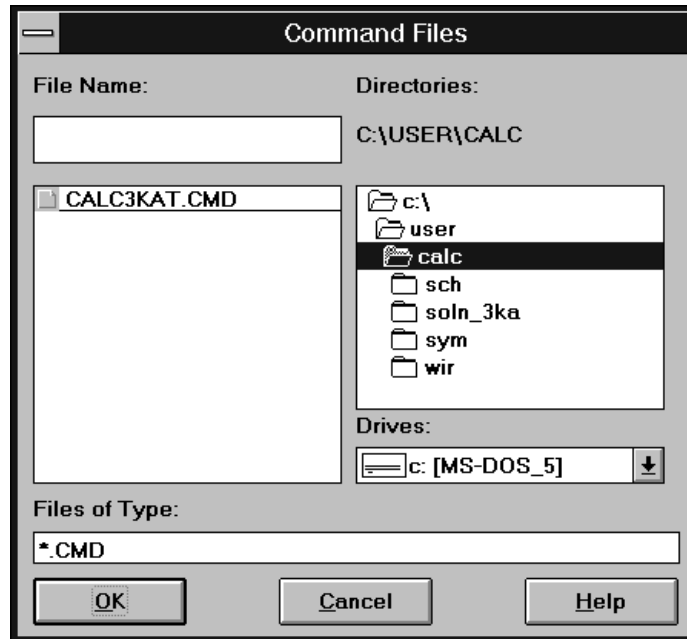
The Command File field is now activated, as indicated in Figure 1-189, so that you can type in the name of the command file. The Browse button is also activated.



**Figure 1-189 Selecting the Command File**

3. Click on **Browse**.

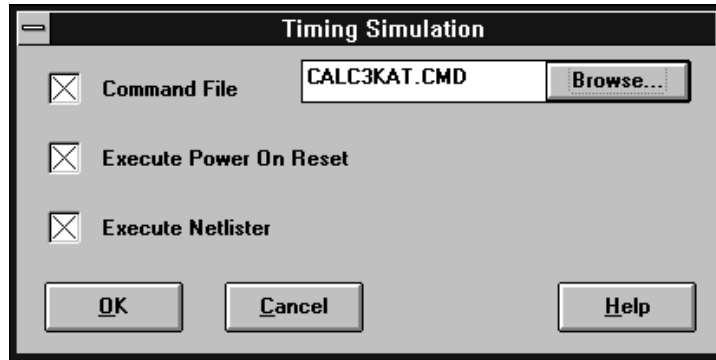
This option brings up the Command Files dialog box, shown in Figure 1-190.



**Figure 1-190 Command Files Dialog Box**

4. Select the `calc3kat.cmd` file for FPGAs or the `calc7kt.cmd` file for EPLDs.
5. Click on `OK`.

The Command Files dialog box closes, and the Timing Simulation dialog box is updated, as pictured in Figure 1-191.



**Figure 1-191 Updated Timing Simulation Dialog Box**

**Note:** To view the PROsim commands, open the command file using Notepad or any text editor.

6. Click on **OK**.

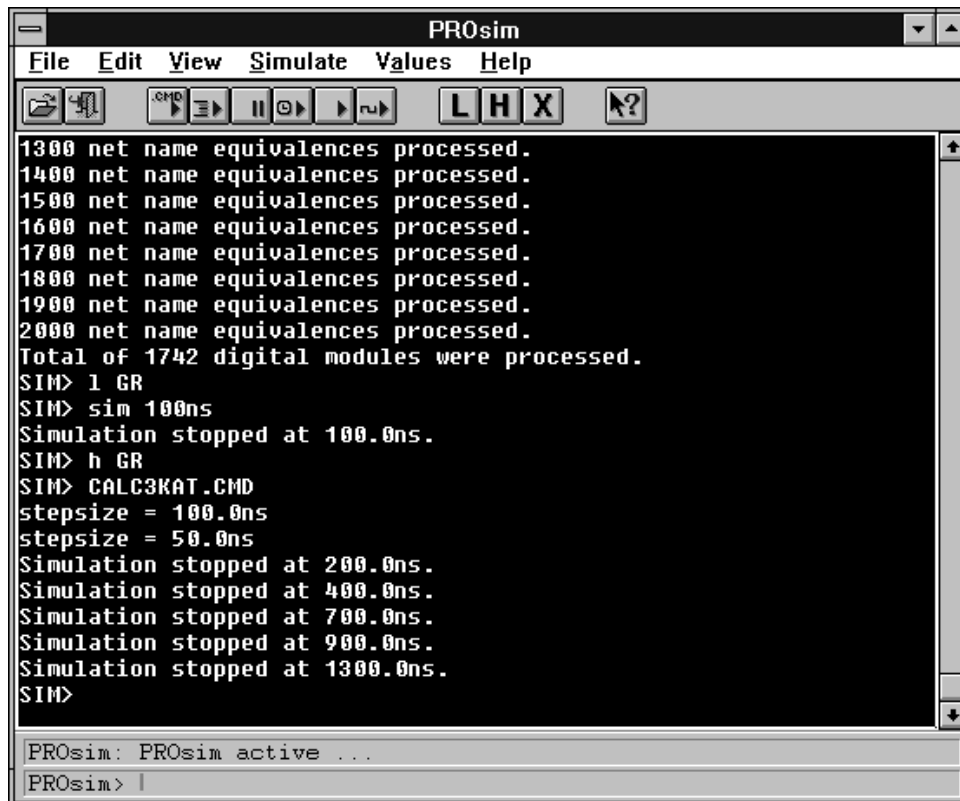
PROflow closes the Timing Simulation dialog box and prepares the timing simulation network, calc.vsm. When it is done, the calc.log file is displayed in Notepad. If the file is too large to be displayed in Notepad, you can use an alternate text editor.

**Note:** For EPLDs, the xsimmake.out file is displayed in Notepad.

7. Verify that no errors were issued during the processing.

8. Select the **File** → **Exit** option to close Notepad.

Once you close Notepad, PROflow invokes PROsim on the calc.vsm simulation network file, asserts the global reset, and runs the calc3kat.cmd or calc7kt.cmd file. The calc3kat.cmd or calc7kt.cmd file sets up the vectors, initializes all the inputs, and simulates the load and push operations as you did manually in the functional simulation section. Figure 1-192 displays PROsim's response to the commands issued by the command file.



The screenshot shows the PROsim application window. The title bar reads "PROsim". The menu bar includes "File", "Edit", "View", "Simulate", "Values", and "Help". Below the menu bar is a toolbar with icons for file operations and simulation control, along with buttons labeled "L", "H", and "X". The main window area is a black terminal with white text. The text shows the following sequence of commands and outputs:

```
1300 net name equivalences processed.
1400 net name equivalences processed.
1500 net name equivalences processed.
1600 net name equivalences processed.
1700 net name equivalences processed.
1800 net name equivalences processed.
1900 net name equivalences processed.
2000 net name equivalences processed.
Total of 1742 digital modules were processed.
SIM> 1 GR
SIM> sim 100ns
Simulation stopped at 100.0ns.
SIM> h GR
SIM> CALC3KAT.CMD
stepsize = 100.0ns
stepsize = 50.0ns
Simulation stopped at 200.0ns.
Simulation stopped at 400.0ns.
Simulation stopped at 700.0ns.
Simulation stopped at 900.0ns.
Simulation stopped at 1300.0ns.
SIM>
```

At the bottom of the window, there is a status bar that says "PROsim: PROsim active ..." and a command prompt "PROsim> |".

Figure 1-192 PROsim and Simulated Command File

## Invoking PROwave

To verify the simulation, you can use PROwave to view the signal waveforms.

1. To invoke PROwave, activate PROflow by making an icon of PROsim and selecting PROflow with the left mouse button.
2. Click on the Timing Simulation PROwave icon, shown in Figure 1-193.



Figure 1-193 Timing Simulation PROwave Icon

The Open dialog box appears, as shown in Figure 1-194.

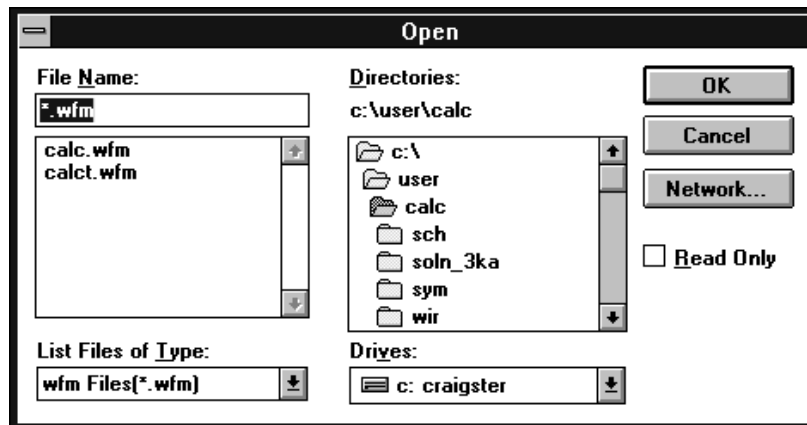


Figure 1-194 Open Dialog Box

There are two waveform files in the current project so you can compare the functional and timing waveforms. The first, `calc.wfm`, was created during functional simulation. The second, `calct.wfm`, was created by the `calc3kat.cmd` or `calc7kt.cmd` command file.

3. In the File Name list box, select the `calct.wfm` waveform display file.
4. Click on `ok`.

The Open dialog box closes, and the `calct.wfm` waveform display file opens, as illustrated in Figure 1-195.



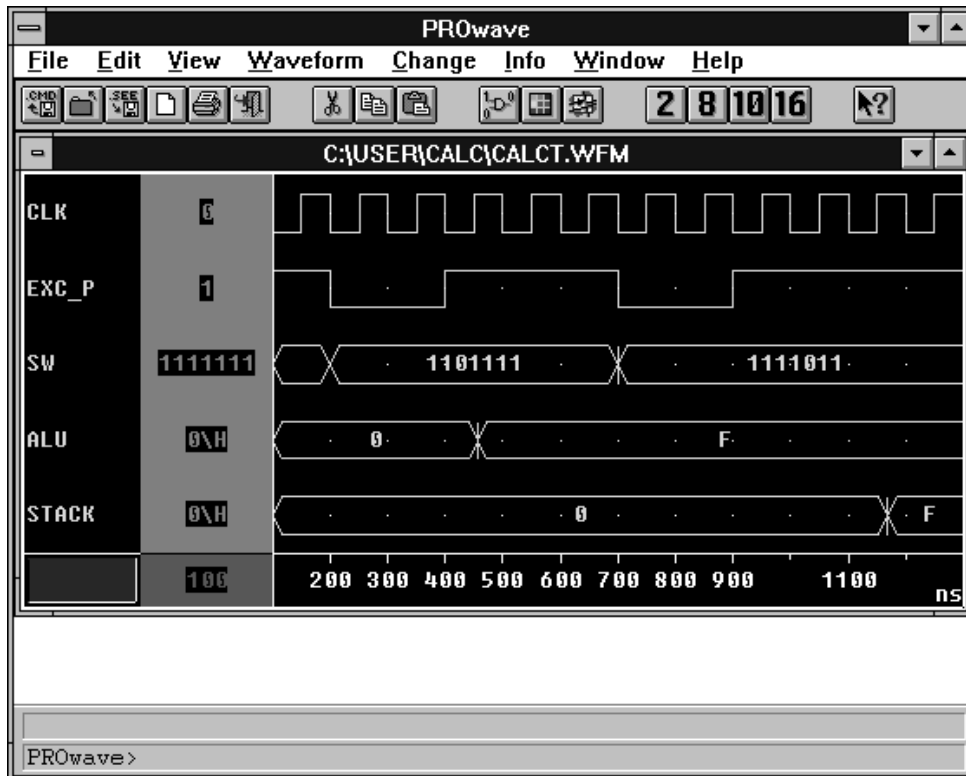


Figure 1-195 Calct.wfm (Timing Simulation) Waveform Display File

## Comparing the Functional and Timing Simulation Files

The timing simulation waveform in calct.wfm should look almost identical to the functional simulation waveform in calc.wfm. To verify it, open calc.wfm.

1. Select the **File** → **Open** → **PROsim** command.

The Open dialog box appears, as shown in Figure 1-196.

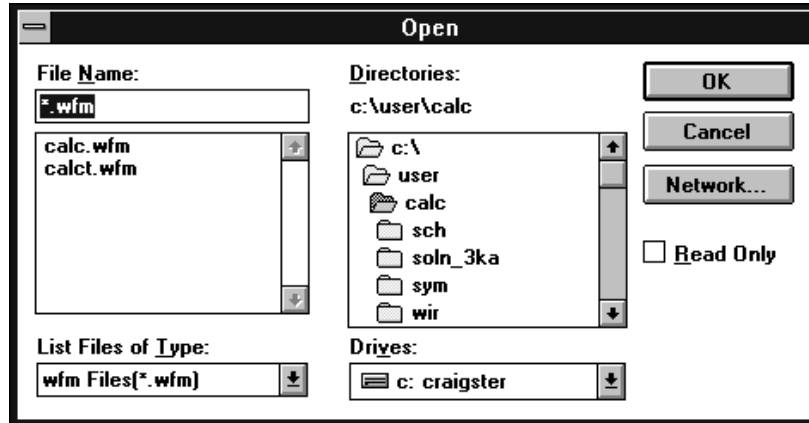


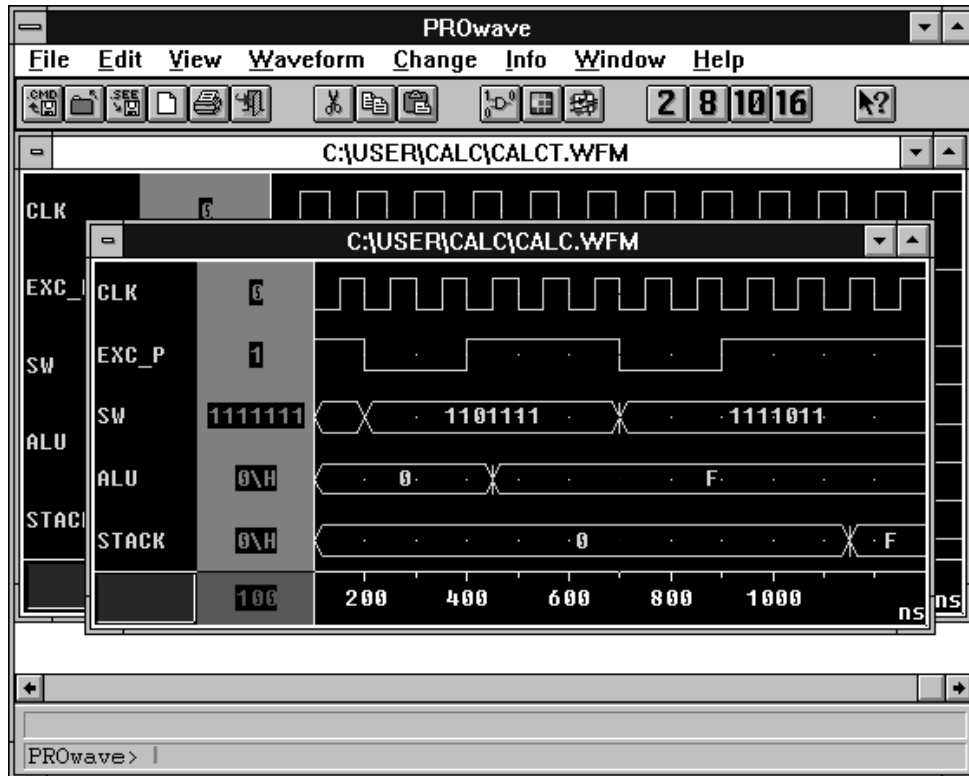
Figure 1-196 Open Dialog Box

2. In the File Name list box, select the `calc.wfm` waveform display file to open the original functional simulation file. If you saved the file as `calc.see` in order to experiment with different operations during functional simulation, enter `see Files (*.see)` in the List Files of Type list box, and select `calc.see`.

**Note:** The tutorial illustrations reflect the `calc.wfm` file, not the `calc.see` file.

3. Click on `OK`.

The Open dialog box closes, and the `calc.wfm` waveform display file opens, as illustrated in Figure 1-197.



**Figure 1-197 Calc.wfm (Functional Simulation) Waveform Display File**

4. Select the **Window** → **Tile** command, which arranges the windows side by side, as Figure 1-198 demonstrates.

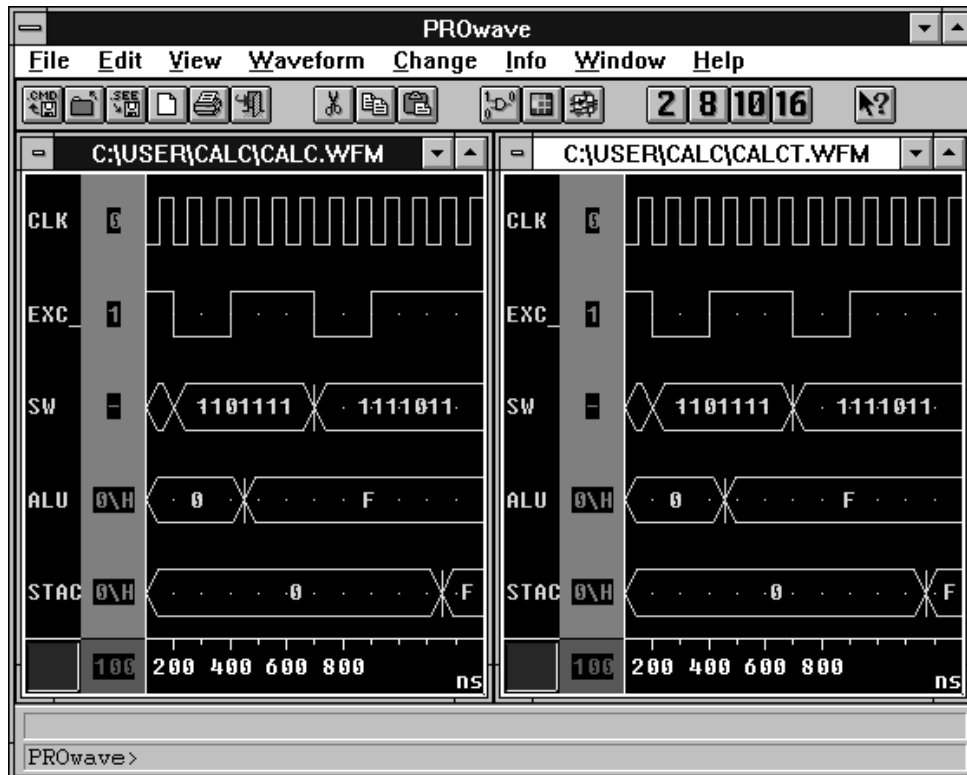


Figure 1-198 Tiled Waveforms

### Zooming the Waveform Files

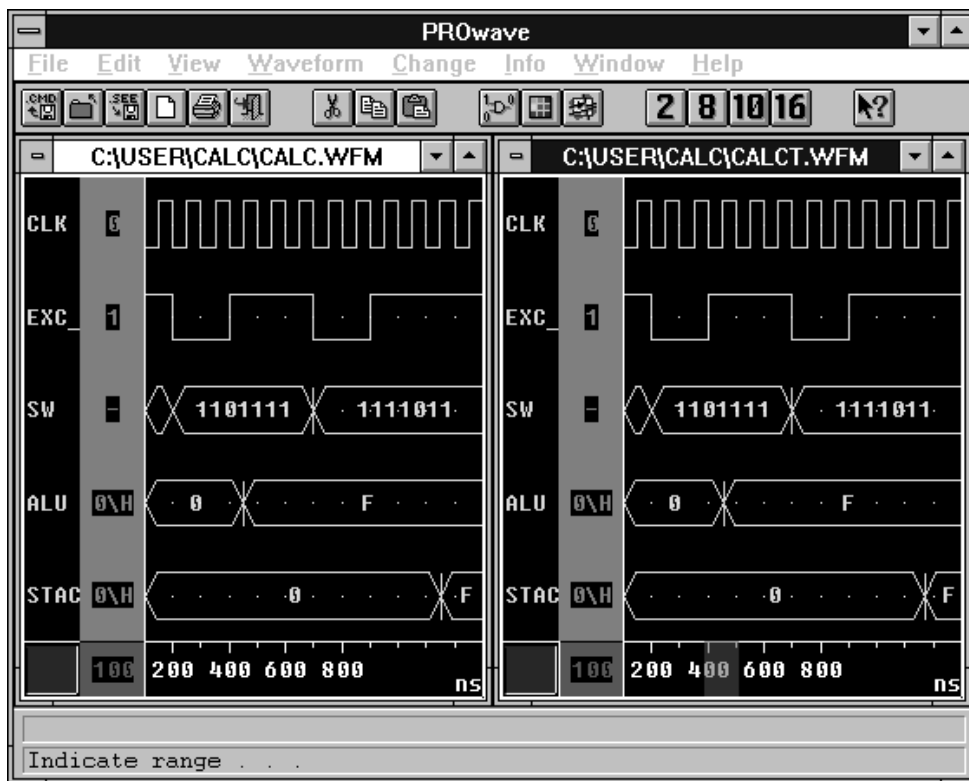
At the present zoom level, it is extremely difficult to distinguish any differences between the two waveforms. Zooming in and changing the grid spacing of the values makes the inspection of the two waveforms much easier.

Zoom into the calct.wfm ALU transition from 0 to F using the following procedure.

1. Select the calct.wfm waveform window by clicking the left mouse button on the calct.wfm title bar.

2. Point the mouse at the 400-ns hash mark at the bottom of the calct.wfm waveform.
3. Press the **F9** function key to begin the View → Region command.

The region that will become the new display area is denoted by the red fill area, shown as a colored area over the 500-ns hash mark in Figure 1-199.



**Figure 1-199 Calct.wfm Display Region**

4. Point the mouse at the 500-ns hash mark.
5. Click the left mouse button to zoom into the defined region. Figure 1-200 displays this zoomed region.



Figure 1-200 Zoomed Region in Calc.t.wfm Display

6. To change the hash mark spacing, select the **View** → **Grid** → **Space** command.

PROwave prompts for the horizontal step spacing at the PROwave command line, as indicated in Figure 1-201.



**Figure 1-201 Horizontal Step Prompt**

7. At the `Horizontal step [100]` prompt, type `15`.

Specifying 15 for the horizontal step changes the interval of the hash marks to 15 ns, as Figure 1-202 illustrates.

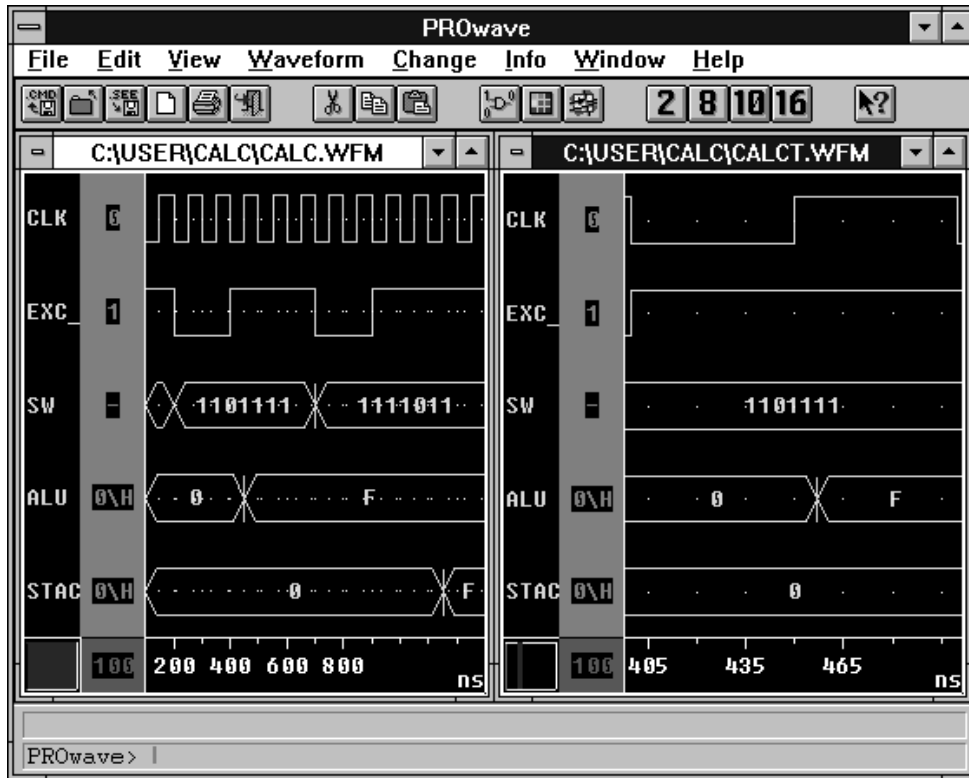


Figure 1-202 Changed Step Interval in Calct.wfm Display

8. Activate the calc.wfm waveform and zoom in to the same region, specifying the same hash mark spacing, as shown in Figure 1-203.



