Integrated Development System - Figaro Tutorial

June 2002

The Tutorial

Welcome to the *IDS Tutorial*. This manual provides detailed examples on working with Figaro, all supported interfaces, and the AT40K/AT6K architecture. It is intended to help users explore the system's full potential, so they can design Field Programmable Gate Arrays (FPGAs) in the most efficient and cost-effective manner.

Conventions Used in This Manual

The following typographical conventions are used in this guide:

- File names, and program names are in Helvetica type, e.g. atmel.ini, PLA2Cdb
- Variables are in *italics*, e.g. *DesignName*.lib
- Text to be entered in input boxes are enclosed in " ", e.g. "4bitalu"
- Italic text is used for names of buttons on the Flow Bars, e.g. *Open*
- Keyboard functions are shown as <*Key*>, e.g. <*Enter*>

Customer Service

Assistance with any matter related to the IDS can be obtained by the following methods:

Calling the PSLI Hotline at 408-436-4119 between 9 am and 5 p.m., Pacific time.

Sending an electronic mail with your question to fpga@atmel.com.

About Figaro and this Guide

Function

Figaro implements your design for downloading to an FPGA. It can:

- Read in Viewlogic or EDIF netlists.
- Map to retarget designs to the Atmel architecture.
- Let you select the target FPGA device(s), then partitions logic between these.
- Automatically places and routes the design, producing an optimized result for the target device architecture.
- Outputs either a bitstream for programming an FPGA, or a macro for adding to a user macro library.

In addition, Figaro can:

- Read in macros from user macro libraries.
- Check the timing across the device, display the delays on signal nets and produce timing reports.
- Output back-annotation files for the Viewlogic, Verilog-XL, VHDL and Mentor simulators.

Operation

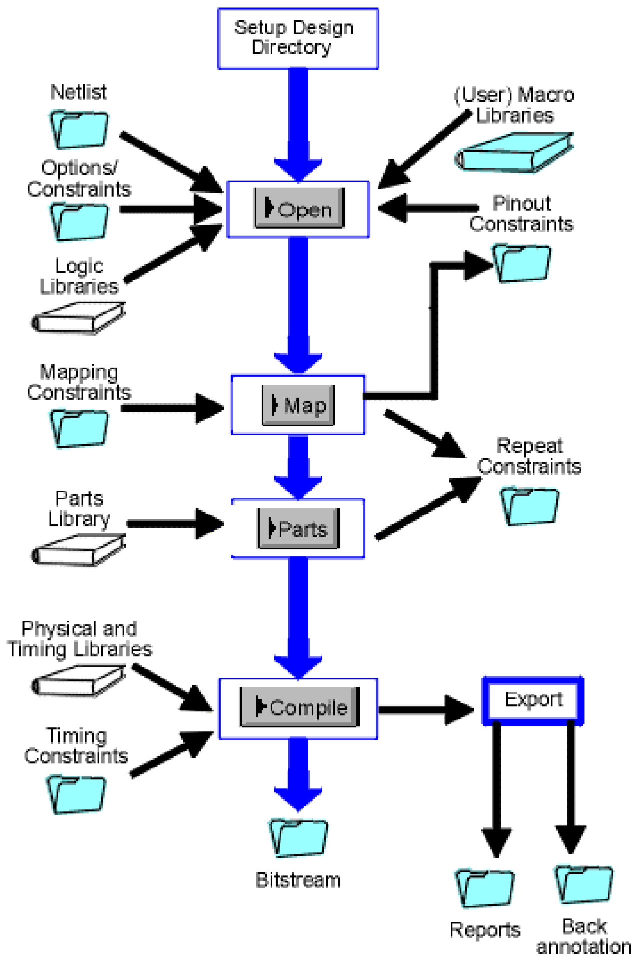
You can either leave Figaro to run automatically, or intervene to perform the following operations manually:

- Read in mapping, partition, pinout and timing constraints.
- Assign I/O functions to specific pins.
- Optimize the placement and routing.
- Output timing reports.
- Define user macros to build your own macro library.

The Figaro Flows

The product of Figaro is either a *bitstream* or a *user macro*. When you import a design, you have to specify which of these you want as the end result. Your choice affects your final product and both the way in which Figaro operates and some of the functions available during the session.

The diagram below summarizes the *bitstream flow*, with Figaro's Flowbar buttons pictured:



The Figaro Bitstream Flow

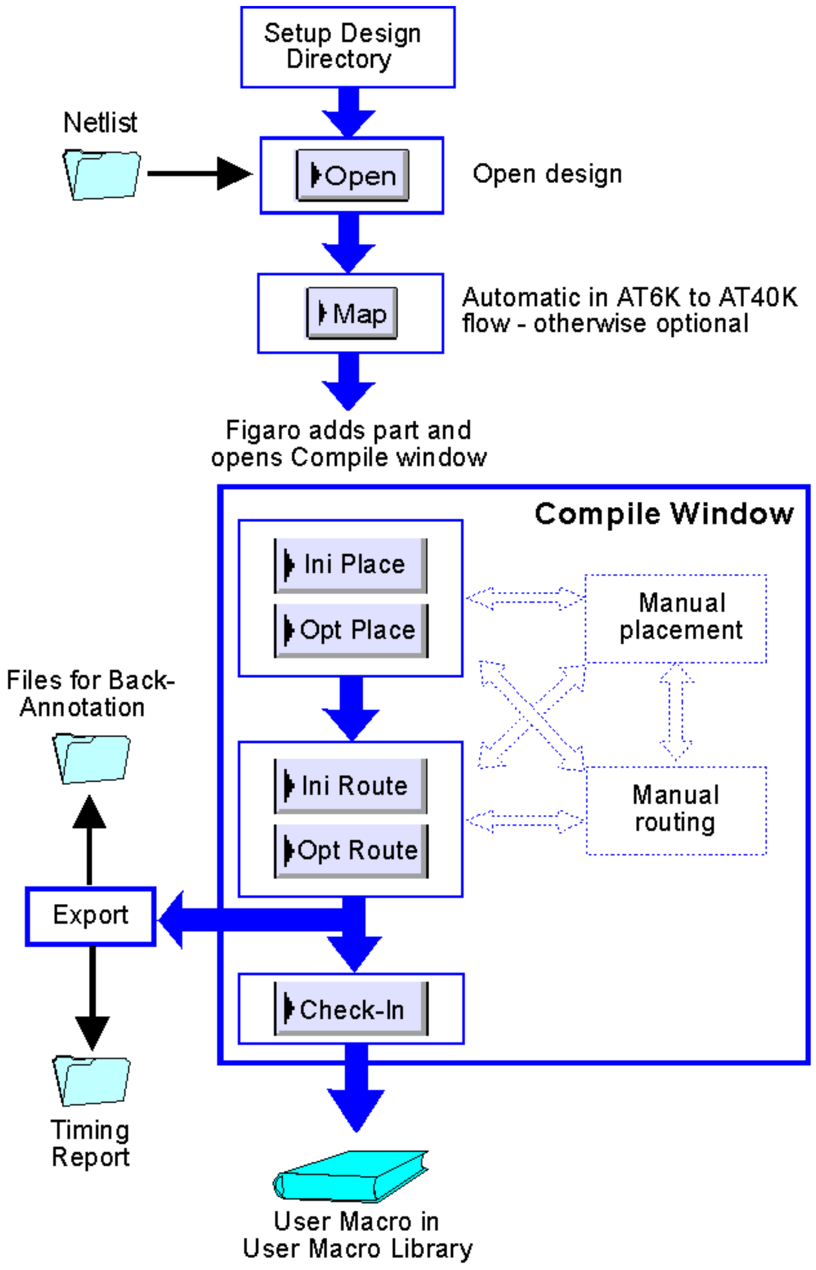
The Bistream flow consists of four easy steps: Set Up a Design Directory, *Open*, *Parts* and *Compile*.

When Figaro needs information, a dialog appears asking you to supply it. Toolbar buttons and menu commands let you perform more specialized tasks.

You can add up to 30 parts as required, then partition logic between these manually or automatically. Partition, like mapping and pinout, can be controlled using a constraints file.

In the diagram, the information shown on either side of the flow buttons represents other Figaro inputs/outputs: some of these are optional while others are used automatically.

The *user macro flow* is illustrated in the picture below:



The Figaro User Macro Flow

There are two main differences from the bitstream flow:

- Figaro adds a suitable part for you, so there's no need to press the *Parts* button.
- Rather than compiling the design automatically, Figaro opens a Compile window in which you can optimize your design manually. The diagram shows the five buttons available in the Compile window in the user macro flow: you can combine the use of the four Place and Route buttons with manual placement and routing functions.

The reason for producing your own macros is that when you use a macro in another design, you get the benefits of the manual optimization. To make proper use of this, you need a good knowledge of the device architecture.

Figaro Tools

Figaro has four main interactive tools:

Design Browser	The Design Browser shows the logic instances in the design imported into Figaro. For hierarchical designs, the hierarchy of instances is preserved within the Design Browser display. The Design Browser indicates any item that has been partitioned into a part with an unfilled rectangle inside the box that represents it. If an instance has been placed, the rectangle is filled.
Parts Window	The Parts window shows the target FPGA device package. When compiling for a bitstream, use the Parts window to select the target device(s) and, if necessary, to lock the pinout. You also partition logic between devices in this window. If you are compiling for a macro, Figaro just adds a suitable device for you and proceeds to the fourth tool, the Compile window.
Map Browser	The Map Browser shows the logic elements and hierarchy of your design, taking account of changes made to the original design during mapping. (The Design Browser always shows the pre-mapping design.) These changes are evident in the different names of associated items in the two browsers. The Map Browser uses the same graphic conventions as the Design Browser in representing partitioned and placed instances.
Compile Window	Compilation is the combination of placement and routing. The Compile window shows the architecture of the target device and the results of compilation: the placement of design logic in the device and the routing used by nets. As the last step in bitstream compilation, you output a bitstream file for each device for later downloading it to an FPGA. In a user macro compilation, you check the routed macro into a user macro library for use in other designs. When you want to optimize the placement and routing manually, rather than leaving it to Figaro, use the Compile window. You can change Figaro's placement and/or routing. When you change routing, the window switches into Manual Routing. If you import timing constraints for your design, you can look at the delays on signal nets. The Compile window switches into Timing Analysis mode.

Using this Manual

In this tutorial you will learn how to use Figaro to implement designs for FPGAs. This guide assumes that you already know how to describe circuits or produce schematic designs for FPGAs.

This manual has the following sections:

Getting Started	Shows how to start Figaro, then describes the user interface and shows how to use the on-line help. This section also shows how to import your design.
Quick Implementation	Describes how to implement a design in a few easy steps using Figaro's automatic tools.
Timing-Driven Implementation	Takes you through the implementation of a complex design, this time using many more of Figaro's functions. You look at the design with the Design Browser, cross-highlight between windows and find objects in the placed and routed design. Finally, you output timing reports and files for back-annotation to a simulator.
Manual Editing and User Macros	Illustrates manual placement and routing techniques and shows you how to produce user macros, check them into and out of libraries and incorporate them in a larger design.
Mapping	Provides a brief introduction to the principles used in mapping design logic to a specific target device architecture. Includes a look at the effects of the various options on the final mapped design.
Mapping AT6K Designs to AT40K	An introduction to using the "AT6K Mapped to AT40K" configuration.
Partitioning	Provides details on Figaro's functions for partitioning design logic between different devices.
Additional Features	Provides information on some useful tasks not covered in other tutorials, such as saving and restarting a session or changing Figaro options.
Troubleshooting and Support	Describes problems you may encounter and shows how to contact your technical support.

The appendices contain additional information:

Appendix A	Summarizes the menu commands and keyboard or button shortcuts.
Appendix B	Describes the files Figaro uses or outputs. These include the .INI files, the log file, timing reports, constraint files and statistics files.

Getting Started

This chapter tells you:

- How to start up Figaro.
- How to use on-line help.
- About the user interface.

  This briefly describes the parts of the Figaro window: menus, buttons and tool windows and so on

Starting Up Figaro

From the Start Menu, choose *Programs>Atmel>IDS 7.5*. This starts up Figaro, briefly displaying the "Copyright" box.

