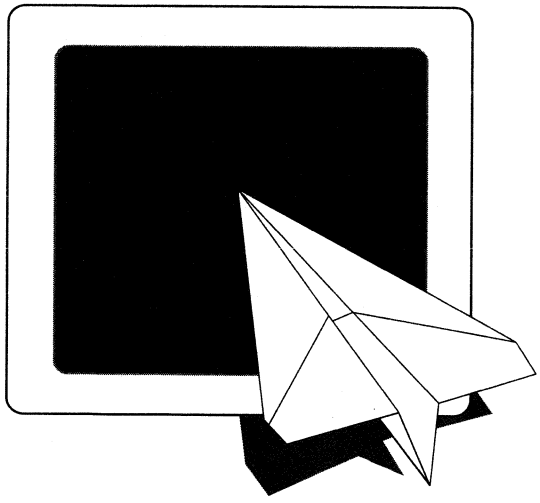
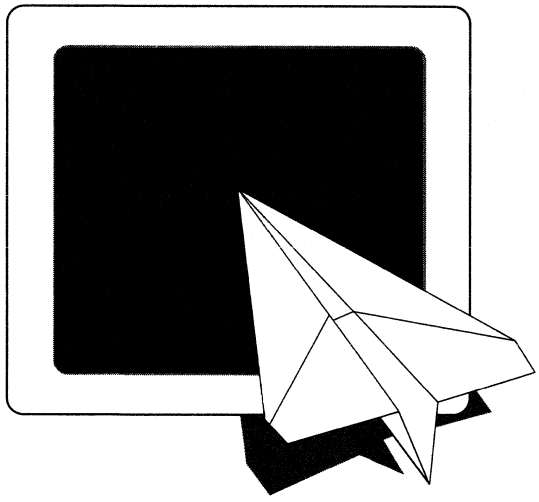


XACT Reference Guide, Volume 1

XACT Reference Guide, Volume 1 — April, 1994 (0401129 01)

Printed in U.S.A.

Table of Contents



Overview

XACT Reference Guide, Volume 1

XACT Reference Guide, Volume 1 — April, 1994 (0401129 01)

Printed in U.S.A.

XACT Reference Guide Overview

The *XACT Reference Guide* contains information on the software programs and hardware peripherals found in the XACT Development System. This guide is arranged into three volumes: Volume 1, "Design Entry and Conversion"; Volume 2, "Design Implementation"; and Volume 3, "Design Verification." Generally, the chapters in these volumes are organized in the following manner.

- A brief summary of program functions
- A syntax statement
- A review of the input files used and the output files generated by the program
- A listing of the commands, options or parameters used by the program
- Examples of how you can use the program
- Warning and error messages provided by the program with suggested recovery procedures for the error messages when appropriate

Typographical Conventions

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

`Courier font regular`	System messages or program files appear in regular Courier font.
`Courier font bold`	Literal commands that you must enter in syntax statements are in bold Courier font.
italic font	Variables that you replace in syntax statements are in italic font.
`[ ]`	Square brackets denote optional items or parameters. However, in bus specifications, such as bus [7:0], they are required.
`{ }`	Braces enclose a list of items from which you must choose one or more.
. . .	A vertical ellipsis indicates material that has been omitted.

...	A horizontal ellipsis indicates that the preceding can be repeated one or more times.
|	A vertical bar separates items in a list of choices.
↵	This symbol denotes a carriage return.

The Design Flow

Figure 1-1 shows the three parts of the LCA™ design flow: design entry, design implementation, and design verification.

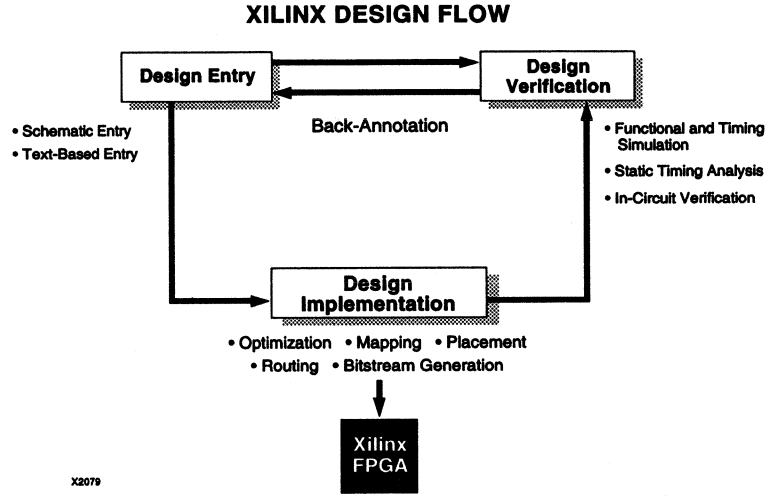


Figure 1-1 The XACT Development System Design Flow

Design entry is the process that takes the design from concept to netlist. There are a number of ways to enter a design, including schematics, Boolean or state expressions, and hardware description languages (HDLs). These entry methods require third-party tools that produce a design file in its own netlist format.

The process of design implementation converts the third-party netlist to Xilinx Netlist Format (XNF). It ultimately produces a configuration bitstream for the target LCA device. It includes optimization and mapping, placement and routing, and bitstream creation. Designs can be implemented automatically or by using a combination of the automatic and manual Xilinx Development System tools.

Design verification includes simulation, static-timing analysis, and in-circuit verification. Simulation is performed on third-party tools supported by Xilinx. The input for these tools requires a tool-specific translation of an XNF file. You can simulate an XNF file at any point or convert to an XNF file. Static-timing analysis tools and in-circuit verification tools are part of the Xilinx Development System. Consult the Design Verification chapter in the *XACT User Guide* for a more detailed description.

In some cases schematic simulation can be performed from the tool that created the design. For these tools, it is possible to simulate a design without creating an XNF file. As a result, simulator input can be another output of the design-entry tools.

Functional Sections of the *XACT Reference Guide*

The *XACT Reference Guide* provides detailed information about converting, implementing, and verifying designs in the XACT environment. Check the program chapters for information on what program works with each family of LCA device. The following is a brief overview of the contents and organization of the *XACT Reference Guide*.

Volume 1 — Design Entry and Conversion

Overview

The Overview provides generic descriptions of reference guide chapter structure, the typographical conventions used, a design flow summary, and descriptions of the programs in the XACT Development System.

XACT Design Manager (XDM)

XDM is the user interface shared by all XACT development programs.

XMake

XMake automatically converts a schematic design file into LCA and BIT files.

MemGen

MemGen creates RAMs or ROMs of any size for use in XC4000 designs.

XACT-Performance Utility

XACT-Performance™ specifies timing requirements on a schematic for any family (XC3000A/L, XC3100A, XC4000A/H) that PPR uses.

XNFCVT

XNFCVT converts an XNF netlist from Version 5 to Version 4, 2, or 1.

HM2RPM

HM2PRM converts hard macros to relationally placed macros (RPMs).

Volume 2 — Design Implementation

Overview

The Overview provides generic descriptions of reference guide chapter structure, the typographical conventions used, a design flow summary, and descriptions of the programs in the XACT Development System.

XNFMerge

XNFMerge merges XNF and MAP files together to form a single XNF file.

XNFPrep

XNFPrep performs a design rule check (DRC) and removes unused and redundant logic from a flattened XNF file. It also checks the syntax of the XACT-Performance parameters found in the design and prepares delay information for PPR path analysis.

XNFMAP

XNFMAP maps the logic defined by an XNF file into LCA elements such as CLBs, IOBs, and TBUFs.

MAP2LCA

MAP2LCA translates a MAP file into an LCA file for input into APR.

APR (Automatic Place and Route)

APR places and routes XC2000, XC2000L, XC3000, and XC3100 designs.

PPR (Partition, Place and Route)

PPR partitions, places, and routes XC3000A/L, XC3100A, XC4000, and XC4000A/H designs.

MakeBits

MakeBits creates a configuration bitstream for the LCA design.

MakePROM

MakePROM converts a configuration bitstream (BIT) file into a file that can be downloaded to a PROM. MakePROM also combines multiple BIT files for use in a daisy chain of LCA devices.

Volume 3 — Design Verification

Overview

The Overview provides generic descriptions of reference guide chapter structure, the typographical conventions used, a design flow summary, and descriptions of the programs in the XACT Development System.

XDelay

XDelay is a static-timing analyzer that reports detailed timing information about the design and can be used for overall performance analysis.

LCA2XNF

LCA2XNF converts an LCA file to a Xilinx Netlist Format (XNF) file for use in timing simulation.

XNFBA

XNFBA creates an XNF file containing delay information and using the original schematic net names.

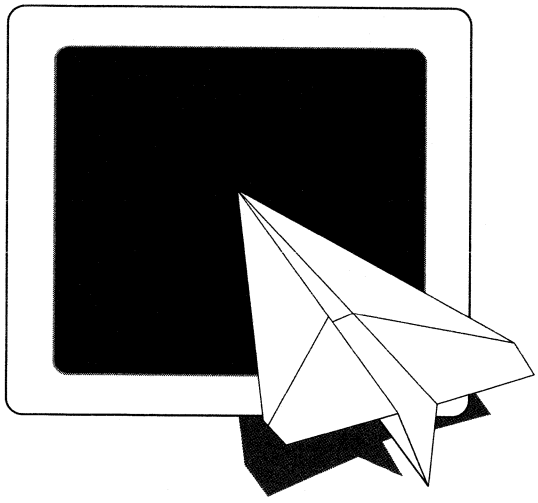
XACT Design Editor (XDE)

XDE allows the designer to manually edit or analyze an LCA design. XDE can also invoke MakeBits and MakePROM.

Other Xilinx Manuals

Other Xilinx manuals that you can refer to for more information are the following:

- *XACT User Guide*
- *XACT Reference Guide*, 3-Volume Set
- *XACT Libraries Guide*
- *Xilinx Hardware & Peripherals Guide*
- *XEPLD Design Guide*
- *XEPLD Reference Guide*
- *Xilinx-Synopsys Interface User Guide*
- *Xilinx ABEL Interface User Guide*
- *Viewlogic Interface User Guide*
- *OrCAD Interface User Guide*
- *Mentor V8 Interface User Guide*
- *The Programmable Logic Data Book*



The XACT Design Manager

XACT Reference Guide, Volume 1

XACT Reference Guide, Volume 1 — April, 1994 (0401129 01)

Printed in U.S.A.

The XACT Design Manager

This Program is Compatible with the Families Indicated.

☑ XC2000	☑ XC3100	☑ XC3100A	☑ XC4000H
☑ XC2000L	☑ XC3000A	☑ XC4000	☑ XC7200
☑ XC3000	☑ XC3000L	☑ XC4000A	☑ XC7300

XDM, the XACT Design Manager, is the common user interface shared by all FPGA and EPLD development packages. It serves as a menu-driven shell for executing all XACT development operations, including the XEPLD Integrator and related interfaces.

To use XDM effectively, you should be familiar with its features. The following subsections describe important XDM features.

- "Online Help" describes how to use the online help facility.
- "The Program List File" describes the proglist.xdm file.
- "Running XMake in XDM" describes how to run the XMake program from within XDM.
- "Using XDM on the PC" describes how to run XDM on the PC and explains the XDM opening screen.
- "Using XDM on a Workstation" describes how to run XDM on a workstation and explains the XDM opening screen.
- "XDM User Interface" describes how to execute commands from the graphical user interface and the command line.
- "The XDM Menu" briefly describes the menus and associated commands. For more details, refer to the appropriate chapter in the *XACT Reference Guide* for detailed descriptions of syntax, options, and error messages for specific programs.

Online Help

XDM includes online help for each menu, program, and command option. For example, you can display help information about the Translate menu, about the XMake program located in the Translate menu, or about the XMake –x option.

There are two methods for displaying help: XDM command-line entry and menu selection for any topic represented in the Design Manager menu structure. To use online help from the command line, use the following format:

help *topic–option*

Using this method, the *topic* can be either a menu name, a program, or command name. You can request information about a specific option by typing a "–" and the option immediately after the topic. Press ↵ to display the online help information. Press F1 to exit the online help display.

On PCs, selecting help from a menu requires that you first highlight the topic with the mouse. With the menu item highlighted, press the F1 key to access the corresponding help screen. Press F1 again to exit the help display.

NOTE *Pressing the F1 key on a highlighted command is called context-sensitive help. While command line entry for help is supported on all platforms, only PCs support context-sensitive help.*

NOTE *To exit from the online help on workstations, press F1.*

The Program List File

XDM maintains a program list file (proglist.xdm) to minimize the time required for start up. The text file contains a list of XDM-supported programs (executable files) installed on your system.

When generating proglist.xdm, XDM attempts to write it in the following order:

1. To the directory specified by the XACTUSER environment variable, if defined.
2. To the "data" subdirectory of the directory specified by the XACT environment variable, for example, C:\XACT\DATA on PCs.
3. To the current working directory.

When reading proglist.xdm, XDM searches for the file in the following order.

1. In the current working directory.
2. In the directory specified by the XACTUSER environment variable, if defined.
3. In the "data" subdirectory of the directory specified by the XACT environment variable, for example, C:\XACT\DATA on PCs.

XACTUSER is a user-defined environment variable that XDM uses only if it is defined.

Using XDM on a PC

The XDM executable file is located in the XACT directory (the default is C:\XACT), which should be included in the PATH specified in your autoexec.bat file. To start XDM at your operating system prompt, enter this command:

xdm

When you enter this command, the Design Manager software loads into memory, reads the proglist.xdm file to set up your menus, and displays a message similar to the one below.

```
Reading c:\xact\data\proglist.xdm...
```

NOTE *If proglist.xdm is missing, as in the very first invocation of XDM, XDM automatically generates this file using the ScanDisk command. See the XDM Menu section for a description of the ScanDisk command.*

After a short while, another message similar to the one below displays.

```
Reading c:\xact\data\xdm.pro...
```

This message means XDM is reading the profile information in the xdm.pro file to configure the Design Manager software as well as the graphics, I/O ports, and any other configurable system parameters. You can customize the xdm.pro file. Refer to the information in the Profile menu section.

When XDM is completely loaded, your monitor displays the XDM main screen. The main screen consists of a menu bar at the top, a command line at the bottom, a status/settings area above the command line, a mouse cursor, and a logo with software version information in the center, as shown in Figure 1-2.

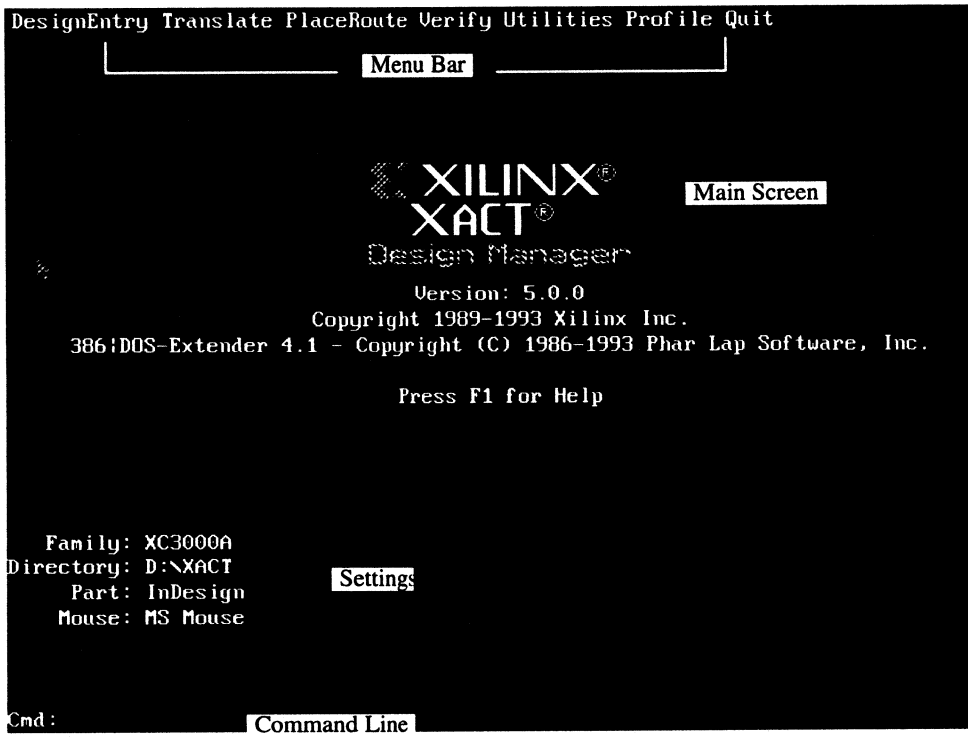


Figure 1-2 XACT Design Manager Opening Screen (PC-type Systems)

Descriptions of the identified parts of the XDM Opening Screen follow:

- **Menu Bar** is at the top of the Main Screen and contains commands and utilities available from XDM. Clicking on the Menu Bar brings up the various menu options.
- **Main Screen** is below the Menu Bar and contains the version number and copyright information for XDM. In the lower-left corner of the Main Screen are four Setting fields: Family, Directory, Part, and Mouse. To change these settings, place the cursor over the field and click the left button. A pop-up menu displays a list of items that you can select.

- **Family** specifies the family of Xilinx devices that is the target for your design. For example, to choose EPLD devices, select XC7200 or XC7300. Your selection in this field determines which commands are available from the menu. For example, the PLUSASM command appears on the Translate menu only if you choose XC7200 or XC7300.
- **Directory** specifies the design directory in which you are working.
- **Part** specifies the part that is the target of your design, for example, XC7236–30PC44.
- **Mouse** specifies the mouse mode (or connection port on the PC).

- **Command Line** is where you enter commands via the keyboard. You can re-execute a command by pressing ↵ when the desired command appears at the Cmd: prompt. Use the keyboard arrow keys to cycle through previously executed commands. You can enter commands when the cursor is blinking on the command line. If the cursor does not appear, click on the Command line with the mouse. Use the backspace and arrow keys to edit commands on the Command line.

To quit XDM and return to DOS, select Quit on the menu bar or enter the quit command on the command line at the bottom of the XDM screen.

NOTE *You can temporarily return to DOS without quitting XDM by using the DOS command in the Utilities menu. See the command description later in this chapter.*

If your menus do not display all of the commands included with your Xilinx software product, check your autoexec.bat file to verify that your PATH is properly set to include the executable directory paths specified during installation. For example,

```
PATH ...;drive:\XACT;...
```

If your PATH is not present, modify your autoexec.bat file to include the missing path, then reboot your system. You might also have to update your proglist.xdm file. To do this, start XDM and select the ScanDisk command from the Utilities menu.

Using XDM on a Workstation

You must have X-Windows running on your workstation before you start XDM. This subsection briefly describes the X-Windows environment and how to start XDM.

NOTE *You should not start XDM in the background mode. XDM calls some programs that go into an interactive mode and can cause XDM to halt if it is running in background mode.*

About X-Windows and Graphic User Interfaces

Xilinx software requires X-Windows to operate on a workstation. The recommended graphic interface is Motif. X-Windows is an industry-standard windowing environment developed by the Massachusetts Institute of Technology (MIT). The appearance of the windows, mechanisms used to manipulate windows, and the mouse definition are part of Motif. X-Windows and Motif are available on Apollo and Sun-4.

Xilinx software is also compatible with the Display Manager on Apollo and Openlook on Sun-4.

Both X-Windows and the Motif window manager are configurable. Configuration files control mouse operation, menu contents, screen display, and window appearance. The .Xdefaults file controls the X-Windows environment and the .mwmrc file controls the Motif window manager. These files should be in your home directory.

There are some basic operations you should know that apply to most configurations. The following paragraphs describe these operations.

Mouse Configuration

You can program the mouse buttons to start menus, select objects, and select text. The modes of a mouse button can vary depending upon the location of the cursor. Buttons operate differently if the cursor is located in an X-terminal window, X-terminal banner or edge, or outside an X-terminal window. Specific mouse button operation is dependent upon your configuration.

Window Operations

You can easily modify the size and location of windows within the X-Windows or Motif environment. You can start window operations through menus or using accelerator keys. The basic window operations are as follows:

- "Move" allows you to move the window to a different location on the screen.
- "Size" allows you to alter the size of the window by clicking an edge or corner of the window and dragging the cursor.
- "Iconify/Minimize" transforms the window into an icon. Double clicking on the icon restores the window to its original size.

- "Front" brings the window to the front of the display, overlapping other open windows.
- "Back" sends the window to the back of the display; other open windows overlap the window sent to the back.
- "Pop" toggles the window between the front and back of the display. When you initially select Pop, the window comes to the front.
- "Close" closes the window and removes it from the active display.

Window Buttons

Many window configurations are defined with three buttons in the banner of the window. These buttons are menu select, iconify and maximize. The menu select is in the top left corner; the window iconify and maximize are in the top right corner.

Window Accelerator Keys

You can execute many window operations with accelerators keys. Typical window operations are window resize, move, and front. You can define these window operations with Esc, Control, or meta ⟨◊⟩ keys, or any other key in your .XDefaults file. For example, you might define Esc v to scroll down a screen.

Using the mouse, resizing a window is a click-and-drag operation on the window edge or corner; moving a window is a click-and-drag operation on the window banner; bringing a window to the front is a click operation on the banner.

Active Window

The active window is usually highlighted to allow easy identification. Xilinx software uses the focus method that is defined in the X-Windows environment.

Edit Functions

Several keys and key sequences that work in the X-Window environment also apply to the Xilinx software. For example, you can use the up/down keys for command recall. Other key sequences are as follows:

- Control–U erases an entire command line
- Backspace or Control–H erases the character to the left of the cursor
- Delete erases the character to the right of the cursor, except at the end of the line, when it erases the last character on the line

- F1 or Help brings up on-line help.

Starting XDM

Enter the following command from the operating system prompt:

```
XDM
```

WARNING!!! *Use all capital letters when you type 'XDM' at the prompt.*

After you enter the XDM command, the Design Manager software loads into memory and reads the proglist.xdm file to set up your menus. While this is being done, the following message displays:

```
Reading proglist.xdm...
```

NOTE *If proglist.xdm is missing, as it will be in the very first invocation of XDM, XDM automatically generates this file, using the ScanDisk command. See the ScanDisk command description in "The XDM Menu" section in this chapter.*

After a short while, this message appears:

```
Reading xdm.pro...
```

When XDM is completely loaded, the XDM opening screen displays. See Figure 1-3 for an illustration of the XDM opening screen. The opening screen consists of four areas: Command Window, Menu Bar, Main Screen, and Settings.

- **Command Window** consists of three areas: Status Line, Instruction Line, and Command Line.
  - The top line is the Status Line, which displays responses to the commands you enter (either with the keyboard or the menus) and a history of commands you have entered at the Command Line.
  - The middle line is the Instruction Line, which tells you to press any key to continue whenever a program or command completes execution. Pressing any key at this time brings back the original XDM window.
  - The bottom line is the Command Line at which you enter commands via the keyboard. You can re-execute a command by pressing Return when the desired command appears at the Cmd: prompt. Use the keyboard arrow keys to cycle through previously executed commands. You can enter commands when the cursor is blinking on the command line. If the cursor does not appear, click on the Command line with the mouse. Use the backspace and arrow keys to edit commands on the Command Line.

