XILINX Interface Guide

Introduction

Purpose

The purpose of this Guide is to familiarize you with ACTIVE-CAD operation and introduce you to new design methodologies, which are provided by tools based on patented incremental compilation method.

Features

ACTIVE-CAD is based on a patented incremental design technology which makes all design changes instantly available in the simulator. Also, all operations in the simulator are visible in the schematic and HDL editors. ACTIVE-CAD is a tightly integrated design environment with many time saving features:

- Designed for Windows, ACTIVE-CAD conforms to the industry standards, which simplifies its use.
- Design Manager automatically finds all EDA tools that can be used for handling the selected technologies or FPGA parts, and creates a control chart for launching different applications.
- Design Manager automatically launches many EDA tools and operations, freeing you from learning these applications and saving on manual operations.
- ACTIVE-CAD is an integrated Windows design environment that allows mixed design entry formats (schematics, FSM, HDL), logic synthesis, simulation, and has direct interfaces to other EDA tools.
- At the core of AACTIVE-CAD is the real-time interactive SUSIE simulator that has been in the industry for over ten years and has an established reputation for speed and quality.
- Patented selective simulation process allows simulation of selected signal paths and design sections at speed which is not burdened with the overall design size.
- Incremental compiler allows for real-time design breadboarding. Design blocks can be instantly isolated from other design sections and subject to local signal stimuli. Signal paths and feedbacks can be manually controlled in real-time while the simulation is in progress.
- Patented virtual hardware™ technology allows for design emulation and hardware modeling on event-by-event basis at exceptionally low cost.
- Designed for workgroups and network applications, ACTIVE-CAD is a new generation of EDA tools.

Benefits

Because ACTIVE-CAD is a tightly integrated design environment, you can do complex designs in a fraction of time. You also do not need to learn various EDA tools because they are automatically controlled from the Design Manager.

- ACTIVE-CAD allows quick start because you do not need to learn various EDA tools
- Because ACTIVE-CAD has been tightly integrated with other EDA tools, it produces quality designs with a minimum effort.
- Since ACTIVE-CAD allows mixed-mode designs (HDL, FSM and schematic editors), you can learn the new design technologies in an incremental manner.
- Because ACTIVE-CAD works with UNIX-based EDA tools installed on corporate networks, you can transition from UNIX to Windows NT tools in an incremental and safe manner.
- ACTIVE-CAD is the lowest cost EDA solution. You can afford to put it every engineering desk for higher corporate productivity.

Where to Get More Information

For more information, consult the following manuals:

- Schematic Editor User’s Guide
- Logic Simulator User’s Guide

- State Machine and HDL Editors
- Xilinx Applications
- VHDL Shorthand
- An Introduction to Simulation And Virtual Hardware

These manuals, with the exception of the last two, are available on the ACTIVE-CAD CD ROM.

Dynamic help in schematic editor and other applications will provide you with additional guidance on how to use the key ACTIVE-CAD features.

Differences Between This and a Full Version of ACTIVE-CAD

The Xilinx version of ACTIVE-CAD has all the power of the full blown software. To minimize the product cost, some features that are not needed for Xilinx applications have been removed. Below is the list of the items not available in the low cost Xilinx solution:

- Symbols for TTL, ECL, CMOS, MPUs, etc.
- System level simulation
- Interfaces to p.c.b. layout software
- Interfaces to EDA tools which are not directly supported by Xilinx.

The Xilinx version of ACTIVE-CAD should allow you economical and reliable development of Xilinx designs. None of the powerful ACTIVE-CAD features have been removed, except those mentioned above.

Installation

Requirements

RAM

- 12 Mbytes for Windows 3.11
- 8 Mbytes for Windows 95
- 460 Kbytes of conventional memory for Btrieve
- Windows swap space is 8 Mbytes for ACTIVE-CAD AND 16 Mbytes for Foundation

Software

- DOS 5.0 or later, e.g. 6.21 or 6.3
- MS Windows 3.1 or later
- Win32S 1.25.142.0 for Foundation; it is provided on the Xilinx CD ROM (must be installed before ACTIVE-CAD!)
- Win32S 1.30 for ACTIVE-CAD (for FSM and HDL editors)

Read for reference pages A-2 and A-3 of the Xilinx Development System

Hardware

- 486 or Pentium class PC
- 30Mbyes to 120 Mbytes of HD space for ACTIVE-CAD
- 80-180 Mbytes of HD space for Foundation

Installing ACTIVE-CAD

- You need to install first XACTstep Ver. 6.0. You need to have installed WIN32S 1.25.142.0 before installing the Foundation software. If the XACT software is not installed, you will get an error message (*set XACT=C:\XACT*) when attempting to install ACTIVE-CAD.
- To install ACTIVE-CAD under Windows 3.11, select the **Run** option from the **File** menu. When the entry window appears, enter the CD ROM path and *install.exe* (e.g. *D:\install.exe*)
- To install ACTIVE-CAD from a local CD ROM drive under Windows 95, select the **My Computer** icon. It will show an icon for the local CD ROM. If the **Autorun** option is set to Enable then activating icon will automatically start installing ACTIVE-CAD. If the **Autorun** option is disabled, double-click on the CD ROM icon and it will automatically install all ACTIVE-CAD software (you may need to respond to a few prompts).
- To install ACTIVE-CAD from a network CD ROM drive (under Windows 95), select the **Run** option from the **Start** menu and enter the CD ROM path (e.g. *P:\active\media\install.exe*).

Installing XACT 6

XACT Variable setting

ACTIVE-CAD automatically invokes the XACT tool. However, it needs to know its location. That location is written into AUTOEXEC.BAT file during installation of the XACT software. If you need to modify the XACT software location on network, enter into the AUTOEXEC.BAT file the new directory. Use the format:

set XACT=n:\full_path\exact

For more information refer to Appendix C of the Getting Started & *Installation Guide* for XACTstep, October, 1995.

How XACT programs are executed

All XACT programs are executed in the background. Clicking on any button in the Design Manager will automatically invoke all EDA tools that are involved in completing the selected design path. For example, activating **SIM Timing** button, will automatically invoke the XACT software. If the design has been entered in ABEL, then the process will start with invoking the X-ABEL software, etc.

Windows 95 compatibility

ACTIVE-CAD is compatible with WINDOWS 3.11, WINDOWS 95 and WINDOWS NT. It also operates in conjunction with XACT 5 and XACT 6 software. However, exiting XACT 6.0 causes in the present release an application error that affects only the XACT 6 software and does not destroy the generated files so that you may continue the design process.

Installing XACT 5

The XACT 5 software is DOS-based. Since ACTIVE-CAD invokes it in the background mode (batch process), you generally will not be aware that it is a non-Windows product.

To run XACT 5 compilations, you need to create the design in the XACT 5 format. You can do it by invoking the NEW PROJECT option in the Design Manager and selecting the XACT5 option from the menu.

Keylock

Sentinel driver installation

The Sentinel keylock driver (REV. 5.1.4) is automatically loaded when installing ACTIVE-CAD from the CD ROM. For the keylock driver to work properly, you need to reboot the computer.

Multiple keylocks

If you have several keylocks on your computer, the keylock software will check in sequence:

1. ACTIVE-CAD keylock; if it present, then all options and libraries in this keylock will be enabled.
2. Xilinx keylock if it is present, it will enable the Xilinx version of ACTIVE-CAD. However, if ACTIVE-CCAD keylock was also detected, then additional options and libraries will be allowed.
3. If neither ACTIVE-CAD nor Xilinx keylock were detected, then the minimum configuration of ACTIVE-CAD for Xilinx applications will be enabled.

Keylock Utility

The keylock utility software lists the keylock number and all options and libraries allowed by the keylock. It may also list expiration date of the software. The keylock can be upgraded with additional options and libraries, and the expiration date of the software can be changed by entering the appropriate code provided by ALDEC, Inc.

On-Line Documentation

Acrobat installation

The Acrobat software is automatically loaded during the ACTIVE-CAD installation.

Navigating documents

After the installation, you will find in the **Start/Programs** menu the **Adobe Acrobat 2.0** option., which provides a tutorial on navigating the documentation.

Searching for topics

To search for a desired topic, activate the Acrobat Reader 2.0 from the icon. Next, select the **Find** option from the **Tools** menu. When a window opens, enter the topic name and click on the **Find** button. If you need additional locations of the topic, select the **Tools** menu and select the **Find Again** option.

On-Line application Help

Each application has a **Help** menu, which is placed at the top of the screen and to the right of other menus. The help is comprised of three options: **Contents**, **Help on Help** and **About…** Clicking on **Contents** button displays the contents of the on-line help menu. At the top of the menu there are six buttons which speed help operations:

Context displays context of the menu
Search displays a window for searching key words of interest
Back returns you to the previous screen
Print prints the current topic
>> button advances display to the next page of the manual
<< button returns display to the previous page of the manual

Help on Help option explains how to use the help features.
About option displays:

- Software logo and trademarks
- Software name and version
- System data (e.g. *Windows 95, 14.9 Mbytes available physical memory, 66% free GDI resources*)
- User data includes the user name, company and keylock number

Context -sensitive Help

Context sensitive help is available in Design Manager, State Machine Editor and HDL editor. Clicking on the icon with arrow and question mark, located right below the **Help** menu, will generate a question mark cursor. Placing the cursor on an item and clicking the mouse button will display local help menu or brief description of the selected item.

Getting Help

Technical support numbers

- Tel. support (805-499-6867) is available to users having direct maintenance agreement with ALDEC. Users who bought the ACTIVE-CAD software from and OEM such as Xilinx, should contact their OEM technical representative. The support is provided from 8:00 A.M. PST till 5:00 P.M. PST.
- FAX (805-498-7945) support is operational 24 hours and it is provided to users having direct maintenance agreement with ALDEC.

Bulletin board

- BBS; tel. **805-498-4086 (USA)**
- Self-adopting modem accepts data rates from 2,400 bauds to 28.8 kbauds.
- The data should be sent as 8 bit, NO STOP and ONE (1) PARITY bit

Internet sites

WWW www.aldec.com

E-mail support

support@aldec.com

Project Manager

The Project Manager is an essential tool for managing ACTIVE-CAD projects and applications. The Project Manager is designed to:
- create and manage project files,
- verify project integrity (dates of modification, etc.),
- control project resources (schematics, netlist, test vectors, libraries, etc.).
- run ACTIVE-CAD applications,
- provide interface to other vendors' applications (e.g., to Xilinx XACT).

Starting Project Manager

To start the Project Manager double-click its icon located in the ACTIVE-CAD group. ACTIVE-CAD always works in association with a single project. This means that the Project Manager forces you to specify the current project. By default, it is the project you were working on during the last session with ACTIVE-CAD. If for some reasons such project cannot be found, the Project Manager will prompt you to either open another project or to create a new one.

Structure of the project

Each ACTIVE-CAD project is stored in a separate subdirectory, called *project directory*. The project directory contains all files, called *project resources*, that were created while working on the project. Project resources may comprise, among others, schematics files, netlists, HDL files, test vectors, PLD jedec fuse maps, memory files, simulator macros (command files), project libraries, etc. By default, all project directories are stored in the directory \\ACTIVE\PROJECT. The name of a project directory name is the same as the name of the appropriate project (Therefore, project name can consists of up to 8 characters).

For each project directory, ACTIVE -CAD creates a file called *Project Description File*. A PDF file contains description of the contents and the current status of the appropriate project. PDF files have the same names as the corresponding project directories, supplied with the extension **.pdf**. PDF files are kept in the same directory as project directories.

The PDF file **current.pdf** has special meaning for ACTIVE-CAD. This file describes the current project, that is, that one you are just working on. When you open an existing project (see the next section), ACTIVE-CAD copies its PDF file into the **current.pdf** file. As long as you do not load another project, ACTIVE-CAD writes all information about the current project into the **current.pdf** file, whereas the actual PDF file remains unchanged. The moment you open another project, ACTIVE-CAD copies the **current.pdf** file into the PDF file belonging to the project being closed. The Project Manager also updates the original PDF file on exit.

NOTE: You can view the PDF file by double-click on the PDF file name in the hierarchy browser.

Managing projects

Creating a new project

To create a new project select the **New Project** option from the **File** menu. In the dialog that appears specify parameters of the new project. These parameters comprise, among others, project name, project type (for projects using Xilinx FPGA & EPLD devices you should choose either XACT5 or XACT step6), and parameters specific for the given project type. In case of Xilinx type project (i.e. XACT5 or XACT step6), you must specify Xilinx family, part type, and speed. Clicking OK causes ACTIVE-CAD to create the new project directory and its PDF file. The PDF file is then copied into the **current.pdf** in order to set up the created project as the current one.

NOTE: The **Unified/Old Libraries** switch that appears for XACT5 projects for families XC2000, XC2000L, XC3000, XC3000A, XC3000L, XC3100, XC3100A, XC4000, XC4000A, XC4000H allows you to choose the version of library for the project. *Old libraries* were used by the XACT4 system. *Unified libraries* came into use with the XACT5 system. ACTIVE-CAD provides both the old and the unified libraries.

XACT step6 projects

In case of XACT step6 projects, the Project Manager perform the following additional operations:
- creates in the project directory subdirectory XPROJECT,
- creates in the subdirectory XPROJECT file **<project_name>.prj** for XACT6 system,
- updates the file **dsgnmgr.ini** (located in the Windows directory) so as to enable XACT6 to find the **<project_name>.prj** file.

These operations are executed when you run XACT6 from the Design Flow in the Project Manager.

Opening existing projects

To open an existing project select the **Open Project** option from the **File** menu. The dialog Open Project lists the names of all projects found in projects directory. Using the **directory** box you may scan other locations on the disk Select the name of the project you want to open and click OK.

NOTE: All document currently open are closed when you change a project.

Deleting a project

To delete a project you have to select the **Delete Project** option from the **File** menu. The dialog Delete Project lists the names of all projects found in projects directory. Using the **directory** box you may scan other locations on the disk Select the name of the project to be deleted and click OK. The **Delete All** check box allows you to choose deleting the entire project directory with its PDF file (box checked) or deleting only the resources files, created by ACTIVE-CAD, in the project directory (box unchecked).

NOTE: You cannot to delete the current project.

Copying a project

To copy a project you have to select the **Copy Project** option from the **File** menu. The dialog Copy Projects has two main fields. In the **Source** field specify the PDF file of the project you want to copy. By default, this is the PDF file of the current project. In the **Destination** field specify the name of the new project and the destination directory, where the new project directory and the PDF file will be created.

Changing Xilinx family

If you want to change the family and part type for the current Xilinx project you should follow the procedure described below:
1. Select the **Project Type** option from the **Menu** file. Change **Family, Part** and **Speed** settings, as desired, and click the **Change** button.
2. Open and save each schematic sheet and each schematic macro. To do so, run the Schematic Editor and select the **Open** option from the **File** menu. The Open Sheet window allows you to quickly open all projects top level sheets and project schematic macros. Inspect the Project Manager messages for any warnings and errors.
3. Re-synthesize and update all FSM and HDL macros. Use hierarchy browser in the Project Manager to search the project for the macros.
4. If the project contains memory macros, re-synthesize their XNF netlists using the **Memory Generator**.

NOTE1: If your projects contains components that are not available in the new system library, you have to modify the project so as to preserve its functionality.

NOTE2: In case of a top level HDL project, the only thing to do is to re-synthesize the entire project

Hierarchy Browser

Hierarchy browser allows you to explore the project hierarchy. Hierarchy browser is located in the upper left part of the Project Manager window. The project hierarchy is presented in the form of a tree with expandable and non-expandable items. The PDF file is placed on the top (root) of the hierarchy. Each item in the hierarchy tree is represented by a small icon and the name. A tree item can be one of the following:

- the PDF file,
- system or project library (non-expandable),
- schematic sheet (expandable),
- schematic macro (expandable),
- schematic primitive (non-expandable),
- HDL macro or top level HDL document (non-expandable),
- FSM macro (non-expandable),
- external text file.

The color of HDL macro icons and FSM macro icons brings additional information about the macro status.
- **gray background** denotes that macro status is O.K. (the source file, macro symbol, and XNF netlist exist and are consistent with each other),
- **yellow background** denotes that the macro needs to be updated,
- **yellow background and red edges** denotes that the macro source file does not exist in the project directory.
The only resources attached to the root (available on the top level in the hierarchy) in a new Xilinx project are system libraries and project library.

Expanding / collapsing hierarchy

Icons representing expandable branches are marked with the sign '+' (when collapsed) or '-' (when expanded). By clicking these icons you can easily expand and collapse branches. You may also use the appropriate options from the **Document** menu.

Starting applications from hierarchy browser

The hierarchy browser allows running applications associated with particular items of the hierarchy tree. To do so, you must double-click the name of the item in the hierarchy tree. Application starts with the selected item (schematic or source file) loaded. Here is the list of the associations.

Double-click on:	*Invoked application:*
PDF file	Windows Notepad
schematic sheet/macro	Schematic Editor
HDL macro / file	HDL Editor
FSM macro	State Diagram Editor
library	Library Manager (*The Library Manager allows, among others, viewing the library contents*)
other file	appropriate Windows application (*the application is associated with the file according to the file extension*)

Push/Pop functionality

If you entered the Schematic Editor by double-clicking a schematic macro, you may use the SC**Hierarchy/Hierarchy Pop** option to go to the upper level schematic, even if the upper level schematic is not loaded. In such case, the Schematic Editor will load the schematic automatically.

I/O port changes

If you change pin specification of a macro used in the project, all schematics containing instance of this macro will not be updated until they are loaded into the Schematic Editor. Before you proceed with the processing the project, you should update these schematics (just open them) and the project netlist.

Non-project documents

Sometimes you may want to attach some non-project documents to the project. This usually occurs when you want, for example, to attach a text file with the description of the project. To do so you have to use the**Add** option from the **Document** menu. Dialog Select Document distinguishes 4 sorts of files:
- schematic files (*.sch),
- state diagram editor (FSM) files (*.asf),
- HDL files (*.vhd and *.abl),
- other files.

The files(s) is attached to the root of the hierarchy.

Design Flow

The Design Flow is located in the upper right part of the Project Manager window. The Design Flow includes a set of buttons designed to provide a process flow control for selected target device. These flows are designed to provide automated transfer of data between various tool used to compile your designs. They look different according to the type of the current project and specific family.

Design Entry Tools

ACTIVE-CAD provides variety of applications which support various methods of description of a design. Here is the short description of each tool.

Schematic Editor
The Schematic Editor enables drawing flat and hierarchical schematics. The editor allows insertion of non-schematics macros created within other design entry tools:
- HDL macros, containing description given in Hardware Description Language (either VHDL or ABEL), created within HDL editor,
- FSM macros, containing finite state machine diagrams, created within State Diagram Editor.

HDL Editor
The HDL Editor is a tool intended for projects utilizing HDL description. Particularly, it is designed to support two languages: VHDL and ABEL. The editor has interface to synthesis tools which generate XNF netlist from HDL source code. Synthesis of HDL code, performed by these tools, consists in conversion of the RTL structure described in terms of the HDL language into XNF netlist built of Xilinx primitives.

State Diagram Editor
The State Diagram Editor supports description in the form of a finite state machine. Definition of a state machine is given in the form of a graphic diagram. Editor generates description of the machine either in VHDL or ABEL source code. This description is the base for synthesis program to generate the XNF netlist.

Memory Generator
The Memory Generator is a Xilinx application, the ACTIVE-CAD provides interface to it. The Memory Generator generates XNF netlist from a MEM file.

File version checking

To check the project integrity the Project Manager compares modification dates of files generated by Xilinx applications and the ACTIVE-CAD file <**project_name>.prj**. The files should be ordered chronologically as described below. In all cases, the oldest should be the file **.prj**.

For XACT5 projects for families other than XC7200 and XC7300:

file type	*extension*	
Schematic Netlist	ALB	
XNF Netlist	XNF	
Xmake output	LCA	
LCA2XNF output	XNR	
Backannotation output	BAX	
Routed Netlist	ALR	(the latest)

For XACT5 projects for families XC7200 and XC7300:

file type	*extension*	
Schematic Netlist	ALB	
XNF Netlist	XNF	
Xemake6 output	VM6	
Tsim output	XNT	
Routed Netlist	ALR	(the latest)

For Xilinx XACT Step6 projects for any families

file type	*extension*	
Schematic Netlist	ALB	
XNF Netlist	XNF	
XACT6 output	BAX	
Routed Netlist	ALR	(the latest)

Project Libraries

Libraries in ACTIVE-CAD can be divided into two general types. *System libraries* are provided with the system and their contents cannot be modified by the user. In addition, each project has its own *project working library* which is automatically created when you start a new project. This library stores macros created by the user. It also includes modified (by the user) versions of components from the system library (e.g. changed pins, part name or timing). The project working library and system libraries attached to a given project are referred to as *project libraries*.

If you want to have a library attached to or removed from the project use the **Libraries** option from the **File** menu. In the dialog Libraries you can easily manage all libraries available in the system.

When you create a new project of a stated type, ACTIVE-CAD automatically creates project working library and adds system libraries which are necessary for the project. For example, when you create a project based on a Xilinx FPGA device, appropriate system libraries with Xilinx primitives will be attached to the project.

NOTE1: You cannot remove the project working library.

NOTE2: You should not remove system libraries that have been attached to the project by ACTIVE-CAD. Even if you do so, during the next session with the project, ACTIVE-CAD will attach them again. This rule does not apply the X-BLOX system library, which can be removed from the project.

NOTE3: Do not attach multiple system libraries to the same project. Even if you do so, during the next session with the project, ACTIVE-CAD will leave only the library specific for the project target device.

System Libraries

ACTIVE-CAD provides system libraries supporting Xilinx device families XC2000, XC2000L, XC3000, XC3000A, XC3000L, XC3100, XC3100A, XC4000, XC4000A, XC4000H, XC4000E, XC5200, XC7200, and XC7300. These libraries contain Xilinx macros and primitives. In addition, family independent X-BLOX system library is available. It is described in the next section

X-BLOX library

The X-BLOX (Blocks of Logic Optimized for Xilinx) library contains macrocells that allow describing a project in terms of high-level functions instead of gate-level primitives. As X-BLOX modules are customizable, each macrocell can describe thousands of unique functions. You can customize these modules using attributes and suitably connecting their control pins.
X-BLOX library is family independent and supports only Xilinx FPGA families. ACTIVE-CAD provides two versions of the library: old and unified (see section *Managing projects/Creating a new project*, in this chapter). Detailed information on X-BLOX you can find in the Xilinx manual '**X-BLOX User Guide**'.

Application Message Log

When you run ACTIVE-CAD, the Project Manager displays in the lower part of its window commands and messages interchanged between applications running in the system. All information displayed in the Project Manager message box may also be stored in a log file. The **Preferences** option from the **Messages** menu allows you to customize both the contents and the way of presentation of the log.

Log settings

In the dialog Message Preferences (menu **Messages**, option **Preferences**) you can specify the following parameters:
- how to present the log (in the Project Manager message box, in the log file, in both),
- log file name,
- capacity of the buffer (number of lines) for Project Manager message box,
- the sorts of messages to be reported (errors, warnings, comments, info of level 1..3)

Log files

Beside the main log file, XACT5 and XACT6 applications create their own report files. In case of XACT5 projects these reports are stored immediately in the ACTIVE-CAD project directories. You may view these reports using the Xact Design Info dialog, which is available after clicking the **Info** button in the Design Flow. To view a report you must double-click the appropriate entry in the **Reports** section.

In case of XACT Step6 projects, the user interface of XACT6 includes appropriate options for viewing report files and you should use them to read the reports.

Schematic Projects vs. Top Level HDL Projects

ACTIVE-CAD distinguishes two fundamental project types: Schematic and Top Level HDL. Schematic projects are based on a classic schematic, built of components, both those coming from standard libraries and user-defined, connected together with wires (nets). Structure of the project may be hierarchical and consist of many levels. This means that each schematic may include lower level schematics in the form of schematic macros. For example, macros belonging to Xilinx libraries often possess multi-level structure. Beside schematic macros, ACTIVE-CAD supports two other macro types: HDL macros and FSM macros. HDL macros are based on the description expressed in terms of a Hardware Description Language. The system supports two common languages VHDL and ABEL. The FSM macros are intended for finite state machines. A state machine is encoded graphically by a state diagram.

The second class of ACTIVE-CAD projects actually comprises two subtypes: Top Level VHDL and Top Level ABEL. These projects consist solely of HDL source files, which describe the functionality of the entire projects. In case of VHDL, a project may contain more than one source file. This release of ACTIVE-CAD supports ABEL projects containing only one source file. As Top Level HDL projects do not have ALDEC binary netlist, the functional simulation is based on the synthesized XNF netlist.

Adding Top Level documents

Each project is described either by a top level schematic (which can comprise one or more sheets) or an HDL file (or files). A document(s) of one of the kinds mentioned above must be attached to the uppermost level of the project hierarchy, i.e., to the root. Otherwise the project is empty.

The type of the documents attached to the root of the hierarchy determines the type of the projects. If you run Schematic Editor with an empty project, a blank sheet will be automatically attached to the hierarchy root. This way, you will set the type of the project as schematic.

Similarly, If you run the HDL Editor and attach the source file to the project hierarchy, you will set the project type as either Top Level VHDL or Top Level ABEL (according to the language of the document).

Functional Simulation of Top Level HDL Projects

This release of ACTIVE-CAD does not support simulation of source VHDL and ABEL code. Functional verification of a Top Level HDL project is possible only be means of simulation based on the synthesized XNF netlist. If the netlist contains parametrized X-BLOX macrocells, which do not have simulation models, the Project Manager invokes the X-BLOX process which produces the XNF netlist consisting of primitives that can be simulated. In case of Top Level ABEL projects based on XC7200 or XC7300 devices, the netlist may also include components for which simulation models do not exist. In such case, Xilinx application, TSIM, is invoked in order to generate the XNF netlist consisting of primitives that can be simulated.

Documents sequence

The order in which the documents appear in the hierarchy tree is important in a few situations. Firstly, sequence of schematic sheets determines the order in which they will be printed in the Schematic Editor. Secondly, the order in which the source files are attached to the hierarchy root of a Top Level VHDL project determines the order in which these files will be analyzed by XVHDL compiler. This issue is discussed in detail in the section *Top Level VHDL projects*, in the chapter *Using VHDL.*

NOTE: You can easily rearrange the items in the hierarchy tree using the drag-and-drop method.

Managing Project Documents

Adding a file

Top level documents may be added to the project in two ways. In the first method, project files are added by design entry tools. The Schematic Editor adds a blank sheet automatically when the project is empty, i.e. does not include any top level project files (schematic sheet or HDL file). Using the HDL Editor you can add the edited file to the hierarchy root choosing the **Add to Project** from the HDE **Project** menu.
option
In the other method, you add existing project files using the **Add** option from the Project Manager's menu **Document**. Remember, that the type of the documents attached to the hierarchy root determines the type of the project. Therefore, you cannot add top level documents of assorted types.

Removing files

You can remove top level documents using the **Remove** option from the **Document** menu. Removed files are not physically deleted but remain in the project directory. If you wish to attach them back, you should use the **Add** option from the **Document** menu.

Adding non-project files

The Project Manager allows attaching also not-project files to the hierarchy root. All files with extension other than **.sch, .vhd, .abl** and **.asf**. are treated as external, i.e., non-project files. The files can be easily viewed and edited by the associated application (see section *Hierarchy browser/starting applications from hierarchy browser*, in this chapter). To do so, double-click on the name of the file to be edited. As a result, the appropriate Windows applications will start with the file loaded.

Drag and drop functionality

The Project Manager allows you to rearrange the order the files appear in the hierarchy tree. This can be achieved with mouse using the drag-and-drop method.

Document Info

You can get information about any file from the hierarchy tree. To do so, use the **Info** option from the **Document** menu. The info box shows the type of the document, library (in case of components), last modification date, and source file (in case of schematic sheets, HDL macros, FSM macros). Each item in the tree belongs to the one of the following categories:
- project description file,
- external document (*all text files attached to the root*),
- state editor diagram,
- HDE document,
- schematic editor sheet,
- library symbol (*library primitives*),
- library macro,
- library,

Project Type

Every project belongs to one of the following categories:
- **schematic**, the project whose top level document is a flat or hierarchical schematic (consisting of one or multiple sheets),
- **VHDL**, (**Top Level VHDL)**, the project whose top level document is a source VHDL file (or files),
- **ABEL**, (**Top Level ABEL**), the project whose top level document is a source ABEL file.

The project type designator, as given above (**schematic**, **VHDL**, **ABEL**), is displayed in the info box Project Info. To see this box, select the **Project Info** option from the **File** menu. The box also gives information about the date of creation, project directory, and the netlist file.

Upgrading Xilinx devices list

If necessary, you can add new devices to the Xilinx device list provided with the system. The list is kept in the file **xilinx.fam**, located in the directory containing executable files (by default \\ACTIVE\EXE). The file includes several sections. Each section begins with a header enclosed in square brackets. The first section, **[Family]**, defines all Xilinx families available in the system. Subsequent sections describe particular families. The header of each section is a family designator.

If you want to add a new device (or new speed to existing device) to a given family, you should find the appropriate section and add a new entry (or update existing). Each entry looks like the following:

```
<device symbol> = speed1, speed2, ...
```

For example, the section for XC2000 family begin with the following lines:

```
[XC2000]
2018PC44=130, 100, 70, 50, 33
2018VQ64=130, 100, 70, 50, 33
......
```

Each entry can include white spaces – they are ignored when the file is read.

NOTE: You should not modify the **xilinx.fam** file while the ACTIVE-CAD is running.

Finding Objects in the design

The Project Manger provides a simple tool which enables finding symbols, nets, and pins on schematic sheets or schematic macros. To invoke the **Find in the Schematic Editor** dialog choose the **Find Object** option from the

Document menu. This tool co-operates strictly with the hierarchy browser (in the Project Manger) and the Schematic Editor. Results of searching are reported in the Project Manager's message box and in the Schematic Editor.

Finding schematic symbols

If you want to find a symbol currently selected on the hierarchy tree, set the **Symbol** option in the **Find What** field. Then click the selected item on the hierarchy tree, to update the contents of the **Symbol Reference Path** edit box. Clicking **Find** causes the Schematic Editor to load the appropriate sheet and select the specified symbol.

Example:
Open the ***CALC*** sample project. Invoke the window **Find in the Schematic Editor**. Set **Find What: Symbol**. In the hierarchy browser expand the ***CALC.SCH*** sheet and then, the ***STACK0*** schematic macro. Select the ***RAM16X4*** macro. Click the **Find** button. The Schematic Editor ill be invoked and loaded with the schematic macro ***STACK_4K***. The macro symbol with the reference name ***RAM16X4*** will be selected.

If you directly type the symbol reference name (without the full path) in the **Symbol Reference Path** box, the symbol will be searched for in all top level sheets. You cannot use this method to find a symbol in schematic macros.

Finding nets and pins

Find Object allows you to finding nets and pins on a specified schematic or schematic macro. To do so, first, specify the path in the **Symbol Reference Path** edit box. You can achieve that in the same way as described in the previous section. Then change the **Find What** setting to either **Net** or **Pin**. Type the net or pin name in the appropriate edit box and click **Find**. If the net or pin exist in the specified location, the Schematic Editor will be loaded with the appropriate sheet, and the item will be selected.

Example (continuing from the previous section):
Switch to the Project Manager, and set **Find What: Net**. Type in the **Net Name** box ***'WE'***. Click the **Find** button. The Schematic Editor will load the schematic macro ***RAM16X4***. The ***WE*** net will be selected.

If you do not specify the symbol reference path, the specified item (net or pin) will be searched for in the top level sheets. You cannot use this method to find an item in schematic macros.

Starting Applications from Project Manager

ACTIVE-CAD application can be started from the Project Manger in a few ways.
All applications can be started be selecting the appropriate item from the **Applications** menu or by clicking the icon in the toolbar (except the Memory Generator and the Keylock Utility, for which such icon are not provided).

In addition, the Schematic Editor, the HDL Editor and the State Diagram Editor can be started:
- by double-clicking the appropriate item in the hierarchy tree, associated with a given application (see section *Hierarchy Browser/Starting applications from hierarchy browser* in this chapter),
- by clicking the appropriate buttons in the Design Flow.

The Simulator can be started also from the Design Flow. In case of Xilinx Projects there are always two buttons available, which invoke the Simulator: **SIM Funct** and **SIM Timing**. It is recommended to use these buttons because they automatically put the Simulator into the appropriate mode (functional or timing, respectively) and load the appropriate netlist.

Archive project option

ACTIVE-CAD provides the option which enables user to store the projects in the form of a compressed (zipped) archive file. The archive file includes all files kept in the project directory, PDF file, current PDF file, and some additional files which describe the state of the system:

autoexec.bat,
config.sys,

susie.ini,
bti.ini,
current.ver.

NOTE: You must have the PKZIP.EXE program installed on your computer to use this option. Configuration options (click the **Configuration** item in the **File** menu to get the Configuration window) allow to set the path to PKZIP.EXE.

Customizing Project Manager

Configuration options are available in the Configuration dialog window (menu**File**, option **Configuration**). You can specify the drive paths for files used by ACTIVE-CAD, and the editor used to view text documents. Clicking on the **View Ini File** allows you to view the **susie.ini** file which determines the ACTIVE-CAD system settings.

Creating Xilinx Schematic

Naming Conventions

FPGA names for nets, buses, and symbols must follow these conventions:
Only A-Z, a-z, 0-9, "_," and "-" are allowed in user-defined names. No other characters should be included in names.
Names must contain at least one non-numeric character.
Names cannot be more than 1024 characters long.
Hierarchical symbol names cannot be longer than 8 characters. Otherwise the DOS netlist file cannot be created with the same name as the symbol.
Symbol pins and macro pins must be no longer than 8 characters.

Reserved Names

The physical names associated with every resource on every part are reserved and cannot be used to name signals and symbols. These include CLBs, IOBs, clock buffers, BUFTs, oscillators, package pin names, CCLK, DP, GND, VCC, M0RT, PWRDN, and RST. Other examples are CLB names such as AA and AB, pin names such as P1 and P2, pad names such as PAD1 and PAD2, and primitive names such as TDO, BSCAN, M0, M1, M2, or STARTUP.

Net Naming Conventions

Hierarchical signal names are fully specified in the XNF file. Here are some examples:
Unlabeled signals are given internal names generated automatically by ACTIVE-CAD that consist of a dollar sign, "N" for net or bus, and a unique number assigned by ACTIVE-CAD for each net. Any change to the schematic changes at least some of the ACTIVE-CAD-assigned internal names.
ABC represents a labeled signal named ABC in the top-level drawing.
$1I11/ABC represents a labeled signal named ABC that is underneath an unlabeled component called $1I11, where $1I11 is the symbol reference designator named with a dollar sign.
$1N118 represents net $1N118, which is on sheet 1 of the root-level drawing.
$1I5/$1N118 represents net $1N118, which is a net within the
top-level symbol $1I5. If the net is within a schematic represented by a symbol on the design's top level, the default signal name reflects this hierarchy.
As these examples clearly show, the more labels you put in your design, the easier you will find it to locate signals for simulation. For more information on adding labels to nets, see the 'Naming nets' section of this guide.