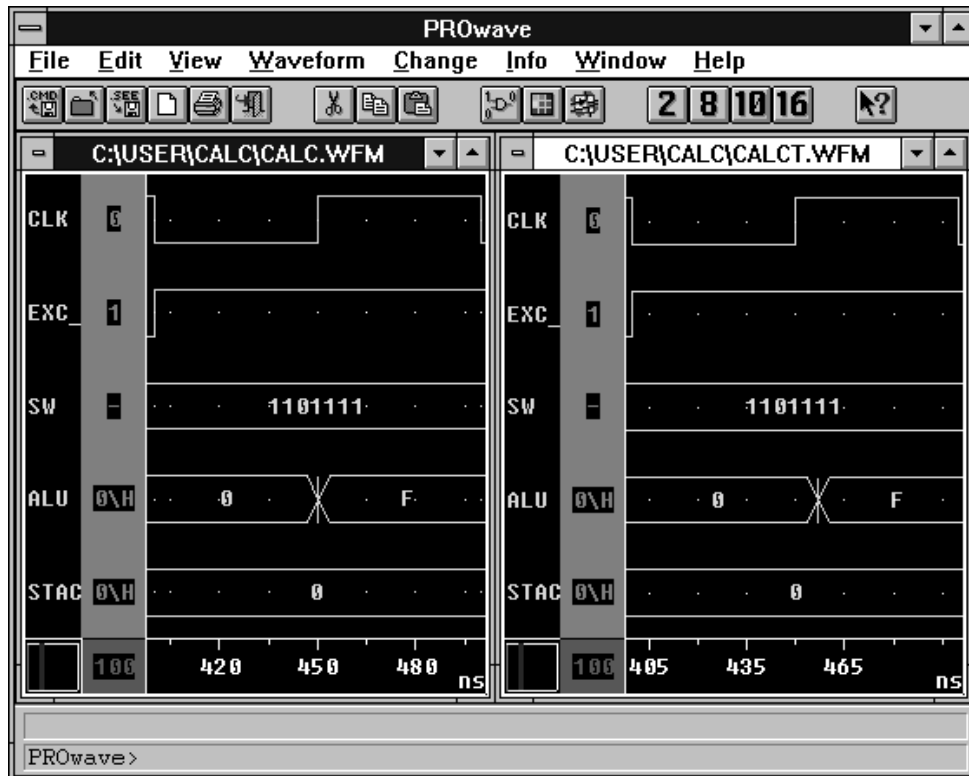


Figure 1-203 Changed Step Interval in Calc.wfm Display

### Obtaining a Transition Time

It is now obvious that the two waveforms are different. Calc.wfm, the functional simulation file, shows the ALU transition just after 450 ns; calct.wfm, the timing simulation file, shows the transition slightly before 460 ns. To obtain an exact time for each of the transitions, place the crosshair at each of the transitions.

**Note:** The specific transition times mentioned here are for FPGAs, although the results for EPLDs are similar.

Your design may be different because of differences in placing and routing.

### In the Calc.wfm File

To find out the exact transition time of the ALU vector from 0 to F in the calc.wfm waveform, follow these steps.

1. Point the mouse at the vertical line in the middle of the ALU transition from 0 to F.
2. Double-click the left mouse button.

This step places the crosshair at the transition, as shown in Figure 1-204.

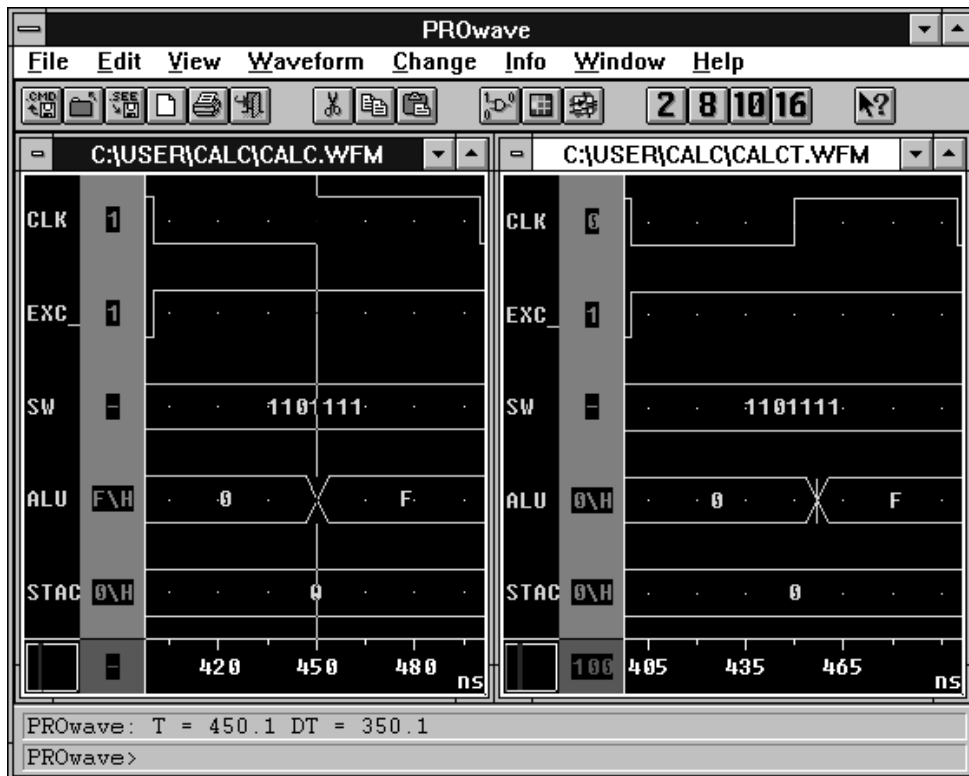


Figure 1-204 ALU Transition in Calc.wfm Display

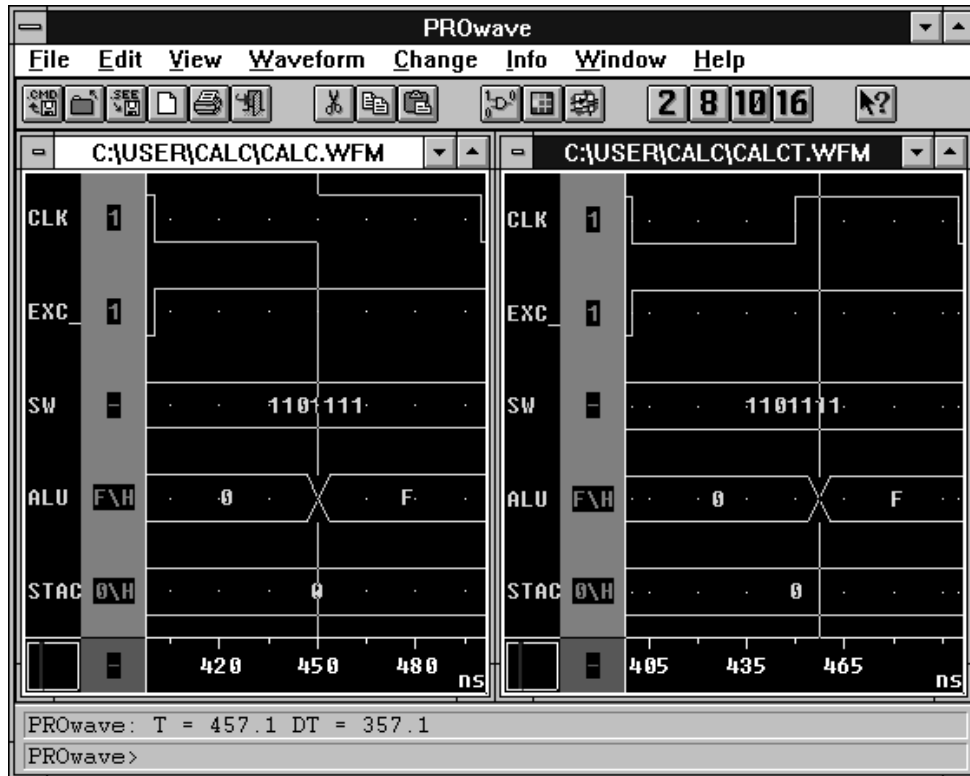
PROwave also displays the time at which the transition was made in the PROwave status line. For this particular transition, PROwave reported that the transition was made at 450.1 ns and that the delta time (DT) from the last crosshair location was 350.1 ns.

### **In the Calct.wfm File**

To find out the exact transition time of the ALU vector from 0 to F in the calct.wfm waveform, use this procedure:

1. Activate the calct.wfm waveform by clicking the left mouse button on the title bar.
2. Point the mouse at the vertical line in the middle of the ALU transition from 0 to F.
3. Double-click the left mouse button.

This step places the crosshair at the transition, as shown in Figure 1-205.



**Figure 1-205 ALU Transition in Calct.wfm Display**

For this particular transition, PROwave reported that the transition was made at 457.1 ns and that the delta time (DT) from the last cursor location was 377.1 ns.

### Obtaining a Delta Time

You can obtain a delta time between the transition of two signals by double-clicking on the first transition, then double-clicking on the second transition. The delta time is reported in the PROsim status bar located above the PROsim command line.

To find out the delta time from the rising edge of CLK or OSC\_7K\XCLK to the transition of the ALU vector, double-click the

left mouse button on the rising edge of the CLK or OSC\_7K\XCLK signal. For this route, PROwave reported 7.1 ns, as you can see in Figure 1-206.

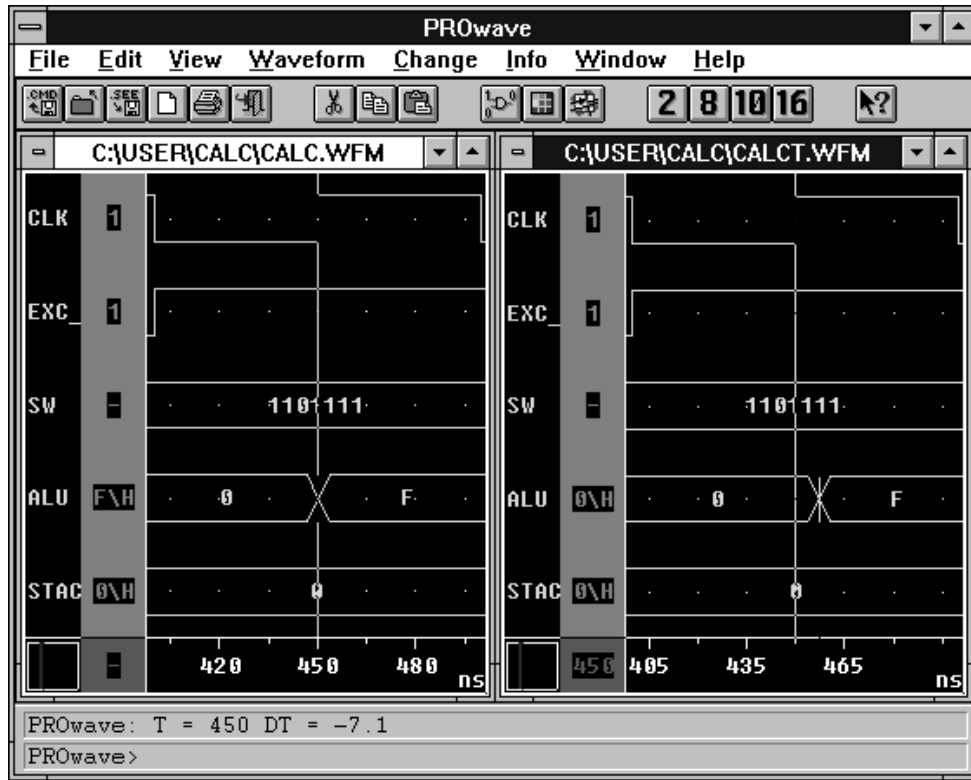


Figure 1-206 Delta Time in Calct.wfm Display

Further compare the two waveforms by zooming in and out and double-clicking on the various transitions.

## Downloading an FPGA Design

Now that you have completed your FPGA design, you are ready to download it to a demonstration board.

The following section uses the XACT 5.2 DOS-based programs to generate and download the bitstream. To learn about this process in the XACTstep V6.0 tools, refer to the “CALC Tutorial Update” chapter of the *Hardware Debugger Reference/User Guide*.

## Using a Demonstration Board

There are three Xilinx demonstration boards in common use. The board that you have depends on what software you purchased and when you bought it. Because the FPGA tutorial targets an XC3020A device, you can download the design only to the first of the following two boards.

- The FPGA demonstration board includes both an XC3000A family socket and an XC4000 family socket. You will use the XC3000A socket.
- The XC3000A demonstration board includes a single XC3000A socket.
- The XC4000 demonstration board includes a single XC4000 socket.

To load the configuration bitstream to the demonstration board, you need one of Xilinx’s 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 to the FPGA on a Xilinx demonstration board, the board must be correctly connected 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 deliver power to the board and cable.

Connect one end of the cable to your demonstration board, as shown in Table 1-3. For the FPGA demonstration board, use the leftmost column of pins, which is missing the pin in the third position.

**Table 1-3 Cable Connections for XC3000A and FPGA Demonstration Boards**

XC3000 Board	Cable Label	FPGA Board	Cable Label
J1-1	V <sub>CC</sub>	J1-1	V <sub>CC</sub>
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

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 “FPGA Demonstration Board” chapter of the *Xilinx Hardware and Peripherals Guide* discuss in detail the various demonstration boards and how to connect them. 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.
- Verify that the power supply is connected to the demonstration board *and is turned on*. The power connections for the supported demonstration boards are shown in Table 1-4.

**Table 1-4 Demonstration Board Power Connections**

XC3000 Board		FPGA Board	
J3-1	+ 5 volts	J9-1	+5 volts
J3-2	Gnd	J9-2	Gnd

## FPGA 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 switches should be set as shown in Table 1-5.

**Table 1-5 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

## XC3000A Demonstration Board

If you have an XC3000A 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.



## Downloading the Bitstream

Once the cable is connected to your PC, you are ready to download the bitstream.

1. Access MS-DOS by double-clicking on the MS-DOS program icon, shown in Figure 1-207, in the Program Manager XACTstep program group.



Figure 1-207 MS-DOS Program Icon

2. Change the directory to `c:\user\calc`.
3. To create a BIT file for downloading, run the MakeBits program by typing the following:

```
makebits -s1 calc
```

4. Set all the input switches High.

This setting selects the No-Op command; SW3 switches High on the FPGA board, and SW1 switches to "1" on the XC3000 board.

**Note:** The XChecker software is used with either hardware cable.

5. Invoke XChecker from the system prompt. Type the following:

```
xchecker -port portname calc↵
```

Here is an example:

```
xchecker -port com2 calc↵
```

Valid port names on a PC are com1 or com2, which must be in lower case.

Once you have used XChecker and set the correct port, that information is saved in a file called xchecker.pro in your design directory; you do not have to specify it each time.

6. Press the ↵ key.

If the FPGA is successfully programmed, the following message appears:

```
DONE signal went high.
```

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 repeat steps 5 and 6.

## Testing the Design

As described in the “Introduction” chapter at 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 inputs 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 board or SW1 on the XC3000 board). The leftmost switch, labeled 1, is the Execute command, which is activated by toggling the switch twice. The next three switches (labeled 2-4) select the opcode. Opcode encoding is shown in the “Processor Operations” table in the “Functional Simulation” section of this manual. Use the rightmost four switches (labeled 5-8) to input data. When the extended instruction set is selected with opcode 111, these switches provide additional bits of opcode. Exercise caution when setting the switches because they may yield unintended results.

**Note:** The rocker switches on the XC3000A demonstration board are On when down, Off when up. Use the 0 and 1 labels on the board as your guide.

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 that 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 initial 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 101x, 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 110x, 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.

# ***Viewlogic Tutorials***

***X-BLOX Tutorial***



## Chapter 2

### 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/L, XC3100A, XC4000, and XC5200 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 Viewlogic PRO Series 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 “PROcapture and PROsim Tutorial” chapter of this manual. If not, please review that chapter before continuing.

### Required Software

You should have access to the following software:

- Viewlogic PROcapture, the Viewlogic schematic entry tool
- Viewlogic PROsim, the Viewlogic simulation tool
- XACT Core Implementation Tools, version 6.0 or later
- X-BLOX (DS-380), which is included in the Standard (DS-VLS-STD-PC1) and Extended (DS-VSL-EXT-PC1) packages. The Base package does not include X-BLOX, but you can purchase it separately.

You should have at least temporary access to all of the software just listed using the temporary licensing available on the programmable key, provided that the temporary licensing has not already been exhausted.

## Preparing the Design

This tutorial uses the completed Calc design, which you can create either by completing the PROcapture tutorial or by copying a completed design from one of the solutions directories.

**Note:** All of the screen outputs refer to the processing of the XC3000A solutions design on a PC. Other parts or platforms have different outputs.

A full solution for the PROcapture tutorial is supplied in the solution directory located under the directory where the PROseries software was installed:

... \tutorial\vwlogic\procalc\calc3ka\soln\_3ka

1. Using the Windows File Manager, copy the contents of this directory to the directory where you will be performing the X-BLOX tutorial.
2. Invoke Xilinx PROflow and select the Design Entry icon.
3. Select the **Project Manager** button.
4. Press the **Create** button.
5. Double-click on the directory that you made in step 1 in the Create Project dialog box.
6. Verify that the directory is shown in the Directory text box, and press **OK**.

The PROJman Create dialog box appears.

7. Select **No** to re-initialize the viewdraw.ini file.

**Note:** This step assumes that the default viewdraw.ini file found in ... \proser\standard is configured correctly for PRO Series installation.

8. Click on **Exit** in the PRO Series Project Manager.
9. In the Select Family dialog box, select **XC3000A** and click on **OK**.



10. After exiting from the Select Family box, select the **CALC.1** design in the Design Entry dialog box, and select **OK** to open the schematic.

## 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 discussion in the “Design Description” section of the “PROcapture and PROsim Tutorial” chapter.

### Adding X-BLOX Modules to CALC

In this section, you will insert an X-BLOX-based replacement for the ALU instance on the CALC schematic. 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. Select the ALU instance on the CALC schematic.
2. From the **Change** menu, select **Component** and then **ALU\_BLOX.1**.

This procedure replaces the original ALU block with **ALU\_BLOX**. The change is reflected by the appearance of the **ALU\_BLOX** name at the top of the symbol.

### Viewing the ALU\_BLOX Schematic

Now view the schematic for **ALU\_BLOX** by pushing into the **ALU\_BLOX** symbol.

1. Select the **ALU\_BLOX** instance on the CALC schematic, if it is not already selected.
2. Select **view** → **Push Into Schematic**. The schematic for **ALU\_BLOX** appears.

A schematic similar to that in Figure 2-1 appears.

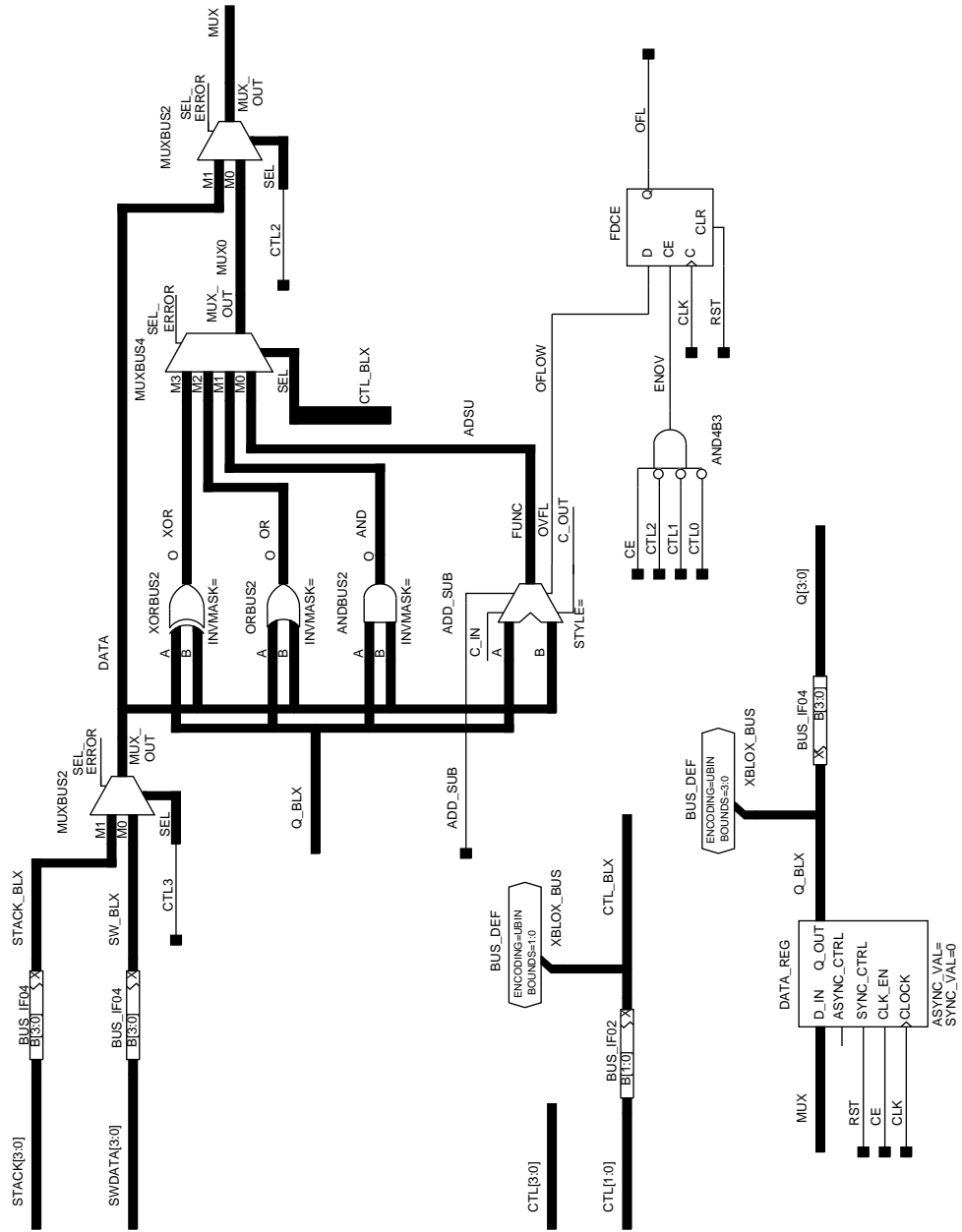


Figure 2-1 ALU\_BLOX Schematic

---

## Completing the ALU\_BLOX Schematic

The schematic on your screen is missing some key X-BLOX elements that you will add by performing the commands in this section.

Complete the ALU\_BLOX schematic using Figure 2-1 and the following steps as a guide.

1. Select **Add** → **Component**.
2. In the Add Component dialog box, select the (**xblox**) entry in the Libraries section. In the Components section of the dialog box, scroll down until you can select the **DATA\_REG.1** symbol, then press **OK**.

*Keyboard Shortcut:* You can add the DATA\_REG component by typing `comp data_reg` on the PROcapture command line.

3. Place the DATA\_REG symbol and connect the MUX and Q\_BLX buses, and the RST, CE, and CLK nets, as shown in Figure 2-1. Be sure to label the buses and nets as shown.
4. Using the same procedure as that in steps 1, 2, and 3, add the X-BLOX BUS\_IF04 component to the right of DATA\_REG, as shown in the figure.
5. Add a BUS\_DEF symbol above and between DATA\_REG and BUS\_IF04, as shown in the figure.
6. Attach the Q\_BLX bus to BUS\_DEF with a bus segment.
7. Attach a dangling bus segment to the BUS\_IF04 pin B[3:0], and label it Q[3:0].

At this point, the ALU\_BLOX schematic appears very similar to the schematic in Figure 2-1. X-BLOX attributes must still be added to complete the schematic.

## Understanding X-BLOX Buses

In Figure 2-1, the rectangular BUS\_IF02 and BUS\_IF04 boxes connecting buses, mostly on the left side of the sheet, are bus interfaces. They interface X-BLOX buses with standard PROcapture buses. An X-BLOX bus is not the same as a bus normally used in PROcapture. 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 OUT[7:0], because the bus pins on X-BLOX symbols do not have indexed names. For example, the input pin of DATA\_REG on the ALU\_BLOX schematic has the unindexed name D\_IN. 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 bus is attached to one of these unindexed bus pins, PROcapture flags the bus as an error. Therefore, BUS\_IF symbols are needed as interfaces between X-BLOX buses and normal Viewlogic buses.