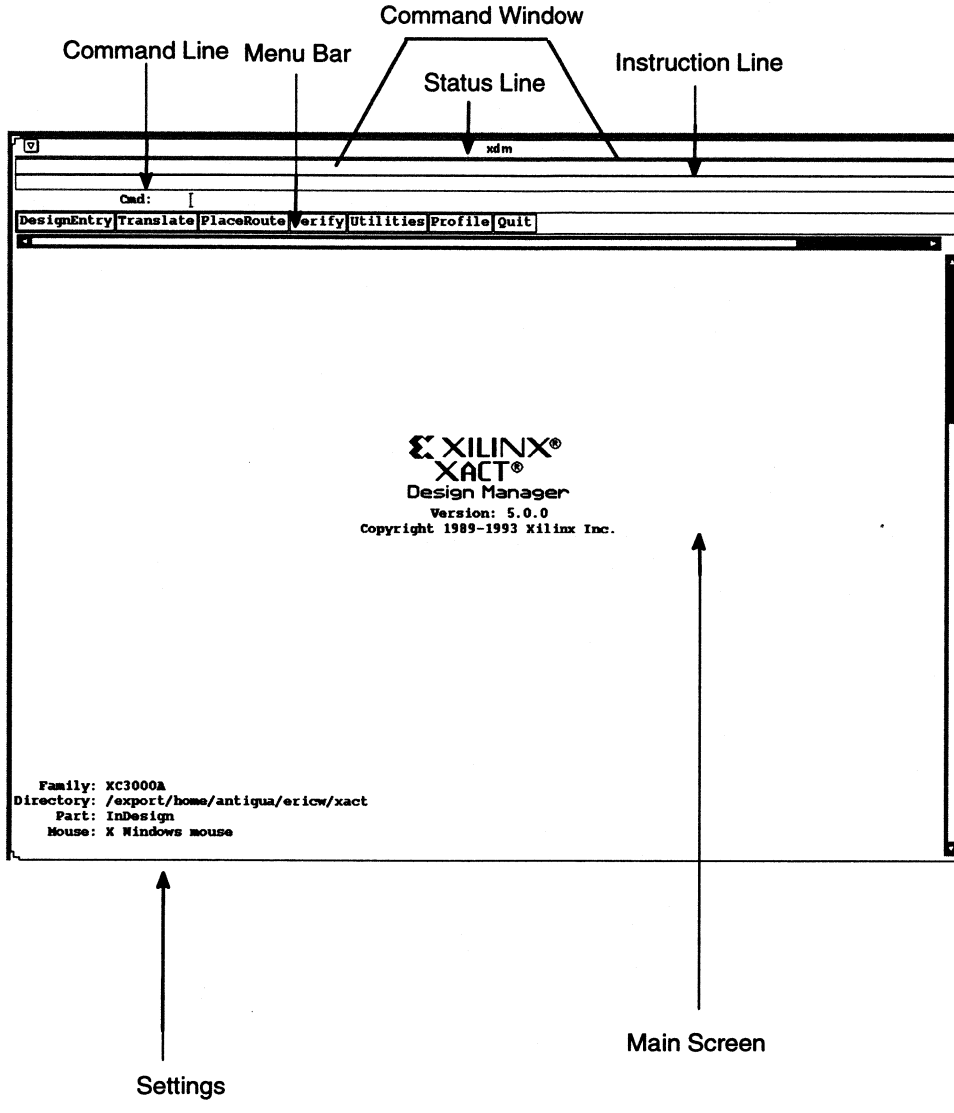


Figure 1-3 XDM Opening Screen (Workstations)

- **Menu Bar** is below the Command Window and contains commands and utilities available from XDM. Clicking on the menu bar brings up the various menu options.
- **Main Screen** is below the Menu Bar and contains the version number and copyright information for XDM. When XDM runs a program, the Main Screen is replaced by a text window where standard output from the program displays. When the program completes, XDM waits for confirmation before removing the text screen and continuing.
  In the lower-left corner of the Main Screen are four Setting fields: Family, Directory, Part, and Mouse. To change these settings, place the cursor over the field and click the left button. A pop-up menu displays a list of items that you can select.
  - **Family** specifies the family of Xilinx devices that is the target for your design. For example, to choose EPLD devices, select XC7200 or XC7300. Your selection in this field determines which commands are available from the menu. For example, the PLUSASM command appears on the Translate menu only if you choose XC7200 or XC7300.
  - **Directory** specifies the design directory in which you are working.
  - **Part** specifies the part that is the target of your design, for example, XC7236–30PC44.
  - **Mouse** specifies the mouse mode.

To quit XDM and return to the operating system prompt, select Quit from the menu bar or enter "quit" on the Command Line.

NOTE *When running your Xilinx software in X-Windows, you can return to a system command prompt by simply opening another window.*

NOTE *If your menus do not display all of the commands included with your Xilinx software product, verify that your path variable is set correctly. It should include the directory in which the Xilinx tools reside. The proglist.xdm file might also need updating. To update this file use the ScanDisk command from the Utilities menu.*

Running XMake within XDM

The most important feature of XDM is the capability of running the XMake program. XMake uses schematic-to-XNF translators and the necessary design implementation tools to automatically convert a design file into a bitstream file. This is the most automated way to implement your design. To run XMake, follow the steps below.

NOTE *XMake supports only FPGA implementation flows. An equivalent capability, called XEMake, also exists in XDM to support EPLD design flows (XC7000 family).*

NOTE *OrCAD users must run XDRAFT before entering the schematic and selecting XMake.*

1. Select the XMake command from the Translate Menu.
2. Select the XMake options you want from the displayed menu. Available options are dependent upon the current Family settings.

NOTE *Unless the current design uses Unified Libraries, you must select the –l option in XMake. This option automatically selects the 'Use-old-library-only' option when starting the translators.*

3. Select Done.
4. Select the input file from the displayed menu.

   The files in this menu vary, depending on the options you selected in step 2. For example, if you selected the –x option, the menu only contains XNF files. If you did not select the –x or –g option, the menu contains all recognized schematic and MAK files.

   When reprocessing a design that you have modified, you should select the corresponding MAK file. However, if any modules have been added or deleted, you must select the original design file as input, so that XMake generates a new MAK file.

5. At this point, XDM displays the target menu. Select the item that describes the stage at which you want XMake to stop.

XMake starts translators and design implementation tools with the current option settings for each tool. Use the Options command from the Profile menu to examine and select options for each tool (program).

NOTE *There are a number of options that XMake does or does not use, regardless of the current setting, when processing a design. This feature ensures that there are no errors caused by the exclusion or inclusion of such options, in the context of automatic, continuous design flow.*

For example, XMake always uses the –b option for WIR2XNF, and the –w option for APR, and always ignores the output_xnf option for ABL2XNF.

XDM User Interface

There are two methods for executing XDM commands. These methods are the same for PCs and workstations. You can use the mouse pointer to open a menu and select the desired command; or, you can enter the command and options on the Command Line.

With each method, messages from commands appear in the window from which you started XDM. While a command is running, the cursor takes the shape of a clock. When the command finishes, a beep sounds, and the cursor changes back to its original form. (You can select the cursor type using the Cursor command in the Profile menu.)

The Graphical Interface

Using the graphical interface to execute commands or programs requires that you first know where that command resides in the XACT Design Manager menu structure.

IMPORTANT

You also need to know to which operations the mouse buttons are set. The following usage descriptions assume that the mouse is set to the default button configuration. The default mouse settings is: Select for B1 (left button), Menu for B2 (middle button), and Done for B3 (right button).

To open a menu, position the cursor on the menu you want to open and click the left mouse button. On PCs, you can also recall the last command selected by pointing anywhere except the menu titles and pressing the middle button.

The Command Line Interface

You can use the keyboard to enter commands that are shown in the menus. Commands with two or more capital letters indicate a keyboard shortcut; commands that appear in all capital letters have no keyboard shortcut.

For example, the Utilities menu displays a command, DirClean. Using the keyboard, you could enter this command in one of three ways; DirClean, DirC, or DC (followed by a ↵).

Keyboard shortcuts are not case sensitive. In the aforementioned example, you could enter dc, Dc, or dC to run DirClean.

On PCs, when you open either the Utilities or Profile menu, the commands display with some or all of the characters highlighted. The highlighted characters represent an additional shortcut for entering the command on the command line.

The XDM Menu

This section includes general information about the XDM menus. Depending on which Xilinx software you have installed, your system might display fewer menus than are described here.

After a general description of the possible menus, more detailed information is provided for the Utilities and Profile menus.

WARNING!!! *On PCs, if your menus do not display all of the commands included with your Xilinx software product, check that your path environment variable is set correctly in your autoexec.bat file. Typically, it should include "C:\XACT" — the directory where the Xilinx tools typically reside. If it is not present, modify your autoexec.bat file to include it, then reboot your system. The proglist.xdm file might also need updating. Do this by starting XDM and selecting ScanDisk from the Utilities menu.*

WARNING!!! *On workstations, if your menus do not display all of the commands included with your Xilinx software product, verify that your path variable is set correctly. It should include the directory in which the Xilinx tools reside. The proglist.xdm file may also need to be updated. Do this by invoking XDM and selecting ScanDisk from the Utilities menu.*

The Design Entry Menu

This menu contains a listing of the design entry software packages installed in your system. Supported packages include Workview, OrCAD, XABEL, and SYMGEN (Xilinx's symbol generator).

SYMGEN

This tool translates an XSF file into a symbol file. SYMGEN can generate symbols in text, OrCAD, Viewlogic, and Cadence formats.

The Translate Menu

This menu contains the translation programs that produce an XNF or LCA design file (though not all of the programs produce these files directly). The Xilinx device family and implementation method you use determine which translation programs appear on the menu, and whether your resulting file type is XNF or LCA. This menu contains all of the translations required to support your design entry package.
In particular, the XMake program (XEMake for XC7200 and XC7300 family) is useful for automatically translating and implementing your design. XMake and XEMake eliminate the need to manually run individual tools.

XMake

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H.

Xmake uses the necessary schematic-to-XNF translation programs and appropriate design implementation tools in succession to automatically convert a design file into a BIT file. At the beginning of this process it generates a MAK file. A MAK file contains the information about how a design is to be processed. You can use the MAK file for any subsequent invocations of XMake for the same design. Running XMake within XDM is the most automated way to implement your design. Refer to the Running XMake within XDM section for more information. For details about the XMake program, refer to the XMake Program chapter in this reference guide.

XEMake

Supports XC7200/XC7300.

XEMake is the EPLD version of XMake, which supports both schematic and behavioral (equation-file) designs. XEMake automates the schematic or equation-file integration process by running the proper EPLD tools in succession. XEMake accepts OrCAD, Viewlogic, PLD files, PDS files, MAK files (generated from XEMake only), or XNF files and integrates it into a database file (VMH/VMD). Optionally, you can specify XEMake to create an Intel HEX programming file (PRG).

ABL2XNF

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H.

This program translates ABL files to an XNF format.

ABL2PLD

Supports XC7200/XC7300

This program creates a PLD file from an Abel file and runs PLUSASM to create a bitmap file. Optionally, it can run FITEQN if your Able file is a top-level file.

Annotate (OrCAD Interface Only)

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A/L, XC4000/XC4000A/H, XC7200/XC7300.

This OrCAD command updates reference designators in OrCAD schematics in the order in which they were placed on the schematic as the first step in preparing an ORCAD schematic for functional

simulation. Annotate can assign new reference designators to all parts including manually edited parts, to ensure that they are unique.

CleanUp (OrCAD Interface Only)

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H, XC7200/XC7300.

This OrCAD command cleans up overlapping objects in OrCAD schematics.

HM2RPM

Supports XC4000/XC4000A/H.

This command translates a Hard Macro into a Relationally Placed Macro. Hard Macros are not supported by PPR versions later than V1.3.1, and must be converted into Relationally Placed Macros for newer versions of PPR.

INET (OrCAD Interface Only)

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H, XC7200/XC7300.

This OrCAD command reads the root schematic and any other related schematics and converts them into netlist files (INF files) as the second step in preparing an OrCAD schematic for functional simulation.

JED2PLD

Supports XC7200/XC7300.

This program imports a JEDEC file and uses it to define the functional behavior of a PLD component in a schematic. First, it translates the JEDEC file to a PLUSASM equation file (with a PLD extension). Then, it starts the PLUSASM assembler, which assembles the equation file into a bitmap. You can view and edit the PLUSASM equation file.

MAP2LCA

Supports XC2000/XC2000L, XC3000/XC3100.

MAP2LCA translates a MAP file into an LCA file. You can edit this LCA file in the XACT Design Editor or use APR to automatically place and route the file.

MemGen

Supports XC4000/XC4000A/H.

MemGen is the Xilinx RAM and ROM memory compiler for the XC4000 family of devices. Given the type of memory and its width, depth, and contents, MemGen creates the appropriate XNF file to implement the memory. It can also create a schematic symbol for your memory function.

PinSave

Supports XC7200/XC7300.

This command saves the pin allocation information into a VMF file. Use this command after a successful integration of your design. If you set the Pinfreeze option of either the FITEQN or FITNET commands to on, the Integrator, during subsequent iterations of your design, assigns the pins to the same locations indicated in the VMF file. You can edit the VMF file to alter pin assignments.

PLUSASM

Supports XC7200/XC7300.

PLUSASM assembles a PLUSASM equation file that describes a PLD used in a schematic design. PLUSASM assembles the source file you select and generates a component bitmap file and a report. FITNET on the Fitter menu uses the Component bitmap files.

SDT2XNF (OrCAD Interface Only)

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H, XC7200/XC7300.

SDT2XNF generates an XNF file from an OrCAD netlist file.

SYN2XNF (Synopsys Interface Only)

Supports XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H.

SYN2XNF generates an XNF file from a Synopsys SEDIF or SXNF file.

WIR2XNF (Viewlogic Interface Only)

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H, XC7200/XC7300.

WIR2XNF translates a WIR file into an XNF file.

X-BLOX

Supports XC3000A/L, XC3100A, XC4000/XC4000A/H.

X-BLOX performs data path and architectural synthesis. X-BLOX generates an XNF file, with an XG extension, from a schematic or netlist drawn with X-BLOX modules.

XDRAFT (OrCAD Interface Only)

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H, XC7200/XC7300.

XDRAFT sets up your SDT.CFG in the current directory (OrCAD SDT configuration file) and/or VST.CFG (OrCAD VST configuration file) to run with the Xilinx OrCAD environment. You must run XDRAFT once for each design before running XMake.

XNFMAP

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L.

This program maps the logic defined in the XNF file into the architectural resources of the LCA device. XNFMAP reads the XTF file and decides which gates to combine into a CLB. XNFMAP first maps the logic that has been assigned to specifica locations in the schematic.

XNFMerge

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H, XC7200/XC7300.

XNFMerge combines multiple XNF files to create a single flattened (non-hierarchical) design file. It also generates a report file (*design*.mrg) listing the files read; the signals that were bound together; and the number of signals, primitive symbols, and unresolved symbols in the output design file.

XNFPrep

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H.

XNFPrep performs a design rule check (DRC) and removes unused and redundant logic from a flattened XNF file. It also checks the syntax of the XACT-Performance parameters found in the design and prepares delay information for PPR path analysis.

The PlaceRoute Menu

This menu contains commands that run the XACT Design Implementation algorithms that place CLBs and IOBs in the LCA and route all of the required connections. For XC3000A/L and XC4000

designs, PPR also partitions the logic in the design file before placing and routing.

APR

Supports XC2000/XC2000L, XC3000/XC3100.

APR reads an input design map file, places the CLBs and IOBs in a specific part, then routes the design. APR produces an LCA file.

APRLoop

Supports XC2000/XC2000L, XC3000/XC3100.

APRLoop allows you to perform multiple iterations of the APR software. This allows you to evaluate several LCA files and select the best solution. When you select APRLoop, the program prompts you to specify the number of iterations you want it to perform. For more information, refer to the APR chapter in the *XACT Reference Guide*.

PPR

Supports XC3000A/L, XC3100A, XC4000, XC4000A/H.

PPR reads an XTF file (generated by XNFPrep) for XC4000 and XC4000A/H designs, maps the logic into CLBs and IOBs, and places and routes the design in a specific part. PPR also reads a MAP file (generated by XNFMAP) for XC3000A/L and XC3100A designs, and places and routes the design in a specific part. PPR produces an LCA file.

XDE

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000, XC4000A/H.

XDE is the XACT Design Editor. XDE is a graphical representation of an implemented design. Use XDE to manually modify your design.

The Fitter Menu (XC7200 and XC7300 only)

This menu appears on the XDM menu bar only when you select an EPLD device (XC7200 or XC7300) for the "Family" field. The commands on this menu execute the XEPLD design implementation algorithms that map the design onto the target EPLD device. For more details about this menu, refer to the *XEPLD User Guide*.

FITEQN

Supports XC7200/XC7300.

FITEQN integrates a behavioral design. The main equation file you specify must follow the required PLUSASM file structure. The equation file is processed by several XEPLD modules to produce a database, VMD (only for XC7272), or VMH (all other EPLD devices). From this database, you can produce a programming file to program the device. You can also produce a timing simulation XNF file using VMH2XNF and save the pinouts using the PinSave program. Use MAKEPRG to generate an Intel HEX file or MAKEJED to generate a JEDEC file.

This command produces several reports: Resource (RES), Mapping (MAP), Pinlist (PIN), Partition (PAR), and Collapse/Logic Optimization (LGC). FITEQN also produces a *design_name*.log file and a behavioral design file, which contains partitioning and other information, with the name *design_name*.eqn.

FITNET

Supports XC7200/XC7300.

This command integrates a schematic-based design. The input file is a merged, flattened XNF file (XFF) from XNFMerge that is processed by several XEPLD modules to produce a database. From this database, a programming file can be produced to program the device. You can also produce a timing simulation XNF file using VMH2XNF and save the pinouts using the PinSave program. Use MAKEPRG to generate an Intel HEX file or MAKEJED to generate a JEDEC file.

This command produces several reports: Resource (RES), Mapping (MAP), Pinlist (PIN), Partition (PAR), and Collapse/Logic Optimization (LGC). FITNET also produces a *design_name*.log file and a behavioral design file, which contains partitioning and other information, with the name *design_name*.eqn.

PALCONVT

Supports XC7200/XC7300.

This command creates a new behavioral design file from existing PAL designs, which are input in the form of PDS or PLD files.

PALCONVT creates a PAL interconnect report (INT) that lists all the signals used by all PALs and indicates any conflicts.

These PAL files can be PLUSASM files that you created with a text editor, files from third-party design entry packages (like ABEL), or JEDEC files that you convert with the JED2PLD command.

After using PALCONVT, choose a target device for your converted design, then use FITEQN to integrate your design. Assuming that there are no edits you want to make to the new PLD file, you can also choose the "run FITEQN target" option.

The Verify Menu

The Verify menu provides a selection of programs associated with design simulation and in-circuit verification. These selections include all simulation and verification programs and all utility programs needed to create the required file formats.

ASCTOVST (OrCAD Interface Only)

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H.

ASCTOVST converts an OrCAD Stimulus or Trace file from ASCII to binary format and from binary to ASCII.

LCA2XNF

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H.

LCA2XNF translates an LCA file into a timing-annotated XNF file that can be used to create a timing simulation file.

MakeBits

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H.

MakeBits creates a bitstream that can be downloaded into an LCA.

MAKEJED

Supports XC7200/XC7300 (except XC7272/XC7272A devices).

MAKEJED creates a JEDEC file that you can use to program XEPLD devices. This command prompts you for a signature, which you must specify. A signature is a series of letters or numbers that indicates the revision of the design. The device programmer reads the signature allowing you to verify that the version is correct.

MAKEPRG

Supports XC7200/XC7300.

MAKEPRG creates a file in Intel HEX format that you can use to program XEPLD devices. This command prompts you for a signature,

which you must specify. A signature is a series of letters or numbers that indicates the revision of the design.

MakePROM

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H.

MakePROM creates a PROM programming file from a configuration BIT file. MakePROM also combines multiple BIT files for use in a daisy chain of LCA devices.

ORCAD (VST)

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H, XC7200/XC7300.

ORCAD places you in the OrCAD/ESP design environment where the OrCAD VST simulator is located.

PROLINK

Supports XC7200/XC7300.

PROLINK starts the PROLINK interface software for controlling and downloading HEX programming files (generated by MAKEPRG) to the Xilinx DS120 device programmer.

VMH2XNF

Supports XC7200/XC7300.

VMH2XNF creates an XNF file with timing parameters for use in timing simulation. The input file can be a VMH or VMD (from XC7272) file.

VSM (Viewlogic Interface Only)

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H, XC7200/XC7300.

VSM reads the model file produced by XNF2WIR and creates a Viewsim wirelist (with a VSM extension) for functional and timing simulation.

VSMUPD (Viewlogic Interface Only)

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H.

VSMUPD updates the Viewsim VSM file to allow complete back-annotation to the original schematic during timing simulation.

VSMUPD adds net-equivalent statements to the VSM file that enable Viewsim to back-annotate more net values to the original schematic.

XSimMake

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H, XC7200/XC7300.

XSimMake automatically prepares any design for either functional or timing simulation. XSimMake is not available for Mentor Graphics users.

XChecker

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H.

XChecker downloads, reads back, and verifies the configuration data, and probes the internal logic states of FPGAs.

XDelay

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H.

XDelay is the static timing analyzer that reports detailed timing information about the design, which you can use for overall performance analysis.

XNFBA

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H.

XNFBA combines the pre-route XG/XFF file and the post-route XNF file into a new file with pre-route names and post-route delays.

XNFCVT

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H.

XNFCVT converts an XNF netlist from Version 5 to Version 4, 2, 1.

XNF2VST (OrCAD Interface Only)

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H, XC7200/XC7300.

XNF2VST creates an OrCAD VST simulation file from an XNF format file.

XNF2WIR (Viewlogic Interface Only)

Supports XC2000/XC2000L, XC3000/XC3100, XC3000A/L, XC3100A, XC4000/XC4000A/H, XC7200/XC7300.

XNF2WIR uses an XNF file as input to create a WIR file that Viewlogic's VSM netlister program can read.

XPP

Supports 1736AMD, 1765AMD, XC1736, XC1736A, XC1718D, XC1718L, XC1736D, XC1736L, XC1765D, XC1765L, XC17128X.

XPP allows you to program a Xilinx serial PROM device with the DS112 programmer.

NOTE *If you need more details about the commands listed in the Design Entry, Translate, and Verify menus, consult the appropriate CAE User Interface Guide.*

The Utilities Menu

The Utilities menu provides several utility commands that let you change working directories, see which versions of Xilinx-supported software are installed on your system, view file contents, and use other file management and system control features. The commands in the Utilities menu are not related to the development software loaded in your system; they are the same regardless of your software configuration.

Browse

This utility allows you to view, as a read-only document, any text file from within XDM. After selecting Browse, you are prompted for a file name. To browse through a file, select the file name from the menu or enter it from the command line. Press F1 to return to XDM. If you want XDM to start your own text viewing program when you select Browse, you can set an environment variable to the name of the text-viewing program. Consider the following PC example.

```
set BROWSE=LIST
```

Selecting Browse will call the LIST program with this setting.

NOTE *Browsing a design within XDM can be slow on a PC and should only be used if you do not have a text editor on your PC. Browse has a maximum display line limit of 800 lines.*

DirClean

This utility helps you manage your design directories by eliminating unwanted files that the design translation process creates. To use DirClean, select any files you want to remove from the current directory; XDM highlights the selected files. When you complete your selections, click on Done or press ↵ to execute their removal.

Directory

The Directory command allows you to move easily through your disk directory structure. You can select the current directory that XDM reads from and writes to. A menu appears displaying the current directory, the parent directory, and all subdirectories of the current directory. Each time you select a new current directory, the menu changes to reflect the new parent directory and subdirectories.

To change disk drives on PCs, enter the following command at the command line.

`dir` ***drive:***

Where *drive* is the disk drive you want as your current drive.

DOS (PC Only)

This command is a gateway to the DOS operating system environment. You can use it on the command line either alone or as a prefix to an operating system command. To access the DOS environment, simply select the DOS command with the mouse or enter the command on the command line. To re-enter the XACT Development System, enter *exit* at the operating system command prompt.

If you want to execute a system command from the Design Manager, enter it in the following manner.

`dos` ***command***

This executes the command or program and returns you to XDM upon completion of the command or program.

Edit

Browsing a file within XDM can be slow on a PC and should only be done if you do not have a text editor on your system. If you want XDM to start your own text editing program when the Edit command is selected, you can do so by setting the editor environment variable to the name of your text editing program. Consider the following PC example:

```
set editor=vi
```

This command is not necessary on workstations, because you can simply open another window to edit a file.

Execute

The Execute command allows you to execute command files inside XDM. If you save a sequence of XDM commands in a text file, you can execute them by first selecting Execute, then entering the command file name. You return to XDM when the file finishes executing. The command file must contain legal XDM commands.

Help

The Help command provides several methods for getting assistance about a particular topic. On a PC, you can select a menu item and press the F1 function key to display the online help. You can also select the Help command from the menu, which displays the following message at the bottom of the screen:

```
Enter help subject:
```

This display prompts you to select a help topic. An alternative method for displaying help information is to enter the Help command, followed by the topic or option.

help [*topic–option*]

For example, to display Help information about the –g option in APR, enter the following syntax at the command line.

help apr-g

NOTE *There are no spaces between the program name and the option.*

Report

The Report command starts the Version command; however, instead of displaying the output on your screen, it redirects the output to a text file called *version*.rpt. You can read the text file at any time using the Browse command.

ScanDisk

The ScanDisk command causes XDM to scan the hard disk drive, according to the current setup, to determine which supported software packages are installed on your system. While scanning, XDM displays the following message:

```
Checking disk for supported software...
```

This messages indicates that XDM is analyzing your system and setting up the contents of the DesignEntry, Translate, PlaceRoute and Verify menus, so they reflect the software that is available.