Using On-line Help

If you're running Figaro, pull down the command *Help>Figaro Overview*.

Here are a few tips to help you use the online help:

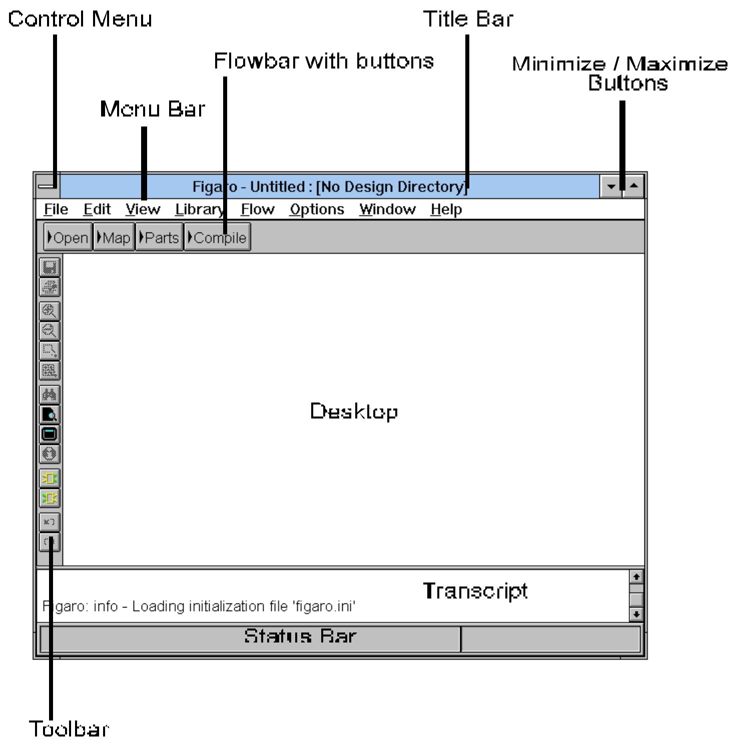
- Remember that the help is context-sensitive. Use the command *Help>Using the Current Window* to display help on the tool you're running.
- Keep the help window just wide enough for its six buttons to fit on one line. This leaves space for you to look at Figaro and the help page simultaneously.
- Most help topics fit on a single page, so there's no need to scroll through masses of text. To reduce the need to scroll, make the help window as deep as the screen itself.
- Click on phrases shown in underlined and/or colored text. This displays more information either as an overlay or a new page.
- Many diagrams in the help show additional information when you click on specific areas of them.
- Return to your previous page by clicking *Back*.
- For a description of any menu command, pull the menu down, click-hold the command and press SHIFT+F1 keys together.
- Use help's Search facility to find information on a topic quickly. Click on the *Search* button at the top of the help screen, then specify the topic you want. All significant areas are listed.
- Use the "Browse" buttons marked << and >> to move backwards and forwards quickly through a series of related topics.

The User Interface

When you start Figaro for the first time, a single window appears. This is the "Figaro window". Most of this window is empty when you first start up. This blank area is called the Figaro desktop.

The Figaro Window

After starting up Figaro for the first time, take a couple of minutes to examine the Figaro window shown as follows:


Control Menu
Title Bar
Flowbar with buttons
Minimize / Maximize Buttons
Menu Bar
Figaro - Untitled : [No Design Directory]
File Edit View Library Flow Options Window Help
Open Map Parts Compile
Desktop
Transcript
Status Bar
Toolbar

Figaro: info - Loading initialization file 'figaro.ini'

The Figaro Window at Startup

Figaro Window Features

The main features of the Figaro window are:

Title Bar	This contains the word Figaro and the design name. Before you open a design, the name is "Untitled".
Menu Bar	The menus run from *File* to *Help*. These "pull-down" menus let you perform all operations. The toolbar buttons (see below) are a quick way of performing the three main operations. To display the list of options under a menu, click on the menu name. To run an option, click on it in this list, or press the key combination, if one is shown next to it. To get help on an option, click-hold it and press the SHIFT and F1 keys simultaneously. Options are "grayed out" when they are not available or do not apply. The range of menus available changes as you open up Figaro tools. The *Compile* menu only becomes available when you move into the Compile window, for example.
Flowbar	This bar is located below the pull-down menus. It contains four buttons in a horizontal row. You can turn the Flowbar display on or off: pull down *Options>Flowbars*. The Flowbar buttons let you perform the main Figaro operations without using menu options. These buttons change color as you use them: see the on-line help for details.
Toolbar	This vertical column of buttons is located at the left of the window. You can turn this toolbar display on or off using *Options> Toolbars*. For quick information on a button, move the cursor over it and look at the status bar. For more details, *Search* for Toolbar in the on-line help.
Desktop	When you open a Figaro tool, its window overlays the desktop. You can change the size of the desktop relative to the transcript by dragging the boundary between them up or down.
Transcript	This area shows messages describing Figaro's progress. Errors and warnings are shown in color. You can scroll back to see messages from earlier in a Figaro session. As the session goes on, early messages are removed, although they do remain in the log file, described in Appendix B. You can turn the transcript display on or off using *Options>Transcript*. You can resize the transcript vertically using the mouse.
Status Bar	This is located at the bottom of the screen. It is blank when you first start up. If you click-hold the cursor on a menu option, this area shows what the option does. As you move the cursor around Figaro the status bar displays various explanatory texts (for example on pin names and types, grid locations) and useful hints (for example on toolbar and Flowbar buttons). It also incorporates a percentage complete bar, enabling you to monitor the progress of the operations being performed.

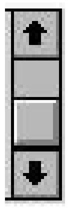
The representation of the following three buttons may differ depending on the platform you are using

Control menu	Click on the button to display commands which let you move or resize the window, reduce it to an icon, or close it.
Minimize Button	Click to reduce the window to an icon.
Maximize Button	Click to expand the Figaro window to fill the screen, or a tool window to fill the desktop.

Desktop Options

You can turn on/off the display of the Flowbar, toolbar, transcript and status bar by pulling down *Options>Display Options* and selecting “Desktop”. Check the boxes to set items on or off. This is different to the commands under the View menu because the settings become the defaults for use in future sessions.

Scroll bars



Scroll bars are not shown in the above diagram. They may appear at the right side or base of a window, depending on its size and contents. Use scroll bars to move horizontally or vertically in the window: either drag the square along the bar or click on the arrows at the ends of the bar.

Tool Windows

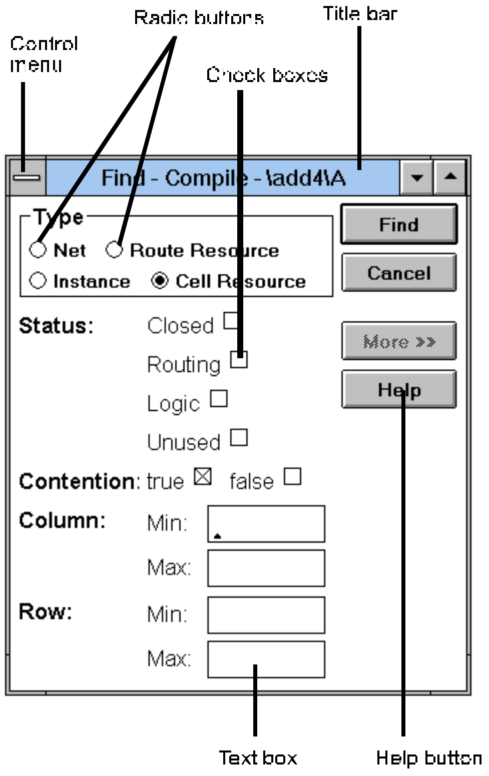
The tool windows have their own control menus, title bars and minimize/maximize buttons. These operate in the same way as for the Figaro window.

Tool windows do not have a separate transcript or status bar. They do not have a separate menu bar: the menu options on the Figaro window menu bar change to reflect the tool window you are in.

The Parts window has its own Flowbar with specialized buttons. The Compile window has its own Flowbar and toolbar. The toolbar buttons provide shortcuts for common Compile window operations. (For more details, *Search* for Toolbar in the on-line help.)

Dialog Boxes

A dialog box appears to ask you to enter information or select something. This example shows the main features:



Dialog Box Features

Multiple and "Lost" Dialogs

When you use some commands, for example *Edit>Find* and *Edit>Info*, you may have to open a series of dialogs, each showing different information. If you then click on the desktop again, these dialogs may disappear behind the Figaro window. To bring them back into view, use the command *Window>Dialogs to Front*.

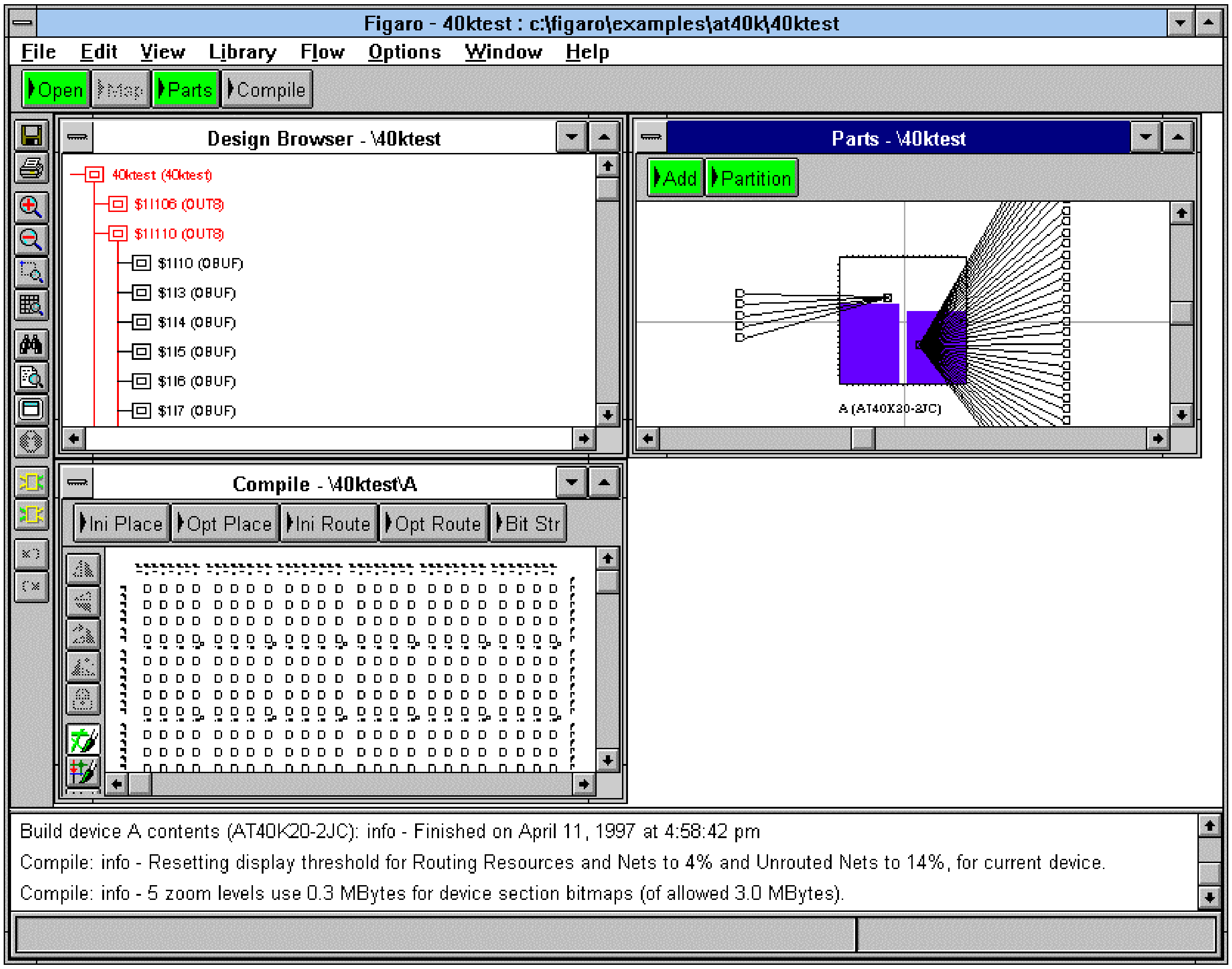
To close all open dialogs in one step, use the command *Window>Close Dialogs*.