The specific BUS\_IF symbol required depends upon the width of the bus being interfaced. In the ALU\_BLOX schematic, for example, the Viewlogic bus STACK[3:0] is four bits wide and thus is interfaced to an X-BLOX bus using a BUS\_IF04, as shown in Figure 2-2. CTL[1:0], a 2-bit bus, is interfaced using a BUS\_IF02. Only X-BLOX buses should be connected to X-BLOX symbol bus pins. No interface is necessary for individual nets that connect to X-BLOX symbols, such as CTL2 and CTL3 on the two MUXBUS2 symbols.

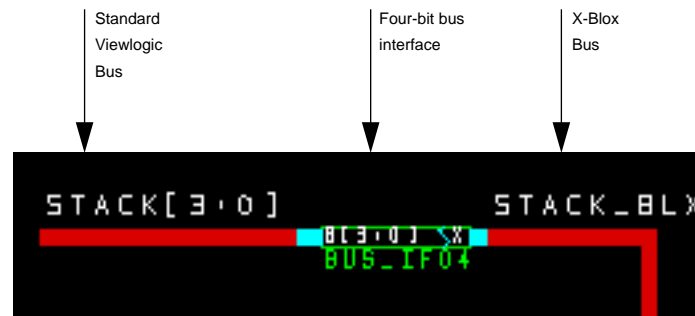


Figure 2-2 Interfacing a Four-Bit Bus to X-BLOX

## 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.

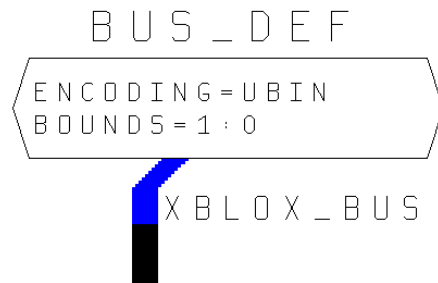
The BOUNDS attribute is placed on a BUS\_DEF to define the width of the bus attached to the BUS\_DEF. In this case, the data path has a width of four bits, giving “3:0” for the value of BOUNDS.

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 (twos 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/subtractor. If you designated a data type of TWO\_COMP, the macro would be implemented as a twos-complement adder/subtractor, 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.

## 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 2-3 gives an example of a BUS\_DEF symbol.



**Figure 2-3 Example of a BUS\_DEF symbol**

1. Select the `BUS_DEF` connected to the `Q_BLX` bus on the lower portion of the page.
2. Select **Change** → **Object Attributes** → **Dialog**.  
The Edit Attributes dialog box appears.
3. By selecting an attribute and pressing the Edit button, add the following Comp Value values to the `BOUNDS` and `ENCODING` attributes, as shown in Table 2-1.

**Table 2-1 BUS\_DEF Attributes**

Attribute Name	Symbol Value	Comp Value
<code>BOUNDS</code>		<code>3:0</code>
<code>ENCODING</code>		<code>UBIN</code>

There are also two other attributes on the symbol: `DEF=BLOX` and `LIBVER`. The `DEF=BLOX` attribute defines this symbol as an X-BLOX symbol, and the `LIBVER` attribute identifies which version of the library part is in use. These two attributes cannot be changed.

4. Click on **OK** within the dialog box.

## Saving Your Changes

Save all of the design changes you have made before continuing with the tutorial.

1. Select **File** → **Save** to save your changes to the ALU\_BLOX schematic.
2. Pop back to the CALC schematic by selecting **View** → **Pop**.
3. Select **File** → **Save** to save the CALC schematic.

## 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 examples show 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 Reference/User Guide* during this discussion.

- DATA\_REG

DATA\_REG in this design has two attributes that you can 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/subtractor, 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, MUXBUS<sub>x</sub>

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, greatly reducing the time necessary to enter the design. The MUXBUS<sub>x</sub> 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, to invert input bit zero on the upper bus connection to the ANDBUS, 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, and 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 therefore 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 be used to represent the functionality of an X-BLOX macro. To illustrate this characteristic of X-BLOX symbols, perform the following steps.

1. Push back into the ALU\_BLOX schematic.
2. Select the DATA\_REG instance on the ALU\_BLOX schematic.
3. View its underlying schematic by pushing into the symbol.



The schematic for this macro is completely blank. The same is true for all X-BLOX macros; when you first create a design using X-BLOX, no information is available even to simulate the design functionally. The schematic page underneath each X-BLOX macro is “filled in” by the X-BLOX synthesis program, which is run by the XSimMake program. XSimMake is invoked by PROflow automatically when you select the Functional Simulation PROsim icon.

## Functional Simulation

The XSimMake program, which you can invoke from PROflow by selecting the Functional Simulation PROsim icon, allows you to easily simulate designs containing X-BLOX components. It coordinates the program execution flow necessary for functional or timing simulation. For more detailed information on XSimMake, refer to the “Functional Simulation” chapter of the *Viewlogic Interface Guide*.

### Creating the Simulation Schematic

From PROflow, use XSimMake to generate a schematic that you can functionally simulate.

1. Minimize the PROcapture window and return to PROflow.
2. Select the Functional Simulation PROsim icon.

The Functional Simulation dialog box appears.

3. Select the **Design Contains XBLOX, RAM, ROM or XABEL Module** check box.
4. Click on **Select Part**.
5. In the Package Selection dialog box, select the **3020APC68-7** part.
6. Select **OK**.
7. Select **OK**.

PROflow now invokes XSimMake. It always produces a new schematic with the same name as the original, with an “s” added to the beginning of the original name. This simulation schematic is placed within a directory beneath the project directory. The directory is given the same name as the simulation design. For the Calc design, XSimMake creates a new directory in the project directory called scalc, and the new directory contains an sch directory with the new SCALC simulation schematic.

**Note:** On DOS/Windows 3.x-based machines, because of the renaming of the simulation schematic, you should not give schematics names that are longer than seven characters. If a design with an eight-character name is given as input to XSimMake, it cannot append the “s” to the beginning of the name and produces an error.

In addition, XSimMake inserts the simulation directory into the viewdraw.ini file so that you can access both the original and simulation schematics from the project directory. For CALC, the following line is added to viewdraw.ini:

```
DIR [w] .\sCALC (sCALC)
```

**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 the ALU\_BLOX module with BLOXSOLN on the CALC schematic, save, and try again.

XSimMake also modifies the simulation schematic so that it is suitable for use in simulation. The names of all unindexed X-BLOX bus pins and buses are changed so that they are indexed. XSimMake notifies you of name changes by changing the color of the modified text to purple. In addition, the name of any symbol whose schematic has been modified also appears in purple text.

Text similar to the following appears as XSimMake processes the design.

**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 : deleting directory scalc  
XSIMMAKE COMMAND : creating directory scalc
```

```
XSIMMAKE COMMAND : creating directory scalc\sch
XSIMMAKE COMMAND : creating directory scalc\sym
XSIMMAKE COMMAND : creating directory scalc\wir
XSIMMAKE COMMAND : creating directory scalc\savexnf
XSIMMAKE COMMAND : creating directory scalc\xbloxxnf
XSIMMAKE COMMAND : creating directory scalc\otherxnf
XSIMMAKE COMMAND : check.exe -p calc
XSIMMAKE COMMAND : wir2xnf.exe -b -v -od scalc\otherxnf calc
calc.xnf -p 3020APC68-7
XSIMMAKE COMMAND : xnfmerge.exe -y -d scalc\otherxnf -q
scalc\otherxnf\calc.xnf scalc\otherxnf\calc.xff
XSIMMAKE COMMAND : xfind.exe scalc\otherxnf\calc.xff calc.xfw
calc.xgs
READING XFW FILE : calc.xfw
XSIMMAKE COMMAND : xnfmerge.exe -z -d scalc\otherxnf -d xnf -
d . scalc\otherxnf\calc.xnf scalc\otherxnf\calc.xff
XSIMMAKE COMMAND : xnfprep.exe scalc\otherxnf\calc.xff
scalc\otherxnf\calc.xtg report= calc.prx ignore_timespec=all
XSIMMAKE COMMAND : xblox.exe scalc\otherxnf\calc.xtg sim=xnf
simdir=scalc\xbloxxnf sim_rerun=true blxfile= calc.blx
XSIMMAKE COMMAND : xfind.exe -o scalc\xbloxxnf\calc.xgs
calc.xfw
XSIMMAKE COMMAND : xnf2wir.exe -b -o calc.xfw
XSIMMAKE COMMAND : xdraw.exe -i calc -o scalc -a scalc
calc.xgs scalc\xbloxxnf\calc.xgs
XSIMMAKE COMMAND : check.exe -p scalc
XSIMMAKE COMMAND : vsm scalc
.
.
0 Errors and 0 Warnings occurred during processing.
```

## **Examining XSimMake Output**

An explanation of the XSimMake functional flow output follows.

```
XSIMMAKE COMMAND : deleting directory scalc
XSIMMAKE COMMAND : creating directory scalc
```

```
XSIMMAKE COMMAND : creating directory scalc\sch
XSIMMAKE COMMAND : creating directory scalc\sym
XSIMMAKE COMMAND : creating directory scalc\wir
XSIMMAKE COMMAND : creating directory scalc\savexnf
XSIMMAKE COMMAND : creating directory scalc\xbloxxnf
XSIMMAKE COMMAND : creating directory scalc\otherxnf
```

First, XSimMake deletes any existing simulation schematic, then creates the directory structure for the new one.

```
XSIMMAKE COMMAND : check -p calc
```

XSimMake then runs the Viewlogic Check program to ensure that the WIR files for the design are up to date.

```
XSIMMAKE COMMAND : wir2xnf.exe -b -v -od scalc\otherxnf calc
calc.xnf -p 3020APC68-7
```

Next, XSimMake runs WIR2XNF to convert the Viewlogic WIR files to standard Xilinx netlist format (XNF) files.

```
XSIMMAKE COMMAND : xnfmerge.exe -y -d scalc\otherxnf -q
scalc\otherxnf\calc.xnf scalc\otherxnf\calc.xff
```

XSimMake runs XNFMerge to merge the XNF files created by WIR2XNF into a single netlist.

```
XSIMMAKE COMMAND : xfind scalc\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.exe -z -d scalc\otherxnf -d xnf -
d . scalc\otherxnf\calc.xnf scalc\otherxnf\calc.xff

XSIMMAKE COMMAND : xnfprep.exe scalc\otherxnf\calc.xff
scalc\otherxnf\calc.xtg report= calc.prx ignore_timespec=all
```

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 flattened XFF file, while XNFPrep verifies that the XFF file is correct.

```
XSIMMAKE COMMAND : xblox.exe scalc\otherxnf\calc.xtg sim=xfnf
simdir=scalc\xbloxxnf sim_rerun=true blxfile= calc.blx
```

X-BLOX is run with the special simulation option `sim=xfnf`, which causes it to produce XNF files for each X-BLOX symbol on the schematic.

```
XSIMMAKE COMMAND : xfind.exe -o scalc\xbloxxnf\calc.xgs
calc.xfw
```

XSimMake runs XFind once again on the output of X-BLOX to determine which files must have simulation models built for them. This information is written into the `calc.xfw` file.

```
XSIMMAKE COMMAND : xnf2wir.exe -b -o calc.xfw
```

XSimMake uses the information in the `calc.xfw` file produced by XFind in the previous step to run XNF2WIR on the appropriate files. It produces WIR files for all of the XNF files generated by X-BLOX, creating simulation models for all of the X-BLOX components in the schematic.

```
XSIMMAKE COMMAND : xdraw.exe -i calc -o scalc -a scalc
calc.xgs scalc\xbloxxnf\calc.xgs
```

Using information found in the `calc.xgs` file produced by X-BLOX, XDraw generates a new schematic called SCALC in the `scalc` directory.

```
XSIMMAKE COMMAND : check.exe -p scalc
```

XSimMake then runs the Viewlogic Check program to verify the validity of the WIR models produced by XNF2WIR.

```
XSIMMAKE COMMAND : vsm scalc
```

Finally, XSimMake runs VSM to generate a simulation file from the WIR models for use in PROsim.

## Performing a Functional Simulation

After XSimMake completes, PROflow invokes Notepad to display the XSimMake log file for review. Once you close Notepad, PROflow invokes PROsim on the `scalc.vsm` simulation network. A simulation command (CMD) file can now be executed on the design.

1. Select **simulate** → **Command File**. At the `File name>` prompt at the bottom of the PROsim window, enter `calc3kaf.cmd`.

2. After the command file has executed, enter the following command at the `PROsim>` prompt:

```
wave scalc.wfm sw alu stack
```

3. Invoke PROwave by pressing the PROwave icon in the Functional Simulation area of the PROflow window. Select `scalc.wfm` as the file to open.

The output of this simulation run is identical to the output of the functional simulation run on the original non-X-BLOX Calc design. For a more detailed inspection of the results of this simulation, refer to the discussion found in the “Functional Simulation” chapter of the *Viewlogic Interface Guide*.

If you wish, you can open the SCALC schematic in PROcapture to view the changes that the XSimMake program has made to the original design.

1. If PROcapture is not already running, invoke it from PROflow.

The original CALC schematic is displayed.

2. Press **File** → **Open** and select the SCALC.1 sheet.

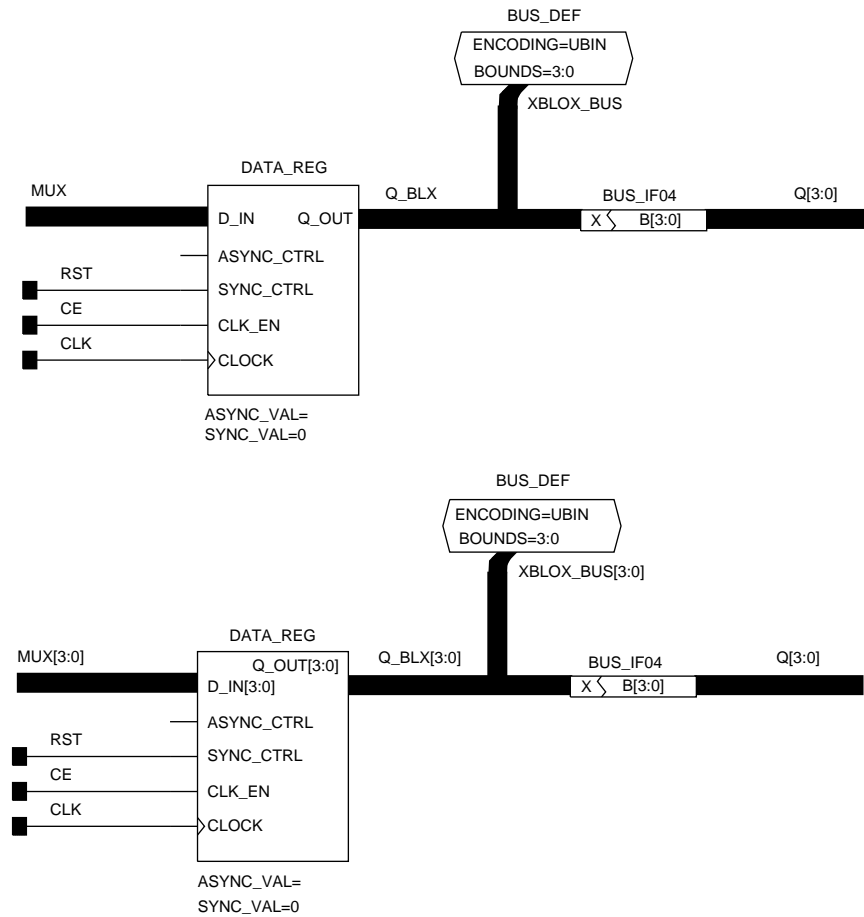
The name of the ALU\_BLOX symbol appears in purple. If anything underneath a symbol is modified, its name appears in purple. Since ALU\_BLOX contains X-BLOX symbols, it had to be redrawn even though the symbol itself was not changed.

3. Push into the ALU\_BLOX schematic.

The X-BLOX symbols have been replaced with equivalents that can be simulated, meaning that all X-BLOX buses and bus pins were modified so that they now have specific widths. In Figure 2-4, for example, note the difference between the DATA\_REG found in the original ALU\_BLOX in the CALC schematic and the one from ALU\_BLOX in the SCALC schematic. The D\_IN pin has become D\_IN[3:0], and the Q\_OUT pin has become Q\_OUT[3:0].

**Note:** The new pin names may be somewhat difficult to read because they tend to overlap on the symbols when they are given indices.

Since these symbols were redrawn, their names appear in purple. In addition, the pin and bus names associated with the X-BLOX symbols were changed, so they also appear in purple.



**Figure 2-4 Original (Top) vs. XSimMake-Generated Schematic**

**Note:** If the X-BLOX symbols do not appear to have been modified, make certain that you have opened the SCALC simulation schematic and not the CALC schematic. Go back and inspect the xsimmake.out, calc.prp, and calc.blx files for errors. Correct any errors found and re-execute XSimMake from PROflow. Check that all attributes, nets, and buses that were added to ALU\_BLOX were spelled correctly. If the problem still cannot be resolved, replace ALU\_BLOX with BLOXSOLN on the CALC schematic, and rerun XSimMake.

## Implementing the Calc Design

The implementation of designs containing X-BLOX components is similar to that of other designs. You can use PROflow's Xilinx Implementation icon to invoke the Xilinx Design Manager, from which you can control the design's implementation in a Xilinx FPGA.

For detailed information on the Xilinx Design Manager, refer to the *Design Manager/Flow Engine Reference/User Guide*.

## Translating the Netlist

Run the Xilinx Design Manager to generate files for device programming.

1. From PROflow, select the Xilinx Implementation icon to start the Xilinx Design Manager.
2. In the Xilinx Design Manager, select **File** → **New Project**.
3. Using the **Browse** button, change to the WIR directory in the tutorial directory and select the CALC.1 sheet.
4. Press **OK**.
5. Change the target family to **XC3000**.
6. Press the **Translate** button.
7. In the Translate Options window, press the **OK** button.

The Design Manager runs the XMake program, which converts the PROcapture schematic into an XFF netlist.

## Examining XMake Netlist Translation Output

XMake produces a screen output similar to the following.

```
XMAKE: Generating makefile 'calc.mak'...
XMAKE: Profile used is 'xdm.pro'.
XMAKE: Execute command 'wir2xnf -B -OD xnf calc calc.xnf'.
XMAKE: Set the part type to '3020APC68-7' from
'xnf\calc.xnf'.
XMAKE: Running with the following XMAKE options:
>>> XDELAY is run always with '-D' and '-W' options by
```



```
XMAKE.  
XMAKE: Makefile saved in 'calc.mak'.  
XMAKE: Making 'calc.xff'...  
XMAKE: Execute command 'xnfmerge -A -D xnf -D . -P 3020APC68-  
7 xnf\calc.xnf calc.xff'.  
XMAKE: 'calc.xff' has been made.
```

An explanation of the output follows.

```
XMAKE: Generating makefile 'calc.mak'...  
XMAKE: Profile used is 'xdm.pro'.  
XMAKE: Execute command 'wir2xnf -B -OD xnf calc calc.xnf'.
```

XMake runs WIR2XNF to convert the Viewlogic WIR file netlists to Xilinx XNF files.

```
XMAKE: Set the part type to '3020APC68-7' from  
'xnf\calc.xnf'.  
XMAKE: Running with the following XMAKE options:  
    >>> 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.xff'...  
XMAKE: Execute command 'xnfmerge -A -D xnf -D . -P 3020APC68-  
7 xnf\calc.xnf calc.xff'.  
XMAKE: 'calc.xff' has been made.
```

Next, XMake runs XNFMerge to flatten the hierarchical XNF files into a single netlist, which is written out as an XFF file.

## Creating a Routed Design

Use the Xilinx Design Manager to implement the converted netlist in the target device.

1. Select **Design** → **Implement**.
2. In the XC3000A Design Implementation dialog box, confirm that the **Produce Configuration Data** check box is selected.

3. Select the **Produce Timing Simulation Data** check box.  
(The timing data will be used in a later part of the tutorial.)
4. Press **Run**.

The Xilinx Flow Engine, invoked by the Design Manager, optimizes, maps, places, and routes the design, and creates a bitstream file that can be downloaded to the part. The resulting Flow Engine history file, `program.his`, is shown following.

```
xnfprep design=calc.xff outfile=calc.xtg savesig=false
partType=3020APC68-7 ignore_timespec=none xblox_prep=true
split_report=true

xblox calc.xtg calc.xg parttype=3020APC68-7 archopt=false
mergeio=false reg_rlocs=false

xnfprep design=calc.xg outfile=calc.xtf savesig=false
ignore_timespec=none split_report=true

xnfmap calc.xtf calc.map

ppr design=calc placer_effort=2 router_effort=2
ignore_timespec=none path_timing=true route_thru_bufg=ok
route_thru_blks=ok guide_blks=all lock_routing=whole_sigs
split_report=true

xdelay -d -w calc.lca

lca2xnf -g calc.lca calc.xnf

xnfba calc.xg calc.xnf

makebits -l -mbo=calc.mbo calc.lca
```

## Examining the Flow Engine History File

An explanation of the history file follows.

```
xnfprep design=calc.xff outfile=calc.xtg savesig=false
partType=3020APC68-7 ignore_timespec=none xblox_prep=true
split_report=true
```

The Flow Engine runs XNFPrep to verify that the flattened XFF file is correct. It creates a report that is stored in the `calc.prp` file. The output of XNFPrep is specified as `calc.xtg`.

```
xblox calc.xtg calc.xg parttype=3020APC68-7 archopt=false
mergeio=false reg_rlocs=false
```

The Flow Engine runs X-BLOX to synthesize the X-BLOX symbols in the design into standard logic.

```
xnfpref design=calc.xg outfile=calc.xtf savesig=false
ignore_timespec=none split_report=true
```

It runs XNFPrep once again to verify that the logic produced by X-BLOX is correct. In this case, the output file name is specified as `calc.xtf`.

```
xnfmap calc.xtf calc.map
```

XNFMAP partitions 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.

```
ppr design=calc placer_effort=2 router_effort=2
ignore_timespec=none path_timing=true route_thru_bufg=ok
route_thru_blks=ok guide_blks=all lock_routing=whole_sigs
split_report=true
```

The Flow Engine then runs PPR to place the partitioned 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.

```
xdelay -d -w calc.lca
```

XDelay writes delay information into the LCA file.

```
lca2xnf -g calc.lca calc.xnf
```

LCA2XNF converts the LCA file, which contains the delay information, back to an XNF file.

```
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.

```
makebits -l -mbo=calc.mbo calc.lca
```

Since you selected the Produce Configuration Data check box, the Flow Engine runs the MakeBits program to create a bitstream that can be downloaded to the part.

## Timing Simulation

You have already performed many of the steps necessary for timing simulation. The schematic created in functional simulation is referenced in timing simulation as well, so it is not necessary to generate a new schematic. The calc.xnf 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 information found in the XNF file to PROsim. The XSimMake utility, which is invoked from PROflow, performs this task.

### Creating the Simulation Network

Run XSimMake from PROflow to generate a network that can be used for timing simulation.

1. Select the Timing Simulation PROsim icon.
2. In the Timing Simulation dialog box, make sure that the **Execute Power On Reset** and **Execute Netlister** option boxes are checked, and press **OK**.

As XSimMake runs, text similar to the following appears in the window.

```
XSIMMAKE COMMAND : check -p scalc
XSIMMAKE COMMAND : vsm scalc
XSIMMAKE COMMAND : xnf2wir -b xnfba wir\xnfba
XSIMMAKE COMMAND : vsm xnfba
XSIMMAKE COMMAND : vsmupd -b scalc xnfba -x xnfba.xnf -o
scalc
```

### Examining XSimMake Output

The steps just performed generate simulation VSM files from the simulation schematic and from the back-annotated netlist. Similar to the function performed by XNFBA, these two files are then read by VSMUPD, which tries to make the back-annotated simulation file resemble the simulation schematic as closely as possible. This step ensures that as many as possible of the net and instance names from the original schematic are usable during timing simulation.

## Performing a Timing Simulation

After XSimMake completes, PROflow invokes Notepad to display the XSimMake log file for review. Once you close Notepad, PROflow invokes PROsim on the `scal3.vsm` simulation network, which now contains timing information. You can submit a simulation command (CMD) file to be executed on the design.

1. Select **Simulate** → **Command File**.
2. At the `File name>` prompt at the bottom of the PROsim window, enter `calc3kat.cmd`.
3. After the command file has executed, invoke PROwave by pressing the PROwave icon in the Timing Simulation area of the PROflow window.
4. Select `calct.wfm` as the file to open.

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. It differs slightly, because different place and route iterations produce different timing. For a more detailed inspection of the results of this simulation, refer to the “Timing Simulation” chapter of the *Viewlogic Interface Guide*.

## 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 “Downloading an FPGA Design” section of the “PROcapture and PROsim Tutorial” chapter for more information.

## Further Reading

Before beginning an X-BLOX design, you should read the descriptions of the X-BLOX macros found in the *X-BLOX Reference/User Guide* in order to understand the abilities and limitations of each macro. You should also review the section on the X-BLOX program itself, found in the same manual.