Then XDM displays a message similar to the one shown below.

```
Writing c:\xact\data\proglist.xdm...
```

After a short while, another message indicates that the setup is complete and XDM has generated a new, updated proglist.xdm file.

```
Writing c:\xact\data\proglist.xdm... done
```

IMPORTANT

Newly installed XDM-supported programs on your system might not appear in the XDM menus until you run the ScanDisk command. For ScanDisk to find an XDM-supported program, your PATH environment variable must be set to include all directory paths specified during installation.

Version

The Version command displays all supported programs currently installed on your system, showing the location and version numbers for each program. XDM might not be able to determine the version number of some programs, such as Xilinx-supported third-party programs.

The Profile Menu

This menu serves to customize XDM. Using the commands in this menu, you can alter such characteristics as screen graphics, mouse port connections (if you are using a serial mouse on a PC), and device type and speed. Changes that you make from the default profile are only valid for the current session until you save them. To save a customized profile use the Saveprofile command.

Cursor

The Cursor command allows you to change the shape of the cursor. You can select an arrow, a bug, a cross, or a gunsight.

Family

The Family command tells XDM the family of LCA devices you are using. XDM only displays valid menu items and command options for the selected family. Current choices include XC2000, XC2000L XC3000, XC3100, XC3000A, XC3000L, XC3100A, XC4000, XC4000A, XC4000H, XC7200/XC7300.

KeyCursor

When KeyCursor is enabled, the arrow keys move the cursor through pull-down menus; pressing ↵ executes the selected option. You must enter commands through the keyboard or use the mouse to select them from pull-down menus. The default for the PC is on. The default for the workstation is off.

Keydef

The keydef command allows you to program your system function keys. After selecting this command, XDM prompts you for a key name (such as, F1, F2, F3, and so forth.) and then a function. The function can be any XDM command. To start an external text editor with the push of a button, program one of the function keys. If you like the vi editor, for example, enter the following on the command line to program F2 to start the vi editor on a PC.

```
keydef F2 dos vi
```

Menucolors

The Menucolors command allows you to change the color of items displayed in menus. Use Help on the individual commands in this menu for more information on their functionality and usage.

Mouse

The Mouse command sets the function of each mouse button. The default settings are Select (B1), Menu (B2), and Done (B3). B1 is the left mouse button, B2 is the middle mouse button, and B3 is the right mouse button.

NOTE *On PCs only, if a Microsoft-compatible mouse driver is loaded, the connection port is automatically determined by XDM. This is indicated by "Mouse: MS Mouse" being displayed on the screen. If the driver is not loaded you must select the connection port through this command. In this case, the selected port name is displayed on the screen (for example, "Mouse: COM1")*

Options

The Options command allows you to select default options for all Xilinx software programs. Once selected, they are valid for the current session. Use the SaveProfile command to save these options in the xdm.pro file.

Palette

The Palette command allows you to choose different color palettes for customizing your screen color.

Part

The Part command allows you to select a default part type to use when translating a design. You can select the InDesign option if the part type is specified in the schematic.

Readprofile

The Readprofile command allows you to load the profile saved in the xdm.pro file. This command tries to read a custom profile from the current directory. If one is not found it loads the default configuration profile in the XACT/data directory.

Saveprofile

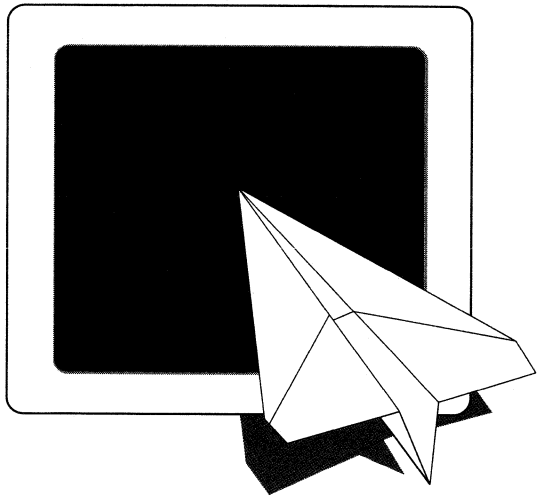
The Saveprofile command allows you to save the current profile into an xdm.pro file in the current directory. Each time XDM is started, it tries to read an xdm.pro file from the current directory. If one is not found it will load the default configuration profile from XACT/data.

Settings

The Settings command displays the current profile configuration of your Design Manager.

Speed

The Speed command allows you to select a speed grade for the device specified with the Part command. If you have selected InDesign as your part type, you must specify the speed grade in the design. In this case, you cannot specify it using the Speed command.



The XMake Program

XACT Reference Guide, Volume 1

XACT Reference Guide, Volume 1 — April, 1994 (0401129 01)

Printed in U.S.A.

The XMake Program

This Program is Compatible with the Families Indicated.

☑ XC2000	☑ XC3100	☑ XC3100A	☑ XC4000H
☑ XC2000L	☑ XC3000A	☑ XC4000	☐ XC7200
☑ XC3000	☑ XC3000L	☑ XC4000A	☐ XC7300

The XMake program uses the necessary schematic-to-XNF translation programs and appropriate design implementation tools in succession to automatically convert a design file into a BIT file. At the beginning of this process XMake creates a MAK file. This file contains information on the programs and options used to process the design. You can use the MAK file for any subsequent invocations of XMake on the same design.

If you are using design-entry software supported by the XACT Design Manager (XDM), XMake automatically activates all the translation programs needed to convert a design into a BIT file. XMake can also automatically translate designs that contain non-schematic modules such as memory modules and Boolean equations.

You can run XMake from the XDM menu system, or by typing the command at the system prompt according to the XMake command line syntax.

Using XMake from the XDM Menu

Using XMake from the XDM menu simplifies starting XMake with the correct arguments. For detailed information, refer to the section, Running XMake from XDM in the 'XACT Design Manager' chapter.

Use the Select mouse button (default is the left mouse button).

1. Select XMake from the Translate menu.
2. Select the XMake options from the displayed menu.
3. Select Done after you have selected the options.
4. Select the top level design or MAK file to be processed from the displayed menu.
5. Specify how far you want XMake to process the design by selecting a target from the displayed menu of targets.

Using XMake from the System Prompt

Use the following syntax to run XMake:

xmake [*options*] *infile* [*target_file*]

where:

- *options* represents valid XMake options that you can specify. Refer to Options in this chapter for descriptions.
- *infile* represents the design file you want XMake to process, or the MAK file generated by the previous run of XMake on the design.

A design file can be one of the following.

- Viewlogic schematic file (*design*.1)
- OrCAD schematic file (*design*.sch)
- XSI output file (*design*.sedif or *design*.sxnf)
- Xilinx-ABEL file (*design*.abl)
- XNF file (*design*.xnf)

XMake automatically generates a new MAK file (*design*.mak), if a design file is specified as *infile*.

XMake reads the option profile to determine with which options each program should be started, when generating a MAK file. This option profile can be one of two things:

- xdm.pro file, if XMake is run at the system prompt, or
- the current XDM settings in memory, if XMake is run from within XDM.

Use a previously generated MAK file to reprocess a design to which changes have been made.

Do not use the existing MAK file for a design if any modules have been added or deleted.

target_file is an optional entry argument that tells XMake to stop processing after this file is generated. By default, XMake processes a design all the way to a configuration bitstream BIT file, unless you specify a different target. For example, with the design rolldice, you can specify rolldice.lca as a target, stopping XMake after it has generated a placed and routed LCA file.

XMake Command Line Usage

XMake checks for the following command line rules, and, if it finds violations, issues an appropriate error message and aborts processing.

- *infile* must exist in the current directory, except when it is a Viewlogic file (*design*.1). In this case, *infile* must exist in the ./sch directory. You can explicitly state the filename extension, or specify the design name and XMake supplies the default extension.
- If you specify *design* (without an extension) as the infile, XMake assumes that the file has a valid extension, and checks, in the following order, for its existence. XMake takes the first file it finds in this search sequence as its input.

  ./sch/*design*.1
  ./*design*.abl
  ./*design*.sch (PC only)
  ./*design*.sedif (Workstation only)
  ./*design*.sxnf (Workstation only)
  ./*design*.xnf

- The –a, –f, and –t options are mutually exclusive; they cannot be selected together on the same command line.
- XMake ignores the –m and –n options if you explicitly specify a target in the target field.
- If you select *design.xxx* as the *infile*, thereby causing XMake to generat a new MAK file, the target (either explicitly specified in the target field, or the default, *design*.bit, in the absence of the target field) is recorded in the DEFAULT_TARGET entry of the generated MAK file.
- If you select *design*.mak as the *infile* and do not specify a target in the target field, XMake uses the target recorded in the MAK file. If you specify the target, you override the target in the MAK file.

Files

The input files that XMake requires to process a design and the output files that XMake generates are described below. Note that the MAK file is both an input file and an output file.

Input Files

To translate a design, XMake requires one of four file formats as input: a top-level schematic file, an ASCII HDL design description file, an XMake-generated MAK file, or (if you use the –x option) a top-level XNF file.

Schematic File

XMake can only accept schematic design files created by Viewlogic and OrCAD schematic editors. See the appropriate Xilinx interface user guide for additional information.

HDL File

XMake can accept Xilinx ABEL and Xilinx Synopsys Interface ASCII design description files as input. See the appropriate Xilinx interface user guide for more details.

design.mak

As XMake runs, it uses and records, in the MAK file, the program options currently saved in the XDM profile (xdm.pro). If you use a MAK file as an input file, XMake uses the MAK file instead of creating a new one. You should create a new MAK file after changing the hierarchy of the design. Do not use a previously generated MAK file if the design hierarchy has changed.

design.xnf

XMake can accept an XNF file as a design file. When specifying an XNF file as input, you must select the –x option.

Output Files

XMake creates a number of output files based on the options specified. The programs that XMake calls also create output files. Some of the files that can be generated are listed below.

design.mak

From a schematic file input, XMake creates a text MAK file (*design*.mak) that documents how each design submodule is processed, including the options used by the translation programs. See the MAK File section in this chapter for more details.

design.out

XMake uses various translation programs and directs translation screen output to a *design*.out file, unless the –o option is selected. This is an ASCII text file containing all the screen output from the programs that XMake uses. Since the OUT file contains all warning or error messages generated during the design process, always review it to determine that your design is error-free.

design.xff

XMake creates a flattened XNF file, *design*.xff, by calling XNFMerge.

design.xtf

XMake produces a completely trimmed and flattened XNF file for the entire design, which is XTF.

design.xg

If the design involves XBLOX modules, or the you have selected the –b option, XMake produces an optimized XNF file, *design*.xg, by calling XBLOX. This applies only to the XC3000A/L, XC3100A, XC4000, and XC4000A/H designs.

design.map

XMake generates a partitioned XNF file, *design*.map, by calling XNFMAP. This applies only to the XC2000, XC2000L, XC3000, XC3100, XC3000A/L, and XC3100A designs.

design.lca

XMake creates an LCA file that is partitioned, placed, and routed by either the Automatic Place and Route (APR) program or the Partition, Place, and Route (PPR) program.

design.bit

XMake creates a bitstream file for all designs that successfully route with either APR or PPR. You can download the BIT file to an LCA. The configuration options for the bitstream generator are determined by the options set in the XDM profile (xdm.pro).

Options

XMake options are listed in alphabetical order with brief functional descriptions.

–a Use map-then-merge Strategy

Supports XC2000, XC2000L, XC3000, XC3100, XC3000A/L, XC3100A only.

The –a option causes XMake to use the map-then-merge mapping strategy. XMake automatically selects the –a option for XNFMAP, regardless of your settings in the option profile. XNFMAP options –q and –u are ignored, if selected.

–b Perform X-BLOX Optimization

Supports XC3000A/L, XC3100A, XC4000, XC4000A/H only.

The –b option causes XMake to generate a MAK file that includes X-BLOX optimization in the design-implementation flow. First, XNFPrep runs, next X-BLOX runs, then XNFPrep runs again generating a *design*.xg file. PPR takes this file as its input, rather than *design*.xtf.

NOTE *If the design file contains the DEF=BLOX or DEF=X-BLOX attribute, then X-BLOX automatically runs during processing. In this case, specifying the –b option ensures X-BLOX is run with 'archopt=TRUE', regardless of your settings in the option profile.*

–f Use map-FILE=-then-merge Strategy

Supports XC2000, XC2000L, XC3000, XC3100, XC3000A/L, XC3100A only.

The –f option causes XMake to use the map-FILE=-then-merge mapping strategy. XMake automatically selects the –q option for XNFMAP, regardless of your settings in the option profile. XNFMAP options –a and –u are ignored, if selected.

–g Generate MAK File/Do Not Process Design

The –g option causes XMake to create a MAK file only, without continuing with the commands in the MAK file to implement the design. Use this option when you want to create a custom MAK file, forcing XMake to generate an initial script that you can edit.

–i Use APR/PPR Guide File

The –i option causes XMake to automatically select the guide file option for APR (–g) and PPR (guide=). If APR –g or PPR guide= option is already selected in the profile, the guide file name specified on the XMake command line after the –i option overrides the currently selected guide file name.

The guide file must have a .lca extension. If the guide file is specified without an extension on the XMake command line, XMake automatically appends .lca to the file name.

For XC2000, XC2000L, XC3000, XC3100, XC3000A/L, and XC3100A designs, XMake also ensures that the mapping of the CLBs is guided via the XNFMAP –k option. XMake automatically runs

LCA2XNF on the guide LCA file to generate a *design*.pgf file for use by XNFMAP.

–l Use Old Library Only

The –l option causes XMake to automatically select the 'use-old-library-only' option when starting translators with such an option. If you do not select this option, XMake defaults to use the Unified Library on the selected design.

–m Make Placed and Routed Design

The –m option sets the target to *design*.lca. The MakeBits program will not be run by XMake. If you specify a target file in the target field, XMake ignores this option.

–n Stop to Review DRC

The –n option sets the target to *design*.xft if you are processing XC4000 devices. XMake will not run PPR, XDelay, or MakeBits during the process. The –n option sets the target to *design*.map if you are processing XC3000A/L devices. XMake will not run PPR, XDelay, or MakeBits during the process. The –n option sets the target to *design*.map if you are processing XC2000, XC3000, or XC3100 devices. XMake will not run MAP2LCA, APR, or MakeBits during the process. If you specify a target in the target field, XMake ignores this option.

–o Direct Output to Screen

The –o option causes XMake to direct all program output to the screen instead of generating a *design*.out file.

–p Use Specified Part Type

The –p option allows you to set or change the part type for the design.

–r Re-Execute All Commands to Translate Design

The –r option guarantees that XMake reprocesses the entire design, including unchanged submodules from the last time the design was processed. There is a difference between running XMake with the –r option on a MAK file and a schematic file, as described below.

MAK File Input with -r

XMake performs every step in the MAK file, regardless of whether the files have been changed since the design was last processed. If you do not use the –r option, XMake only reprocesses those parts of the design that have been changed.

Schematic File Input with -r

XMake recreates the MAK file and reprocesses the entire design. If you do not use the –r option, XMake recreates the MAK file and only reprocesses those modules that have changed.

–t Use merge-then-map Strategy

Supports XC2000, XC2000L, XC3000, XC3100, XC3000A/L, XC3100 only.

The –t option causes XMake to use the merge-then-map mapping strategy. XMake automatically excludes the –a, –q, and –u options for XNFMAP, regardless of your settings in the option profile.

–v Verbose Mode

The –v option causes XMake to display explanatory information on its progress.

–x XNF Only (Interface to Third-Party Schematics)

The –x option causes XMake to search for XNF files only, instead of schematic files. This option allows you to use third-party design-entry tools not directly supported by XMake to translate a design (in XNF file format) into an LCA file.

Each time you reprocess your design, you must do the following.

- Manually (perhaps using a batch file) translate each design submodule into an XNF file using the interface program(s) for your design-entry program.
- Use XMake with the –x option and any other required options.

To compile the design into a BIT file, XMake bypasses the design-to-XNF translation (since the files are already in XNF format) and runs the required design-implementation programs.

XMake Design Flow

This section contains flow diagrams to explain the design flows that XMake performs on the Xilinx LCA devices. Refer to the flow applicable to the device family that you are using.

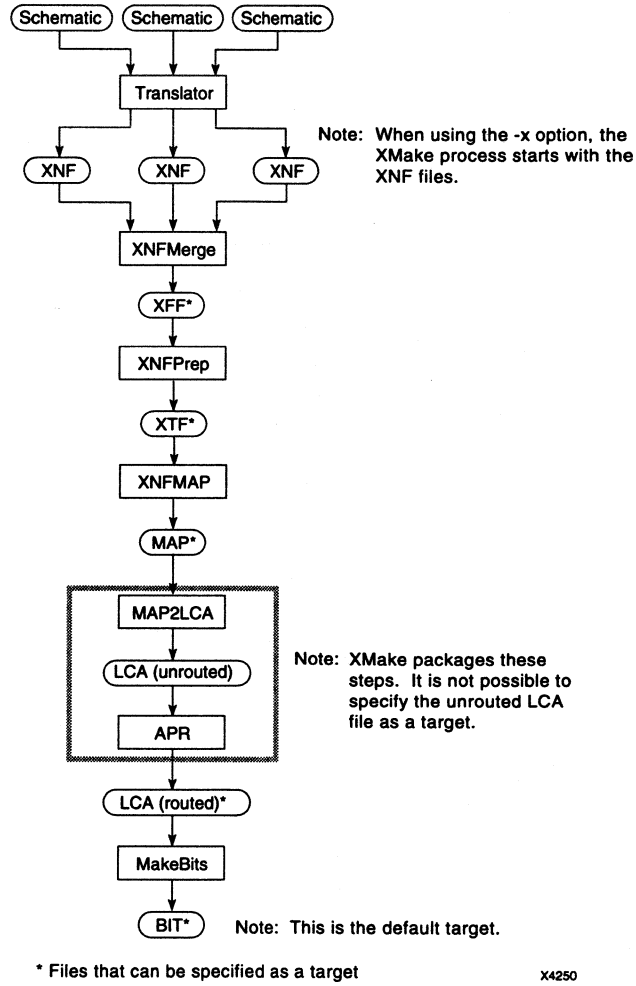


Figure 1-4 XMake Flow for XC2000, XC2000L, XC3000, XC3100 Designs

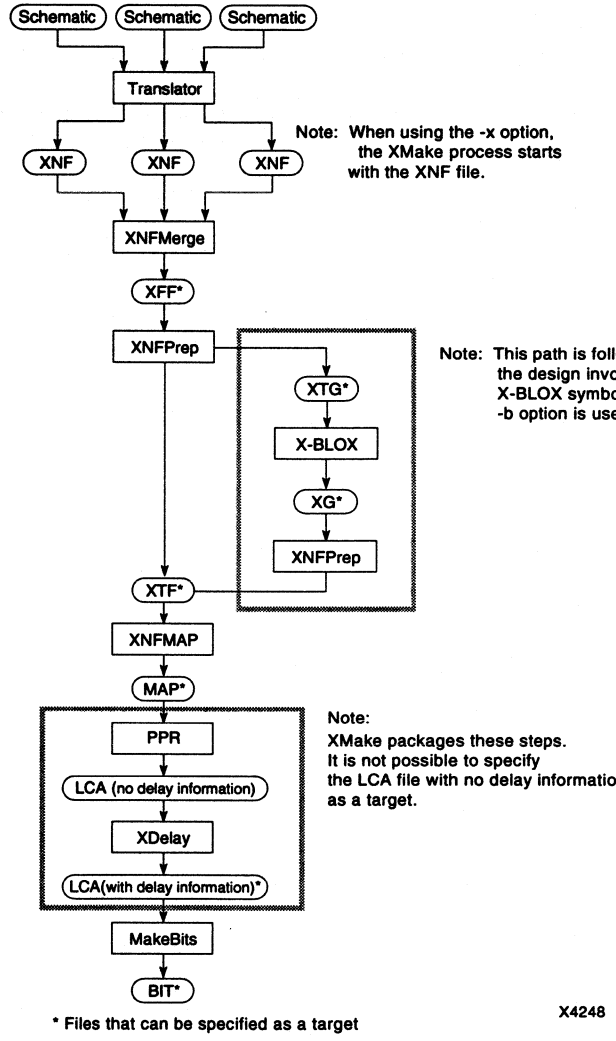


Figure 1-5 XMake Flow for XC3000A, XC3000L, XC3100A Designs

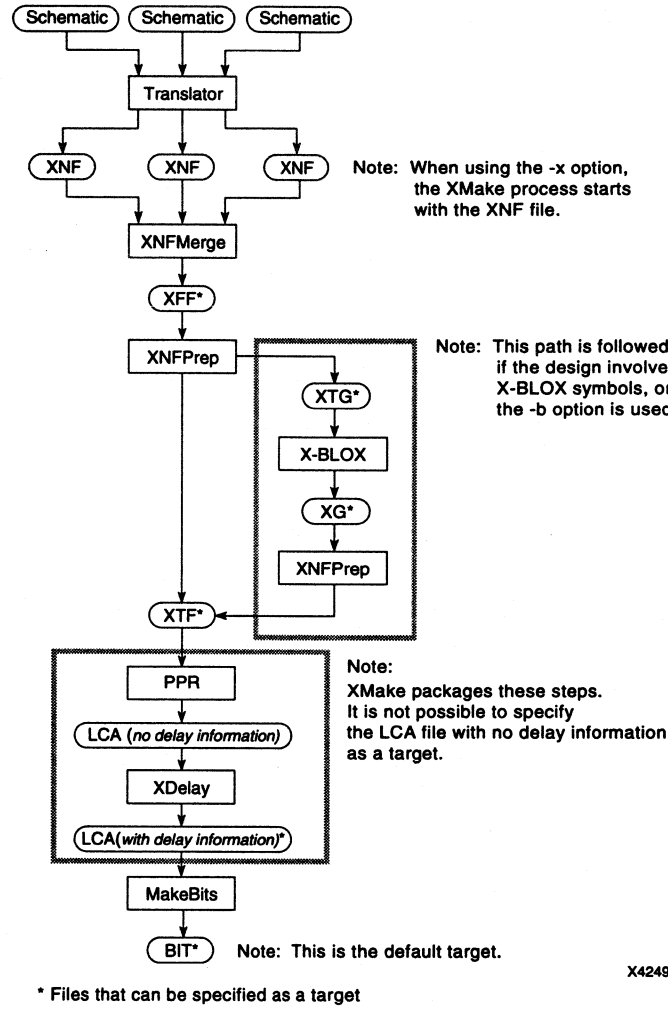


Figure 1-6 XMake Flow for XC4000, XC4000A, and XC4000H Designs

MAK File

The MAK file contains a series of target entries (statements and commands) required to produce a BIT file for the design. Each entry specifies which type of file *(target_file)* is generated, based on the changes in the *dependent_file*, and the programs (*command*) used.

Each target file entry has the following format.

target_file: dependent_file ...
command [options] ...
.
.
.

The following list explains the specific syntax requirements for the MAK file.

- The target file name must start in the first column. Leading blank spaces or tabs are not permitted.
- Each field (file name, command name, command line argument, etc.) must be separated from any others by at least one space or tab character.
- Command lines must start with at least one space or tab character.
- Blank lines are ignored.
- Comment lines must start with a # in the first column.

A Simple MAK File Example

The simple MAK file example in Table 1-1 instructs XMake to convert the Viewlogic schematic select.1 into an XC4000 BIT file. If you to perform a manual translation, follow these steps:

1. Use WIR2XNF to convert the WIR file into an XNF file.
2. Use XNFMerge to create a flattened XNF file.
3. Use XNFPrep to perform design rule check and trim redundant logic.
4. Use PPR to convert the XNF file into an LCA file.
5. Use XDelay with the –d and –w options to add delay information to the LCA file
6. Use MakeBits to create a bitstream file for configuring the LCA.

Table 1-1 shows an extremely simpleMAK file that accomplishes these steps. The command syntax might vary depending upon the current profile.

Table 1-1 Contents of a MAK File

MAK File Contents
`select.bit : select.lca` `  makebits select.lca`

`select.lca : select.xtf`
`  ppr select.xtf parttype=4005pc84-5`
`  xdelay -D -W select.lca`

`select.xtf : select.xff` `  xnfprep select.xff select.xtf parttype=4005pc84-5`
`select.xff : select.xnf` `  xnfmerge -P 4005pc84-5 select.xnf select.xff`
`select.xnf : sch\select.1` `  wir2xnf -B select select.xnf`

When instructed to make select.bit for the first time, XMake automatically follows these steps (only the Viewlogic schematic file exists at first), starting from the first target entry at the top of the file.