Component names

To give components more meaningful names than those issued by ViewDraw, use the Add Label command to name symbols, just as you would nets. The following are examples of symbol names.
MYSYM0 is a component located at the top level of the drawing.
TOP/MYSYM0 is a component located one level below TOP.
Components with or without user-assigned labels are assigned names as follows:
top-level_instance/instance
For example, $1I3/$1I5 represents a component located one level below symbol $1I3. For more information on adding labels to components, see the 'Changing symbol reference names' section of this guide.

Naming buses

To ensure that bus signals are processed correctly, use the following naming conventions.
All buses and all nets going into buses must be labeled. For example, a bus labeled A[0:2] should have nets labeled A0, A1, and A2.

ACTIVE-Cad netlist export expands bus notation. All bus and symbol pin names are expanded into individual signal or pin names. For example, a bus labeled DATA[0:3] is converted into four nets labeled DATA0, DATA1, DATA2, and DATA3.
You must be consistent in the order of bus indices for a single bus. For example, do not connect busa[0:3] to busb[3:0] at another level of your schematic unless you are deliberately reversing the bus order.

Creating a new schematic

New schematic is always created during the project creation. To add a new schematic sheet press the **New Schematic** button on the toolbar or selecting **New Sheet** from the **File** menu. The new schematic is given the name equal to the project name with sequential numbers added at the end. If that name is not suitable, the schematic can be saved under new name by invoking **File | Save As** command.

Selecting Libraries

To select libraries used by your project, invoke **Project Libraries** command from the **File** menu. Select the required libraries in the **Attached Libraries** list and click on the **Add >>** button; selected libraries are copied to the **Project Libraries** list. To remove a library from the **Project Libraries** list select it and click on the **<< Remove** button.

Xilinx Libraries

The Unified Libraries conform to standards set for the appearance, function, and naming conventions of the library elements. This standardization allows you to easily convert from one Xilinx architecture to another. The primitives and macros in the Unified Libraries should be used to create new designs. Creating new designs from previous libraries is not recommended. It is also not recommended that you mix components from the old libraries and the Unified Libraries in the same design. Refer to the *XACT Libraries Guide* for detailed information on the Xilinx libraries.

Primitives and Macros

The Unified Libraries contain three types of components: primitives, soft macros, and relationally placed macros (RPMs). Primitives are those symbols recognized directly by the implementation software such as pads, gates, latches, flip-flops, buffers, and oscillators. Soft macros are schematics that contain primitives and other soft macros. Soft macros have pre-defined functionality but have flexible mapping, placement, and routing. RPMs, available for XC4000 devices, are soft macros that contain placement information and that can contain carry logic elements.

X-BLOX

The X-BLOX library contains module generators that describe a system using high-level functions instead of gate primitives. The X-BLOX synthesis tool processes these modules. The X-BLOX library can be used only with XC3000A/L, XC3100A, XC4000 and XC5200 FPGA designs.
The X-BLOX library is supplied with ACTIVE-CAD, but the X-BLOX software required for processing designs with X-BLOX symbols is not. The X-BLOX software is included in the standard and extended solutions packages and is also sold separately.

Libraries Guide

ACTIVE-CAD libraries contain the basic information on each symbol; that information can be seen in the **SC Symbols** window above the selected symbol name. To review the detailed description of the symbols, please refer to the XILINX documentation provided in the Acrobat Reader format.
The **Symbol filter** button in the **SC Symbols** window allows selecting symbols by the attributes (tags that specify the function of the symbol). Checking suitable attribute checkboxes in the **Library filter** window selects for display only the symbols with specified attributes.

Navigating schematic window

If the schematic does not fit in the editor window, standard windows scrollbars appear at the right and/or bottom edge of the window. To see the invisible part of the schematic, click on the arrow keys on the scrollbars or drag the rectangles placed on the scrollbars between the arrow keys.
The **Center** command from the **Display** menu (or **F9** keyboard key) can be used to navigate the schematic, too. After selecting the command, moving the bull's eye mouse pointer to the edge of the editor window scrolls the schematic so that the part behind the edge becomes visible. Clicking the right mouse button cancels the **Center** mode.

Placing Symbols

To place symbols on the schematic click on the **Symbol Toolbox** button, press the **F3** key on the keyboard or select **Symbol** option from the **Mode** menu. In the **SC Symbols** window select the desired symbol from the list or type its name in the lowermost field. When the mouse pointer is moved out of the window, a symbol outline is visible

Placing I/O pins

Using I/O terminals

The I/O terminal symbol can be placed on the top level schematic to represent I/O pin; it works the same way as the PAD symbols. However, to maintain compatibility with XILINX libraries using PAD symbols is recommended.

Using PAD symbols

The pad is a special symbol representing the pin of the XILINX chip. There are three types of pads:

- IPAD for input pins,
- OPAD for output pins,
- IOPAD for bi-directional pins.

There are also symbols representing groups of 4, 8 and 16 pins; their names begin with the pad type (as listed above) and end with the number of pins in the symbol. For example, the IPAD8 symbol represents eight input pins, the OPAD4 symbol represents four output pins.
Please remember that a buffer symbol must be placed between the pad and the rest of the circuit to ensure proper placement and routing.

Placing I/O pins in hierarchical macros

I/O pins can be put inside the hierarchical macro by placing the PAD symbol on the macro schematic.

Drawing nets

To draw nets, click on the **Draw wires** button on the vertical toolbar, press **F4** key on the keyboard or select **Draw Wires** from the **Mode** menu. When in **Draw Wires** mode, move the mouse pointer to draw temporary segments of the net; click the left mouse button to fix the next net segment; click the right mouse button to suspend drawing mode.

Starting net from pins

To start drawing net from a pin, click on the pin while in **Draw Wires** mode. Schematic Editor anchors the net at the pin end.

Starting net in empty space

To start drawing net in empty space, click on the desired start point while in **Draw Wires** mode. Schematic Editor draws **UNCONNECTED** symbol (blue filled circle) and anchors the net at that point.

Ending nets in empty space

To end drawing a net in empty space, click the right mouse button to suspend drawing mode and press the **End** button on the horizontal toolbar. Schematic Editor draws **UNCONNECTED** symbol (blue filled circle) at the end of the net.

Ending nets with net name

To end a net with net name, click the right mouse button to suspend **Draw Wires** mode and click on the **Add Net or Bus Name** button on the vertical toolbar. Schematic Editor displays **Net Name** window; enter desired name in the **Net Name** field and click on the **OK** button. Net name is displayed at the end of the net.

Built-in net autorouting

Schematic Editor is by default in the Autorouting mode; to finish drawing a net you need only to click on the end point - remaining segments of the net are drawn automatically without crossing existing objects on the schematic. To disable this feature, uncheck **Autorouting** box in the **Wire & Bus Settings** window displayed by the **View | Preferences | Wires and Buses** command.

Connecting named nets

Only unnamed nets can be freely connected with each other - connecting two different named nets is not allowed. To force connection between such nets change one of the net names to the same as the other.

Naming nets

New nets drawn in the Schematic Editor are given the automatic name of the *$NetXXXXX_* format, where XXXXX is a unique number. Nets with names beginning with $ (dollar sign) are treated as unnamed during editing operations. However, named nets are required in certain circumstances.

Adding name to a net

The user can add names to the nets in two ways:

- After clicking on the **Add Net or Bus Name** button the name of the net can be entered in the **Net Name** field of the **Net Name** window. Pressing the **OK** button places floating rectangle at the mouse pointer; clicking on the net names the net and places the net name at the click point.
- Double clicking on the net (or clicking on the **Properties** button) also invokes the **Net Name** window. After entering the name in the **Net Name** field and pressing the **OK** button the name is displayed near the mid-point of the wire.

Changing net name

The name of the selected net can be changed by double clicking on the net or clicking on the **Properties** button. The new name should be entered in the **Net Name** field of the **Net Name** window. After clicking on the **OK** button the new name is updated on the schematic.

Placing net name without wire

Net name without a wire (also called a label) can be placed on the schematic. To place a label click on the **Add Net or Bus Name** button and enter the name in the **Net Name** field of the **Net Name** window. After pressing the **OK** button, click on the place where the label is to be placed.

Labels can be used for setting probes or as a starting point of the nets.

Drawing buses

To draw buses, click on the **Draw buses** button on the vertical toolbar, press **F5** key on the keyboard or select **Draw Buses** from the **Mode** menu. When in **Draw Buses** mode, move the mouse pointer to draw temporary segments of the bus; click the left mouse button to fix the next net segment; click the right mouse button to suspend drawing mode. After pressing the right mouse button, drawing a bus can be finished by pressing the **End** button on the horizontal toolbar or by adding bus terminal (button on the vertical toolbar).

Naming buses

New bus drawn on the schematic has NO NAME attribute and default bus range (set in the **Wire & Bus Settings** window in the **View | Preferences | Wires and Buses** command). To change bus name and range double click on the bus and enter new name and range in the **Edit Bus** window; click on the **OK** button to confirm changes. Bus tap shape and bus terminal type can also be set in the **Edit Bus** window.

Pin to pin connections

You cannot connect two bus pins by placing symbols so that pin tips are connected. Drawing a bus joining two bus pins is required to establish electrical connection.

Placing bus taps

Connection between a bus and a single pin or net is established via bus taps. To place single bus tap enter the **Draw Wires** mode, click on the bus in the point where the tap is to be placed and draw a wire connecting tap with the pin or net. Remember to name the tap you have placed (see below).

Naming existing bus taps

Bus tap must be given the correct name - equal to the expanded name of the one of the bus members. To name the bus double click on the tap (or the wire that is connected to it) and enter the name in the **Net Name** window.

Automatic naming and drawing

To facilitate drawing bus taps, press the **Draw bus taps** button on the vertical toolbar or press **F6** key on the keyboard or select **Draw Bus Taps** from the **Mode** menu. When in **Draw Bus Taps** mode, click on the bus name on the schematic - the name of the first bus member is displayed at the bottom of the Schematic Editor window. After clicking on the pin a tap connecting that pin with the selected bus member is drawn automatically and the next bus member name is displayed. The other bus member can be selected by pressing up or down arrow keys on the keyboard. Pressing **Esc** key cancels **Draw Bus Taps** mode.

Moving bus taps

Bus taps can be dragged with the mouse to the new position. Because bus tap is a tiny object, zooming in is suggested prior to moving a tap. It helps to avoid accidental selecting of the net or bus instead of the tap.

Power and Ground

Using power nets

Power nets are used for tying hanging pins to the fixed logic level. Note that tying control pins to the VCC or GND may influence the behavior of some XILINX primitives during the global set/reset operation

Using Power symbols

To connect a pin to the power symbol:

- draw short wire starting from the pin,
- suspend drawing by pressing the right mouse button,
- click on the **Power Symbol** button on the vertical toolbar,
- select **PWR Signal Type**, **PWR Shape** and enter **PWR Name** in the **PWR** window; if you select **Signal Type** first, the remaining parameters will be given their default values,
- click on the **OK** button.

Editing schematic

Deleting objects

Prior to making any changes on the schematic, enter the **Select and Drag** mode by clicking on the **Select and drag** button, pressing the **F2** key or selecting **Select and Drag** from the **Mode** menu. To delete an object, select that object with the mouse and press the **Del** key on the keyboard.

NOTE:

Deleting a symbol removes all nets connected to that symbol. To retain those nets disconnect the symbol prior to deleting (use **Disconnect** button on the horizontal toolbar).

Moving objects

Objects can be moved by dragging while in the **Select and Drag** mode. Nets connected to the moved symbol are dragged with that symbol.

NOTE:

After moving a symbol wires can overlap - press **Ctrl+W** to redraw wires.

Adding Parameters to Symbols, Nets and Pins

External software may require for its proper function some additional information on the objects placed on the schematic. **Net**, **Pin** and **Symbol** Parameters serve this function.

Symbol Parameters

Double clicking on the symbol displays **Symbol Properties** window.

- to add new parameter, enter its **Name** and **Description** in the appropriate fields and click on the **Add** button,
- to edit the parameter name or description, select it on the list, edit it in the **Name:** or **Description:** field, and press the **Change** button,
- to remove the parameter, select it on the list and click on the **Delete** button,
- to move the parameter to the new location, select it on the list and click on the **Move** button, move the mouse pointer to the new location and click the left mouse button,
- to control the parameter display, double click on the parameter on the list: double dot to the left of the parameter name means that both the name and description will be displayed, single dot means that only the description will be displayed, no dot means no display.

Net Parameters

Double clicking on the net and pressing the **Attributes** button in the **Net Name** displays **Net Attributes** window.

- to add new parameter, enter its **Name** and **Description** in the appropriate fields and click on the **Add** button,

- to edit the parameter name or description, select it on the list, edit it in the **Name:** or **Description:** field, and press the **Change** button,
- to remove the parameter, select it on the list and click on the **Delete** button,
- to move the parameter to the new location, select it on the list and click on the **Move** button, move the mouse pointer to the new location and click the left mouse button,
- to control the parameter display, double click on the parameter on the list: double dot to the left of the parameter name means that both the name and description will be displayed, single dot means that only the description will be displayed, no dot means no display.

Pin Parameters

Double click on the symbol to display **Symbol Properties** window and click on the **Pin Parameters** button. In the **Pin Parameters** window select the desired pin on the list.

- to add new parameter, enter its **Name** and **Description** in the appropriate fields and click on the **Add** button,
- to edit the parameter name or description, select it on the list, edit it in the **Name:** or **Description:** field, and press the **Change** button,
- to remove the parameter, select it on the list and click on the **Delete** button,
- to move the parameter to the new location, select it on the list and click on the **Move** button, move the mouse pointer to the new location and click the left mouse button,
- to control the parameter display, double click on the parameter on the list: double dot to the left of the parameter name means that both the name and description will be displayed, single dot means that only the description will be displayed, no dot means no display.

Query

The **SC Query/Find** window invoked by the **Query** button displays useful information on the selected object:

- object name,
- object parameters,
- other objects connected to the selected object.

Clicking on the object in the Query window highlights the pointed on the schematic; double clicking on the object moves the selection to that object.

Verifying bus connections

Selecting a bus while in **Query** mode displays a list of all pins and nets connected to the bus. It enables verification of the bus connections and is suggested after making complicated connections on the schematic.

Importing Viewlogic Schematics

Procedure for importing ViewLogic projects into ACTIVE-CAD

Important note

A Viewlogic (VL) project should be in the form of a directory containing all needed files. Particularly, it should include the following subdirectories:

SCH	;subdirectory with schematic description files
SYM	;subdirectory with symbol description files

and other files, like *.ABL*, *.PLD*, *.MEM*, *.CMD*, etc.

Importing schematics and symbols

To import a VL project into ACTIVE-CAD, follow this procedure:

1. Create a new Xilinx project; select the appropriate FPDA/CPLD family and part type.
2. Run Schematic Editor and select the **File/Import ViewLogic Schematic** option.
3. From the sub-directory *SCH*, select the top level schematic file, which usually has the same name as the project directory. As an option, you can select all schematic files in the subdirectory*SCH*. In such a case, ACTIVE-CAD will automatically find the top level schematic. After making the selection(s), click on the OK button.
4. When import is finished, the top level schematic(s) will be added to the project. All hierarchical schematic macros (if present) will be added to the project library.

During the import of VL projects, names of some nets, buses, instances and macros must be changed. All the changes for a given schematic file are listed in the corresponding VAM file located in the project directory. VAM files are needed for ACTIVE-CAD and user must not remove them.

Since ACTIVE-CAD has a limitation on the length of the signal and macro names, it creates an VLIMPORT.LOG log file that keeps track of the new names relate to their corresponding VL names. This file is located in the project directory and contains detailed import report. Since the file is in ASCII, it is easy to read by the user.

Additional information about importing non-schematic macros

If a VL project contains ABEL or MEM macros (files), it is necessary to update them in ACTIVE-CAD project library by updating their XNF netlists. All *.ABL* and *.MEM* source files are automatically copied from the VL project directory into the ACTIVE-CAD project directory

ABEL Macros

To update an ABEL macro follow these steps:
1. Start Schematic Editor with schematic sheet containing the ABEL macro, and double-click on the ABEL macro in the schematic to invoke the **Symbol Properties** window.
2. Add in the **Symbol Properties** dialog window the new parameter:
$FILE=macro_name.abl
To enter this parameter, enter in the parameter field $FILE, and type in the description field *macro_name.abl* (which stands for the actual file name with .abl extension). Make sure that the *macro_name.abl* file exist in the project directory. The name of the macro must be the same as the ABL file. To complete the parameter entry, activate the **Add** button.
3. Select **Hierarchy/Hierarchy Push** option and double-click on the macro symbol. The HDL Editor will display the *macro_name.abl* file.
4. Activating the **Project/Update Macro** option in HDL Editor will automatically update the XNF netlist and the macro in the library.
5. Exit the HDL Editor.

MEM files based macros

To update a macro created from a *MEM* file follow these steps:

- In the Project Manager, choose the **Applications/Memory Generator** option.

- Click on the **Select** button in the **Memory Generator** dialog window.
- Select the appropriate *.MEM* file (the file name must be the same as the macro name) and click the **OK** button.
- Clicking on the **Generate** button will automatically update the macro and its XNF netlist.

PLD files based macros

Do nothing with macros described by PLD files. All *.PLD* files are automatically copied from the VL project directory to the ACTIVE-CAD project directory. Note that unrouted PLD based macros cannot be simulated in ACTIVE-CAD Simulator. Only the macros based on post-routed and back-annotated XNF netlist of the entire project can be simulated.

Simulation

The ViewLogic's *.CMD* file can be used directly with the ACTIVE-CAD simulator. If you want to refer to the original *.CMD* files, check the VLIMPRT.LOG file for changes to the signal and instance names. If wish, you can make manually the necessary changes in the *.CMD* files. The most common changes to be made are:

- All backslashes ("\") in names of hierarchical signals must be replaced with slashes ("/").
- Names of component instances are case-sensitive; ACTIVE-CAD simulator requires them to be capitalized.

Example:
The following Viewlogic's vector declaration:
  vector sw sw1\sw[6:0]_p
can be changed manually to:
  vector sw SW1/sw[6:0]_p.   ; this format is acceptable to ACTIVE-CAD simulator

Conversion limitations

NOTE: Below is the important information about translating designs from Viewdraw.

- Regardless of option, ignore the built-in (see next section) symbols. *IN,OUT,BI* and symbols from the 'builtin' library: *VCC,GND* and *VDD* are not imported. *IN,OUT* and *BI* symbols are replaced with terminals *INPUT*, *OUTPUT* and *BIDIR* respectively. Nets labeled *VCC,GND* and *VDD* are treated as special nets of the POWER type.
- If line "Y 0" is found in symbol description , property *'$SymbolType=Annotate'* will be automatically generated
- Property *$ARRAY* is not supported
- if properties *FILE=file* and *DEF=XABEL* or *DEF=VHDL* are found in schematic or symbol description then if corresponding file *'..\file.abl'* or *'..\file.vhd'* exists, the property *$FILE=file.abl* or *$FILE=file.vhd* will be automatically generated (for *DEF=ABEL* , file *..\file.abl* will be additionally copied to the project directory)
- All files *..*.mem, ..*.pld, *.txt* will be copied to project directory
- if reference name is missing, it will be generated as follows:
  '$' + page number + *'I'* + instance identifier
  Example:
  for instance 'I 516 primary:TBLOCKB 1 1370 0 0 1 ' on schematic ADXCMP2.2 (2nd page), the resulting name will be :" $2I516"
- All schematic coordinates are transformed to nonnegative coordinates
- Multi sheet macros (can include up to 9 sheets) are placed by the single sheet
- All signal names of type *NAME[n]* are converted into *NAMEn*
- Bus labels and net labels are truncated to 24 characters

- All inverted labels are changed by adding '_' at the beginning
  Example:
  inverted label *P1* will become *_P1*
- If labels with missing nets occur in complex buses, these nets are added in right upper corner of the schematic as single wires
- Port names are truncated to 14 characters. If this truncation results in duplicating names, resulting names are indexed from 0
  Example:
  PORT0, PORT1,...
  All changes in port names are reported in file *<name>.vam* in project directory (<name> is symbol name)
- For macro design, if symbol ports are changed, the corresponding signal labels are also changed (changes made are based on *.vam* file contents)
- For macro design, if there are ports with missing signals, signals with corresponding names are automatically generated. Terminals are assigned to macro nets; their names correspond to symbol port names.
- Reference names are truncated to 15 characters.All changes in port names are reported in file *<name>.vam* in project directory (<name> is schematic name)
- Value of property *REFDES* is truncated to 2 characters
- Two dimensional buses of type AB[i:j]CD[k:l]EF are allowed for which k=0 or l=0.
  Bus of that type will be transformed into single dimensional bus as follows:
  ABCDEF[(abs(i-j)+1)* (max(k,l)+1)+ j*(max(k,l)+1)-1:j*(max(k,l)+1)]

Customizing importing procedure

The importing procedure is based on a set of options which can be adjusted to obtain optimal results. All of them are defined in section [SC_ALV_options] of "susie.ini" file, which can be found in WINDOWS directory. The options with their default values and descriptions are listed below.

auto_library_search= YES
determines if undefined libraries should be browsed

recursive= YES
determines if hierarchical schematic import is enabled

multiply_schematics= YES
determines if multi-sheet schematics will be packed into single sheet

ignore_builtin= NO
determines if schematics that include symbols from the 'builtin' library will be ignored

remove_busbox= YES
determines if boxes on bus ports will be removed

autopinname= YES
determines if symbol pin names should be automatically generated

autopin= YES
determines if pin graphics should be automatically generated

xblox_bus= YES
determines if xblox symbols will be treated as buses with range[0:0]

invisible_refname= NO

determines if instance reference names, which are hidden in VL, should remain hidden (names will be modified by adding '$' at the beginning)

ignore_internal_prop= YES
determines if internal properties (with '@' at the beginning) should be ignored

Note:
Internal properties which should be treated individually (accepted or ignored) are listed in [SC_ALV_options] section as additional options

Example:
@DIVIDE1_BY=YES
@PROP1=YES
@PROP2=NO

Hierarchy operations

Schematic hierarchy rules

Defining hierarchy pins

Standard schematic terminals placed on the macro schematic are the electrically connected with pins of the macro symbol visible on higher hierarchy levels.
Creating schematic macro in the bottom-up style (starting from the macro schematic) the user must draw a terminal for every future pin of the macro symbol. During saving the macro into the library, suitable macro symbol will be created automatically.
If the symbol is created first (top-down design) every pin must be defined in the Symbol Editor. When hierarchy push into that symbol is performed, corresponding terminals will be created on the empty macro schematic.

Using $FILE parameters

Usually macro schematics are stored in the project library. The user may choose the other way of storing macro schematics: assigning **$FILE** parameter to the macro symbol and entering schematic file name as the value of that parameter. For example,

$FILE=BUFF7.SCH

parameter assigned to a macro symbol means that the macro schematic is stored in the BUFF7.SCH file. In that case only the macro symbol is stored in the library.

Saving macro in the library

Saving a macro in the library may be forced by selecting **File | Save** command. However, macro is saved automatically when hierarchy pop is performed and any changes have been made to the macro schematic.

Creating new symbols

New symbol can be created automatically by the New Symbol Wizard or manually in the Symbol Editor.

New Symbol wizard

New Symbol wizard can be invoked from the **Hierarchy** menu in the Schematic Editor. A series of windows displayed by the Wizard prompts the user for information such as symbol name, description, ports, contents. After filling that information, Wizard creates and stores the symbol in the library

Symbol Editor

Symbol Editor can be invoked from the **Options** menu in the Schematic Editor. It allows creating the graphical shape of the symbol and entering its pins and parameters. Please refer to the documentation or on-line help system for information on creating symbols in the Symbol Editor

Converting empty symbol to a macro

When hierarchy push is performed on the empty symbol, Symbol Wizard is invoked automatically and prompts the user for entering information required for macro creation. When the macro contents type (schematic, HDL code) is selected, wizard starts the appropriate application to enable creating the macro. Manual assignment of the **$FILE** parameter to the macro symbol is another way of converting the symbol into a macro. Entering schematic file name as the $FILE parameter value turns the symbol into schematic macro, entering HDL source code file name - converts it into HDL macro.

Converting a schematic into a macro

To convert a schematic into a macro select **Create Macro Symbol From Current Sheet** option from the **Hierarchy** menu in the schematic editor. Current schematic sheet will be stored in the project library and macro symbol will be created automatically.

Hierarchical Push/Pop

Hierarchy Push and Pop operations are enabled when hierarchy cursor is active. To activate hierarchy cursor press **Hierarchy Push/Pop** button on the vertical toolbar of the schematic editor. Double clicking on the macro symbol with the hierarchy cursor performs **Push** operation (opens the macro contents); double clicking on the empty space of the macro schematic performs **Pop** operation (closes the macro and moves to the higher hierarchy level).

Using hierarchy browser

Hierarchical structure of the project can be reviewed in the hierarchy browser in the Project Manager. Please refer to the Project Manager section of this manual for details.

Deleting, renaming macros in the library

Deleting and renaming of the macros in the library can be performed in the **Library Manager** application (invoked from the Project Manager). To delete or rename a macro:

- double click on the macro library on the list in the main Library Manager window;
- select the macro on the list of library objects;
- select appropriate command from the object menu.

Using the same macros in multiple projects

You can use the same macro symbol in different projects.

You can choose if you want changes to be updated in both projects, or you can make a copy of the macro to use in each project. You can also create a separate user library that will store the macros for use in multiple designs.

Using the same copy of a macro in another project
To use the same copy of the macro in another project assign the library where macro is stored as a project library of that project.

Using separate copy of a macro in another project
To use separate copy of the macro in another project, copy the macro the project library of that project.

Using user macro library

Editing hierarchy macros

To edit the hierarchical macro you can use one of the following methods:

Push into the hierarchical symbol starting from the top level schematic.
Select File/Open option in the schematic menu and select the macro name from the Macros list.
In the Project Manager hierarchy window double click on the macro schematic name.

Once you open the macro you can make changes. To save these changes use the Save option. Please note that the schematic macro is saved both to the project directory as a schematic file and to the project library.

Each time you save the macro a new netlist is generated for that schematic and it is saved into the project library.

Save macro also verifies if the I/O terminals on the schematic match the symbol pins. If they do not match then a warning message is displayed whether you want to update the symbol. Adding or removal of the pins is done automatically based on the current schematic I/O terminals. The pins that did not change will not be moved.

After you save the schematic macro with different pins, the symbols are updated on all open schematic, so that you can see if the changes in pin location result in net connection changes. The pins that are no longer present will be disconnected from the pins and you will have to correct these connections manually.

Changing pins in hierarchical macros

Adding/removing pins on the schematic

Push into the schematic macro in the schematic editor.
Add remove any i/o terminals as desired.
Save the schematic macro.
The message about different symbol pins will be reported. Select Yes to change the symbol.
The new symbol will be updated on the schematic, so that removed pins will be deleted from a symbol and new pins will be added in the available outline space. If there is no room to new pins, the outline will be stretched up. Any disconnected wires resulting from removed pins will be terminated with a hanging wire bubble symbol.
If necessary, select the Symbol Editor to further rearrange the new pins. Every time you save the symbol in the Symbol Editor, the schematic is updated so you can see the change.

NOTE: You cannot add/remove pins using symbol editor because it cannot update the schematic i/o terminals. If you change pins in the symbol editor, it will force you to save the symbol under different name.

Using $FILE links on the schematic macros

If you have used the Symbol Wizard to create your macro symbol, it already has a $FILE link to a schematic file.

If the macro does not have $FILE attribute you need to perform the following steps:

Push into the macro in the schematic editor
Select Save As option from the file menu and type in the macro schematic name. This will save the copy of the macro into project directory as *.SCH file.
Close the schematic and double click on the macro symbol on the higher level schematic.

In the Edit Symbol Window add the parameter $FILE=name.SCH, where the name is the schematic file name.

This operation will allow you to save the schematic file into the project directory, so that the pin consistency between symbol and schematic is not enforced at all times.

Once you have the symbol with $FILE parameter you can continue with the further steps to change the symbol pins:

Select the symbol in the symbol editor and add/remove any pins of the macro symbols.
Save the macro. When the message about the different pins is reported select to save it anyhow and type in the same symbol name again. This will prompt you if you want to overwrite the old symbol. Select Yes to save it.
Close the symbol editor and push to the new symbol in the schematic editor. The new empty symbol should still have a link to the macro schematic, so the macro should open to allow you match the i/o terminal to the new symbol pins.
When you attempt to save this macro schematic, the consistency between the schematic and symbol will be verified. The schematic will then be save both in the schematic file and into the project library.

NOTE: Before you overwrite the symbol with non-matching pins, be sure that the schematic saved into the project directory is correct. If you do not have the schematic saved in the project directory and you overwrite the schematic macro with an empty symbol, you may delete your own macro schematic.

HDL Editor

The HDL Editor is an essential tool for projects utilizing HDL description. Particularly, it is designed to support two languages: VHDL and ABEL. The HDL Editor may be used both as a standalone application for writing HDL code and as one of the design entry tools for creating ACTIVE-CAD projects. It provides very useful instrument, Language Assistant, which, thanks to the use of prepared templates, enables the user to create HDL code quickly and conveniently without the necessity for wide knowledge of the language. Syntax correctness of the code created within the HDL Editor can be easily verified thanks to the useful Check Syntax option. The editor has interface to synthesis tools which generate XNF netlist from HDL source code. Synthesis of HDL code, performed by these tools, consists in conversion of the RTL structure described in terms of the HDL language into XNF netlist built of Xilinx primitives.

Usage of HDL Editor

Generally, HDL Editor is a tool used to create source files either with VHDL or ABEL code. Top level HDL projects are described entirely by HDL files. All the files are attached immediately to the root of the project hierarchy in the Project Manager. In schematic type projects, HDL files describe functionality of HDL macros. An HDL macro is a combination of symbol that can be placed on a sheet in the Schematic Capture, source code, which describes the functionality of the macro, and synthesized XNF netlist. As such, a macro can be placed on each level of the project hierarchy except the top level, which is reserved for schematic sheets.

The type of the project implies the way the HDL Editor is used. In case of top level HDL projects, the editor is invoked only from the Project Manager. Files created within it are attached to the project hierarchy usually by the **Add to Project** command from the **Project** menu (in the HDL Editor). Two other items in this menu: **Create Macro** and **Update Macro** are inactive and grayed. Important issues concerned with synthesizing top level VHDL and top level ABEL projects are discussed in the appropriate sections devoted to top level HDL projects, in the chapters *Using VHDL* and *Using ABEL*.

In case of schematic type projects, the HDL Editor may be invoked both from the Project Manager and from Schematic Capture. It is used to create new and edit existing HDL macros. The editor may also be called from the State Editor. In this case it works as a browser of the code generated from a given state machine diagram.

Creating HDL macro

The need to create an HDL macro arises when you want to describe a part of the project by means of HDL language. A HDL macro usually is to be fitted into schematic belonging to an upper level of the project hierarchy. The process of creating an HDL macro comprises writing the source code, synthesizing XNF netlist, and preparing the suitable symbol that will represent the macro on schematic sheet.

There can be two approaches to the process of creating an HDL macro. The first methodology, which can be put as bottom-up, assumes that the source code describing the functionality of the macro will be created first. In this method, you call the HDL Editor from the Project Manager. The HDL Wizard (**File** menu, **New** item) and the Language Assistant (**Tools** menu, **Language Assistant** item) will help you to write the code quickly and conveniently. To create the macro you may also use existing source files. If you run HDL Wizard (see appropriate sections in the chapters *Using VHDL* and *Using ABEL*), it will create the frame the source code will be based on. In case of a VHDL file, this frame comprises declarations of entity and architecture, in case of an ABEL file it consists of pin declarations. In the process, you can use **Check Syntax** option to ensure that the code is free of bugs. The last thing to do is to execute the **Create Macro** command from the **Project** menu. This will invoke the synthesis process, produce suitable symbol for the macro, and insert the symbol into the project working library. You can also do the synthesis on your own using the **Synthesize** command from the **Synthesis** menu. In such case, if you thereafter execute the **Create Macro** command, synthesis will not be repeated.

In the second, top-down method, you first create the symbol for the HDL macro. This method involves usage of the Schematic Capture. From the **Hierarchy** menu in SC you must select the **New Symbol Wizard** option. The Wizard will prompt you to specify the language (VHDL or ABEL) and pins of the new symbol. As a result, a new macro symbol will be created and inserted into the project working library. The Wizard generates source HDL file containing the appropriate frame, as described above. The next step is to write the rest of the source code, which describes functionality of the macro, and synthesize the XNF netlist. To accomplish that, you must follow the procedure given in the next section, which describes updating existing macros.

SUMMARY:
To create a new HDL macro, you may choose on of the two methods

The first, bottom-up method comprises the following steps:
1. Run the HDL Editor from the Project Manager.
2. Edit the contents of the source file. Use the HDL Wizard and the Language Assistant to speed up and simplify your work. At any moment, you can check the syntax correctness of the file using the **Synthesis | Check Syntax** option.
3. Select the **Project | Create Macro** option to create the XNF netlist and the macro symbol.
4. Quit the HDL Editor.

The second, top-down, method assumes you are running the Schematic Capture. Follow this procedure:
1. In Schematic Capture, select the **New Symbol Wizard** option from the **Hierarchy** menu.
2. Proceed with the Wizard - you will be prompted to specify the Language for the macro and its pins.
3. To complete the process of building of the macro follow the instructions for updating HDL macros, given in the next section.

Updating HDL macros

Updating of an existing HDL macro that is stored in the project library is possible only if the macro is placed on a schematic sheet. Source files describing HDL macros are stored immediately in the project directory and the specification what symbol is associated with a given source file cannot be accessed until the macro symbol is placed on a schematic sheet.

In order to update the macro you have to "get inside" it. This can be achieved either from within the Schematic Capture or the Project Manager. In the first method, you have to open the sheet containing the macro to be updated. Having selected the **Hierarchy Push** option from the **Hierarchy** menu in Schematic Editor you have to double-click on the macro symbol. This will invoke the HDL Editor and enable editing the source file (Note that you can also modify source files omitting this procedure, however, the changes will not take any effect, i.e. the symbol and the XNF netlist will remain unchanged, until you update macro). To update the corresponding XNF netlist and the symbol (in case the pin specification has changed) you have to execute the **Update Macro** command from the **Project** menu.

In the second method you need not run the Schematic Capture. Note that the Project Manager enables searching the project hierarchy in the hierarchy browser. All you have to do is to find the macro you wish to update in the hierarchy tree, and double-click on its name which causes the HDL Editor to start with the appropriate file. Further proceeding is the same as described above.