Running Multiple Windows

When you open any Figaro tool, a new window appears, overlaying the Figaro desktop.

After a while you may find you have three or four windows open at once. If you have several windows open, but are only working in one of them, you can make more space for it by reducing the other windows to icons. To do this for any window, click on its *Minimize* button. The icons remain within the global Figaro window.

You can arrange the windows or icons in several ways, using the *Arrange Icons*, *Cascade* and *Tile* options under the *Window* pull-down menu. A display with three windows tiled, for example, looks like this:



Build device A contents (AT40K20-2JC): info - Finished on April 11, 1997 at 4:58:42 pm
Compile: info - Resetting display threshold for Routing Resources and Nets to 4% and Unrouted Nets to 14%, for current device.
Compile: info - 5 zoom levels use 0.3 MBytes for device section bitmaps (of allowed 3.0 MBytes).

Tool Windows Tiled

Resizing Windows

There are two ways to make a window the exact size you want. Click in the window to make it the current window, then:

- Move the mouse to the window edge, where it changes to a "↔" shape. Holding down the left mouse button, drag the window boundary to the new location and release the button.
- Move the mouse to a window corner, where it changes to a "corner" shape (for example, in the bottom right-hand corner). Holding down the left mouse button, drag the window boundary to the new location and release the button.

The Cursor

The shape of the cursor is usually an arrow, but changes, depending on the task being performed. For more information, open the on-line help and *Search* for "cursor".

If the cursor changes to a trashcan or hourglass shape, this means Figaro is compacting memory or just processing. Do not abort the operation.

Colors

Colors tell you a lot about a window. For example, an object changes color when you click on it in a window to show that it's been selected.

You can change the colors of many features shown in the windows. To change default color settings for a particular window, pull down *Options>Display Option*s. Don't change colors yet: wait until you've run through the exercises in the next chapter, so you can assess the default colors Figaro starts with. (Instructions on changing colors are given in "Changing Figaro Options" in Chapter 9, and in the on-line help.)

NOTE All references to colors in this manual assume you are using the default colors. If the defaults for your system have been modified by another user, you can reset all colors to their initial defaults by renaming the file FIGARO.INI stored in the directory from which Figaro was invoked (usually Figaro's BIN directory).

Other Types of Window

You can open two types of Window which are independent of the Figaro desktop.

The Log Viewer

You can use *Window>New Viewer>Log File* to open a separate Log Viewer window containing all the session's log messages. This window is independent of other Figaro windows. You can

- Update the display to show messages output by Figaro since you last used *Update*.
- Print the log.
- Search the log for a string. This displays a pop-up box: type in the text you're looking for and press ENTER.
- Save the log to a file.

To move within the Log Viewer window, use the scroll bars, arrow keys and Page-Up and Page-Down keys.

There are also similar viewers that you can use to look at path analysis reports, net delay tables and statistics files.

A Quick Implementation

This first tutorial shows you how to implement a design with a minimum of effort, leaving almost everything to Figaro.

All Screenshots in this tutorial only represent AT40K examples.

The steps are:

1. Preparing Input to Figaro
2. Setting Up the Design Directory
3. Opening the Design
4. Specifying the Target Device
5. Compiling the Design
6. Looking at the Implementation
7. Saving the Design and Exiting Figaro

Work through all the sections in the sequence given. This tutorial uses the buttons on the Figaro window Flowbar. If this is not visible, display it by pulling down *Options>Flowbars*.

If Figaro is already open, you can go straight to the next section. If not, repeat the process described under "Starting Up Figaro" in the previous chapter.

Preparing Input to Figaro

As well as its own saved design sessions (*.FGD files), Figaro also accepts netlists in various other formats as input. You probably already have netlists for your own designs, produced using a schematic editor, but run the tutorials in this manual first using the examples provided.

For AT40K devices: The example used here is a Viewlogic design called TEST40K, which has been installed automatically in /SYSTEMDESIGNER/EXAMPLES\AT40K\VLOGIC\TEST40K under your main Figaro directory.

For AT6K devices: The example used here is a Viewlogic design 4BITALU, which has been installed automatically in /SYSTEMDESIGNER/EXAMPLES\VLOGIC\4BITALU under your main Figaro directory.

NOTE

When you come to import your own designs, and you want details on the interface between Figaro and a particular schematic editor, refer to the section *Interfaces to Figaro*. This details netlist formats, the design checks Figaro runs and other information.

Before You Begin

Figaro provides various time-saving features designed to save you having to re-enter the same standard information each time you work with a particular design. However, this tutorial assumes that you are working with the system for the first time, so before you begin, please check the following:

The 4BITALU.RCT/TEST40K.RCT files

If other users on your system have used the Figaro tutorials already, check whether there is a file named 4BITALU.RCT or TEST40K.RCT in the design directory.

This file contains *repeat constraints* saved from previous Figaro sessions with this design and, if the "Auto-import Repeat Constraints" option is enabled, Figaro will import this file and automatically add the part and lock any pins it specifies.

You will find out more about this file in later tutorials and in Appendix B, "Figaro Files": for now, delete or rename it before starting the tutorial yourself.

The Viewlogic symbol file

The Viewlogic symbol file for this example may specify part and package attributes to be used in the implementation of the design. Figaro uses these attributes to add the appropriate part from the AT40K library automatically when the design is imported.

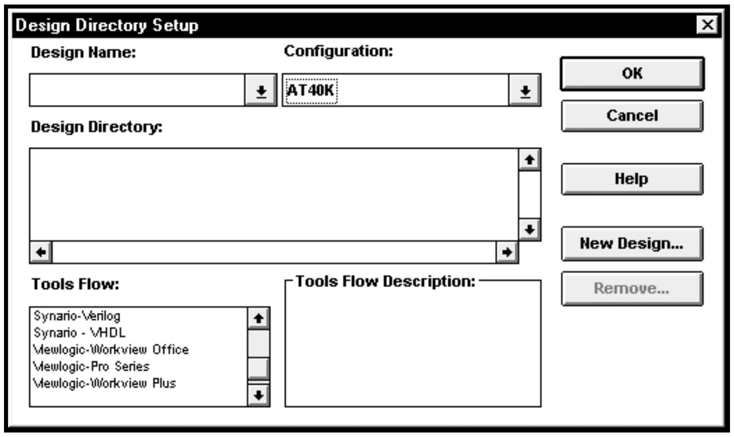
However, in this tutorial you should add the part manually, so pull down *Options>Options*, select the topic "Viewlogic Import" and clear the check box labeled "Use Part/Package Attributes".

Unconnected Reset Ports

The TEST40K design includes a number of unconnected reset ports, which by default are reported on import as errors by the design checker. To be able to read in the design, you will have to disable these unconnected resets. Pull down *Options>Options*, select "Design Checker" from the topic list and then "Disable" for the "Reset Ports" option.

Setting Up the Design Directory

You must set up a design directory before you can open a design. The design directory is the only place Figaro will look for the design. Pull down *File>Design Setup*. The Design Directory Setup dialog is displayed:



Design Name:

Configuration: AT40K

Design Directory:

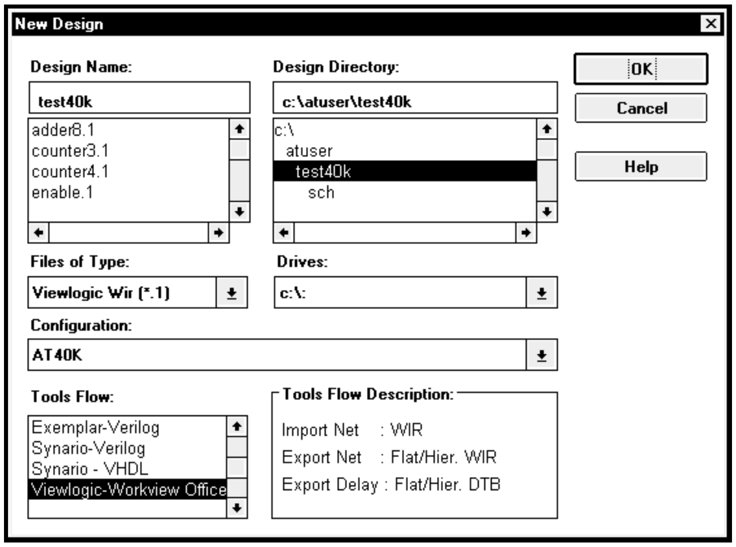
Tools Flow:
Synario-Verilog
Synario - VHDL
Viewlogic-Workview Office
Viewlogic-Pro Series
Viewlogic-Workview Plus

Tools Flow Description:

Design Directory Setup Dialog Box

If this is the first time you've opened Figaro since installation, no directory will be listed. Click the *New Design* button to add the directory which contains the supplied 4BITALU or TEST40K example.

The New Design dialog box is displayed:



Design Name: test40k
adder8.1
counter3.1
counter4.1
enable.1

Design Directory: c:\atuser\test40k
c:\
atuser
test40k
sch

Files of Type: Viewlogic Wir (*.1)

Drives: c:\:

Configuration: AT40K

Tools Flow:
Exemplar-Verilog
Synario-Verilog
Synario - VHDL
Viewlogic-Workview Office

Tools Flow Description:
Import Net : WIR
Export Net : Flat/Hier. WIR
Export Delay : Flat/Hier. DTB

New Design Dialog Box

Do the following:

1. If you are using a PC and have installed the examples on a different drive to that shown under "Drives", select that drive in the pull-down list.
2. Under "Design Directory" specify the directory /SYSTEMDESIGNER/EXAMPLES\VLOGIC\4BITALU for 6K devices or /SYSTEMDESIGNER/EXAMPLES\AT40K\VLOGIC\TEST40K for 40K devices. To do this, click on successive directories in the path name: each time you do this, the relevant subdirectories are shown. Use the scroll bar if necessary.
3. Under "Files of Type", select the type for the schematic editor used: for this example this is "Viewlogic Wir (*.1)".

   When the "Design Name" list displays all the files in the directory, select 4BITALU.1 or TEST40K.1.
4. Under "Tools Flow" select "Viewlogic WIR". These are the systems used to produce the example design on the different platforms.
5. Under "Configuration", select AT6K or AT40K.
6. Click *OK*. This returns you to the first dialog, where /SYSTEMDESIGNER/EXAMPLES\VLOGIC\4BITALU or /SYSTEMDESIGNER/EXAMPLES\AT40K\VLOGIC\TEST40K is listed under "Design Directory" and 4BITALU or TEST40K under "Design Name". Click *OK* to finish design directory setup.

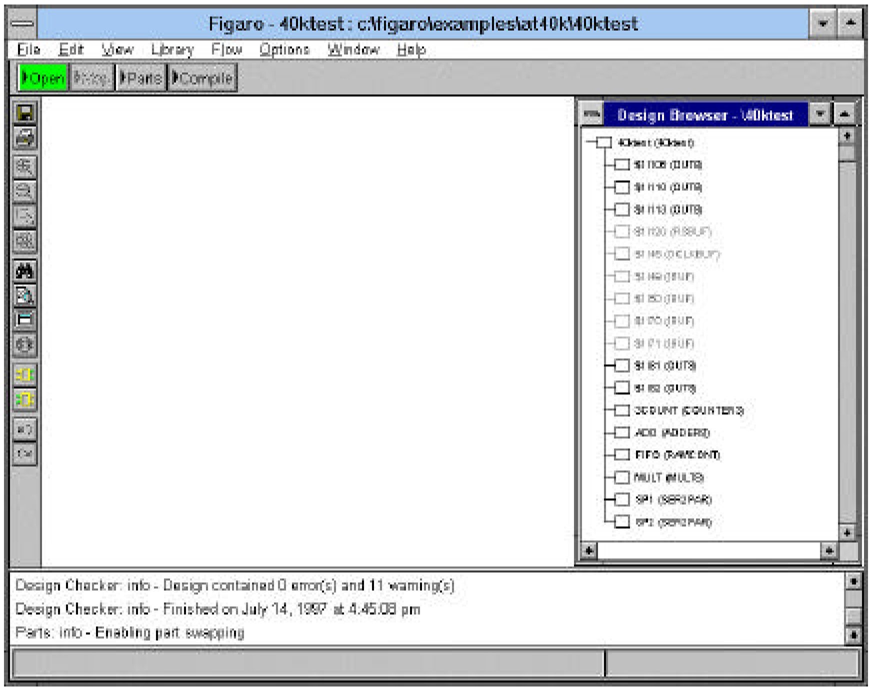
Opening the Design

To read in the design, do the following:

1. Click the *Open* button on the Flowbar. The button turns yellow to show you've started opening a design.
2. You are asked whether you want to open the netlist as a design or a macro. Click *Design* to enter *compile for bitstream* mode, meaning that the end result will be a bitstream file rather than a user macro. Figaro displays the Open as Design dialog.
3. Make sure "Files of Type" is set to "Viewlogic Wir (*.1)". The directory and file names will change to the ones you specified.
4. Click *OK* to load the design. This causes the following to happen:

- Figaro writes messages to the transcript showing the design is being opened (these may include warnings from the Design Checker, depending on how the options are set).
- A Design Browser is opened on the Figaro desktop. This lists the logic elements in the netlist(s).
- The *Open* button turns green, showing that the design opened successfully. (If an error did occur, the button will turn red and the transcript will provide information on the reason.)