1. Select.bit depends on select.lca which currently does not exist, so XMake searches the MAK file to see how to make select.lca.
2. Select.lca depends on select.xtf which currently does not exist, so XMake searches the MAK file to see how to make select.xtf.
3. Select.xtf depends on select.xff which currently does not exist, so XMake searches the MAK file to see how to make select.xff.
4. Select.xff depends on select.xnf which currently does not exist, so XMake searches the MAK file to see how to make select.xnf.
5. Select.xnf depends on select.1, which does exist, so XMake runs WIR2XNF, which creates select.xnf.

NOTE *Step 5 is slightly different, if you started with an OrCAD schematic. In this case, select.xnf depends on select.sch, which does exist, so XMake runs ANNOTATE, INET, and SDT2XNF, which creates select.xnf.*

6. Now that select.xnf exists, XMake makes select.xff by running XNFMERGE.
7. Now that select.xff exists, XMake makes select.xtf by running XNFPREP.
8. Now that select.xtf exists, XMake makes select.lca by running PPR.
9. To add delay information to the select.lca, XMake runs XDelay with the –d and –w options.
10. Now that select.lca exists, XMake makes select.bit by running the MakeBits program.

A Complete MAK File Example

An example of a complete XC4000-type MAK file is provided below for reference. The demo design is an XC4003 design that was created using Viewlogic.

```
# Created by XMAKE Version 5.0.0 on Mon Jan   3 18:44:42 1994
#
# The following options were used: -G
#
# The following is the hierarchy of the design 'sch\righton.1'
#
# sch/righton.1
#      statmach.abl
#      sch/rightled.1
#      sch/leftled.1
#      sch/points_b.1
#          xnf/bus_if04.xnf
#          xnf?bus_if04.xnf
#      sch/sw5.1
#      sch/8m2-1.1
#          xnf/m2-1.xnf
#          xnf/m2-1.xnf
#          xnf/m2-1.xnf
#          xnf/m2-1.xnf
#          xnf/m2-1.xnf
#          xnf/m2-1.xnf
#          xnf/m2-1.xnf
#          xnf/m2-1.xnf
#      xnf/compm8.xnf
#      sch/ledbar.1
#      sch/lights.1
#          memory.mem
#          xnf/c16bcr.xnf
#
DEFAULT_TARGET righton.bit

righton.bit : righton.lca
        makebits righton.lca

righton.lca : righton.xtf
        ppr righton.xtf parttype=4003PC84-5
        xdelay -D -w righton.lca

righton.xtf : righton.xg
        xnfprep righton.xg righton.xtf parttype=4003PC84-5

righton.xg : righton.xtg
        xblox righton.xtg righton.xg parttype=4003PC84-5

righton.xtg : righton.xff
        xnfprep righton.xff righton.xtg parttype=4003PC84-5

righton.xff : xnf/c16bcr.xnf xnf/memory.xnf xnf/lights.xnf \
xnf/ledbar.xnf xnf xnf/compm8.xnf xnf/m2-1.xnf xnf/8m2-1.xnf \
xnf/sw5.xnf xnf/bus_if04.xnf xnf/points_b.xnf xnf/leftled.xnf \
xnf/rightled.xnf xnf/statmach.xnf xnf/righton.xnf
     xnfmerge -D xnf -D . -P 4003PC84-5 xnf\righton.xnf righton.xff

xnf/righton.xnf : sch/lights.1 sch/ledbar.1 sch/8m2-1.1 sch/sw5.1 \
sch/points_b.1 sch/leftled.1 sch/rightled.1 sch/righton.1
     wir2xnf -B -OD xnf righton righton.xnf
```

```
xnf/statmach.xnf : statmach.abl
     abl2xnf statmach.abl output_directory=xnf family=XC4000 \
parttype=4003PC84-5

xnf/memory.xnf : memory.mem
     memgen memory.mem output_directory=xnf
```

Macros in the MAK File

XMake does not create macros in the MAK file. However, user-created MAK files can include user-defined macros in which the macro value replaces the macro name in any subsequent occurrences in the MAK file.

The syntax for defining a macro is as follows.

macro_name=value

The macro name must start in the first column; the value is the string between the first non-space character after the equal sign to the end of the line.

To use the macro, include the following string wherever you want the macro value inserted in the MAK file. If the macro name is more than one character, it must be enclosed in parentheses.

%(*macro_name*)

The following example shows how to use a part type macro when the part number is used in several programs.

```
NEWPART=3020PC68-100

rolldice.lca : rolldice.map
   map2lca -p%(NEWPART) rolldice.map rolldice.lca
   apr -w -y rolldice.lca rolldice.lca

rolldice.map : rolldice.xtf
   xnfmap -p%(NEWPART) rolldice.xtf rolldice.map

rolldice.xtf : rolldice.xff
   xnfprep rolldice.xff rolldice.xtf parttype=%(NEWPART)
```

Error Messages and Recovery Techniques

Command *command* **failed (rc=***n***),** *file* **removed.**

A fatal error occurred in a program called by XMake. Check your ***design***.out file for details and correct any errors.

Command *command* **not found,** *file* **removed.**

A program called by XMake was not found. Verify that the program is installed correctly, and that the required path and environment variables are specified correctly.

Command line length exceeds maximum allowed by the system (*n* **chars). Cmd=***command***.**

The command line is too long. The maximum characters allowed is 127 characters on a PC, 2047 on a Sun, and 511 on an Apollo.

Corrupted speeds file *file***.**

XMake found a corrupted speeds file. Reinstall the relevant data files to obtain the correct speeds files.

design **Design must not have a preceeding path name.**

XMake must be run on files in the current directory. Change directories to the directory containing the design, and rerun XMake on the design name without the preceding pathname.

Encryption checksum failed for speeds file *file***.**

XMake failed to validate the reported encrypted speeds file. The speeds file might be corrupted. Reinstall the relevant data files to obtain the correct speeds files.

Expanded line length exceeds maximum allowed by the program (1024 characters).

The MAK file contains a line with an expanded XMake macro (not a design macro) that is too long. The maximum characters allowed in an expanded line is 1024.

Failed to find part type *part_type* **in 'partlist.xct'.**

Specified part type is not available. Select another, valid part type.

Failed to find speed grade *speed* **for part** *parttype* **in part list.**

XMake was unable to find the specified speed grade for the specified part type. Make sure you have specified a valid speed grade.

Failed to find user-defined subhierarchy *symbol* **in** *file*.

While scanning an XNF file, XMake found a user-defined symbol with no corresponding schematic or XNF file. Check the design and ensure there is a design file for this symbol in the current directory. For user–created libraries in Viewlogic, the library components must have a LEVEL=MXILINX attribute.

Failed to make *file*.

XMake failed to produce the specified file. Check the *design*.out file for error details.

Failed to open *file*.

XMake could not open the specified file. This could be a system environment problem. On a PC, check in the config.sys file to see that the 'FILES=' command is present and set to a value of at least 20.

Failed to copy stderr handle to term.

System error, which is not recoverable. There are no known recovery techniques. XMake cannot be executed on this machine. Manually run the appropriate design implementation tools.

Failed to force term to be unbuffered.

System error, which is not recoverable. There are no known recovery techniques. XMake cannot be executed on this machine. Manually run the appropriate design implementation tools.

Failed to join stdout and outfile.

Due to a system error, XMake cannot redirect the output message from each tool into a *design*.out file. Use XMake with the –o option. This enables output messages to display on the screen.

Failed to reopen *file*.

Due to a system error, XMake cannot redirect the output message from each tool into a *design*.out file. Use XMake with the –o option. This enables output messages to display on the screen.

Failed to join stderr and outfile.

Due to a system error, XMake cannot redirect the output message from each tool into a *design*.out file. Use XMake with the –o option. This enables output messages to display on the screen.

Failed to touch *file*.

The attempt to update the time stamp failed on the reported file. If the file is protected against such changes, remove the protection and rerun XMake.

family **not supported by this program.**

XMake supports the following device families:
XC2000/XC2000L/XC3000/XC3100
XC3000A/XC3000L/XC3100A
XC4000/XC4000A/XC4000H

For XC7200 and XC7300 run XEMAKE.

In 'partlist.xct'. Missing alias name.

In 'partlist.xct'. Missing aliased-to part.

In 'partlist.xct'. Missing device name.

In 'partlist.xct'. Missing STYLE.

In 'partlist.xct'. Missing SPEEDFILE.

In 'partlist.xct'. Unknown STYLE *style*.

In 'partlist.xct'. Unknown aliased-to part *part_type*.

These seven messages indicates that the partlist.xct file is corrupted. You can obtain the correct partlist.xct file by reinstalling partlist.xct.

In speeds file *file*. **Expecting 'VERSION", found** *string*.

In speeds file *file*. **Missing format.**

In speeds file *file*. **Expecting format** *format*, **found** *string*.

These three messages indicates that the reported speeds file is corrupted. Reinstall the relevant data files to obtain the correct speeds files.

Invalid argument *argument*.

The reported argument violates XMake command line syntax. Refer to the Using XMake from the System Prompt section in this chapter for the valid XMake syntax.

Invalid option *option*.

The reported option is not a valid XMake option. Refer to the Options information for valid XMake options.

Invalid speed grade *speed_grade* **for part** *part-type*.

Specified speed grade is not available for the part. Select another valid speed grade for the desired part.

Item length exceeds maximum allowed by the program (64 chars).

The MAK file contains an item, such as file name or macro value that is too long. The maximum characters allowed in a MAK file is 64.

Macro name length exceeds maximum allowed by the program (64 chars).

The MAK file contains a line with an XMake macro that is too long. The characters allowed for a macro name in a MAK file is 64.

Mismatch between 'partlist.xct' and *file*.

The partlist.xct file and the reported speeds file are incompatible. The speeds file might be corrupted. Reinstall the relevant data files to obtain the correct speeds files.

Missing design name.

The command line is missing an input design name or a MAK file name. Check the syntax and correct the error.

Missing macro name to follow macro modifier '%'.

While reading the MAK file, XMake encountered a macro modifier % without a macro name to expand. Refer to the Macro information in this chapter and edit the MAK file to correct the error.

Missing ')' to complete macro name macro_name.

While reading the MAK file, XMake found a macro name to be expanded which consisted of more than one character (indicated by the opening parenthesis), but failed to find the closing parenthesis. Refer to

the Macro information in this chapter and edit the MAK file to correct the error.

Missing part type to follow option '-P'.

The –p option was specified on the command line without a part type. Select a valid part type.

Not allowed to read unencrypted speeds file *file*.

XMake is only allowed to read the reported speeds file in the encrypted form.

Not enough memory to execute command *command*, *file* **removed.**

There is insufficient memory to execute the command. Make additional memory available by removing any TSRs or, rerun XMake from the command line instead of from within XDM.

Number of hierarchy levels exceeds maximum allowed by the program (100 levels).

The design has too many levels of hierarchy. The maximum number of levels allowed by XMake is 100. There is no workaround, except to reduce the levels of hierarchy to meet the program requirements.

Options '-A', '-F' and '-T' cannot be specified together.

Option –a (Use 'map-then-merge' strategy), –f (Use 'map-FILE=-then-merge' strategy), and option –t (Use 'merge-then-map' strategy) are mutually exclusive. Select the most appropriate strategy and specify the corresponding option on the command line.

Original line length exceeds maximum allowed by the program (1024 chars).

The MAK file contains a line that is too long. The maximum characters allowed for a line in a MAK file is 1024.

Out of memory. Needed *n* **objects of** *n* **bytes.**

There is insufficient memory for XMake to continue execution. Make additional memory available by removing any TSRs, and rerun XMake from the command line instead of from within XDM.

Part type not specified on the command line or in the top level XNF file.

The attempt to get the part type failed. XMake finds the part type from the command line if you selected the –p option or, from the top level design file (if none is specified on the command line). This message is issued only if you selected the –x option on the command line, and the top level design file is an XNF file.

Party type not specified on the command line or in the top level schematic file.

The attempt to get the part type failed. XMake finds the part type from the command line if you selected the –p option or, from the top level design file (if none is specified on the command line.) This message is issued only if you did not select the –x option on the command line and the top level design file is a schematic file.

Recursive reference made to *file*.

XMake found a recursive loop in the schematic design or in the MAK file. Either there is a schematic functional block that contains itself, or there is an error in the MAK file made during editing. Check either the design or the MAK file.

Syntax error in *file*. **'=' expected following FILE parameter.**

Syntax error in *file*. **'=' expected following MAP parameter.**

Syntax error in *file*. **'=' expected following DEF parameter.**

These three messages indicates that the reported XNF file is corrupted. Use XMake with the –r option, if the schematic-to-XNF translation for the design is directly supported by XMake (Viewlogic or OrCAD); or, manually regenerate the XNF file, and use XMake with the –x option.

Target *target* **not found in makefile** *file*.

XMake failed to find the specified target file name in the MAK file. Target file name must match one of the output files generated by the programs called by XMake such as *design*.xtf, *design*.lca, etc.

Top level XNF file for design *design* **does not exist.**

XMake failed to find the top-level design file, *design*.xnf. Make sure you are in the directory that contains the specified design files. This

message is issued only if you selected the –x option from the command line in which case the top-level design file must be an XNF file.

Top level schematic file for design *design* **does not exist.**

XMake failed to find the top level schematic file for the specified design. Make sure you are in the directory that contains the specified design files. This message is issued if you did not select the –x option on the command line in which case the top-level design file must be a schematic file.

Unable to create subdirectory *name*.

XMake failed in an attempt to create a subdirectory *name* in the current directory. Check for a pre-existing file or directory by that name and remove it from the current directory.

Unable to create subdirectory *name*. **File** *name* **exists.**

XMake failed in an attempt to create a subdirectory *name* in the current directory, because such a subdirectory already exists. Remove the existing *name* file and rerun XMake.

Unable to write to *file*. **Disk full.**

There is insufficient disk space for XMake to write to the reported file. Provide additional disk space and rerun XMake.

Unable to write to subdirectory *name*.

The subdirectory *name* is write-protected. Remove the protection and rerun XMake.

Undefined macro *macro_name*.

While reading the MAK file, XMake tried to expand a macro name but found it to be undefined. Edit the MAK file and correct the error.

'xdm.pro' contains the following invalid program options.

This error message can be issued if you are running XMake at the system prompt (not from within XDM), and there is an old 'xdm.pro' file in the current directory, which contains program options that are obsolete or invalid. Run XDM to generate a correctly customized 'xdm.pro' and try again.

Warning Messages and Recovery Techniques

Option *-option* **is ignored when using makefile.**

Options –a –b, –f, –g, –i, –l, –m, –n, –p, –t, and –x only control the MAK file generation phase of the XMake program, and are ignored by XMake if a MAK file is specified as input.

Option '-M' is redundant when '-N' is specified.

Option –n implies that the –m option has also been selected.

Option *-option* **is ignored when target is specified.**

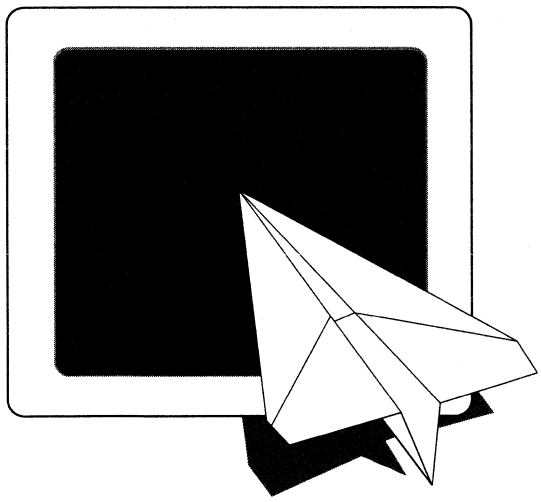
XMake interprets the –m and –n options to set the target appropriately in the absence of the target field on the Command Line. Both options are ignored if the target is explicitly stated on the command line.

Option *-option* **is ignored when using** *family* **part.**

Options –a, –f, and –t apply only to the following device families:
XC2000/XC2000L/XC3100/XC3000
XC3000A/XC3000L/XC3100A
Option –b applies only to the following device families:
XC3000A/XC3000L/XC3100A
XC4000/XC4000H/XC4000A

No speed grade specified for part *part_type*. **Default** *speed_grade* **will be used.**

If a valid part type is selected without speed grade, XMake uses the default speed grade for the part.



The MemGen Program

XACT Reference Guide, Volume 1

XACT Reference Guide, Volume 1 — April, 1994 (0401129 01)

Printed in U.S.A.

The MemGen Program

This Program is Compatible with the Families Indicated.

☐ XC2000	☐ XC3100	☐ XC3100A	☑ XC4000H
☐ XC2000L	☐ XC3000A	☑ XC4000	☐ XC7200
☐ XC3000	☐ XC3000L	☑ XC4000A	☐ XC7300

MemGen creates RAMs or ROMs that can be from to 1 to 32 bits wide and up to 256 words deep. It generates an XNF file for the specified memory, and optionally creates a schematic symbol to represent it. It creates address-boundary-checking logic for memories with depths that are not powers of two.

Syntax

MemGen accepts a memory definition (MEM) file to describe a memory. The syntax for executing MemGen is shown below.

memgen *filename* **[.mem]** **[***options***]**

Files

MemGen requires one input file and generates several output files as described below.

Input Files

filename.mem

The input file must have a MEM extension, but you do not need to specify the extension on the command line, since MEM files are recognized when MemGen is run. See the Memory Definition File section for more information.

If the specified file does not exist, MemGen prompts you for the memory type, its width and depth, and the type of symbol to generate. If given a new file name, you can use MemGen interactively to completely define a RAM and its symbol. Where applicable, you can interactively define a ROM except for its initialization value and bus representation style, which must be defined in the MEM file.

Output Files

filename.xnf

This output file is an XNF file that contains a description of the memory specified in the MEM file.

filename.cmd

This file is the command file that MemGen creates when you specify the – o option. This command file is used within the OrCAD LibEdit program to create an OrCAD/SDT symbol for the specified memory.

filename.1

This file is the Viewdraw symbol file that MemGen creates when you specify the –v option. The symbol file is placed in the /sym subdirectory of the current project directory.

memgen.log

This is the log file that contains all the information displayed on the screen during MemGen execution. You can specify a different log file name by using the Logfile parameter. Refer to the Options and Parameters section for a description of the Logfile parameter.

Memory Definition File

A memory definition file, or MEM file, defines a memory and its contents to MemGen. A description of the commands used in this file follows the example. An example of a MEM file is shown here.

```
;=========================================================
; EXAMPLE.mem: A 256-word deep by 16-bit wide ROM memory.
;=========================================================
TYPE ROM    ; The memory is a ROM
DEPTH 256   ; The memory is 256 words deep
WIDTH 16    ; Each memory word is 16 bits wide
SYMBOL VIEWLOGIC BUS ; Build a VIEWLOGIC symbol with bus
        ; inputs ;
DEFAULT FFFF ; Each unspecified location is set to HIGH

DATA 2#1111_0000_1111_0000#,   ; Binary data
   8#177777#,     ; Octal data
   10#23#,        ; Decimal data
   16#4a#,        ; Hexadecimal data
   4f             ; Unspecified base assumed
                  ; to be hexadecimal
```

Specifying Memory Characteristics

A MEM file consists of a series of commands that specify the dimensions and contents of a memory. The commands used in a MEM file include Type, Depth, Width, Symbol, Default, and Data (see the example above). With the exception of the Data and Default commands, you can use parameters to specify all memory characteristics listed in the MEM file. See the Options and Parameters section for more information.

Type

The Type command defines the type of memory to build. RAMs are read-write memories; ROMs are read-only memories.

`type [RAM|ROM]`

Depth

The Depth command defines the depth, in words, of the memory. Each word is "*width*" bits wide. Specify the memory depth with a positive decimal integer value between 2 and 256, inclusive.

`depth` *memory_depth*

Any memory-depth value less than 2 or greater than 256 is flagged as an error.

Width

The Width command defines the width of the memory, which is the number of bits in each word. Specify the memory width with a positive integer between 1 and 32, inclusive

`width` *memory_width*

Any memory-width value larger than 32 is flagged as an error.

Symbol

The Symbol command specifies the schematic editor for which MemGen should create the memory symbol. It also defines the format of the data and address lines, which are created as either individual pins or as bus signals.

`symbol` *editor* **`[bus|pins]`**

The *editor* is one of the following: Viewlogic, OrCAD, or None.

The first field of the Symbol command defines the type of schematic editor. The second field specifies the format for address and data lines, either bus (BUS) or individual pins (PINS). Since the OrCAD/SDT schematic editor does not support the creation of bus pins on symbols, you can only specify the PINS selection for an OrCAD symbol.

If no symbol is to be created, specify None for the Symbol command, or omit the Symbol command from the MEM file.

The Symbol command directs MemGen to generate a new symbol or command file every time it is run. You can prohibit MemGen from generating a new symbol or command file by removing the Symbol command from the .mem file. The XNF file will still be generated if the Symbol command is removed.

NOTE *If a memory module is to be used in an unsupported schematic editor, you must manually create a symbol. MemGen supports only Viewlogic, and OrCAD schematic editors.*

Default

The Default command defines the value of any ROM location not specified by the Data command. If no default value is specified, all unspecified locations are zeros. Use the Default command to fill unspecified locations with a value other than zero. The Default command is not permitted for RAMs, since initial values are not supported on XC4000 RAMs.

`default` *datavalue*

You can write the specified *datavalue* in binary, octal, decimal, or hexadecimal. See the Data Formats section later in this chapter for more details.

Data

The Data command specifies the complete contents of a ROM.

`data` *datavalue1*, *datavalue2*, ... *datavaluen*

When used, the data specification must be the last command in a MEM file. The Data command is not permitted for RAMs, since initial values are not supported on XC4000 RAMs.

NOTE *The Data command can be specified over multiple lines, although the data keyword is used only once. Individual data values must be separated by commas or by blank characters, such as spaces, carriage returns, or tabs.*

You can write the specified data values in binary, octal, decimal, or hexadecimal. See the Data Formats section later in this chapter for more details. The first *datavalue* is location zero; the next *datavalue* is location one, and so forth.

Comments

`;` *comment_strings*

MemGen ignores all text to the right of a semicolon until the end of the line.

Data Formats

The data values specified in either the Default or Data commands can be binary, octal, decimal, or hexadecimal. Hexadecimal is the default numeric base for data values. For the other values, precede the data with the appropriate numeric base in the following format.

base#*value*#

- *base* — The base is a decimal number and must be either 2 (binary), 8 (octal), 10 (decimal), or the default, 16 (hexadecimal). The default need not be specified.
- *value* — The data value must contain only characters valid for the specified numeric base. Table 1-2 defines valid characters. You can use an underscore character (_), which MemGen ignores, to format data within a field.

For example, the following value is difficult to read.

```
2#0001001001110010011001111011010001#
```

Using the underscore character, data fields are easier to read, as seen below.

```
2#0001_0010_0111_0010_0110_0111_0110_1001#
```

Table 1-2 Various Numerical Bases and Their Valid Characters

Base Type	Base	Valid Characters
Binary	2	0 1 _
Octal	8	0 1 2 3 4 5 6 7 _
Decimal	10	0 1 2 3 4 5 6 7 8 9 _
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 _ a b c d e f A B C D E F

The following example shows how to express the decimal value 17.

```
2#10001#     (binary)
8#21#        (octal)
10#17#       (decimal)
16#11#       (hexadecimal)
```

Options

The following options and parameters control the type of memory and schematic symbol that MemGen creates. With the exception of the Logfile parameter, you can specify all options from the MEM file.

memory_depth= Number of Words in Memory

The memory_depth= parameter specifies the number of words in the memory. The depth must be a number between 2 and 256, inclusive.

parttype= Target LCA Device

The parttype= parameter specifies the target LCA device part. This parameter defines the PART record in the output XNF file that MemGen creates. If you do not specify a part type, MemGen uses 4005PG156 as the default.

NOTE *The part type you specify should be a valid XC4000 device.*

type= Memory Type

The type= parameter specifies either a ROM or a RAM.

word_width= Number of Bits per Memory Word

The word width= parameter specifies the number of bits in each memory word. The word width must be between 1 and 32, inclusive.

–b Bus Notation

If you enable the –b option, MemGen uses bus notation for data and address lines. The bus naming convention that MemGen uses for the symbol and the XNF file is determined by the specified schematic editor. Therefore, it is important to specify the correct schematic-editor option when using this option, otherwise the symbol does not bind correctly.

NOTE *This parameter does not apply to OrCAD, which does not support bus notation or the symbols.*

–o OrCAD/SDT Symbol

If you use the –o option, MemGen creates a memory symbol for the OrCAD/SDT schematic editor. The symbol description stored in *filename*.cmd is an import file for the OrCAD LibEdit symbol editor program.