SUMMARY:

An existing HDL macro may be modified and updated only if there is at least one instance of the macro in any schematic sheet attached to the project.

If you are in the Schematic Capture, do the following to modify an HDL macro:
1. Open the sheet containing the instance of the macro.
2. Select the **Hierarchy Push** option from the **Hierarchy** menu. Letter 'H' will appear next to the mouse cursor.
3. Double-click on the macro. As a result, the HDL Editor will start.
4. In the HDL Editor, modify the source file describing the macro. The Language Assistant may be helpful.
5. Execute the **Update Macro** command from the **Project** menu.
6. Quit the HDL Editor.

If you want to modify the macro without using Schematic Capture do the following:
1. In the Project Manager, find the macro in the project hierarchy tree (using the hierarchy browser).
2. Double-click on the macro name in the tree to run the HDL Editor with the appropriate source file
3. In the HDL Editor, modify the source file describing the macro. The Language Assistant may be helpful.
4. Execute the **Update Macro** command from the **Project** menu.
5. Quit the HDL Editor.

NOTE: If you change the pin specification in the macro being updated, the macro symbol will change. This usually corrupts the schematic sheet the macro is placed on.

Editing a file

The HDL Editor's window splits into two parts. The upper part is the working area intended for editing files. In the lower part, messages sent by synthesis and syntax analysis tools are displayed. The spliter that divides the window may be moved up and down so that the user can change the size of the both areas.

What differentiate the HDL Editor from a normal text editor is its keyword highlighting feature. The editor analyzes the edited text and displays in various colors words belonging to particular syntax categories (e.g. keywords, identifiers, constants, comments). This significantly improves readability of the source code. The HDL language for the current document (either ABEL or VHDL) is set in the configuration dialog window which is invoked from the **Synthesis** menu (**Configuration** item).

The HDL Editor has all typical editing functions of a non-formatting text editor which are available both from the Edit menu and immediately from the keyboard. It uses standard key combinations which are common for most Windows applications, e.g. Notepad. Text edited within HDL Editor can be exported to and imported from the Windows Clipboard. Bookmarks enables user to quickly move throughout long files. User may customize the HDL Editor using the options available in the Preferences dialog invoked by the **Tools | Preferences** command. Options available in the **View** menu control the visibility of main components of the HDL Editor window.

Syntax check

The Check Syntax command performs the syntax analysis of the text being edited. It is available in the **Synthesis** menu (**Check Syntax** item). The operation involves either the internal built-in checker (only for VHDL) or external applications (XABEL for ABEL and XVHDL for VHDL). The built-in checker supports most of the VHDL 1074-1993 standard (full support planned). The tool for syntax check is chosen in the Configuration dialog (**Synthesis | Configuration**). The messages generated in the process are displayed in the lower part of the HDL Editor window. In addition, the general message, which informs about the success or failure of the analysis, appears in the separate message box.

Finding errors

If syntax check fails, HDL Editor marks places of occurrence of syntax errors through underlining. In addition, a red mark (arrow), showing the place of occurrence of the current error, is displayed on the margin. The text of the message corresponding to the current error is reversed. Using **Next Error** and **Previous Error** options from the **Search** menu you can easily find all errors that have been encountered during the last check.

Selecting language, synthesis tool and synthesis options

Each source file describing a macro, edited within the HDL Editor have assigned set of parameters which specify the language, the synthesis tool (application used to generate XNF netlist from the source code) and synthesis options.

In case of top level HDL projects, these parameters are common for all HDL project files. This issue is discussed in the sections devoted to top level HDL projects, in the chapters *Using ABEL* and *Using VHDL.*

The language and the synthesis tool is set in the dialog Configuration invoked from the **Synthesis** menu. The dialog window comprises three fields:
- **Language**, which indicates whether the file contains VHDL code, ABEL code, or other text,
- **Check Syntax**, which indicates whether the syntax check is performed by the external synthesis tool specified in the Tool field, or by the internal built-in checker (available only for VHDL),
- **Tool**, which specifies the synthesis tool.

Synthesis options define the method of synthesis, possible optimizations and some other detailed parameters. The parameters are set in the dialog invoked by the **Options** command from the **Synthesis** menu. Parameters displayed in the dialog may vary according to the selected language, the synthesis tool, and Xilinx device family (which is specified for the entire project in the Project Manager). Here is short description of these parameters.

<u>VHDL General Options</u>

Compile:
Chip - enables insertion of IBUF and OBUF pads; used for HDL type projects.
Macro - disables insertion of IBUF and OBUF pads; used for schematic type projects.
Optimize - options for the logic optimizer Improvex:
Area - optimization for utilization (number of used CLBs), trading off performance (speed).
Speed - optimization for performance (number of CLBs in the critical path), trading off utilization (number of CLBs).
Standard - intermediate solution between Area and Speed options.
Effort level - specifies optimization effort level:
High
Medium (conserve memory)
Low (simulation only)
X-BLOX - enables inference of xblox macrocells.
Improvex - enables the logic optimizer Improvex which optimizes the synthesized XNF netlist

<u>VHDL Advanced Options</u>

Entity / Architecture - specifies the root (top) of the design to be elaborated.

VHDL Library Aliases

Library Alias - specifies an alias for a VHDL library, overriding the default mapping of VHDL library name to file name.

In XVHDL compiler a library is simply an external VHDL file located in the subdirectory VHDL_LIB that is located in the same directory that XVHDL.EXE. Besides standard VHDL libraries (STD, IEEE) and libraries provided with the system (e.g. VLBIT, SYNOPSYS), the subdirectory can also contain user-defined library files. By default, library files are associated with references in the VHDL code (in the clauses LIBRARY and USE) by name (e.g. file 'LIBX.VHD' describes library accessed by identifier LIBX'). Library Alias option allows user to override the default association by explicitly specifying which source files are complied into given library. Thanks to this option, user need not modify the contents of the original library files.

Example:
Assuming that the library IEEE has to be supplemented with the package STD_LOGIC_ARITH whose source code is included in the file SYNOPSYS.VHD, user can use the Library Alias option to associate the library IEEE with the files:

```
        IEEE.VHD
        SYNOPSYS.VHD
```

XVHDL analyzes these files in the order they appear in the dialog window (in the field **VHDL Files**). It may be important when one of the files contain references to another (e.g. in this case package SYNOPSYS uses package IEEE, so the file IEEE.VHD must be analyzed first).

NOTE: Library Aliases are not supported by the internal VHDL checker.

ABEL General Options for Xilinx FPGA devices

Compile:
Chip - enables insertion of IBUF and OBUF pads; used for HDL type projects.
Macro - disables insertion of IBUF and OBUF pads; used for schematic type projects.
Optimize - options for the logic optimizer Improvex:
Area - optimization for utilization (number of used CLBs), trading off performance (speed).
Speed - optimization for performance (number of CLBs in the critical path), trading off utilization (number of CLBs).
Standard - intermediate solution between Area and Speed options.
Improvex - enables the logic optimizer Improvex which optimizes the synthesized netlist

ABEL General Options for Xilinx EPLD devices

Compile:
Chip - enables insertion of IBUF and OBUF pads; used for HDL type projects.
Macro - disables insertion of IBUF and OBUF pads; used for schematic type projects.
Optimize - reduction options:
Auto Polarity - reduces the logic so that each signal will have the minimum number of terms possible. The optimization will produce both ON-set and OFF-set equations so that the minimum number of product terms will be used in programmable polarity parts.
Fixed Polarity - reduces so that each signal will have the minimum number of product terms, maintaining the polarity of the signals as specified in the source file. This should only be used when you want to force output signals to a specified polarity (typically through the use of the 'pos' and 'neg' signal attributes).
No Reduction - merges all the compiled equations into a single ABEL-PLA file. No logic

reduction is performed. This option must be used if you want redundant logic to be preserved for any of the design outputs.

<u>ABEL Advanced Options for Xilinx FPGA devices</u>

Encoding - options for state-encoding method:
Standard - automatic selection of the best encoding scheme.
Binary - for small state machines, or state machines with few (less than four) input variables, binary encoded state machines may be more efficient.
One Hot - the One-Hot method uses one flip-flop per state machine state.
Unspecified state - these options specify how to synthesize incompletely specified state machines. Incompletely specified state machines are those which do not have a state assignment for every possible input combination.
Initial - incompletely specified state machine defaults to the initial state whenever the next state for the current input combination is not specified.
Current - incompletely specified state machine defaults to the current state.
Don't Care - no default for incompletely specified state machines.

Synthesizing macro

When editing of the source code is finished, you can generate XNF netlist built of Xilinx primitives. It is achieved through the use of synthesis tools. According to the size of the source file, synthesis may take up to a few minutes. During synthesis, the Project Manager window appears on the top so that the user can watch the messages generated by the synthesis tool, which are displayed in the message window (beneath the spliter). In addition, all these messages are saved in the log file, which can be viewed after the synthesis.

Viewing synthesis LOG

Synthesis log file is generated during each synthesis process and is located in the current project directory. It can be viewed by the **View Report** command from the **Synthesis** menu. This command invokes an auxiliary application, Report Browser, with the current log file loaded. If there is no log file in the current project directory, suitable message box will appear.

Language Assistant

Language Assistant is a simple yet very efficient tool which dramatically simplifies the process of writing HDL description. This is a sort of data base containing ready-made language templates for typical syntax constructs. These templates comprise the following sets:
- VHDL syntax templates,
- VHDL synthesis templates,
- ABEL syntax templates,
- ABEL synthesis templates.
In addition, user can create and save for later use his own templates.

Language Assistant is invoked from the **Tools** menu. All templates are collected together in the form of a hierarchical tree. Expandable branches may contain both template items and further branches. In order to insert the desired template into the edited HDL file, you have to either double-click on it or select it and click on the **Use** button. Language Assistant allows to view the contents of the selected template. Use the button **Show/Hide Preview** to turn on and off this feature.

Editing existing templates

You can edit a template (and/or its name) if its parent branch has the attribute Read Only inactive. (To toggle the attribute, select the branch and click the right mouse button. From the menu that will appear select the **Read Only** item). Click on the **Show Preview** button to open the preview area and place the cursor there. Then edit the template as desired.

If you want to change the template name, select it and click on the **Edit** button. The name will be inserted into the editing box, where you can modify it.

Deleting templates

You can delete a template if its parent branch has the attribute Read Only inactive. (To toggle the attribute, select the branch and click the right mouse button. From the menu that will appear select the **Read Only** item). Select the template and click on the **Delete** button.

Creating user templates

You can add your own templates to expand functionality of the Language Assistant. To attach a new item to the tree follow these steps:

1. Select the branch to which you want to add the new item.
2. Click the **New** button.
3. Enter the item name in the editing box that will appear.
4. If the new item is to be a template, click on the preview area (you may have to click on the **Show Preview** button) and edit its contents. If the item is to be an expandable branch skip this step.

NOTE1: As long as a given tree item has no text assigned, you can add new items to it, thus creating the new branch.

NOTE2: Operations executed through clicking on the buttons **Use**, **Delete**, **New**, **Edit** can be achieved through selecting appropriate items from the right mouse button pop-up menu.

NOTE3: Changes made to existing templates, and new templates are saved when you quit HDL Editor. The editor asks for confirmation so you can abandon all changes if you wish.

Using X-BLOX

What are X-BLOX symbols?

The X-BLOX (Blocks of Logic Optimized for Xilinx) is a library of modules that you can use to describe a system by means of high-level functions instead of gate-level primitives. Because X-BLOX modules are customizable, each module can describe thousands of unique functions. You can customize these modules using attributes and by connecting appropriate control pins of the modules.
X-BLOX supports only Xilinx FPGA families. Detailed information about X-BLOX you can find in the Xilinx '**X-BLOX User Guide**' manual.
Main X-BLOX features there are:

- block diagram design entry
- generic data path sizes and encoding
- optimized implementations

You can use X-BLOX in two ways:

- place X-BLOX macros on the schematic
- synthesize VHDL projects/macros with option X-BLOX enabled which causes synthesis with X-BLOX macros

If you want to use X-BLOX you must have the library XBLOXU (X-BLOX Unified library) attached to your project. When you create a new project with Project Manager this library is attached by default if X-BLOX supports selected Xilinx family.

Placing X-BLOX symbols

Attaching a X-BLOX library
When you create a project X-BLOX library named XBLOXU is automatically attached by Project Manager (if selected Xilinx family is supported by X-BLOX).

Bringing up a symbol
You place the X-BLOX symbol like any other symbol from the library. Follow the procedure:

- open the SC Symbol selection window pressing the button 'Symbol toolbox' (or selecting the option 'Symbols' from the menu 'Edit'),
- select the symbol from the list, place in the desired position and click the mouse button.

Specifying the module attributes
Invoke the 'Symbol properties' window (double click on the symbol). The section 'Parameters' describe all symbol parameters. For all X-BLOX symbols the basic parameters are already defined without their values.
If you want to add or change the value of already defined parameter follow the procedure:

- select the parameter you wan to change with the mouse and click on it; selected parameter appears in the field 'Name' and it's value (if any) in the field 'Description',
- enter desired value in the field 'Description' and press button 'Change'; new value of the parameter appears in the parameter's window.

If you want to add a new parameter (which is not listed in the window) follow the procedure:

- enter the name of the parameter in the field 'Name'; you can do it in two ways: by selecting the parameter from the pull-down list (suggested) or by typing the name in the field 'Name',
- enter the value of the parameter in the filed 'Description' and press the button 'Add'.

For each parameter you can define its visibility on the sheet which is marked by number of dots on the left side of the parameter's name:

- none - nothing is displayed,
- one - only parameter's value is displayed,
- two - both the parameter's name and value are displayed.

NOTES:

1) You can change visibility of the parameter with double click on its line in the parameters window.
2) You can change visibility of all parameters with buttons ‘Display All’ and ‘Clear Display’.
3) You can move displayed parameters to any position with the button ‘Move’.

When you set all parameters and their visibility to desired values press the button ‘OK’.

Connecting to X-BLOX symbols

Single pins

Single pins of the X-BLOX symbols are the same as single pins of all other symbols. This pins can be connected to any other single pins and standard buses (by bus taps). You connect them with nets (button ‘Draw wires’ or option ‘Draw wires’ from ‘Mode’ menu).

Bus pins

X-BLOX bus pins are not the same as bus pins of other symbols and **cannot** be connected to standard buses (see section ‘X-BLOX buses’ for more information). You draw X-BLOX buses on the schematic in the same way like standard buses (button ‘Draw buses’ or option ‘Draw buses’ from ‘Mode’ menu).

IMPORTANT: The X-BLOX buses must have names without range. When you are naming a X-BLOX bus you have to set both indexes to zero. In this case the name of the bus is displayed without the range.

Note that bus pins of X-BLOX symbols have different width and color then bus pins of standard symbols. It should help you to avoid connecting X-BLOX bus pins to standard bus pins.

X-BLOX buses

All X-BLOX buses are generic in size. This differentiate them from standard buses. So, X-BLOX buses **cannot** be connected with standard buses which have stated size. What’s more, the X-BLOX bus is not just a collection of wires - a bus defines the kind of data that travels through the bus.
When you run X-BLOX, it synthesizes the buses expanding them to the proper sizes as it generates the simulation models and performs chip-level implementation. Generic-sized busses allow the schematic to be resized quickly because the bus size needs to be specified only once on each data path.

Specifying X-BLOX buses
To label an X-BLOX bus use a name without specifying the bus range (set both indexes to zero). Contrast the following specifications for X-BLOX and non X-BLOX buses:

X-BLOX bus:	*ADDR*
standard bus:	*ADDR[15:0]*

You have to assign two attributes of the X-BLOX bus to establish a data type:
- BOUNDS which defines the bus width (two integers separated with colon),
- ENCODING which defines the data encoding scheme (BIT, UBIN, ONE_HOT or TWO_COMP).

You specify the bounds and encoding of X-BLOX bus anywhere on the data path. See Xilinx ‘X-BLOX User Guide’ for more details.

Connecting to X-BLOX buses
You can connect directly only two X-BLOX buses with the same size (BOUNDS) and data type (ENCODING). If you need to connect X-BLOX bus to anything else you have to use special interface elements (they do not add any logic to your design). The detailed information about X-BLOX bus manipulations you can find in Xilinx ‘X-BLOX User Guide’. The following section tells you what element you have to use to connect X-BLOX bus with:

- other X-BLOX bus with the same size and different indexes (BOUNDS) and/or data type (ENCODING): **CAST,**
- other X-BLOX bus with different size: **SLICE,**
- net: **ELEMENT,**
- standard (non X-BLOX) bus: **BUS_IFxx** (where ‘xx’ is the standard bus size),
- power / ground (force constant value on the bus): **FORCE**.

Using X-BLOX in hierarchy macros

You create X-BLOX macro in the same way like other schematic macros. The only difference is that X-BLOX bus pins have names without ranges (range [0:0] is displayed on the symbol).
You have to specify bus parameters at last once per data path whether or not data the data path crosses hierarchical boundaries.
If you do not specify the data type in the underlying schematic, this schematic become generic and can be resized by the bus type definitions specified in the main schematic.

Functional simulation

X-BLOX symbols are generic in size and their function is selected with parameters and/or control pins. X-BLOX buses are generic in size and the size can be defined once per data path. This reasons causes that there is impossible to simulate X-BLOX elements before they are implemented. Implementation of the X-BLOX symbols performs Xilinx program Xblox.
This causes that functional simulation of the project containing at least one X-BLOX symbol requires running Xblox program before. So, if you want to perform functional simulation you must have Xblox package installed. The Xblox program is automatically run by the Project Manager when required (i.e.when you change macro containing X-BLOX).

Using VHDL

Requirements

If you want to use the VHDL feature in your designs you have to install Xilinx VHDL software package. The package comprises compiler-synthesizer XVHDL and logic optimizer ImproveX. It also provides a set of useful libraries, among others:
STD, with the basic package *standard*
IEEE, with package *std_logic_1164* for multi value logic system,
LPM. with package *macros* supporting LPM modules,
XBLOX with package *macros* supporting XBLOX modules,
METAMOR, with package *attributes* which defines synthesis-specific attributes,
SYNOPSYS, with packages supporting arithmetic operators and functions for the multi value logic system.

Using HDL Design Wizard for VHDL

The HDL Design Wizard is available within the HDL Editor. It is run whenever you select the **New** option from the **File** menu. It can be also invoked immediately from the logo window that appears when you start the HDL Editor. The Wizard is intended to help you to start creating a source file. This is accomplished through the use of a number of dialogs which prompts you to define the information needed to create the basic elements of the source file. The first choice you have to make applies to the language of the document: either ABEL or VHDL.

In case of VHDL, the basic elements of the code, as mentioned above, comprise the following:
1. **Library clause** with reference to the IEEE library, and **use clause** with reference to the STD_LOGIC_1164 package.
    These two elements are inserted into the file only if you declare ports (see next entries) of type *std_logic* or *std_logic_vector*.
2. **Entity declaration**.
    The Wizard asks you to specify the entity name.
3. **Ports declarations**.
    The Wizard asks you to specify the name, type and direction of each port. Ports may be of one of the following types:
    - scalar types: *bit*, *boolean*, *character*, *integer*, *std_logic*,
    - composite types: *bit_vector*, *std_logic_vector*.
4. **Architecture declaration**.

The Wizard creates the file with the name same as the entity name, containing the elements listed above. The user should then supply the file with the appropriate architecture description.

NOTE: Similar tool, New Symbol Wizard, available in the Schematic Capture, allows you to create simultaneously macro symbol and the corresponding source file with all the elements mentioned above.

Quick modifying of port clause

The HDL Editor provides useful feature allowing you to quickly modify the port clause, which describes the interface (i.e. pins) of a macro (or the entire project). The port clause consists of a number of port declarations. When you select the **Symbol** option from the **Edit** menu, you will invoke the window showing the port clause in the form of pins attached to a 'black box'. The window also includes the list of all ports declared in the file. The list can be easily modified. Using the mouse you can add, delete and modify ports. Use the left mouse button to modify the parameters of existing ports (direction, type, bus range). With the right mouse button you activate pop-up menu which allows you do add, delete and rename ports. Clicking OK closes the window and causes the modifications to take effect (the port clause will change).

Managing VHDL libraries

In XVHDL compiler a library is simply an external VHDL source file or files. These files are stored in the subdirectory VHDL_LIB located in the VHDL data files directory (by default \\ACTIVE\VHDL). Besides standard VHDL libraries (STD, IEEE) and libraries provided with the system (e.g. VLBIT, SYNOPSYS), the subdirectory can also include user-defined library files. Library WORK is a dynamic object that exists only when compiler is running.

By default, library files are associated with references in the VHDL code (in the clauses LIBRARY and USE) by name (e.g. file 'LIBX.VHD' describes library accessed by identifier LIBX'). This might be inconvenient in the situation when you want to add, for example, some new packages to a standard library (standard, i.e., provided with the software) without modifying the original library files. In such cases, you may found useful the Library Alias option. This option allows user to override the default association by explicitly specifying which source library files are complied into given library. Thanks to this option, user need not modify the contents of the original library files. This options in available in the HDL Editor's Configuration dialog, invoked form the **Synthesis** menu.

Example:
Assuming that the library IEEE has to be supplemented with the package STD_LOGIC_ARITH whose source code is included in the file SYNOPSYS.VHD, user can use the Library Alias option to associate the library IEEE with the files:

    IEEE.VHD
    SYNOPSYS.VHD

XVHDL analyzes these files in the order they appear in the dialog window (in the field **VHDL Files**). It may be important when one of the files contain references to another (e.g. in this case package SYNOPSYS uses package IEEE, so the file IEEE.VHD must be analyzed first).

Managing VHDL libraries for built-in syntax checker

Libraries for the built-in VHDL syntax checker must be compiled before use. ACTIVE-CAD does not provide interface for compilation of these libraries. The compilation can be performed from the command line. To do so, switch to DOS prompt, and enter the following command:

comp95.exe -d <VHDL directory with full path> -l <library name> <file to be compiled>

VHDL directory is that containing subdirectories VLIB and DAT (by default \\ACTIVE\VHDL). The file that is to be compiled should be specified with extension (**.vhd**).

As a result, the appropriate compiled file with extension **.vlb** will be put into the VLIB directory.

NOTE: When you want to add a VHDL library to the ACTIVE-CAD, you should make it visible both for internal built-in VHDL checker and for XVHDL compiler. To do so, you must follow the procedures described in the above and the previous section.

Top level VHDL projects

In a top level VHDL project, source files describe not some isolated fragments of the project (macros) but the entire project (Schematic Capture is not used at all). All the files must be attached to the root of the hierarchy. Note that the Project Manager does not allow to attach to the hierarchy root files of assorted types (e.g. a schematic sheet and an HDL file). Therefore, when you have opened a new project you should not run the Schematic Capture for this would cause a blank sheet to be attached to the hierarchy root. For the same reason, if you have a SCHEMATIC type project and you want to have the type changed to VHDL or ABEL, first you must detach all sheets from the hierarchy root, and than you can attach HDL files.

Source files can be attached to the hierarchy root in two ways. When you edit a file in the HDL Editor, you can attach the file using the **Add to Project** option from the **Project** menu. You can do the same from within the Project Manager, using its **Add** option from the **Document** menu. Top level VHDL projects may include more than one source file (e.g. one file describes top level entity, other files – entities of lower level). In such cases, the order the files are attached to the hierarchy root is the order these files are analyzed during syntax analysis and synthesis. As the XVHDL compiler does not keep the static library WORK, this order is very important.

VHDL source files should be synthesized with the synthesis parameter **Chip** set on (see the section *Selecting synthesis options*). The synthesis options are set for the entire project, what means they are common for all the VHDL files attached to the project. Therefore, when you change some settings in the HDL Editor, these changes will apply not only to the currently loaded file but to all the VHDL project files. If the project contains multiple VHDL source files you may run the synthesis from within the HDL Editor with *any* of the project files loaded. However, it is important to indicate in the synthesis options the top level entity and architecture.

NOTE: If you attempt to invoke functional simulation or Place & Route processing without having done synthesis, ACTIVE-CAD will run synthesis on its own (without involving the HDL Editor). After that, you can view the synthesis log using the appropriate option in the HDL Editor.

VHDL synthesis options

Synthesis options define the method of synthesis, possible optimizations and some other detailed parameters. You will find more detailed description of these options and the usage and in the section *Synthesis Options* in the chapter *HDL Editor*. Here is short description of these parameters.

General Options

Compile:
- **Chip** - enables insertion of IBUF and OBUF pads; used for VHDL type projects.
- **Macro** - disables insertion of IBUF and OBUF pads; used for schematic type projects.

Optimize - options for the logic optimizer Improvex:
- **Area** - optimization for utilization (number of used CLBs), trading off performance (speed).
- **Speed** - optimization for performance (number of CLBs in the critical path), trading off utilization (number of CLBs).
- **Standard** - intermediate solution between Area and Speed options.

Effort level - specifies optimization effort level:
- **High**
- **Medium** (conserve memory)
- **Low** (simulation only)

X-BLOX - enables inference of xblox macrocells.
Improvex - enables the logic optimizer Improvex which optimizes the synthesized XNF netlist

Advanced Options

Entity / Architecture - specifies the root (top) of the design to be elaborated.

Library Aliases

Library Alias - specifies an alias for a VHDL library, overriding the default mapping of VHDL library name to file name.

NOTE: Library Aliases are not supported by the internal VHDL checker.

Available VHDL Templates

The HDL Editor provides Language Assistant, simple yet very efficient tool, which simplifies the process of writing HDL files. This is a sort of data base containing ready-made language templates for typical syntax constructs. The user can write and add his own templates. You will find detailed description on the usage of Language Assistant in the section *Language Assistant* in the chapter *HDL Editor*.

VHDL templates provided with the software comprise two kinds of constructs:
- syntax templates, which comprise typical language objects,
- synthesis templates, which comprise synthesizable constructs describing basic elements (e.g. gates, flip-flops, multiplexers, counters, etc.)

Using On-Line VHDL Guide

HDL Editor gives you access to the on-line help provided with the XVHDL compiler. To get the help choose the **Help Topics** option from the **Help** menu in the HDL Editor and click on the **Metamor Reference** button.

Using ABEL

Requirements

If you want to use ABEL-HDL feature in your designs you have to install Xilinx ABEL software package. Active-CAD uses the following programs for ABEL synthesis:

FPGA synthesis: abl2xnf.exe
ELPD synthesis: ahdl2x.exe, blifoptx.exe, pla2eqnx.exe, readpld.exe

NOTE: programs mentioned above runs other XABEL programs, so you must have all Xilinx ABEL package installed.

Using Xilinx-ABEL in FPGA and EPLD designs

Identifiers case sensitivity

Identifiers in ABEL-HDL are case sensitive but in the XNF netlist are not. So, identifiers consist of the same letters which differs only in case should not be used. Otherwise an error will appear during synthesis.

Module names

Although the Xilinx ABEL software allows you give the modules in the ABEL-HDL file any names that you choose, Xilinx recommends that each ABEL-HDL file contain only one module, which should have the same name as that of the ABEL-HDL file.

Signal declarations for ABEL macros

Pin and node declarations declare all signals used in the design. Declaring signals you should complied with the following rules:

- **PIN** declarations define external connections to the ABEL-HDL macro; this signals will become I/O pins of the created symbol.
- **NODE** declarations define internal signals which are not connected with any external signal; signals declared as nodes are not guaranteed to be retained in the output XNF netlist unless you explicitly save them by using Xilinx property save (see ‘Including Xilinx FPGA properties’ section); this signals will not appear on the created symbol.
- Declare **BUS** pins with the ABEL-HDL range operator ‘..’; example:
  Declaration

  ```
  D7 .. D0  pin ;
  ```

  will create symbol with one bus pin named D[7:0],
  declaration

  ```
  D7,D6,D5,D4,D3,D2,D1,D0  pin ;
  ```

  will create symbol with eight separate pins named D7,D6,D5 etc.

Including Xilinx FPGA Properties

The **property** declaration allows you to specify additional design information associated with the Xilinx processing modules. The syntax of the **property** statement for Xilinx FPGA devices is the following:

XILINX **PROPERTY** ‘*string*’ ;

The Xilinx **property** statements:

- XILINX PROPERTY ‘initialstate *state_name*’ ;
  XILINX PROPERTY ‘initialstate *state_register_name state_name*’ ;
  This property defines the initial power-up state of the state register and is used for symbolic state machines only. It instructs the compiler to arrange logic so that the state machine always goes to the specified state during power-up or global reset. If this statement or the **async_reset** statement is not used, the initial state is chosen by the compiler. The second syntax is required if there are multiple state machines.
- XILINX PROPERTY ‘map *output_pin input1 input2 input3 ...*’ ;
  The property **map** ensures that the sumnetwork between the output pin and the specified inputs is mapped into one CLB. If Improvex cannot fit the entire map into one CLB it issues an error and stops processing.
- XILINX PROPERTY ‘save *signal_name*’ ;
  Normally only pin names are preserved in the final XNF netlist that XABEL produces, intermediate node and signals may disappear. The property **save** ensures that the specified signal name is saved in the final XNF netlist.
- XILINX PROPERTY ‘dlc2s *max_value*’ ;
  XILINX PROPERTY ‘dlc2p *max_value*’ ;
  XILINX PROPERTY ‘dlp2s *max_value*’ ;
  XILINX PROPERTY ‘dlp2p *max_value*’ ;
  This properties set maximum number of logic levels on:
  dlc2s - all paths from flip-flop to flip-flop,
  dlc2p - all paths from flip-flop to output pin,
  dlp2s - all paths from input pin to flip-flop,
  dlp2p - pure combinatorial logic paths in the module.

Including Xilinx EPLD Properties

Some device-specific features of the Xilinx EPLD architectures, like the XOR operator, are supported directly in the ABEL-HDL syntax.
Some other features of the XEPLD architecture, such as input pad registers, are supported using PLUSASM **property** statement. Other features, like the built-in arithmetic circuitry, can be specified through include files written in PLUSASM language.
Any PLUSASM declaration statement or equation can be specified in an ABEL-HDL file with a **property** statement. The syntax for this statement is the following:

PLUSASM **PROPERTY** ‘statement’;

or

XEPLD **PROPERTY** ‘statement’;

These two statements are equivalent. The statement can be any PLUSASM declaration.
Files written in the PLUSASM language can be included with the statement INCLUDE_EQN. The syntax for this statement is the following:

INCLUDE_EQN ‘filename.pld’

The ABEL-HDL **property** statement for this construction is the following:

PLUSASM **PROPERTY** ‘INCLUDE_EQN “filename.pld” ‘;

Note that single quotation marks required within the PLUSASM declaration string are replaced by double quotation marks in ABEL-HDL **property** statement.

Using HDL Design Wizard for ABEL

The HDL Design Wizard is available within the HDL Editor. It is run whenever you select the option **New** from the menu **File**. It can be also invoked immediately from the logo window that appears when you start the

HDL Editor. The Wizard is intended to help you to start creating a source file. This is accomplished through the use of a number of dialogs which prompts you to define the information needed to create the basic elements of the source file. The first choice you have to make applies to the language of the document (either ABEL or VHDL).

In case of ABEL, the basic elements of the code, as mentioned above, comprise the following:

1. **File name**

   The Wizard asks you to specify the name of the file where ABEL code will be saved.

2. **Pin declarations**.

   The Wizard asks you to specify the name, direction and type (for output pins only) of each pin. Output pins can be combinatorial or registered.

The Wizard creates the file with the specified name and basic ABEL-HDL source file sections:

- Header with *module* declaration and *title*; both - module name and title - are the same as the file name,
- Declarations section, where all pins are declared; types of outputs are specified with *istype* statement,
- Keyword *equations* which starts design description section,
- *End* statement.

The user should then supply the file with the appropriate design description and (optionally) additional declarations (i.e. set, constant, macro declarations).

Available ABEL Templates

The HDL Editor provides Language Assistant, simple yet very efficient tool, which simplifies the process of writing HDL files. This is a sort of data base containing ready-made language templates for typical syntax constructs. The user can write and add his own templates. You will find detailed description on the usage of Language Assistant in the section *Language Assistant* in the chapter *HDL Editor*.

ABEL templates provided with the software comprise two kinds of constructs:
- syntax templates, which comprise typical language objects,
- synthesis templates, which comprise synthesis-specific constructs.

ABEL designs at the top level

Creating top level ABEL design

In a top level ABEL project, ABEL files describe not some isolated fragments of the project (macros) but the whole project (Schematic Capture is not used at all). ABEL-HDL file describing the project must be attached to the root of the hierarchy. In order to edit and attach a new source file you have to run the HDL Editor, edit the file, and attach it to the project using the option **Attach to Project** from the menu **Project**.

Do the following to create a new ABEL file for a top level ABEL project:

- Run the HDL Editor from the Project Manager.
- Edit the contents of the source file. Use the HDL Wizard and the Language Assistant to speed up and simplify your work. At any moment, you can check the syntax correctness of the file using the option **Synthesis/Check Syntax**.
- Select the option **Project/Add to Project** to add the file to the root of the project hierarchy.

The procedure for synthesizing top level HDL projects is similar to that for macros. A top level ABEL project contains single source file. Synthesis of top level ABEL projects is invoked from within HDL editor. To run the editor with the file you must double-click on the name of the file in the hierarchy tree in the Project Manager (in the hierarchy browser). This is an obligatory condition that the synthesis parameter **Chip** must be set on (see the section *ABEL synthesis options*). Otherwise, Place & Rout process will fail.

Do the following to synthesize a top level ABEL project:

- Run the HDL Editor from the Project Manager double-clicking of the ABEL source file name you want to synthesize in the hierarchy browser.
- In the synthesis options dialog (menu **Synthesis**, item **Options**) set the option **Chip** on.
- Execute the command **Synthesize** from the menu **Synthesis**.

Using multiple files

Top level project in the ABEL-HDL can contain only one design file. Project manager does not allow to add more than one ABEL file to the design.
It is possible to create a multi-file design. Such design still contains a single top level file, other files can be included in the designing using one of two methods:

1. With ABEL-HDL **@INCLUDE** directive which includes any other file to ABEL source file. Syntax of this directive is the following:

```
          @INCLUDE 'filename'
e.g.:
          @INCLUDE  'C:\\ACTIVE\\PROJECTS\\ABELSRC\\MACROS.ABL'
```

2. Only for Xilinx EPLD designs
   With ABEL-HDL **property** statement and PLUSASM **INCLUDE_EQN** statement which allows to include PLUSASM file to the design (for more information see 'Including Xilinx EPLD Properties' section).

NOTES:

- Files linked with the methods described above are compiled into a single XNF file. The device fitter does not recognize that they were once separate files.
- When you want to synthesize multi-file project you have to load to the HDL Editor the top level file and then run synthesis.
- Included files are not displayed in the hierarchy browser in the Project manager.