The display looks like this:



```
Design Checker: info - Design contained 0 error(s) and 11 warning(s)
Design Checker: info - Finished on July 14, 1997 at 4:45:08 pm
Parts: info - Enabling part swapping
```

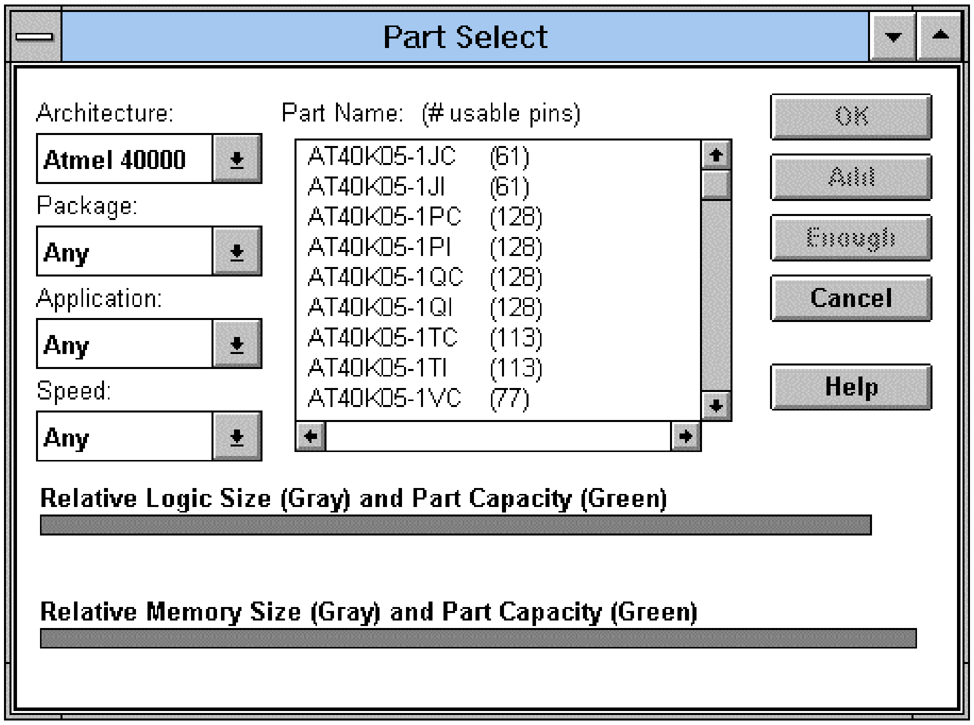
Figaro Window with Opened Design

Note that this flow does not include the *Map* step, so the corresponding Flowbar button is grayed out. The button is enabled and disabled using the "Mapping" options in the Options dialog.

Specifying the Target Device

The Parts window is a representation of the PCB that holds the FPGA. It is a virtual board to let you visualize the orientation of the part and you use it to specify the target device, which is then displayed in the window.

To select the target device (called a "part" by Figaro), click the *Parts* button in the Flowbar. This displays an empty Parts window and the Part Select dialog:



The Part Select Dialog Box

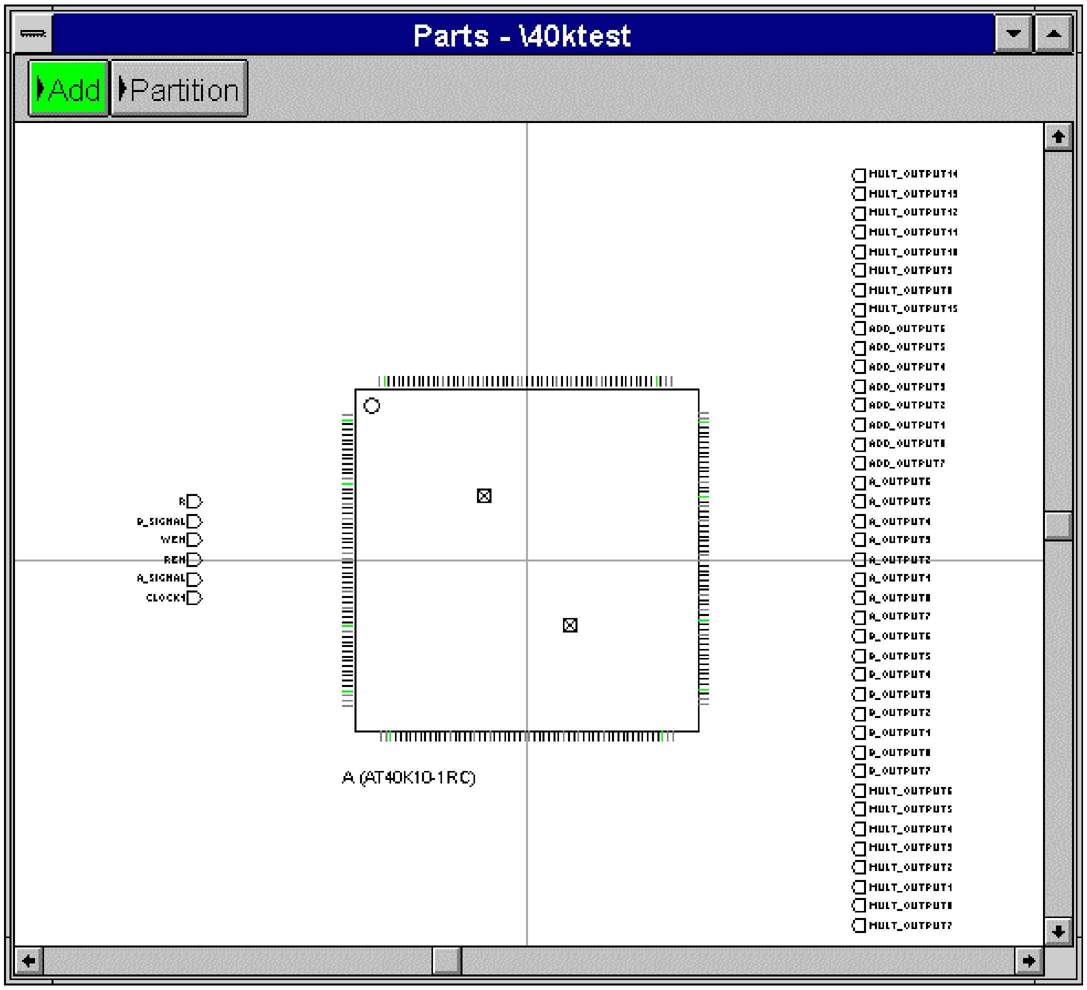
The Package, Application and Speed lists filter out unwanted device types from the list. You can, for example, list just those devices suitable for commercial applications.

NOTE

You cannot change the architecture in this dialog: if you want to use an AT6K part in your implementation, you have to return to the Design Directory Setup dialog. You cannot mix AT6K and AT40K parts in a single implementation.

To add a part, do the following:

1. Scroll down the "Part Name" list and select "AT6003-2JC" or "AT40K05-2JC".
2. A green bar appears below the gray bar at the base of the dialog, showing whether the part is large enough to hold the design. One part will be large enough for 4BITALU or TEST40K.
3. Click *Add* to add one of these parts and then *OK* to return to the Parts window. An image of the selected part appears in the Parts window, which becomes the current window:



Parts Window Displaying Target Device

The part is labeled "A" and is followed by its name. Clock and reset pin locks may be assigned automatically.

Compiling the Design

Click on the *Compile* button to implement the design on the part that you just added. As compilation proceeds, look at the progress reports in the status bar and transcript to see what Figaro is doing.

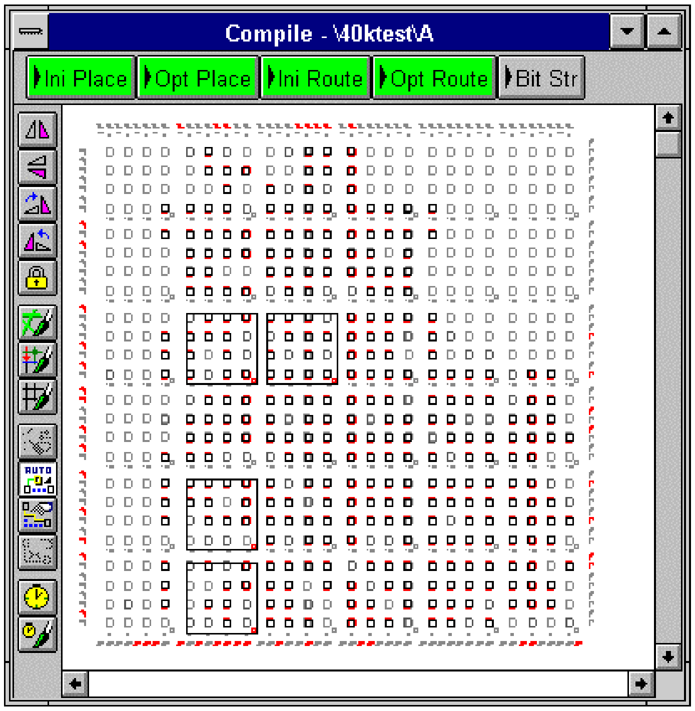
Compilation for bitstream involves the following steps:

Partitioning	As only one part was added, this involves moving all the design instances to that part.
Placement	This step positions the design instances in logic blocks in the device, producing an optimized result for the device architecture. Placement is run in three stages: initial placement then two optimize placement passes (global and detailed).
Routing	Makes connections between logic blocks using available routing resources on the device. Optimal routes are used. Routing is also run in three stages, two initial routing passes then optimized routing. In non-timing-driven implementations like this one, optimized routing runs only if initial routing leaves routing contention.
Bitstreaming	Outputs a bitstream for downloading to an FPGA. The file containing the bitstream is written to the design directory and is called *design*.BST.

When compilation finishes the *Compile* button turns green, indicating that the run was successful.

Looking at the Implementation

To see the result of compilation, pull down the menu *Window>New Compile Window*. This opens a window like that shown below depicting the device architecture. The compiled design has its logic placed and routes completed:



The Compile Window after Compilation

Larger boxes covering more than one sector are RAM blocks. Use *View>Zoom to Area* to take a closer look. Don't worry if your implementation of the design does not match the picture exactly: continuous improvements to Figaro's placement and routing algorithms may mean the pattern is different.

In the next tutorial you'll look at the Compile window in more detail, including its toolbar and the meaning of the various colors used.

Saving the Design and Exiting Figaro

To save the implementation under the original design name, simply pull down *File>Save*. Figaro saves the session in a file called *design*.FGD, which you can reload at a later time.

To save the session under a name other than the original design name, pull down *File>Save As* and enter the new name in the field provided.

If you don't intend to start the next tutorial immediately, save the session using either of the above methods and then pull down *File>Exit*. When you are asked if you really want to exit Figaro, click *Yes*.