–v Viewlogic Viewdraw Symbol

If you enable the –v option, MemGen creates a symbol file for the memory symbol in the Viewlogic Viewdraw schematic editor. The Symbol file (filename.1) is placed in the /sym subdirectory of the current project directory.

logfile MemGen Log File Name

By default, the screen output from MemGen is stored in a file called memgen.log. You can specify an alternate file name with this parameter.

`logfile=` *filename.extension*

If you do not specify a file name extension, MemGen uses the default extension LOG. If the Logfile parameter is not used, MemGen writes the screen output to the file memgen.log, overwriting any previous versions of this LOG file.

old_library= Output XNF File for Old (non-Unified) Libraries

Set the old_library= parameter to TRUE if the output XNF file is to be included in a design that was created from a schematic symbol library that existed prior to the Xilinx Unified Library. Designs for the "old" libraries do not include the new LIBVER parameter on the symbols. Designs from the Unified Library are distinguished by the presence of the LIBVER= parameter on the symbols. If old_library is set to TRUE, the LIBVER= parameter is omitted from the output design so the XNF file is compatible with the old library.

The default value is old_library=FALSE. The default generates an XNF file that includes LIBVER parameters on the symbols so that the file is compatible with the Unified Library.

IMPORTANT

Xilinx requires all files in each completed design to be consistent with either the Unified Library or an old library. If your design is drawn using the Unified Library leave the old_library= parameter set to the default of FALSE. If your design is not from the Unified Library, set the old_library= parameter to TRUE.

output_directory= Set Output Directory

Set the output_directory= parameter to a directory name if you want the XNF file put in a directory other than the current directory. The other MemGen-generated files are not affected by the output_directory= parameter setting. The default is to put the XNF file into the current directory.

Examples

This example shows how to use MemGen to create an XNF file and a Viewlogic memory symbol from a memory definition file called mymemory.mem, using the bus notation

```
memgen mymemory -v -b
```

This sytnax creates a file called mymemory.1 that is placed in the /sym subdirectory of the current project directory. If a MEM file that already contains information that contradicts the command line arguments exists, the MEM file information overrides the command line. For example, if an existing mymemory.mem is specified in the ORCAD editor, the –v option for the Viewlogic editor would be ignored. In this case, if the Viewlogic editor was desired, you would have to either edit the mymemory.mem file or create a new MEM file.

Address Boundary Checking

The MemGen memory compiler automatically includes logic to check address boundaries for any memory where the depth is not a power of two. In this case, MemGen adds an active-High output called ERR, or ADDR-ERR if bus notation has been specified for the memory.

An example of automatic address boundary checking is shown in Figure 1-7. Four address lines are required to access the 12 memory locations. However, using 4 address lines, 16 address values are possible. Since only 12 locations are valid for this example, there are 4 illegal or invalid addresses. If any of these four locations are addressed, the ERR output would go High, indicating that the address inputs are attempting to address an invalid location.

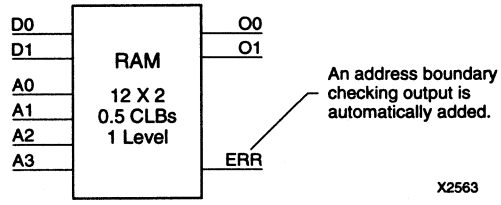
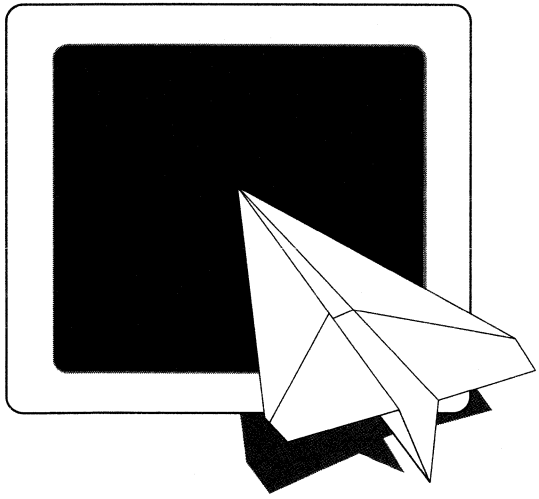


Figure 1-7 Address Boundary Checking

The ERR or ADDR-ERR output is not created for memories that have depths that are a power of 2 (for example, memories that are 8, 16, or 32 words deep).

PPR automatically removes all unused extra logic created for address boundary checking from the design.



XACT Reference Guide, Volume 1

XACT-Performance Utility

XACT Reference Guide, Volume 1 — April, 1994 (0401129 01)

Printed in U.S.A.

XACT-Performance Utility

This Program is Compatible with the Families Indicated.

☐ XC2000	☐ XC3100	☑ XC3100A	☑ XC4000H
☐ XC2000L	☑ XC3000A	☑ XC4000	☐ XC7200
☐ XC3000	☑ XC3000L	☑ XC4000A	☐ XC7300

XACT-Performance™ enables you to specify precise timing requirements for your Xilinx FPGA designs. Use XACT-Performance to specify the maximum allowable delay on any given set of paths in your design. You identify a set of paths by identifying a group of start and end points. The start and end points can be flip-flops, I/O pads, IOB latches, or XC4000 RAMs. You can control the worst-case timing on the set of paths by specifying a single delay requirement for all paths in the set.

The primary method of specifying timing requirements involves entering them on the schematic. However, you can also specify timing requirements via the constraints file as well as PPR command-line options. These command-line options do not provide as much control and flexibility as entering timing information directly on the schematic. See the "PPR" chapter in the *XACT Reference Guide, Volume 2* for more information about PPR command-line options. For detailed information about the constraints you can use with your schematic-entry software, refer to the *XACT Libraries Guide.*

Once you define timing specifications, PPR maps, places, and routes your design based on these requirements.

To analyze the results of your timing specifications use the XDelay program. Refer to "The XDelay Timing Analysis Program" in the *XACT Reference Guide, Volume 3* for more information.

This chapter covers the following topics:

- Defining timing requirements using groups
- Defining timing requirements using path-type specifications

WARNING!!! *Although you can use end-point specifications (using groups) in the same design with existing path-type specifications, Xilinx discourages mixing the two methods. If you are modifying an existing design that uses path-type timing specifications, refer to the "Defining Timing Requirements Using Path-Type Specifications" section in the middle of this chapter.*

Defining Timing Requirements Using Groups

You can specify timing requirements by specifying a set of paths and the maximum allowable delay on these paths. You specify the set of paths by grouping start and end points in one of the following ways:

- You can refer to a predefined group by specifying one of the corresponding keywords — FFS, PADS, LATCHES, or RAMS.
- You can create arbitrary groups within a predefined group by tagging symbols with TNM (pronounced *tee-name*) attributes.
- You can create groups that are combinations of existing groups using TIMEGRP symbols.
- You can create groups by pattern matching on signal names.

The following sections discuss each method in detail.

Understanding the Basics

This section introduces the following concepts that you need to know to begin using XACT-Performance:

- TIMESPEC Primitive
- From-To Statement Syntax

TIMESPEC Primitive

The TIMESPEC primitive, as illustrated in Figure 1-8, serves as a *placeholder* for timing specifications, which are called TS attribute definitions. Every TS attribute must be defined in a TIMESPEC primitive. Every TS attribute begins with the letters "TS" and ends with a unique identifier that can consist of letters, numbers, or the underscore character (_).

OrCAD Users — The implementation of XACT-Performance described in this chapter differs slightly from the OrCAD implementation; for example, the TIMESPEC primitive does not exist in the Xilinx OrCAD library. For more details, refer to the *OrCAD Interface User Guide*.

Mentor Users — The term *attribute* in this chapter is equivalent to *property* as used in the Mentor environment.

The TIMESPEC primitive is 30 characters wide; however, you can create TS attribute definitions of any length. Each TIMESPEC primitive can hold up to eight TS attributes. If you want to include more than eight TS attributes, you can use multiple TIMESPEC primitives in your schematic.

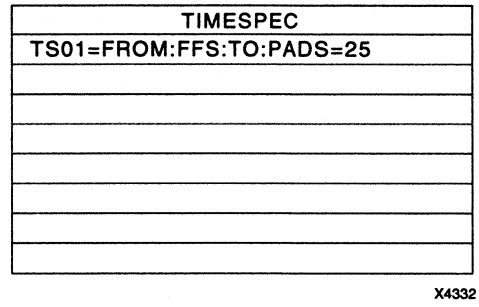


Figure 1-8 TIMESPEC Primitive

How you add a TIMESPEC primitive to your schematic depends on your specific schematic-entry software. Refer to the appropriate Xilinx Interface User Guide for step-by-step instructions.

From-To Statement Syntax

You can use From-To statements to specify timing requirements between specific end points by using the following syntax within the TIMESPEC primitive:

TS*identifier*=**FROM:***group1***:TO:***group2*=*delay*

From-To statements are TS attributes that reside in the TIMESPEC primitive. The parameters *group1* and *group2* must be predefined groups, previously created TNM identifiers, or groups defined in TIMEGRP symbols. Predefined groups consist of FFS, LATCHES, RAMS, or PADS and are discussed in the "Using Predefined Groups" section. TNMs are introduced in the "Creating Arbitrary Groups Using TNMs" section. TIMEGRP symbols are introduced in the "Creating New Groups from Existing Groups" section.

NOTE *Keywords, such as FROM and TO, appear in the documentation in upper case; however, you can enter them in the TIMESPEC primitive in either upper or lower case.*

The *delay* parameter defines the maximum delay for the attribute. Nanoseconds are the default units for specifying delay time in TS attributes. You can also specify delay using other methods, including megahertz, which are described in "Time Delay Specifications in TS Attributes" section at the end of this chapter.

The following examples illustrate the use of From-To TS attributes:

```
TS01=FROM:FFS:TO:FFS=30
TS_OTHER=FROM:PADS:TO:FFS=25
TS_THIS=FROM:FFS:TO:RAMS=35
TS_THAT=FROM:PADS:TO:LATCHES=35
```

NOTE *Latches refer to input latches (INLATs or ILDs) only. Using From-To syntax is the only way you can specify timing requirements for a RAM.*

You can place TS attributes containing From-To statements in either of two places: in the TIMESPEC primitive on the schematic as discussed in this chapter or in the PPR constraints (CST) file. See the *XACT Libraries Guide* for more information about specifying timing requirements in the constraints file.

You can also define timing requirements by creating groups using TNMs, creating TIMEGRP attributes, and creating groups by pattern matching. These methods are discussed in the following sections.

Using Predefined Groups

You can refer to a group of flip-flops, input latches, pads, or RAMs by using the corresponding keywords:

FFS	CLB or IOB flip-flops
LATCHES	input latches only; not latches built from function generators
PADS	input/output pads
RAMS	for the XC4000 family only

XACT-Performance From-To statements enable you to define timing specifications for paths between predefined groups. The following examples are TS attributes that reside in the TIMESPEC primitive. This method enables you to easily define default timing specifications for the design, as illustrated by the following examples:

```
TS01=FROM:FFS:TO:FFS=30
TS02=FROM:LATCHES:TO:LATCHES=25
TS03=FROM:PADS:TO:RAMS=70
TS04=FROM:FFS:TO:PADS=55
```

As used in the previous example, the predefined groups represent all symbols of that type. To create more specific groups see the next section, "Creating Arbitrary Groups Using TNMs."

Creating Arbitrary Groups Using TNMs

A TNM (timing name) is a flag that you place directly on your schematic to tag specific pads, flip-flops, input latches, or RAMs. All symbols tagged with the TNM identifier are considered a group. Place TNM attributes directly on your schematic using the following syntax:

TNM=*identifier*

where *identifier* is a value that consists of any combination of letters, numbers, or underscores. Keep the TNM short for convenience and clarity.

WARNING!!! *Do not use reserved words, such as FFS, LATCHES, RAMS, or PADS, for TNM identifiers.*

You can specify as many groups of end points as is necessary to describe the performance requirements of your design. However, to simplify the specification process and reduce the place-and-route time, use as few groups as possible.

You can use several methods for tagging groups of end points — placing identifiers on primitive symbols, macro symbols, nets, or load pins. Which method you choose depends on how the path end points are related in your design. The following subsections discuss the different methods.

Placing TNMs on Primitive Symbols

You can group individual logic primitives explicitly by flagging each symbol, as illustrated by the following figure:

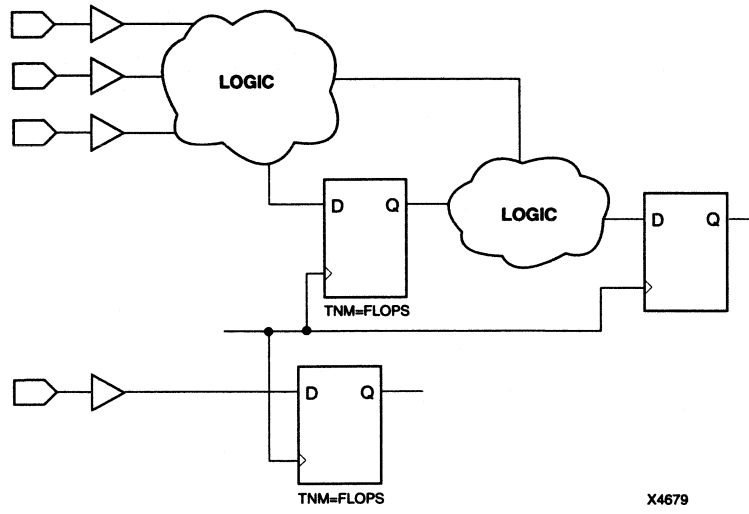


Figure 1-9 TNM on Primitive Symbols

In Figure 1-9, the flip-flops tagged with the TNM form a group called "FLOPS." The untagged flip-flop is not part of the group.

NOTE *You cannot use TNMs to group instances of incompatible symbols; for example, it is incorrect to tag a pad and a flip-flop with the same TNM. The only compatible predefined groups are flip-flops and input latches, on which you can use the same TNM.*

Place only *one* TNM on each symbol, load pin, or macro load pin. If you want to assign more than one identifier to the same symbol, include all identifiers on the right side of the equal sign (=) separated by a semicolon (;), as follows:

```
TNM=joe;fred
```

Placing TNMs on Macro Symbols

When a macro contains only one symbol type, you can place a TNM directly on the macro. If the macro contains only flip-flops, all the flip-flips are identified by the given TNM.

When a macro contains more than one symbol type, use the TNM identifier in conjunction with one of the predefined groups: FFS, RAMS, PADS, or LATCHES as indicated by the following syntax examples:

TNM=FFS: *identifier*
TNM=RAMS: *identifier*
TNM=LATCHES: *identifier*
TNM=PADS: *identifier*

If you want to place an identifier on more than one symbol type, separate each symbol type and identifier with a semicolon (;) as illustrated by the following examples:

```
TNM=FFS:FLOPS;PADS:OPADS
TNM=RAMS:MEMS;LATCHES:INLATS
```

For example, if multiple flip-flops are contained in the same hierarchical block, you can simply flag that hierarchical symbol, as illustrated by the following figure:

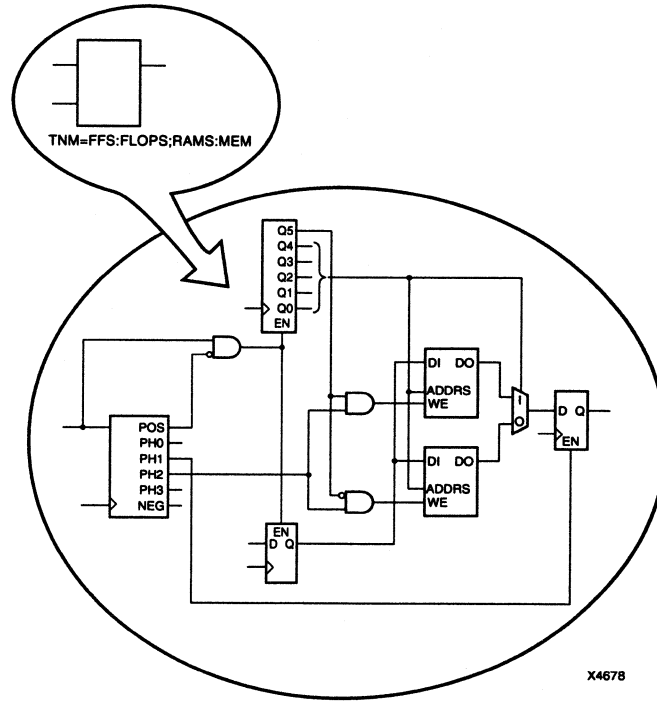


Figure 1-10 TNM on Macro Symbol

In Figure 1-10, all flip-flops included in the macro are tagged with the TNM "FLOPS" and all RAMs are tagged with the TNM "MEM." By tagging the macro symbol, you do not have to tag each underlying symbol individually.

Placing TNMs on Signals or Pins to Group Flip-Flops

You can easily group flip-flops by flagging a common input signal, typically either a clock signal or an enable signal. If you attach a TNM to a signal or load pin, that TNM applies to all flip-flops and/or input latches that are reached through the signal or pin. That is, PPR traces forward on that path, through any number of gates or buffers, until it reaches a flip-flop or input latch. PPR adds that element to the specified TNM group. This mechanism is called forward tracing.

Placing a TNM on a signal is equivalent to placing that TNM attribute on every load pin of the signal. Use pin TNM attributes when you need finer control.

The following figure illustrates the use of a TNM on a net that traces forward to create a group of flip-flops:

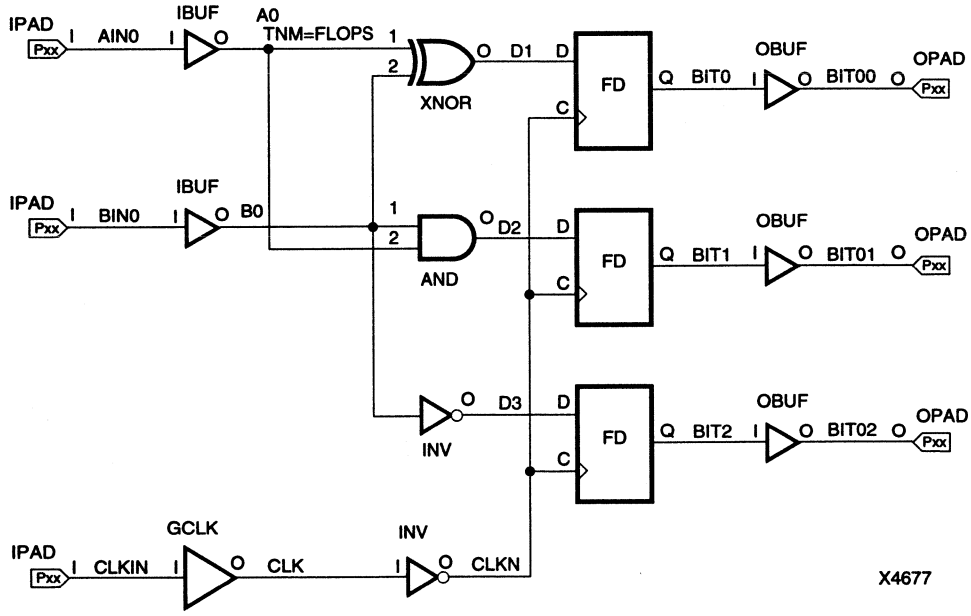


Figure 1-11 TNM on Signal Used to Group Flip-Flops

In Figure 1-11, the TNM traces forward to the first two flip-flops, which form a group called FLOPS. The bottom flip-flop is not part of the group FLOPS.

The following figure illustrates placing a TNM on a clock pin, which traces forward to all three flip-flops and forms the group Q_FLOPS:

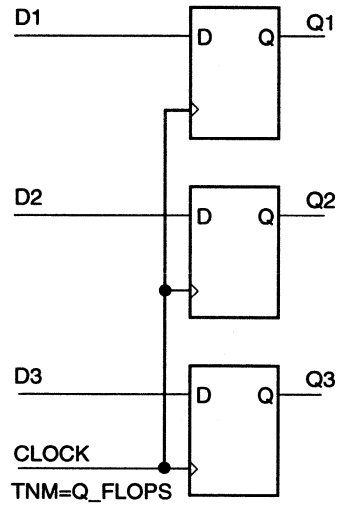


Figure 1-12 TNM on Clock Pin Used to Group Flip-Flops

Creating New Groups from Existing Groups

In addition to naming groups using the TNM identifier, you can also define groups in terms of other groups. You can create a group that is a combination of existing groups by defining a TIMEGRP attribute as follows:

newgroup=existing_grp1:*existing_grp2* [:*existing_grp3* . . .]

where *newgroup* is a newly created group that consists of existing groups created via TNMs, predefined groups, or other TIMEGRP attributes.

Mentor Users — You must specify a leading equal sign (=) when defining TIMEGRP attributes, for example, *=newgroup*. The preceding equal sign lets Mentor know that this is a user-defined attribute. Refer to the *Mentor Interface User Guide* for more information.

TIMEGRP attributes reside in the TIMEGRP primitive, as illustrated in Figure 1-13. Once you create a TIMEGRP attribute definition within a TIMEGRP primitive, you can use it in the TIMESPEC primitive. Each TIMEGRP primitive can hold up to eight group definitions. Since your design might include more than eight TIMEGRP attributes, you can use multiple TIMEGRP primitives.

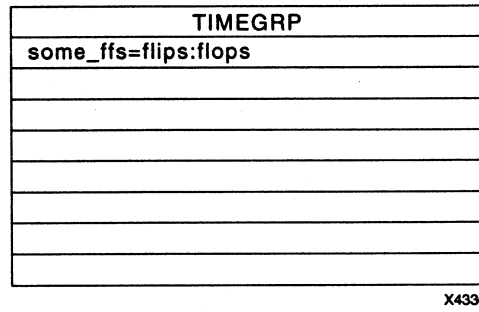


Figure 1-13 TIMEGRP Primitive

You can place TIMEGRP attributes in either of two places: in the TIMEGRP primitive on the schematic as discussed in this section or in the constraints (CST) file. See the *XACT Libraries Guide* for more information about specifying timing requirements in the constraints file.

You can use TIMEGRP attributes to create groups using the following methods:

- Combining multiple groups into one
- Creating groups by exclusion
- Defining flip-flop subgroups by clock sense

The following subsections discuss each method in detail.

Combining Multiple Groups into One

You can define a group by combining other groups. The following syntax example illustrates the simple combining of two groups:

```
big_group=small_group:medium_group
```

In this syntax example, small_group and medium_group are existing groups defined using a TNM or TIMEGRP attribute. Within the TIMEGRP primitive, TIMEGRP attributes can be listed in any order; that is, you can create a TIMEGRP attribute that references another TIMEGRP attribute that appears after the initial definition.

WARNING!!!

A circular definition, as shown below, causes an error.

```
many_ffs=ffs1:ffs2
ffs1=many_ffs:ffs3
```

Creating Groups by Exclusion

You can define a group that includes all elements of one group except the elements that belong to another group, as illustrated by the following syntax examples:

group1=*group2*:**EXCEPT**:*group3*

where *group1* represents the group being defined; *group2* and *group3* can be a valid TNM, predefined group, or TIMEGRP attribute.

As illustrated by the following example, you can specify multiple groups to include or exclude when creating the new group.

group1=*group2*:*group3*:**EXCEPT**:*group4*:*group5*

WARNING!!! *Do not use reserved words, such as FFS, PADS, RISING, FALLING, or EXCEPT, as group names or TNMs.*

Defining Flip-Flop Subgroups by Clock Sense

You can create subgroups using the keywords RISING and FALLING to group flip-flops triggered by rising and falling edges.

group1=**RISING:ffs**
group2=**RISING:***ffs_group*
group3=**FALLING:ffs**
group4=**FALLING:***ffs_group*

where *group1* to *group4* are new groups being defined. The *ffs_group* must be a group that includes only flip-flops.

NOTE *Keywords, such as EXCEPT, RISING, and FALLING, appear in the documentation in upper case; however, you can enter them in the TIMESPEC primitive in lower or upper case.*

The following example defines a group of flip-flops that switch on the falling edge of the clock.

```
falling_ffs=FALLING:ffs
```

Creating Groups by Pattern Matching

When creating groups, you can use wildcard characters to define groups of symbols whose associated signal names match a specific pattern:

group=predefined_group(pattern)

where *predefined_group* can only be one of the following predefined groups — FFS, LATCHES, PADS, or RAMS. A *pattern* is any string of characters used in conjunction with one or more wildcard characters.