Declaring signals

In the ABEL-HDL source file signals are declared as PINS or NODES. In a one-file design or the top-level file of a multi-file design, signals that connect to actual device pins should be declared as pins, and all other internal signals should be declared as nodes.
In "included" files, signals that are used in the top-level file, either as device pins or to connect to other files, or in any other included file, should be declared as pins and signals that are used only inside the same included file should be declared as nodes.

FPGA design

Device declaration

Do not specify the name of a Xilinx FPGA device with the **device** statement. Device is specified in the Project Manager.

Assigning device pins

Pin numbers cannot be given in the ABEL source file (like for Xilinx EPLD devices) - they must be specified in a separate file with the name same as the project name and the extension 'CST'. This file may be created in two ways:

1) Write all declarations using any non-formatting (ASCII) text editor. You have to provide the following entry for each pin:

```
place instance <signal_name>_#PAD_pad  :  <pin_number> ;
```

where # is:

I	- for input pins,
O	- for output pins,
IO	- for bidirectional pins.

e.g.:

```
place instance RESET_IPAD_pad  :  P56 ;
place instance Q12_OPAD_pad  :  P72;
```

2) Invoke Place & Route process without the 'CST' file - in such case, XACT will add the pin numbers on his own and adds the suitable information about that in the report file (the file with the name same as project name and the extension 'RPT'). The report file will be supplemented with the section 'CST File Format' which contains ready-made pin assignments in the format described above. This fragment can be easily cut with any text editor and saved as 'CST' file. Then the pin numbers should be suitably modified as desired. With so prepared 'CST' file the process Place & Rout must be run once again.

EPLD design

Device declaration

The following ABEL-HDL **device** statement should be specified in the header of the source ABL file used as a top-level design o as a single file design. This statement tells XABEL that this file represents complete stand-alone design. It has the following syntax:

modulename **DEVICE**;

In an included file the **device** statement should not appear.
Do not specify the name of a Xilinx EPLD device in the **device** statement. Device is specified in the Project Manager.

Assigning device pins

To assign signals to pins in a top level ABEL design for Xilinx EPLD simply specify the pin numbers in the **pin** declarations in the ABEL-HDL source file.

ABEL synthesis options

FPGA synthesis

General options:

Compile:

Chip - enables insertion of I/O buffers and pads; used for HDL type projects.
Macro - disables insertion of I/O buffers and pads; used for schematic type projects.

Improvex - enables the logic optimizer Improvex which optimizes the synthesized XNF netlist

Optimize - options for the logic optimizer Improvex:

Area - optimization for utilization (number of used CLBs), trading off performance (speed).
Speed - optimization for performance (number of CLBs in the critical path), trading off utilization (number of CLBs).
Standard - intermediate solution between Area and Speed options.

Advanced options:

Encoding - options for state-encoding method:

Standard - automatic selection of the best encoding scheme.

Binary - for small state machines, or state machines with few (less than four) input variables, binary encoded state machines may be more efficient.

One Hot - the One-Hot method uses one flip-flop per state machine state.

Unspecified state - these options specify how to synthesize incompletely specified state machines. Incompletely specified state machines are those which do not have a state assignment for every possible input combination.

Initial - incompletely specified state machine defaults to the initial state whenever the next state for the current input combination is not specified.

Current - incompletely specified state machine defaults to the current state.

Don't Care - no default for incompletely specified state machines.

EPLD synthesis

Compile:

Chip - enables insertion of I/O buffers and pads; used for HDL type projects.

Macro - disables insertion of I/O buffers and pads; used for schematic type projects.

Optimize - reduction options:

Auto Polarity - reduces the logic so that each signal will have the minimum number of terms possible. The optimization will produce both ON-set and OFF-set equations so that the minimum number of product terms will be used.

Fixed Polarity - reduces so that each signal will have the minimum number of product terms, maintaining the polarity of the signals as specified in the source file. This should only be used when you want to force output signals to a specified polarity (typically through the use of the 'pos' and 'neg' signal attributes).

No Reduction - merges all the compiled equations into a single ABEL-PLA file. No logic reduction is performed. This option must be used if you want redundant logic to be preserved for any of the design outputs.

Using Memory Generator

Requirements

Xilinx memory generator (MemGen) is a convenient tool for creating RAMs and ROMs for Xilinx FPGA families which supports memories. MemGen creates XNF file, log file and few other files used to create a schematic symbol, which can be used as a part of the design.

You need two things to create memory macro:

- Xilinx tool (MEMGEN.EXE) installed in the XACT directory,
- Text file (.MEM) containing memory description; Active-CAD helps you to create this file (see *Creating memory macro* section).

For detailed information about MemGen program and its capabilities refer to Xilinx *XACT Reference Guide, Volume 1*.

Memory definition file

Memory definition file is a simple text file (with .MEM extension) which defines memory and its contents for MemGen tool. This file consists of series of commands that specify:

- memory type,
- memory dimensions (number of words and word's length),
- symbol format and symbol pins style,
- memory contents (only for ROMs and RAMs with initial values).

Following commands may be used in the memory definition file:

TYPE [RAM|ROM]

TYPE command is used to specify the type of memory to be created. There are two types supported: RAM (read-write memory) and ROM (read-only memory).

DEPTH *number_of_words*

DEPH command is used to define the depth the memory (in words). The value of *number_of_words* parameter must be positive decimal integer between 2 and 256, inclusive.

WIDTH *word's_width*

WIDTH command is used to define the width of memory word (number of bits within word) . The value of *word's_width* parameter must be a positive decimal integer between 1 and 32, inclusive.

SYMBOL [VIEWLOGIC|ORCAD|NONE] [BUS|PINS]

SYMBOL command is used to specify symbol format and pin style. It has 2 parameters. First specifies to which schematic editor the symbol is dedicated. The second one specifies whether data and address lines should be created as individual pins (PINS) or as bus signals (BUS).

NOTES:

- for Active-CAD specify ORCAD symbol format,
- symbol is always created with individual pins, no matter what value, BUS or PINS, you specify as the symbol style. If you want to create memory symbol with bus pins you have to change symbol created by MemGen using Symbol Editor.

DEFAULT *data_value*

DEFAULT command is used to define default value of all memory cells not specified with the **data** command. If no default value is specified all undefined locations are zeroes.

NOTES:

- **DEFAULT** command is allowed only for ROM and RAM memories with initial values,

- *data_value* parameter can be specified in binary, octal, decimal or hexadecimal format described below (see. **DATA** command description**)**,

DATA *value0, value1, ... , valueN*
DATA command is used to specify the contents of ROM or RAM memory with initial values. This command, if used, must be the last one used within the memory definition file. **DATA** command may be specified in multiple lines but the DATA keyword should appear only once. Individual data values must be separated by commas or blank characters (spaces, carriage returns or tabs). You can specify data in the following formats:

- binary: 2#*value*#
- octal: 8#*value*#
- decimal: 10#*value*#
- hexadecimal: 16#value# or *value* (the default numeric base is 16)

The *value0* is location zero, the *value1* is location one, and so forth. Underscores can be used to separate data fields in long numbers which makes them easier to read (i.e. 2#0100_1100_0010_1101#).

; *comment*
All text to the right of a semicolon until the end of the line is the comment and is ignored by MemGen.

Creating memory macro

To create a memory macro you have to prepare memory definition file (.MEM) containing description of the memory. There are two ways of creating this file:

- use Active-CAD dialog . You simply fill specified fields and the memory definition file is automatically created. If you create RAM you need not add anything to this file. If you create ROM or RAM with initial values, you have to specify memory contents (**DATA** command),
- create whole memory definition file using any non-formatting (ASCII) text editor.

To create a new memory macro with Active-CAD follow the procedure:

1. Select the *Memory Generator* option from the *Applications* menu in Project Manager.
2. Enter the memory definition file name in the *Block field*. Created memory macro will have the same name as the file name.
3. Mark the memory type (RAM or ROM) and specify the memory width and depth in the appropriate fields.
4. If you create ROM or RAM with initial values press the *Edit* button to invoke Notepad. Add information about the memory contents (see *Memory definition file* section), save the file and close Notepad.
5. Press the button *Generate* to run MemGen and create the memory macro. After successful creation macro is added to the project library.
6. If an error appears view the log file (file with the same name as the .MEM file and the LOG extension) to see the error description.

To create a macro with the previously created memory definition file follow the procedure:

1 Select the *Memory Generator* option from the *Applications* menu in Project Manager.
2. Enter the memory definition file name in the field *Block* or press the button *Select* to browse the file.
3. Press the *Generate* button to run MemGen and create the memory macro. After successful creation macro is added to the project library.
4. If an error appears view the log file (file with the same name as the .MEM file and the LOG extension) to see the error description.

NOTE:
Created memory symbol will have individual pins. If you want a symbol with bus pins use Symbol Editor to modify the existing symbol.

Placing memory macro on the schematic

Memory macro is added to the project directory and you place it on the schematic like any other symbol. Use 'Symbol toolbox' button or *Symbol* option from the *Mode* menu to invoke symbol selection window. Then select the desired memory macro and place it on the sheet. See *Schematic editor* section for more details.

Changing memory macro contents

To change the memory macro follow the procedure:

1. Select the *Memory Generator* option from the *Tools* menu of the Project Manager.
2. Press the *Edit* button and select the memory definition file with the same name as the memory macro you want to modify. Notepad editor will be invoked with this file. Make desired changes in the memory definition file, save it and close Notepad.
3. Press the *Generate* button to run MemGen and update the memory macro.
4. If an error appears view the log file (file with the same name as the .MEM file and the LOG extension) to see the error description.

NOTE:
Created memory symbol will have individual pins. If you want a symbol with bus pins use Symbol Editor to modify the existing symbol.

Functional Simulation

Functional simulation methodology

Functional simulation is used for verification of design behavior. Since this simulation does not include any timing delays, the device outputs are more clearly related to inputs and it is easier to understand the device or circuit behavior.

Starting functional simulation

The functional simulation assumes either

- a zero propagation delay (**FM** mode),
- current simulation resolution (**UN** mode),
- the short simulation Step length (**GL** mode).

The simulator is set by default to the functional simulation with zero propagation delays (**FM** mode). To set other functional modes, click on the center button in the **Simulation toolbox** which has a 3x3 button arrangement. Each click on that button will select the next simulation mode (**TM**-timing mode, **UN**-unit delay functional mode, **GL**-glitch functional simulation mode).

The simulation resolution, which is used by the **UN** mode, is set by selecting the **Simulation Precision** option from the **Options** menu.

The short simulation step, which is used by the **GL** mode, is set by selecting the **Simulation Step** option from the **Utilities** menu.

Functional simulation with X-BLOX symbols

If your design includes some X-BLOX symbols, you will be able to simulate such design by clicking on the **SIM** button in the simulator or Design Manager, which invokes the simulator. ACTIVE-CAD will automatically:

1. Find on network all Xilinx design related tools, such as XNFPREP and X-BLOX, and create pointers for direct access by the Design Manager.
2. Generate an XNF netlist file from your schematic design (it will invoke the **Create Netlist** and then **Export Netlist** option from the **Options** menu)
3. Run through the XNFPREP utility to flatten the netlist
4. Invoke the X-BLOX compiler to generate a gate level netlist from the X-BLOX netlist.
5. Load the newly generated gate level netlist into the simulator and set it to the functional (**FN**) simulation mode (with 0 propagation delays).

If the XNFPREP and X-BLOX (DS380) software is not available on the network, you will have to manually perform the above listed operations, using ACTIVE-CAD and Xilinx programs.

Functional simulation with VHDL files

If your design includes some blocks described by VHDL code, you will be able to simulate such blocks by clicking on the **SIM** button in the simulator or Design Manager, which invokes the simulator. ACTIVE-CAD will automatically:

1. Find on the network all VHDL design related software, such as VHDL analyzer and logic synthesis tools, and create pointers for direct access by the Design Manager.
2. Generate an XNF netlist from your VHDL design files.
3. Load the newly generated gate level XNF netlist into the simulator and set it to the functional (**FN**) simulation mode with 0 propagation delays.

If your project is comprised entirely of VHDL files and schematic does not exist, you should invoke the simulator directly from the Design Manager.

To manually convert a VHDL file into a gate level netlist, select the **Update Macro** option in the **Project** menu of the HDL Editor. ACTIVE-CAD will select the appropriate logic synthesis tool and create the netlist. ACTIVE-CAD will automatically use these files for simulation, instead of the VHDL source files.

Functional simulation with ABEL files

If some schematic blocks are described in ABEL hardware description language, you will be able to simulate them by clicking on the **SIM** button in the schematic editor or in Design Manager, which invokes the simulator. ACTIVE-CAD will automatically:

1. Find the ABEL compiler on the network and generate a pointer for direct access by the Design Manager.
2. Generate an XNF netlist file from your ABEL design by invoking the X-ABEL software
3. Load the newly generated gate level netlist into the simulator and switch it to the functional (**FN**) simulation mode with 0 propagation delays.

If your project is comprised entirely of ABEL files and schematic does not exist, you should invoke the simulator directly from the Design Manager.

To manually convert ABEL files into gate level netlists, select the **Update Macro** option in the **Project** menu of the HDL Editor. ACTIVE-CAD will automatically select theses files for simulation instead of the ABEL source files.

Functional simulation of EPLD designs

If your EPLD 7000 design has been entered in ABLEL design language, it should be processed as a typical ABEL file.

- Create a project for the ABEL design file. Invoke the HDL editor if you need to review the design in ACTIVE-CAD. Do not start the schematic editor.
- Click on the **SIM Funct**(ional) button in the Design Manager. ACTIVE-CAD will automatically find on network the X-ABEL converter and convert the HDL design file into a flat netlist.
- The flat netlist will be loaded into the simulator for functional simulation.
- Load the desired test vector file using the **Load Test Vectors** or **Load ASCII Test Vectors** option in the **File** menu. You can also simulate the netlist with the ready-made test vectors that are available on-line in ACTIVE-CAD.

You can import the EPLD netlist into schematic editor as a schematic and simulate it interactively as a typical schematic with an on-line background simulator.

Selecting probes in the schematic

You can select any schematic test point for viewing or direct control with external signals. To select the test points, enter the PROBES assigning mode (by clicking on the **Simulation toolbox** button in the top horizontal toolbar), and click on the desired test points. The selected items will be marked with a gray square and will be automatically listed in the simulator once it is invoked.

Selecting probes in the simulator

Because of direct correlation to the design, it is much easier to place probes on schematic test points than select them from signal and chip listings in the simulator. However, if a need arises to select probes from the simulator, follow this procedure:

1. Select the **Add Signals** option from the **Signals** menu. In response, **Component Selection for Waveform Viewer** will appear.

2. **Signals Selection** field lists all signals, including I/O terminals, that can be directly selected to the simulation.
3. **Chip Selection** field lists all devices which pins can be dynamically selected for simulation. Double-clicking on a device will display its pins in the **Pins For:** field. You can select any pin by double-clicking on it, or use the group selection options, which are available from a field activated by positioning the cursor in the **Pins For** field and clicking the right mouse button.
4. To speed component search in hierarchical designs, use the **Scan Hierarchy** field. If this field is invisible on the screen, click on the **Hierarchy** button in **the Component Selection for Waveform Viewer** window.

All selected test points are displayed in the simulator's **Signals** field and on appropriate hierarchical schematic sheets.

Assigning stimulators

To apply an external signal to the schematic design, you need either to choose one of the ready-made general purpose signals that come with the simulator or create yourself an appropriate test vector.

Ready-made general purpose stimulator signals

The ACTIVE-CAD simulator comes with a set of general purpose signal waveforms, also called the stimulators. Clicking on the **Add Stimulators** option in the **Stimulator** menu displays **Stimulator Selection** window which is similar to a keyboard button arrangement. All of the buttons in that display represent signals which can be assigned to schematic test points by dragging them over the selected signal name in the simulator window and releasing the left mouse button. They can also be assigned by selecting the signal name first (turns blue) and then clicking on the appropriate signal button in the **Stimulator Selection** window.

1. The *A-Z* keyboard keys can be assigned to any number of signal lines and can be manually toggled between 0 and 1 logical levels while the simulation is in progress.
2. The round LED lamps represent outputs of a 16 stage binary counter. The input clock frequency of that counter is set from the **Clock Settings** option in the **Options** menu, and the counter can operate from the kiloHertz to the GigaHertz clock frequencies. The **Bc** lamps represent the true counter outputs and the **NBc** lamps represent the Inverse counter outputs.

Formula-based stimulator signals

The square LED lamps in the **Stimulator Selection** window represent custom signals generated from a formula entered by the user. The formula entry is activated by clicking on the **Formula** button. In response, the **Set Stimulator** window appears. Double-clicking on a formula name in the **Select Stimulator** field (F0-F15), selects it for editing in the **Formula** window.

1. Enter the logical state, e.g. *H* for high, *x* for unknown, etc.
2. Enter the length of the logical state in nanoseconds, e.g. 20.14
3. Apply parenthesis. The number after the closing parenthesis indicates how many times the formula in the parenthesis should be repeated. Unlimited nesting of formulas is allowed. For example, ((H20.14L37.86)2Z22)993 will create a waveform high for 20.14 ns, low for 37.86 ns. This waveform will be repeated twice. After the second set of H=20.14 and L=37.86, there will be tri-state for 22 ns. This waveform will be repeated 993 times. After the 993rd repetition, the waveform will stay at its last logical state (Z=tri-state).

Clock signals are repetitive signals which do not terminate in any state. If you select the **Clocks** option in the **Set Stimulator** window, a new window for assigning formula-based signals to clocks C1-C4 will appear. For example, if the formula ((H20.14L37.86)2Z22)993 is assigned to one of the C1-C4 clocks, the signal will not stay in the last state (Z) after the 993rd cycle but will instead start a new set of 993 cycles. Since the clock signal automatically repeats the basic signal infinite number of times, there is no need to place "993" after the 2nd parenthesis. Assigning to clock signal the formula ((H20.14L37.86)2Z22) will produce the same results.

The C1-C4 clock buttons that have formulas assigned to them will be displayed in red. Clock buttons without assigned signal formulas will be gray. Clock buttons allow to assign to signal names periodically repetitive custom clock signals

The *Control* buttons in the **Stimulation Selection** window allow quick and efficient operations on stimulator signals:

- **Delete** deletes the stimulator assigned to the currently selected signal line
- **EN** button enables the stimulator signal that has been disabled before
- **DS** button temporarily disconnects the stimulator from the circuit. If you do not plan to use that signal again, use the **Delete** button instead of the **DS** button.
- **CC** (Chip Controlled) button activates the stimulator only if the chip output is in the tri-state. Otherwise, the chip output will have full control of the node.
- **OV** (Stimulator Override) forces all selected signals to be unconditionally controlled by the stimulators signals. This state is assigned by default to all test vectors, so that they automatically control the associated nodes.
- **CS** (Confirmation Marker) confirms status of the associated test vector. Selecting a signal name and clicking on the **CS** button will force the currently displayed signal waveform as the driving signal. This operation allows any hand drawn waveform diagrams to be forced at schematic test points as test vectors.

Hand-drawn test vectors
You can manually edit any signal waveform on the screen and force it as a design stimuli signal. To edit a signal waveform, select the **Edit** option in the **Waveform** menu. In response, the **Test Vector State Selection** window will appear. Each time you click the cursor on the screen, a blue vertical cursor will appear at the selected location. If you click on the logical state button in the**Test Vector State Selection** window, a new segment of the waveform will be drawn between the last signal transition on the selected line and the blue cursor location. Any of the 15 logical states can be selected for the waveform by clicking on the appropriate button.

Overriding device pins

There are three basic rules in applying test vectors to devices pins.

- Each time you assign a test vector to an output pin, it will automatically control the signals in the entire node
- Assigning a test vector to an input pin will control only that input pin. All other device pins in the node will be driven from the signal source in the node.
- If you assign a signal to a pin connected to tri-state bus, it will control the entire node. It is recommended that all output pins connected to a tri-state node be set to chip override (**CC** button in the **Stimulator Selection** window), so that their outputs be overridden only it he they are in the tri-state condition.

Emulating design modifications

Since ACTIVE-CAD allows feeding signals at any test point in the design and forcing the desired circuit behavior, you can override bad design sections with the desired signal waveforms. This allows you to create signal waveforms that emulate design sections before they are actually entered on the schematic.

In a large design, you can quickly separate a circuit from other design sections by overriding its inputs with known test vectors. After the circuit has been verified locally, you can remove the overriding test vectors and let the actual design blocks control the circuit under test.

If a faulty design section has a feedback, it may be difficult to isolate the problem without opening the feedback loop. Assigning a keyboard key to an pin in the feedback loop, and toggling its status to either logical high or low, will allow operating the circuit in an open loop.

Since the device pin override works without any compilations, you can instantly check design sections by emulating its external conditions. Because the override works independent of the hierarchical levels, ACTIVE-CAD is a universal deign breadboard that allows quick design troubleshooting.

Saving signals (test vectors)

ACTIVE—CAD simulator allows for saving test vectors as either binary or ASCII files. Since the binary format is much more efficient than the ASCII, all internal simulator test vector files are saved in the binary format. The ASCII format is used primarily for interfaces with other EDA tools, such as test equipment.

1. To save the current signal waveforms as a *binary* file, select the **Save Test Vectors** option from the **File** menu. The waveforms will be saved with their sources, such as clock C1, B2 (3rd bit of the binary counter), etc. Since these are repetitive signals, you will be able to continue simulation beyond the saved and loaded waveforms.
2. To save current signal waveforms as an *ASCII* file, select the **Save ASCII Test Vectors** option from the **File** menu. The waveforms will not be saved with their sources, such as B2, C1, etc., and upon re-simulation these files will end at the last previously simulated cycle. All input signals in the ASCII files will be assigned the **CS** symbol, indicating that they are forced signals (into the schematic design). Since the **CS** markers are in gray, they may be overridden by active device outputs in the same node. To force an unconditional override, select the signal name and click on the **OV** button in the **Stimulator Selection** window.

To load a binary file select the **Load Test Vectors** option from the **File** menu. To load an ASCII file, select the **Load ASCII Test Vectors** option from the **File** menu.

Running simulation steps

The **Simulation toolbox** has **Step** and **Long Step** buttons. These buttons determine the simulation time and can be set independently from each other by selecting the **Simulation Step** option from the **Utilities** menu. The schematic editor has **SC Probes** toolbox which includes the **STEP** button and **->** (*event*) button. The **STEP** simulation interval is set in the **Simulation step** field of the **General Settings** window, which is invoked by clicking on the **View/Preferences/General** menu. This setup is independent from the **Step** setup in the simulator. The **->** button allows for a single event simulation, which is very important when monitoring detailed circuit behavior.
To review past simulation steps, select in the simulator the reference signal(s), which transitions you want to use as a reference. Next, click on the **<-** (*back step*) button in the **SC Probes** toolbox. Each activation of this button will move the blue cursor to the previous event on the selected signal line(s). The <- button allows a close examination of the relationships between various signal transitions in the design.

Cross-probing with the schematic

When you start the simulator for the first time, schematic editor creates a reference netlist. From that point on, all design changes are incrementally added to the netlist. Because of this cross-coupling between the schematic editor and simulator, all design changes are instantly visible in the simulator and all signal changes in the simulator are instantly displayed on the schematic.

Selecting a test probe in the schematic, automatically adds it to the simulator display. Similarly, adding a probe in the simulator instantly displays it on the appropriate schematic sheet.

To find a location of a signal or device pin on the schematic sheet, select the item in the simulator's **Signals** field and press the right mouse button. Select in the new menu the **Find in SC** option. The simulator instantly displays the appropriate schematic page with a pink blob over the selected pin. If you have selected an I/O terminal, then the entire node will be marked red.

Incremental design changes

If you ever want to make sure that all schematic changes are in the simulator, click on the **Add Signals** option in the **Signal** menu. This displays the **Component Selection for Waveform Viewer** window. The **Chip Selection** field will list all project components. If the design is hierarchical, select the appropriate hierarchy from the **Scan Hierarchy** field. The hierarchy display should list all components at the selected hierarchical level.

To check the simulator netlist for connectivity, click on the **Add Signals** option in the **Signal** menu. This displays the **Component Selection for Waveform Viewer** window. Place the cursor in the **Signals Selection** field, and select the signal to be traced. Click the right mouse button and when a set of options is displayed, choose the **View Connections** option. The simulator will display all pin and I/O terminal connections in the selected node. Double-clicking on any pin in the node will display the associated device with all its pins. Double-clicking on any of the device pins will display a new node with all wires connected to that pin. This procedure allows you to trace connections in the design. Some users prefer tracing netlists from tracing schematic pages because the display is less cluttered.

If you have done some extensive schematic editing, then ACTIVE-CAD may have inefficient memory usage, and you need to invoke the **Update Simulation** option from the **Options** menu.

Deleting of buses requires invoking of the **Update Simulation** option from the **Options** menu. This optimizes simulation of buses.

Updating functional simulation

The functional simulation is automatically updated when you change the schematic design. No other manual operations are needed. The only exception is deleting buses. In such a case, invoking of the **Update Simulation** option from the **Options** menu is recommended.

Design Reset In Functional Simulation
Because the CLB reset lines are physically connected only during the layout process, the pre-layout designs have unconnected CLB (flip-flop) reset pins. To compensate for this lack of common reset line in pre-layout designs, all models have been upgraded to include a reset line that is activated by the **Power on** button in the **Simulation toolbox**. To change the power on process, select the **Power On Settings** in the **Options** menu.

Simulating tri-state signals

Tri-state signals are used in bus connections. These signals are displayed as yellow squares with a letter *Z* inside. The strongest signal in the node overrides the weaker ones, such as resistive high, resistive low, high impedance unknown, etc. A table of signal overriding priorities is built into the simulator to emulate the real circuit behavior and you do not need to be concerned with the issue.

If you apply a test vector to a tri-state pin, it will control the entire node independent of other active drivers in the node. It is recommended that you select **Chip Override** (**CC** button in the **Stimulator Selection** window) for the signal applied to a tri-state output pin.

Those interested in detailed simulator operation, should refer to the *Signal States* section of *An Introduction To Simulation And Virtual Hardware*, which is available in a book format from ALDEC.

Simulating bi-directional signals

Since the strongest signal overrides the weakest ones in the node, schematic does not allow you to view the actual outputs produced by the devices. However, the simulator allows viewing the individual device outputs before they are overridden by strong signals in the node.

To view a device output before it is overridden by other strong signal in the node, select the device pin for viewing its connectivity and logical states:

- Click on the **Add Signals** option in the **Signal** menu. This displays the **Component Selection for Waveform Viewer** window.
- Double-click on the selected device in the **Chip Selection** field, which displays all device pins in the **Pins For** field.
- Choose the desired pin and click the right mouse button. Select from the new menu the **View Connections** option. The simulator will list the pin connections in the **Connections** window.
- Click the mouse cursor on the **States** button. A set of new columns will appear: **Node**, **Conv**, **Model** and **Stim**. **Model** lists the actual model output before it is overridden by another strong signal in the node.
- **Node** indicates the resulting node signal, **Conv** denotes the way model will view the input signal and **Stim** displays the directly applied external stimulator signal. The **Connections** table with States display gives you an excellent insight into what is happening in a bi-directional node.

Interface to XACT Step

This chapter discusses issues connected with processing XACT5 and XACT step6 projects.

Loading project into XACT6 system

Before you load a project into XACT6, you have to export its ALDEC netlist into the XNF format. You can do this in the Schematic Editor using the **Export netlist** option from the **Options** menu. XACT6 is invoked from the design flow by clicking the **XACT** button. If the XNF netlist is older than ALDEC schematic netlist, the Project Manager suggests to update it. In case the XNF netlist does not exist at all, the Project Manager asks whether to create the netlist (if so, you should answer 'Yes').

Before the Project Manager invokes XACT6 system, it creates in the current project directory subdirectory XPROJECT, and inserts the XACT project description file, **<project_name>.prj**, into it. XACT6 keeps in the XPROJECT directory its working files.

The first thing you have to do in the XACT6 Design Manager is to execute the **Translate** command from the **Design** menu. This command opens the Translate Options dialog box. Make sure that the option **Read Part From Design** is checked. The Design Manager automatically translates the project netlist and creates a new version icon in the Project View. When translation is finished, suitable message box will appear on the screen. You may view the translation log file if you want.

NOTE: If you have modified your project and you want to have it processed in XACT6 in order to update the post-process (routed) netlist, you must re-translate the new version of the XNF netlist, thus creating a new version of the project in XACT6 Design Manager (see the next section).

Revision control

XACT6 Design Manager allows to create multiple versions of the same design. Each version has its own merged netlist XFF. The XFF netlist is generated from the XNF netlist in the translation process. Note, that ACTIVE-CAD allows you to keep only single-version projects. This means that if you modify the project contents and export a new XNF netlist, the old XNF netlist will be lost (overwritten). Summarizing, though a project processed in XACT6 may have multiple version (each with its own XFF netlist), ACTIVE-CAD keeps only the latest (i.e., created during the latest **Export Netlist** operation) version of the XNF netlist.

After you create a design version by translating your ACTIVE-CAD project, you can try different implementation strategies on that design. This allows you to vary how your design is implemented in order to achieve your design objectives. For example, you can maximize speed and density for specific functions in your design by controlling the implementation process. Each of these implementation strategies is called an *implementation revision*. Each implementation revision contains the output files and reports that are created based on a specific set of implementation strategies. You can delete implementation revisions that are no longer useful.

When you create a project, a revision icon is placed in the Project View with the status indicated as translated. As you process the implementation, its status (i.e. routed or placed) is indicated next to its icon in the Project View.
You use the New Revision command in the Design menu to create a new revision. This creates a revision that contains just the translated netlist and the status is "translated."

SUMMARY:
To Create a New Implementation Revision in the Design Manager Project View, do one of the following:
· Choose New Revision from the Design Manager's Design menu.
· Click the right mouse button in the Project View and select New Revision from the pop-up menu.
The Design Manager creates a new revision and displays its icon in the Project View.

Routing design with XACT 6

When a project is translated you can proceed to routing process. In the Project View select the implementation revision you want to process and choose the **Implementation** command from the **Design** menu. As a result, the **Design Implementation Option** window will open. Set the options according to the desired implementation strategy and click the **Run** button. This will invoke the Flow Engine window and start the operation. In the Flow Engine window you can watch the status of particular phases of the operation and messages sent in the process. End of the processing of the project is signaled by a message box. Clicking **OK** closes the Flow Engine window.

NOTE: Before processing, make sure that the **Produce Timing Simulation Data** option box in the **Design Implementation Option** window is checked.

Finding DRC errors on the schematic

If you encounter DRC errors during processing a project in XACT6, you can use a simple procedure, described below, to find the symbols reported in the DRC report file:
1. In the Report Browser window, double-click the **DRC Report** icon in order to view its contents in a text editor.
2. Find the fragment of an error description containing symbol reference path. Select the path and copy it to the Windows Clipboard (e.g., by pressing Ctrl+C)
3. Switch to the ACTIVE-CAD Project Manager and select the **Find Object** option from the **Document** menu. Place the cursor in the **Symbol Reference Path** edit box and paste the contents of the Clipboard (e.g., by pressing Ctrl+V). Click the **Find** button. The Schematic Editor will start with the appropriate schematic loaded. The specified symbol will be selected on the schematic.

Using Hardware Debugger and PROM File Formatter

The Hardware Debugger is Xilinx configuration and verification tool. Use it to download PROM and BIT files to FPGAs to program the devices. You may also use it to read back signal values for debugging purposes. Specifically, you can read internal states of registers, flip-flops, RAMs, and I/Os, and verify the configuration data for a device. The program can be used with three different cables: parallel, serial, and XChecker cables.

Each cable allows you to download data from your computer directly to one or several FPGA devices. If you are using an XChecker cable, you can also do readback and verification with one device at a time. You can then display the readback data in waveforms based on readback snapshots.

Use the Hardware Debugger to do the following tasks:
- download a BIT file to an FPGA,
- download a PROM file to a daisy chain,
- verify configuration data using an XChecker cable,
- read back the internal logic states of a configured device using the XChecker cable.

The PROM File Formatter provides a graphical user interface that allows you to format BIT files into a PROM file compatible with Xilinx and third-party PROM programmers. It is also used to concatenate multiple bitstreams into a single PROM file for daisy chain applications. Most of all, the PROM File Formatter enables you to take advantage of the Xilinx FPGA reconfiguration capability, as you can store several applications in the same PROM file.

PROM files are also compatible with the Hardware Debugger. You can therefore use the Hardware Debugger to download a PROM file to a single FPGA or to a daisy chain of FPGA devices.

Both the Hardware Debugger and the PROM File Formatter can be invoked either from the XACT6 Design Manager (appropriate icons or items in the **Tools menu**) or immediately from the ACTIVE-CAD Project Manager (the **Prog/Debug** button in the Design Flow).

Using DOS-based XACT Tools in XACT5 projects

This section provides information that may be useful when the user must, for some reasons, run XACT applications from the command line. The term 'FPGAs', as it is used below, denotes devices belonging to one of the Xilinx families other than XC7200 or XC7300; the term 'EPLDs' denotes devices belonging to XC7200 or XC7300 family.

Functional simulation

In general, functional simulation is based on ALDEC binary netlist (file **project.alb**) generated directly from the schematic. However, for some project this method cannot be used:
- top level HDL projects have no schematic files, and functional simulation must be performed on the synthesized XNF netlist (If the synthesized XNF netlist includes X-BLOX macros, it must be converted first so that it can be simulated - see below),
- schematic projects using X-BLOX macros – schematic netlist must be exported to the XNF format, and then the XNF netlist must be converted so that it can be simulated - see below.

If the project XNF netlist includes X-BLOX macros, the following programs must be run before functional simulation (the following are complete command lines in order of execution):
xnfmerge.exe <project>
xnfprep.exe <project> outfile=<project>.xtg
xblox.exe <project>
As the result, the **<project>.xg** netlist file is generated, which can be simulated. This netlist is loaded into the Simulator.

Functional simulation of projects for EPLDs, which include ABEL macros must also be performed on the external netlist. Before simulation the following applications must be run:
xnfprep.exe -f -d <xact_path>\data\xnf7000 <project>.xnf
fsim.exe <project>.xff <project>.xf
The **fsim** program generate netlist **<project>.xf** which can be loaded into the Simulator

Place & Route processing

The essential file for P&R processing is the XNF netlist obtained either by exporting ALDEC schematic netlist or by synthesizing top level HDL source files.

P&R for FPGAs is performed by the **xmake** program:
xmake.exe -x <project>

P&R for EPLDs is performed by the **xemake6** program:
xemake6.exe -x <project>

Timing simulation

Before timing simulation of projects for FPGAs, the following programs must be run:
lca2xnf.exe -g <project> <project>.xnr
xnfba.exe <project> <project>.xnr -o<project>.bax -r<project>.bxr
The backannotated netlist **<project>.bax** can be loaded into the Simulator.

Before timing simulation of projects for EPLDs, the following program must be run:
tsim.exe <project> <project>.xnt
The backannotated netlist **<project>.xnt** can be loaded into the Simulator.