Summary

In this tutorial you clicked the *Open*, *Parts* and *Compile* buttons in turn (*Map* was disabled). You could have implemented the design just by clicking the *Compile* button on the Figaro window's toolbar instead: this would have run the *Open* and *Parts* steps too, with Figaro prompting you for input as required.

The following tutorials look in greater detail at the steps introduced here.

Timing-Driven Implementation

This tutorial works through a process similar to that described previously, but this time you intervene more actively in the process. You compile the design in stages and learn how to set timing constraints that will drive the way Figaro implements the design. There are also extra steps between the main tasks of the previous exercise:

- Looking at the meaning of colors in the display.
- Cross-highlighting between the Browser and the Compile windows.
- Zooming in to look more closely at the device structure.
- Using *Edit>Find* to find named nets in the design.

These tasks are optional and can be skipped without affecting the tutorial flow. You should look at these sections at some stage however, as they contain valuable information on Figaro's user interface and search features.

The individual steps in this tutorial are:

1. Opening the Design and Adding a Part
2. Browsing the Hierarchy
3. Setting Up for Timing-Driven Design
4. Running Initial Placement
5. Analyzing Timing After Placement
6. Importing Timing Constraints
7. Running Optimize Placement
8. Routing
9. Interactive Timing Analysis
10. Bitstream and Export
11. Troubleshooting Timing-Driven Compilation

You can use the keyboard shortcuts shown to the right of commands in a menu. For a full list, see “Command Shortcuts” in Appendix A, or *Search* for "Key Shortcuts" in the on-line help.

Opening the Design and Adding a Part

Before you begin, make sure there are no repeat constraints from a previous design session in the design directory.

To read in the 4BITALU/TEST40K design, do the following:

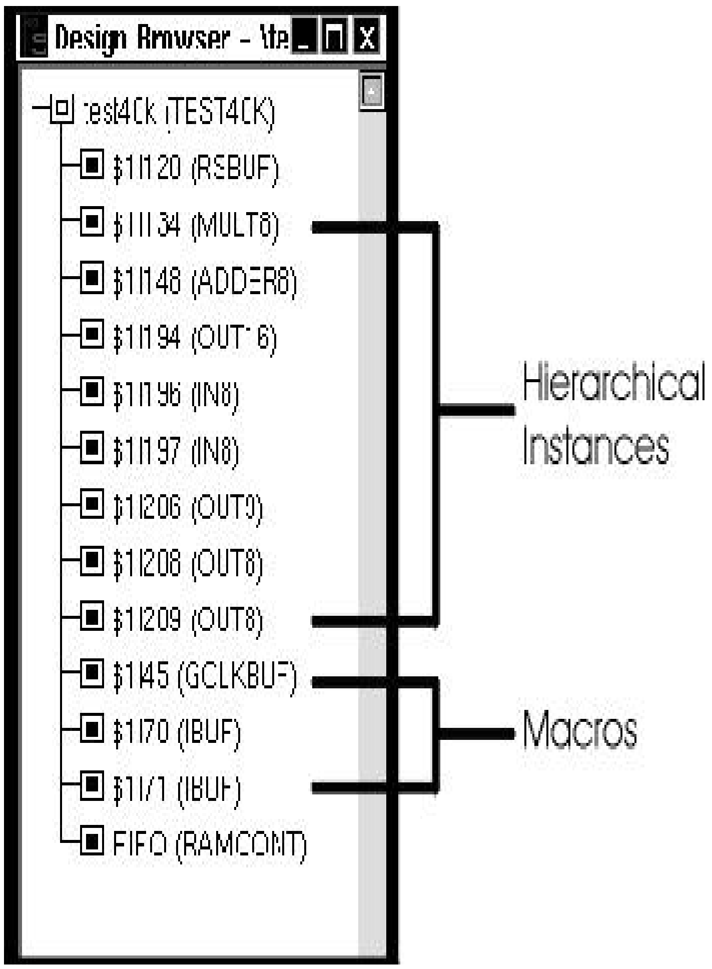
1. Pull down *File>Open As Design* to display the Open as Design dialog box.
2. Select \SYSTEMDESIGNER\EXAMPLES\VLOGIC\4BITALU or \SYSTEMDESIGNER\EXAMPLES\AT40K\VLOGIC\TEST40K from the pull down design directory list.

   If you need to set up the design directory first, refer to Chapter 3, "Setting Up a Design Directory".
2. Set "Files of Type" to "ViewLogic Wir (*.1)" and click *OK*. Figaro reads in the netlist and opens a Design Browser.
3. Click *Parts* to open the Parts window and use the Part Select dialog to add a single AT6003-2JC or AT40K05-2JC device.

   Use the green and gray bars at the bottom of the dialog to check whether the part is large enough to accommodate the design.
4. Click *OK* to close the dialog and return to the Parts window.
5. Go to the Parts window and click *Partition* to assign the design logic to the selected part. In AT40K, part fill is represented by two separate blocks, logic on the left and memory on the right.

Browsing the Hierarchy

First look at the Design Browser:



```
test4Ck (TEST4CK)
  $1I120 (RSBUF)
  $1I134 (MULT8)
  $1I148 (ADDER8)
  $1I194 (OUT16)
  $1I196 (IN8)
  $1I197 (IN8)
  $1I206 (OUT9)
  $1I208 (OUT8)
  $1I209 (OUT8)
  $1I45 (GCLKBUF)
  $1I70 (IBUF)
  $1I71 (IBUF)
  FIFO (RAMCONT)
```

The Design Browser

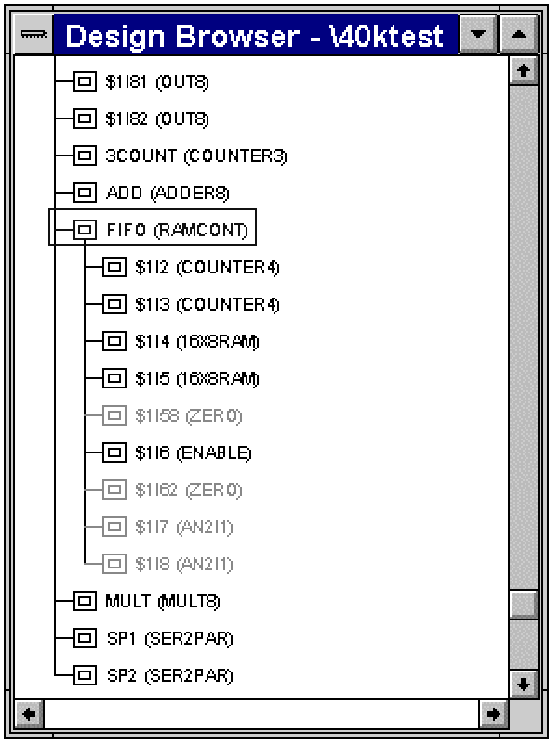
Note the following:

- The Design Browser shows the design hierarchy as a tree stru cture.
- Logic elements in the design are shown as labeled boxes and the hierarchical relationships between them as lines. Indentation is used to illustrate the hierarchical level.
- Hierarchical instances are represented in a different color from macro instances.
- The 4BITALU or TEST40K element at the top is a hierarchical instance (i.e. one that contains other instances), as are REGISTR4 and CONTROL for AT6K devices or MULT8 and OUT8 for AT40K devices.
- For macros, the ViewLogic instance and library names are shown. For example, for the tenth element in the picture shown above, these are (for AT40K Devices):

$1I45	Instance name from ViewLogic (i.e. the name given to the symbol in ViewLogic, or a label supplied by the engineer, e.g. "MULT").
GLCKBUF	ViewLogic symbol master name.

Expanding the Hierarchy

You can change the way the design is displayed in the Design Browser. Scroll down the list to find $1I72(CONTROL) (for AT6K Devices) or FIFO(RAMCONT) (for AT40K Devices) and click on it. When you select it, a box surrounds it and its color changes to the selection color. Pull down *View>Expand One Level* to display the next level of hierarchy for this element. You can also use double-click on the selected element as a quick method of expanding and contracting branches.



```
$1I81 (OUT8)
$1I82 (OUT8)
3COUNT (COUNTER3)
ADD (ADDER8)
FIFO (RAMCONT)
    $1I2 (COUNTER4)
    $1I3 (COUNTER4)
    $1I4 (16X8RAM)
    $1I5 (16X8RAM)
    $1I58 (ZERO)
    $1I6 (ENABLE)
    $1I62 (ZERO)
    $1I7 (AN2I1)
    $1I8 (AN2I1)
MULT (MULT8)
SP1 (SER2PAR)
SP2 (SER2PAR)
```

Expanded Branches in the Browser (for AT40K Devices)

Now select instance MULT8 and pull down *View>Expand Branch*. This time, Figaro expands **all** the sub-levels of hierarchy under the selected instance.

Use *View>Collapse Branch* to contract the branches again. Experiment by selecting other elements and expanding and contracting these using both double-click and menu commands until you're familiar with the Browser commands.

Setting Up for Timing-Driven Design

Enable the *timing-driven mode* by checking the appropriate box in the "Place and Route" options dialog. In timing-driven mode, Figaro seeks to optimize frequencies for all primary, secondary and derived clocks in the design. To enable it to do this, you need to specify the assertions against which you want the placement and routing functions to run:

Primary clock assertion	Defines a regular clock cycle for the design. You can define only one primary clock and it must be placed on an input to the design.
Secondary clock assertion	Defines a regular clock cycle referenced to the primary clock. Like primary clocks, these must be placed on inputs, but you can define as many secondary clocks as you require.
Derived clock assertion	Required wherever a primary or secondary clock signal passes through other logic on the path between the external input and the port driving the internal clock net. You need as many of these as there are primary or secondary clocks reaching the port.

]

If you've specified input and/or output assertions that take into account off-chip areas of a design, these will also be optimized:

Input assertion	Specifies a required time for arrival of an external input signal at a device I/O.
Output assertion	Specify a required arrival time for an internal signal going through an I/O out to the board.

NOTE Using timing driven compilation on a purely combinatorial design will not yield a better solution than non-timing-driven, as asynchronous delay assertions (like cut cycle and false path assertions) are not optimized by this function.

There are two main ways of specifying timing assertions:

- interactively, by using Figaro's Constraints Editor.
- by importing a timing constraints (*.TMG) file you produced earlier.

You'll use the former method in the course of this tutorial, but rather than enter your own assertions just now, the next section shows how you can have Figaro generate essential defaults using the Timing Analysis options.

NOTE If you want Figaro to generate default assertions, you must set these options *before* you open the Constraints Editor for the first time. If you open the Constraints Editor first, Figaro will not subsequently overwrite your entries, even if those "entries" are blank.

Before you Proceed

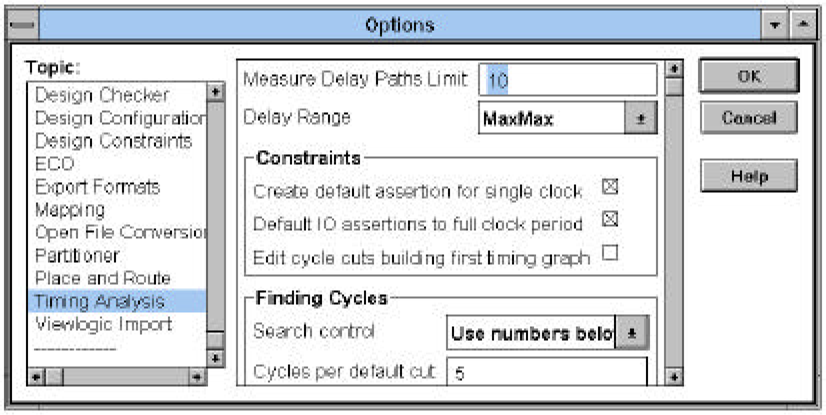
Before you create default assertions for your design:

- If you haven't already done so, pull down *Options>Options*, select "Place and Route" from the Topic list and enable timing-driven mode.