# ***Viewlogic Tutorials***

***Xilinx ABEL Tutorial***





# 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). You can then include these logic blocks as part of a larger design, allowing the same design to contain logic defined by both graphical and text-based entry.

This chapter gives a practical example using Xilinx ABEL within the Viewlogic PRO Series 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 “PROcapture and PROsim Tutorial” chapter of this manual. If not, please review that chapter before continuing.

## Required Software

You should have access to the following software:

- Viewlogic PROcapture, the Viewlogic schematic entry tool
- Viewlogic PROsim, the Viewlogic simulation tool
- XACT Core Implementation Tools, version 6.0 or later
- Xilinx ABEL (DS-371), which is included in the Standard (DS-VLS-STD-PC1) and Extended (DS-VSL-EXT-PC1) packages. The Base package does not include Xilinx ABEL, but you can purchase it separately.

You should have at least temporary access to all of the software just listed using the temporary licensing available on the programmable key, provided that the temporary licensing has not already been exhausted.

## Preparing the Design

This tutorial uses the completed Calc design, which you can create either by completing the PROcapture tutorial or by copying a completed design from one of the solutions directories.

**Note:** All of the screen outputs refer to the processing of the XC3000A solutions design on a PC. Other parts or platforms have different outputs.

A full solution for the PROcapture tutorial is supplied in the solution directory located under the directory where the PROseries software was installed:

```
...\tutorial\vwlogic\procalc\calc3ka\soln_3ka
```

1. Using the Windows File Manager, copy the contents of this directory to the directory where you will be performing the Xilinx ABEL tutorial.
2. Invoke Xilinx PROflow and select the Design Entry icon.
3. Select the **Project Manager** button.
4. Press the **Create** button.
5. Double-click on the directory that you made in step 1 in the Create Project dialog box.
6. Verify that the directory is shown in the Directory text box, and press **OK**.

The PROJman Create dialog box appears.

7. Select **No** to re-initialize the viewdraw.ini file.

**Note:** This step assumes that the default viewdraw.ini file found in ... \proser\standard is configured correctly for PRO Series installation.

8. Click on **Exit** in the PRO Series Project Manager.
9. In the Select Family dialog box, select **XC3000A** and click on **OK**.

10. After exiting from the Select Family box, select the **CALC.1** design in the Design Entry dialog box, and select **OK** to open the schematic.

## 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 STATMACH, so this substitution in no way alters the function of the CALC design.

**Note:** For more information on the function of the Calc design, refer to the discussion in the “Design Description” section of the “PROcapture and PROsim Tutorial” chapter.

Stat\_abl.abl is the name of the Xilinx ABEL ABEL-HDL (ABL) 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. From PROflow, select the Design Entry icon.
2. In the Design Entry dialog box, select the **X-ABEL** button.  
The STAT-ABL.ABL design is automatically selected.
3. Press the **OK** button.

The file displayed in Figure 3-1 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_ADDRpin;

"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 EXCOP ]->[ PUSH,WE,OTHER,ADD_SUB,RST,CE_ALU,CE_ADD, R
CTL])

"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 3-1 Stat\_abl.abl File**

A breakdown of the contents of the Xilinx ABEL ABEL-HDL 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 Initialstate 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 CLK, EXC, and OP vectors. 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 now verifies the STAT\_ABL design, using the test vectors just described as input.

1. Hold down the **Alt** key and type **c** to select the Compile menu.
2. Select **Simulate Equations**.

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 ABEL-HDL file could be compiled within Xilinx ABEL using the `Compile` → `Xilinx FPGA Netlist` command. Instead, perform this step from the command line using a program called `ABL2XNF`. `ABL2XNF` performs three functions: compilation of the ABEL file, generation of an XSF file that is used to create a Viewlogic symbol for the Xilinx ABEL design, and generation of an XAS simulation netlist file.

1. Exit the Xilinx ABEL application.
2. Access DOS from Windows.
3. From the design directory, execute `ABL2XNF` by using the following syntax:

```
ABL2XNF STAT_ABL.ABL↵
```

`ABL2XNF` compiles the ABEL 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. The SymWin (SymGen for Windows) 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. SymWin uses this file to generate an appropriate symbol.

1. Invoke the Symbol Generation Utility by clicking on the Symbol Generation Utility icon in the Program Manager XACTstep program group. (The program name is symwin.exe.)

The Symbol Generator dialog box appears, as shown in Figure 3-2.

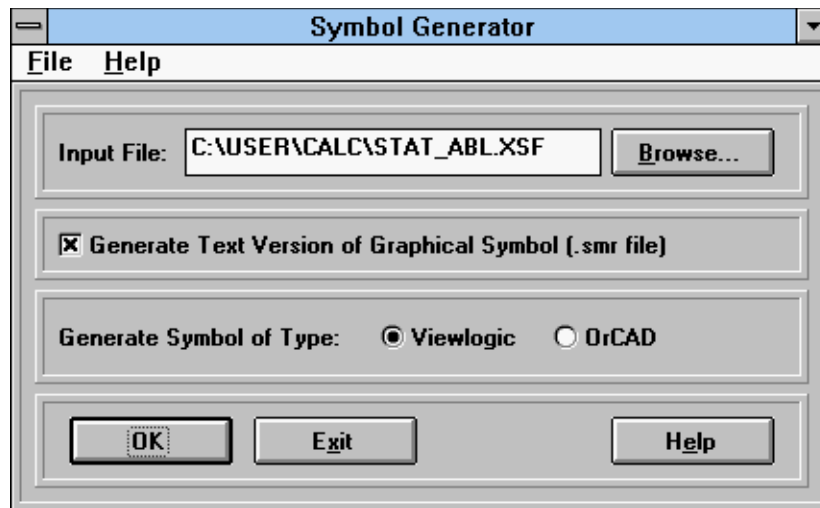


Figure 3-2 Symbol Generator Dialog Box

2. Using the Browse button, go to the design directory, select the STAT\_ABL.xsf file, and press OK.
3. Make sure that **viewlogic** is selected in the Generate Symbol of Type field.
4. Press the OK button.

After SymWin finishes creating the symbol, a report is displayed in the SymGen Results window.

5. After viewing the report, press any key to close the SymGen Results window.

SymWin creates a Viewlogic symbol file called `stat_abl.1` and places it in the sym directory. The symbol is shown in Figure 3-3.

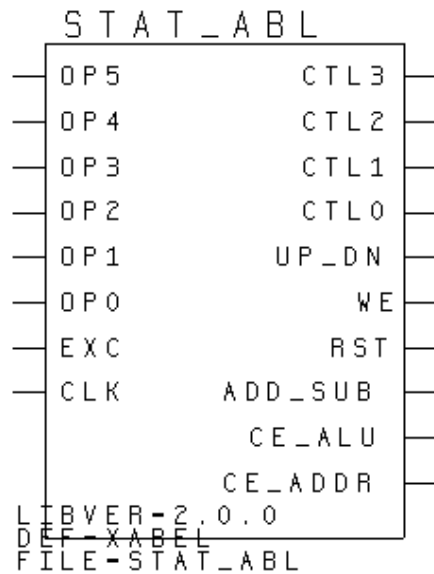


Figure 3-3 STAT\_ABL Symbol

## 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 `STAT-MACH` state machine.

1. From PROflow, select the Design Entry icon.
2. In the Design Entry dialog box, click on **Schematic Design Type**.
3. From the file list, select `CALC.1`.

4. Press **OK** to open the schematic.

**Note:** If the text layers are not visible in PROcapture, select the **Change** → **PROcapture Colors** option from the PROcapture menus, and select the **Classic Defaults** button. You will have to close and then re-open the schematic for the new color scheme to take effect.

5. Select the CONTROL instance on the CALC schematic.

6. Select **View** → **Push Into Schematic**.

The schematic for CONTROL is displayed.

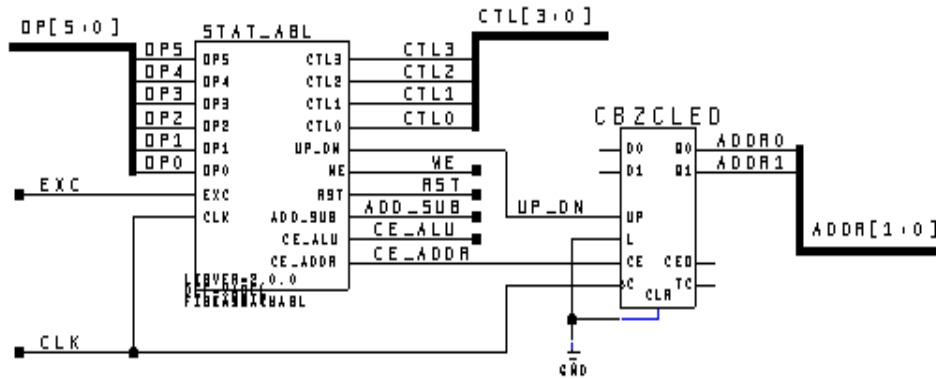
7. Select the STATMACH instance on the CONTROL schematic.

8. Select **Change** → **Component**.

9. Select STAT\_ABL.1 from the Components browse window of the Change Component dialog box.

This procedure replaces the original STATMACH block with the functionally equivalent Xilinx ABEL module called STAT\_ABL. This change is reflected by the appearance of the name STAT\_ABL at the top of the symbol, as shown in Figure 3-4.

10. Select **File** → **Save** to save the change.



<b>XILINX</b> <sup>TM</sup> Calc Tutorial Design	
Title: CONTROL	
Reference: Control Logic	
Date: August 23, 1993	Applications Group

Figure 3-4 CONTROL Schematic with STAT\_ABL

## Viewing the STAT\_ABL Symbol

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 Viewlogic libraries.

### Viewing the STAT\_ABL Schematic

View the schematic for STAT\_ABL by pushing into the symbol.

The STAT\_ABL symbol on the CONTROL schematic is already selected.

1. Select **View** → **Push Into Schematic**.

A blank schematic page appears. The logic description for the STAT\_ABL block is not defined inside PROcapture but by the

netlist in the stat\_abl.xnf file, which was generated earlier from the Xilinx ABEL code.

2. Select **View** → **Pop** to return to the CONTROL schematic.

### Verifying the Symbol Type

Since the schematic does not contain the logic description, you must pass two important pieces of information to the translation programs: that the logic description does not exist on the underlying schematic of STAT\_ABL and that the logic description does exist in a file elsewhere.

In Viewlogic, there are two commonly used types of symbols: composite symbols, which have underlying schematics, and module symbols, which do not. You can change the symbol type by editing the symbol and selecting **Change** → **Block** → **Type**. Both the Viewlogic programs and the Xilinx translation programs recognize that a symbol of type Module does not have an underlying schematic.

Check to be sure that the symbol generated by the SymGen macro is of the proper type. (This step is not necessary when processing your own designs. The steps in this section are included only to familiarize you with Xilinx ABEL symbols.)

The STAT\_ABL symbol on the CONTROL schematic is already selected.

1. Select the STAT\_ABL symbol and choose **Info** → **Object Detail**.

A window containing information about the object appears. One line in the window reads as follows:

```
Block: STAT_ABL .(M)
```

(You may have to scroll down to see this line.)

The letter in parentheses after the name of the symbol is “M,” which specifies that the symbol is of type Module.

2. Press the **OK** button to dismiss the window.

## Verifying the Symbol Attributes

You must tell the Xilinx programs where to find the logic description for a Module-type symbol by attaching the FILE attribute to the symbol. 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, automatically added by SymGen, is DEF=XABEL.

Verify that the macro created by SymGen placed the appropriate FILE and DEF attributes on the macro. (This step is not necessary when processing your own designs. The steps in this section are included only to familiarize you with Xilinx ABEL symbols.)

1. The STAT\_ABL symbol on the CONTROL schematic sheet is already selected.
2. Select **View** → **Push Into Symbol**.
3. Without selecting anything on the symbol, choose **Change** → **Object Attributes** → **Dialog**.

A dialog box displaying the symbol attributes appears, as shown in Figure 3-5.

4. Select **Cancel** to close the dialog box, and then exit PROcapture.



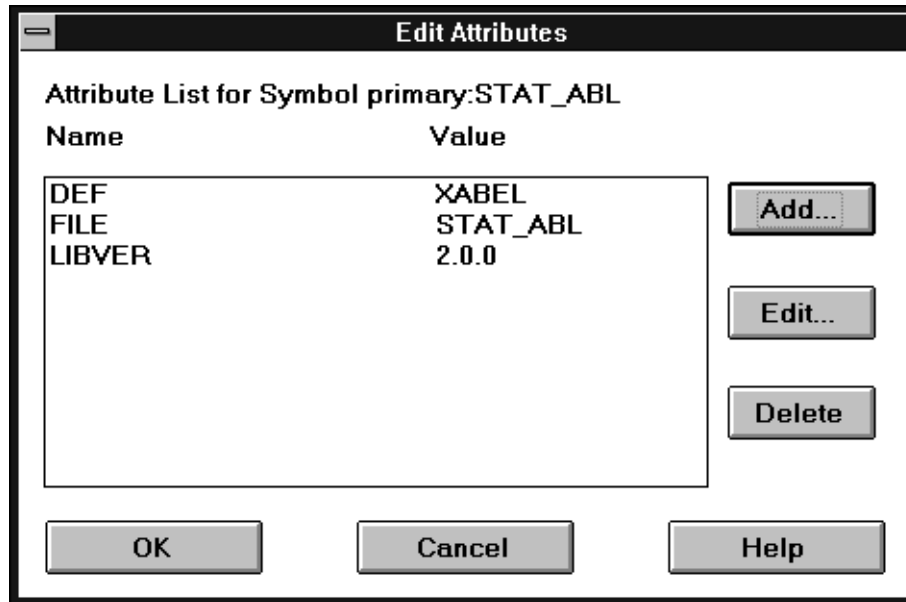


Figure 3-5 File Attributes for STAT\_ABL

## Functional Simulation

As shown earlier, there is no schematic representation of the logic for the STAT\_ABL symbol. The logic is within an XNF file, which PROsim cannot access, so you must partially translate the design to merge the logic for STAT\_ABL with the rest of the schematic to enable the design to be simulated.

The XSimMake program, which you can invoke from PROflow by selecting the Functional Simulation PROsim icon, allows you to easily simulate designs containing Xilinx ABEL components. It coordinates the program execution flow necessary for functional or timing simulation.

For more detailed information on XSimMake, refer to the “Manual Translation” chapter of the *Viewlogic Interface Guide*.

Because of the presence of the Xilinx ABEL module in the design, some design translation is necessary even for functional simulation. Preparing a design containing a Xilinx ABEL block for functional

simulation simply requires the invocation of XSimMake from PROflow.

## Creating the Simulation Schematic

From PROflow, use XSimMake to generate a schematic that you can functionally simulate.

1. Minimize the PROcapture window and return to PROflow.
2. Select the Functional Simulation PROsim icon.

The Functional Simulation dialog box appears.

3. Select the **Design Contains XBLOX, RAM, ROM or XABEL Module** check box.
4. Click on **Select Part**.
5. In the Package Selection dialog box, select the **3020APC68-7** part.
6. Select **OK**.
7. Select **OK**.

PROflow now invokes XSimMake. It always produces a new schematic with the same name as the original, with an “s” added to the beginning of the original name. This simulation schematic is placed within a directory beneath the project directory. The directory is given the same name as the simulation design. For the Calc design, XSimMake creates a new directory in the project directory called scalc, and the new directory contains an sch directory with the new SCALC simulation schematic.

**Note:** On DOS/Windows 3.x-based machines, because of the renaming of the simulation schematic, you should not use schematic names that are longer than seven characters. If a design with an eight-character name is given as input to XSimMake, it cannot append the “s” to the beginning of the name and produces an error.

In addition, XSimMake inserts the simulation directory into the viewdraw.ini file so that you can access both the original and simulation schematics from the project directory. For CALC, the following line is added to the viewdraw.ini file:

```
DIR [w] .\sCALC (sCALC)
```

As XSimMake runs, text similar to the following appears in the 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 : deleting directory scalc
XSIMMAKE COMMAND : creating directory scalc
XSIMMAKE COMMAND : creating directory scalc\sch
XSIMMAKE COMMAND : creating directory scalc\sym
XSIMMAKE COMMAND : creating directory scalc\wir
XSIMMAKE COMMAND : creating directory scalc\savexnf
XSIMMAKE COMMAND : creating directory scalc\xbloxxnf
XSIMMAKE COMMAND : creating directory scalc\otherxnf
XSIMMAKE COMMAND : check.exe -p CALC
XSIMMAKE COMMAND : wir2xnf.exe -b -v -od sCALC\otherxnf CALC
CALC.xnf -p 3020APC68-7
XSIMMAKE COMMAND : xnfmerge.exe -y -d sCALC\otherxnf -q
sCALC\otherxnf\CALC.xnf sCALC\otherxnf\CALC.xff
XSIMMAKE COMMAND : xfind.exe sCALC\otherxnf\CALC.xff CALC.xfw
CALC.xgs
XSIMMAKE COMMAND : abl2xnf.exe stat_abl.abl output_directory=
sCALC\otherxnf parttype=3020APC68-7
XSIMMAKE COMMAND : xnf2wir.exe -b sCALC\otherxnf\stat_abl
sCALC\wir\stat_abl
XSIMMAKE COMMAND : xdraw.exe -i CALC -o sCALC -a sCALC
CALC.xgs
XSIMMAKE COMMAND : check.exe -p sCALC
XSIMMAKE COMMAND : vsm.exe sCALC
```

## Examining XSimMake Output

An explanation of the XSimMake functional flow output follows. While it is not necessary to know anything about how XSimMake works, it sometimes gives useful perspective to have some idea of how it prepares the design.

```
XSIMMAKE COMMAND : deleting directory scalc
```

```
XSIMMAKE COMMAND : creating directory scalc
XSIMMAKE COMMAND : creating directory scalc\sch
XSIMMAKE COMMAND : creating directory scalc\sym
XSIMMAKE COMMAND : creating directory scalc\wir
XSIMMAKE COMMAND : creating directory scalc\savexnf
XSIMMAKE COMMAND : creating directory scalc\xbloxxnf
XSIMMAKE COMMAND : creating directory scalc\otherxnf
```

First, XSimMake deletes any existing simulation schematic, then creates the directory structure for the new one.

```
XSIMMAKE COMMAND : check -p CALC
```

It runs the Viewlogic Check program to ensure that the Viewlogic WIR files for the design are up to date.

```
XSIMMAKE COMMAND : wir2xnf.exe -b -v -od sCALC\otherxnf CALC
CALC.xnf -p 3020APC68-7
```

Next, XSimMake runs WIR2XNF to convert the Viewlogic WIR files to standard Xilinx netlist format (XNF) files.

```
XSIMMAKE COMMAND : xnfmerge.exe -y -d sCALC\otherxnf -q
sCALC\otherxnf\CALC.xnf sCALC\otherxnf\CALC.xff
```

XSimMake runs XNFMerge to merge the XNF files created by WIR2XNF into a single netlist.

```
XSIMMAKE COMMAND : xfind.exe sCALC\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. In addition, XFind generates a file called calc.xgs, which contains instructions on how to redraw the STAT\_ABL symbol in the simulation schematic so that it can be used in simulation.

The program flow is changed by XFind so that a schematic that can be simulated is created from a schematic containing Xilinx ABEL modules.

```
XSIMMAKE COMMAND : xnf2wir.exe -b sCALC\otherxnf\stat_abl
sCALC\wir\stat_abl
```

XSimMake runs XNF2WIR on the XNF file representation of STAT\_ABL to generate a simulation model for the STAT\_ABL symbol on the schematic.

```
XSIMMAKE COMMAND : xdraw.exe -i CALC -o sCALC -a sCALC
CALC.xgs
```

XSimMake then runs XDraw to generate the new simulation schematic, SCALC, using the information found in the calc.xgs file and the original schematic. The STAT\_ABL symbol type on the simulation schematic is changed from Module to Composite. As noted earlier, this change enables a simulation model to be placed underneath the symbol. Otherwise, the schematic appears identical to the original in all respects.

```
XSIMMAKE COMMAND : check.exe -p sCALC
```

XSimMake then runs the Viewlogic Check program to verify the validity of the WIR model produced by XNF2WIR.

```
XSIMMAKE COMMAND : vsm.exe sCALC
```

Finally, XSimMake runs VSM to generate a simulation file from the WIR models for use in PROsim.

## Performing a Functional Simulation

After XSimMake completes, PROflow invokes Notepad to display the XSimMake log file for review. Once you close Notepad, PROflow invokes PROsim on the scalc.vsm simulation network. A simulation command (CMD) file can now be executed on the design.

1. Select **Simulate** → **Command File**.
2. At the **File name>** prompt at the bottom of the PROsim window, enter **calc3kaf.cmd**.
3. After the command file has executed, enter the following command at the **PROsim>** prompt:

```
wave scalc.wfm sw alu stack
```
4. Invoke PROwave by selecting the PROwave icon in the Functional Simulation area of the PROflow window.
5. Select **scalc.wfm** as the file to open.

The output of this simulation run is identical to the output of the functional simulation run on the original Calc design before the addition of the Xilinx ABEL module. All nodes and vectors on the SCALC simulation schematic have values back-annotated to them.

A schematic was not generated for the logic underneath the STAT\_ABL symbol, only a simulation WIR file. Because there is no schematic, although you can probe the logic for STAT\_ABL in the simulator, you cannot push into the STAT\_ABL symbol to see back-annotated simulation values. This is not normally a problem, however, since the logic within the Xilinx ABEL module has been verified using the Xilinx ABEL simulator. Since you totally described and verified its behavior within Xilinx ABEL, there should be no reason to view points within the STAT\_ABL symbol.

For a more detailed inspection of the results of this simulation, refer to the discussion found in the “Functional Simulation” chapter of the *Viewlogic Interface Guide*.

## Implementing the CALC Design

The translation of designs containing Xilinx ABEL blocks is similar to the translation of other designs. You can use PROflow’s Xilinx Implementation icon to invoke the Xilinx Design Manager. From here, you will control the design’s implementation in a Xilinx FPGA, 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 performed.

For detailed information on the Xilinx Design Manager, refer to the *Design Manager/Flow Engine Reference/User Guide*.

## Translating the Netlist

Run the Xilinx Design Manager to generate files for device programming.

1. From PROflow, press the Xilinx Implementation icon to start the Xilinx Design Manager.
2. In the Xilinx Design Manager, select **File** → **New Project**.

3. Using the **Browse** button, change to the WIR directory in the tutorial directory and select the **CALC.1** sheet.
4. Press **OK**.
5. Change the Target Family to **XC3000**.
6. Press the **Translate** button.
7. In the Translate Options window, press the **OK** button.

The Design Manager runs the XMake program, which converts the PROcapture schematic into an XFF netlist.

## Examining XMake Netlist Translation Output

XMake produces a screen output similar to the following.

```
XMAKE Version Beta-5.2.0b
XMAKE: Generating makefile 'calc.mak'...
XMAKE: Profile used is 'xdm.pro'.
XMAKE: Execute command 'wir2xnf -B -OD xnf calc calc.xnf'.
XMAKE: Set the part type to '3020APC68-7' from
'xnf\calc.xnf'.
XMAKE: Running with the following XMAKE options:
XMAKE: Execute command 'abl2xnf stat_abl.abl
output_directory=xnf family=XC3000A parttype=3020APC68-7'.
>>> XDELAY is run always with '-D' and '-W' options by
XMAKE.
XMAKE: Makefile saved in 'calc.mak'.
XMAKE: Making 'calc.xff'...
XMAKE: Execute command 'xnfmerge -A -D xnf -D . -P 3020APC68-
7 xnf\calc.xnf calc.xff'.
XMAKE: 'calc.xff' has been made.
```

An explanation of the output follows.

```
XMAKE: Generating makefile 'calc.mak'...
XMAKE: Profile used is 'xdm.pro'.
XMAKE: Execute command 'wir2xnf -B -OD xnf calc calc.xnf'.
```

XMake runs WIR2XNF to convert the Viewlogic WIR file netlists to Xilinx XNF files.

```
XMAKE: Set the part type to '3020APC68-7' from
'xnf\calc.xnf'.

XMAKE: Running with the following XMAKE options:

XMAKE: Execute command 'abl2xnf stat_abl.abl
output_directory=xnf family=XC3000A parttype=3020APC68-7'.

>>> 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.xff'...

XMAKE: Execute command 'xnfmerge -A -D xnf -D . -P 3020APC68-
7 xnf\calc.xnf calc.xff'.

XMAKE: 'calc.xff' has been made.
```

Next, XMake runs XNFMerge to flatten the hierarchical XNF files into a single netlist, which is written out as an XFF file.

## Creating a Routed Design

Use the Xilinx Design Manager to implement the converted netlist in the target device.