Timing Simulation

Timing simulation methodology

To perform timing simulation, ACTIVE-CAD updates the functional simulation netlist with the PDF timing parameter file. This file is generated by the propagation delay calculation software, which is a part of the placement and layout software tools.

If you have loaded a pre-layout netlist into simulator and selected the TIM (timing mode), then ACTIVE-CAD will assign 1 nanosecond to all cells except gate primitives.

Creating timing simulation netlist

ACTIVE-CAD automatically generates timing simulation netlist when you click on the **SIM Timing** button in the Design Manager. This operation requires that the appropriate XILINX design tools reside on the network.

Starting timing simulation

You can automatically invoke timing simulation only if the Xilinx design tools (X-ACT) are present on the network. To start timing simulation, click on the **SIM Timing** button in the Design Manager.

If the Xilinx design tools are not present on the network, you need to compile off line the XNF functional design netlist under the X-ACT software. Next, you need to import the newly generated post-layout netlist into simulator by selecting the **Load Netlist** option in the simulator's **File** menu.

Loading test vectors

Generally, you should use for timing simulation the same test vector files that you have used for functional simulation. To load a test vector file, select either **Load Test Vectors** option or **Load ASCII Test Vectors** option from the **File** menu.

Using global reset

The FPGA devices have a reset line connected to all active flip-flops during the layout process. Unlike the pre-layout simulation which has been reset by the **Power on** button, the post-layout simulation requires selecting to the **Signals** field of the physical (on-chip) reset line such as GSR or GS. This reset line needs to be manually toggled before the timing simulation process may commence.

Reviewing timing delays

The timing delays are displayed in a tabular form, listing the minimum, average and maximum CLB and IOB cell propagation delays (some values may be missing for some of the devices). To display the post-layout timing delays, click on the **Edit Timing Specification** option in the **Patching** menu. The delays for input pins represent the line delays and are designated **DEL pin parameters**.

You can edit the listed propagation delays to check what effect they have on the design behavior. After emulating some changes, you may have a general idea on how to change the design parameters so that a new compilation will produce the desired propagation delay values. For example, you may group some design sections or assign a higher routing priority to some signal lines.

Scaling device timing

The Xilinx devices come in several speed grades. If your design is failing for one speed grade device, it may be operating correctly with another, higher speed device. However, recompiling the same design for different devices takes a long time. ACTIVE-CAD offers a quick and much simpler solution that may point directly to the correct device. Follow this simple procedure for finding the appropriate higher speed device:

- Find the *largest timing violation* (LTV) in the design, e.g. 3.4 nsec.
- Calculate the *fastest clock period* (FCP) in the design. For example if the max. clock is 40 MHz, then its fastest clock period will be 25 nsec.
- Add the largest timing violation (LTV) to FCP (3.4 nsec + 25 nsec = 28.4 nsec), and call the result DCP (*desired clock period*).
- Calculate the propagation *delay scaling factor* (DSF) as the ratio of PFC (current clock period) to the DCP (desired clock period). For example, DSF = FCP/DCP = 25/28.3 = .88
- Select **Edit Timing Specification** from the **Patching** menu.
- Double-click on **Root** in the **Chip Selection** field.
- When the **Set Time For Block** window appears, select the **Level** Mode which affects all items in the entire design, and enter the DFS factor into the **% of max** field (e. g. .88). Click the **OK** button to complete the operation of rescaling of all propagation delays in the entire design.
- Continue simulation. If any new timing violations appear, repeat the process of DSF calculation, till the design shows no additional timing violations.

  If the design has passed all test vectors after its propagation delays have been rescaled with the DSF factor, then the design should work flawlessly with the original propagation delays when used with appropriate higher speed device. The clock speed of the new device is calculated by dividing the current rating of the device by the DSF factor. For example, if the current device has been graded at 50 MHZ, then 50MHz/.88=62.5 MHz.

  Compile your design for the closest device that exceeds the 62.5 MHz clock speed and verify its operation by simulating the previous test vectors.

Timing violations

ACTIVE-CAD automatically detects and displays such timing violations as setup and hold violations, minimum clock width, and similar. Clicking on the **Error Reporting** option in the **Options** menu displays the **Error Reporting Options** window, which is used for selecting error types that will be stored and displayed.

Selecting the **Error Viewer** option from the **Utilities** menu displays **the Error Report Viewer** window which allows selecting the time period for gathering design errors.

Tracing simulation errors

ACTIVE-CAD timing violations messages are detailed enough so that they can point you directly to the source of errors. For better understanding of the timing delays that have caused violations, select the **Edit Timing Specification** option in the **Patching** menu.

Sometimes the simulation errors are related to unwanted logical states, e.g. you are getting unknown state (*X*) instead of logical *1*. To find the cause of such design errors, select the device pin that shows the wrong signal state and trace its connectivity and logical states. The process of tracing signals is listed below:

- Click on the **Add Signals** option in the **Signal** menu. This displays the **Component Selection for Waveform Viewer** window.
- Double-click on the selected device in the **Chip Selection** field, which displays all device pins in the **Pins For** field.

- Choose the desired pin and click the right mouse button. Select from the new menu the **View Connections** option. The simulator will list the pin connections in the **Connections** window.
- Click on the **States** button. A new table with **Node**, **Conv**, **Model** and **Stim** columns will appear. The **Model** column lists the actual model output before it is overridden by another strong signal in the node. Analyze this signal. If it is incorrect, trace it further. If it is correct, look for a problem in the current signal node

Learn how to effectively trace signals. It is one of the basic skills required in design analysis.

Resolving bus conflicts

ACTIVE-CAD explicitly reports all bus conflicts. To view a device output before it is overridden by other strong signals in the node, select the device pin for connectivity and states viewing:

- Click on the **Add Signals** option in the **Signal** menu. This displays the **Component Selection for Waveform Viewer** window.
- Double-click on the selected device in the **Chip Selection** field, which displays all device pins in the **Pins For** field.
- Choose the desired pin and click the right mouse button. Select from the new menu the **View Connections** option. The simulator will list the pin connections in the **Connections** window.
- Click on the **States** button. A new set of columns will appear: **Node**, **Conv**, **Model** and **Stim**. The **Model** column lists the actual model outputs before it is overridden by another (strong) signal in the node. Analyze this signal. If it is incorrect, trace it further. If it is correct, look for a problem in the current signal node.

The **Connections** table with States display gives you an excellent insight into what is happening in a bi-directional node.

Simulating external netlist

ACTIVE-CAD automatically loads post-layout netlists into the simulator. However, if the design netlist was created in a different EDA tool or if the Xilinx tools were not available on-line, you may have to manually import an external netlist.
To manually import an external netlist, click on the **Load Netlist** option in the **File** menu. ACTIVE-CAD displays the **Load Netlist** window, which allows selecting the input netlist format and its network/drive location.

EPLD Design Issues

This chapter explains how to create your schematic design so it can be fitted to an EPLD device. It describes XC7000 primitives and macros and the EPLD-specific attributes. It explains how to take advantage of the EPLD features, how to adhere to the EPLD design rules, and how to assign logical High and Low values to unconnected inputs and symbols. This chapter also explains how to control logic optimization and other fitting options.

Schematic Library Components

To complete an EPLD design, you must use only the components included in the X7000U library. The X7000U library contains components common to EPLD and FPGA designs. A few components are specific to EPLD. This section describes how to use the common and EPLD-specific components.
There are three basic types of components:

- Buffers or pads, which define input and output ports that represent physical pins on the device
- Standard components, which represent fixed logic functions such as gates, adders, and counters
- PLD components, which are defined with a PLUSASM equation file

Many of the components in the X7000U library have special features that take advantage of EPLD architecture. The following sections describe some of those features.

Buffers and Pads

Input and Output Buffer Connections

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

Input Buffers

To take advantage of the input-pad registers and latches available in EPLD devices, use one of the IFD, IFDX1, or ILD symbols instead of the IBUF; do not connect an IBUF to the D input of an IFD/ILD symbol. Refer to the XACT Libraries Guide for specific application rules for the symbols.
To represent an ordinary device output pin, use an OBUF buffer that is driven by one (and only one) on-chip logic source. Connect the output of the OBUF to an OPAD symbol. You could also use one of the 3-state output buffers (OBUFE or OBUFT) instead of OBUF. Drive the output enable control input (E or T) using any on-chip logic source or input signal (from IBUF). The EPLD fitter looks for opportunities to automatically assign the enable signal to one of the EPLD's fast output enable (FOE) global enable lines.
If you want to take advantage of an FOE global line explicitly, use a BUFFOE input buffer instead of IBUF, and connect it to an OBUFEX1 output buffer instead of OBUFE.

Assigning an FOE Line

You should always label all the nets connecting PAD symbols and input/output buffer symbols. These names are used by the software to refer to your device pins in the reports and during simulation.
To represent a bi-directional I/O pin, use an IOPAD symbol connected to both the input of an IBUF or IFD/ILD and the output of an OBUFE, OBUFT, or OBUFEX1.

Output and 3-State Buffers

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

Output Enable Behavior in EPLDs

If you use a PLD symbol in your schematic and connect one of its outputs to an output buffer like OBUF, you can control the EPLD device output pin using a 3-state control equation in the PLD.

Controlling Output Using a PLD Equation

If you want to use a PLD output with a TRST equation to control
a bi-directional I/O pin of the EPLD, connect the OBUF output to an IOPAD and IBUF or IFD/ILD. If the same PLD symbol that generates the output is also to receive the I/O pin input, you must use a separate pin of the PLD symbol to receive the signal from the IBUF. Do not tie the signal received from an IBUF to the net driving the OBUF of the same IOPAD; these input and output nets must remain separate.

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

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

On-Chip 3-State Multiplexing

EPLD components emulate 3-state signals internally by gating the macrocell feedback to the universal interconnect matrix (UIM). (Macrocell feedback signals are never physically in a high-impedance state.) You can tie together the outputs of multiple 3-state buffer symbols like BUFE or BUFT or 3-state PLD outputs to multiplex these signals on-chip. You cannot connect such tied signals to an output buffer; you must pass a tied signal through a logic symbol like BUF before driving an output port.

Input Buffers, Clocks, and Global Control Nets

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

The input of a BUFG symbol must connect directly to a PAD symbol representing a FastCLK pin; there can be no other components between the PAD and the BUFG.

IFD and ILD components must have a BUFG clock input.

After assigning any BUFGs to FastCLK pins, the XEPLD software tries to assign IBUF signals that drive only clock inputs onto the remaining FastCLK pins.

The XEPLD software also attempts to optimize FD components into IFDs on the input pads. No other registers are ever optimized into the input pad.

If your design requires a global clock enable, you must use IFDX1 components. The CE input to these components can only be driven by a BUFCE, and the clock must be from a BUFG.

Use of the IFDX1 Symbol

You can prevent input register optimization using the REG_OPT=OFF attribute. You can prevent clock optimization using the CLOCK_OPT=OFF attribute. These attributes are described in detail in the “Attributes” section later in this chapter.

Sample Designs and Tutorials

This table describes example projects that can be used to demonstrate ACTIVE-CAD Xilinx interface features. The table comprises of the following entries:

Family - indicates which Xilinx product families are supported by the project

Author - indicates where the project was created,
VL denotes projects imported from ViewLogic format

Family - describes design entry tools that were used to create the project:
S - Schematic Editor
H - HDL Editor (HDE)

HDL - indicates that the project contains macros based on HDL description:
A - ABEL macros
V - VHDL macros

Verified - indicates whether the project was verified on the Xilinx Demoboard XC40xx-PC84

Project	Author	Family	Tools	HDL	Verified
ABELTEMP	ALDEC	4k	S,H	A	
ATIMER	ALDEC	4k	S,H	A	X
BUS_CTRL	ALDEC	4k	H	V	
CALC	VL	4k	S		
CALC3KA	VL	3k	S		
CALC5K	VL	5k	S,H	A	
CALC7K	VL	7k	S		
DACDEMO	VL	4k	S		
FIB	ALDEC	4k	S		X
FIBABEL	ALDEC	4k	H	A	X
FIBABL7K	ALDEC	7k	H	A	
FIBXBLOX	ALDEC	4k	S		X
FIBVHDL	ALDEC	4k	H	V	X
FLASH	ALDEC	4k	S,H	V	X
FIFO	VL	4ke	S		
MULTI	ALDEC	7k	S		
UARTTOP	VL	7k	S		
VHDLTEMP	ALDEC	4k	S,H	V	
VTIMER	ALDEC	4k	H	V	X
XDACDEMO	VL	4k	S,H	A	

FLASH Introductory Tutorial

Introductory XILINX Tutorial is based on the FLASH design provided with the ACTIVE-CAD. While reading the tutorial the user recreates FLASH design from scratch. Following topics are covered:

- creating a new project
- verify project libraries
- starting the schematic editor
- placing symbols
- drawing wires

- adding pin locations
- schematic changes
- importing a ViewLogic macro schematic
- repeating net names
- creating a new symbol
- creating a VHDL macro
- editing the VHDL code
- finding VHDL errors
- running synthesis
- creating a schematic macro
- connecting VCC and GND
- placing bus taps
- saving the macro schematic
- finishing the top level schematic
- creating test vectors
- running simulation
- viewing results on the
- adding timespec symbol
- running XACT

The tutorial is available in Acrobat Reader format.

XCALC Advanced Tutorial

XCALC Advanced Tutorial available in Acrobat Reader format covers following topics:

- copying CALC project to XCALC
- opening XCALC project
- importing ViewLogic macro
- editing the imported macro
- saving macro as the schematic sheet
- assigning alu_blox.sch to the alu symbol
- replacing schematic macro with Abel file
- functional simulation
- op-codes for XCALC calculator

XCALC design is a simple calculator performing basic arithmetic and logic operations on 4-bit numbers.
DESING TYPE:
    XILINX (chip XC4003APC84-4)
CONTROLS:
    Set of switches SW7/SW6..SW7/SW0 for opcode/data setting,
    Switch EXC_P for operation execution.
OUTPUTS:
    7-segment LED display for ALU register monitoring,
    4 LEDs for top stack cell monitoring.
OPERATION:
    Arithmetic and logic operations are performed on 4-bit ALU register;
    if two operands are required the second one is taken either from
    SW3..SW0 switches or from the top cell of the stack.
    Operations are coded on SW6..SW4 switches (stack-dependent
    operations use SW3..SW1, too).
    EXC_P low for at least two CLK pulses executes operation.
TIMINGS:
    FUNCT - prerouted design simulation results; only selection of
        operations tested.

ROUTED - routed design timing simulation results; full set of operations tested.

NOTE: Switch on comments viewing in the simulator to see symbolic operation codes on the timing!!!

FIB Timing Simulation Tutorial

Timing Simulation Tutorial (available in Acrobat Reader format) is based on the FIB sample design provided with ACTIVE-CAD. The design generates the Fibonacci number sequence, where each new number is a sum of two previous numbers. Following topics are covered:

- selecting signals for simulation
- assigning stimuli
- running simulation
- measuring delays on the waveform
- defining buses
- waveform deletion
- observing and tracing timing violations
- reviewing timing delays
- editing timing specification

Comparison of Schematic, ABEL and VHDL Design Methodology

Comparison of the synthesis results for different methods of the design description

Sample project of Fibonacci generator contains logic and arithmetic operations which makes it a good subject for testing and comparison.

Generally better results are for synthesis with X-BLOX macros.

ABEL synthesis is the worst. The main reason is that ABEL synthesizes arithmetic function as simple combinatorial blocks and cannot use CLB fast carry logic (in the project FIBABEL a directive '@CARRY 1' is used to optimize design for area; it causes that adder is synthesized as a ripple carry adder; without this directive the adder is synthesized as a three-level combinatorial circuit which produces hundreds of products for 8-bit adder in this design !).
Synthesis from VHDL gives good results, especially when X-BLOX is used. The results for sample project are almost like for the synthesis from schematic (without X-BLOX).

VHDL gives the possibility of improvement in the design by mixed behavioral and structural description (-> VHDLm project shows the best implementation of the display decoder is the memory; replacing equations for this decoder with a PROM element saves 6 CLB's).

<u>Synthesis results (all optimized for AREA):</u>

ABEL   - Project FIBABEL

Schematic & ABEL macro; schematic contains only input/output buffers (it is necessary because there is no possibility to add pin locations in ABEL design file);

Optimize: Area,   Improvex +

VHDL - Three versions of the project FIBVHDL

All projects are top-level VHDL

VHDL - without X-Blox,
VHDLx - with X-Blox,
VHDLm - with X-Blox and memories as a display decoders

Optimize: Area, Improvex +

SCH - Project FIB

Schematic, standard macros only

XBLOX - Project FIBXBLOX

Schematic, X-Blox macros optimized for area

	ABEL	VHDL	VHDLx	VHDLm	SCH	XBLOX
Occupied CLBs	52	41	38	32	37	25
Bonded I/O Pins	25	25	25	25	25	25
F and G Function Generators (*)	62	40	36	33	42	26
H Function Generators	16	7	8	4	2	2
CLB Flip Flops	16	16	16	16	16	16
CLB Fast Carry Logic			5	5	5	5

(*) If RAM/ROM elements are present in the design, this count includes the function generators used for them. A 16x1 memory uses 1 function generator; a 32x1 uses two.

FIBXBLOX

FIBXBLOX is an X-BLOX based project of the Fibonacci generator
Project verified on the XC4000 Demoboard.

DESIGN TYPE:
XILINX (chip XC4003PC84-5)
CONTROLS (Inputs):
CLK - clock input; every falling edge causes the generation of a number,
RESET - asynchronous reset; forcing LOW starts the process from the initial values (0 and 1),
ENABLE - clock enable; forcing LOW freezes the generator (the numbers are not generated).
OUTPUTS:
FIBINV [7..0] - generated number (binary); it is INVERTED so as to correctly drive LED displays,
DISPA [6..0] - 7-segment display of more significant four bits,
DISPB [6..0] - 7-segment display of less significant four bits.
DESCRIPTION:
The Fibonacci generator is a simple pseudo-random number generator.
Each number is a sum of the two previous ones: Fn = Fn-1 + Fn-2.
For example, if two first numbers are 0 and 1, the generated sequence is:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...
Note that if two first numbers are zeroes, the generator will not start.
TIMINGS:
FIB_FUN - prerouted design simulation results;
FIB_TIM - routed design timing simulation results;

FIBVHDL

FIBVHDL is a top level VHDL project of the Fibonacci generator
Project verified on the XC4000 Demoboard.

DESIGN TYPE:
XILINX (chip XC4003PC84-5)
CONTROLS (Inputs):
CLK - clock input; every falling edge causes the generation of a number,
RESET - asynchronous reset; forcing LOW starts the process from the initial values (0 and 1),
ENABLE - clock enable; forcing LOW freezes the generator (the numbers are not generated).
OUTPUTS:
FIBINV [7..0] - generated number (binary); it is INVERTED so as to correctly drive LED displays,
DISPA [6..0] - 7-segment display of more significant four bits,
DISPB [6..0] - 7-segment display of less significant four bits.
DESCRIPTION:
The Fibonacci generator is a simple pseudo-random number generator.
Each number is a sum of the two previous ones: Fn = Fn-1 + Fn-2.
For example, if two first numbers are 0 and 1, the generated sequence is:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...
Note that if two first numbers are zeroes, the generator will not start.
TIMINGS:
FIB_FUN - prerouted design simulation results;
FIB_TIM - routed design timing simulation results;

FIBABEL

FIBABEL is a top level ABEL project of the Fibonacci generator.
Project verified on the XC4000 Demoboard.

DESIGN TYPE:
XILINX (chip XC4003PC84-5)
CONTROLS (Inputs):
CLK - clock input; every falling edge causes the generation of a number,
RESET - asynchronous reset; forcing LOW starts the process from the initial values (0 and 1),
ENABLE - clock enable; forcing LOW freezes the generator (the numbers are not generated).
OUTPUTS:
FIBINV [7..0] - generated number (binary); it is INVERTED so as to correctly drive LED displays,
DISPA [6..0] - 7-segment display of more significant four bits,
DISPB [6..0] - 7-segment display of less significant four bits.
DESCRIPTION:
The Fibonacci generator is a simple pseudo-random number generator.
Each number is a sum of the two previous ones: Fn = Fn-1 + Fn-2.
For example, if two first numbers are 0 and 1, the generated sequence is:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...
Note that if two first numbers are zeroes, the generator will not start.
TIMINGS:

FIB_FUN - prerouted design simulation results;
FIB_TIM - routed design timing simulation results;

DECLARATIONS OF PIN NUMBERS IN ABEL TOP LEVEL PROJECTS WITH FPGA DEVICES

Pin numbers cannot be given in the ABEL source file (like for XC7000 family devices) - they must be specified in a separate file with the name same as the project name and the extension 'CST'. This file may be created in two ways:

1) Write all declarations using any non-formatting (ASCII) text editor. You have to provide the following entry for each pin:

```
  place instance <signal_name>_IPAD_pad  :  <pin_number> ;
```

e.g.:

```
      place instance RESET_IPAD_pad  :  P56 ;
```

2) Invoke Place & Route process without the 'CST' file - in such case, XACT will add the pin numbers on his own and adds the suitable information about that in the report file (the file with the name same as project name and the extension 'RPT'). The report file will be supplemented with the section 'CST' File Format' which contains ready-made pin assignments in the format described above. This fragment can be easily cut with any text editor and saved as 'CST' file. Then the pin numbers should be suitably modified as desired. With so prepared 'CST' file the process Place & Rout must be run once again.

WARNING: The synthesis of an ABEL file (in HDL Editor) must be done with the option chip set, which causes automatic insertion of I/O buffers and pads.

DACDEMO

DACDEMO is a sample design presented by Xilinx on the DAC conference. Its function is to display scrolling message on the LED matrix display. The message data is fetched from the ROM, processed and placed on the output in the form suitable for driving the display.

FIFO

FIFO is a 16x32 First In-First Out stack with simultaneous read/write operations.
DESING TYPE:
  XILINX (chip XC4020EHQ240-3)
CONTROLS:
  DIP[31:0] - data input,
  PUSHP - write-to-stack control
  POPP - read-from-stack control
  CLKP - clock
OUTPUTS:
  DOP[31:0] - data output
  FULLP - stack-full signal
  EMPTY - stack-empty signal
  LASTP - last-free-cell signal
OPERATION:
  This 16-cell 32-bit wide First-In-First-Out stack is designed with XC4000E-specific dual port RAMs. When PUSHP is active, data from DIP pins is put on the stack on the CLKP transition from low to high.

When POPP is high, the first available data from stack is sent to the DOP pins on the rising CLKP edge. EMPTY goes high when no data is available on the stack during read operation. LASTP high signals that there is only one cell available for writing, FULLP high means that FIFO cannot store more data.

TIMINGS:

FIFO_FUN - functional simulation results
FIFO_TIM - routed design timing simulation results
FIFO_SIM - timing simulation with simultaneous read/write
SIGNAL - empty timing (with selected signal names)

BUS_CTRL

BUS_CTRL project is an example of VHDL design with multiple VHDL files at the top level. This type of project is called VHDL Top Level project.

VHDL files, currently included in the project, are shown in the Project Manager Hierarchy Browser. Each file can be edited in HDL Editor by double clicking on the name. Synthesis options are common for the entire design and can be set while editing any one of these project files.

The synthesis tool is always invoked to compile all VHDL files with selected options. In case of the Schematic Top Level project type, synthesis option for VHDL macros are set separately for each macro. Please note that VHDL macros on the schematic designs can only be associated with a single VHDL file.

Creating a VHDL Top Level project comprises the following steps:

1. Create the new project
2. Add desired VHDL files to the project
3. Set synthesis options
4. Synthesize the design into an XNF netlist
5. Simulate design before Place and Route
6. Run Place and Route
7. Perform Timing Simulation based on the backannotated XNF netlist

The synthesis process is started from HDL by executing Synthesis -> Synthesize command. Synthesis process can also be invoked from Project Manager by pressing SIM Funct, SIM Timing, Place & Route or Static Timing buttons Synthesis process is updated automatically in the following cases:

- XNF netlist does not exist
- one of the VHDL project files has changed after the last synthesis run

MULTI

MULTI design is a circuit multiplying two hex digits read from the MPD_IN and MPR_IN inputs. The product is present on the RESULT output.

DESIGN TYPE:

XILINX (chip XC73108-10BG225)

CONTROLS (Inputs):

CLK - clock input; internally transformed to two-phase clock
RESET - asynchronous reset controlling two-phase clock generator
MPD_IN[0:3], MPR_IN[0:3] - data inputs

OUTPUTS:

  RESULT[0:7] - product output

  PHASE1_CLK, PHASE2_CLK - two-phase clock outputs

OPERATION:

  Input data is read on the rising edge of the PHASE1_CLK and fed through the multiplying matrix. The result is stored in the output register on the rising PHASE2_CLK edge.

TEST VECTORS:

  MULTI_FN.TVE - functional simulation results

  MULTI_TM.TVE - timing simulation results

UART

UARTTOP is a project of a simple asynchronous serial receiver

DESIGN TYPE:

  XILINX (chip XC7354-68, any speed)

DESCRIPTION:

A complete description of the UARTTOP design is provided in Session 6 of the XEPLD Tutorial chapter of the Xilinx ViewLogic Interface User Guide.

Appendix

Attributes, Constraints, and Carry Logic

This chapter lists and describes all the attributes and constraints that you can use with your schematic entry software or enter in a constraints file. In particular, it describes the relative location (RLOC) constraint. It also describes PPR placement constraints, relationally placed macros (RPMs), and carry logic.
Attributes are instructions placed on symbols or nets in an FPGA or EPLD schematic to indicate their placement, implementation, naming, directionality, and so forth. This information is used by the design implementation software during placement and routing of a design. Constraints, which are a type, or subset, of attributes, are used only to indicate where an element should be placed.
All the attributes listed in this chapter are available in the schematic entry tools directly supported by Xilinx unless otherwise noted, but some may not be available in textual entry methods such as VHDL.
Attributes applicable only to a certain schematic entry tool are described in the documentation for that tool.
For third-party interfaces, consult the interface user guides for information on which attributes are available and how they are used.

Attributes

There are three types of attributes discussed in this section:

- Component attributes, which affect only the component instances on which they are placed.
- Global attributes, which affect the entire design. These attributes apply to EPLD devices only.
- Net attributes, which affect individual component outputs or inputs and are represented by attributes applied to nets.

In some software programs, particularly Mentor Graphic's, attributes are called properties, but their functionality is the same as that of attributes. In this document, they are referred to as attributes.
There are two types of Mentor Graphics properties: in one, a property is equal to a value, for example, LOC=AA; in the other, the property name and the value are the same, for example, DECODE. In the first type, "attribute" refers to the property; in the second, "attribute" refers to the property and the value.
The attributes in this section are listed in alphabetical order.

BASE

Architectures

The BASE attribute applies to the XC2000 and XC3000 families only.

Description

The BASE attribute defines the base configuration of a CLB or an IOB. For an IOB primitive, it should always be set to IO. For a CLB primitive, it can be one of three modes in which the CLB function generator operates.
In XC2000 devices, these three modes are the following:

- F mode allows any function of up to four variables to be implemented, where one of the inputs can be the Q output of the flip-flop in the CLB.
- FG mode allows two three-input functions to be implemented, where the input can be chosen from the four inputs to the CLB and the Q output of the flip-flop in the CLB.
- FGM mode is similar to FG mode except that the inputs must be chosen from four inputs to the CLB or the Q feedback. The B input to the CLB acts as the control for a multiplexer between the two four-input functions.
- The three modes are very similar in XC3000 devices:
- F mode allows the CLB to implement any one function of up to five variables.
- FG mode gives the CLB any two functions of up to four variables. Of the two sets of four variables, one input (A) must be common, two (B and C) can be either independent inputs or feedback from the Qx and

Qy outputs of the flip-flops within the CLB, and the fourth can be either of the two other inputs to the CLB (D and E).

- FGM mode is similar to FG, but the fourth input must be the D input. The E input is then used to control a multiplexer between the two four-input functions, allowing some six- and seven-input functions to be implemented.

Syntax

The syntax of the BASE attribute is the following:

BASE=mode

where mode can be F, FG, or FGM for a CLB, or IO for an IOB.

BLKNM

Architectures

The BLKNM attribute applies to all FPGA families.

Description

The BLKNM attribute assigns LCA block names to qualifying primitives and logic elements. If the same BLKNM attribute is assigned to more than one instance, the software attempts to map them into the same LCA block. Conversely, two symbols with different BLKNM names are not mapped into the same block. Placing similar BLKNMs on instances that do not fit within one LCA block creates an error.

Specifying identical BLKNM attributes on FMAP and/or HMAP symbols tells the software to group the associated function generators into a single CLB. Using BLKNM, you can partition a complete CLB without constraining the CLB to a physical location on the device.

For an XC4000 CLB, the maximum number of elements that can be assigned the same block name is two flip-flops, two FMAPs, and one HMAP. For an XC3000 CLB, the maximum number of elements that can be assigned the same block name is two flip-flops or one CLBMAP. For an XC2000 CLB, the maximum number is one flip-flop or one CLBMAP.

BLKNM attributes, like LOC constraints, are specified from the schematic. Hierarchical paths are not prefixed to BLKNM attributes, so BLKNM attributes for different CLBs must be unique throughout the entire design. See the section on the HBLKNM attribute for information on attaching hierarchy to block names.

Use the BLKNM attribute to attach a name to the following symbols:

- XC4000 flip-flop primitives (FDCE, FDPE)
- XC3000 flip-flop primitives (FDCE)
- XC2000 flip-flop and latch primitives (FDCP, LDCP)
- I/O buffers, flip-flops, and latches (IBUF, OBUF, OBUFT, ILD, IFD, OFD, OFDT)
- PAD primitives (PAD, IPAD, OPAD, BPAD, UPAD, PADU)
- I/O block primitives (IOB symbols)
- Configurable logic blocks (CLB symbols)
- 3-state buffers (BUFT symbols)
- Mapping control symbols (CLBMAP, FMAP, HMAP)

Syntax

The syntax of the BLKNM attribute is the following:

BLKNM=blockname

where blockname is a valid LCA block name for that type of symbol. For a list of prohibited block names, see the “Naming Conventions” section of each user interface manual.

For information on assigning hierarchical block names, see the HBLKNM attribute description in this chapter.

Example

Suppose that you want to map together two flip-flops within one CLB. You give both the BLKNM=FFGRP1 attribute. You then translate, place, and route the design. When you examine it in EditLCA, you see that both flip-flops reside within a CLB named FFGRP1.

CAP

Architectures

The CAP attribute applies to the XC4000H family only.

Description

You can specify an XC4000H output driver as operating in either resistive (RES) or capacitive, "softedge" (CAP) mode. In resistive mode, the output is faster and draws more power. Use this mode when the output is attached to purely resistive loads, or when ground bounce is not expected to be a problem with the output. The CAP attribute allows you to specify capacitive mode. Use capacitive mode when connecting an output to a capacitive mode, or when ground bounce is predicted to be a problem with the output. In capacitive mode, the pull-down transistor is slowly turned off as the output is pulled to ground, minimizing the likelihood of ground bounce.
See the section on the RES attribute for more information.
Use the CAP attribute on the following symbols:

- IOB output symbols OBUF, OBUFT
- IOB pads OPAD, IOPAD, UPAD
- Special function access symbols TDI, TMS, TCK

Syntax

The CAP attribute has the following syntax:

CAP

CLOCK_OPT

Architectures

The CLOCK_OPT attribute applies to the XC7200 and XC7300 families only.

Description

The CLOCK_OPT global attribute controls FastCLK optimization for the entire design. FastCLK optimization changes a product-term clock to a FastCLK global signal, which reduces the number of universal interconnect matrix (UIM) inputs and product terms required by each function block.

Syntax

Use the following syntax with the CLOCK_OPT attribute:

```
CLOCK_OPT={on|off}
```

The On setting enables FastCLK optimization; the Off setting inhibits it. On is the default.

CMOS

Architectures

The CMOS attribute applies to the XC4000H family only.

Description

The CMOS attribute configures output drivers on the XC4000H to drive to CMOS-compatible levels. Similarly, it configures IOBs to have CMOS-compatible input thresholds.

To configure output drive levels, attach the CMOS attribute to any of the following output symbols: OBUF, OBUFT, OUTFF/OFD, OUTFFT/OFDT.

To configure input threshold levels, attach the CMOS attribute to any of the following input symbols: IBUF, INFF/IFD, INLAT/ILD, INREG.

See the section on the TTL attribute for more information.

Syntax

The syntax of the CMOS attribute is the following:

CMOS

CONFIG

Architectures

The CONFIG attribute applies to XC2000 and XC3000 families only.

Description

The CONFIG attribute specifies logic element inputs and the storage element function for a CLB or IOB symbol.

CONFIG attributes can only be attached to IOB and CLB symbols.

Syntax

Use the following syntax for the CONFIG attribute:

```
CONFIG=tag:[value]:[value]
```

where *tag and value are derived from the following tables.*

XC2000 CLB Configuration Options

Tag	**BASE F**	**BASE FG**	**BASE FGM***
X	F, Q	F, G, Q	M, Q
Y	F, Q	F, G, Q	M, Q
Q	FF, LATCH	FF, LATCH	FF, LATCH
SET	A, F	A, F	A, M
RES	D, F	D, G	D, M
CLK	K, C, F, NOT	K, C, G, NOT	K, C, M, NOT
F	A, B, C, D, Q	A, B, C, D, Q	A, B, C, D, Q
G	None	A, B, C, D, Q	A, B, C, D, Q

*For BASE FGM, M=F if B=1, and M=G if B=0.

XC2000 IOB Configuration Options

Tag	**BASE IO**
I	PAD, Q
BUF	ON, TRI

XC3000 CLB Configuration Options

Tag	**BASE F**	**BASE FG**	**BASE FGM***
X	F, QX	F, QX	M, QX
Y	F, QY	G, QY	M, QY

DX	DI, F	DI, F, G	DI, M
DY	DI, F	DI, F, G	DI, M
CLK	K, NOT	K, NOT	K, NOT
RSTDIR	RD	RD	RD
ENCLK	EC	EC	EC
F	A,B,C,D,E,QX, QY	A,B,C,D,E,QX, QY	A,B,C,D,QX, QY
G	None	A,B,C,D,E,QX, QY	A,B,C,D,QX, QY

*For BASE FGM, M=F if E=0, and M=G if E=1.

XC3000 IOB Configuration Options

Tag	BASE IO
I N˘	I, IQ, IKNOT, FF, LATCH, PULLUP
OUT	O, OQ, NOT, OKNOT, FAST
TRI	T, NOT

Example

Following is an example of a valid XC2000 CLB CONFIG attribute value:

```
X:Q Y:G CLK:K:NOT Q:FF SET:A RES:D
```

Here is an example of a valid XC3000 CLB CONFIG attribute value:

```
X:QX Y:QY DX:F DY:G CLK:K ENCLK:EC
```

DECODE

Architectures

The DECODE attribute applies to the XC4000 family only.