Creating Default Assertions in the Constraints Editor

To have Figaro generate essential timing assertions automatically, do the following:

1. Pull down *Options>Options* and select "Timing Analysis" from the Topic list. Figaro displays the Timing Analysis options:



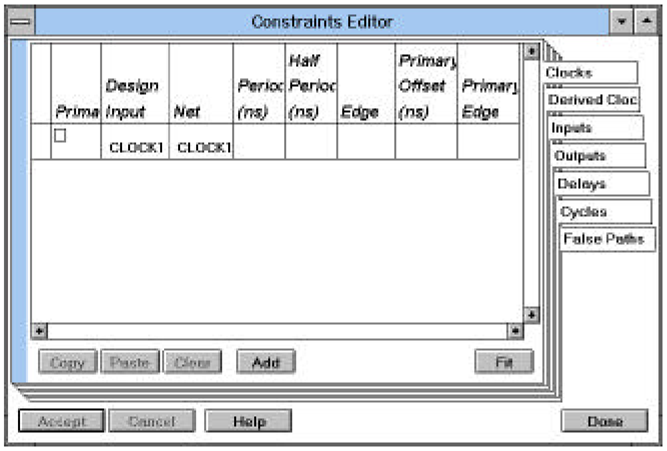
Measure Delay Paths Limit: 10
Delay Range: MaxMax
Create default assertion for single clock ☒
Default IO assertions to full clock period ☒
Edit cycle cuts building first timing graph ☐
Search control: Use numbers belo
Cycles per default cut: 5

Timing Analysis Options Dialog Box

2. Look at the first two options in the "Constraints" section: "Create default assertion for single clock" and "Default IO assertions to full clock period". Where a design contains only one clock, the first option will cause Figaro to place a default clock assertion of 1ns on that clock, unless you specify otherwise.

The second option controls whether Figaro creates default input and output assertions where these would be required by the design. Initially, these defaults are symbolic values, indicating that true values will be computed and entered when timing analysis is actually run.

3. Set the options as shown and click *OK* to return to the desktop.
4. Pull down *Edit>Timing Constraints* and look at the defaults Figaro has entered. On the "Clocks" page, Figaro has entered the CLOCK1 input and net:

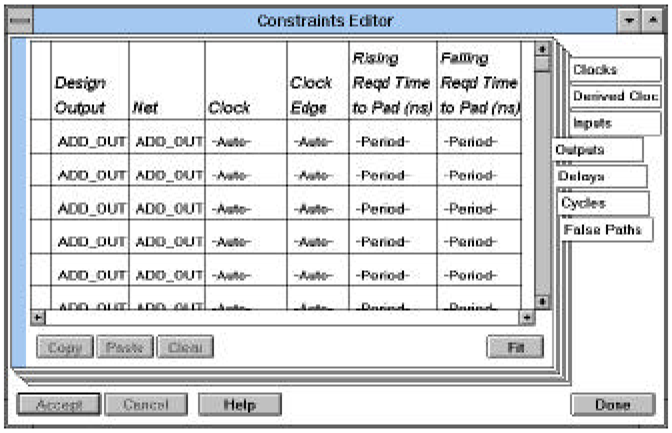


Prima	Design Input	Net	Perioc (ns)	Half Perioc (ns)	Edge	Primary Offset (ns)	Primary Edge
☐	CLOCK1	CLOCK1					

Constraints Editor

Figaro adds a 1 ns Period, 0.5 ns Half Period and Edge Rising. This will happen only if there is 1 clock in the system. If there are multiple clocks, or derived clocks, they will be shown in the dialog, but not be assigned values.

Now look at the "Inputs" and "Outputs" pages (the latter is shown below). Both include automatically generated entries that will be substituted with actual values when timing analysis is run. This will happen for single or multiple clocks:



Design Output	Net	Clock	Clock Edge	Rising Reqd Time to Pad (ns)	Falling Reqd Time to Pad (ns)
ADD_OUT	ADD_OUT	-Auto-	-Auto-	-Period-	-Period-
ADD_OUT	ADD_OUT	-Auto-	-Auto-	-Period-	-Period-
ADD_OUT	ADD_OUT	-Auto-	-Auto-	-Period-	-Period-
ADD_OUT	ADD_OUT	-Auto-	-Auto-	-Period-	-Period-
ADD_OUT	ADD_OUT	-Auto-	-Auto-	-Period-	-Period-
ADD_OUT	ADD_OUT	-Auto-	-Auto-	-Period-	-Period-

Default Output Constraints

5. When you have finished browsing, click *Done* to close the Constraints Editor without changing any of the default assertions.

Running Initial Placement

Placement positions design instances in the FPGA's logic locations. It is run in two distinct stages: initial placement and optimize placement. Before you start, ensure that timing-driven mode is enabled and that the quality level is set to 2 ("Place and Route" options).

The previous tutorial showed you how to use the flowbar buttons to compile your design, but this time you'll use the *Flow* menu to run the individual steps. This allows you to compile the design without opening a Compile window, which may be useful if your computer is close to the recommended minimum criteria required for Figaro.

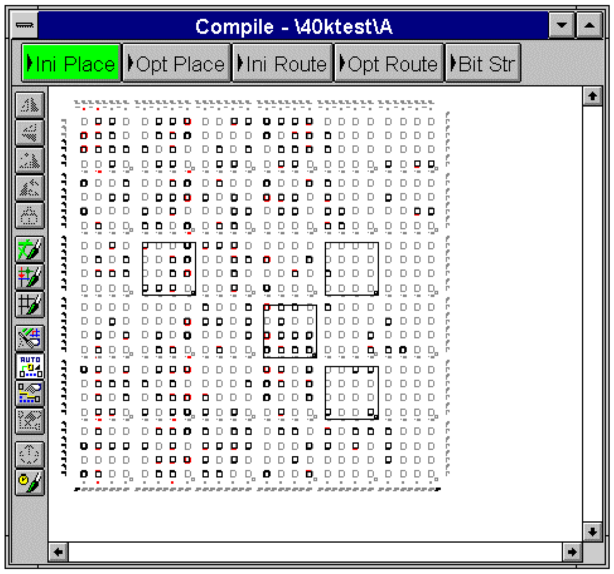
Pull down *Flow>Compile>Initial Placement* to perform initial placement.

NOTE If Figaro had not successfully generated default timing, it would produce an error message and ask you to enter the required constraints. .

Progress is displayed in the status bar at the bottom of the desktop and the *Compile* button in the Figaro flowbar turns yellow to show that compilation is in progress.

When initial placement is finished, view the transcript to see where the global clock and reset nets have been allocated. You'll also see that each of the boxes representing the instances in the Design Browser is filled with a solid rectangle to show it has been placed in the device. The *Compile* button in the Figaro window flowbar has gone back to gray.

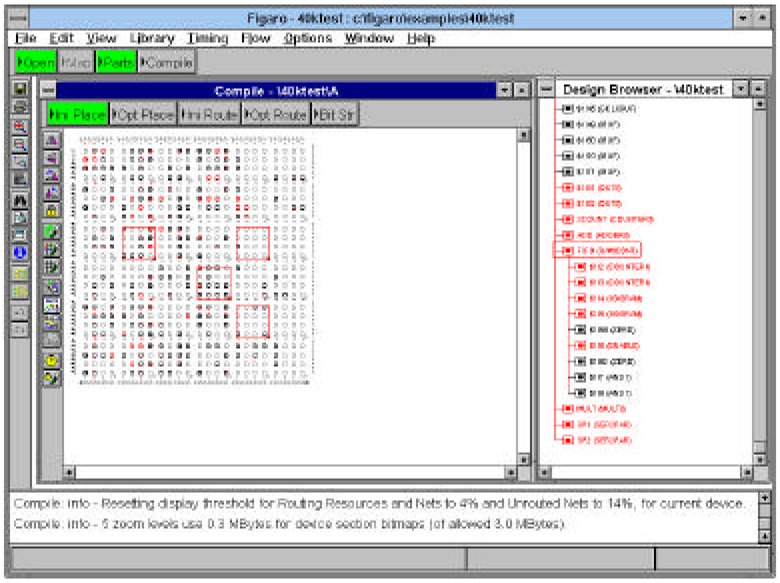
Open the Compile window using *Window>New Compile Window*:



The Device after Initial Placement

The Compile window also includes a flowbar with buttons for each of the main compile steps. Initially, all the flowbar buttons were gray, but now the *Ini Place* button is colored green to indicate that the step has been run successfully. As you compile the design in stages, detail is added and the other buttons also change color. The individual stages are described in turn below.

4. Expand the Compile window to fill the remaining desktop space. The Figaro window should now look something like this:


Figaro - 40ktest : c:\figaro\examples\40ktest
File Edit View Library Timing Flow Options Window Help
Open
Map
Parts
Compile
Compile - \40ktest\A
Ini Place
Opt Place
Ini Route
Opt Route
Bit Str
Design Browser - \40ktest

Compile: info - Resetting display threshold for Routing Resources and Nets to 4% and Unrouted Nets to 14%, for current device.
Compile: info - 5 zoom levels use 0.3 MBytes for device section bitmaps (of allowed 3.0 MBytes).

Compile Window and Design Browser

Note the following:

- Besides the gray unused locations, there are now locations in several different colors (these are the darkest areas in the above example).
- Some of the pads at the sides of the device have also changed color.

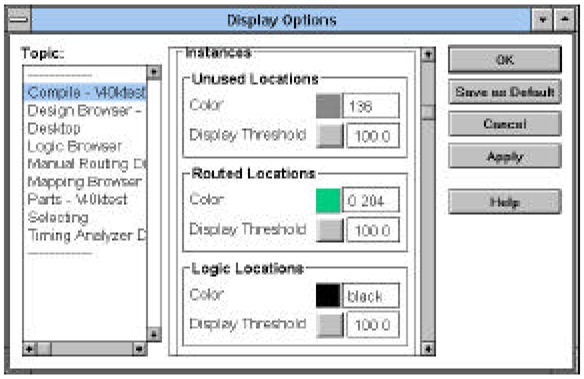
To understand this display, you need to:

- Find what the different colors mean.
- Locate objects from the design shown in the Design Browser.

Finding What Colors Mean (Optional)

This section shows how to find out about colors in displays:

1. Pull down *Options>Display Options* from the Compile window. In the dialog scroll down past the "Miscellaneous" parameters which control background color and so on. The main types of object in the window are listed like this:



Instances

Unused Locations
Color 138
Display Threshold 100.0

Routed Locations
Color 0 204
Display Threshold 100.0

Logic Locations
Color black
Display Threshold 100.0

Compile Window Display Options

For more information on what the various objects are, click *Help*. (Note that you probably will not see all the objects listed in the dialog in any one Compile window.)

2. Drag the dialog to one side of the screen so that you can see the Compile window at the same time. (To do this, click-hold in the dialog's title bar and drag it.)
3. Compare the colors in the scrolling list to the items in the Compile window. (Where three numbers replace the name of a color, these are red, green and blue values, each in the range 0 to 255.) The list begins with locations and instances, and then shows net colors. The third item under "Instances" is orange, the same color as most of the instances that changed color after Initial Place. The dialog shows that this is the color of logic locations (physical parts of the device into which design logic has been placed).
4. Scroll down the list until you find an instance type that uses cyan. The list shows that these are "contention locations".
5. To find the meaning of a particular color, search for that color in the dialog and see what type of component it represents. For example, magenta is used to represent locked instances, green for locations used to accommodate routing between instances.
6. Click *OK* to close the dialog box.

In summary, this inspection showed that:

- Most logic locations which have design instances placed in them are colored orange or pink. (Design instances are from your netlist, as shown in the Design Browser.) Macros and hierarchical instances are shown in the same color as in the Design Browser.
- The third location color is cyan. This shows where multiple parts of the design have been placed in a single location. This situation is called contention. If there's no contention in the design, do not worry, this means Figaro has placed everything correctly first time.
- The pads connected to the ports you locked using the Parts window are colored magenta.

Cross Highlighting (Optional)

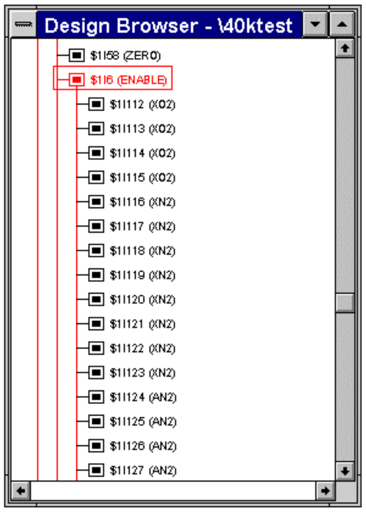
Another problem is finding where individual objects from the design have been placed. Cross highlighting makes it easy to identify objects in different windows. It means that selecting an object in one window also selects it (and changes its color) in other windows. You'll look at this briefly here.