1. Select **Design** → **Implement**.
2. In the XC3000A Design Implementation dialog box, confirm that the **Produce Configuration Data** check box is selected.
3. Select the **Produce Timing Simulation Data** check box. (The timing data will be used in a later part of the tutorial)
4. Press **Run**.

The Xilinx Flow Engine, invoked by the Design Manager, optimizes, maps, places, and routes the design, and creates a bitstream file that can be downloaded to the part. The resulting Flow Engine history file, program.his, is shown following.

```
xnfprep design=calc.xff outfile=calc.xtf savesig=false
parttype=3020APC68-7 ignore_timespec=none split_report=true
```



```

xnfmap  calc.xtf calc.map

ppr design=calc placer_effort=2 router_effort=2
ignore_timespec=none path_timing=true route_thru_bufg=ok
route_thru_blks=ok guide_blks=all lock_routing=whole_sigs
split_report=true

xdelay -d -w calc.lca

lca2xnf -g calc.lca calc.xnf

xnfba calc.xff calc.xnf

xcopy c:\xabeltut\project\ver1\rev1\xnfba.xnf /y

makebits -l -mbo=calc.mbo calc.lca

```

## Examining the Flow Engine History File

An explanation of the history file follows.

```

xnfprep design=calc.xff outfile=calc.xtf savesig=false
parttype=3020APC68-7 ignore_timespec=none split_report=true

```

The Flow Engine runs XNFPrep to verify that the flattened XFF file is correct. It creates a report that is stored in the calc.prp file. The output of XNFPrep is specified as calc.xtf.

```

xnfmap  calc.xtf calc.map

```

XNFMAP partitions 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.

```

ppr design=calc placer_effort=2 router_effort=2
ignore_timespec=none path_timing=true route_thru_bufg=ok
route_thru_blks=ok guide_blks=all lock_routing=whole_sigs
split_report=true

```

The Flow Engine then runs PPR to place the partitioned 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.

```

xdelay -d -w calc.lca

```

XDelay writes delay information into the LCA file.

```

lca2xnf -g calc.lca calc.xnf

```

LCA2XNF converts the LCA file, which contains the delay information, back to an XNF file.

```
xnfbfa calc.xff 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 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.

```
makebits -l -mbo=calc.mbo calc.lca
```

Since the Produce Configuration Data check box was selected, the Flow Engine runs the MakeBits program to create a bitstream that can be downloaded to the part.

## Timing Simulation

Preparing a design containing Xilinx ABEL-based components for timing simulation is similar to the method used for functional simulation. XSimMake is used again, this time to create files suitable for use in timing simulation. It uses timing information found in the calc.lca file generated earlier by XMake to add delays to the simulation netlist. Simulation values are back-annotated to the schematic created in the functional simulation flow. The schematic looks identical to the original, but the simulation results reflect the addition of actual delay values.

### Creating the Simulation Netlist

Run XSimMake from PROflow to generate a netlist that can be used for timing simulation.

1. Select the Timing Simulation PROsim icon.
2. In the Timing Simulation dialog box, make sure that the **Execute Power On Reset** and **Execute Netlister** option boxes are checked, and press **OK**.

As XSimMake runs, text similar to the following appears in the window.

```
XSIMMAKE COMMAND : check -p scalc
XSIMMAKE COMMAND : vsm scalc
XSIMMAKE COMMAND : xnf2wir -b xnfba wir\xnfba
XSIMMAKE COMMAND : vsm xnfba
XSIMMAKE COMMAND : vsmupd -b scalc xnfba -x xnfba.xnf -o
scalc
```

## Examining XSimMake Output

The steps just performed generate simulation VSM files from the simulation schematic and from the back-annotated netlist. Similar to the function performed by XNFBA, these two files are then read by VSMUPD, which tries to make the back-annotated simulation file resemble the simulation schematic as closely as possible. This step ensures that as many as possible of the net and instance names from the original schematic are usable during timing simulation.

## Performing a Timing Simulation

After XSimMake completes, PROflow invokes Notepad to display the XSimMake log file for review. Once you close Notepad, PROflow invokes PROsim on the scalc.vsm simulation network, which now contains timing information. You can submit a simulation command (CMD) file to be executed on the design.

1. Select **Simulate** → **Command File**.
2. At the **File name>** prompt at the bottom of the PROsim window, enter **calc3kat.cmd**.
3. After the command file has executed, invoke PROwave by pressing the PROwave icon in the Timing Simulation area of the PROflow window.
4. Select **calc.ct.wfm** as the file to open.

The output of this simulation run is similar to the output of the timing simulation run on the original (non-Xilinx ABEL) Calc design. It differs slightly, because different mapping, placing, and routing runs produce different timing. For a more detailed inspection of the results of this simulation, refer to the “Timing Simulation” chapter of the *Viewlogic Interface Guide*.

## **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 “Downloading an FPGA Design” section of the “PROcapture and PROsim Tutorial” chapter for more information.

## **Further Reading**

This tutorial shows you the basic functions involved in including Xilinx ABEL-based components in a PROcapture design. Before attempting to build your own Xilinx ABEL design, please review the *Xilinx ABEL User Guide*.

# ***Viewlogic Tutorials***

## ***XACT-Performance and Timing Analyzer Tutorial***



# XACT-Performance and Timing Analyzer 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, placing, and routing of the design.

The Timing Analyzer is the companion tool that allows you to obtain exact timing information about the routed design created by PPR. When you use XACT-Performance, verify the path timing with the Timing Analyzer program. To reduce run time, XACT-Performance does not use the highest possible level of accuracy in computing delays. The Timing Analyzer reports completely accurate worst-case delays for all Xilinx FPGAs. Differences between the two reports are minor, but when they occur, use the Timing Analyzer output as the definitive source for timing information.

**Note:** Since APR does not interpret XACT-Performance specifications, only XC3000A/L, XC3100A, XC4000, and XC5200 family designs can take advantage of the features described in this tutorial. XACT-Performance does not function on XC3000, XC3100, or XC2000 family designs.

This tutorial gives a practical example using XACT-Performance and the Timing Analyzer within the Viewlogic design environment. It is not intended to fully explain all of the functionality found within XACT-Performance or the Timing Analyzer. 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 “PROcapture and PROsim Tutorial” chapter of this manual. If not, please review that chapter before continuing.

### Required Software

You should have at access to the following software:

- Viewlogic PROcapture, the Viewlogic schematic entry tool
- Xilinx Design Manager and the placing and routing tools that are contained in the FPGA Core Tools Package

You should have at least temporary access to all of the software just listed using the temporary licensing available on the programmable key, provided that the temporary licensing has not already been exhausted.

### Preparing the Design

This section uses the completed Calc design, which you can create either by completing the PROcapture tutorial or by copying a completed design from one of the solutions directories.

**Note:** All of the screen outputs refer to the processing of the XC3000A solutions design on a PC. Other parts or platforms have slightly different outputs.

A full solution for the PROcapture tutorial is supplied in the solution directory located under the directory where the PROseries software was installed:

```
...\tutorial\vwlogic\procalc\calc3ka\soln_3ka
```



1. Using the Windows File Manager, copy the contents of this directory to the directory where you will be performing the XACT-Performance tutorial.
2. Invoke Xilinx PROflow and select the Design Entry icon.
3. Select the **Project Manager** button.
4. Press the **Create** button.
5. Double-click on the directory that you made in step 1 in the Create Project dialog box.
6. Verify that the directory is shown in the Directory text box, and press **OK**.  
The PROJman Create dialog box appears.
7. Select **No** to re-initialize the viewdraw.ini file.
8. Click on **Exit** in the PRO Series Project Manager.
9. In the Select Family dialog box, select **XC3000A** and click on **OK**.
10. After exiting from the Select Family box, select the **CALC.1** design in the Design Entry dialog box, and select **OK** to open the schematic.

## 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, and these groups are then used as starting points or ending points for timing specification. Timing requirements are defined as the maximum acceptable delays from the sources in one defined group, through intermediate combinatorial logic and interconnect, to the associated loads in another group.

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. This step is not necessary if you are using only predefined groups.

2. Add a TIMEGRP symbol and add attributes to the symbol. These attributes can combine the groups defined in step 1 into additional, more complex, groups. This step is optional.
3. Add a TIMESPEC symbol and add attributes to the symbol, defining the timing requirements for the groups defined in steps 1 and 2.

## Grouping Symbols with TNM Attributes

The most basic and flexible way of defining these groups 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 group, which you can reference in a from:to statement.

### TNMs on Logic Primitives

TNMs are applicable to four types of primitives: flip-flops, latches, RAMs, and I/O pads. A group must not contain more than one symbol type, with the exception of flip-flops and latches, which can be included in the same group. 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 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.

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 group name, with the exception of FFS and LATCHES, which can be in the same group if desired.

### TNMs on Nets to Tag Flip-Flops

The TNM attribute can also be placed on a net, using the TNM=*identifier* syntax. The software pushes the attribute forward through combinatorial logic fed by the net 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. Consequently, if a flip-flop is sourced by the output of a RAM, and a TNM attribute is attached to the write enable of the RAM, the flip-flop becomes part of the group, not the RAM.

## Grouping Symbols by Predefined Groups

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.

To simplify the grouping procedure in such cases, Xilinx provides four predefined groups: FFS (flip-flops), LATCHES, PADS (I/O pads), and RAMS (XC4000 family RAM elements). Instead of placing a TNM attribute on each symbol, you can reference the entire group in a from:to statement by taking advantage of the predefined keyword.

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 FFS predefined group.

## 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 groups that are related in some way, such as

instances within the same macro, flip-flops driven by a common clock or clock enable, and so forth.

## Combining Groups with the TIMEGRP Symbol

Once groups are defined with TNM attributes, it can be useful to define new groups in terms of existing ones. You may wish to combine two or more groups, define a group of all symbols not already included in another group, or designate a group of flip-flops triggered by a given clock edge. You can also use TIMEGRP to designate a group by the output net names of the primitive symbols.

To create these new groups, add the TIMEGRP symbol to your schematic, then add an attribute for each new group definition. The name of the attribute is the new group name. The value of the attribute is the group definition.

Each TIMEGRP symbol has room for eight group definitions. If you need to define more than eight groups, add additional TIMEGRP symbols to your schematic. You can place TIMEGRP symbols at any level of the hierarchy.

### Joining Two or More Groups into One

You can define a new group as the combination of two or more existing groups using the following syntax:

```
newgroup=group1:group2[ . . . :groupn ]
```

### Using the EXCEPT Statement

A group 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 group that consists of all flip-flops not already in the first group. You can create the new group by defining an attribute that contains an EXCEPT statement. Use the following syntax:

```
newgroup=group1:EXCEPT:group2
```

*Group1* is replaced by one of the predefined groups (FFS, PADS, RAMS, or LATCHES) or by the name of a user-defined group. *Group2* is replaced by the name of a user-defined group.

For example, in the situation just discussed, assume that the group defined using TNMs is called FFGRP1, and the new group name is FFGRP2. You can create the group of all flip-flops not in the FFGRP1 group by adding the following attribute to the TIMEGRP symbol.

```
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:

```
newgroup=RISING:group
```

```
newgroup=FALLING:group
```

The new group consists of all symbols within *group* that are clocked by the specified clock edge.

### Forming Groups by Output Net Name

You can define a new group 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 full hierarchical external net name is used, that is, the name of the net joining the PAD to the I/O primitive.)

Specify the group of all blocks with output net names beginning with *name* using the following syntax:

```
newgroup=class(name*)
```

*Class* is one of FFS, RAMS, LATCHES, or PADS. This designation defines a new group called *newgroup*, which consists of all blocks in the designated class with output net names starting with the string *name*.

You can use the wildcard characters \* and ? to represent any character string and any single character, respectively.

**Note:** This wildcard capability must be used with caution. If your design contains unrelated net names beginning with the same string, they may be included in your time group and subsequently cause errors in XNFMerge or XNFPrep. If you attempt to apply the attribute to all blocks in a given schematic by specifying the instance

name of the symbol, but the outputs of some flip-flops are renamed at a higher level of hierarchy, they will not be included in the group.

## Attaching Timing Specifications with the TIMESPEC Symbol

Once you have defined appropriate groups by attaching TNM attributes to symbols and, optionally, by combining these groups using a TIMEGRP symbol, the next step is to add the timing specifications to the schematic. First, place a TIMESPEC symbol on the schematic, then add the from:to timing requirements in the form of Viewlogic attributes. As with the TIMEGRP symbol, the TIMESPEC symbol itself has no electrical functionality but serves as a placeholder for XACT-Performance attributes.

Use the following syntax to add timing specification attributes to a TIMESPEC symbol:

```
TSid=FROM:group:TO:group=time
```

All TIMESPEC attribute names must start with TS, followed by a unique identifier (*id* in the example just given). The two *group* references are replaced with the appropriate group names, as defined by TNMs, TIMEGRP symbols, or predefined groups. *Time* specifies the timing requirement, in microseconds ( $\mu$ s), nanoseconds (ns), kilohertz (kHz), or megahertz (MHz). If no units are specified, *time* is assumed to be in nanoseconds.

For example, to specify that the pad-to-setup path delay between all pads and all flip-flops should be no greater than 40 ns, add the following attribute to the TIMESPEC symbol:

```
TS01=FROM:PADS:TO:FFS=40NS
```

**Note:** FROM:PADS:TO:FFS is not exactly equivalent to the pad-to-setup path delay, 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. 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 group referenced by the pad-to-setup specification, use the TIMEGRP symbol to define a group such as PADS:EXCEPT:*pads\_sourcing\_clocks*, where *pads\_sourcing\_clocks* is another created group, which includes all of the clock pads.

Each TIMESPEC symbol has room for eight timing specification attributes. If you need more than eight specifications, add additional TIMESPEC symbols to your schematic. You can place TIMESPEC symbols at any level of the hierarchy.

## Deciding When to Use XACT-Performance

One 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 that it calculates on the basis of 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 inform PPR to ignore selected paths by using the IGNORE statement. To do this, define an attribute attached to a TIMESPEC primitive using the following syntax:

```
TSid=IGNORE
```

*Id* is a unique identifier. Attaching this *TSid* attribute to a net or load pin causes PPR to ignore any paths that include the net or load pin.

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, which are designed to work with 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 the Floorplanner (planning the placement of your logic to simplify the data flow) or using CLBMAPs, FMAPs, HMAPs, or LOC constraints to lock down symbol locations. See the *Libraries Guide*, the *Development System Reference Guide*, or the *Floorplanner Reference/User Guide* for more information on how to floorplan your design.

## Setting Default Timing Requirements

In this tutorial, you add XACT-Performance symbols and 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.

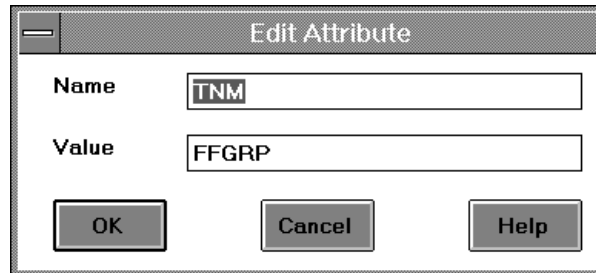
**Note:** For more information on the Calc design, refer to the “PROcapture and PROsim Tutorial” chapter.

### Adding a TNM Attribute

First, use a TNM attribute to define a group of flip-flops. The group is specifies the clock-to-pad default timing requirement.

1. Open PROcapture and load the CALC top-level schematic.
2. Select the main clock net, CLK, and then **Change** → **Object Attributes** → **Dialog**.
3. When the Edit Attributes dialog box appears, click on the **Add** button.
4. Add a name and value, as shown in Figure 4-1, and select **OK**.





**Figure 4-1 Edit Attribute Dialog Box**

You should see the name and value of the attribute you just added in the Edit Attribute dialog box.

5. Select **OK** to save the change.
6. You may wish to select the attribute and use the **Edit → Move** command to move it below the net, where it is easier to read.

You have defined a set with the name FFGRP that consists of all flip-flops driven by the CLK clock net.

**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 to demonstrate how to attach TNM attributes 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 500 kHz. This clock speed is quite slow, so the placing and routing software has no problem meeting the timing requirements.

1. Select **Add → Component**.
2. Place a TIMESPEC symbol on the CALC schematic in the open area on the left.
3. Select the TIMESPEC symbol by clicking the left mouse button on the bar labeled “TIMESPEC” across the top.

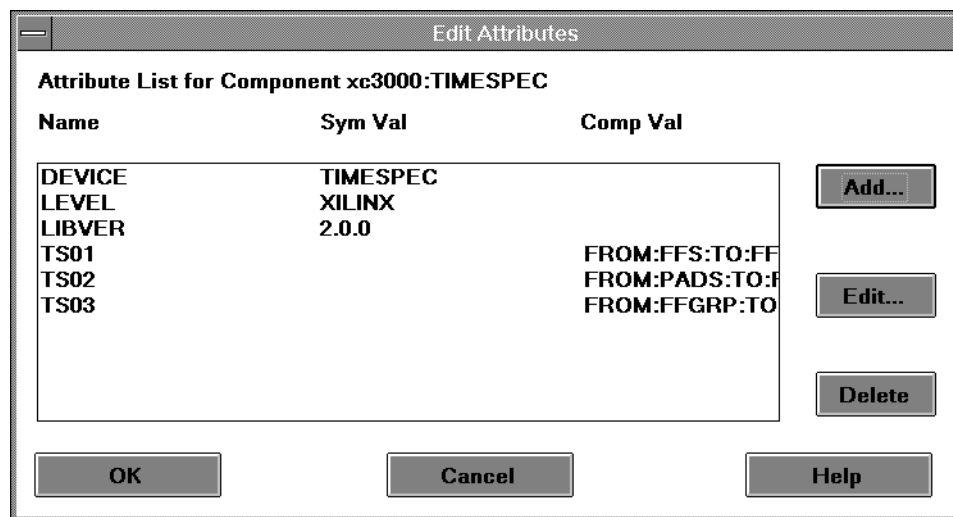
4. Select **Change** → **Object Attributes** → **Dialog**.

The Edit Attributes dialog box appears in the lower right corner of the screen. The DEVICE, LEVEL, and LIBVER attributes are predefined, and you cannot change them.

5. Click on the **Add** button on the Edit Attributes dialog box, and add the following three attributes.

Attribute Name	Comp Value
TS01	FROM:FFS:TO:FFS=500KHZ
TS02	FROM:PADS:TO:FFS=1MHZ
TS03	FROM:FFGRP:TO:PADS=1000NS

Your Edit Attributes dialog box should now look like the illustration in Figure 4-2.



**Figure 4-2 Edit Attributes Dialog Box**

6. Select **OK** to save the changes.
7. Select **File** → **Save** to save your changes to the CALC schematic.

The new TS01 attribute specifies that all clock-to-setup paths must have timing such that they can be driven by a 500-kHz 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 1000 ns and 1 MHz are interchangeable. An equivalent specification for TS03 is TS03=FROM:FFS:TO:PADS=1MHZ.

Figure 4-3 shows a portion of the CALC schematic with the TNM attribute and the TIMESPEC symbol.

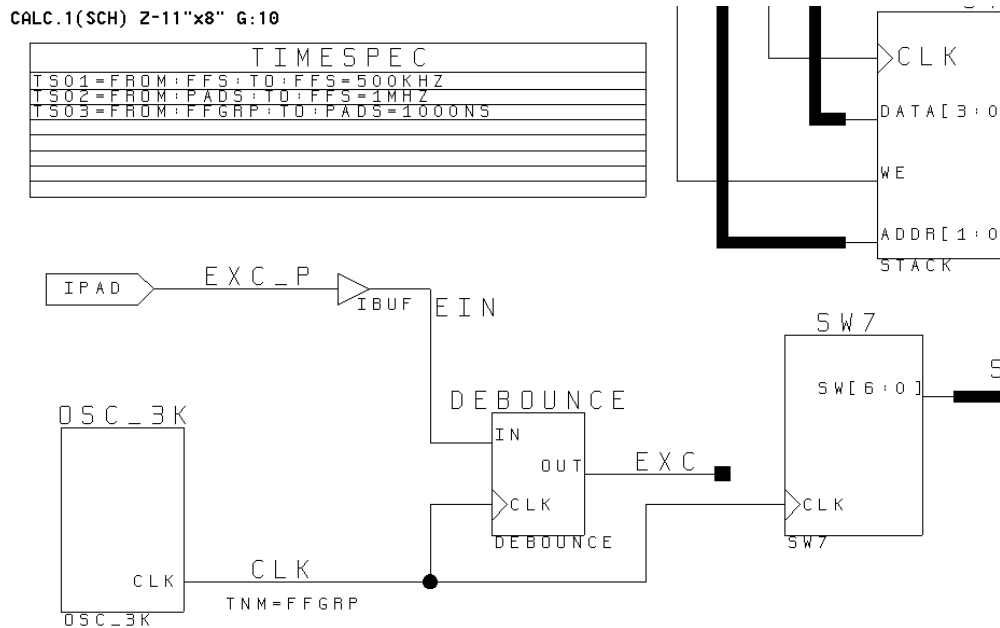


Figure 4-3 CALC Schematic with Default Timing Constraints

## Adding Timing Constraints to Specific Paths

In the previous section, you applied default timing specifications to the Calc tutorial design. In simple applications, the specifications described to this point in the tutorial 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 the groups to be used as end points for timing specification. You can use these groups to apply timing requirements from one group to another or from one group to the same group.

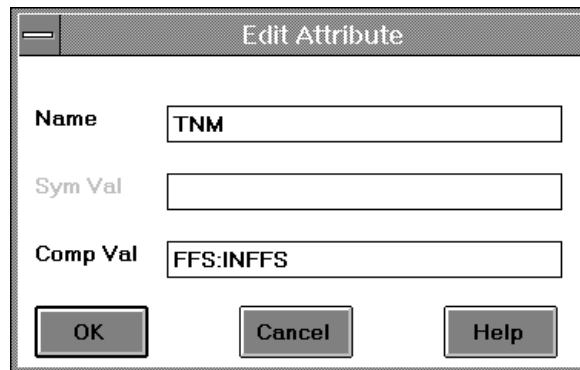
#### Defining the INFFS Group

The INFFS group includes all input flip-flops from the SW7 macro.

1. Select the SW7 symbol from the top-level CALC schematic.
2. Select **Change** → **Object Attributes** → **Dialog**.
3. Select **Add** from the Edit Attributes dialog box.

The Edit Attribute dialog box appears.

4. Add the following attribute in the Edit Attribute dialog box, shown in Figure 4-4.



**Figure 4-4 Edit Attribute Dialog Box**

5. Select the **OK** button to save the new attribute.
6. In the Edit Attributes dialog box, select **OK**.

This attribute definition places the IFDs (IOB flip-flops) in the SW7 macro into a new timing group called INFFS. There are two types of symbols in SW7: FFS and PADS. Therefore, the FFS keyword is necessary to specify that the group contains all symbols of type FFS. Since TNM=PADS:INFFS is specified, the group refers to the input pads. If neither group is specified, XMake fails while running XNFMerge (and issues a message explaining the error) since FFS and PADS cannot be in the same group.

### Defining the STACKER Group (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 included in a group called STACKER. If your design is an XC3000A design, skip this section and continue with the next section, “Defining the STACKER Group (XC3000A Only).”

1. Select the STACK\_4K symbol from the top-level CALC schematic.
2. Select **Change** → **Object Attributes** → **Dialog**.
3. Select **Add** from the Edit Attributes dialog box and add the following data.

Attribute Name	Comp Value
TNM	RAMS:STACKER

4. Select the **OK** button to save the new attribute.
5. In the main Edit Attributes dialog box, select **OK**.

This attribute defines the new group named **STACKER** to contain all RAM symbols in the **STACK\_4K** macro.

**Note:** The RAMS predefined group could have been used instead of defining **STACKER** since the **STACK\_4K** schematic contains only symbols of the RAMS type. However, this example shows how a subset of the RAM primitives would be grouped in a more realistic design.

### Defining the **STACKER** Group (XC3000A Only)

The Calc design for the XC3000A implements the stack using flip-flops. In this section, these flip-flops are included in a group called **STACKER**. If your design is an XC4000 family design, skip this section and continue with the next section, "Defining the **ALUFF** Group."

1. Select the **STACK** symbol from the top-level **CALC** schematic.
2. Select **Change** → **Object Attributes** → **Dialog**.
3. Select **Add** from the Edit Attributes dialog box and add the following data.

Attribute Name	Comp Value
TNM	FFS:STACKER

4. Select the **OK** button to save the new attribute.
5. In the main Edit Attributes dialog box, select **OK**.

This attribute defines a new group named **STACKER** that contains all flip-flops in the **STACK** macro.

**Note:** The FFS predefined group is optional in defining STACKER since the STACK macro contains only symbols of type FFS. This example shows how only the FF primitives inside the macro should be grouped in a more realistic design.

### Defining the ALUFF Group

Follow these steps to define the ALUFF group.