Description

The DECODE attribute defines where a wired-AND (WAND) instance is placed, either within a BUFT or an edge decoder. If the DECODE attribute is placed on a single-input WAND1 gate, the gate is implemented as an input to a wide-edge decoder in an XC4000 design.

Syntax

The syntax of the DECODE attribute is the following:

DECODE

DECODE attributes can only be attached to a WAND1 symbol.

DOUBLE

Architectures

The DOUBLE attribute applies to the XC3000 family only.

Description

The DOUBLE attribute specifies double pull-up resistors on the horizontal longline. On XC3000 parts, there are internal nets that can be set as 3-state with two programmable pull-up resistors available per line. If the DOUBLE attribute is placed on a PULLUP symbol, both pull-ups are used, enabling a fast, high-power line. If the DOUBLE attribute is not placed on a pull-up, only one pull-up is used, resulting in a slower, lower-power line.
When the DOUBLE attribute is present, the speed of the distributed logic is increased, as is the power consumption of the part. When only half of the longline is used, there is only one pull-up at each end of the longline.
While the DOUBLE attribute can be used for the XC4000 family, it is not recommended. PPR activates both pull-up resistors if the entire longline (not a half-longline) is used.

Syntax

The syntax of the DOUBLE attribute is the following:

```
        DOUBLE
```

The DOUBLE attribute can only be attached to a BUFT symbol.

EQUATE_F and EQUATE_G

Architectures

The EQUATE_F and EQUATE_G attributes apply to the XC2000 and XC3000 families only.

Description

The EQUATE_F and EQUATE_G attributes set the logic equations describing the F and G function generators of a CLB, respectively.

Syntax

The syntax of the EQUATE_F and EQUATE_G attributes is the following:

EQUATE_F or EQUATE_G

The following table lists the Boolean operators used in the logic equations.

Valid XC2000 and XC3000 Boolean Operators

Symbol	Boolean Equivalent
~	NOT
*	AND
@	XOR
+	OR
( )	Group expression

Example

Here are two examples illustrating the use of the EQUATE_F attribute:

```
        EQUATE_F=F=((~A*B)+D))@Q
        F=A@B+(C*~D)
```

FAST

Architectures

The FAST attribute applies to XC3000, XC3000A/L, XC4000, and XC4000A families only.

Description

The FAST slew-rate attribute is attached to an output buffer, output flip-flop, or pad to increase the speed of an IOB output. It produces a faster output but may increase noise and power consumption.
The FAST attribute can be attached to the following symbols:

- IOB symbols OBUF, OBUFT, OFD, OFDI, OFDT, OFDTI, OPAD, IOPAD, UPAD
- Special function access symbols TDI, TMS, TCK

Syntax

The syntax of the FAST attribute is the following:

```
FAST
```

FILE

Architectures

The FILE attribute applies to all FPGA families.

Description

The FILE attribute is placed on symbols that do not have underlying schematics. It references the XNF file containing the Xilinx netlist for the logic represented by the symbol. When XNFMerge encounters such a symbol, it looks in the design directory for the XNF file and replaces the description of the symbol in the XNF file with the functionality found in the XNF file.

Syntax

Use the following syntax for the FILE attribute:

```
FILE=filename
```

where filename is the name of an XNF file without the .xnf extension.

Example

Suppose that a symbol is created, called new_and2, whose function mimics that of a 2-input AND gate. A Xilinx ABEL file describes the function of the new_and2 symbol and is translated to an XNF file called new_and2.xnf. A FILE attribute is placed on the symbol, and the attribute is given a value of new_and2. The top-level design containing the new_and2 symbol is translated to an XNF file, and the following lines are found within it:

```
SYM, I$2, NEW_AND2, FILE=NEW_AND2
PIN, I1, I, NET_IN1
      PIN, I2, I, NET_IN2
      PIN, O1, O, NET_OUT1
      END
```

The new_and2.xnf file contains the following lines:

```
SYM, I$1, AND2
PIN, 1, I, I1
      PIN, 2, I, I2
      PIN, O, O, O1
      END
```

The top-level file is then processed by XNFMerge, which reads new_and2.xnf and replaces the description of the symbol with the description of the functionality, resulting in the following lines in the top-level design:

```
SYM, I$2/I$1, AND2
PIN, 1, I, NET_IN1
      PIN, 2, I, NET_IN2
      PIN, O, O, NET_OUT1
      END
```

The functionality of the symbol is added to the top-level design, while the connectivity found in the top-level design is maintained.

FOE_OPT

Architectures

The FOE_OPT attribute applies to the XC7200 and XC7300 families only.

Description

The FOE_OPT global attribute controls the optimization of the fast output enable (FOE) for the entire design. FOE optimization generally applies only to BUFE, OBUFE, or 3-state PLD outputs driving an OBUF. FOE optimization changes a product-term 3-state signal to an FOE global control signal. Like FastCLK assignment, it reduces the number of UIM inputs and product terms required by each function block.

Syntax

Use the following syntax with the FOE_OPT attribute:

FOE_OPT={on|off}

Off inhibits FOE optimization of the entire design, and On, which is the default, activates it.

HBLKNM

Architectures

The HBLKNM attribute applies to all FPGA families.

Description

The HBLKNM attribute assigns hierarchical LCA block names to logic elements and controls grouping in a flattened hierarchical design. When elements on different levels of a hierarchical design carry the same block name and the design is flattened, XNFMerge prefixes a hierarchical path name to the HBLKNM value.
Like BLKNM, the HBLKNM attribute forces function generators and flip-flops into the same CLB. Symbols with the same HBLKNM attribute map into the same CLB, if possible. However, using HBLKNM instead of BLKNM has the advantage of adding hierarchy path names during translation, and therefore the same HBLKNM attribute and value can be used on elements within different instances of the same macro.
Use the HBLKNM attribute to attach a name to the following symbols:

- XC4000 flip-flop primitives (FDCE, FDOP)
- XC3000 flip-flop primitives (FDCE)
- XC2000 flip-flop and latch primitives (FDCP, LDCP)
- I/O buffers, flip-flops, and latches (IBUF, OBUF, OBUFT, ILD, IFD, OFD, OFDT)
- PAD primitives (PAD, IPAD, OPAD, BPAD, UPAD, PADU)
- I/O block primitives (IOB symbols)
- Configurable logic blocks (CLB symbols)
- 3-state buffers (BUFT symbols)
- Mapping control symbols (CLBMAP, FMAP, HMAP)

Syntax

The syntax of the HBLKNM attribute is the following:

```
HBLKNM=blockname
```

where blockname is a valid LCA block name for that type of symbol. For a list of prohibited block names, see the “Naming Conventions” section of each user interface manual.

Example

A schematic is created that contains a four-input function and a flip-flop. The logic function is mapped using an FMAP symbol. Both the FMAP and the flip-flop are given the attribute HBLKNM=GROUP1. A symbol is created to represent the schematic, and both are given the name of FUNC. Another schematic is then created, and four instances of FUNC are placed on it. Because hierarchy is taken into account when the design is translated, the software recognizes four distinct groups, as opposed to one large group called GROUP1, and each instance of FUNC is mapped into a separate CLB.

HU_SET

Architectures

The HU_SET constraint applies to the XC4000 and XC4000A/H families only.

Description

Like the H_SET constraint, the HU_SET constraint is defined by the design hierarchy. However, it also allows you to specify a set name. It is possible to have only one H_SET constraint within a given hierarchical element (macro) but by specifying set names, you can specify several HU_SET sets. XNFMerge hierarchically qualifies the name of the HU_SET as it flattens the design and attaches the hierarchical names as prefixes. The difference between an HU_SET constraint and an H_SET constraint is that an HU_SET has an explicit user-defined and hierarchically qualified name for the set, but an H_SET constraint has only an implicit hierarchically qualified name generated by the design-flattening program. An HU_SET set "starts" with the symbols that are assigned the HU_SET constraint, but an H_SET set "starts" with the instantiating macro one level above the symbols with the RLOC constraints.
For detailed information about this attribute, refer to the "Relative Location (RLOC) Constraints" section later in this chapter.

Syntax

To designate a design element as a member of a HU_SET set, apply the following syntax to a design element:

```
        HU_SET=name
```

where name is the identifier for the set; it must be unique among all the sets in the design.

INIT

Architectures

The INIT attribute applies to the XC4000 and XC4000A/H families only.

Description

The INIT attribute initializes ROMs.
On a ROM, the INIT attribute gives an initial value to the contents of the ROM. Either four or eight hexadecimal digits are required, depending on the width of the ROM.

Syntax

Use the following syntax to implement the INIT attribute:

```
        INIT=value
```

For ROMs, value can be four or eight hexadecimal digits, depending on whether the ROM is a 16- or 32-word-deep ROM, respectively.

LOC

Architectures

The LOC constraint applies to all families.

Description for FPGAs

For FPGAs, the LOC constraint defines where a symbol can be placed within an FPGA. It specifies the absolute placement of a design element on the FPGA die. It can be a single location, a range of locations, or a list of locations. The LOC constraint can only be specified from the schematic. However, statements in a constraints file can also be used to direct placement.
The LOC constraint can be used on the following elements:

- BUFTs
- Elements that map into a CLB: flip-flops, FMAPs, HMAPs, CLBMAPs,CLBs
- Elements that map into an IOB: pads, IBUFs, OBUFs, INFFs, OUTFFs, and so forth
- For XC4000 only, WANDs and clock buffers

If a LOC constraint is placed on a macro symbol, XNFMerge passes it down onto every symbol of the appropriate type underneath that macro. For example, if LOC=CLB_R3C7 is placed on a macro, that LOC constraint is passed to flip-flops and map symbols but not to BUFTs.
You can use the LOC constraint to assign a specific LCA location to the following symbols:

- All flip-flop and latch primitives
- Xilinx soft macros (only flip-flops are affected)
- User-created symbols (only flip-flops are affected)
- Input buffers, output buffers, or pad symbols
- Clock buffers (ACLK, GCLK, BUFGP, BUFGS)
- I/O block primitives (IOB symbols) — XC2000, XC3000, XC3000A/L, XC3100, and XC3100A only
- Configurable logic blocks (CLB symbols) — XC2000, XC3000, XC3000A/L, XC3100, and XC3100A only
- 3-state buffers (BUFT symbols) — XC3000, XC3000A/L, XC3100, XC3100, and XC4000 only
- XC3000 horizontal longline pull-up resistors (PULLUP symbols)
- XC4000 wide-edge decoders (WAND*n and DECODEn symbols)*
- Mapping control symbols (CLBMAP, FMAP, HMAP)

You can ignore LOC constraints in the design or in various parts of the design by using the Ignore_xnf_locs option in XNFPrep and PPR.
You can specify multiple LOC constraints for the same symbol by using a semicolon (;) to separate each LOC within the field. It specifies that the symbols be placed or prohibited from being placed in any of the locations specified. Also, you can specify an area in which to place a symbol or group of symbols.
The legal names are a function of the target LCA part type. However, to find the correct syntax for specifying a target location, you can load an empty part into the XACT Design Editor (XDE). Place the cursor on any block to display its location in the lower left corner of the screen. Do not include the pin name such as .I, .O, or .T as part of the location.
You can use the LOC constraint for logic that uses multiple CLBs, IOBs, soft macros, or other symbols. To do this, use the LOC attribute on a soft macro symbol, which passes the location information down to the logic on the lower level. However, the location restrictions are only applied to the flip-flops within the logic block or to mapping symbols or 3-state buffers in user-created macros.

Description for EPLDs

For EPLDs, use the LOC=pinname attribute on a PAD symbol to assign the signal to a specific pin. The PAD symbols are IPAD, OPAD, IOPAD, and UPAD.
Pin assignments are unconditional; that is, the software does not attempt to relocate a pin if it cannot achieve the specified assignment. You can apply the LOC constraint to as many PAD symbols in your design as you like. However, each pin assignment further constrains the software as it automatically allocates logic and I/O resources to internal nodes and I/O pins with no LOC constraints.
To save all resulting pin assignments so they are preserved the next time you modify and re-integrate the design, use the PinSave command in the XDM Translate menu. This command saves the pin assignments to a design_name.vmf file. You can override individual pin assignments saved in the VMF file by changing or adding LOC=pinname attributes in the schematic.
Note: Pin assignment using the LOC attribute is not supported for bus pad symbols such as OPAD8.

Syntax for FPGAs

The syntax for specifying single LOC constraints for FPGAs is the following:

`LOC=`*location*

where *location is a legal LCA location for the LCA part type.*

You can specify areas of CLBs or BUFTs using the LOC constraint. Specify the upper left and lower right corners of an area in which logic is to be placed. Use a colon (:) to separate the two boundaries.

`LOC=`*location:location*

Conversely, you can also prohibit the placement of logic into a particular CLB or IOB by using the following syntax. Single locations or an entire area can be prohibited.

```
LOC<>location
LOC<>location:location
```

LOC= and LOC<> constraints can be used on the same symbol. If multiple LOC= constraints are placed on a single symbol or group of symbols, such as a macro, they are interpreted by the software as “ORing” each of the constraints together. Multiple LOC<> constraints are interpreted as “ANDing” the constraints together. The convention for specifying multiple LOC constraints is to separate each of them with a semicolon (;). Examples are shown in the “Examples” section, following.

Syntax for EPLDs

For EPLDs, the LOC syntax is the following:

```
LOC=pinname
```

where the pin name is Pnn for PC packages; nn is a pin number. The pin name is rc (row number and column number) for PG packages. Examples are LOC=P24 and LOC=G2.

Examples

This section gives several examples of the LOC syntax for FPGAs.

Single LOC Constraints

Examples of the syntax for single LOC constraints are given in table shown below.

Single LOC Constraint Examples

Attribute	**Description**
`LOC=P12`	Place I/O at location P12.
`LOC=B`	Place decode logic or I/O on the bottom edge.
`LOC=TL`	Place decode logic or I/O on the top left edge, or global buffer in the top left corner.
`LOC=AA(XC2000 and XC3000 only)`	Place logic in CLB AA.
`LOC=TBUF.AC.2 (XC2000 and XC3000 only)`	Place BUFT in TBUF above and one column to the right of CLB AC.
`LOC=CLB_R3C5 (XC4000 only)`	Place logic in the CLB in row 3, column 5.
`LOC=CLB_R4C5.ffx (XC4000 only)`	Place CLB flip-flop in the X flip-flop of the CLB in row 4, column 5.
`LOC=CLB_R4C5.F(XC4000 only)`	Place CLB function generator in the F generator of CLB-R4C5.
`LOC=TBUF_R2C1.1(XC4000 only)`	Place BUFT in row 2, column 1, along the top.
`LOC=TBUF_R*C0(XC4000 only)`	Place BUFT in any row in column 0.

Area LOC Constraints

Examples of LOC constraints used to specify area are given in table shown below.

Area LOC Constraint Examples

Attribute	**Description**
LOC=AA:FF(XC2000 and XC3000 only)	Place CLB logic anywhere in the top left corner of the LCA bounded by row F and column F.
LOC=CLB_R1C1:CLB_R5C5(XC4000 only)	Place logic in the top left corner of the LCA in a 5 x 5 area bounded by row 5 and column 5.
LOC=TBUF_R1C1:TBUF_R2C8(XC4000 only)	Place BUFT anywhere in the area bounded by row 1, column 1 and row 2, column 8.

Prohibit LOC Constraints

Examples of the correct syntax for prohibiting locations are shown below.

Prohibit LOC Constraint Examples

Attribute	**Description**
LOC<>T	Do not place I/O or decoder on the top edge.
LOC<>A* (XC2000 and XC3000 only)	Do not place logic anywhere in the top row.
LOC<>CLB_R5C*.ffy (XC4000 only)	Do not place the CLB flip-flop in the Y flip-flop of any CLB in row 5.
LOC<>CLB_R1C1:CLB_R5C5(XC4000 only)	Do not place the logic in any CLB in the top left corner extending to row 5, column 5.
LOC<>TBUF_R*C0 (XC4000 only)	Do not place BUFT anywhere in column 0.

Multiple LOC Constraints

Examples of multiple LOC constraints are provided in the table shown below.

Multiple LOC Constraint Examples

Attribute	**Description**
LOC<>*A;LOC<>*D(XC2000 and XC3000 only)	Do not place flip-flop in first or fourth column of CLBs
LOC=T:LOC=L	Place I/O or decoder (XC4000) on the top or left edge.
LOC=CLB_R1C1:CLB_R5C5; LOC<>CLB_R5C5 (must be specified in one continuous line) (XC4000 only)	Place CLB logic in the top left corner of the LCA in a 5 x 5 area, but not in the CLB in row 5, column 5.

CLB Placement Examples

You can assign soft macros and flip-flops to a single CLB location, a list of CLB locations, or a rectangular block of CLB locations. You can also specify the exact function generator or flip-flop within a CLB. CLB locations are identified as CLB_R#C# for an XC4000, or nn for an XC2000 or XC3000, where nn is a two-letter designator. The upper left CLB is CLB_R1C1 or AA.

The following examples illustrate the format of CLB constraints. Enter LOC= or LOC<> and the pin or CLB location. If the target symbol represents a soft macro, the LOC constraint is applied to all appropriate symbols (flip-flops, maps) contained in that macro. If the indicated logic does not fit into the specified blocks, an error is generated.
The following statements place logic in the designated CLB.

```
LOC=AA          (XC2000 and XC3000)
LOC=CLB_R1C1    (XC4000)
```

The following statements prohibit the placement of logic in the designated CLB.

```
LOC<>AA         (XC2000 and XC3000)
LOC<>CLB_R1C1   (XC4000)
```

The following statements place logic within the first column of CLBs. The asterisk (*) is a wildcard character.

```
LOC=*A          (XC2000 and XC3000)
LOC=CLB_R*C1    (XC4000)
```

The next two statements place logic in any of the three designated CLBs. There is no significance to the order of the LOC statements.

```
LOC=AA;LOC=AB;LOC=AC      (XC2000 and XC3000)
LOC=CLB_R1C1;LOC=CLB_R1C2;LOC=CLB_R1C3 (XC4000)
```

The following statements place logic within the rectangular block defined by the first specified CLB in the upper left corner and the second specified CLB in the lower right corner.

```
LOC=AA:HE       (XC2000 and XC3000)
LOC=CLB_R1C1:CLB_R8C5     (XC4000)
```

The next statement places logic in the X flip-flop of CLB_R2C2. For the Y flip-flop, use the FFY tag.

```
LOC=CLB_R2C2.FFX   (XC4000)
```

IOB Placement Examples

You can assign I/O pads, buffers, and registers to an individual IOB location or to a specified die edge or half-edge. IOB locations are identified by the corresponding package pin designation or by the edge of the FPGA array.
The following examples illustrate the format of IOB constraints. Specify either LOC= or LOC<> and the pin location. If the target symbol represents a soft macro containing only I/O elements, for example, INFF8, the LOC constraint is applied to all I/O elements contained in that macro. If the indicated I/O elements do not fit into the specified locations, an error is generated.
The following statement places the I/O element in location P13. For PGA packages, the letter-number designation is used, for example, B3.

```
LOC=P13
```

The next statement places I/O elements in IOBs along the top edge of the die. For the other three die edges, use B (bottom), L (left), or R (right).

```
LOC=T
```

The following statement places I/O elements in IOBs along the top half of the left edge of the die. The first letter in this code represents the die edge, and the second letter represents the desired half of that edge. Other legal half-edge values are LB (left bottom), RT (right top), RB (right bottom), TL (top left), TR (top right), BL (bottom left), and BR (bottom right).

```
LOC=LT
```

The next statement prohibits the placement of I/O elements on the left edge of the die.

```
LOC<>L
```

Note: The edges referred to in these constraints are die edges, which do not necessarily correspond to package edges. View the device in EditLCA to determine which pins are on which die edge.

BUFT Placement Examples

You can assign internal 3-state buffers (BUFTs) to an individual BUFT location, a list of BUFT locations, or a rectangular block of BUFT locations. In XC4000, BUFT locations are identified by the adjacent CLB. Thus, TBUF_R1C1.1 is just above CLB_R1C1, and TBUF_R1C1.2 is just below it in an XC4000 part. In XC2000 and XC3000, BUFT locations are not as straightforward. View the device in EditLCA to determine the exact BUFT names.

BUFT constraints all refer to locations with a prefix of TBUF, which is the name of the physical element on the device.
The following examples illustrate the format of BUFT LOC constraints. Specify either LOC= or LOC<> and the BUFT location.
The following statements place the BUFT in the designated location.

```
LOC=TBUF.AA.1        (XC2000 and XC3000)
LOC=TBUF_R1C1.1      (XC4000)
```

The next statements place BUFTs at any location in the first column of BUFTs. The asterisk (*) is a wildcard character.

```
LOC=TBUF.*A  (XC2000 and XC3000)
LOC=TBUF_R*C0        (XC4000)
```

The following statements place BUFTs within the rectangular block defined by the first specified BUFT in the upper left corner and the second specified BUFT in the lower right corner.

```
LOC=TBUF.AA:TBUF.BH          (XC2000 and XC3000)
LOC=TBUF_R1C1:TBUF_R2C8      (XC4000)
```

The following statements prohibit the placement of BUFTs at any location in the first row of BUFTs.

```
LOC<>TBUF.A*         (XC2000 and XC3000)
LOC<>TBUF_R1C*       (XC4000)
```

Global Buffer Placement Examples (XC4000 Only)

You can assign global buffers (BUFGP and BUFGS) to one of the four corners of the die. Specify either LOC= or LOC<> and the global buffer location. The following example illustrates the format of global buffer constraints.

```
LOC=TL
```

This statement places the global buffer in the top left corner of the die. For the other three corners, use TR (top right), BL (bottom left), and BR (bottom right).
You cannot assign placement to the GCLK or ACLK buffers in the XC2000 and XC3000 families, since there is only one of each, and their placements are fixed on the die.

Decode Logic Placement Examples (XC4000 Only)

In an XC4000 design, you can assign the decode logic to a specified die edge or half-edge. All elements of a single decode function must lie along the same edge.
The following example illustrates the format of decode constraints. Specify either LOC= or LOC<> and the decode logic symbol location. If the target symbol represents a soft macro containing only decode logic, for example, DECODE8, the LOC constraint is applied to all decode logic contained in that macro. If the indicated decode logic does not fit into the specified decoders, an error is generated.

```
LOC=L
```

This statement places the decoder logic along the left edge of the die. For the other three edges, use T (top), B (bottom), or R (right).

LOGIC_OPT

Architectures

The LOGIC_OPT attribute applies to the XC7200 and XC7300 families only.

Description

The LOGIC_OPT global attribute controls the default logic optimization for the entire design.

Syntax

The syntax for this attribute is the following:

```
LOGIC_OPT={on|off}
```

To inhibit logic optimization for the whole design, set this attribute to Off. The default is On. You can override the global setting for individual symbols using the OPT=on or OPT=off component attribute.

LOWPWR

Architectures

The LOWPWR attribute applies to the XC7300 family only.

Description

You can use the LOWPWR attribute as either a global or component attribute. When used as a component attribute, it determines the power setting of the macrocells used by an individual symbol. When used as a global attribute, it makes low power the global default power setting.
This attribute is ignored if it is assigned to a symbol that uses no macrocells, such as an inverter, AND/OR gate (when optimized), input register, and so on.

Syntax

To make low power the setting of the macrocells used by an individual symbol, use the following syntax:

```
        LOWPWR={on|off}
```

To make low power the global default power setting, place the following syntax in the schematic:

```
        LOWPWR=ALL
```

The default is LOWPWR=off, indicating a high-speed power setting for all macrocells used in the design unless otherwise specified.

MAP

Architectures

The MAP attribute applies to all FPGA families.

Description

The MAP attribute is placed on an FMAP, HMAP, or CLBMAP to specify whether pin swapping and the merging of other functions with the logic in the map are allowed. If pin swapping is allowed, the net connections to the pins on the CLB may differ from the connections to the map symbol. If merging with other functions is allowed, other logic can also be placed within the CLB, if space allows.

Syntax

The syntax of the MAP attribute is the following:

```
        MAP={PLC|PUC|PLO|PUO}
```

where the keywords have the following meanings:

- PLC means that the CLB pins are locked, and the CLB is closed.
- PLO means that the CLB pins are locked, and the CLB is open.
- PUC means that the CLB pins are unlocked, and the CLB is closed.
- PUO means that the CLB pins are unlocked, and the CLB is open.

“Locked” in these definitions means that the software cannot swap signals among the pins on the CLB; “unlocked” indicates that it can. “Open” means that the software can add or remove logic from the CLB; conversely, “closed” indicates that the software cannot add or remove logic from the function specified by the MAP symbol.
The default is PUC.

Example

A two-input function is mapped using an FMAP. Upon reaching the place and route stage of the design, the software determines that additional logic could be merged into the function generator containing the first function. If the MAP attribute value is PLO or PUO, the logic is merged into the function generator. If the MAP attribute value is PLC or PUC, the logic is not merged into the function generator. The software also determines that routing can be improved if the first and second pins on the function generator containing the 2-input function are swapped. If the MAP attribute is PUC or PUO, the pins are swapped. If the MAP attribute value is PLC or PLO, the pins are not swapped.

MEDFAST and MEDSLOW

Architectures

The MEDFAST and MEDSLOW attributes apply to the XC4000A family only.

Description

MEDFAST and MEDSLOW specify the slew rate of an XC4000A output driver. MEDFAST decreases output transition time and is slightly faster than MEDSLOW, possibly resulting in more noise and power consumption that an output driver specified as MEDSLOW.
The MEDFAST and MEDSLOW attributes can be attached to the I/O symbols and the special function access symbols TDI, TMS, and TCK.

Syntax

The syntax of the MEDFAST and MEDSLOW attributes is the following:

```
MEDFAST
```

or **MEDSLOW**

MINIMIZE

Architectures

The MINIMIZE attribute applies to the XC7200 and XC7300 families only.

Description

The MINIMIZE global attribute determines whether or not the software minimizes the logic for the whole design. If the logic is minimized, any redundant or non-effective logic found in any user-specified equation files is eliminated through Boolean minimization.

Syntax

The syntax of the MINIMIZE attribute is the following:

```
MINIMIZE={on|off}
```

where On allows logic minimization, and Off inhibits it. The default is On.

MRINPUT

Architectures

The MRINPUT attribute applies to the XC7300 family only.

Description

The MRINPUT global attribute in an XC7354 or XC7336 design changes the master reset pin to an ordinary input pin. If this attribute is set to On, the EPLD device is initialized only on power-up.

Syntax

The syntax of the MRINPUT attribute is the following:

```
        MRINPUT={on|off}
```

The On setting changes the master reset pin to an ordinary input pin.
The default is Off.

Net

Architectures

Net attributes apply to all families except where noted in the following paragraphs.

Description

Attaching attributes to nets affects the mapping, placement, and/or routing of the LCA design. Net attributes can be any of the following values:

- C Critical (all FPGA families)
  The C net attribute flags a net as critical so the software tries to route the net earlier than others. See also W, the weight net attribute.
  Note: The use of the C (critical) and W (weight net) attributes is *not recommended. In many cases, their use can degrade rather than improve routability and performance.*
- F (XC7300 only)
  The F net attribute in an XC7300 device specifies that the macrocell implementing a component output should be placed in a fast function block (FFB). When placed on the output of an IBUF, the F attribute specifies that the input signal is to use the FastInput (FI) path when the signal is used in a fast function block.
  The F attribute is not valid on outputs of components that require features only present in high-density function blocks, such as PLFB9, ADD, ADSU, ACC, COMPM, LD, FDCP, FDCPE, XOR7, XOR8, and XOR9.
  Note: The BUFE symbol can be assigned to FFB only when driving an OBUF, and it must allow FOE optimization.
- G G Output (XC2000 and XC2000L only on flip-flop clock pins and latch enable pins)
  Any CLB clocks driven by this net are connected to the G function output.
- H (XC7300 only)
  The H net attribute in an XC7300 device specifies that the macrocell implementing a component output should be placed in a high-density function block.
  The H attribute is not valid on outputs of a PLFFB9 or any of the input/ output buffer symbols.
- I C Input (XC2000 and XC2000L only on flip-flop clock pins and latch enable pins)
  Any CLB clocks driven by this net are connected to the C input pin.
- K K Input (XC2000 and XC2000L only on flip-flop clock pins and latch enable pins)
  Any CLB clocks driven by this net are connected to the K input.
- L Longline (XC2000, XC3000, and XC3100 only)
  The APR router attempts to use a longline to route this net; a longline is useful for nets with high fan-out that need low skew.
- N Non-critical (all FPGA families)
  The N attribute flags a net as non-critical so the routing software gives this signal low priority. See also W, the weight net attribute.
  Note: The use of the N (non-critical) and W (weight net) attributes is not recommended. In many cases, their use can degrade rather than improve routability and performance.
- P Pin-lock (XC2000 and XC3000 only on CLBMAP primitives; XC4000 only on FMAPs and HMAPs)
  The P attribute specifies that the signal should not be moved from the CLB pin to which it is assigned. It is useful for aligning CLB inputs with a specified longline.
- S Save (all FPGA families)

The S attribute prevents the removal of unconnected signals, which is useful when using the map-then-merge method on lower-level hierarchy. If you do not have the S attribute on a net, any signal not connected to logic and/or an I/O primitive is removed.

- W Weight Net (all FPGA families)
  The W attribute indicates the routing order of the specified net by assigning it a net weight. For XC4000 and XC3000A/L (PPR) designs, legal values are 1-99, with 0 being equivalent to the N (non-critical) attribute and 100 being equivalent to the C (critical) attribute. For XC2000 and XC3000 devices (APR), a value of 0 or 1 means non-critical, 10 or higher means critical, and net weights of 2 through 9 are not graded.
  Note: The use of the C (critical) or N (non-critical) and W (weight net) attributes is not recommended. In many cases, their use can degrade rather than improve routability and performance.
- X Explicit or External (all FPGA families)
  With this attribute, XNFMAP or PPR ensures that a net is not mapped inside the combinational logic of a CLB, which would make the net "disappear." For example, an external net between a logic gate and a flip-flop forces the software to place the combinational logic and the flip-flop in different CLBs. This mapping may make the mapping of the design less efficient, but it guarantees that the flagged net exists at a CLB output, which allows the signal to be probed in XDE.

Syntax

Methods of entering this attribute vary by user interface. Consult the appropriate user interface guide for instructions.

NODELAY

Architectures

The NODELAY attribute applies to the XC4000 and XC4000A families only.

Description

The default configuration of IOB flip-flops in XC4000 and XC4000A designs includes an input delay that results in no external hold time on the input data path. However, this delay can be removed by placing the NODELAY attribute on input flip-flops or latches, resulting in a smaller setup time but a positive hold time. The NODELAY attribute can be attached to the I/O symbols and the special function access symbols TDI, TMS, and TCK.

Syntax

The syntax of the NODELAY attribute is the following:

```
NODELAY
```

OPT

Architectures

The OPT attribute applies to the XC7200 and XC7300 families only.

Description

The OPT attribute controls the optimization of all macrocells used by a symbol.
If you build combinational logic using low-level gates and multiplexers, the logic optimizer attempts to pack all logic bounded between device I/O pins and registers into a single macrocell.

The logic optimizer optimizes components forward into components connected to their outputs. It also moves forward any logic, whether combinational or sequential, that is buffered by a 3-state buffer. However, logic that itself contains a 3-state control is not moved forward.
The OPT=off attribute prevents any logic in a component from optimizing forward.
The OPT attribute has no effect on any symbol that contains no macrocell logic, such as an input/output buffer.

Syntax

The syntax of the OPT attribute is the following:

```
OPT={on|off}
```

OPT=on allows optimization of macrocell logic; OPT=off inhibits optimization. The default is the value of the LOGIC_OPT attribute, which is On unless otherwise specified.

PLD

Architectures

The PLD attribute applies to XC7200 and XC7300 families only.

Description

The PLD attribute is placed on a PLD symbol to specify the name of the file containing the logic equations for that PLD. Use it on custom primitive symbols and the following PLDs: PL20V8, PL22V10, PL20PIN, PL24PIN, PL48PIN, PLFB9, and PLFFB9.
All PLD components in your schematic design must be assigned the PLD attribute. Running XEMake automatically assembles all equation files named by all PLD=filename attributes found in the schematic. If you do not use XEMake, you must assemble each PLD file in the design using PLUSASM before you run the FITNET command.
Like PLDs, user-specified (custom) primitives are defined by PLUSASM equation files. The PLD=filename attribute is not required but can be applied as a convenient way to have your equation file automatically assembled when XEMake is invoked. If you omit the PLD attribute, FITNET will expect to find a bitmap file for the symbol (symbol_name.vmh) in your local CLIB subdirectory.

Syntax

Following is the syntax of the PLD attribute:

```
PLD=filename
```

Do not specify the filename extension. You must specify this filename as the first parameter of the CHIP statement inside the equation file, as described in the “PLUSASM Language Reference” section of the XEPLD Reference Guide. Here is an example:

```
CHIP filename PL22V10
```

PRELOAD_OPT

Architectures

The PRELOAD_OPT attribute applies to XC7200 and XC7300 families only.

Description

The PRELOAD_OPT global attribute allows the XEPLD software to change the preload values in the design to match the preload values supported by specified device resources such as fast function blocks and input registers. The XEPLD software can therefore map your design most efficiently, using the device resources most suited to the elements of your design. Unless you specify PRELOAD_OPT=off, the software is free to change the initial register states of any component, including PLD (custom) components defined in

PLUSASM. Use PRELOAD=off to preserve the initial states specified in this manual for library components and in the PRLD equations in your PLUSASM file for PLD or custom components.
You can set a high or low preload for high-density function blocks. The preload value of fast function blocks depends on the use of Set or Reset. Input register preload values are fixed at 1, except for those on the XC7272, which are undefined.

Syntax

The syntax of the PRELOAD_OPT attribute is the following:

```
        PRELOAD_OPT={on|off}
```

The On setting, which is the default, allows XEPLD to change the preload values; Off preserves all preload values defined in the library and specified in your PLD equation files.

REG_OPT

Architectures

The REG_OPT attribute applies to XC7200 and XC7300 families only.

Description

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

Syntax

Use the following the syntax with the REG_OPT option:

```
        REG_OPT={on|off}
```

To inhibit input register optimization, set this attribute to Off. To enable this optimization, set it to On, which is the default.

RES

Architectures

The RES attribute applies to the XC4000H family only.

Description

You can specify an XC4000H output driver as operating in either resistive (RES) or capacitive, "softedge" (CAP) mode. In resistive mode, the output is faster and draws more power. Use this mode when the output is attached to purely resistive loads, or when ground bounce is not predicted to be a problem with the output. The RES attribute allows you to specify resistive mode.
Use capacitive mode when connecting an output to a capacitive mode, or when ground bounce is predicted to be a problem with the output. In capacitive mode, the pull-down transistor is slowly turned off as the output is pulled to ground, minimizing the likelihood of ground bounce.
See the section on the CAP attribute for more information.
The RES attribute can be attached to the I/O symbols and the special function access symbols TDI, TMS, and TCK.