Do the following:

1. In the Browser find instance $1I69\A0 (FULL_ADD) (for AT6K devices) or FIFO\$1I6 (for AT40K devices).

   To do this, select $1I69 (ADD4) (for AT6K Devices) or FIFO(RAMCONT) (for AT40K devices) and double-click to expand it. Select A0 (FULL_ADD) (for AT6K devices) or $1I6(ENABLE) (for AT40K devices) and double-click to expand it.

   The Browser should now look like the picture below, with A0 (for AT6K devices) or $1I6 (for AT40K devices) highlighted and boxed and its component items dis played. (These components are not highlighted in the Browser, but they are in fact selected).

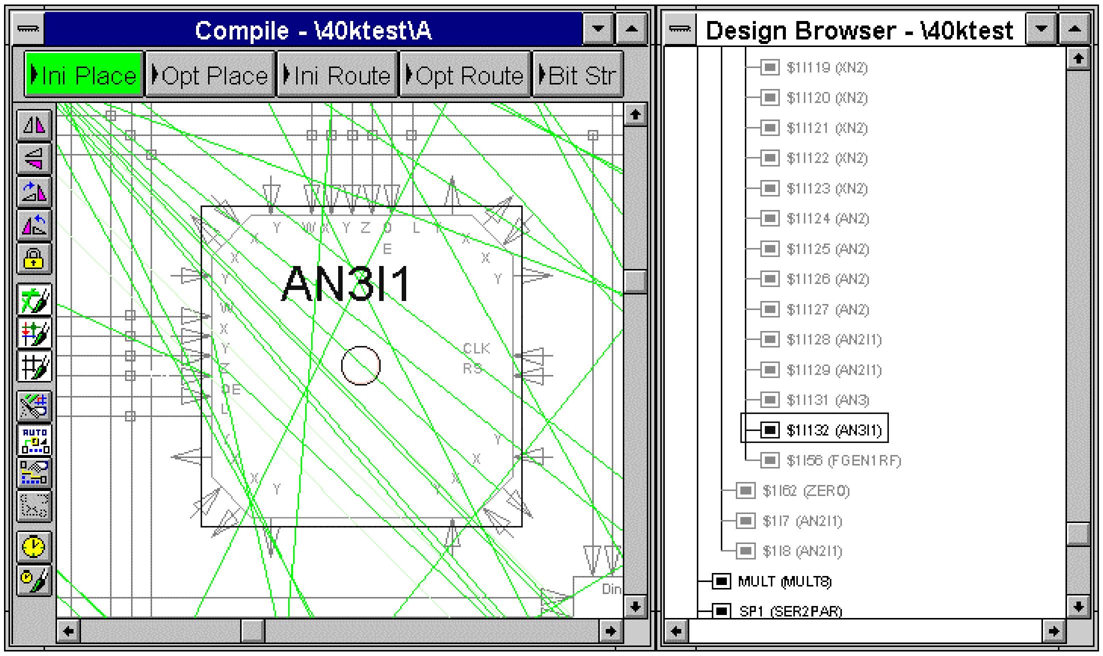


Design Browser - \40ktest

```
$1I58 (ZERO)
$1I6 (ENABLE)
    $1I112 (XO2)
    $1I113 (XO2)
    $1I114 (XO2)
    $1I115 (XO2)
    $1I116 (XN2)
    $1I117 (XN2)
    $1I118 (XN2)
    $1I119 (XN2)
    $1I120 (XN2)
    $1I121 (XN2)
    $1I122 (XN2)
    $1I123 (XN2)
    $1I124 (AN2)
    $1I125 (AN2)
    $1I126 (AN2)
    $1I127 (AN2)
```

Design Browser Displaying Components of $1I6

2. Look closely at the Compile window. Several objects in it are highlighted in yellow, the default color for selected items. These are the components cross-highlighted from Browser to Compile window. Note that the individual items change to the selection color in the Compile window, but not in the Browser.
3. Select G5 (OR3) (for AT6K devices) or $1I132(AN3I1) (for AT40K devices) in the list below the expanded item in the Design Browser. Just one object in the Compile window remains yellow: this is the only selected object now.
4. Click in the title bar of the Compile window to make it the current window and pull down *View>Zoom to Selected*. The instance type shown in the Compile window is OR3 (for AT6K devices) or AN3I1 (for AT40K devices): this text appears in the top left-hand corner of the macro. The displays in the two windows should look something like this:



$1I119 (XN2)
$1I120 (XN2)
$1I121 (XN2)
$1I122 (XN2)
$1I123 (XN2)
$1I124 (AN2)
$1I125 (AN2)
$1I126 (AN2)
$1I127 (AN2)
$1I128 (AN2I1)
$1I129 (AN2I1)
$1I131 (AN3)
$1I132 (AN3I1)
$1I56 (FGEN1RF)
$1I62 (ZERO)
$1I7 (AN2I1)
$1I8 (AN2I1)
MULT (MULT8)
SP1 (SER2PAR)

Cross-highlighting with $1I32(AN3I1) Selected

The Zoomed View

When you zoom in on an area, as you did just now, small scale detail of the device is shown. This is not shown before you zoom in because the mass of detail would make it impossible to see relevant items at the larger scale. The “display threshold” which can be set for each instance or net type controls this concept. You change the value for the "Display Threshold" under *Options>Display Options*. For more information, jump from the on-line help index to "Glossary of Terms", and then scroll to "Display Threshold".

The zoomed view shows the elements of the Atmel architecture.

What is Contention? (Optional)

Using the interactive flow, it is very likely that you will come across contention. This is a technique used by the automatic placement and routing algorithms allowing design instances and routes to be temporarily stored in the same place on the device until a solution is found for them. The default color for cell and routing resources in contention is cyan.

After initial placement, it is likely that your design contained a lot of contention. You don’t need to worry about this as the optimize placement step will usually remove all or most of it.

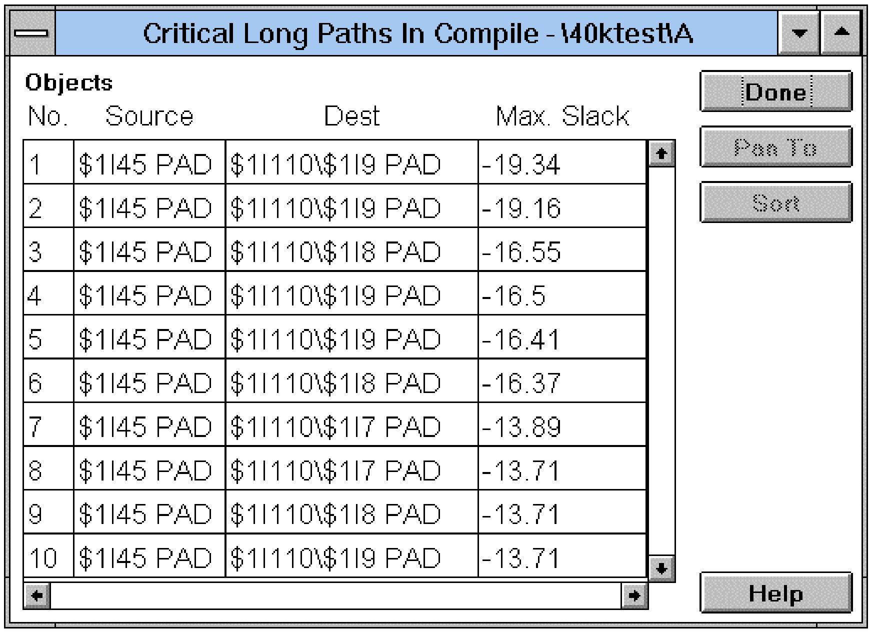
For more information on resolving contention, see the troubleshooting section at the end of this chapter.

Analyzing Timing after Placement

This section describes how you can obtain information on the timing that can be achieved by the current implementation of your design:

Using Show Analyzed Paths

1. Pull down *Timing>Show Analyzed Paths* and confirm that you want to run timing analysis. Figaro displays the Find Paths of Type dialog box.
2. Select "Critical Long" from the list and click *OK*. Figaro lists the ten paths with the most critical long path slack values. These values are logic delays only, as we have not done any routing yet.



No.	Source	Dest	Max. Slack
1	$1I45 PAD	$1I110\$1I9 PAD	-19.34
2	$1I45 PAD	$1I110\$1I9 PAD	-19.16
3	$1I45 PAD	$1I110\$1I8 PAD	-16.55
4	$1I45 PAD	$1I110\$1I9 PAD	-16.5
5	$1I45 PAD	$1I110\$1I9 PAD	-16.41
6	$1I45 PAD	$1I110\$1I8 PAD	-16.37
7	$1I45 PAD	$1I110\$1I7 PAD	-13.89
8	$1I45 PAD	$1I110\$1I7 PAD	-13.71
9	$1I45 PAD	$1I110\$1I8 PAD	-13.71
10	$1I45 PAD	$1I110\$1I9 PAD	-13.71

Critical Long Paths in Compile Dialog Box

The *Max. Slack* column tells you how much slack is available to accommodate the routing for the path. A positive value means that the path can be routed within the current timing constraints, but here all the slack values are negative, meaning that Figaro will not be able to compile the design given the current timing constraints.

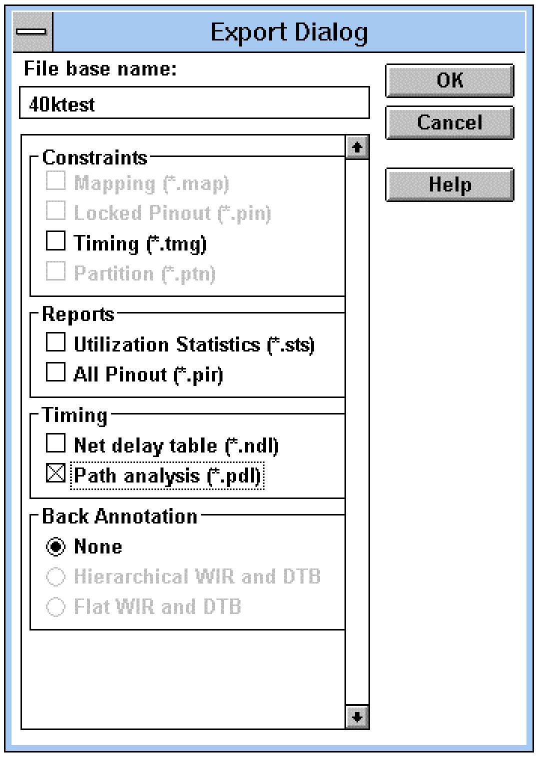
NOTE You can, of course, still go ahead and compile the design, examine the best case, then manually adjust any nets that are over the limit. *Remember:* if you have already done some manual routing, you may not be getting purely logic delays.

3. Click *Done* to return to the desktop before proceeding.

Examining Path Values in the PDL Viewer

Show Analyzed Paths provided some basic information on the critical paths, but if you want more details on how the figures were arrived at, you have to look at the *Path Analysis Report*:

1. Pull down *File>Export* and check the box marked "Path analysis (*.pdl)":



File base name: 40ktest

Constraints
- [ ] Mapping (*.map)
- [ ] Locked Pinout (*.pin)
- [ ] Timing (*.tmg)
- [ ] Partition (*.ptn)

Reports
- [ ] Utilization Statistics (*.sts)
- [ ] All Pinout (*.pir)

Timing
- [ ] Net delay table (*.ndl)
- [✓] Path analysis (*.pdl)

Back Annotation
- [✓ None]
- Hierarchical WIR and DTB
- Flat WIR and DTB

Exporting a Path Analysis Report

2. Click *OK* to export the file.
3. Now pull down *Window>New Viewer>Path Analysis Report*. Figaro opens the PDL Viewer and loads the file "4BITALU.PDL" (for AT6K devices) and TEST40K.PDL" (for AT40K devices).
4. In the PDL Viewer, pull down *Edit>Find* and use it to find the second occurrence of the text "Long critical path analysis" (the search is case-sensitive) in the file.