WARNING!!! *When specifying a signal name, you must use its full hierarchical path name so PPR can find the signal in the flattened design.*

For flip-flops, input latches, and RAMs, specify the output signal name. For pads, specify the external signal name unless you placed a BLKNM or HBLKNM on the pad in the schematic; in this case, you should specify its value instead.

In XACT-Performance, any place you specify a predefined group, you can specify a predefined group qualified by a pattern as follows:

TS*identifier***=FROM:***group(pattern)***:TO:***group(pattern)***=***delay*

The following example illustrates creating a group that includes the flip-flops that source signals whose names begin with $1I3/FRED.

```
group1=ffs($1I3/FRED*)
```

The following example illustrates a group that excludes certain flip-flops whose output signal names match the specified pattern:

```
this_group=ffs:EXCEPT:ffs(a*)
```

In this example, this_group includes all flip-flops except those whose output signal names begin with the letter "a."

How to Use Wildcards to Specify Signal Names

The wildcard characters, * and ?, enable you to select a group of symbols whose output signal names match a specific string or pattern. The asterisk (*) represents any character string. The question mark (?) indicates a single character.

For example, DATA* indicates any signal name that begins with "DATA," such as DATA1, DATA22, DATABASE, and so on. The string NUMBER? specifies any signal names that begin with "NUMBER" and end with one single character, for example, NUMBER1, NUMBERS but not NUMBER12.

You can also specify more than one wildcard character. For example, *AT? specifies any signal names that begin with any series of characters followed by "AT" and end with any one character such as BAT1, CAT2, and THAT5.

How to Define Groups by Signal Name

This subsection gives you more examples of how to create groups by pattern matching. The following defines a group name "some_latches":

```
some_latches=latches($1I3/xyz*)
```

The group some_latches contains all input latches whose output signal names start with "$1I3/xyz."

Instead of specifying just one pattern, you can also specify a list of patterns separated by a colon (:) as illustrated below:

```
some_ffs=ffs(a*:b?:c*d)
```

The group "some_ffs" contains flip-flops whose output signal names:

- Start with the letter "a"
- Contain two characters; the first character is "b"
- Start with "c" and end with "d"

When Multiple Specifications Apply to the Same Path

When you apply more than one From-To specification to the paths between a given pair of end points, PPR analyzes all of them. In this case, PPR chooses the fastest one, which might not be the specification you want. For example, suppose you have the following timing requirements:

```
TS_ALL=FROM:PADS:TO:FFS:=30
TS_SLOWER=FROM:PADS:TO:SLOW_FFS=40
```

For TS_SLOWER to control the paths to SLOW_FFS, it would have to disable TS_ALL on these paths; however, no specification can disable another. Therefore, the TS_SLOWER paths are specified faster than intended. You can avoid this problem by either of two methods:

- Make sure your broad specifications are always the slowest ones. You could create a FAST_FFS group instead of a SLOW_FFS group, as follows:

  ```
  TS_ALL=FROM:PADS:TO:FFS=40
  TS_FASTER=FROM:PADS:TO:FAST_FFS=30
  ```

- Explicitly exclude slower paths from faster, more general specifications by using a TIMEGRP EXCEPT statement, as explained in the "Creating Groups by Exclusion" section.

  Using this method, you could create a TIMEGRP attribute to define the FAST_FFS group as a complement of the SLOW_FFS group as follows:

  ```
  FAST_FFS=FFS:EXCEPT:SLOW_FFS
  ```

  In the TIMESPEC primitive, you can define two non-overlapping specifications:

  ```
  TS_FASTER=FROM:PADS:TO:FAST_FFS=30
  TS_SLOWER=FROM:PADS:TO:SLOW_FFS=40
  ```

Ignoring Selected Paths

In a design, some paths do not require path analysis. These are paths that exist in the design, but are never used during time-critical operations. If you indicate a timing requirement on these paths, more

important paths might be slower, which can result in failure to meet the timing requirements. You can use IGNORE to disable any paths that you do not need to control by using the following syntax within the TIMESPEC primitive:

TS*id*=IGNORE

Attaching this TS flag to a net or load pin causes PPR to ignore any paths that include the net or load pin.

You cannot perform path analysis in the presence of combinational loops. Therefore, PPR ignores certain connections to break combinational loops. You can use IGNORE to direct PPR to ignore specified nets or load pins, consequently controlling how loops are broken.

Specifying Time Delay in TS Attributes

Nanoseconds are the default units for specifying delay times in TS attributes. However, after specifying the maximum delay or minimum frequency numerically, you can enter the unit of measure by specifying the following:

- NS for nanoseconds
- MHZ for megahertz
- US for microseconds
- KHZ for kilohertz

The XNFPrep program converts all units to nanoseconds and rounds them to 0.1 ns accuracy.

The following subsections discuss alternate methods of specifying time delay in TS attributes:

- Selecting Automatic Delay
- Specifying a TS Attribute Delay in Terms of Another

Selecting Automatic Delay (AUTO)

If you want to minimize delays along a certain group of paths but are unsure what delay to request, use AUTO in place of numerical values on the TS attributes as follows:

```
TS_PADS=FROM:PADS:TO:PADS=AUTO
```

PPR chooses a moderately aggressive path delay target for the specified paths and attempts to meet this target.

Specifying a TS Attribute Delay in Terms of Another

Instead of specifying a time or frequency in a TS attribute definition, you can specify a multiple or division of another TS attribute. This is useful in a system where all clocks are derived from a master clock; in this situation, changing the timing specification for the master clock changes the specification for all clocks in the system.

Use the syntax below to specify a TS attribute delay in terms of another.

TS*identifier*=*specification*: *reference_TS_attribute* [* , /] *number*

where *number* can be either a whole number or a decimal. The *specification* can be any From-To statement as illustrated by the following examples:

```
FROM:PADS:TO:PADS
FROM:group1:TO:group2
FROM:tnm_identifier:TO:FFS
FROM:LATCHES:TO:group1
```

Use "*" to represent multiplication and "/" to represent division. The specification type of the reference TS attribute does not need to be the same as the TS attribute being defined; however, it must not be specified in terms of AUTO or IGNORE.

Examples of specifying a TS attribute in terms of another are as follows. In these cases, assume that the reference attributes were specified as delays (not frequencies).

`TS08=FROM:FFS:TO:PADS=TS05*10`	The paths between flip-flops and pads are placed and routed so that their delay is at most 10 times the delay specified in the TS05 attribute.
`TS1=FROM:PADS:TO:PADS=TS07/8`	The paths between input and output pads are placed and routed so that their delay is at most one-eighth the delay specified in the TS07 attribute.

NOTE *When a reference attribute is specified as a frequency, a multiple represents a faster specification; a division represents a slower specification.*

You can also specify a TS attribute in terms of a TS attribute that is already a specification of another. The following example provides an illustration.

```
TS09=FROM:FFS:TO:FFS=50
TS10=FROM:FFS:TO:PADS=TS09*2
TS11=FROM:PADS:TO:PADS=TS10*4
```

Sample Schematic Using End-Point Specifications

TNM identifiers define symbols or groups of symbols that are used in timing specifications. They can also define other groups. Figure 1-14 shows an example of a TNM attribute attached to an individual symbol. In this circuit, the flip-flop D_FF has the attribute TNM=D_FF attached to it.

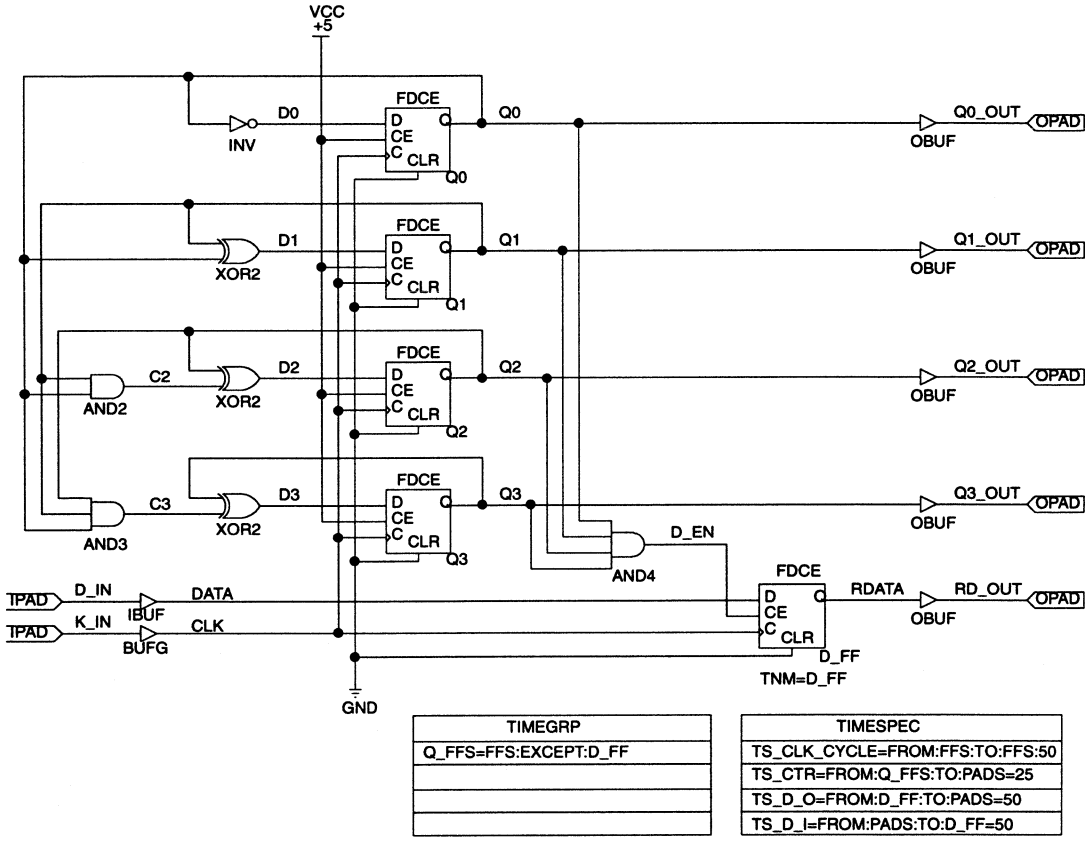


Figure 1-14 Example of Using TNMs and TIMEGRPs in Your Schematic

The TIMEGRP symbol contains an attribute that defines a group of flip-flops called Q_FFS, which includes all flip-flops in the schematic

except the one labeled D_FF. You can then use the group Q_FFS to create timing specifications in the TIMESPEC primitive. The flip-flop D_FF has its clock enable driven at 1/16th of the clock frequency; therefore, its flip-flop to pad and pad to flip-flop timing specifications are longer than the flip-flop to pad specifications in the Q_FFS group.

Default Specifications Inserted by PPR

The three types of paths that involve flip-flops — flip-flops to flip-flops, pads to flip-flops, and flip-flops to pads — usually benefit from timing specifications. If you do not specify any timing specification to control these paths, PPR applies the following specifications:

- Paths that start at pads and end at flip-flops

  `TS_DEFAULT_FROM_PADS_TO_FFS=FROM:PADS:TO:FFS=AUTO`

- Paths that start at flip-flops and end at flip-flops

  `TS_DEFAULT_FROM_FFS_TO_FFS=FROM:FFS:TO:FFS=AUTO`

- Paths that start at flip-flops and end at pads

  `TS_DEFAULT_FROM_FFS_TO_PADS=FROM:FFS:TO:PADS=AUTO`

Defining Timing Requirements Using Path-Type Specifications

The path-type specifications described in this section are provided for backward compatibility with previous versions of XACT-Performance. Since it is more complicated to define arbitrary timing requirements using path-type specifications, Xilinx strongly recommends that you use From-To statements, TIMEGRP attributes, and TNMs. Refer to "Defining Timing Requirements Using Groups" at the beginning of this chapter for more information.

This section discusses the following concepts:

- The Four Basic Path Types
- When Multiple Path-Type Specifications Apply to the Same Flip-Flop
- The Forward Tracing Mechanism
- Combinational Delay and Timing Specifications on Clock-Related Paths

- Specifying a Path-Type TS Attribute Delay in Terms of Another
- Placing TS Flags
- Other Specification Parameters
- How are Path-Type and End-Point Specifications Different?

The Four Basic Path Types

There are four basic path types in any design:

- Pad to setup (P2S) — A path that starts at an input to the device and ends at an input pin of a flip-flop.
- Clock to setup (C2S) — A path that starts at the Q pin of a flip-flop and ends at an input pin of a flip-flop.
- Clock to pad (C2P) — A path that starts at the Q pin of a flip-flop and ends at an output of the device.
- Pad to pad (P2P) — A purely combinatorial path that starts at a device input and ends at a device output.

You can define TS attributes that correspond to these four basic path types. TS attributes reside in the TIMESPEC attribute, decribed at the beginning of this chapter. For clock-related paths (C2S, C2P, P2S), you need to assign a TS attribute to the schematic by attaching a TS flag, described in the "Placing TS Flags" section. For P2P specifications, you do not need to place TS flags on your schematic.

TS flags are propagated forward until they reach a flip-flop. When more than one clock-type specification applies to the same flip-flop, some specifications override others. See "When Multiple Path-Type Specifications Apply to the Same Flip-Flop" for more information.

WARNING!!! *Although you can use TNMs and From-To syntax in the same design with path types, Xilinx discourages mixing end-point and path-type specifications.*

The following subsections describe each path type in detail.

Clock to Setup (C2S, DC2S)

TS*identifier*=**C2S:***clock_period* [:*high_time*]
TS*identifier*=**DC2S:***default_clock_period* [:*high_time*]

The C2S or DC2S parameter enables you to specify timing information for paths from Q outputs to non-clock flip-flop inputs, that is, D, CE, PRE/SD, or CLR/RD. The following figure illustrates a C2S path:

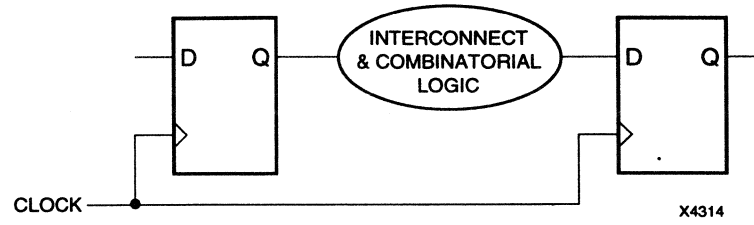


Figure 1-15 Clock-to-Setup Configuration

Clock_period represents the length of one clock signal, which indicates the interval from a rising edge to the next rising edge. *High_time* is an optional parameter that represents the clock period high time. *High_time* must be less than *clock_period,* and is only required when there are paths between flip-flops clocked by different edges of the same clock net. If *high_time* is not specified, PPR assumes it is one-half of the clock period (50% duty cycle).

A C2S or DC2S specification applies to a path between two flip-flops if it applies to both the sourcing and the destination flip-flops.

In its default mode, PPR does not control paths between two flip-flops tagged by different C2S specifications. To control such paths, set the PPR command-line parameter "user_faster_c2s" to TRUE. In this case, PPR uses the faster of the requirements at the two ends and the complete period, even if different edges clock the two flip-flops. Refer to the *XACT Libraries Guide* for more information.

If the flip-flop at either end of a path has *no* associated timing specification, the path is not controlled.

NOTE *With end point specifications, you can directly specify different requirements for arbitrary source → destination pairs. Since the end point method is easier than attaching C2S specifications to flip-flops so that the faster of the two is always appropriate, the end point style is preferable. Refer to "Defining Timing Requirements Using Groups" at the beginning of this chapter for more information on using end points.*

If the same specification governs two flip-flops in a clock-to-setup path clocked by different clock edges, the high time or the low time of the C2S specification controls the path. The high time is used if the sourcing flip-flop is rising-edge triggered, and the low time is used if the sourcing flip-flop is negative-edge triggered. If you do not specify the optional high time in the C2S specification, the system assumes a 50% duty cycle.

Examples of TS attributes using C2S and DC2S are as follows:

TS01=DC2S:40:25	The default clock-to-setup time is a period of 40 ns with a high time of 25 ns. You do not need to attach a TS flag for TS01 to any nets or load pins in the schematic because it represents a default value.
TS02=C2S:200	The clock-to-setup time for any pair of flip-flops reachable from a net with a TS02 flag attached is 200 ns.
TS03=C2S:40MHZ	The clock-to-setup frequency for any pair of flip-flops reached by tracing forward from a net with a TS03 flag attached is 40 MHz.

Pad to Setup (P2S, DP2S)

TS*id***=P2P:***pad_to_pad_time:source:destination*
TS*id***=DP2P:***default_pad_to_pad_time*

The P2S or DP2S parameter enables you to specify timing information from external pads to non-clock flip-flop inputs, that is, D, CE, PRE/SD, or CLR/RD. The following figure illustrates a P2S path:

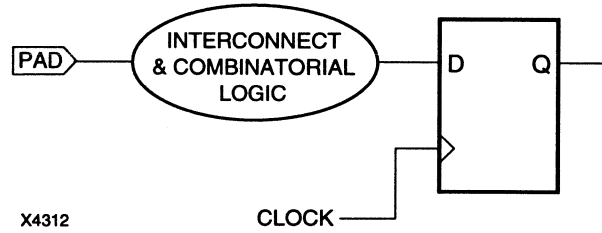


Figure 1-16 Pad-to-Setup Configuration

Pad_to_setup_time represents the desired time from external pads to the inputs. *Source* specifies an external pad. When specifying the source pad, you must use the hierarchical name for the pad listed in the XNF file. You can substitute part of the pad name with a wildcard character (* or ?). You cannot specify only a wildcard symbol as the source pad; use DP2S if you want to specify a default timing specification for all paths from external pads to inputs. Refer to the following table for the proper usage.

Occasionally, two or more unqualified P2S TS attributes propagate forward to the same flip-flop. In this case, PPR uses the fastest specification to control all paths from pads to this flip-flop. In addition, PPR applies all P2S specifications qualified by a pad name or partial pad name to the paths from the corresponding pads.

Examples of TS attributes using P2S and DP2S are as follows:

TS02=DP2S:10	The default pad-to-setup time is 10 ns. You do not need to attach a TS flag for TS02 to any nets in the schematic because it represents a default value.
TS04=P2S:110US	The pad-to-setup time for any flip-flop reachable from a net with a TS flag for TS04 is 110 µs.
TS06=P2S:25MHZ:A*	The frequency of paths from pads named with "A" as the first character to flip-flops that can be reached by tracing forward from a net with the TS06 flag attached is 25 MHz.

Clock to Pad (C2P, DC2P)

TS*id***=C2P:***clock_to_pad_time* [:*destination*]
TS*id***=DC2P:***default_clock_to_pad_time*

The C2P parameter enables you to specify timing information from Q outputs to external pads. DC2P represents the default timing specification for all paths from Q outputs to external pads in the design. The following figure illustrates a C2P path:

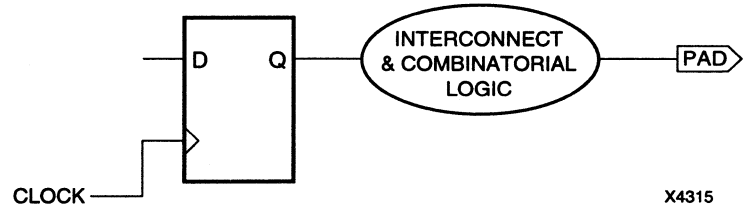


Figure 1-17 Clock-to-Pad Configuration

Clock_to_pad_time represents the desired time from Q outputs to external pads. *Destination* specifies an external pad. When specifying the destination pad, you must use the hierarchical name for the pad listed in the XNF file.

You can substitute part of the pad name with one or more wildcard characters. You cannot specify only a wildcard symbol as the destination pad; use DC2P if you want to specify a default timing specification for all paths from Q outputs to external pads. Refer to the following table for examples of proper usage.

Occasionally, two or more unqualified C2P TS attributes propagate forward to the same flip-flop. In this case, PPR uses the fastest specification to control all paths from flip-flops to pads. In addition,

PPR applies all C2P specifications qualified by a pad name or partial pad name from the paths to the corresponding pads.

Examples of TS attributes using C2P and DC2P are as follows:

TS01=DC2P:25	The default clock-to-pad time is 25 ns. You do not need to attach TS01 to any nets or load pins in the schematic because it represents a default value.
TS04=C2P:110US:*/A_OUT	The delay of a path signal from a Q output to a pad whose name ends with "/A_OUT" is 110 µs if the flip-flop is reachable from a net or load pin with the TS04 flag attached.
TS07=C2P:8MHZ	The frequency of paths to pads from a flip-flop reached by tracing forward from a net with TS07 attached to it is 8 MHz.

Pad to Pad (P2P, DP2P)

TS*id***=P2P:***pad_to_pad_time*:*source*:*destination*
TS*id***=DP2P:***default_pad_to_pad_time*

The P2P parameter enables you to specify timing information for paths between external pads. DP2P represents the default timing specification for all paths between external pads. The following figure illustrates a P2P path:

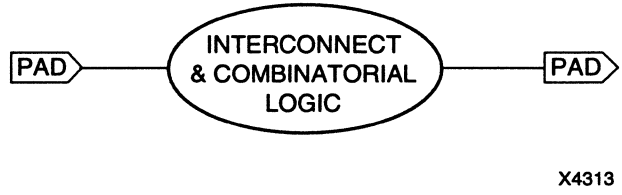


Figure 1-18 Pad-to-Pad Configuration

Pad_to_pad_time represents the desired time for paths running between external pads. *Source* and *destination* are required parameters that specify specific pads. When specifying a source or destination pad, you must use the hierarchical name for the pad listed in the XNF file. You can substitute part of the pad name with a wildcard character (* or ?). Refer to the following table for an example.

Examples of TS attributes using P2P and DP2P are listed below.

TS03=DP2P:100KHZ	The default frequency for pad-to-pad signals is 100 kHz. TS03 is not attached to any nets.
TS05=P2P:25:*:DI*	The time from any pad input to pad outputs with names that begin with "DI" is 25 ns.

You cannot, however, use a wildcard for both the input and output pad names, as illustrated by the following example.

TS10=P2P:70:*:*

This TS attribute generates an error because you should use DP2P to specify default timing requirements for this path type.

WARNING!!! *The default P2P specification must define the slowest P2P requirement of the system because the default specification applies globally to all pad-to-pad paths.*

For P2P specifications, you do not need to place TS flags on nets in your schematic since they identify by name the IOBs to which they apply. The IOB name is usually the name of the signal between the pad and I/O primitives, unless you use a BLKNM attribute to name an IOB explicitly. See the appropriate CAE Interface User Guide for more information.

Figure 1-19 shows an example of a purely combinatorial circuit and its corresponding timing specification. The timing requirement is 50 ns for all P2P paths except those leading to CS1, for which the timing requirement is 30 ns.

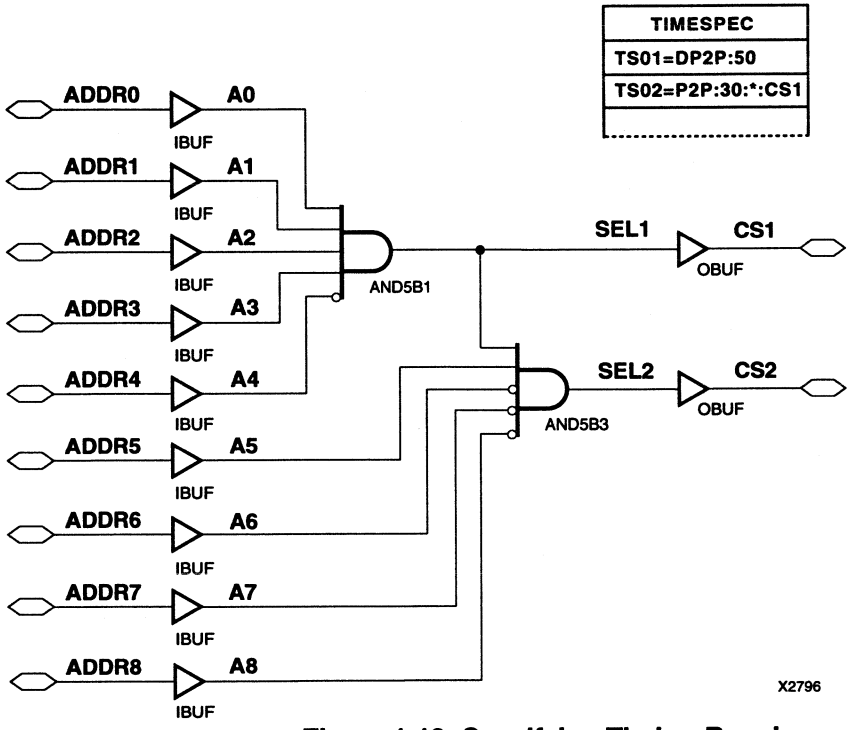


Figure 1-19 Specifying Timing Requirements on Combinatorial Paths

When Multiple Path-Type Specifications Apply to the Same Flip-Flop

Sometimes two or more timing specifications apply to the same path. XACT-Performance must then determine which timing specification to use. Since C2S, P2S, and C2P path-type specifications use flip-flops as anchor points, the pin at which the signal arrives at a flip-flop determines which timing specification takes precedence for each type as follows:

- Default TS attributes, such as DC2S, DP2S, and DC2P, have the lowest priority. A "D" preceding the path type indicates a default value.
- TS attributes of type C2S, P2S, and C2P that have a flag attached to a net that can be traced forward to the flip-flop's clock pin override default TS attributes.
- TS attributes of type C2S, P2S, and C2P that have a flag attached to a net that can be traced forward to an input pin (other than a clock pin) override TS attributes of other types.