Syntax

The syntax of the RES attribute is the following:

```
        RES
```

RLOC

Architectures

The RLOC constraint applies to XC4000 and XC4000A/H families only.

Description

Relative location (RLOC) constraints group logic elements into discrete sets and allow you to define the location of any element within the set relative to other elements in the set, regardless of eventual placement in the overall design. See the "Relative Location (RLOC) Constraints" section later in this chapter for detailed information about this type of constraint.

Syntax

Use the following syntax with the RLOC constraint:

```
        RLOC=Rrow#Ccolumn#[.extension]
```

where the row and column numbers can be any positive integer, including *zero*.
The optional .extension can take all the values that are available with the current absolute LOC syntax: FFX, FFY, F, G, H, 1, and 2. The 1 and 2 values are available for BUFT primitives, and the rest are available for primitives associated with CLBs. Only extensions for the XC4000 family designs are currently supported.
The RLOC value cannot specify a range or a list of several locations; it must specify a single location.
See the "Relative Location (RLOC) Constraints" section later in this chapter for information on the RLOC syntax.

RLOC_ORIGIN

Architectures

The RLOC_ORIGIN constraint applies to XC4000 and XC4000A/H families only.

Description

An RLOC_ORIGIN constraint fixes the members of a set at exact die locations. This constraint must specify a single location, not a range or a list of several locations. For detailed information about this constraint, refer to the "Relative Location (RLOC) Constraints" section later in this chapter.
The RLOC_ORIGIN constraint is required for a set that includes BUFT symbols.

Syntax

The syntax of the RLOC_ORIGIN constraint is the following:

```
        RLOC_ORIGIN=Rrow#Ccolumn#
```

where the row and column numbers are positive non-zero integers.

RLOC_RANGE

Architectures

The RLOC_RANGE constraint applies to XC4000 and XC4000A/H families only.

Description

The RLOC_RANGE constraint is similar to the RLOC_ORIGIN constraint except that it limits the members of a set to a certain range on the die. The range or list of locations is meant to apply to all applicable elements with RLOCs, not just to the origin of the set.

Syntax

The RLOC_RANGE constraint has the following syntax:

```
        RLOC_RANGE=Rrow1#Ccol#:Rrow2#Ccol2#
```

where the row numbers and the column numbers can be non-zero positive numbers or the wildcard (*) character. This syntax allows three kinds of range specifications, which are defined in the RLOC_RANGE section of the “Relative Location (RLOC) Constraints” section later in this chapter.

TNM

Architectures

The TNM attribute applies to XC3000A/L, XC3100A, and XC4000 families only, and only when XACT-Performance is used.

Description

The TNM attribute tags specific flip-flops, RAMs, pads, and input latches as members of a group to simplify the application of timing specifications to the group.
See the “XACT-Performance Utility” chapter of the XACT Reference Guide for detailed information about this attribute.

Syntax

Following is the syntax of the TNM attribute:

```
        TNM=identifier
```

where identifier can be any combination of letters, numbers, or underscores.
Do not use reserved words, such as FFS, LATCHES, RAMS, or PADS for TNM identifiers.

TSidentifier

Architectures

The TSidentifier attribute applies to XC3000A/L, XC3100A, and XC4000 families only.

Description

TSidentifier properties beginning with the letters “TS” are placed on the TIMESPEC symbol. The value of the TSidentifier attribute corresponds to a specific timing specification that can then be applied to paths in the design.
See the “XACT-Performance Utility” chapter of the *XACT Reference Guide for detailed information about this attribute.*

Syntax

The syntax of the TSidentifier attribute is the following:

```
        TSidentifier
```

where identifier can be any combination of letters, numbers, or underscores. It is commonly 01, 02, 03, and so forth. In Mentor, it must be 01, 02, 03, and so forth.

TTL

Architectures

The TTL attribute applies to the XC4000H family only.

Description

The TTL attribute configures output drivers on the XC4000H to drive to TTL-compatible levels. Similarly, it configures IOBs to have TTL-compatible input thresholds.
To configure output drive levels, attach the TTL attribute to any of the following output symbols: OBUF, OBUFT, OUTFF/OFD, OUTFFT/OFDT.
To configure input threshold levels, attach the TTL attribute to any of the following input symbols: IBUF, INFF/IFD, INLAT/ILD, INREG.
See the section on the CMOS attribute for more information.

Syntax

The syntax of the TTL attribute is the following:

```
        TTL
```

UIM_OPT

Architectures

The UIM_OPT attribute applies to the XC7200 and XC7300 families only.

Description

UIM optimization extracts AND expressions and inverters out of macrocell logic functions and moves them into the UIM, which reduces the use of function block resources. The UIM_OPT global attribute turns this type of optimization on or off.

Syntax

The syntax of the UIM_OPT attribute is the following:

```
        UIM_OPT={on|off}
```

where On activates UIM optimization, and Off inhibits it. The On setting is the default.

USE_RLOC

Architectures

The USE_RLOC constraint applies to the XC4000 and XC4000A/H families only.

Description

The USE_RLOC constraint turns on or off the RLOC constraint for a specific element or section of a set. For detailed information about this constraint, refer to the “Relative Location (RLOC) Constraints” section later in this chapter.

Syntax

The syntax of the USE_RLOC constraint is the following:

```
        USE_RLOC={true|false}
```

where True turns on the RLOC attribute for a specific element, and False turns it off.

U_SET

Architectures

The U_SET constraint applies to the XC4000 and XC4000A/H families only.

Description

The U_SET constraint groups design elements with attached RLOC constraints that are distributed throughout the design hierarchy into a single set. The elements that are members of a U_SET can cross the design hierarchy; that is, you can arbitrarily select objects without regard to the design hierarchy and tag them as members of a U_SET. For detailed information about this attribute, refer to the "Relative Location (RLOC) Constraints" section later in this chapter.

Syntax

The syntax of the U_SET constraint is the following:

```
U_SET=name
```

where name is the identifier of the set. This name is absolute; you specify it, and it is not prefixed by a hierarchical qualifier.

Relative Location (RLOC) Constraints

This section describes the relative location (RLOC) constraint, RLOC sets, and RLOC set constraints and modifiers.

Description

Relative location constraints group logic elements into discrete sets. You can define the location of any element within the set relative to other elements in the set, regardless of eventual placement in the overall design. For example, if RLOC constraints are applied to a group of eight flip-flops organized in a column, PPR maintains the columnar order and moves the entire group of flip-flops as a single unit. In contrast, absolute location (LOC) constraints constrain design elements to specific locations on the FPGA die with no relation to other design elements.

RLOC constraints allow you to place logic blocks relative to each other to increase speed, use die resources efficiently, and take advantage of the special carry logic built into the control logic blocks (CLBs) of the XC4000 devices. They provide an order and structure to related design elements without requiring you to specify their absolute placement on the FPGA die. They allow you to replace any existing hard macro with an equivalent that can be directly simulated.

The relationally placed macro (RPM) library, which replaces the hard macro library, uses RLOC constraints to define the order and structure of the underlying design primitives. The RPM library offers the functionality and precision of the hard macro library with added flexibility. You can optimize RPMs and merge other logic within them. Because these macros are built upon standard schematic parts, they do not have to be translated before simulation.

In the Unified Libraries, you can use RLOC constraints with BUFT- and CLB-related primitives, that is, DFF, HMAP, FMAP, and CY4 primitives. You can also use them on non-primitive macro symbols. There are some restrictions on the use of RLOC constraints on BUFT symbols. See the section on the RLOC_ORIGIN attribute later in this chapter. However, you cannot use RLOC constraints with decoders, clocks, or I/O primitives. LOC constraints, on the other hand, can be used on all primitives: BUFTs, CLBs,IOBs, decoders, and clocks.

The libraries created before the release of the Unified Libraries do not include RLOC constraints on the primitive symbols below the macro symbols. To add RLOC constraints to the underlying macro primitives, make a copy of the library in your local directory and add the RLOC=R0C0 constraint to the underlying primitives. You can also attach RLOC constraints directly to non-macro primitives as you can for the Unified Libraries.

The following symbols (primitives) accept RLOCs:

FDCE
FDPE
FMAP
HMAP
RAM16X1
RAM32X1
ROM16X1

ROM32X1
BUFT

Syntax

The syntax of the RLOC constraint is the following:

```
    RLOC = Rrow#Ccolumn#[.extension]
```

where the optional .extension can take all the values that are available with the current absolute LOC syntax: FFX, FFY, F, G, H, 1, and 2. The 1 and 2 values are available for BUFT primitives, and the rest are available for primitives associated with CLBs. Only extensions for the XC4000 family designs are currently supported.

The row and column numbers can be any positive integer, including zero. Absolute die locations, in contrast, cannot have zero as a row or column number. Because row and column numbers in RLOC constraints define only the order and relationship between design elements and not their absolute die locations, their numbering can include zero. Even though you can use any positive integer in numbering rows and columns for RLOC constraints, it is recommended that you use small integers for clarity and ease of use.

It is not the absolute values of the row and column numbers that is important in RLOC specifications but their relative values or differences. For example, if design element A has an RLOC=R3C4 constraint and design element B has an RLOC=R6C7 constraint, the absolute values of the row numbers (3 and 6) are not important in themselves. However, the difference between them is important; in this case, 3 (6 -3) specifies that the location of design element B is three rows away from the location of design element A. To capture this information, a normalization process is used at some point in the design implementation. In the example just given, normalization would reduce the RLOC on design element A to R0C0, and the RLOC on design element B to R3C3.

In Xilinx programs, rows are numbered in increasing order from top to bottom, and columns are numbered in increasing order from left to right. RLOC constraints follow this numbering convention.

RLOC Sets

As noted previously, RLOC constraints give order and structure to related design elements. This section describes RLOC sets, which are groups of related design elements to which RLOC constraints have been applied. You can create multiple sets, but a design element can belong to one set only.

Sets can be defined in several ways: explicitly through the use of a set parameter or implicitly through the structure of the design hierarchy.

There are four distinct types of rules associated with each set:

- Definition rules define the requirements for membership in a set.
- Linkage rules specify how elements can be linked to other elements to form a single set.
- Modification rules dictate how to specify parameters that modify RLOC values of all the members of the set.
- Naming rules specify the nomenclature of sets.

These rules are discussed in the sections that follow.

The following sections discuss three different set constraints: U_SET, H_SET, and HU_SET. Elements must be tagged with both the RLOC constraint and one of these set constraints to belong to a set.

U_SET

U_SET constraints enable you to group into a single set design elements with attached RLOC constraints that are distributed throughout the design hierarchy. The letter U in the name U_SET indicates that the set is user-defined. U_SET constraints allow you to group elements, even though they are not directly related by the design hierarchy. By attaching a U_SET constraint to design elements, you can explicitly define the members of a set. The design elements tagged with a U_SET constraint can exist anywhere in the design hierarchy; they can be primitive or non-primitive symbols. When attached to non-primitive symbols, the U_SET constraint propagates to all the primitive symbols with RLOC constraints that are below it in the hierarchy.

The syntax of the U_SET constraint is the following:

```
    U_SET=name
```

where *name* is the user-specified identifier of the set. All design elements with RLOC constraints tagged with the same U_SET constraint name belong to the same set. Names therefore must be unique among all the sets in the design.

H_SET

In contrast to the U_SET constraint, which you explicitly define by tagging design elements, the H_SET (hierarchy set) is defined implicitly through the design hierarchy. The combination of the design hierarchy and the presence of RLOC constraints on elements defines a hierarchical set, or H_SET set. You do not use an HSET constraint to tag the design elements to indicate their set membership. The set is defined automatically by the design hierarchy. All design elements with RLOC constraints at a single node of the design hierarchy are considered to be in the same H_SET set unless they are tagged with another type of set constraint such as RLOC_ORIGIN or RLOC_RANGE. These constraints are discussed later in this chapter. If you explicitly tag any element with an RLOC_ORIGIN, RLOC_RANGE, U_SET, or HU_SET constraint, it is removed from an H_SET set. Most designs contain only H_SET constraints, since they are the underlying mechanism for relationally placed macros.

The design-flattening program, XNFMerge, recognizes the implicit H_SET set, derives its name, or identifier, attaches the H_SET constraint to the correct members of the set, and writes them to the output file. The syntax of the H_SET constraint as generated by XNFMerge follows:

```
H_SET=name
```

Name is the identifier of the set and is unique among all the sets in the design. The base name for any H_SET is “hset,” to which XNFMerge adds a hierarchy path prefix to obtain unique names for different H_SET sets in the XNFMerge output file.

The name of the H_SET set is derived from the symbol or node in the hierarchy that includes all the RLOC elements. Constraints that modify sets are discussed later in this chapter.

Set Modification

As noted earlier, the RLOC constraint assigns a primitive an RLOC value (the row and column numbers with the optional extensions), specifies its membership in a set, and links together elements at different levels of the hierarchy.

When the design is flattened, the row and column numbers of an RLOC constraint on an element are added to the row and column numbers of the RLOC constraints of the set members below it in the hierarchy. This feature gives you the ability to modify existing RLOC values in submodules and macros without changing the previously assigned RLOC values on the primitive symbols. This modification process also applies to the optional extension field. However, when using extensions for modifications, you must ensure that inconsistent extensions are not attached to the RLOC value of a design element that may conflict with RLOC extensions placed on underlying elements. For example, if an element has an RLOC constraint with the FFX extension, all the underlying elements with RLOC constraints must either have the same extension, in this case FFX, or no extension at all; any underlying element with an RLOC constraint and an extension different from FFX, such as FFY or F, is flagged as an error. After resolving all the RLOC constraints, extensions that are not valid on primitives are removed from those primitives. For example, if XNFMerge generates an FFX extension to be applied on a primitive after propagating the RLOC constraints, it applies the extension if and only if the primitive is a flip-flop. If the primitive is an element other than a flip-flop, the extension is ignored. Only the extension is ignored in this case, not the entire RLOC constraint.

The ability to modify RLOC values down the hierarchy is particularly valuable when instantiating the same macro more than once.

HU_SET

The HU_SET constraint is a variation of the implicit H_SET (hierarchy set). Like H_SET, HU_SET is defined by the design hierarchy. However, you can use the HU_SET constraint to assign a user-defined name to the HU_SET.

The syntax of the HU_SET constraint is the following:

```
HU_SET=name
```

where name is the identifier of the set; it must be unique among all the sets in the design. You must define the base names to ensure unique hierarchically qualified names for the sets after XNFMerge flattens the design and attaches the hierarchical names as prefixes.
This user-defined name is the base name of the HU_SET set. Like the H_SET set, in which the base name of "hset" is prefixed by the hierarchical name of the lowest common ancestor of the set elements, the user-defined base name of an HU_SET set is prefixed by the hierarchical name of the lowest common ancestor of the set elements.
The HU_SET constraint defines the start of a new set: all design elements at the same node that have the same user-defined value for the HU_SET constraint are members of the same HU_SET set. Along with the HU_SET constraint, elements can also have an RLOC constraint. The presence of an RLOC constraint in an H_SET constraint links the element to all elements tagged with RLOCs above and below in the hierarchy. However, in the case of an HU_SET constraint, the presence of an RLOC constraint along with the HU_SET constraint on a design element does not automatically link the element to other elements with RLOC constraints at the same hierarchy level or above.

Set Modifiers

A modifier, as its name suggests, modifies the RLOC constraints associated with design elements. Since it modifies the RLOC constraints of all the members of a set, it must be applied in a way that propagates it to all the members of the set easily and intuitively. For this reason, the RLOC modifiers of a set are placed at the start of that set. This section discusses the different modifiers that you can use to modify the RLOC set constraints.

RLOC

As discussed previously, the RLOC constraint associated with a design element modifies the values of other RLOC constraints below the element in the hierarchy of the set. Regardless of the set type, RLOC row, column, and extension values on an element always propagate down the hierarchy and are added at lower levels of the hierarchy to RLOC constraints on elements in the same set.

RLOC_ORIGIN

Specifying RLOC constraints to describe the spatial relationship of the set members to themselves allows the members of the set to float anywhere on the die as a unit. You can, however, fix the exact die location of the set members. The RLOC_ORIGIN constraint allows you to change the RLOC values into absolute LOC constraints that respect the structure of the set.
Following is the syntax of this constraint:

```
        RLOC_ORIGIN=Rrow#Ccolumn#
```

where the row and column numbers are positive non-zero integer values. When an RLOC_ORIGIN constraint is applied to a set, the row and column values of the RLOC_ORIGIN are added to the individual RLOC values of the members of the set to obtain a final LOC constraint for each element in the set. Since the row and column numbers of an RLOC_ORIGIN constraint refer to actual die locations, its value must exclude zero.
Note: In the XACT 5.0 release, you must use the RLOC_ORIGIN constraint with sets that include BUFT symbols. Sets with BUFT symbols must be fixed to an exact die location.
The design flattening program, XNFMerge, translates the RLOC_ORIGIN constraint into LOC constraints. The row and column values of the RLOC_ORIGIN are added individually to the members of the set after all RLOC modifications have been made to their row and column values by addition through the hierarchy. The final values are then turned into LOC constraints on individual primitives.
When this constraint is used in conjunction with an implicit H_SET (hierarchy set), it must be placed on the element that is the start of the H_SET set, that is, on the lowest common ancestor of all the members of the set. If you apply an RLOC_ORIGIN constraint to an HU_SET constraint, place it on the element at the start of the HU_SET set, that is, on an element with the HU_SET constraint. However, since there could be several elements linked together with the HU_SET constraint at the same node, the RLOC_ORIGIN constraint can be applied to only one of these elements to prevent more than one RLOC_ORIGIN constraint from being applied to the HU_SET set. Similarly, when used with a U_SET constraint, the RLOC_ORIGIN constraint can be placed on only one element with the U_SET constraint. If you attach the RLOC_ORIGIN constraint to an

element that has only an RLOC constraint, the membership of that element in any set is removed, and the element is considered the start of a new H_SET set with the specified RLOC_ORIGIN constraint attached to the newly created set.

RLOC_RANGE

As noted in the previous discussion, you can fix the members of a set at exact die locations with the RLOC_ORIGIN constraint. In the XACT 5.0 release, you must use the RLOC_ORIGIN constraint with sets that include BUFT symbols. However, for sets that do not include BUFT symbols, you can limit the members of a set to a certain range on the die. In this case, the set could “float” as a unit within the range until a final placement. Since every member of the set must fit within the range, it is important that you specify a range that defines an area large enough to respect the spatial structure of the set.
The syntax of this constraint is the following:

```
        RLOC_RANGE=Rrow1#Ccol1#:Rrow2#Ccol2#
```

where row1, row2, col1, and col2 can be non-zero positive numbers, or the wildcard (*) character. This syntax allows for three kinds of range specifications:

- Rr1Cc1:Rr2Cc2 — A rectangular region enclosed by rows r1, r2, and columns c1, c2
- R*Cc1:R*Cc2 — A region enclosed by the columns c1 and c2 (any row number)
- Rr1C*:Rr2C*— A region enclosed by the rows r1 and r2 (any column number)

For the second and third kinds of specifications with wildcards, applying the wildcard asterisk differently on either side of the separator colon creates an error. For example, specifying R*C1:R2C* is an error since the wildcard asterisk is applied to rows on one side and to columns on the other side of the separator colon.
The values of the RLOC_RANGE constraint are not simply added to the RLOC values of the elements. In fact, the RLOC_RANGE constraint does not change the values of the RLOC constraints on underlying elements. It is an additional constraint that is attached automatically by XNFMerge to every member of a set. The RLOC_RANGE constraint is attached to design elements in exactly the same way as the RLOC_ORIGIN constraint. The values of the RLOC_RANGE constraint, like RLOC_ORIGIN values, must be non-zero positive numbers since they directly correspond to die locations.

USE_RLOC

Another important set modifier is the USE_RLOC constraint. It turns the RLOC constraints on and off for a specific element or section of a set.
The syntax of this constraint is:

```
        USE_RLOC=value
```

where value is either True or False.
The application of the USE_RLOC constraint is strictly based on hierarchy. A USE_RLOC constraint attached to an element applies to all its underlying elements that are members of the same set. If it is attached to a symbol that defines the start of a set, the constraint is applied to all the underlying member elements, which represent the entire set. However, if it is applied to an element below the start of the set, only the members of the set below the specified element are affected. You can also attach the USE_RLOC constraint directly to a primitive symbol so that it affects only that symbol.
When the USE_RLOC=false constraint is applied, the RLOC and set constraints are removed from the affected symbols in the XNFMerge output file. This process is different than that followed for the RLOC_ORIGIN constraint. For RLOC_ORIGIN, XNFMerge generates and outputs a LOC constraint in addition to all the set and RLOC constraints in the output file. XNFMerge does not retain the original constraints in the presence of a USE_RLOC=false constraint because these cannot be turned on again in later programs.
Applying the USE_RLOC constraint on U_SET sets is a special case because of the lack of hierarchy in the U_SET set. Because the USE_RLOC constraint propagates strictly in a hierarchical manner, the members of a U_SET set that are in different parts of the design hierarchy must be tagged separately with USE_RLOC constraints; no single USE_RLOC constraint is propagated to all the members of the set that lie in different parts of the hierarchy. If you create a U_SET set through an instantiating macro, you can attach the USE_RLOC constraint to the instantiating macro to allow it to propagate hierarchically to all the members of the set. You can create this instantiating macro by placing a U_SET constraint on a macro and letting XNFMerge propagate that constraint to every symbol with an RLOC constraint below it in the hierarchy.

While propagating the USE_RLOC constraint, XNFMerge ignores underlying USE_RLOC constraints if it encounters elements higher in the hierarchy that already have USE_RLOC constraints. For example, if XNFMerge encounters an underlying element with a USE_RLOC=true constraint during the propagation of a USE_RLOC=false constraint, it ignores the newly encountered True constraint.

Xilinx Macros

Xilinx-supplied flip-flop macros include an RLOC_R0C0 constraint on the underlying primitive, which allows you to attach an RLOC to the macro symbol. This symbol links the underlying primitive to the set that contains the macro symbol. Simply attach an appropriate RLOC constraint to the instantiation of the actual Xilinx flip-flop macro. XNFMerge adds the RLOC value that you specified to the underlying primitive so that it has the desired value.
If you do not put an RLOC constraint on the flip-flop macro symbol, the underlying primitive symbol is the lone member of a set. XNFMerge removes RLOC constraints from a primitive that is the only member of a set or from a macro that has no RLOC objects below it.

LOC Propagation Through Design Flattening

XNFMerge continues to propagate LOC constraints down the design hierarchy. It adds this constraint to appropriate objects that are not members of a set. While RLOC constraint propagation is limited to sets, the LOC constraint is applied from its start point all the way down the hierarchy.

Summary

Table below summarizes the RLOC set types and the constraints that identify members of these sets.

Summary of Set Types

Type	**Definition**	**Naming**	**Linkage**	**Modification**
Set	A set is a collection of elements to which relative location constraints are applied.			
U_SET= name	All elements with the same user-tagged U_SET constraint value are members of the same U_SET set.	The name of the set is the same as the user-defined name without any hierarchical qualification.	U_SET links elements to all other elements with the same value for the U_SET constraint.	U_SET is modified by applying RLOC_ORIGIN or RLOC_RANGE constraints on, at most, one of the U_SET constraint-tagged elements.
H_SET (implicit through hierarchy) is not available as a constraint that you can attach to symbols.	RLOC on the node. Any other constraint removes a node from the H_SET set.	The lowest common ancestor of the members defines the start of the set. The name is the hierarchically qualified name of the start followed by the base name, "hset."	H_SET links elements to other elements at the same node that do not have other constraints. It links down to all elements that have RLOC constraints and no other constraints. Similarly, it links to other elements up the hierarchy that have RLOC constraints but no other constraints.	H_SET is modified by applying RLOC_ORIGIN and RLOC_RANGE at the start of the set: the lowest common ancestor of all the elements of the set.
HU_SET= name	All elements with the	The lowest common	HU_SET links to	The start of the set is

same hierarchically qualified name are members of the same set.	ancestor of the members is prefixed to the user-defined name to obtain the name of the set.	other elements at the same node with the same HU_SET constraint value. It links to elements with RLOC constraints below.	made up of the elements on the same node that are tagged with the same HU_SET constraint value. An RLOC_ORIGIN or an RLOC_RANGE can be applied to, at most, one of these start elements of an HU_SET set.

Relationally Placed Macros (RPMs)

The Xilinx libraries contain three types of elements.

- Primitives are basic logical elements such as AND2 and OR2 gates.
- Soft macros are schematics made by combining primitives and sometimes other soft macros.
- Relationally placed macros (RPMs) are soft macros that contain relative location constraint (RLOC) information, carry logic symbols, and FMAP/HMAP symbols, where appropriate. RPMs are currently only available in the XC4000 library.

Designs created with RPMs can be functionally simulated.

The HM2RPM utility translates old custom hard macro files into RPM files. If you created your own hard macro files, you must run HM2RPM on each hard macro file and place the new XNF file in your current working directory or in a search directory specified for XNFMerge. For instructions on using the HM2RPM utility, see the “HM2RPM” chapter of the XACT Reference Guide.

RPMs can, but need not, include all the following elements:

- FMAPs, HMAPs, and CLB-grouping attributes to control mapping. FMAPs and HMAPs have pin-lock attributes, which allow better control over routing. FMAPs and HMAPs are described in the “Mapping Constraints” section of the “PPR Placement Constraints” section earlier in this chapter.
- Relative location (RLOC) constraints to provide placement structure. They allow positioning of elements relative to each other. They are discussed in the “Relative Location Constraints” section earlier in this chapter.
- Carry logic primitive symbols. Carry logic is discussed in the next section, “Carry Logic in XC4000 LCAs.”

These elements allow you to access carry logic easily and to control mapping and block placement. Because RPMs are a super-set of ordinary macros, you can design them in the normal design entry environment. They can include any primitive logic. The macro logic is fully visible to you and can be easily back-annotated with timing information.

RPMs do not include routing capability. XACT-Performance specifications address timing issues more effectively.

Carry Logic in XC4000 LCAs

This section describes the use of carry logic in XC4000 CLBs and lists all the carry logic configuration mnemonics available.

The XC4000 CLB contains a feature called dedicated carry logic. This carry logic is independent of the function generators, although it shares some of the same input pins. Dedicated interconnect propagates carry signals through a column of CLBs. The carry logic in each CLB can implement approximately 40 different functions, which you can use to build faster and more efficient adders, subtracters, counters, comparators, and so forth.

Primitives and Symbols

The schematic capture libraries that Xilinx supports contain one generic carry logic primitive and several specific carry mode primitive symbols. The generic carry logic primitive represents the complete carry logic in a single CLB. The carry mode primitive symbols represent unique carry modes, such as ADD-FG-CI. To specify the particular mode that you wish, connect a carry mode symbol to the C0-C7 mode pins of the carry logic symbol. It is the pair of symbols that defines the specific kind of carry logic desired.
A carry logic symbol requires you to place either a LOC or an RLOC constraint on it. If a LOC constraint is used, it must be a single LOC= constraint; it cannot be an area or prohibit LOC constraint or use wildcards in its syntax.
Table below lists the carry mode names and symbols.

Carry Modes

Carry Mode Name	**Symbol**
ADD-F-CI	cy4_01
ADD-FG-CI	cy4_02
ADD-G-F1	cy4_03
ADD-G-CI	cy4_04
ADD-G-F3	cy4_05
ADDSUB-F-CI	cy4_12
ADDSUB-FG-CI	cy4_13
ADDSUB-G-F1	cy4_14
ADDSUB-G-CI	cy4_15
ADDSUB-G-F3	cy4_16
FORCE-0	cy4_37
FORCE-1	cy4_38
FORCE-F1	cy4_39
FORCE-CI	cy4_40
FORCE-F3	cy4_41
EXAMINE-CI	cy4_42
DEC-F-CI	cy4_24
DEC-FG-CI	cy4_25
DEC-FG-0	cy4_26
DEC-G-0	cy4_27
DEC-G-F1	cy4_28
DEC-G-CI	cy4_29
DEC-G-F3-	cy4_30
INC-F-CI	cy4_17
INC-FG-CI	cy4_18
INC-FG-1	cy4_19
INC-G-1	cy4_20
INC-G-F1	cy4_21
INC-G-CI	cy4_22
INC-G-F3-	cy4_23
SUB-F-CI	cy4_06
SUB-FG-CI	cy4_07
SUB-G-1	cy4_08
SUB-G-F1	cy4_10
SUB-G-CI	cy4_09
SUB-G-F3	cy4_11
INCDEC-F-CI	cy4_31
INCDEC-FG-CI	cy4_32
INCDEC-FG-1	cy4_33
INCDEC-G-0	cy4_34
INCDEC-G-F1	cy4_35

INCDEC-G-CI	cy4_36

cy4 and cy4_n are not supported by XC7000.

Carry Logic Handling in XNFPrep

The XNFPrep program checks for legal connections between carry logic symbols and also performs simple trimming on some carry modes. CY4 symbols might be trimmed as follows:

- If neither the COUT0 pin nor the COUT pin is used, the CY4 symbol is removed from the design. However, if the signal on the CIN pin connects to other logic, XNFPrep converts the CY4 to the EXAMINE-CI mode. An EXAMINE-CI mode CY4 is trimmed only if there is no other load on the signal on the CIN pin.
- If the COUT0 pin is used but the COUT pin is not, XNFPrep attempts to convert the CY4 symbol to use a 1-bit equivalent mode. That is, if the mode was originally of the form -FG-CI, it converts it to the equivalent -F-CI mode, allowing signals to be removed from the CY4 A1 and B1 operand inputs, which may save routing resources.
- If the specified mode does not require any of the A0, B0, A1, B1, and/or ADD CY4 inputs, XNFPrep removes the signals from these pins, which may save routing resources.

Carry Mode Configuration Mnemonics

The first step in configuring a CLB for carry logic is to choose the appropriate carry mode configuration mnemonic. Each of the 42 possible configurations of the carry logic has been assigned a three-part mnemonic code, for example:

```
ADD-FG-CI
```

- The first field (ADD) describes the operation performed in the CLB function generators, in this case, a binary addition. By implication, the carry logic in this CLB calculates the carry for this addition.
- The second field (FG) indicates which of the two function generators is used in the specified operation, in this case, both F and G.
- The last field (CI) specifies the source of the carry-in signal to the CLB, in this case, the CIN pin itself.

Consider another example:

```
INCDEC-G-F1
```

This mnemonic describes a CLB in which the G function generator performs an increment/decrement function. The carry-in to this CLB is sourced by the F1 pin.

All available carry mode configuration mnemonics are listed in the next section, “Carry Logic Configurations.”

Carry Logic Configurations

This section lists and describes all the available carry mode configuration mnemonics. The following information is given for each mnemonic:

- The name of the mode mnemonic
- A brief description of the CLB function
- The COUT0 and COUT1 equations performed by the carry logic
- Default equations for the F and G function generators
- Default assignments for the F4, G2, and G3 inputs

The default F and G functions and default F4, G2, and G3 inputs are based on the generic CLB function described. You can change these defaults as required, allowing for features such as parallel enable or synchronous reset. However, if these defaults are changed, the CLB may no longer function as the mnemonic describes.

The COUT0 and COUT1 equations are absolutely determined by the carry mode configuration mnemonic. The only way to change these carry logic outputs is by selecting a different mnemonic.

ADD-F-CI

The ADD-F-CI configuration performs a 1-bit addition of A+B in the F function generator, with the A and B inputs on the F1 and F2 pins. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. This configuration can be used as the MSB of an adder, with the G function generator accessing the carry-out signal or calculating a twos-complement overflow.
F=(F1@F2)@F4
COUT0=(F1*F2) + CIN*(F1+F2)
G=
COUT1=COUT0
F4=CIN
G2=G2I (COUT0 for overflow, OFL=G2@G3, or for carry-out, CO=G2)
G3=G3I (CIN for overflow, OFL=G2@G3)

ADD-FG-CI

The ADD-FG-CI configuration performs a 2-bit addition of A+B in both the F and G function generators, with the lower-order A and B inputs on the F1 and F2 pins, and the higher-order A and B inputs on the G1 and G4 pins. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. This configuration comprises the middle bits of an adder.
F=(F1@F2)@F4
COUT0=(F1*F2) + CIN*(F1+F2)
G=(G4@G1)@G2
COUT1=(G4*G1) + COUT0*(G4+G1)
F4=CIN
G2=COUT0
G3=G3I

ADD-G-F1

The ADD-G-F1 configuration performs a 1-bit addition of A+B in the G function generator, with the A and B inputs on the G1 and G4 pins. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of an adder, where the carry-in signal is routed to F1. The F function generator is not used.
F=
COUT0=F1
G=(G4@G1)@G2
COUT1=(G4*G1) + COUT0*(G4+G1)
F4=F4I
G2=COUT0
G3=G3I

ADD-G-CI

The ADD-G-CI configuration performs a 1-bit addition of A+B in the G function generator, with the A and B inputs on the G1 and G4 pins. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COUT pin. This configuration is for the middle bit of an adder, where the F function generator is reserved for another purpose.
F=
COUT0=CIN
G=(G4@G1)@G2
COUT1=(G4*G1) + COUT0*(G4+G1)
F4=F4I
G2=COUT0
G3=G3I

ADD-G-F3-

The ADD-G-F3- configuration performs a 1-bit addition of A+B in the G function generator, with the A and B inputs on the G1 and G4 pins. The carry signal enters on the F3 pin, is inverted by the F carry logic, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of an adder, where the inverted carry-in signal is routed to F3. The F function generator is not used.
F=
COUT0=~F3
G=(G4@G1)@G2
COUT1=(G4*G1) + COUT0*(G4+G1)
F4=F4I
G2=COUT0
G3=G3I

SUB-F-CI

The SUB-F-CI configuration performs a 1-bit twos-complement subtraction of A-B in the F function generator, with the A input on F1 and the B input on F2. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. This configuration can be used as the MSB of a subtracter, with the G function generator accessing the carry-out signal or calculating a twos-complement overflow.
F=(F1@F2)@~F4=~(F1@F2@F4)
COUT0=(F1*~F2) + CIN*(F1+~F2)
G=
COUT1=COUT0
F4=CIN
G2=G2I (COUT0 for overflow, OFL=G2@G3, or for carry-out, CO=G2)
G3=G3I (CIN for overflow, OFL=G2@G3)

SUB-FG-CI

The SUB-FG-CI configuration performs a 2-bit twos-complement subtraction of A-B in both the F and G function generators. For the lower bit, the A input is on F1 and the B input is on F2. For the upper bit, the A input is on G4 and the B input is on G1. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. This configuration comprises the middle bits of a subtracter.
F=(F1@F2)@~F4=~(F1@F2@F4)
COUT0=(F1*~F2) + CIN*(F1+~F2)
G=(G4@G1)@~G2=~(G4@G1@G2)
COUT1=(G4*~G1) +COUT0*(G4+~G1)
F4=CIN
G2=COUT0
G3=G3I

SUB-G-1

The SUB-G-1 configuration performs a 1-bit twos-complement subtraction of A-B in the G function generator, with the A input on G4 and the B input on G1. The carry-in is tied High (no borrow). The carry signal propagates through the G carry logic and exits on the COUT pin. This configuration comprises the LSB of a subtracter with no carry-in. The F function generator is not used.
F=
COUT0=1
G=(G4@G1)
COUT1=(G4+~G1)
F4=F4I
G2=G2I
G3=G3I

SUB-G-F1

The SUB-G-F1 configuration performs a 1-bit twos-complement subtraction of A-B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of a subtracter, where the carry-in signal is routed to F1. The F function generator is not used.