1. Select the ALU symbol from the top-level CALC schematic.
2. Select **Change** → **Object Attributes** → **Dialog**.
3. Select **Add** from the Edit Attributes dialog box and add the following data.

Attribute Name	Comp Value
TNM	ALUFF

4. Select the **OK** button to save the new attribute.
5. In the main Edit Attributes dialog box, select **OK**.

This attribute defines a new group named ALUFF that contains all flip-flops in the ALU macro.

### Defining the CTLFF Group

Next, define the CTLFF group using the following procedure.

1. Select the CONTROL symbol from the top-level CALC schematic.
2. Select **Change** → **Object Attributes** → **Dialog**.
3. Select **Add** from the Edit Attributes dialog box and add the following data.

Attribute Name	Comp Value
TNM	CTLFF

4. Select the **OK** button to save the new attribute.
5. In the main Edit Attributes dialog box, select **OK**.

This attribute defines a new group named CTLFF that contains all flip-flops in the CONTROL macro.

6. Select **File** → **Save** to save your changes to the CALC schematic.

### Defining the STFF Group

Define the STFF group by following the steps in this section.

1. Verify that the CONTROL symbol is still selected.
2. Select **View** → **Push Into Schematic**.
3. Select the STATE\_4K symbol (XC4000 family designs) or the STATMACH symbol (XC3000A designs).
4. Select **Change** → **Object Attributes** → **Dialog**.
5. Select **Add** from the Edit Attributes dialog box, and add the following data.

Attribute Name	Comp Value
TNM	STFF

6. Select the **OK** button to save the new attribute, and in the main Edit Attributes dialog, select **OK**.

This attribute defines a new group named STFF that contains all flip-flops in the state machine. STFF is a subset of the CTLFF group. Even though this TNM attribute is defined on a lower-level schematic, it is perfectly valid to use it in the TIMESPEC symbol on a different level of the hierarchy.

**Note:** The TNM attributes just discussed are all attached to macros, which simplifies grouping. Alternatively, you can attach TNM attributes to individual primitives, such as one or more individual flip-flops. In this case, attach the `TNM=group_name` attribute individually to each flip-flop you want included in the group.

### Grouping Using TIMEGRP

In addition to the TNM groups, you can use the TIMEGRP primitive symbol to define new groups in terms of existing sets or predefined symbol types (FFS, RAMS, PADS, LATCHES).



Use the TIMEGRP symbol to create additional groups. Because the CALC schematic sheet is already crowded, place the TIMEGRP symbol in the CONTROL schematic. TIMESPEC and TIMEGRP symbols can be placed at any level in the hierarchy.

1. Select **Add** → **Component**, and place a TIMEGRP symbol on the CONTROL schematic in the open area at the lower left.
2. Select the TIMEGRP symbol by clicking the left mouse button on the bar labeled “TIMEGRP” across the top.
3. Select **Change** → **Object Attributes** → **Dialog**.
4. Select **Add** from the Edit Attributes dialog box, and add the following data.

Attribute Name	Comp Value
LEDPADS	PADS(LED*)
CTL_ALU_FF	CTLFF:ALUFF
CTL_ADR_FF	CTLFF:EXCEPT:STFF

5. Select the **OK** button to save the new attribute, and in the main Edit Attributes dialog, select **OK**.

This procedure defines three new groups.

- The first new group, LEDPADS, represents all symbols of the PADS type that begin with the “LED” character string. 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 named LED/LED0\_P, LED/LED1\_P, and so forth.
  - The second group, CTL\_ALU\_FF, combines all flip-flops in the two TNM groups, CTLFF and ALUFF.
  - The third group, CTL\_ADR\_FF, represents all symbols in the CTLFF group except for those in the STFF group. Since CTLFF includes all flip-flops in the CONTROL block, and STFF includes all flip-flops in the state machine (STATE\_4K or STATMACH), CTL\_ADR\_FF represents all flip-flops in the CB2CLED macro below CONTROL.
6. Select **File** → **save** to save your changes to the CONTROL schematic.



3. Select **Add** from the Edit Attributes dialog box and add the following data.

Attribute Name	Comp Value
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

4. Select **Accept** to save the new attributes.
5. Select **File** → **Save** 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 group 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 TIMEGRP CTL\_ADR\_FF to the group of flip-flops named ALUFF to be 50 ns.
- TS06 specifies the clock-to-setup timing from the TIMEGRP CTL\_ALU\_FF to the TNM group 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 group ALUFF to all the pads in the design to be 45 ns.

## Making a Final Check

Lastly, check to make sure that the TIMEGRP and TIMESPEC symbols look like the ones in Figure 4-6 and Figure 4-7. The order of the attributes is unimportant.

The completed CALC schematic is shown in Figure 4-8.

<b>TIMEGRP</b>
LEDPADS=PADS(LED*)
CTL_ALU_FF=CTLFF:ALUFF
CTL_ADR_FF=CTLFF:EXCEPT:STFF

**Figure 4-6 Completed TIMEGRP Symbol**

<b>TIMESPEC</b>
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 4-7 Completed TIMESPEC Symbol**

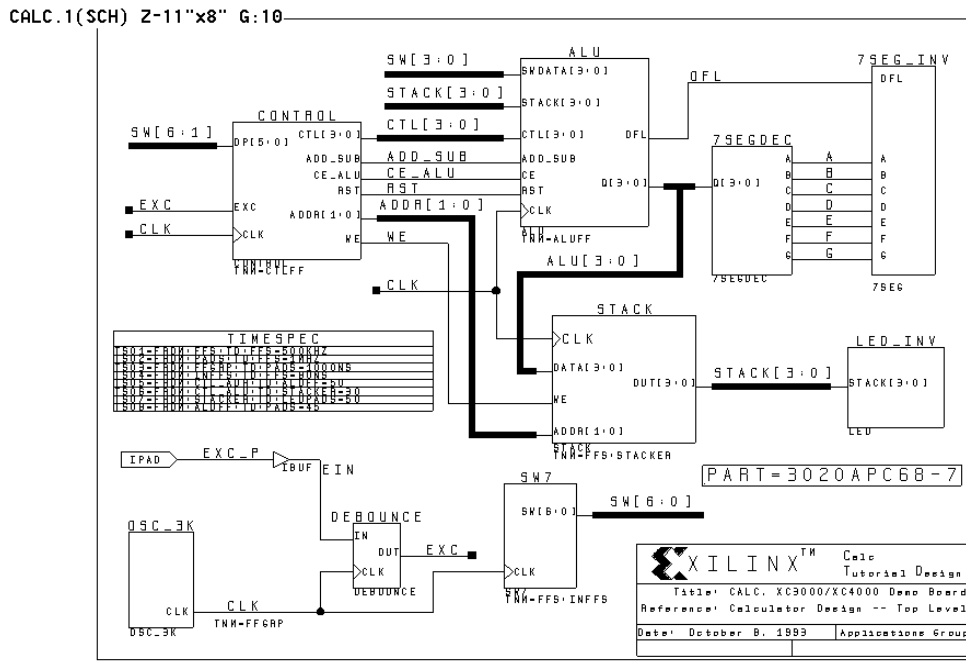


Figure 4-8 CALC Schematic with TNMs and TIMESPEC Symbol

All desired XACT-Performance specifications have been entered on the schematic. The next step is to implement the design using the Design Manager and to verify the results in the calc.out and ppr.log files.

## Implementing the Calc Design

The translation of designs containing XACT-Performance attributes is exactly the same as that 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 process, which includes mapping, placing, 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.

You can use PROflow's Xilinx Implementation icon to invoke the Xilinx Design Manager, from which you can control the design's implementation in a Xilinx FPGA. For detailed information on the Xilinx Design Manager, refer to the *Design Manager/Flow Engine Reference/User Guide*.

## Translating the Netlist

Run the Xilinx Design Manager to generate files for device programming.

1. From PROflow, press the Xilinx Implementation icon to start the Xilinx Design Manager.
2. In the Xilinx Design Manager, select **File** → **New Project**.
3. Using the **Browse** button, change to the WIR directory in the tutorial directory and select the CALC.1 sheet.
4. Press **OK**.
5. Change the target family to **XC3000**.
6. Press the **Translate** button.
7. In the Translate Options window, press the **OK** button.

The Design Manager runs the XMake program, which converts the PROcapture schematic into an XFF netlist.

## Examining Translation Output

You can use the Report Browser tool to examine the reports produced by the implementation process. To access the Report Browser, click on **Utilities** → **Report Browser** or use the Report Browser toolbar icon, shown in Figure 4-9.



Figure 4-9 Report Browser Toolbar Icon

If your XACT-Performance specifications have any syntax errors, they are flagged by the optimization stage of the process, and the errors are listed in the DRC report. If errors are detected, the implementation process stops. The DRC report includes a list of all errors and warnings issued during the timing checks.

Always check the DRC report after translation.

## Creating a Routed Design

Use the Xilinx Design Manager to implement the converted netlist in the target device.

1. Select **Design** → **Implement**.
2. In the XC3000A Design Implementation dialog box, confirm that the **Produce Configuration Data** check box is selected.
3. Select the **Produce Timing Simulation Data** check box. (The timing data will be used in a later part of the tutorial.)
4. Press **Run**.

The Xilinx Flow Engine, invoked by the Design Manager, optimizes, maps, places, and routes the design, and creates a bitstream file that can be downloaded to the part. The resulting Flow Engine history file, `program.his`, is shown following.

```
xnfprep design=calc.xff outfile=calc.xtg savesig=false
partType=3020APC68-7 ignore_timespec=none xblox_prep=true
split_report=true

xblox calc.xtg calc.xg parttype=3020APC68-7 archopt=false
mergeio=false reg_rlocs=false

xnfprep design=calc.xg outfile=calc.xtf savesig=false
ignore_timespec=none split_report=true

xnfmap calc.xtf calc.map

ppr design=calc placer_effort=2 router_effort=2
ignore_timespec=none path_timing=true route_thru_bufg=ok
route_thru_blks=ok guide_blks=all lock_routing=whole_sigs
split_report=true

xdelay -d -w calc.lca

lca2xnf -g calc.lca calc.xnf
```

```

xnfb calc.xg calc.xnf
makebits -l -mbo=calc.mbo calc.lca

```

## Examining the Implementation Output

The implementation process generates a number of reports, such as the Routing Report and the XACT-Performance report. Following is an extract from a typical XACT-Performance report.

```

XACT-PERFORMANCE RESULTS FOR DESIGN CALC
From PPR Version Beta-5.2.0b
1995/07/25 18:35:04
Xilinx, Inc.

(c) Copyright 1995. All Rights Reserved.

XACT Performance Summary
-----
Parttype Used : 3020APC68
Speed Grade : -7

Limit      Actual      End-
(ns)       * (ns)      Points
-----
45.0       23.1        0/1      TS08=FROM:ALUFF:TO:PADS
50.0       43.9        0/4      TS07=FROM:STACKER:TO:LEDPADS
30.0       24.8        0/16     TS06=FROM:CTL_ALU_FF:TO:STACKER
50.0       54.0        1/1      TS05=FROM:CTL_ADR_FF:TO:ALUFF
80.0       52.0        0/8      TS04=FROM:INFFS:TO:FFS
1000.0     48.7        0/5      TS03=FROM:FFGRP:TO:PADS
1000.0     18.1        0/12     TS02=FROM:PADS:TO:FFS
2000.0     54.0        0/64     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 use the -FailedSpec and/or -
TSMaxpaths options of the Timing Analyzer -TimeSpec command,
accessible through the XDE or the Timing Analyzer program.

```

This summary reports that the TS05 timing specification does not meet the deadline. It tells you which XACT-Performance specification failed and gives the timing that XACT-Performance achieved for the associated set of paths.

The timing reported depends on the target device. Additionally, unless you specify identical input parameters, each place and route 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.

Both reports contain errors and warnings related to the paths in the device and should be examined after every implementation run. Examples of some of the common warnings that can be generated are the following.

- Warning 6811 refers to the oscillator loop in the XC3000A design. Since this loop is deliberate in the Calc design, you can safely ignore this warning.
- 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, these warnings are not a matter of concern.
- Warning 7028 is a reminder that PPR does not trace timing through asynchronous Set/Reset control signals. Since the Timing Analyzer traces these paths by default, you must disable this tracing in the Timing Analyzer to compare PPR and Timing Analyzer results. This procedure is discussed in the “Disabling False Paths” section in the Timing Analyzer 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 run and can safely be ignored.

## Using the Timing Analyzer

The next step is to verify the timing of your routed design using the Timing Analyzer.

The Timing Analyzer is a static timing analysis tool that reports the worst-case timing delays of a routed FPGA design. It can generate the following three timing reports.

- The Performance Summary report lists 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. The clock-related path types are reported for each clock in the design.
- The Performance To TimeSpecs report indicates which XACT-Performance constraints are met and reports all missed paths in detail.
- The Detailed Path report displays detailed path timing information according to the selected options. It offers insight on which paths in the design are the most critical. This information helps you determine where to make modifications to meet the design timing requirements.

The Timing Analyzer also has options that control the information generated in a report. A typical design can generate a large amount timing data, and these filters manage this data. Many of them are demonstrated in this tutorial.

- Path filters are commands that limit or control the type of information presented in a report.
- Timing specification filters allow you to generate reports restricted to those path delays that did not meet XACT-Performance timing specifications, to ignore paths that have no timing specifications, or to ignore the paths related to specific timing specifications. These filters apply only when you generate a Performance to Timespec report.
- Path Analysis filters limit the data to paths from certain sources, certain destinations, or both. Also included is a filter for the path type, which only affects the Performance Summary and Detailed Path reports.
- Common filters is a collection of useful filters. They can filter on the basis of the names of nets in paths, possible false paths, and logic loops at defined points.
- Macro commands allow you to create and save a sequence of commands for use at a future time.

For more information on the Timing Analyzer, refer to the *Timing Analyzer Reference/User Guide*.

## Analyzing the Calc Design

This section analyzes the results of the XACT-Performance design created earlier in this tutorial. The routed LCA file contains actual timing delays as well as XACT-Performance specifications. The Timing Analyzer analyzes this information and displays different types of delay paths according to the options that you select.

This section demonstrates a typical Timing Analyzer analysis command sequence.

**Note:** The sample Timing Analyzer output in this tutorial is from a single Calc LCA file, targeted to the XC3020APC68-7 device. Your results will vary.

## Invoking the Timing Analyzer

Invoke the Timing Analyzer by following these steps.

1. Ensure that the placed and routed revision of the design is currently highlighted in the Design Manager window.
2. Click on the Timing Analyzer icon in the Tools section, or select the **Tools** → **Timing Analyzer** command.

## Disabling False Paths

In the Calc design, there are a number of flip-flops with asynchronous Reset signals. Normally you would not be concerned with asynchronous reset paths when considering clock-to-setup requirements, so you could ignore the paths through these asynchronous inputs during timing analysis. By default, the Timing Analyzer ignores these paths to match PPR's operation.

The categories of potential false path and default settings are summarized in Table 4-1.

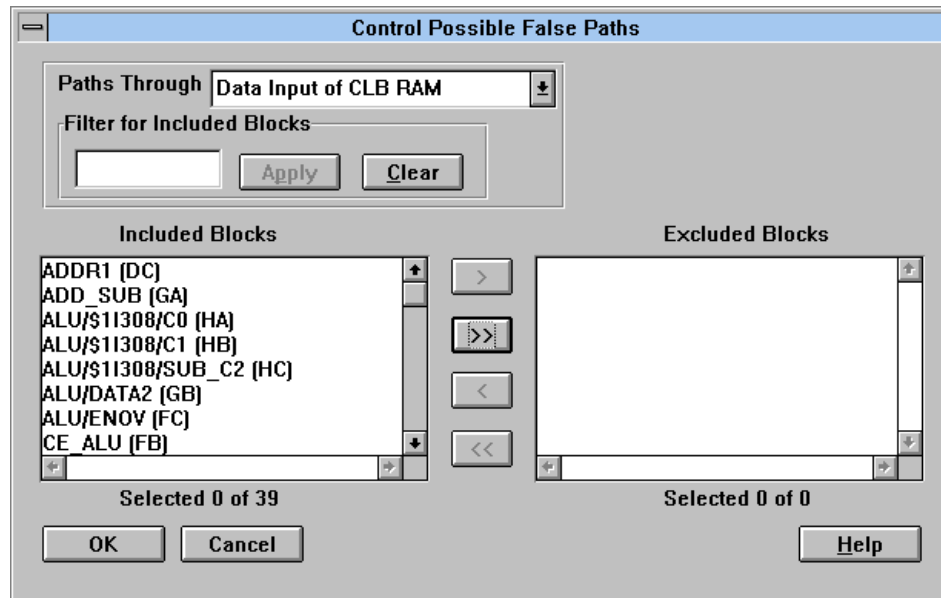
**Table 4-1 False Paths**

<b>Paths Through...</b>	<b>Default Setting for Qualifying Elements</b>
Data Input to CLB RAM	All included in timing analysis
Write Enable of CLB RAM	All included in timing analysis
Tristate to Output pin of BUFT	All included in timing analysis
Input Pin to Output pin of BUFT	All included in timing analysis
Output pin to Input pin of IOB	All excluded in timing analysis
Tristate pin to Input pin of IOB	All excluded in timing analysis
Set/Reset to Q pin of Flip-flop	All excluded in timing analysis

You can change these default settings as follows.

1. From the Timing Analyzer menu, select **Path Filters** → **Common Filters** → **Control Possible False Paths**.

The Control Possible False Paths dialog box appears, as shown in Figure 4-10.



**Figure 4-10 Control Possible False Paths Dialog Box**

2. In the Paths Through field, choose the path category that you want to change.  
 In the Included Blocks and Excluded Blocks fields are the names of the elements whose paths match the chosen category.
3. Move elements between the Included Blocks and Excluded blocks fields, if desired, using the buttons in between the two lists.
4. When the selection process is complete, click on **OK**.

## Resetting Path Filters

You may want to reset the path filters to their defaults to avoid the effects previous modifications. To do this, select **Path Filters** → **Reset Path Filters**.

## Displaying Current Settings

Select the **Analyze** → **Show Settings** command to display the current settings for the Timing Analyzer. Figure 4-11 gives an example of this command.

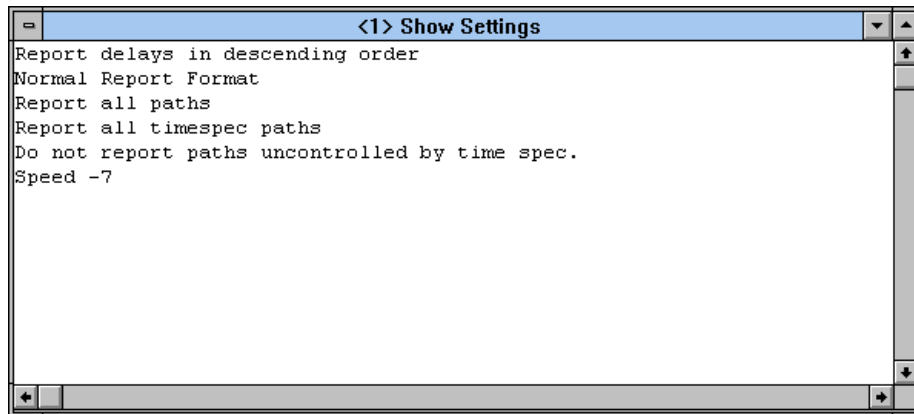


Figure 4-11 Show Settings Window

## Generating a Performance Summary Report

You can perform a quick analysis to determine the worst-case timing for the Calc design using the following procedure.

1. Select **Analyze** → **Performance Summary**.

The report appears in a window.

2. To print the report to your default printer, select **File** → **Print**, and click on **OK**.
3. To close the report, select **File** → **Close Report**.

For the XC3020APC68 design, the Timing Analyzer reports the same combinatorial logic loop detected by PPR. Since this loop is deliberately included to create an oscillator, you can ignore this message.

A partial report file for the XC3020APC68-7 design is shown in Figure 4-12. The last line of the file shows that the design operates at approximately 18.6MHz under worst-case conditions.

The report lists the worst-case pad-to-pad, pad-to-setup, clock-to-pad, and clock-to-setup delays. 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.
Timing Analyzer Report File:
        Design: c:\workarea\calc\xproject\ver2\rev1\calc.lca (3020APC68-7)
        Program: TIMING ANALYZER Beta-5.2.0b
        Speedsfile: File 3020a.spd, Version 3000A.1, Revision 3020A.5

Timing Analyzer timing analysis options
From all.
To all.
Worst-case pad-to-pad path delay      :   37.0ns   (1 block level)
  Pad "OSC_3K/CQ" (P47) to Pad "OSC_3K/CQL" (P42.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                          :   48.6ns   (2 block levels)
(Includes an external output margin of 0.0ns.)
Clock to Q, net "ADDR0" to Pad "LED/LED3_P" (P32.O)

Clock to Setup (same edge)            :   54.0ns   (5 block levels)
Clock to Q, net "ADDR0" to FF Setup (D) at "OFL.D"
Target FFX drives output net "OFL"

        Minimum Clock Period :   54.0ns
        Estimated Maximum Clock Speed : 18.6Mhz
```

**Figure 4-12 Performance Summary for XC3020APC68-7 Design**

## Generating a Performance to TimeSpecs Report

Use the Analyze → Performance To TimeSpecs command to evaluate the timing of your design with respect to the XACT-Performance attributes that you added to your schematic.

The Report Paths Failing TimeSpec and Ignore TimeSpecs commands on the TimeSpec Filters submenu of the Path Filters menu are

particularly useful for evaluating XACT-Performance results. Report Paths Failing TimeSpec reports all path delays that do not meet timing specifications, and Ignore TimeSpecs allows you to select which XACT-Performance specifications you wish to be considered.

**Note:** Used without the Report Paths Failing TimeSpec option, the Ignore TimeSpecs option allows you to create reports showing the worst paths for each XACT-Performance specification.

To generate a sample Performance to TimeSpec report, use the following procedure.

1. Click on the **Path Filters** → **TimeSpec Filters** → **Report Paths Failing TimeSpec** command to specify that you want to view only the failed timing specifications.
2. Select **Analyze** → **Performance To TimeSpecs**.

The Performance To TimeSpecs report is created in a window so you can scroll through it.

3. To print the report to your default printer, select **File** → **Print**, and click on **OK**.
4. To close the report, select **File** → **Close Report**.

**Note:** The Options → Report Options command specifies the maximum number of paths to display for each failed timing specification. If you do not specify this number, the report file lists the delay of every path controlled by each XACT-Performance specification in your design. The Timing Analyzer may run out of memory in this case; if not, it produces a very large output file.

As noted earlier, for the XC3020APC68 design, the Timing Analyzer reports the same combinatorial logic loop detected by PPR. Since this loop is deliberately included to create an oscillator, you can ignore this message.

A portion of the Performance To TimeSpecs report for the XC3020APC68-7 design is shown in Figure 4-13.

```
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.
Timing Analyzer Report File:
```



## XACT-Performance and Timing Analyzer Tutorial

---

Design: c:\workarea\calc\xproject\ver2\rev1\calc.lca (3020APC68-7)  
Program: TIMING ANALYZER Beta-5.2.0b  
Speedsfile: File 3020a.spd, Version 3000A.1, Revision 3020A.5

Timing Analyzer path report options:

TimeSpec `TS01' from group `FFS' to group `FFS' is 2000.0ns.  
TimeSpec `TS02' from group `PADS' to group `FFS' is 1000.0ns.  
TimeSpec `TS03' from group `FFGRP' to group `PADS' 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:

OFL

TimeGroup `CTL\_ADR\_FF' contains these Flip-Flop output nets:

ADDR0 ADDR1

TimeGroup `CTL\_ALU\_FF' contains these Flip-Flop output nets:

ADDR0 ADDR1 CONTROL/STATMACH/OTHER CONTROL/STATMACH/PUSH OFL WE

.  
.  
(OTHER TIMEGROUP DEFINITIONS OMITTED)  
.

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 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.

-----  
TimeSpec `TS01' summary:

From TimeGroup `FFS'  
To TimeGroup `FFS'

TimeSpec limit is : 2000.0ns (Spec speed = 500.0KHz)  
Worst path delay is : 53.9ns (Real speed = 18.6MHz)  
TimeSpec passes by : 1946.1ns

NOTE: This analysis does not include paths which start and end in the  
same block (CLB or IOB) and use no external routing.

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 : 18.4ns (Real speed = 54.3MHz)
TimeSpec passes by : 981.6ns
```

NOTE: This analysis does not include paths which start and end in the same block (CLB or IOB) and use no external routing.

List of delay paths that fail the TimeSpec:  
There are no paths that fail the TimeSpec.