NOTE *PPR analyzes all P2S or C2P specifications qualified by a pad or partial pad name.*

The Forward Tracing Mechanism

The forward tracing mechanism propagates an attribute through multiple levels of combinatorial logic until it reaches a flip-flop.

Specifications apply only at those flip-flops that can be reached by tracing forward from the tagged net or load pin. Whenever the system reaches a flip-flop by forward tracing, it stops. The examples in Figure 1-20 and Figure 1-21 show cases where improper placement of TS attributes results in the system ignoring timing specifications.

In Figure 1-20, the C2S specification TS01 is forward traced and attached to flip-flop B only. To control the path from A to B, some DC2S or C2S specification must also be applied to flip-flop A. Since no such specification for A is supplied here, that path is not controlled.

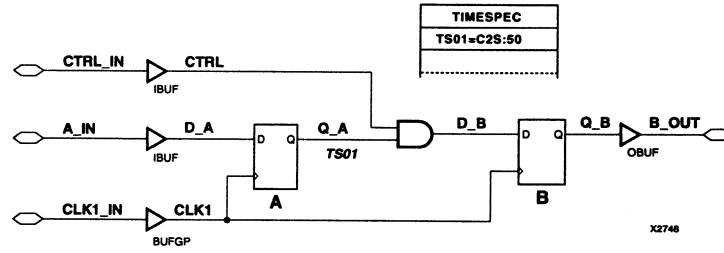


Figure 1-20 Improper Specification — Example 1

In Figure 1-21, the system cannot forward trace the C2P specification TS22 to any flip-flops and it is ignored. When a timing specification is ignored, PPR generates appropriate warnings onscreen and in the log file.

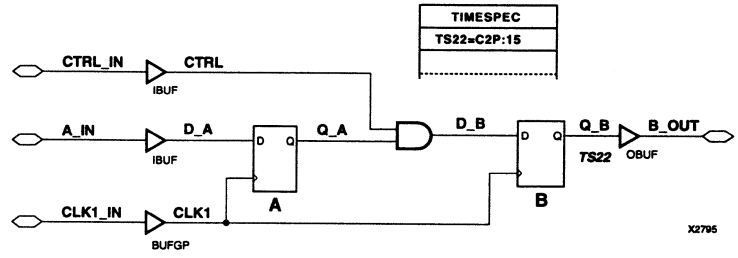


Figure 1-21 Improper Specification — Example 2

Combinational Delays and Timing Specifications on Clock-Related Paths

Flip-flop specifications (C2S, P2S, and C2P) incorporate delays, such as setup time, input/output buffer delays, and clock-to-Q delays. For example, if a P2S time is specified as 30 ns, and the clock signal has a setup time of 6 ns, then PPR uses 24 ns (30 – 6 = 24) as the maximum time allowed from the pad, through the input buffer and any intervening logic, to the D pin on the flip-flop.

Specifying a Path-Type TS Attribute Delay in Terms of Another

Use the syntax below to specify a TS attribute in terms of another.

TS*identifier*=*path_type*:*reference_TS_attribute* [*****, **/**] *integer*

where *path_type* is P2P, C2P, C2S, or P2S. The path type must not be specified in terms of a LINK, AUTO, or IGNORE path type. Use "*" to represent multiplication and "/" to represent division.

Examples of specifying a TS attribute in terms of another are as follows. In these cases, assume that the reference attributes were specified as delays.

TS08=C2P:TS05*10	A C2P path associated with the TS08 attribute is placed and routed so that its delay is at most 10 times the delay specified in the TS05 attribute.
TS14=P2P:TS07/8	A P2P path associated with the TS14 attribute is placed and routed so that its delay is at most one-eighth the delay specified in the TS07 attribute.

A circular definition, as follows, causes an error:

```
TS01=C2P:TS02*2
TS02=P2P:TS01*3
```

NOTE *When a reference attribute is specified as a frequency, a multiple represents a faster specification; a division represents a slower specification.*

Placing TS Flags

This section describes general guidelines for placing TS flags. For specific instructions on placing TS flags with your particular CAE interface, refer to the appropriate Xilinx Interface User Guide for your schematic editor.

A TS flag attaches timing information from a non-default TS attribute to a net, load pin, or macro load pin in the schematic.

WARNING!!!

You cannot place a TS flag on a source pin.

After you place the TS flags on the schematic, PPR reads this timing information and applies it to the relevant paths in the design using the forward tracing mechanism to determine the flip-flop to which the specification applies.

If your schematic entry package does not allow you to place TS flags on pins, you can place the TS flag on the net that propagates forward to the appropriate load pin. If you need to flag only part of a net, use a buffer to separate these pins from others on the net. Then place the TS flag on the output net of the buffer.

You do not need to place TS flags on the schematic for default TS attributes because timing information from default TS attributes applies throughout the design. Also, you do not need to place TS flags for P2P attributes.

Careful placement of TS flags ensures that PPR properly interprets your timing specifications. The following sections discuss the proper placement of TS flags.

Placing TS Flags on the Schematic

Figure 1-22 provides an example of how PPR interprets information represented by TS attributes and TS flags. TS01 represents a default specification that applies to the entire design. TS01 ensures that PPR maps, places, and routes the net so that no clock-to-setup paths in the design exceed 50 ns. TS02 is a non-default specification and has a TS02 flag attached to a net (CLK2). TS02 supersedes the default specifications for paths that start and end at a flip-flop reachable by CLK2.

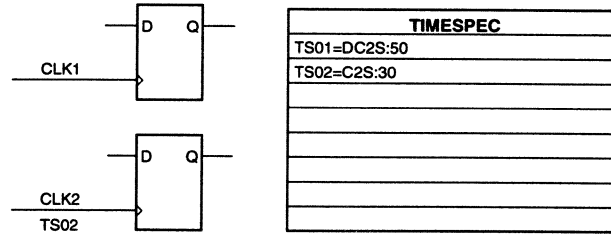


Figure 1-22 Default and Non-Default Timing Specifications

As illustrated by the following figure, flip-flop-related timing specifications (C2P, P2S, C2S) do not need to be attached to a net directly connected to the flip-flop. PPR uses forward tracing to

determine which flip-flops the timing specification controls. In both cases, TS02 traces forward to the flip-flop.

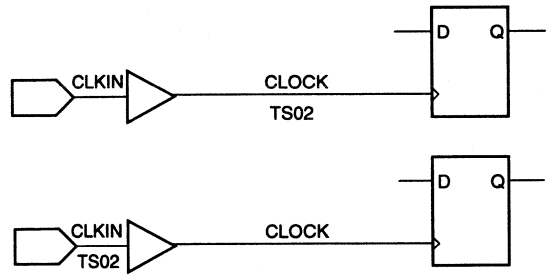


Figure 1-23 Equivalent Specifications on Clock Net

The timing specification for TS flags attached to a clock net are applied to every flip-flop to which the clock net can be forward traced. However, since TS flags can be placed on signals other than clock nets, you can attach TS flags so that flip-flops clocked by the same signal have different timing specifications. Forward tracing stops upon reaching a flip-flop or input latch.

Figure 1-24 illustrates an example of flip-flops with a common clock but different timing specifications. The C2P specification for paths sourced by Q2 and Q3 is 80 ns, and the C2P specification for paths sourced by Q1 is 120 ns.

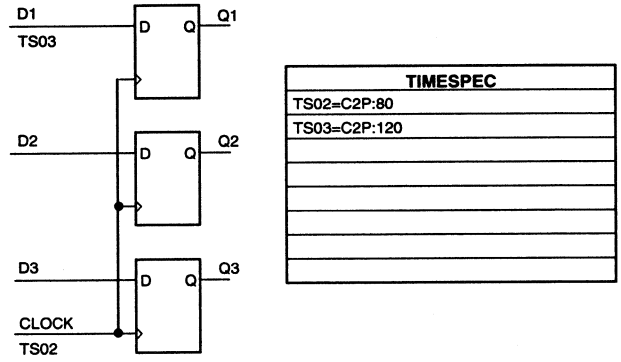


Figure 1-24 Flip-Flops Clocked by Same Signal, but with Different Timing Specifications

A TS flag can be attached to any of a flip-flop's input nets, since its corresponding specification is traced forward to the flip-flop.

As illustrated by the following figure, a TS flag attached to a net sourced by the flip-flop's Q output does not apply to the flip-flop, as the flip-flop is not along the forward path from the net to which the TS

flag is attached. In Figure 1-25 the P2S specification represented by TS02 is applied only to flip-flop B.

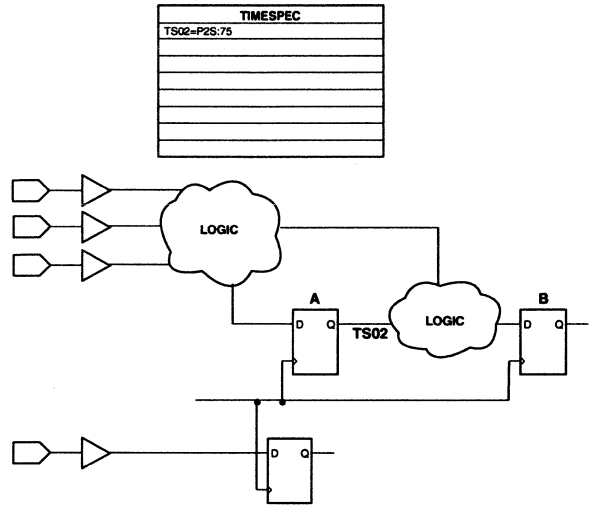


Figure 1-25 Example of Timing Specification that Cannot Be Traced to Both Flip-Flops

To apply the P2S specification to flip-flops A and B, place the TS02 flag as shown in Figure 1-26.

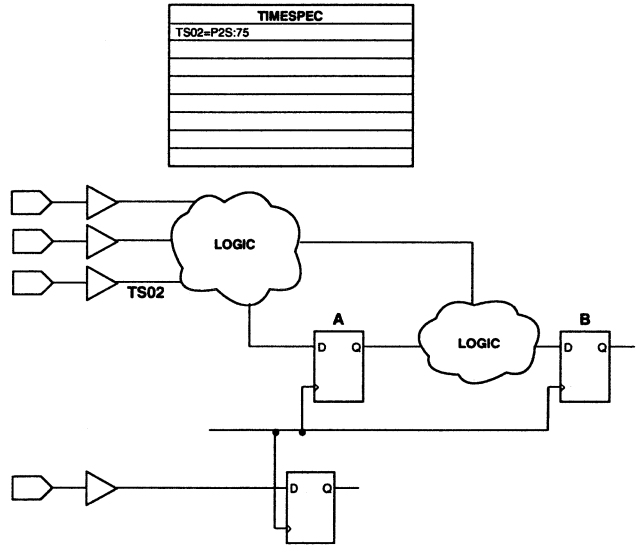


Figure 1-26 Example of Timing Specification that Can Be Traced to Both Flip-Flops

As illustrated by the following example, you need to carefully place TS flags that represent C2S specifications because they involve pairs of flip-flops. Figure 1-27 shows a design where the C2S timing specification cannot be interpreted because it can only be traced forward to one flip-flop.

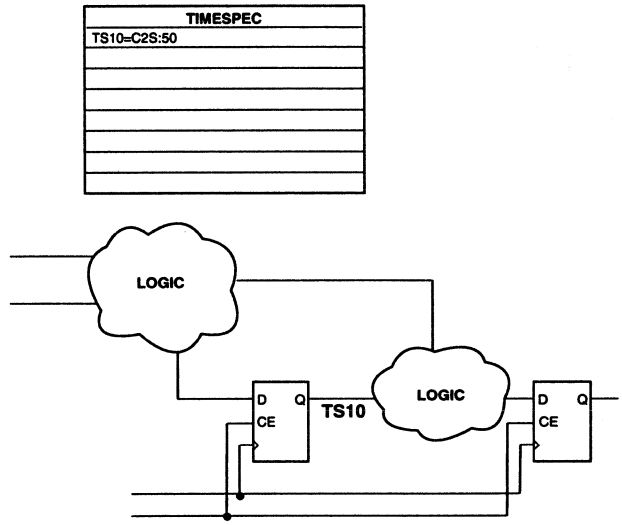


Figure 1-27 Improper Placement of TS Flag for Clock-to-Setup Specification

To ensure the specification is applied correctly, apply the TS flag to a net that feeds all flip-flops to which it applies, as illustrated in Figure 1-28, where TS10 is attached to the clock enable net.

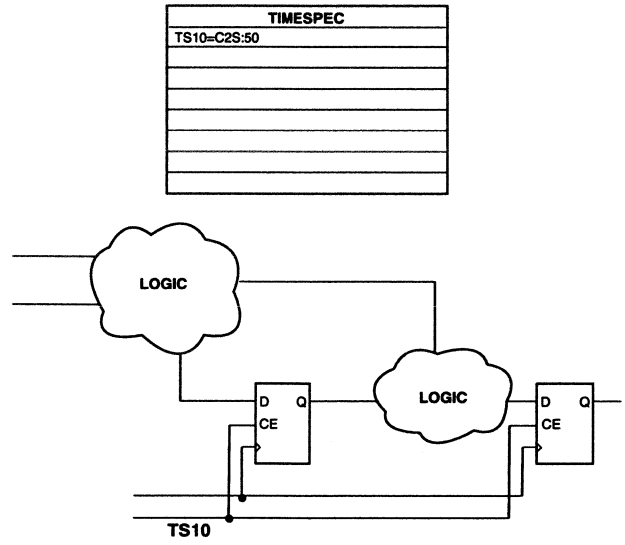


Figure 1-28 Proper Placement of TS Flag for Clock-to-Setup Specification

Placing TS Flags on Cascaded Counters

If your design contains a cascaded counter where a signal from one counter enables another via its CE pin and both counters are driven by the same clock, the whole circuit does not need to work at the frequency of the clock signal. If you place a single TS flag as shown in Figure 1-29, then the specification is applied to the whole circuit.

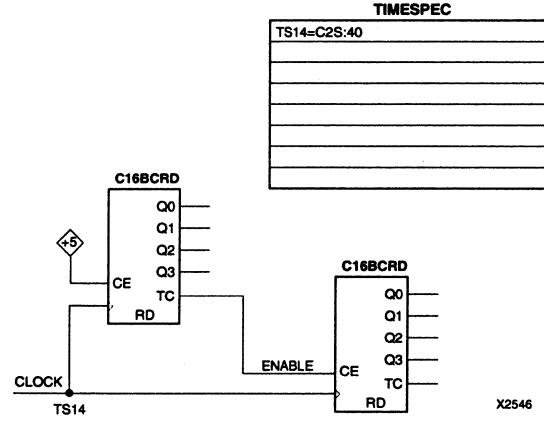


Figure 1-29 Cascaded Counter with Timing Controlled by One TS Attribute

You can use an additional TS flag, placed on the net connecting the counters, to clock the second counter at a fraction of the first, as illustrated in Figure 1-30.

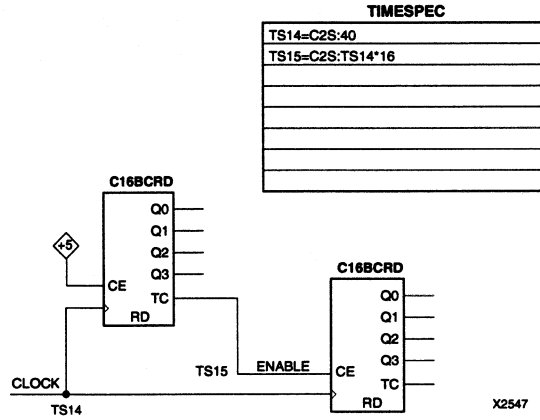


Figure 1-30 Cascaded Counter with Timing Controlled by Two TS Attributes

Other Specification Parameters

The following subsections describe other parameters you can use when defining TS attributes:

LINK (Link TS Attributes)

The LINK parameter creates one TS attribute that incorporates multiple TS attributes. You can use the LINK parameter to reduce clutter from TS attributes within your schematic.

LINK:TS*id_1***:TS***id_2***[:TS***id_3***]**

For example, suppose TS05 represents a clock-to-pad value, TS06 represents a pad-to-setup value, and TS07 represents a clock-to-setup value. TS08 is defined as follows:

TS08=LINK:TS05:TS06:TS07

A TS flag for TS08 could then be attached to a net, creating the same effect as attaching TS flags for TS05, TS06, and TS07 to the net.

TS05, TS06, TS07 must represent a clock-to-pad, clock-to-setup, pad-to-setup, or LINK parameter. If you specify other values, XNFPrep generates an error in its report file.

IGNORE (Ignore Path Type)

When a TIMESPEC attribute is defined using the following syntax, PPR ignores paths of the specified type that are not explicitly defined.

TS*id=default_path_type***:IGNORE**

Sample Schematic Using Path-Type Specifications

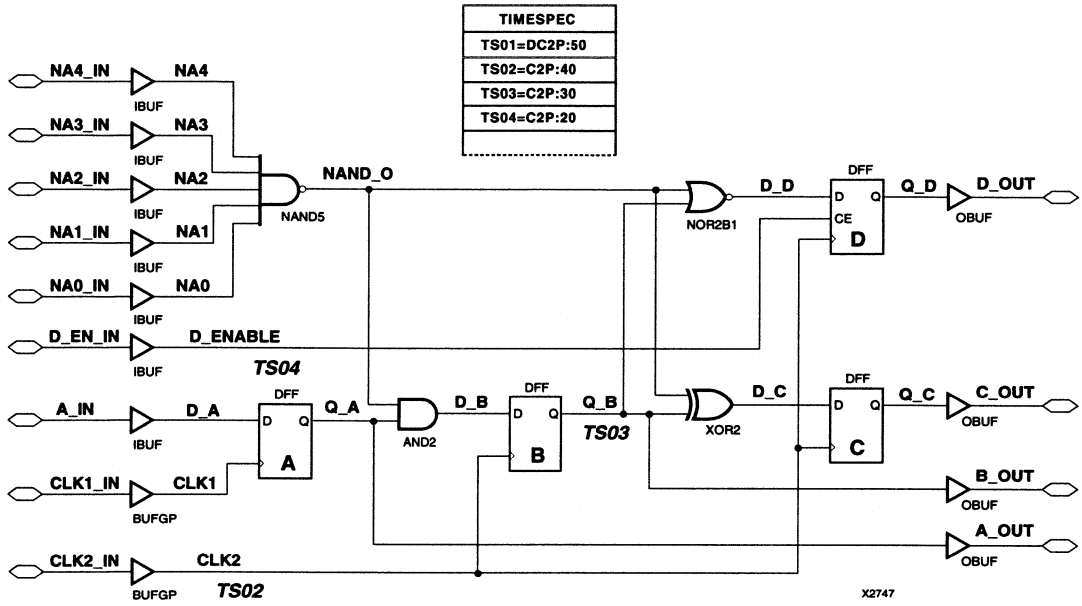


Figure 1-31 Example of Schematic Using XACT-Performance

The following notes explain the example in Figure 1-31.

- TS01=DC2P:50 is a TS attribute that specifies a default clock-to-pad time of 50 ns. Only one default specification is allowed for each path type.
- The TS02=C2P:40 TS attribute has a corresponding TS flag attached to net CLK2. This net drives the clock pins of flip-flops B, C, and D. TS02 overrides the default specification (TS01) for these flip-flops.

  The clock-to-pad paths that TS02 is specifying run from these flip-flops forward to output pads. The net CLK2, flagged with TS02, is not itself on these paths. Instead, the single flag on a clock net controls several paths associated with the flip-flops that the clock net drives.

- The TS03=C2P:30 TS attribute has a corresponding TS flag attached to net B_OUT. This net can be forward traced to the data inputs of flip-flops C and D. It overrides the default specification (TS01) and other specification (TS02) for each flip-flop.
- The TS04=C2P:20 TS attribute has a corresponding TS flag attached to the net D_ENABLE. This net drives the clock-enable input of flip-flop D. TS04 is the second specification that arrives at flip-flop D; it overrides TS03 because it is a faster specification. The general rule is that when two or more specifications of the same level arrive at a flip-flop, the system applies the fastest timing specification unless a pad name is specified.
- All flip-flop-related path-type timing specifications are assumed to be relative to the clock pin on the flip-flops being analyzed. Clock skew and input clock delay are not considered during the path analysis.

How are Path-Type and End-Point Specifications Different?

You can use both path-type specifications and end-point specifications to specify the same set of paths as shown in the following table:

Path-Type Specifications	End-Point Specifications
TS01=C2S:20	TS01=FROM:FFS:TO:FFS=20
TS02=P2S:20	TS02=FROM:PADS:TO:FFS=20
TS03=C2P:20	TS03=FROM:FFS:TO:PADS=20
TS04=P2P:20	TS04=FROM:PADS:TO:PADS=20

However, the two specification types differ in the following ways:

- A C2S or P2S specification does not control a path that ends at the clock pin of a flip-flop. An end-point specification, on the other hand, can control a path that ends at the clock pin of a flip-flop. Therefore, the end-point specification can limit clock skew.
- A slower path-type specification can override a path-type specification as explained in the section "When Multiple Path-Type Specifications Apply to the Same Flip-Flop." No end-point specification overrides another.
- End-point specifications are the only way to specify timing requirements on paths that start or end at RAMs or input latches.

Syntax Summary

The following sections summarize the XACT-Performance syntax.

TNM Attributes

The following table lists the syntax used when creating TNMs, which you enter directly on the primitive symbol, macro symbol, signal, or load pin.

Flag Type	TNM Attribute Syntax
Symbol Macro symbol	**TNM**=*group1* [;*group2* . . .] **TNM**=*predefined_group*:*group1* [;*predefined_group*:*group2* . . .]
Signal	**TNM**=*group*
Load pin	**TNM**=*group*

TIMEGRP Attributes

The following lists the syntax used within the TIMEGRP primitive.

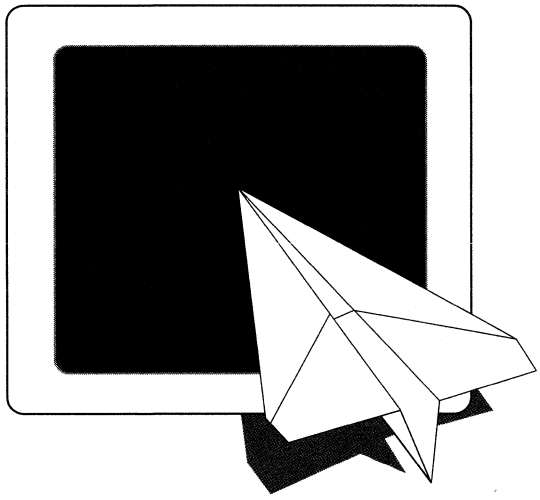
Group Type	TIMEGRP Attribute Syntax
Combine	*new_group*=*group1*:*group2* [:*group3* . . .]
Exclude	*new_group*=*group1*: [:*group2* . . .]:**EXCEPT**:*group3*: [:*group4* . . .]
Clock Edge	*new_group*=**RISING**:*group1* *new_group*=**FALLING**:*group1*

TIMESPEC Attributes

The following lists the syntax used for parameters that define TS attributes, which reside in the TIMESPEC primitive.