F=
COUT0=F1
G=(G4@G1)@~G2=~(G4@G1@G2)
COUT1=(G4*~G1) + COUT0*(G4+~G1)
F4=F4I
G2=COUT0
G3=G3I

SUB-G-CI

The SUB-G-CI configuration performs a 1-bit twos-complement subtraction of A-B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COUT pin. This configuration is for the middle bit of a subtracter, where the F function generator is reserved for another purpose.

F=
COUT0=CIN
G=(G4@G1)@~G2=~(G4@G1@G2)
COUT1=(G4*~G1) + COUT0*(G4+~G1)
F4=F4I
G2=COUT0
G3=G3I

SUB-G-F3-

The SUB-G-F3- configuration performs a 1-bit twos-complement subtraction of A-B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the F3 pin, is inverted by the F carry logic, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of a subtracter, where the inverted carry-in signal is routed to F3. The F function generator is not used.

F=
COUT0=~F3
G=(G4@G1)@~G2=~(G4@G1@G2)
COUT1=(G4*~G1) + COUT0*(G4+~G1)
F4=F4I
G2=COUT0
G3=G3I

ADDSUB-F-CI

The ADDSUB-F-C1 configuration performs a 1-bit twos-complement add/subtract of A+B in the F function generator, with the A input on F1 and the B input on F2. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. The F3 input indicates add (F3=1) or subtract (F3=0). This configuration can be used as the MSB of an adder/subtracter, with the G function generator accessing the carry-out signal or calculating a twos-complement overflow.

F=(F1@F2)@F4@~F3=~(F1@F2@F4@F3)
COUT0=F3*((F1*F2) + CIN*(F1+F2)) + ~F3*((F1*~F2) + CIN*(F1+~F2))
G=
COUT1=COUT0
F4=CIN
G2=G2I (COUT0 for overflow, OFL=G2@G3, or for carry-out, CO=G2)
G3=G3I (CIN for overflow, OFL=G2@G3)

ADDSUB-FG-CI

The ADDSUB-FG-CI configuration performs a 2-bit twos- complement add/subtract of A+B in both the F and G function generators. For the lower bit, the A input is on F1 and the B input is on F2. For the upper bit, the A input is on G4 and the B input is on G1. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. The F3 and G3 inputs indicate add (F3=G3=1) or subtract (F3=G3=0): the add/subtract control signal must be routed to both the F3 and G3 pins. This configuration comprises the middle bits of an adder/subtracter.
F=(F1@F2)@F4@~F3=~(F1@F2@F4@F3)
COUT0=F3*((F1*F2) + CIN*(F1+F2)) + ~F3*((F1*~F2) + CIN*(F1+~F2))
G=(G4@G1)@G2@~G3=~(G4@G1@G2@G3)
COUT1=F3*((G4*G1)+COUT0*(G4+G1))+~F3*((G4*~G1)+COUT0*(G4+~G1))
F4=CIN
G2=COUT0
G3=G3I

ADDSUB-G-F1

The ADDSUB-G-F1 configuration performs a 1-bit twos-complement add/subtract of A+B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COUT pin. The F3 and G3 inputs indicate add (F3=G3=1) or subtract (F3=G3=0): the add/ subtract control signal must be routed to both the F3 and G3 pins. This configuration comprises the LSB of an adder/subtracter, where the carry-in signal is routed to F1. The F function generator is not used.
F=
COUT0=F1
G=(G4@G1)@G2@~G3=~(G4@G1@G2@G3)
COUT1=F3*((G4*G1)+COUT0*(G4+G1))+~F3*((G4*~G1)+COUT0*(G4+~G1))
F4=F4I
G2=COUT0
G3=G3I

ADDSUB-G-CI

The ADDSUB-G-CI configuration performs a 1-bit twos-complement add/subtract of A+B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COUT pin. The F3 and G3 inputs indicate add (F3=G3=1) or subtract (F3=G3=0): the add/ subtract control signal must be routed to both the F3 and G3 pins. This configuration is for the middle bit of an adder/subtracter, where the F function generator is reserved for another purpose.
F=
COUT0=CIN
G=(G4@G1)@G2@~G3=~(G4@G1@G2@G3)
COUT1=F3*((G4*G1)+COUT0*(G4+G1))+~F3*((G4*~G1)+COUT0*(G4+~G1))
F4=F4I
G2=COUT0
G3=G3I

ADDSUB-G-F3-

The ADDSUB-G-F3 configuration performs a 1-bit twos-complement add/subtract of A+B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the F3 pin, is inverted by the F carry logic, propagates through the G carry logic, and exits on the COUT pin. Because the F3 input also indicates add (F3=1) or subtract (F3=0), the carry-in is always null (0 for add, 1 for subtract). This configuration comprises the LSB of an adder/subtracter with no carry-in. The F function generator is not used.
F=
COUT0=~F3

G=(G4@G1)
COUT1=F3*G4*G1 + ~F3(G4+~G1)
F4=F4I
G2=COUT0
G3=G3I

INC-F-CI

The INC-F-CI configuration performs a 1-bit increment in the F function generator, with the input on the F1 pin. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. The G function generator can be used to output the terminal count of a counter.
F=(F1@F4)
COUT0=CIN*F1
G=
COUT1=COUT0
F4=CIN
G2=G2I (COUT0 for terminal count, TC=G2)
G3=G31

INC-FG-CI

The INC-FG-CI configuration performs a 2-bit increment in both the F and G function generators, with the lower-order input on the F1 pin and the higher-order input on the G4 pin. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. This configuration comprises the middle bits of an incrementer.
F=(F1@F4)
COUT0=CIN*F1
G=(G4@G2)
COUT1=COUT0*G4
F4=CIN
G2=COUT0
G3=G3I

INC-G-1

The INC-G-1 configuration performs a 1-bit increment in the G function generator, with the input on the G4 pin. The carry-in is tied High. The carry signal propagates through the G carry logic and exits on the COUT pin. This configuration comprises the LSB of an incrementer that is always enabled. The F function generator is not used. This configuration is identical to DEC-G-0, since the LSB of an incrementer is identical to the LSB of a decrementer.
F=
COUT0=0
G=~(G4)
COUT1=G4
F4=F4I
G2=G2I
G3=G3I

INC-G-F1

The INC-G-F1 configuration performs a 1-bit increment in the G function generator, with the input on the G4 pin. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of an incrementer where F1 is an active-High enable. The F function generator is not used.
F=
COUT0=F1
G=(G4@G2)
COUT1=COUT0*G4

F4=F4I
G2=COUT0
G3=G3I

INC-G-CI

The INC-G-CI configuration performs a 1-bit increment in the G function generator, with the input on the G4 pin. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COUT pin. This configuration is for the middle bit of an incrementer where the F function generator is reserved for another purpose.
F=
COUT0=CIN
G=(G4@G2)
COUT1=COUT0*G4
F4=F4I
G2=COUT0
G3=G3I

INC-G-F3-

The INC-G-F3- configuration performs a 1-bit increment in the G function generator, with the input on the G4 pin. The carry signal enters on the F3 pin, is inverted in the F carry logic, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of an incrementer where F3 is an active-Low enable. The F function generator is not used.
F=
COUT0=~F3
G=(G4@G2)
COUT1=COUT0*G4=~F3*G4
F4=F4I
G2=COUT0
G3=G3I

INC-FG-1

The INC-FG-1 configuration performs a 2-bit increment in both the F and G function generator, with the lower-order A input on the F1 pin and the higher-order A input on the G4 pin. The carry-in is tied High. The carry signal propagates through the F and G carry logic and exits on the COUT pin. This configuration comprises the two least significant bits of an incrementer that is always enabled.
F=~(F1)
COUT0=F1
G=G2@G4
COUT1=COUT0*G4
F4=F4I or CIN
G2=COUT0
G3=G3I or CIN

DEC-F-CI

The DEC-F-CI configuration performs a 1-bit decrement in the F function generator, with the input on the F1 pin. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. The G function generator can be used to output the terminal count of a counter.
F=~(F1@F4)
COUT0=F1+CIN*~F1
G=
COUT1=COUT0
F4=CIN
G2=G2I (COUT0 for terminal count, TC=G2)
G3=G31

DEC-FG-CI

The DEC-FG-CI configuration performs a 2-bit decrement in both the F and G function generators, with the lower-order input on the F1 pin and the higher-order input on the G4 pin. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. This configuration comprises the middle bits of a decrementer.
F=~(F1@F4)
COUT0=F1+CIN*~F1
G=~(G4@G2)
COUT1=G4+COUT0*~G4
F4=CIN
G2=COUT0
G3=G3I

DEC-G-0

The DEC-G-0 configuration performs a 1-bit decrement in the G function generator, with the input on the G4 pin. The carry-in is tied High (no borrow). The carry signal propagates through the G carry logic and exits on the COUT pin. This configuration comprises the LSB of a decrementer that is always enabled. The F function generator is not used. This configuration is identical to INC-G-1, since the LSB of an incrementer is identical to the LSB of a decrementer.
F=
COUT0=0
G=~(G4)
COUT1=G4
F4=F4I
G2=G2I
G3=G3I

DEC-G-F1

The DEC-G-F1 configuration performs a 1-bit decrement in the G function generator, with the input on the G4 pin. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of a decrementer where F1 is an active-Low enable. The F function generator is not used.
F=
COUT0=F1
G=~(G4@G2)
COUT1=COUT0 + G4
F4=F4I
G2=COUT0
G3=G3I

DEC-G-CI

The DEC-G-CI configuration performs a 1-bit decrement in the G function generator, with the input on the G4 pin. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COUT pin. This configuration is for the middle bit of a decrementer, where the F function generator is reserved for another purpose.
F=
COUT0=CIN
G=~(G4@G2)
COUT1=G4+COUT0*~G4
F4=F4I
G2=COUT0
G3=G3I

DEC-G-F3-

The DEC-G-F3- configuration performs a 1-bit decrement in the G function generator, with the input on the G4 pin. The carry signal enters on the F3 pin, is inverted in the F carry logic, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of a decrementer, where F3 is an active-High enable. The F function generator is not used.

F=
COUT0=~F3
G=~(G4@G2)
COUT1=COUT0 + G4
F4=F4I
G2=COUT0
G3=G3I

DEC-FG-0

The DEC-FG-0 configuration performs a 2-bit decrement in both the F and G function generator, with the lower-order input on the F1 pin and the higher order input on the G4 pin. The carry-in is tied Low. The carry signal propagates through the F and G carry logic and exits on the COUT pin. This configuration comprises the two least significant bits of a decrementer that is always enabled.

F=~(F1)
COUT0=F1
G=~(G4@G2)
COUT=COUT1=(COUT0*~G4) + G4
F4=F4I
G2=COUT0
G3=G3I

INCDEC-F-CI

The INCDEC-F-CI configuration performs a 1-bit increment/decrement in the F function generator, with the input on the F1 pin. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. The F3 input indicates increment (F3=1) or decrement (F3=0). The G function generator can be used to output the terminal count of a counter.

F=(F1@F4)@~F3
COUT0=~F3*(F1+ CIN) + F3*F1*CIN
G=
COUT1=COUT0
F4=CIN
G2=G2I (COUT0 for terminal count, TC=G2)
G3=G31

INCDEC-FG-CI

The INCDEC-FG-CI configuration performs a 2-bit increment/decrement in both the F and G function generators, with the lower-order input on the F1 pin and the higher-order input on the G4 pin. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. The F3 and G3 inputs indicate increment (F3=G3=1) or decrement (F3=G3=0): the increment/decrement control signal must be routed to both the F3 and G3 pins. This configuration comprises the middle bits of an incrementer/decrementer.

F=(F1@F4)@~F3
COUT0=~F3*(F1+ CIN) + F3*F1*CIN
G=(G4@G2)@~G3
COUT1=~F3*(G4+ COUT0) + F3*G4*COUT0
F4=CIN
G2=COUT0
G3=G3I

INCDEC-G-0

The INCDEC-G-0 configuration performs a 1-bit increment/decrement in the G function generator, with the input on the G4 pin. The carry-in is tied High. The carry signal propagates through the G carry logic and exits on the COUT pin. This configuration comprises the LSB of an incrementer/decrementer that is always enabled. The F function generator is not used. F3 is not required for increment/decrement control, since the LSB of an incrementer is identical to the LSB of a decrementer; this configuration is identical to INC-G-1 and DEC-G-0.

F=
COUT0=0
G=~(G4)
COUT1=G4
F4=F4I
G2=G2I
G3=G3I

INCDEC-G-F1

The INCDEC-G-F1 configuration performs a 1-bit increment/decrement in the G function generator, with the input on the G4 pin. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of an incrementer/decrementer where the carry-in signal is routed to F1. The carry-in is active-High for an increment operation and active-Low for a decrement operation. The F function generator is not used. The F3 and G3 inputs indicate increment (F3=G3=1) or decrement (F3=G3=0): the increment/decrement control signal must be routed to both the F3 and G3 pins.

F=
COUT0=F1
G=(G4@G2)@~G3
COUT1=F3*(G4*COUT0) + ~F3*(G4+COUT0)
F4=F4I
G2=COUT0
G3=G3I

INCDEC-G-CI

The INCDEC-G-CI configuration performs a 1-bit increment/decrement in the G function generator, with the input on the G4 pin. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COUT pin. The F3 and G3 inputs indicate increment (F3=G3=1) or decrement (F3=G3=0): the increment/decrement control signal must be routed to both the F3 and G3 pins. This configuration is for the middle bit of an incrementer/decrementer, where the F function generator is reserved for another purpose, although the F3 pin is used by the carry logic.

F=
COUT0=CIN
G=(G4@G2)@~G3
COUT1=~F3*(G4+ COUT0) + F3*G4*COUT0
F4=F4I
G2=COUT0
G3=G3I

INCDEC-FG-1

The INCDEC-FG-1 configuration performs a 2-bit increment/decrement in both the F and G function generator, with the lower- order input on the F1 pin and the higher-order input on the G4 pin. The F3 and G3 inputs indicate increment (F3=G3=1) or decrement (F3=G3=0): the increment/decrement control signal must be routed to both the F3 and G3 pins. The carry-in is always active (High in increment mode and Low in decrement mode). The carry signal propagates through the F and G carry logic and exits on the COUT pin. This configuration comprises the two least significant bits of an incrementer/decrementer that is always enabled.

F=~(F1)

COUT0=F1
G=(G2@G4)@~G3
COUT=COUT1=~F3*((COUT0*~G4)+G4) + F3*(G4*COUT0)
F4=F4I
G2=COUT0
G3=G3I

FORCE-0

The FORCE-0 configuration forces the carry-out signal on the COUT pin to be 0.
COUT0=0
COUT1=0

FORCE-1

The FORCE-1 configuration forces the carry-out signal on the COUT pin to be 1.
COUT0=1
COUT1=1

FORCE-F1

The FORCE-F1 configuration forces the signal on the F1 pin to pass through to the COUT pin.
COUT0=F1
COUT1=COUT0=F1

FORCE-CI

The FORCE-CI configuration forces the signal on the CIN pin to pass through to the COUT pin.
COUT0=CIN
COUT1=COUT0=CIN

FORCE-F3-

The FORCE-F3- configuration forces the signal on the F3 pin to pass inverted to the COUT pin.
COUT0=~F3
COUT1=COUT0=~F3

EXAMINE-CI

The EXAMINE-CI configuration allows the carry signal on the CIN pin to be used in the F or G function generators. This configuration forces the signal on the CIN pin to pass through to the COUT pin and is equivalent to the FORCE-CI configuration. EXAMINE-CI is provided for CLBs in which the carry logic is unused but the CIN signal is required.
COUT0=CIN
COUT1=COUT0=CIN

List of ACTIVE-CAD File Types

File	Description
ABL	ABEL source file
ALB	Binary netlist file generated from the schematic
ALR	Binary netlist file generated from netlist import (e.g. routed BAX file)
ALX	Binary netlist file imported from X-BLOX functional simulation
ASC	ASCII Test Vector file
ASX	Port description file for HDL macro
ASF	State Machine drawing file
BAX	XNF file created by renaming XNFBA.XNF file for timing simulation
BSC	Automatically saved schematic backup file

BPR	Pin assignment back-annotation file for schematic editor. Created during post layout XNF import.
BRI	Bus definition file generated by XNF import
CMD	Viewsim compatible simulation script
CTL	Control file for XEMAKE 6 programs
DES	Simulation state file. Includes test vectors and simulation model states.
ENT	Entity/Architecture definitions for VHDL macro
ER	Formatted Error list from synthesis program or HDL analyzer
FRM	Simulation formula file
INI	Configuration files for applications
LCA	Xilinx device configuration file created by XACT software
LOG	Log files with messages
OPT	Synthesis option file for each macro. Defines compiler and options
PAR	Parameters for synthesis program. Includes command line and list of files to compile.
PDF	Project description file. Stores information about the project.
PRJ	Document modification LOG file. Each save records a change in this file.
PRO	Profile file containing Xilinx file option for batch processing
SCH	Schematic file
TVE	Binary test vector file
VHD	VHDL source file
XAS	XABEL generated file
XFF	Merged XNF files for entire project. Created by XNFMERGE program
XNF	Xilinx netlist file. Created from schematic or synthesis program.
XNR	XNF file created from routed LCA file when routed in batch mode (XMAKE)
XSF	Port definition file generated by synthesis program for each generated XNF file

Hot Keys

Project Manager:

Ctrl+N	New Project
Ctrl+O	Open Project
Ctrl+C	Copy Project
Ctrl+D	Delete Project
Ctrl+L	List Libraries
Ctrl+T	Project Type
Del	Remove Document
Enter	Open Document
Ctrl+I	Document Info
+	Expand One Level
*	Expand Branch
Ctrl+*	Expand All
-	Collapse Branch

Schematic Editor:

Ctrl+N	New Sheet
Ctrl+O	Open
Ctrl+S	Scratchpad
Ctrl+P	Print
Ctrl+T	Printer Setup
Ctrl+B	Table Setup
Alt+X	Exit
Ctrl+A	Undo
Ctrl+Z	Redo

Ctrl+X	Cut
Ctrl+C	Copy
Ctrl+V	Paste
Del	Delete
Ctrl+U	Deselect All
Ctrl+W	Redraw Wires
F2	Select and Drag
F3	Symbols
F4	Draw Wires
F5	Draw Buses
F6	Draw Bus Taps
F7	Query
F8	Test Points
Shift+F2	Create Netlist
Ctrl+F2	Integrity Test
Ctrl+K	Annotate
Ctrl+E	Symbol Editor
Ctrl+H	Hierarchy Push
Ctrl+I	Hierarchy Pop
Ctrl+G	Snap to Grid
Ctrl++	Zoom In
Ctrl+-	Zoom Out
PgDn	Full Page
PgUp	Previous Zoom
F9	Center
F10	Redraw
Shift+F4	Tile
Shift+F5	Cascade
Ctrl+L	Rotate Symbol
Ctrl+M	Mirror Symbol
Shift+F1	Help Contents

Symbol Editor:

Ctrl+N	New
Ctrl+O	Open...
Ctrl+S	Save
Ctrl+T	Test Symbol
Ctrl+P	Print...
Alt+Bksp	Undo
Ctrl+X	Cut
Ctrl+C	Copy
Ctrl+V	Paste
Del	Delete
Ctrl++	Zoom In
Ctrl+-	Zoom Out
F10	Redraw
Shift+F4	Tile
Shift+F5	Cascade

HDL Editor:

Ctrl+N	New
Ctrl+O	Open...
Ctrl+S	Save

Ctrl+P	Print...
Ctrl+Z	Undo
Ctrl+A	Redo
Ctrl+X	Cut
Ctrl+C	Copy
Ctrl+V	Paste
Del	Delete
Alt+F3	Find...
F3	Find Next
Shift+F4	Previous Error
F4	Next Error
Ctrl+G	Go to...
Ctrl+F2	Toggle Bookmark
Shift+F2	Previous Bookmark
F2	Next Bookmark
Ctrl+D	Delete to the End of Word
Ctrl+Y	Cut Line

State Editor:

Ctrl+N	New
Ctrl+O	Open...
Ctrl+S	Save
Ctrl+P	Print...
Ctrl+Z	Undo
Ctrl+A	Redo
Ctrl+X	Cut
Ctrl+C	Copy
Ctrl+V	Paste
Del	Delete
Alt+F3	Find String
F3	Find Next
Shift+F4	Previous Error
F4	Next Error
Ctrl++	Zoom In
Ctrl+-	Zoom Out
PgDn	Page
PgUp	Previous
F10	Redraw
Ctrl+H	HDL Code Generation

Simulator:

F2	Cascade
F3	Tile
F8	Short Step
F9	Long Step

BTI.INI File Settings

Btrieve for Windows v6.15

BTI.INI File Settings

This document describes the settings in the BTI.INI file for Btrieve. It does not include all of the settings used by the Btrieve utilities. Each utility maintains its own set of parameters within the program.

The BTI.INI file must reside in the local Windows directory. All parameters apply only to applications running on that one computer. Many of the parameters are settable using the Btrieve Technologies, Inc. Database Setup (DBSETUP) utility. You may use any standard text file editor to edit the parameters in the BTI.INI. Be sure to save the file as an ASCII text file.

Btrieve for Windows Settings

The BTI.INI contains three sections for entering Btrieve parameters, each of which is described below. The three sections are:

Btrieve Engine [Btrieve] Section
Btrieve Client [Btrieve Client] Section
Btrieve Requester [BrequestDPMI] Section

NOTE: The meaning of the /T parameter in *options* parameter in the Btrieve Client section has changed from the previous Btrieve releases. The new /T parameter sets the maximum number of Btrieve clients that can simultaneously have active transactions at the workstation. The old /T parameter specified the name of the directory where Btrieve placed transaction files. To specify this directory name now, use "trnfile=<dir-name>". See the discussion in the Btrieve Client section below.

Btrieve Engine [Btrieve] Section

The Btrieve Engine Section contains those parameters which are required by the Btrieve engine (WBTR32.EXE) and the Loader and Requester interface (WBTRCALL.DLL) to set up internal data structures and values. The section will normally be written by the DBSETUP utility by obtaining values for various parameters from the user.

```
[Btrieve]
tasks=30
local=Yes
requester=No
verbose=0
chkparms=No
```

tasks Entry

The **tasks** entry specifies the maximum (total) number of applications using either the local engine or the requester. This entry is directly settable by using the DBSETUP utility.

Range : 1..64,000 tasks
Default : 30

local Entry

The **local** entry specifies whether the local Btrieve engine should be loaded. This engine runs on the workstation of a user, not a NetWare file server. This entry is directly settable by using the DBSETUP utility, activated by selecting the CLIENT Engine Usage button.

Range : [Yes | No]
Default : Yes

requester Entry

The **requester** entry specifies whether the Btrieve requester interface will allow user access to a Btrieve engine running on a NetWare file server. This entry is directly settable by using the DBSETUP utility, activated by selecting the REQUESTER Engine Usage button. If you select Yes for this entry, you must also load the Btrieve for DOS requester (BREQUEST.EXE) before loading Windows.

Range : [Yes | No]
Default : No

verbose Entry

The **verbose** entry determines if errors encountered during the loading of the Btrieve client engine, or Btrieve requester will be displayed immediately, or detected when an application attempts to execute a Btrieve operation. When you set this entry to 1, the Btrieve icon will appear when your application accesses Btrieve. The possible settings and actions are as follows:

0	Detects an error on an attempt to execute Btrieve operation.
1	Displays an error message immediately.

This entry is not settable by the DBSETUP utility.

Range : 0..1
Default : 0

chkparms Entry

The **chkparms** entry determines if the input parameters for a Btrieve call will be checked (validated) by the Btrieve engine every time an engine access occurs. This parameter should be used for debugging only, as it will cause slight performance degradation on each call. This entry is not settable by the DBSETUP utility.

Range : [Yes | No]
Default : No

Btrieve Client [Btrieve Client] Section

The Btrieve Client Section contains those parameters required by the engine (WBTR32.EXE) in order to set up internal data structures and values. The section will normally be written by the Database Setup Utility (DBSETUP) by obtaining values for various parameters from the user.

```
[Btrieve Client]
options=/f:20 /h:60 /l:20 /t:15 /m:512 /u:0
trnfile=
desktopicon=No
SharingOnLocalFiles=SingleEngine
SharingOnRemoteFiles=MultiEngine
EnableSharingBias=Yes
echoargs=No
trace=No
tracefile=
traceops=
datalist=32
keylist=32
```

options Entry

The **options** entry details the load parameters for the local Btrieve engine. The entry is a sequence of load parameters of the form **/<option>:<value>**. All the default options are settable using the DBSETUP utility.

Range : Not Applicable (N/A)
Default : /f:20 /h:60 /l:20 /t:15 /m:512 /u:0

Other options may also be set. These options, /a, /b, /e, /g, /i, /o, and /q are described in the following paragraphs.

/a option
The /a option controls whether Btrieve keeps a log of all operations executed on the specified files. Btrieve v6.15 introduces an enhanced log file format. However, for compatibility with existing Btrieve v5.x clients, Btrieve for Windows v6.15 can also write the pre-v6.x log format. If a log file already exists, the existing format of the log file overrides any parameter setting. For any log files that do not exist, you can specify the desired format by setting /a:5 or /a:6 on the options entry. If /a:5 is selected, Btrieve creates a log file in the format specified by the Btrieve file version. If /a:6 is selected and no log file exists, all log files are created in the v6.x format.

/b option
The /b option allows you to specify a buffer size that is allocated for temporary use by three different features: extended get/step, get chunk, and create index operations. The buffer size is specified in KB. The valid range for this value is 0 to 64,000 KB. The default value is 16KB. For extended get/step operations, the buffer size must be large enough to hold the largest record in uncompressed format, plus the filter expression and the field extraction information. For the get chunk operations, the buffer size must be large enough for the chunk descriptors. For the create index operation, the buffer is only used for autoincrement indexes, and any size (except 0) is sufficient. For example, to allocate a 4 KB buffer, you would specify /b:4.

/e option
This option enables transaction durability when your Btrieve application issues an End Transaction operation. When this option is set, Btrieve will ensure that all updates within a single transaction are completed before returning the status code. A status code of 0 implies that the transaction was successfully written to disk.

/g option
The /g option can be configured to specify transaction bundling for system transactions. The format is as follows:

```
/g:<bundle limit>:<system transaction time limit>
```

bundle limit refers to the number of explicit Btrieve transactions or implicit transactions that are combined or bundled into one system transaction. An explicit transaction involves the use of the Begin and End Transaction operations; an implicit transaction is a single insert, update, or delete operation.

system transaction time limit refers to the number of milliseconds that elapses while Btrieve collects operations into a system transaction.

When either the bundle limit or the system transaction time limit is reached (whichever happens first), Btrieve initiates the writing of the system transaction to the operating system.

The default setting is /g:100:1000 meaning that Btrieve will bundle 100 transactions **or** wait one second (1000 milliseconds) before it initiates a write to the operating system. In changing these settings, the following guidelines should be observed:

If running in SEFS mode, your performance will be better if you set both values high.

If running in MEFS mode, you should set both values low to increase concurrency between Btrieve engines running on the network.

There is a third factor that can trigger Btrieve to write a system transaction to the operating system; that is, if the ratio of bundled pages and all memory cache pages exceeds the maximum amount of cache available. The higher the values you choose for bundle limit and system transaction limit, the more cache you will need in order to fully take advantage of these settings. You may increase the cache size by specifying a larger value for the /m option.

/i option
The /i option specifies a drive letter other than the drive where your Btrieve files are located for storing pre-image files for pre v6.x files. For v6.x formatted files, this drive letter specifies the location of the file sharing lock files. The file sharing feature of Btrieve for Windows v6.15 creates a lock file to control access to the Btrieve files from different workstations. The lock file is a temporary file with the same name as the Btrieve file that is opened, but with a .LCK extension. One lock file is created for each Btrieve file that is updated when file sharing is enabled.

By default, this lock file resides in the same directory as the actual Btrieve file that it is locking. To redirect the pre-image and lock files to drive F:, for example, you would specify /i:F. The specified drive must contain a directory with the same name at each level as the directory containing your Btrieve file.

IMPORTANT: If you are using multi-engine file sharing (MEFS), described below, the lock file drive must be the same drive for all Btrieve engines and this drive must be one that is accessible to all of the Btrieve engines on the network.

/o option
The /o option specifies the way Btrieve responds to DOS critical errors, such as "Drive Not Ready". This option does not require an argument. If you specify the /o option and a DOS critical error occurs during a Btrieve operation, Btrieve returns a status code, usually Status 2 (I/O Error) or 12 (File not found), to your application. If you do not specify the /o option, Btrieve routes the error to the standard DOS critical error handler, which usually results in a DOS error message being written to the screen. You can then either abort or retry the operation in response to the error.

/q option
The /q option is used in conjunction with the Btrieve create index operation. This option allows you to set an upper bound on the amount of memory (in KB) that Btrieve will allocate for sort/merge buffers. If an upper bound value is not set, Btrieve will attempt to allocate as much memory as it needs for sort/merge operations to complete the create index operation. This could cause conflicts for other applications that may also be attempting to allocate memory. This is especially true if your application is using the callback function, and other applications may run while an index is being created on a Btrieve file.

trnfile Entry

The **trnfile** entry identifies the location (path) of the transaction file used by the local Btrieve engine to manage transactions for applications on that workstation. If transactions will not be used on a particular workstation, then this entry may be left blank (NULL). This entry is settable using the DBSETUP utility. Each workstation should define a unique location for this file. If Btrieve terminates abnormally during a transaction, then upon reactivation, Btrieve uses this file to determine which incomplete transactions should be rolled back.

Range : Any valid and existing directory path.
Default : NULL(use default path, the Windows installation directory)

desktopicon Entry

The **desktopicon** entry informs the local Btrieve engine to display the icon representing the engine upon the Windows desktop. The icon can then be accessed by a user. This entry is not settable using the DBSETUP utility. If you set **verbose=1** in the Btrieve section, the Btrieve icon will appear when Btrieve is accessed, regardless of the current desktopicon setting.

Range : [Yes | No]
Default : No

SharingOnLocalFiles Entry

The **SharingOnLocalFiles** entry is utilized by the client Btrieve engine to determine if a Btrieve file located on client disk drives (floppy and hard) may be accessed by other Btrieve engines located on different workstations. The **SingleEngine** value indicates only Btrieve clients on this workstation may access local Btrieve files (SEFS). The **MultiEngine** value allows Btrieve clients on other workstations in a peer-to-peer network to access local Btrieve files (MEFS). See Chapter 2 of the *Btrieve for Windows Installation and Operations* manual for more information on SEFS and MEFS concepts.

Range : [SingleEngine|MultiEngine]
Default : SingleEngine

SharingOnRemoteFiles Entry

The **SharingOnRemoteFiles** entry is utilized by the client Btrieve engine to determine if a Btrieve file located on a file server or on a remote drive in a peer to peer network may be accessed by single or multiple Btrieve engines. The **SingleEngine** value indicates the Btrieve files cannot be accessed by other Btrieve engines, that is, only Btrieve users on this workstation can access remote Btrieve files in a sharing mode (SEFS). The **MultiEngine** value allows Btrieve users on other workstations to simultaneously access remote Btrieve files that are in use by this engine (MEFS). See Chapter 2 of the *Btrieve for Windows Installation and Operations* manual for more information on SEFS and MEFS concepts.

Range : [SingleEngine|MultiEngine]
Default : MultiEngine

EnableSharingBias Entry

The **EnableSharingBias** entry is utilized by the client Btrieve engine to determine if an application is allowed to override the file sharing modes defined in the BTI.INI file. When this parameter is set to Yes, the application may specify a bias on the Btrieve OPEN call to override the current configuration setting.

Range : [Yes | No]
Default : Yes

echoargs Entry

The **echoargs** entry causes the Btrieve client engine to display information before and after each API call. This information is written to Btrieve's window. The last 30K bytes of output is kept and can be scrolled. Also, the information can be saved as plain text using the File|Save menu options. The desktopicon entry must be set to Yes when this option is enabled. This parameter should be used for debugging only.

Range : [Yes | No]
Default : No

tracefile Entry

The **tracefile** entry enables Btrieve to write trace information for each API call. The information is written to the file specified. This parameter should be used for debugging only, as it will cause the performance of Btrieve to degrade. The information is written to the file using the 'forced write' mode to ensure that the data is written in the event that Btrieve ends or terminates abnormally while running.

Range : Any valid file path
Default : None

trace Entry

The **trace** entry is used to enable or disable the trace capability. If set to yes, then the**tracefile** entry should specify a valid file name. To disable the trace, you simply set trace=No.

Range : [Yes | No]
Default : No

traceops Entry

The **traceops** entry allows you to specify the set of Btrieve operations that you wish to trace. By default, Btrieve will write trace information for all Btrieve operations. You can specify the list of operations separated by blanks or commas, for example:

```
traceops=0,1,2,3
```

You can also specify that you want to trace operations that have a lock bias applied. For example, if you wanted to trace only the Get Equal with single record wait locks, you would enter 105.

Range : N/A
Default: All Btrieve operations

datalist Entry

The **datalist** entry allows you change the size of the data buffer that is written when the trace is enabled. The default value is 32 bytes.

Range : 0..64000
Default : 32

keylist Entry

The **keylist** entry allows you change the size of the key buffer that is written when the trace is enabled. The default value is 32 bytes.

Range : 0..255
Default : 32

Btrieve Requester [BrequestDPMI] Section

The Btrieve Requester Section will contain those parameters needed by
the Loader and Requester interface (WBTRCALL.DLL) in order to communicate with a Novell NetWare Server and transfer data between the workstation and the server. Currently, this section is not created or maintained by any existing utility, this task must be performed by the user.

```
[BrequestDPMI]
freememory=No
```

freememory Entry

The **freememory** entry instructs the requester to allocate and free real-mode memory upon each request. For any application executing strictly
in a Windows environment, the setting should be No. For those applications running in conventional memory, a setting of Yes may be appropriate, however, performance will suffer due to the overhead of allocation and free operations. This entry is not settable using the DBSETUP utility.

Range : [Yes | No]
Default : No