-----  
TimeSpec `TS03' summary:  
From TimeGroup `FFGRP'  
To TimeGroup `PADS'

```
TimeSpec limit is : 1000.0ns (Spec speed = 1.0MHz)
Worst path delay is : 48.6ns (Real speed = 20.6MHz)
TimeSpec passes by : 951.4ns
```

NOTE: This analysis does not include paths which start and end in the same block (CLB or IOB) and use no external routing.

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 : 52.0ns (Real speed = 19.2MHz)
TimeSpec passes by : 28.0ns
```

NOTE: This analysis does not include paths which start and end in the same block (CLB or IOB) and use no external routing.

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 : 53.9ns (Real speed = 18.6MHz)
*** TimeSpec FAILS by : 3.9ns ***
```

NOTE: This analysis does not include paths which start and end in the same block (CLB or IOB) and use no external routing.

List of delay paths that fail the TimeSpec:

## XACT-Performance and Timing Analyzer Tutorial

```
Logical Path                                     Delay Cumulative
-----
Source clock net : "CLK" (Rising edge)
From: Blk ADDR1          CLOCK to DC.Y          : 4.5ns ( 4.5ns)
Thru: Net ADDR0          to EF.E               : 7.2ns ( 11.7ns)
Thru: Blk STACK/$1I15/M01 to EF.X             : 5.1ns ( 16.8ns)
Thru: Net STACK/$1I15/M01 to FE.B             : 2.7ns ( 19.5ns)
Thru: Blk STACK0        to FE.Y               : 5.6ns ( 25.1ns)
Thru: Net STACK0        to HA.D               : 5.4ns ( 30.5ns)
Thru: Blk ALU/$1I308/C0 to HA.X               : 5.1ns ( 35.6ns)
Thru: Net ALU/$1I308/C0 to HB.B               : 0.6ns ( 36.2ns)
Thru: Blk ALU/$1I308/C1 to HB.X               : 5.6ns ( 41.8ns)
Thru: Net ALU/$1I308/C1 to HC.B               : 1.4ns ( 43.2ns)
Thru: Blk ALU/$1I308/SUB_C2 to HC.Y           : 5.1ns ( 48.3ns)
Thru: Net ALU/$1I308/SUB_C2 to GC.D           : 0.6ns ( 48.9ns)
To: FF Setup (D), Blk OFL                       : 5.0ns ( 53.9ns)
```

```
Target FFX drives output net "OFL"
Dest clock net : "CLK" (Rising edge)
Clock delay to Source clock pin : 2.6 ns
Clock delay to Dest clock pin   : 2.6 ns
Clock net "CLK", delta clock delay [skew] : 0.0 ns
```

```
. (OTHER 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 : 24.8ns (Real speed = 40.4MHz)
TimeSpec passes by : 5.2ns
```

```
NOTE: This analysis does not include paths which start and end in the
same block (CLB or IOB) and use no external routing.
```

```
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 : 43.9ns (Real speed = 22.8MHz)
TimeSpec passes by : 6.1ns
```

NOTE: This analysis does not include paths which start and end in the same block (CLB or IOB) and use no external routing.

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 : 23.1ns (Real speed = 43.3MHz)  
TimeSpec passes by : 21.9ns

NOTE: This analysis does not include paths which start and end in the same block (CLB or IOB) and use no external routing.

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 4-13 Performance to TimeSpecs Report for XC3020APC68-7

The first section of the report 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 — the default specification is set to “auto,” which means that PPR assigns some reasonable value as the timing specification.

The report 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.

In Figure 4-13, 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 CTL\_ADR\_FF set was defined in the “Grouping Sets with TIMEGRP” section. It is defined as CTLFF:EXCEPT:STFF and should contain only the flip-flops in the CB2CLED macro below CONTROL. The outputs of the CB2CLED macro are ADDR0 and ADDR1, and Figure 4-13 verifies that the CTL\_ADR\_FF set includes only these two flip-flop outputs.

The worst path delay is then reported for each XACT-Performance specification.

For TS05, which missed the target timing, the report includes a detailed listing of the paths that failed. 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 correct the design.

For example, for the failed path shown in Figure 4-13, the longest delay on the path is a 7.2-ns delay between the Y output of CLB DC and the E input of CLB EF. (The block name of CLB DC 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 Y output of HC and the D input of CLB GC; the delay between these adjacent CLBs is only 0.6 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 Performance To TimeSpecs report in Figure 4-13 and the Performance Summary report in Figure 4-12 shows that the PPR and the Timing Analyzer results vary by only a few tenths of nanoseconds. When there is a discrepancy between the two, use the Timing Analyzer results.

For example, the Performance To TimeSpecs report in Figure 4-14 shows a worst-case delay for TS05 paths of 53.9 ns. The Performance Summary report in Figure 4-12 shows the worst path delay to be 54.0 ns.

## **Generating a Detailed Path Report**

You can create a Detailed Path report either by analyzing general path types or analyzing designated paths from specific sources to specific destinations.

## Reporting by Path Type

To simplify the analysis of designs without XACT-Performance specifications, you can restrict the Detailed Path report to certain path types.

1. Select **Path Filters** → **Path Analysis Filters** → **Select Path Types**.
2. Select the desired path type, either Clock To Setup, Clock To Pad, Pad To Setup, Pad To Pad, or Paths Ending at Clock Pin of Flip-flop.
3. Click on **OK**.

The number of paths reported depends on the value of the Maximum Number of Paths value specified in the dialog box activated by the **Options** → **Report Options** command.

Following is an example using the Clock To Setup path filter to report the single slowest path between any two flip-flops clocked by the same edge of the clock.

1. Select **Options** → **Report Options** and type 1 in the **Maximum Number of Paths** box.
2. Select **OK**.

**Note:** If you do not set the Maximum Number of Paths option, the report file lists delays for every path in your design. The Timing Analyzer may run out of memory in this case; if not, it produces a very large output file.

3. Select **Path Filters** → **Reset Path Filters** to clear all previous settings.
4. Select **Path Filters** → **Path Analysis Filters** → **Select Path Types**.
5. In the Select Path Types dialog box, deselect all the path types except Clock to Setup.
6. Click on **OK**.
7. Select **Analyze** → **Detailed Path Report**.

This command opens a report screen that allows you to scroll through the report. A portion of this file, calcc2s.xrp, is shown in Figure 4-14.

8. To send the report to the default printer, select **File** → **Print** and click on **OK**.
9. To close the report, select **File** → **Close Report**.

The only paths reported are those between flip-flops, in other words, the paths that fall into the clock-to-setup or FROM:FFS:TO:FFS category. Since the Maximum Number of Paths option was set to 1, 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.
-----
Paths not used in TimeSpecs :

Logical Path                                Delay Cumulative
-----
Source clock net : "CLK" (Rising edge)
From: Blk ADDR1          CLOCK to DC.Y      : 4.5ns ( 4.5ns)
Thru: Net ADDR0          to EF.E           : 7.2ns ( 11.7ns)
Thru: Blk STACK/$1I15/M01 to EF.X         : 5.1ns ( 16.8ns)
Thru: Net STACK/$1I15/M01 to FE.B         : 2.7ns ( 19.5ns)
Thru: Blk STACK0        to FE.Y           : 5.6ns ( 25.1ns)
Thru: Net STACK0        to HA.D           : 5.4ns ( 30.5ns)
Thru: Blk ALU/$1I308/C0  to HA.X         : 5.1ns ( 35.6ns)
Thru: Net ALU/$1I308/C0  to HB.B         : 0.6ns ( 36.2ns)
Thru: Blk ALU/$1I308/C1  to HB.X         : 5.6ns ( 41.8ns)
Thru: Net ALU/$1I308/C1  to HC.B         : 1.4ns ( 43.2ns)
Thru: Blk ALU/$1I308/SUB_C2 to HC.Y       : 5.1ns ( 48.3ns)
Thru: Net ALU/$1I308/SUB_C2 to GC.D       : 0.6ns ( 48.9ns)
  To: FF Setup (D), Blk OFL                : 5.0ns ( 53.9ns)
Target FFX drives output net "OFL"
Dest clock net : "CLK" (Rising edge)
Clock delay to Source clock pin : 2.6 ns
Clock delay to Dest clock pin   : 2.6 ns
Clock net "CLK", delta clock delay [skew] : 0.0 ns
-----

```

Figure 4-14 Clock To Setup Output for XC3020APC68-7 Design

## Reporting by Sources and Destinations

To generate a report on a specific path, use the Select Sources and Select Destinations filters. 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. The Timing Analyzer cannot report path delays that span more than one flip-flop.)

Follow these steps to generate a detailed report of the delays on this path.

1. Select **Path Filters** → **Reset Path Filters** to clear all previous settings.
2. Select **Path Filters** → **Path Analysis Filters** → **Select Sources**.

The Select Sources dialog box appears. The Selected Sources list box displays all sources by default.

3. Click on the Remove All (<<) button.
4. In the Available Sources list box, find the source flip-flop DEBOUNCE/DELAY1 by scrolling down the list. Alternatively, you can type **\*DELAY\*** in the Filter for Available Sources box, and click on **Apply** to reduce the selection of names in the Available Sources box. If you make a mistake while typing in the Filter for Available Sources field, click on the **Clear** button and try again.
5. Select the source called DEBOUNCE/DELAY1, and click on the Add (>) button to transfer it to the Selected Sources list box.
6. Click on **OK**.
7. Select **Path Filters** → **Path Analysis Filters** → **Select Destinations**.

The Select Destinations dialog box appears. The Selected Destinations list box displays all destinations by default.

8. Click on the Remove All (<<) button.
9. In the Available Destinations list box, find the destination flip-flop DEBOUNCE/DELAY2 by scrolling down the list. Alternatively, you can type **\*DELAY\*** in the Filter for Available Destinations list box, and click on **Apply** to reduce the selection of names in the



Available Destinations box. If you make a mistake while typing in the Filter for Available Destinations field, click on the **Clear** button and try again.

10. Select the destination called DEBOUNCE/DELAY2, and click on the Add (>) button to transfer it to the Selected Destinations list box.

11. Click on **OK**.

12. Select **Analyze** → **Detailed Path Report**.

This command creates a report screen that allows you to scroll through the report. A portion of this report is shown in Figure 4-15.

13. To send the report to the default printer, select **File** → **Print** and click on **OK**.

14. To close the report, select **File** → **Close Report**.

The report details a single path, the path between the Delay1 flip-flop and the Delay2 flip-flop, as shown in Figure 4-15. 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 9.7 ns under worst-case conditions. There is no clock skew between the two flip-flops.