Spec Type	TIMESPEC Attribute Syntax	TS Flag Required?
From-To	**TS***id*=**FROM**:*group*:**TO**:*group*=*delay*	No
C2S	**TS***id*=**C2S**:*delay* [:*high_time*]	Yes
C2P	**TS***id*=**C2P**:*delay* [:*to_pad_name*]	Yes
P2S	**TS***id*=**P2S**:*delay* [:*from_pad_name*]	Yes
P2P	**TS***id*=**P2P**:*delay* :*from_pad_name*:*to_pad_name*	No
DC2S	**TS***id*=**DC2S**:*delay* [:*high_time*]	No
DC2P	**TS***id*=**DC2P**:*delay*	No

Spec Type	TIMESPEC Attribute Syntax	TS Flag Required?
DP2S	**TS***id*=**DP2S:***delay*	No
DP2P	**TS***id*=**DP2P:***delay*	No
Ignore	**TS***id*=**IGNORE**	Yes
Link	**TS***id*=**LINK:TS***id_1*:**TS***id_2* [:**TS***id_3* . . .]	Yes
Auto	**TS***id*=*path_type*:**AUTO**	Yes



XACT Reference Guide, Volume 1

The XNFCVT Program

XACT Reference Guide, Volume 1 — April, 1994 (0401129 01)

Printed in U.S.A.

The XNFCVT Program

This Program is Compatible with the Families Indicated.

☑ XC2000	☑ XC3100	☑ XC3100A	☑ XC4000H
☑ XC2000L	☑ XC3000A	☑ XC4000	☐ XC7200
☑ XC3000	☑ XC3000L	☑ XC4000A	☐ XC7300

XNFCVT converts a version 5 Xilinx Netlist Format (XNF) file to a version 4 or version 1 XNF file. XNFCVT also converts a version 4 XNF to a version 2 XNF, or a version 2 XNF to a version 1 XNF.

XNF syntax has been modified to support improved design flow, new design features, and new LCA families. As a result, some programs that function with a version 1 XNF file might not function if a version 2 XNF file is used as input.

A similar situation occurs if a version 4 netlist is used as an input to a program that expects a version 2 netlist. The XNFCVT program translates the netlist to the desired version without impacting design function.

NOTE *Version 4 and 5 XNF files that contain XC4000 specific symbols cannot be converted to version 1 or version 2 XNF files.*

Currently there are four XNF versions — version 1, version 2, version 4, and version 5. To determine the version of an XNF file, look at the first line of the file.

LCANET, 1 (this is a version 1 XNF file)
LCANET, 2 (this is a version 2 XNF file)
LCANET, 4 (this is a version 4 XNF file)
LCANET, 5 (this is a version 5 XNF file)

Syntax

Use the syntax shown here to create a lower version XNF file from your XNF file.

xnfcvt [*options*] *input*[**.xnf**] *output*[**.xnf**]

Files

This section describes the files associated with the XNFCVT program.

input.xnf

This is the version 5, 4, or 2 XNF file that you want to convert to a lower version.

output.xnf

This is the target, version 4, 2, or 1 XNF file. You must specify the file name. See the –v option description in the Options section for information on specifying the target version.

Options

The XNVCVT program has two command line options.

– a Do Not Use an AKA File

This option instructs XNFCVT not to use the existing AKA file for generating hierarchical prefixes; instead, XNFCVT generates a new AKA file. If you do not specify the –a option, XNFCVT automatically looks for an AKA file with the same name as the input file. If one is found, XNFCVT uses that file to generate shortened name prefixes. This option allows you to use prefixes from previous runs.

– v Specifies the Version of the XNF File

The –v option specifies the version (4, 2, or 1) of the target XNF file. If no version number is specified, XNFCVT converts the input XNF file to the previous netlist version with one exception. If the input file is a version 4 netlist for an XC4000 design, XNFCVT does not translate the XNF file. If no version is specified, XNFCVT defaults to the earlier version (5 to 4, 4 to 2, 2 to 1).

Summary of Version Differences

The differences between version 1 and version 2 XNF files are summarized below.

- MAP=type Symbol parameter is added for use with the CLBMAP and IOBMAP symbols
- A Pin Lock signal flag is added (P flag)
- A Save signal flag is added (S flag)
- Each symbol and signal now have a full hierarchical path name
- The "/" character is included in signal and symbol names
- The LCANET parameter (line 1 of the file) changed from 1 to 2

The changes from version 2 to version 4 are listed below.

- Pin parameters have been added to aid delay-driven routing
- Symbols have been added to support XC4000 architecture, such as, WAND, BSCAN, BUFGP, and so forth
- Bus record has been added
- OUTFFZ and OBUFZ have been changed to OUTFFT and OBUFT
- Parameters have been added to help with logic placement
- The LCANET parameter (line 1 of the file) has changed from 2 to 4

The changes from version 4 to version 5 are listed below.

- The TIMGRP symbol has been added for XACT Performance
- CY4_01 to CY4_42 carry mode symbols have been added
- The BUFG symbol has been added
- The TS parameter associated with the TIMESPEC symbol are no longer user parameters (no preceding equal sign is necessary, although Xilinx supports these as user parameters, too)
- The PAD and PADU symbols are obsoleted and replaced in the Unified Library by IPAD, OPAD, IOPAD, and UPAD symbols
- The DOUBLE parameter on a PULLUP that is connected to an external is considered an error; it is not ignored
- Signal names are no longer valid in EQN symbol; use pin names instead
- LOC ranges must be specified with a colon (:) instead of a semi-colon (;)
- The INFF and INLAT symbols no longer have the buffered input as part of the symbol
- The I pin is the invertible pin on a WAND with DECODE specified instead of the O pin
- Pin names have changed for the following (LCANET 4 pin names are still supported)
  - Combinatorial gates have changed input pin names from 1–5 to I0–I4
  - WORAND input pin names have changed from I1 and I2 to I0 and I1

  - INLAT and DLAT L pin name has been changed to G
  - DFF and DLAT RD pin name has been changed to CLR (clear)
  - DFF and DLAT SD pin name has been changed to PRE (preset)
- New signal and pin parameters TNM and TSidentifier have been added
- New symbol and EXT parameters TNM and HBLKNM have been added
- The following new IO parameters TTL, CMOS, RES, and CAP have been added for the XC4000H architecture
- The following new symbol parameters CYMODE, SCHNM, LIBVER, TS, RLOC, USER_R-LOC, U_SET, HU_SET, RLOC_ORIGIN, and RLOC_RANGE have been added

XNFCVT Program Process

The XNFCVT program removes all attributes for the target LCA file that are illegal and performs the following functions on an XNF file:

1. Reads in an XNF file
2. Reads and updates the AKA (alias names) file (version 2 to version 1)
3. Shortens hierarchical path names of symbols and signals (version 2 to version 1)
4. Generates an XNF file that corresponds to the specified version
5. Removes bus records (version 4 to version 2 or 1)
6. Changes OUTFFT and OBUFT to OUTFFZ and OBUFZ (version 4 to version 2 or 1)

The AKA File (Version 2 to Version 1 Only)

The first time XNFCVT is run on a file, it generates an AKA file containing automatically generated prefix names and the corresponding path name that the prefixes represent. In each successive run without the –a option, the AKA file is read with the XNF file. The program uses existing prefixes from the AKA file for identical path names and only generates new prefixes when a new path name is encountered. You can also edit this file to make prefix names more meaningful, as in the following examples.

```
# design.aka alias file created by XNFCVT on Tue Oct 13
# 14:00:45 1992
# WARNING! If you edit the prefix names, DO NOT use the
# WARNING! '$' character as the first character in your
# WARNING! own prefix names.
$1    /TOP/U12
$2    /TOP/U28
$3    /TOP/U12/COUNTER
```

In the XNF file produced by XNFCVT, symbols and signals that are at the /TOP/U12 level have a shortened name: $1 replaces the path /TOP/U12. For example, if there is a signal in the input XNF file called /TOP/U12/SIG1, it is called $1-SIG1 in the output XNF file.

NOTE

Since XNF version 1 does not support the slash "/" character,; the hyphen "-" character is used to separate the prefix from the symbol or signal name.

If you run XNFCVT again after adding a new hierarchy level called U30 and deleting the one called U28, the changes in the AKA file looks like the following example:

```
$1    /TOP/U12
$3    /TOP/U12/COUNTER
$4    /TOP/U30
```

If you run XNFCVT with the –a option, the existing AKA file is ignored. The new generated AKA file looks like the following example:

```
$1    /TOP/U12
$2    /TOP/U12/COUNTER
$3    /TOP/U30
```

You can edit the prefixes to make them more meaningful. Prefix names should not contain the separator character "–".

Error Messages and Recovery Techniques

Error 201 **Unable to open file** *file_name* **for** *reading/writing*.

You can get this error if the input file cannot be opened for reading or it does not exist; check the file name. If the output file cannot be opened for writing, check if the hard disk is full and if the output file already exists and is write-protected.

Error 203 **Illegal part, [%s] for target LCANET version.**

Designs containing XC4000 parts cannot be converted to any LCANET version lower than LCANET 4. If you are converting from an LCANET version 4 or version 5 to any lower version, you must ensure that you are using either an XC2000 or XC3000 part.

Error 205 **No parttype specified in source file.**

The part type must be specified in the input XNF file.

Error 206 **Invalid LCANET %s. Valid types 2 and 4 and 5.**

XNFCVT converts only XNF versions 5, 4, and 2. (There is no version 3.)

Error 207 **Invalid conversion path from LCANET version** *source version* **to** *target version*.

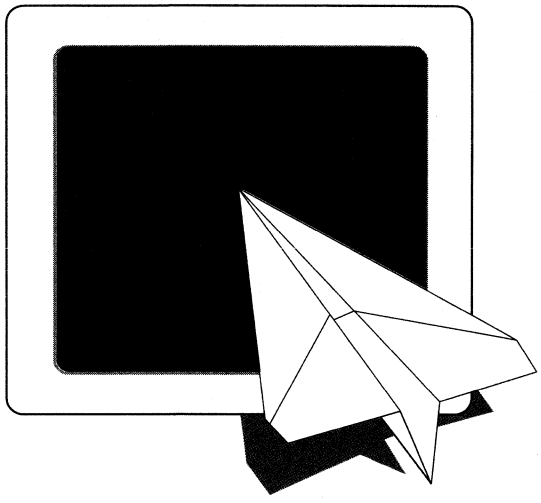
The only legal conversion paths are from:
LCANET version 5 to 4, 2, or 1.
LCANET version 4 to 2 or 1.
LCANET version 2 to 1.
All other conversion paths are illegal.

Error 208 **Bad command line option.**

The -v option requires an input number. This value must be an integer.

Error 211 *sym name* **of type** *symtype* **is illegal for target LCANET net version** *target version*.

A symbol of a type that is illegal in the target LCANET version was found.



XACT Reference Guide, Volume 1

HM2RPM

XACT Reference Guide, Volume 1 — April, 1994 (0401129 01)

Printed in U.S.A.

HM2RPM

This Program is Compatible with the Families Indicated.

☐ XC2000	☐ XC3100	☐ XC3100A	☑ XC4000H
☐ XC2000L	☐ XC3000A	☑ XC4000	☐ XC7200
☐ XC3000	☐ XC3000L	☑ XC4000A	☐ XC7300

The HM2RPM program translates a hard macro (HM) file into an XNF file that contains a relationally placed macro (RPM). This translation is necessary for some designs because the XACT 5 software release only supports the RPM files that replace hard macro files.

Hard macros are encoded files representing segments of XC4000 LCA logic that are mapped into LCA logic blocks, then placed and routed for a specific FPGA part. RPMs are a super-set of soft macros that, unlike hard macros, include standard logic gates that can be simulated. RPMs group logic into LCA blocks where appropriate. They replace the hard macro placement information with relative location (RLOC) constraints. RPMs can include carry logic symbols and BUFT symbols as well as other CLB-related logic. Unused logic is automatically trimmed from RPMs so that only the necessary logic is implemented in the FPGA.

For designs that currently include user-created hard macro files but that are enhanced or completed with the XACT 5 release, you must translate the hard macro files into RPM files. Using the HM2RPM translator is also necessary if you want to move your current hard macro files into a form that is usable with the Xilinx Unified Libraries. You do not need to use HM2RPM when you use a schematic entry tool to design new RPMs with the Unified Libraries, or use the Xilinx-supplied RPM macros. The standard schematic translator translates the Unified Libraries' RPM logic into XNF files that can be merged into the complete design like any other soft macro.

There are two types of hard macros: macros that you create and macros that Xilinx supplied with libraries created before the Unified Libraries. With the XACT 5 release, Xilinx provides the RPM replacement files for the hard macro files found in the previously supplied libraries. If your design uses these libraries and includes Xilinx-supplied hard macros, you do not need to translate any files or make any modifications to the symbols that instantiate the hard macros. The file-flattening program, XNFMerge, automatically finds the Xilinx-supplied RPM replacement files. If you created your own hard macro files, you must use the HM2RPM program to convert the hard macro files and place the new RPM files in a search directory so that XNFMerge can find them to include in the design.

When hard macros were generated, the HMGEN program prompted for default logic values for each of the hard macro's input pins. If some of the input pins are left unconnected when the hard macro is placed in a design, PPR ties those symbol pins to their defined default logic values. For example, if a counter hard macro in a schematic is left with its CE pin unconnected with a default logic value of 1, PPR ties this symbol pin to VCC. No logic is trimmed from this hard macro.

When HM2RPM translates a hard macro, it reads the default logic value information from the HM file, then creates additional logic that multiplexes either a default logic value or the input signal to the symbol pin. XNFPrep trims this additional logic appropriately. If any input pin on the symbol is left unconnected, XNFPrep trims this additional logic so that the default logic value is applied. If any input pin on the symbol is connected, XNFPrep trims this additional logic to remove the default logic and propagate the input signal correctly.

When XNFPrep trims logic, the trimmed logic and nets contain the FLOAT_HMINPUT string in their names and can be safely ignored.

User-Created Hard Macros

You must convert any user-created hard macros into RPMs with the HM2RPM program when you use these macro symbols in a design. When you perform this conversion, you must specify whether your design uses elements from the Unified Libraries or from previous libraries. By default, HM2RPM outputs an XNF file that is compatible with schematics generated with previous library elements. The r=true option generates an XNF file compatible with the Unified Libraries. This option is described in more detail in the "HM2RPM Options" section later in this chapter.

You do not need to modify the original user-created hard macro symbols in your design. Simply run HM2RPM on the HM file to which each of these symbols points.

When you run XNFMerge on a design containing user-created hard macro symbols, it automatically searches for the corresponding XNF file in the following order: in the current directory, in the -d search directories that you specify, and in the $XACT/data/hmlib directory. Because of this search order, it is recommended that you place all user-created RPMs in the current directory or in the -d directory.

NOTE *Before the XACT 5 release, LOC constraints on hard macros applied to the lower left corner of the CLB. They anchored the corner of the hard macro structure at a specific location. With the XACT 5 software, the LOC attribute is still applied to the lower left corner of the CLB if the DEF=HM attribute is retained on the symbol. However, if the DEF=HM attribute is removed from the symbol, the software treats the LOC constraint on the symbol as it does any other soft macro. It propagates the same LOC constraint to all the symbols in the macro that do not already have a LOC or RLOC constraint. It does not anchor the underlying structure of logic. You can use the RLOC_ORIGIN attribute to anchor RPM logic in the same way as you formerly used the previous hard macro symbols.*

Designs with Elements from Previous Libraries

Hard macro symbols in designs composed of elements from libraries created before the Unified Libraries retain the DEF=HM attribute originally attached to them so that XNFMerge can later correctly process any LOC constraints on the hard macro symbol.

When using symbols from previous libraries, do not use the r=true option during the conversion. By default, HM2RPM outputs an XNF file that is compatible with schematics generated with previous library elements.

Designs with Elements from the Unified Libraries

When using the Unified Libraries symbols, be sure that you use the r=true option during the conversion of your hard macro HM files so that HM2RPM will output an XNF file that is compatible with schematics generated with the Unified Libraries.

Xilinx-Created Hard Macros

Xilinx-created hard macro symbols retain the DEF=HM attribute originally attached to them so that XNFMerge can correctly process any LOC constraints on the hard macro symbol.

You do not need to modify the original Xilinx hard macro symbols in your design.

When you run XNFMerge on a design containing Xilinx hard macro symbols, it automatically searches for the corresponding XNF file in the following order: in the current directory, in the –d search directories that you specify, and in the $XACT/data/hmlib directory.

Designs with Elements from Previous Libraries

The XACT 5 versions of the XNFMerge, XNFPrep, and PPR programs do not accept hard macros. The hard macros previously found in the Xilinx hard macro library have already been converted to RPMs and placed in the $XACT/data/hmlib directory. You can only use the RPMs in this directory in designs that contain symbols from libraries released before the Unified Libraries. In these designs, Xilinx hard macros already point to Xilinx RPM models rather than to HM files, so you do not need to modify your symbols or convert the Xilinx HM files. XNFMerge finds and merges these Xilinx RPM files.

Designs with Elements from the Unified Libraries

You can use Xilinx hard macro symbols in a Unified Libraries design, but it is recommended that you find the equivalent RPM in the Unified Libraries or create your own at the schematic level. If you use any hard macro symbols, including Xilinx hard macros, in a Unified Libraries design, you must convert them into RPMs with the HM2RPM program.

When starting a new design using the Unified Libraries symbols, be sure that you use the r=true option during the HM2RPM conversion so that the output XNF file will be compatible with the Unified Libraries. It is required that you place the XNF file in the current project directory. If it is not placed there, XNFMerge may find the wrong file and XNFPrep may fail because the Unified Libraries and earlier libraries have been mixed.

Table 1-3 summarizes when to use HM2RPM.

Table 1-3 Unified Libraries vs. Previous Libraries

Hard Macro Source	Unified Libraries	Previous Libraries
Xilinx-created macros	Use equivalent RPM or run HM2RPM r=true.	No action required.
User-created macros	Run HM2RPM r=true.	Run HM2RPM.

Design Flow

Figure 1-32 describes where HM2RPM fits into the general Xilinx design flow.

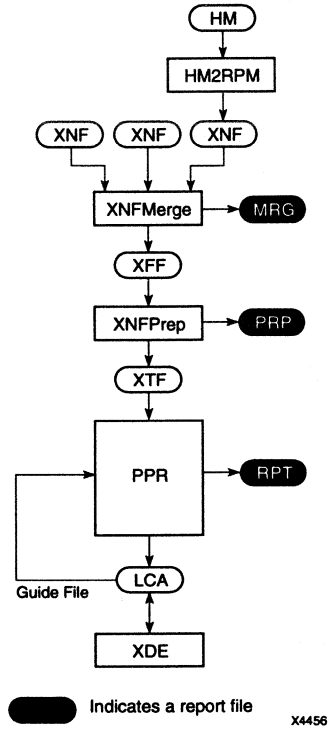


Figure 1-32 HM2RPM in XC4000 Design Flow

Follow these design steps to process an XC4000 design with hard macros.

1. Install the XACT 5 or later release of the XNFMerge, XNFPrep, and PPR programs.
2. Run HM2RPM on all hard macros that you have created. The "How to Use HM2RPM" section later in this chapter gives you specific instructions on this procedure.
3. Functionally simulate the design, if desired.

   If your design does not have any user-created hard macros, X-BLOX elements, or Xilinx ABEL elements, you can stay within your design environment to simulate, provided that all the elements of your design have simulation models. The Xilinx hard macros have simulation models.

   If your design has user-created hard macros, X-BLOX elements, or Xilinx ABEL elements, you can use XSimMake to simulate. XSimMake creates all the simulation models needed for these elements, which have been processed previously by HM2RPM, X-BLOX, or Xilinx ABEL, respectively.
4. Run XMake on the top-level design file. XMake automatically runs XNFMerge.
5. Continue with the rest of the Xilinx design flow.

Files

Input Files

The input to HM2RPM is an HM file containing a hard macro.

Output Files

HM2RPM outputs the following files.

- As a default, HM2RPM outputs a Version 5 XNF file that is compatible with the schematic libraries that were created before the Xilinx Unified Libraries.

When you use the r=true option, HM2RPM outputs a version 5 XNF file that is compatible with the Unified Libraries.

HM2RPM always generates XNF files with CY4 primitives when carry logic is used. It also always generates a version 5 XNF file since it contains RLOC information.

- The log file contains error, warning, and informational messages produced by HM2RPM as it processes the design. The name of this file is hm2rpm.log.

How to Use HM2RPM

This section describes how to use HM2RPM.

Invoking HM2RPM

You can access HM2RPM through the XACT Design Manager (XDM) or through the operating system command line.

From XDM

To run HM2RPM from XDM, access XDM according to the instructions in the "XDM" chapter of this manual.

1. Click on **Translate** → **HM2RPM**.
2. Click on any options that you want to set. The available options are listed in the "HM2RPM Options" section of this chapter.
3. Click on **Done**.

By default, HM2RPM outputs a version 5 XNF file that is compatible with the schematic libraries that were created before the Xilinx Unified Libraries.

From the Command Line

To invoke HM2RPM on the command line, use the following syntax:

hm2rpm *inputfile_name* **.hm** *outputfile_name* **.xnf [***options***]**

By default, this syntax outputs a version 5 XNF file that is compatible with the schematic libraries that were created before the Xilinx Unified Libraries.

Inputfile_name is the name of the HM input file containing a hard macro; it must be begin with an alphabetic character. This parameter is required. If no extension is specified on the input file name, an .hm extension is used as a default.

Outputfile_name is the name of the output XNF file. It must begin with an alphabetic character. If no extension is specified on the output file name, an .xnf extension is assumed.

Options can be any one of the options listed in the "HM2RPM Options" section of this chapter.

You must run HM2RPM on each hard macro file that you have created. When the new XNF file is created, place this file in either the current working directory or in a search directory that is specified for XNFMerge. When XNFMerge reads a symbol with the DEF=HM attribute, it searches for the corresponding XNF file in the following order: the current directory, the -d search directories specified through the -d option for XNFMerge, and finally the $XACT/data/hmlib directory. Do not place your translated macro files into the $XACT/data/hmlib directory, which should be reserved for Xilinx-supplied files.

Creating Unified Libraries-Compatible XNF File

To generate an XNF file that is compatible with schematics generated with the Unified Libraries, use the following syntax:

hm2rpm *inputfile_name* **.hm** *outputfile_name* **.xnf r=true**

Obtaining Help

You can obtain help in two ways when using HM2RPM. You can type **hm2rpm** ↵ or **hm2rpm -helpall**↵. Either command brings up a description of the options available in HM2RPM and their settings, but the latter also gives information on the log file and the parameter file. Any other options entered at the same time as -Helpall are ignored.

HM2RPM Options

This section describes the options that are available in HM2RPM.

–Helpall

The –Helpall option brings up a description of the HM2RPM options and their settings, input files, and output files.

Command line syntax:	`-helpall`
Values:	None
Default value:	None
Applicable family:	XC4000

r

The r option controls whether the output XNF file is compatible with schematics generated with the Xilinx Unified Libraries or with previous libraries.

Command line syntax:	`r={true|false}`
Values:	`true`, `false`
Default value:	`false`
Applicable family:	XC4000

By default, HM2RPM generates an XNF file that is compatible with designs from the schematic libraries created before the Xilinx Unified Libraries. If you select r=true, HM2RPM generates an XNF file that is compatible with the Unified Libraries. If you select r=false, it generates an XNF file compatible with previous libraries. The default value for this option is False.

Error Messages

HM2RPM can issue the following error messages.

Error 12601 **Cannot find file *name*.hm.**

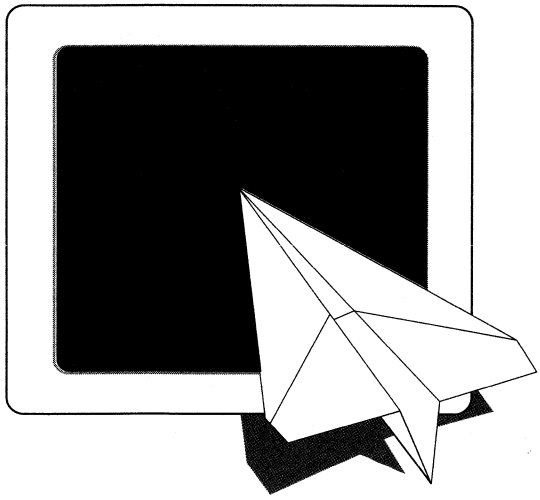
HM2RPM cannot find the *filename*.hm input file.

Error 12602 **Cannot open file *name*.hm.**

HM2RPM cannot open the *filename*.hm input file.

Error 12603 **The hard macro file *name*.hm has been corrupted and cannot be properly translated.**

HM2RPM detected an error in the *name*.hm hard macro file, which cannot be translated to an RPM.



XACT Reference Guide, Volume 1

Index

XACT Reference Guide, Volume 1 — April, 1994 (0401129 01)

Printed in U.S.A.

Index

O

P

R

S

T

U

V

W

X