```

.
.
.
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 CA.Y          : 4.5ns ( 4.5ns)
Thru: Net DEBOUNCE/DELAY2 to DA.A              : 0.7ns ( 5.2ns)
      To: FF Setup (D), Blk DEBOUNCE/DELAY3    : 4.5ns ( 9.7ns)
Target FFY drives output net "DEBOUNCE/DELAY3"
Dest clock net : "CLK" (Rising edge)
Clock delay to Source clock pin : 2.5 ns
Clock delay to Dest clock pin   : 2.5 ns
Clock net "CLK", delta clock delay [skew] : 0.0 ns
-----

```

**Figure 4-15 Dpath.xrp File for an XC3020APC68-7 Design**

The source and destination used in this example are both flip-flops. A pull-down menu in the Select Sources and Select Destinations dialog boxes allows you to specify sources and destinations using the names of flip-flops, IOBs, clocks, nets, pins, and CLBs. Refer to the *Timing Analyzer Reference/User Guide* for more details.

## Using the Console Window

As you modify options and settings, the commands being executed are logged in the Console window. You can see this window by selecting **View** → **Console**. An example of the Console window is shown in Figure 4-16. You can also type in keyboard commands in the Console window.

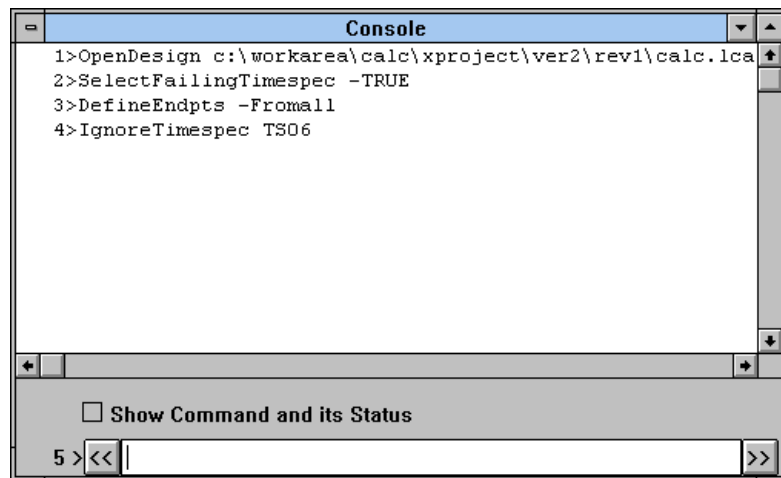


Figure 4-16 Console Window

## Creating Macros

The **File** → **New Macro** command opens a New Macro window and displays a supplemental toolbar for macros. Using the Copy and Paste toolbar icons, you can copy commands from the Console window and paste them into the New Macro window. Once you create a macro, you can run it by clicking on the Run icon on the macro toolbar or by selecting **File** → **Run Macro**.

You can save macros with the **File** → **Save Macro** or **File** → **Save As Macro** commands.

You can load any existing macros with the **File** → **Open Macro** command.

## **Further Reading**

Before using XACT-Performance for your own designs, you should read the “XACT-Performance Utility” section of the *Xilinx Reference Guide*. You can find more information on the Timing Analyzer in the *Timing Analyzer Reference/User Guide*.



# ***Viewlogic Tutorials***

***Index***



# Index

---

## Numerics

7SEG\_INV block, 1-3

7SEG\_TRU block, 1-3

7SEGDEC block, 1-3

## A

ABEL Hardware Description Language, 3-1

ABEL-HDL file, 3-3, 3-6

    compiling, 3-10

    DCSET statement, 3-8

    End statement, 3-9

    Equations statement, 3-8

    Istype statement, 3-7

    Module statement, 3-6

    Pin statement, 3-7

    State statement, 3-7

    State\_diagram statement, 3-8

    State\_register statement, 3-7

    Test\_vectors statement, 3-9

    Title statement, 3-6

    Xilinx Property Initialstate statement, 3-8

ABL2XNF, 3-10

ACLK symbol, 1-103

Add Component dialog box, 1-62, 1-64, 1-91, 1-93, 1-94

Add Label dialog box, 1-38, 1-98

Add Text dialog box, 1-48

Add toolbar icon, 1-62

ADD\_SUB symbol, 2-9

ALU block, 1-2, 1-26, 1-31, 1-87, 1-88, 1-132, 2-3

ALU vector, 1-120, 1-121, 1-122

ANDBLK2 symbol, 1-31, 1-59, 1-87, 1-90, 1-97

ANDBUS2 symbol, 2-10

APR, 1-104, 4-1

Assign command, 1-124, 1-132, 1-137

Attribute toolbar icon, 1-41

attributes, 1-30

    adding PINTYPE, 1-39

    adding to nets, 1-29, 1-108

    BLKNM, 4-7, 4-19

    BOUNDS, 2-7

    changing size, 1-45

    changing visibility, 1-47

    DEF, 2-8, 3-16

    DEVICE, 4-12

    ENCODING, 2-7

    entering X-BLOX, 2-7

    EXT, 1-106

    FAST, 1-112

    FILE, 3-16, 3-22

    HBLKNM, 4-7

    INVMASK, 2-10

    LEVEL, 1-106, 4-12

    LIBVER, 1-106, 2-8, 4-12

    LOC, 1-105, 4-10

    purpose, 1-104

    SLOW, 1-112

autoexec.bat file, 1-4

Available Sources list box, 4-42

## B

back-annotation, 2-21, 2-22, 3-22, 3-26, 3-27

binary radix toolbar icon, 1-130

- BIT file, 1-195, 2-23, 3-28
  - bitstream, 1-192, 1-195, 2-21, 3-26
  - BLKNM attribute, 4-7, 4-19
  - block sheet, 1-33
  - Block Type command, 3-15
  - BLX file, 2-12, 2-17
  - BOUNDS attribute, 2-7
  - Box command, 1-29, 1-34, 1-36
  - Box toolbar icon, 1-34, 1-36
  - boxes, 1-29
  - BUFG symbol, 1-103
  - Bus command, 1-30, 1-72
  - bus definition symbols, 2-6
  - Bus toolbar icon, 1-72
  - BUS\_DEF symbol, 2-6, 2-7
  - BUS\_IF symbols, 2-6
  - buses, 1-30, 1-69, 1-120, 1-121
    - adding labels, 1-74
    - adding to schematic, 1-69, 1-72, 1-96
    - legal names, 1-76
    - vectors, 3-7
    - X-BLOX, 2-5, 2-12, 2-16
- C**
- Calc design, 1-2, 1-6, 1-8, 1-12, 2-2
  - CALC.1 schematic, 1-8, 1-13, 1-15, 1-17, 1-18, 1-21
  - calc.log file, 1-176
  - calc.see file, 1-139
  - calc.vsm file, 1-116, 1-173, 1-176
  - calc.wfm file, 1-120, 1-128, 1-178, 1-186, 1-187, 1-188
  - calct.wfm file, 1-187, 1-189
  - canceling current command, 1-28
  - Change Component dialog box, 1-84
  - Change Text dialog box, 1-49, 1-53, 1-76
  - Check program, 2-14, 2-15, 3-20, 3-21
  - Classic Defaults option, 1-21, 3-13
  - CLBMAPs, 4-10
  - CLK signal, 1-120, 1-123
  - clock, 3-7, 3-9, 4-6, 4-7, 4-8, 4-11, 4-13, 4-40, 4-43
  - clock skew, 1-102
  - Clock To Pad option, 4-40
  - Clock To Setup option, 4-40
  - clock-to-pad path delay, 4-10, 4-13, 4-21, 4-33
  - clock-to-setup path delay, 4-5, 4-13, 4-21, 4-33
  - Close command, 1-21, 1-29, 1-57
  - Close Report command, 4-34, 4-41, 4-43
  - Close toolbar icon, 1-57, 1-58
  - command file, 1-140, 1-176
    - functional simulation, 1-117, 1-140, 2-15, 3-21
    - timing simulation, 1-174, 2-23, 3-27
  - Command File command, 2-15, 3-21, 3-27
  - Command File option, 1-117, 1-174
  - Command Files dialog box, 1-174, 1-175
  - common filters, 4-28
  - Compile Xilinx FPGA Netlist command, 3-10
  - Component command
    - Add menu, 1-30, 1-62, 1-91, 4-11, 4-19
    - Change menu, 1-31, 1-84, 2-3, 2-5, 3-13
  - Components list box, 1-63, 1-91, 1-92
  - Composite block type, 3-21
  - Console command, 4-44
  - Console window, 4-44
  - constraints file, 1-153, 1-155
  - Constraints File option, 1-153
  - CONTROL block, 1-2
  - Control Possible False Paths dialog box, 4-30
  - Copy command, 1-30, 1-65, 1-80
  - Copy toolbar icon, 1-66, 4-44
  - Create Project dialog box, 1-8, 1-9
- D**
- D flip-flops, 3-7
  - dangling nets, 1-69, 1-70
  - DATA\_REG symbol, 2-9
  - DCSET statement, 3-8
  - DEBOUNCE block, 1-2



- DEF attribute, 2-8, 3-16
  - delta time, 1-190
  - Design Contains XBLOX, RAM, ROM or XABEL Module option, 1-117, 2-11, 3-18
  - Design Entry dialog box, 1-13, 1-14, 2-3, 3-3, 3-12
  - Design Entry icon, 1-13, 1-18, 1-116, 2-2, 3-2, 3-3, 3-12, 4-3
  - design implementation
    - creating Xilinx project, 1-142, 2-18, 3-22, 4-24
    - implementing EPLD designs, 1-165
    - implementing FPGA designs, 1-152
    - invoking Design Manager, 1-140
    - translating the design, 1-145, 2-18, 3-22, 4-24
    - XACT-Performance designs, 4-23
    - X-BLOX designs, 2-18
    - Xilinx ABEL designs, 3-22
  - Design Implementation dialog box, 1-153, 1-154, 1-156, 1-161, 1-165, 1-169, 1-171, 3-24
  - Design Manager, 1-1
    - implementing design, 1-140, 2-18, 4-24, 4-25
    - invoking, 1-140, 2-18, 3-22, 4-24
    - invoking Flow Engine, 1-152, 2-20, 3-24, 4-25
    - invoking XMake or XEMake, 1-150, 2-18, 3-23, 4-24
    - placing project directory, 1-144
    - updating after fitting, 1-172
    - updating after implementation, 1-164
    - updating after translation, 1-151
    - updating for EPLD design, 1-151
    - window, 1-140, 1-141, 1-152
  - Design Name option, 1-15
  - Design Type option, 1-15
  - Detailed Path report, 4-28, 4-39
  - Detailed Path Report command, 4-40, 4-43
  - DEVICE attribute, 4-12
  - Device option, 1-147
  - DIP switch, 1-194
  - down arrow, 1-27
  - Download cable, 1-192, 1-193
- E**
- Edit Attribute dialog box, 1-42, 1-106, 1-110, 1-112, 4-11
  - Edit Attributes dialog box, 1-42, 1-43, 1-105, 1-106, 1-109, 1-112, 2-8, 4-10, 4-12, 4-15, 4-16, 4-17, 4-18, 4-19
  - Edit Paste command, 1-31
  - ENCODING attribute, 2-7
  - End statement, 3-9
  - EPLD fitter, 1-103
  - Equations statement, 3-8
  - EXC\_P signal, 1-105, 1-120, 1-124, 1-131
  - EXCEPT statement, 4-6, 4-8
  - Excluded Blocks option, 4-31
  - Execute command, 1-2, 1-131, 1-196
  - Execute Netlister option, 1-117, 2-22, 3-26
  - Execute Power On Reset option, 1-117, 1-119, 1-174, 2-22, 3-26
  - Exit command, 1-2, 1-118
  - Explicit flag *see* X flag
  - EXT attribute, 1-106
  - External flag *see* X flag
- F**
- F flag, 1-109
  - Family option, 1-147
  - FAST attribute, 1-112, 1-113
  - Fast Function Blocks, 1-109
  - FD4CE component, 1-87, 1-94
  - FFS set, 4-5, 4-6, 4-7, 4-11, 4-15, 4-17, 4-18, 4-20
  - FILE attribute, 3-16, 3-22
  - File Open dialog box, 1-31, 1-59
  - File Save As dialog box, 1-54
  - Filter for Available Destinations option, 4-43
  - Fitting tab, 1-167

- flags, 1-108
- flip-flops
  - D, 3-7
  - grouping into FFS set, 4-5
  - grouping with EXCEPT statement, 4-6
  - grouping with TIMEGRP symbol attributes, 4-7
  - grouping with TNM attribute, 4-4, 4-5, 4-6
  - in macros, 4-4
  - IOB, 4-15
  - outputs, 4-38
  - path delay, 4-8, 4-29, 4-40, 4-42
  - reducing logic levels on critical paths, 4-10
  - stack, 4-16
  - state machine, 3-7, 4-18
- floorplanning, 4-10
- Flow Engine
  - creating a configuration bitstream, 1-162
  - creating timing simulation data, 1-162, 1-170
  - fitting design, 1-170
  - history file, 2-20, 3-24, 4-25
  - invoking, 1-161, 1-169, 2-20, 3-24, 4-25
  - optimizing design, 1-161, 1-169
  - placing and routing design, 1-162
  - purpose, 1-152, 1-165
  - running LCA2XNF, 2-21, 3-25
  - running MakeBits, 2-21, 3-26
  - running PPR, 2-21, 3-25
  - running X-BLOX, 2-20
  - running XDelay, 2-21, 3-25
  - running XNFMAP, 2-21, 3-25
  - running XNFPrep, 2-20, 3-25
  - window, 1-169
- FMAPs, 4-10
- forward tracing, 4-5
- FPGA demonstration board, 1-2, 1-3, 1-100, 1-103, 1-192, 1-193, 1-194
- Full command, 1-24, 1-28, 1-56, 1-61
- Full toolbar icon, 1-24, 1-56
- functional simulation, 1-1
  - adding signals to waveform, 1-120
  - adding vectors to waveform, 1-121
  - changing display radix, 1-129
  - changing radices for PROcapture display, 1-134
  - creating simulation network, 1-116, 1-117, 2-11, 3-18
  - defining clocks, 1-123
  - defining design inputs, 1-123
  - defining input values, 1-124
  - designs with X-BLOX modules, 2-11
  - designs with Xilinx ABEL modules, 3-17
  - invoking PROwave, 1-127
  - re-creating previous simulation, 1-140
  - simulating Calc design, 1-131
  - simulating design inputs, 1-125
  - viewing waveforms, 1-138
- Functional Simulation dialog box, 1-116, 1-119, 2-11, 3-18
- Functional Simulation PROsim icon, 1-116, 2-11, 3-17, 3-18
- Functional Simulation PROwave icon, 1-127
- G**
- GCLK symbol, 1-103
- Generate Symbol of Type option, 3-11
- global clock buffers, 1-102, 1-103
- global reset, 1-117, 1-119, 1-125, 1-176
- GR, 1-117
- Grid command, 1-184
- grid spacing, 1-61
- GSR, 1-117
- Guide Design option, 1-153
- Guide/Resource tab, 1-158
- guided design, 1-153

**H**

HBLKNM attribute, 4-7  
 Help command, 1-28  
 High-Density Function Blocks, 1-109  
 HMAPs, 4-10

**I**

I/O pads, 4-4, 4-5  
 IBVER attribute, 1-106  
 icons, 1-22, 1-27  
 IFD, 4-15  
 IFD component, 1-114  
 IFDX1 component, 1-115  
 Ignore TimeSpecs command, 4-33, 4-34  
 ILD component, 1-114  
 Implement command, 1-153, 1-165, 2-19, 3-24, 4-25  
 Implementation icon, 1-140, 2-18, 3-22, 4-24  
 implementation *see* design implementation  
 Implementation tab, 1-158  
 Implementation Template dialog box, 1-157, 1-159, 1-166, 1-167  
 In command, 1-25, 1-26, 1-28  
 In toolbar icon, 1-25  
 Included Blocks option, 4-31  
 instance names, 1-97  
 INVMASK attribute, 2-10  
 IOBs
 

- flip-flops, 1-114, 1-115, 4-15
- latches, 4-5

 IPAD symbol, 1-105  
 Istype keyword, 3-7

**K**

Karnaugh maps, 3-8

**L**

labels, 1-29
 

- adding to schematic, 1-74, 1-96
- format, 1-97
- pin, 1-37

latches, 4-4, 4-5

LATCHES set, 4-5, 4-6, 4-7, 4-18, 4-20  
 LCA file, 2-21, 3-25, 4-9, 4-27, 4-29  
 LCA2XNF, 2-21, 3-25  
 LED\_INV block, 1-3  
 LED\_P vector, 1-120, 1-121, 1-122  
 LED\_TRU block, 1-3  
 LEVEL attribute, 1-106, 4-12  
 Libraries list box, 1-63, 1-91, 1-95  
 LIBVER attribute, 2-8, 4-12  
 List Files of Type option, 1-14  
 LOC constraint, 1-105, 4-10  
 log file, 1-140

**M**

macro symbols, 4-4  
 MAK file, 2-19, 3-24  
 MakeBits program, 1-195, 2-21, 3-26  
 Maximum Number of Paths option, 4-40, 4-41  
 MDI *see* Multiple Document Interface  
 Module block type, 3-15, 3-21  
 Module statement, 3-6  
 Move command, 1-30, 1-51, 1-68, 4-11  
 Move toolbar icon, 1-52, 1-69  
 MS-DOS, 1-195  
 Multiple Document Interface, 1-22, 1-55  
 MUXBUS<sub>x</sub>, 2-10

**N**

Net command, 1-30, 1-69  
 Net toolbar icon, 1-69  
 nets, 1-30
 

- adding flags, 1-108
- adding labels, 1-74
- adding to schematic, 1-69, 1-96
- dangling, 1-69, 1-70

 New Macro command, 4-44  
 New Macro window, 4-44  
 New Project command, 1-142, 2-18, 3-22, 4-24  
 New Project dialog box, 1-142, 1-143

Notepad, 1-117, 1-118, 1-140, 1-176, 2-15, 2-23, 3-21, 3-27

## O

Object Attribute command, 1-40, 1-44  
Object Attributes command, 1-29, 1-30, 1-41, 1-47, 2-8, 3-16, 4-10, 4-12, 4-14, 4-16, 4-17, 4-18, 4-19, 4-20  
Object Detail command, 3-15  
Object Label command, 1-29, 1-38, 1-74, 1-98  
OFD component, 1-114  
OFDT component, 1-115  
opcodes, 1-2, 1-120, 1-124, 1-132, 1-196  
Open command, 1-21, 1-29, 1-31, 1-56, 1-59, 1-78, 1-88, 1-179, 2-16  
Open dialog box, 1-128, 1-142, 1-154, 1-178  
Open Macro command, 4-45  
Open toolbar icon, 1-31  
Optimization tab, 1-158, 1-159, 1-167  
ORBLK symbol, 1-87  
ORBLK2 symbol, 1-53, 1-90, 1-93, 1-98  
ORBUS2 symbol, 2-10  
OSC\_3K block, 1-2, 1-100  
OSC\_7K block, 1-2, 1-101  
OSC\_7KXCLK signal, 1-120  
oscillators, 1-100, 4-27  
Out command, 1-25, 1-26, 1-28  
Out toolbar icon, 1-25

## P

Pack Design check box, 1-160  
Package option, 1-147  
Package Selection dialog box, 2-11, 3-18  
Pad To Pad option, 4-40  
Pad To Setup option, 4-40  
PADS set, 4-5, 4-6, 4-7, 4-15, 4-18, 4-20  
pad-to-clock path delay, 4-8  
pad-to-pad path delay, 4-33  
pad-to-setup path delay, 4-8, 4-13, 4-33  
panning, 1-22, 1-23, 1-28  
Part Selector dialog box, 1-146, 1-147, 1-148

Paste command, 1-81  
Paste toolbar icon, 4-44  
path analysis filters, 4-28  
path filters, 4-28  
PATH variable, 1-5  
Paths Ending at Clock Pin of Flip-flop option, 4-40  
Paths Through option, 4-31  
Performance Summary command, 4-32  
Performance Summary report, 4-28, 4-32, 4-39  
Performance To TimeSpecs command, 4-33, 4-34  
Performance To TimeSpecs report, 4-28, 4-34, 4-39  
Pin command, 1-29, 1-36  
Pin grid array (PGA) package pins, 1-108  
pin labels, 1-37  
Pin statement, 3-7  
Pin toolbar icon, 1-36, 1-37  
pinout, 1-104, 3-11  
PINTYPE attribute, 1-39  
pipelining, 4-10  
PLCC package pins, 1-108  
Pop command, 1-29, 1-99, 1-103, 1-114, 1-116, 2-9, 3-15, 4-20  
Pop Schematic toolbar icon, 1-99  
Powerview, 1-4  
PPR, 1-104, 4-1  
    default path timing, 4-9, 4-23, 4-38  
    detecting combinatorial logic loops, 4-32, 4-34  
    interaction with XACT-Performance, 4-9, 4-14  
    LCA file, 4-9, 4-27  
    log file, 4-9, 4-23, 4-24  
    running with Flow Engine, 2-21  
    running with XMake, 3-25  
ppr.log file, 4-9, 4-23, 4-24  
PRG file, 1-165  
primitives

- grouping symbols with TNM, 4-4, 4-5
  - macro, 4-4
- Print command, 4-34, 4-41, 4-43
- PRLD, 1-117
- PRO Series
  - exiting, 1-2
  - function keys, 1-17
  - installing tutorial, 1-5
  - mouse button functions, 1-17
  - plotting directory, 1-5
  - programs, 1-1
  - required software, 1-3
  - search path variable, 1-5
  - tutorial devices, 1-1
  - tutorial length, 1-1
- PRO Series dialog box, 1-52
- PRO Series Project Manager, 1-7, 1-8, 1-9, 1-10, 2-2, 3-2, 4-3
- PROcapture
  - adding buses, 1-30, 1-72, 1-96
  - adding components, 1-30, 1-61
  - adding FAST attribute, 1-111
  - adding flags to nets, 1-108
  - adding labels, 1-29, 1-74, 1-96, 1-97
  - adding LOC attributes, 1-105
  - adding nets, 1-30, 1-69, 1-96
  - adding pin labels, 1-37
  - adding pins, 1-36
  - adding PINTYPE attributes, 1-39
  - adding SLOW attributes, 1-111
  - adding symbol boxes, 1-29, 1-34
  - adding symbol pins, 1-29
  - adding symbol text, 1-48
  - adding text, 1-30
  - adding X-BLOX module, 2-3
  - adding Xilinx ABEL module, 3-3, 3-12
  - changing attribute size, 1-45
  - changing components, 1-82
  - changing display radices, 1-134
  - changing symbol size, 1-34
  - changing symbol text size, 1-50
  - changing window colors, 1-20
  - closing windows, 1-29, 1-57
  - command summary, 1-28
  - controlling attribute visibility, 1-30, 1-47
  - controlling schematic visibility, 1-87
  - copying components, 1-30, 1-31, 1-65
  - copying schematics, 1-78
  - creating symbols, 1-31, 1-53
  - exchanging components, 1-103
  - help, 1-28
  - invoking, 1-18, 2-16
  - listing object descriptions, 3-15
  - making icons of schematics, 1-27
  - moving components, 1-30, 1-50, 1-68
  - opening schematics, 1-29, 1-59
  - panning, 1-22, 1-23, 1-28
  - pasting components, 1-31
  - placing FD4CE component, 1-94
  - placing symbols, 1-90
  - popping out of schematics, 1-29, 1-99
  - purpose, 1-1
  - pushing into schematics, 1-28, 1-89, 1-99, 1-101
  - pushing into symbols, 1-29, 1-88
  - refreshing screen, 1-28
  - saving schematics, 1-29, 1-52, 1-86, 1-99
  - selecting components, 1-30, 1-31, 1-82
  - verifying Xilinx ABEL symbol attributes, 3-16
  - verifying Xilinx ABEL symbol type, 3-15
  - viewing schematics, 1-28, 1-100, 3-14
  - viewing symbols simultaneously, 1-55
  - X-BLOX buses, 2-5, 2-6, 2-7
  - zooming, 1-22, 1-23, 1-28
- PROcapture Colors command, 1-20, 3-13
- PROcapture icon, 1-125, 1-126, 1-136
- PROcolor Manager dialog box, 1-20, 1-21
- Produce Configuration Data check box, 2-19, 2-21, 3-26, 4-25

- Produce Timing Simulation Data check box, 1-157, 2-20, 3-24, 4-25
  - PROflow, 1-127
    - controlling design flow, 1-11
    - functional simulation, 1-116, 1-117
    - icon, 1-6
    - initial design status, 1-12
    - invoking Notepad, 1-117, 1-176, 2-15, 2-23, 3-21, 3-27
    - invoking PROcapture, 2-16
    - invoking PROsim, 1-118, 1-119, 1-176, 2-15, 3-21, 3-27
    - invoking PROwave, 1-116, 2-16, 2-23, 3-21, 3-27
    - invoking VSM, 1-117
    - invoking XSimMake, 1-117, 2-12, 3-18, 3-26
    - reactivating, 1-16, 1-116, 1-127, 1-177
    - simulating startup sequence, 1-118
    - starting, 1-6
    - timing simulation, 1-173, 1-176
    - window, 1-11
  - Program Option Templates option, 1-154, 1-157, 1-166
  - project definition, 1-8
  - project directory, 1-144
  - PROjman Create dialog box, 2-2, 3-2, 4-3
  - PROM, 1-194
  - PROsim
    - adding signals to waveform, 1-120
    - adding vectors to waveform, 1-121, 1-122
    - changing radices for PROcapture display, 1-134
    - creating log file, 1-140
    - defining clocks, 1-123
    - defining input values, 1-124
    - functional simulation, 1-118, 2-15, 3-21
    - invoking, 1-116, 1-118, 1-173, 1-176, 2-15, 2-23, 3-21, 3-27
    - processing calc.vsm file, 1-176
    - purpose, 1-1, 1-116
    - simulating Calc design, 1-131, 1-176
    - simulating design inputs, 1-125
    - timing simulation, 1-173, 2-23, 3-27
  - PROsim icon, 1-116, 1-134, 1-173, 2-11, 2-22, 3-17, 3-18, 3-26
  - PROsynthesis, 1-15
  - PROwave
    - changing display radix, 1-129
    - invoking, 1-127, 1-177, 2-16, 2-23, 3-21, 3-27
    - obtaining delta time, 1-190
    - obtaining transition time, 1-187
    - purpose, 1-116
    - specifying signals displayed, 1-120
    - viewing waveforms, 1-138, 1-177
    - zooming, 1-182
  - PROwave icon, 1-127, 1-134, 1-138, 1-177, 2-16, 3-21, 3-27
  - PRP file, 2-12, 2-17, 2-20, 3-25
  - Push Into Schematic command, 1-28, 1-89, 1-99, 1-112, 1-115, 2-3, 3-13, 3-14, 4-18
  - Push Into Symbol command, 1-29, 3-16
  - Push Schematic toolbar icon, 1-89
- R**
- radices, 1-129, 1-134
  - RAM modules, 1-117
  - RAMs, 4-4, 4-5, 4-15, 4-16
  - RAMS set, 4-5, 4-6, 4-7, 4-16, 4-18, 4-20
  - Read Part From Design check box, 1-146
  - Refresh command, 1-28, 1-47
  - Region command, 1-25, 1-28, 1-183
  - Report Browser, 1-163, 1-171, 4-24
  - Report Browser command, 4-24
  - Report Browser toolbar icon, 4-24
  - Report Options command, 4-34, 4-40
  - Report Paths Failing TimeSpec command, 4-33, 4-34
  - Reset Path Filters command, 4-31, 4-40, 4-42
  - Resource tab, 1-167

- reverse polish notation, 1-196
- Review Log option, 1-150
- rocker switches, 1-196
- ROM modules, 1-117
- Routing Report, 4-26
- Run Macro command, 4-44
- S**
- Save As command, 1-29, 1-53, 1-55
- Save As Macro command, 4-45
- Save As toolbar icon, 1-53
- Save command, 1-2, 1-29, 1-52, 1-77, 1-86, 1-99, 1-114, 1-139, 2-9, 3-13, 4-12, 4-18, 4-19, 4-21
- Save Macro command, 4-45
- schematics *see* PROcapture
- Select command, 1-30
- Select Component command, 1-82
- Select Component dialog box, 1-82
- Select Destinations command, 4-42
- Select Destinations dialog box, 4-44
- Select Family dialog box, 1-10, 2-2, 3-2, 4-3
- Select Part option, 1-146
- Select Path Types command, 4-40
- Select Path Types dialog box, 4-40
- Select Sources command, 4-42
- Select Sources dialog box, 4-42, 4-44
- Selected Sources list box, 4-42
- Set Radix command, 1-130
- Sheet Size command, 1-34
- sheets, 1-14, 1-24, 1-33, 1-59
- Show Settings command, 4-32
- Simulate Equations command, 3-9
- simulation network, 1-116, 1-117, 1-118, 1-173, 1-176, 2-15, 3-21, 3-27
- Size command, 1-46
- slew rate, 1-112
- SLOW attribute, 1-112
- STACK block, 1-2
- STACK vector, 1-120, 1-121, 1-122
- Start PROcapture check box, 1-15, 1-16, 1-18
- STAT\_ABL block, 3-3
- State keyword, 3-7
- State\_diagram statement, 3-8
- State\_register statement, 3-7
- Status command, 1-12, 1-16
- Status dialog box, 1-12, 1-16
- step size, 1-123
- SW signal, 1-124
- SW7 block, 1-2
- SW7SW\_P vector, 1-120, 1-121
- Symbol Generation utilit, 3-11
- Symbol Generation Utility icon, 3-11
- Symbol Generator dialog box, 3-11
- symbol windows, 1-32, 1-57
- symbolic state machine, 3-7
- symbols
  - adding pin labels, 1-37
  - adding pins, 1-29, 1-36
  - adding text, 1-48
  - block sheets, 1-33
  - BUS\_DEF, 2-6, 2-7
  - BUS\_IF, 2-6
  - changing size, 1-34
  - changing text size, 1-50
  - CLBMAP, 4-10
  - creating, 1-31, 1-53
  - creating boxes, 1-34
  - FMAP, 4-10
  - HMAP, 4-10
  - placing, 1-90
  - popping out of, 1-29
  - pushing into, 1-29
  - saving, 1-52
  - TIMEGRP, 4-4, 4-6, 4-18
  - TIMESPEC, 4-4, 4-8, 4-18, 4-20
  - viewing, 1-55
- SymGen, 3-15, 3-16
- SymGen Results window, 3-12
- SymWin, 3-11
- SYSPLT variable, 1-5

## T

- Target Family option, 1-144
- test vectors, 3-7, 3-9
- Test\_vectors statement, 3-9
- Text command, 1-30, 1-46, 1-48, 1-76
- Text toolbar icon, 1-48
- Tile command, 1-56, 1-79
- TIMEGRP symbol, 4-4
  - clock edges, 4-7
  - combining sets, 4-6, 4-18
  - defining groups by output net names, 4-6, 4-7
  - defining sets by output net names, 4-7
  - EXCEPT statement, 4-6, 4-8
  - purpose, 4-6
- TIMESPEC symbol, 4-4
  - purpose, 4-8
  - specifying timing constraints, 4-20
  - TNM attribute, 4-18
- Timing Analyzer
  - Console window, 4-44
  - creating macros, 4-44
  - Detailed Path report, 4-28, 4-39
  - displaying current settings, 4-32
  - filters
    - common, 4-28
    - path, 4-28
    - path analysis, 4-28
    - resetting, 4-31
    - timing specification, 4-28
  - ignoring false paths, 4-29
  - invoking, 4-29
  - Performance Summary report, 4-28, 4-32, 4-39
  - Performance To TimeSpecs report, 4-28, 4-39
  - purpose, 4-1, 4-27
  - resetting filters, 4-31
- Timing Analyzer command, 4-29
- timing simulation
  - comparing to functional simulation, 1-179
  - creating simulation network, 1-117, 1-173, 2-22, 3-26
  - designs with X-BLOX modules, 2-22, 2-23, 3-26
  - designs with Xilinx ABEL components, 3-26
  - invoking PROwave, 1-177
  - obtaining delta time, 1-190
  - obtaining transition time, 1-187
  - simulating Calc design, 1-176
  - viewing waveforms, 1-177
- Timing Simulation dialog box, 1-173, 1-175, 1-176, 3-26
- Timing Simulation PROsim icon, 1-173, 2-22, 3-26
- Timing Simulation PROwave icon, 1-177
- timing specification filters, 4-28
- Title statement, 3-6
- TNM attribute
  - adding to schematic, 4-10
  - defining sets, 4-14
  - flip-flops, 4-4, 4-5
  - forward tracing, 4-5
  - grouping symbols by predefined sets, 4-5
  - I/O pads, 4-4
  - latches, 4-4
  - macros, 4-4, 4-18
  - nets, 4-5
  - primitives, 4-4, 4-18
  - purpose, 4-4
  - RAMs, 4-4, 4-5
  - syntax, 4-4
- transition time, 1-187
- Translate, 1-145
- Translate option, 1-145
- Translate Options dialog box, 1-145, 1-148

## U

- Unified Libraries, 1-5, 1-31, 1-34



up arrow, 1-23, 1-27, 1-87

## V

vectors, 3-7

VHDL designs, 1-15

viewdraw.ini file, 1-53, 2-12, 3-18

VSM file, 2-22, 3-27

VSM log file, 1-117

VSM program, 1-117, 2-15, 3-21

VSMUPD

generating simulation netlist, 2-22, 3-27

## W

WDIR variable, 1-5

WIR file

input to VSM, 3-21

input to WIR2XNF, 2-14, 2-19, 3-20, 3-24

output by Check program, 2-14, 3-20, 3-21

output by XNF2WIR, 2-15

simulation, 3-22

WIR2XNF, 1-3

running with XMake, 2-19, 3-24

running with XSimMake, 2-14, 3-20

Workview 4.1.3a, 1-4

Workview PLUS, 1-4

## X

X flag, 1-108, 1-109, 1-111

XABEL, 1-15

XACT variable, 1-5, 3-10

XACT-Performance

adding timing constraints to specific paths, 4-14

adding TNM attribute to schematic, 4-10

computing delays, 4-1

creating routed design with Flow Engine, 4-25

defining sets with TNM attribute, 4-14

design implementation, 4-23

devices supported, 4-1

entering default timing specifications, 4-11

grouping sets with TIMEGRP symbol, 4-18

interaction with PPR, 4-9

purpose, 4-1

setting default timing requirements, 4-10

specifying timing constraints with TIMESPEC symbol, 4-20

syntax, 4-3

TIMEGRP symbol *see* TIMEGRP symbol

TIMESPEC symbol *see* TIMESPEC symbol

TNM attribute *see* TNM attribute

XACT-Performance report, 4-26

XACTstep Development System software, 1-4

XAS file, 3-10

X-BLOX

adding module to PROcapture schematic, 2-3

attributes, 2-6, 2-7

buses, 2-5, 2-6, 2-7, 2-12, 2-16

creating routed design with Flow Engine, 2-20

design implementation, 2-18

examining XSimMake output, 2-13

functional simulation, 2-11, 2-15, 2-16

macros, 2-9, 2-10

schematics, 2-10

symbol library, 2-9

symbols, 2-16, 2-20

ADD\_SUB, 2-9

ANDBUS2, 2-10

DATA\_REG, 2-9

MUXBUSx, 2-10

ORBUS2, 2-10

XORBUX2, 2-10

- timing simulation, 2-22, 3-26
- X-BLOX modules, 1-117
- XC3000 demonstration board, 1-3, 1-196
- XC3000A demonstration board, 1-2, 1-3, 1-100, 1-103, 1-192, 1-194
- XC4000 demonstration board, 1-192
- XChecker cable, 1-192, 1-193
- XChecker program, 1-195
- xchecker.pro file, 1-196
- XCLK signal, 1-123
- XDelay, 2-21, 3-25
- XDraw, 2-15, 3-21
- XEMake, 1-150
- XEPLD optimization software, 1-120
- XFF file, 2-14, 2-18, 2-19, 2-20, 3-24
- XFind, 2-14, 3-20
- XFW file, 2-14
- XG file, 2-21
- Xi-BLOX
  - creating routed design with Flow Engine, 2-20
- Xilinx ABEL
  - ABEL-HDL file *see* ABEL-HDL file
  - ABL2XNF, 3-10
  - bus vectors, 3-7
  - compiling ABEL-HDL file, 3-10
  - creating routed design with Flow Engine, 3-24
  - creating Viewlogic symbol, 3-11
  - design implementation, 3-22
  - examining XSimMake output, 3-19
  - functional simulation, 3-17, 3-18, 3-21
  - replacing block with Xilinx ABEL module in PROcapture, 3-12
  - simulator, 3-7, 3-9, 3-22
  - test vectors, 3-7, 3-9
  - timing simulation, 3-26
  - verifying symbol attributes in PROcapture, 3-16
  - verifying symbol type in PROcapture, 3-15
  - viewing schematic in PROcapture, 3-14
- Xilinx ABEL modules, 1-117
- Xilinx project, 1-142
- Xilinx Property Initialstate statement, 3-8
- XMake, 1-150
  - output, 2-18, 3-23
  - running WIR2XNF, 2-19, 3-24
  - running X-BLOX, 2-20
  - running XDelay, 2-21, 3-25
  - running XNFMerge, 2-19, 3-24, 4-15
- XNF file
  - FILE attribute, 3-16, 3-22
  - input to XNF2WIR, 3-21
  - input to XNFMerge, 2-19, 3-24
  - merging, 2-19, 3-24
  - output by ABL2XNF, 3-10, 3-14, 3-17
  - output by LCA2XNF, 2-21, 3-25
  - output by WIR2XNF, 2-14, 2-19, 3-20, 3-24
  - output by X-BLOX, 2-15
  - output by XNFBA, 2-21, 3-26
- XNF2WIR, 1-3
  - outputs, 2-15, 3-21
  - running with XSimMake, 2-15, 3-21
- XNFBA, 2-21, 3-26
- XNFMAP, 2-21, 3-25
- XNFMerge
  - running with XMake, 2-19, 3-24, 4-15
  - running with XSimMake, 2-14
  - TIMEGRP errors, 4-7
- XNFPrep, 2-14, 2-20, 3-25, 4-7
- XORBUS2 symbol, 2-10
- XSF file, 3-10, 3-11
- XSimMake
  - creating functional simulation network, 1-117, 2-11, 3-17
  - creating simulation directories, 2-12, 3-18
  - creating timing simulation network, 2-22, 3-26
  - invoking, 2-12

output, 2-13, 3-19  
programs run automatically, 3-20  
running Check program, 2-14, 2-15, 3-20, 3-21  
running VSM, 2-15, 3-21  
running VSMUPD, 2-22, 3-27  
running WIR2XNF, 2-14, 3-20  
running XDraw, 2-15, 3-21  
running XFind, 2-14, 3-20  
running XNF2WIR, 2-15, 3-21  
simulating designs with X-BLOX modules, 2-11, 2-23  
simulating designs with Xilinx ABEL modules, 3-17, 3-27  
xsimmake.out file, 1-176  
XTF file, 2-21  
XTG file, 2-20, 3-25  
**Z**  
zooming, 1-22, 1-23, 1-28, 1-182