   Figaro scrolls to the appropriate section of the PDL file.
5. Look at *Path #1* which will look similar to the extract below:

For AT6K devices:

```
Path #1

Slack = 35.3ns
Type  = Flop - Flop (''$1I213\$1I8 CLK'->'$2I73\$1I10\$1I34 D'')

Clock Edge: 'FINISH' on '$1I72\OR0 Q'                  _/  7.30ns
Clock Delay: '$1I72\OR0 Q' -> '$2I73\$1I10\$1I34 CLK' _/  0.00ns
Clock Period: gcd(FINISH, CLOCK)                        100.00ns
                                                     -----------
Required Arrival Time:                                 \_100.30ns

Clock Edge: 'CLOCK' on '$1I187 A'                      _/  0.00ns
Clock Delay: '$1I187 A' -> '$1I213\$1I8 CLK'           \_  9.50ns
Data Path: '$1I213\$1I8 CLK' -> '$2I73\$1I10\$1I34 D' \_ 60.40ns
Setup '$2I73\$1I10\$1I34 D'                            \_  2.10ns
                                                       ----------
Actual Arrival Time:                        \_ 72.00ns
```

For AT40K devices:

```
Long critical path analysis
---------------------------

Path #1

Slack = -34.39ns
Type  = Flop -> Output ('FIFO\$1I5\$1I3 CLK' ->'$1I194\IOB15 PAD')

Clock Edge: 'CLOCK' on '$1I45 PAD'                          _/  0.00ns
Asserted Required Time: 'MULT_OUT15' on '$1I194\IOB15 PAD' _/  1.00ns
                                                               -------
Required Arrival Time:                                      \_  0.96ns

Clock Edge: 'CLOCK' on '$1I45 PAD'                          _/  0.00ns
Clock Delay: '$1I45 PAD' -> 'FIFO\$1I5\$1I3 CLK'           _/  1.02ns
Data Path: 'FIFO\$1I5\$1I3 CLK' -> '$1I194\IOB15 PAD'      \_ 34.33ns
                                                               -------
Actual Arrival Time:                                        \_ 35.35ns
```

The path report provides breakdown of how the timing analysis values were calculated. The *required arrival time* is computed by adding the clock edge to the delay and the period. The calculation of the *actual arrival time* replaces the clock period with the delay along the data path and the setup value where appropriate

The slack value at the very top is calculated by subtracting the actual arrival time from the required arrival time and is the same as that listed in the dialog. Figaro also gives you the path type and its start and end points.

The *data path delay* is broken down into further detail directly after the summary. Scroll down the PDL file to see how it is represented.

The path shown leads from CLOCK to an output pad.

NOTE

It is recommended that you check the viability of your timing constraints after the initial placement step, and before routing delays are added to complicate the situation.

You should always check that any default assertions and constraints entered by Figaro are appropriate for your design. Automatically generated constraints are set to the period of the most appropriate external clock. If you are using derived clocks, the default output assertions may not be appropriate.

Changing the Default Constraints

You can change the timing constraints set for your design either in the Constraints Editor, or by creating and importing an ASCII .tmg file containing the corrected constraints:

1. Return to the Figaro desktop and pull down *Edit>Timing Constraints*.
2. In the “Clocks” page, change the Period to 40 ns and Half Period to 20 ns. Click *Accept*. The logic delay was ~35 ns, so setting the delay to 40 ns leaves 5 ns for routing. This may not be enough, but gives the tool a more realistic value to work with.
3. Click *Done* on the Timing Constraints dialog.
4. Now re-run timing analysis by exporting the path analysis report again and look at the updated PDL file. The slack values are now all positive.

This example highlights the problems of relying too heavily on auto-generated constraints. Always take care when defining IO assertions and be aware of the interaction of the clocks in the design, particularly where you are dealing with multiple clock domains.

Remember that the default constraints generated by Figaro are intended as a starting point for timing-driven compilation. To get the most out of your implementation, it is likely that you will want to define your own assertions.

Running Optimize Placement

Optimize placement is the second compile stage. It can be split into two phases: global placement and detailed placement. In timing-driven mode, both will seek to meet your clock frequencies and eliminate contention.

1. Pull down *Flow>Compile>Optimize Placement* or click the *Opt Place* button in the Compile window. The status bar and transcript keep you informed of Figaro's progress.
2. When the *Opt Place* button turns green, pull down *Window>New Viewer>Log File* and take a closer look at the timing reports output at various stages of the optimize placement step.
3. Scroll back to the two Timing Reports sections. An example of one is shown below.

   ```
   Clock domain: CLOCK
   Asserted period:                40.00ns (  25.00MHz)
   Estimated best period:          36.89ns (  27.10MHz)
   Target period:                  40.00ns (  25.00MHz)
   ```

   The pre-place delay model assumes that direct connects can be made for all routes, so the estimated frequency given here is the absolute best case that the design can achieve. If the timing constraints have not been met at this stage, you need to check that the assertions were entered correctly or rethink your design.
4. Scroll down to the "Timing report using phase 1 place delay model". This model is less critical and provides an approximate indication of the routing delays after the global placement step.

Routing

Routing makes connections between logic locations using available routing resources on the circuit. Routing is split into two stages, Initial Route and Optimize Route.

Initial Route

Before performing initial routing, workstation users should pull down *Window>New Shell Window* and use the **more** command to view the messages written to the "4BITALU.LOG" (for AT6K devices) “TEST40K.LOG" (for AT40K devices) file while initial route is running (PC users must wait until initial routing is complete before viewing this information in the log file).

Click the *Ini Route* button or pull down *Flow>Compile>Initial Route*. This step has five phases: after phase 3, the log file will show a timing report using the initial route delay model. This is the best possible routing for each path in the design given the current placement, with each path routed in isolation and no other nets taken into consideration.

If routing is taking a long time, then it is worth checking this value to see if it is close to meeting the constraints. If Figaro fails to meet the constraints at this stage, you may want to terminate routing early and return to the placement step and make some improvements there.

If the Log Viewer is still open from the previous section, you can update the contents using *File>Update*. If you closed it before starting initial route, re-open it as before or using the appropriate toolbar button.

Scroll down the log file to the second set of timing data. This is the first report of **actual** values, namely those obtained after initial routing. Together with the route contention score, these values provide a good indication of whether or not optimize routing is likely to meet the constraints you set. In this case, the signs are not good.

Optimize Route

When using timing-driven compilation you **must** run optimize route, since it can almost always improve on the timing obtained by initial routing. For non-timing-driven designs, this step need only be run if there is contention left after the initial route step.

Click *Opt Route* or pull down *Flow>Compile>Optimize Route* to run the step.

Interactive Timing Analysis

When your design has completed, you can view the final clock frequencies in the log file. There are also a number of other ways of examining timing information interactively within Figaro.

The Path Analysis Report

You have already looked at an example of this report earlier in this tutorial. Its contents are affected by the assertions you set:

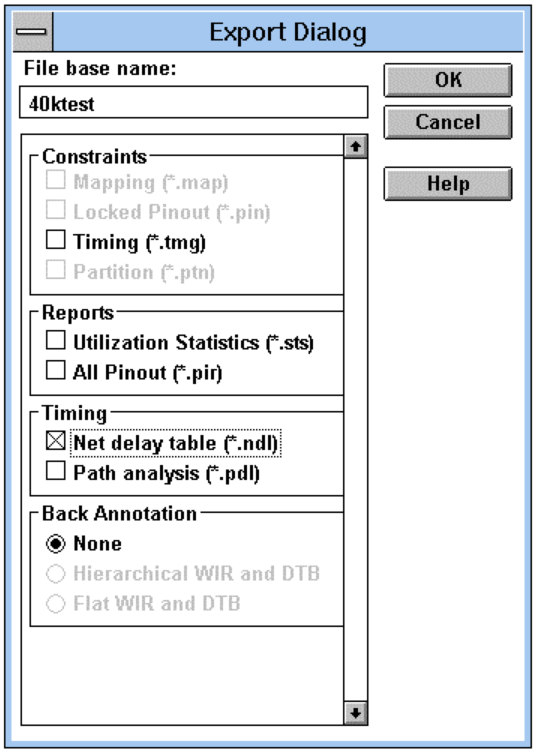
- With no assertions specified, the report lists the longest and shortest paths in the design.
- With clock and/or input and output assertions specified, the longest and shortest *critical* paths are also reported.
- With asynchronous delay assertions, you will get a report on asynchronous longest and shortest critical paths.

You can changes the types of reports and the numbers of paths reported in the Timing Analysis options dialog.

The Net Delay Table

The net delay table lists the delay between the source and destination of every net. You export in the same way as the path analysis report:

1. From the Compile window, pull down *File>Export:*



Export Dialog

File base name: 40ktest

Constraints
- [ ] Mapping (*.map)
- [ ] Locked Pinout (*.pin)
- [ ] Timing (*.tmg)
- [ ] Partition (*.ptn)

Reports
- [ ] Utilization Statistics (*.sts)
- [ ] All Pinout (*.pir)

Timing
- [x] Net delay table (*.ndl)
- [ ] Path analysis (*.pdl)

Back Annotation
- ● None
- ○ Hierarchical WIR and DTB
- ○ Flat WIR and DTB

Export Dialog Box

2. Under “Timing”, check the “Net delay table” box, as shown above.
3. Enter a name and path for the file, then click *OK*. If you leave this set to “4BITALU” (for AT6K devices) or “TEST40K” (for AT40K devices), as in the above example, the table is written to a file called 4BITALU.NDL (for AT6K devices) or TEST40K.NDL (for AT40K devices) in the current design directory. Examples of the timing reports are shown in Appendix B.

Performing Interactive Timing Analysis

Figaro's main interactive timing analysis features are the commands in the *Timing* menu and the Paths in Compile and associated dialogs. These dialogs summarise the information contained in the "Data Paths" sections of the Path Analysis Report, and facilitate interactive investigation of individual paths in the design:

1. Make sure the Compile window is the current window.
2. Pull down *Timing>Show Analyzed Paths*.
3. Select "Critical Long", then select the first path in the table.

   This selects the path and its components in all Figaro windows. Now click on the *Inst/Nets* button to see a more detailed breakdown of this path.
4. Pull down *View>Zoom to Selected*.

   Figaro zooms to the source of the selected path in the Compile window. With the normal display, it may be prove difficult to discern the source and destination of the path. The Timing Display mode filters all extraneous information from the display.
5. Pull down *Timing>Timing Display*.

   This filters out unused resources and turns all instances gray.
6. Pull down *Timing>Measure Delay*, with the path still selected.

   The colors used to represent the selected path change: the source instance is now blue, intermediate nets and instances green, and the destination red. (In this example the source is a clock, so the appropriate pad is colored blue as is the input to the first instance on the path.)

   The cursor is now followed by a small green clock. This is a visual reminder that you are in Measure Delay mode. Use the function keys to move within the window: F6 to center, F7 and F8 to zoom in and out respectively. If you take the cursor to the edge of the window, it will scroll, assuming there is more device off-screen. If you hold the SHIFT key down, it will scroll one page at a time.
7. Look again at the selected path and click on one of the green intermediate instances on the path.

   Note that the path from the source to this point changes color and the delay for this section is reported in the transcript. This can be useful for examining complex paths, where individual path sections are not easy to follow. You can step backwards or forwards in the path.
8. Exit Measure Delay mode either by clicking on a destination of the path (red instance) or by clicking the right mouse button. You must reselect the source to enter Measure Delay mode again.

If you want to examine paths that are not in the Paths in Compile dialog, select the source instance and pull down *Timing>Measure Delay*. This will show you a list of paths from that source. The number of paths listed is controlled by the "Measure Delay Paths Limit" parameter in the Timing Analysis options (default is 10).

You can also select single or multiple nets in the design, either interactively or using *Edit>Find* and *Edit>Info*, and the pull down *Timing>Show Net Delays*. The results are written to the transcript.

All these functions are available even if you are not in Timing Display mode, however, the simplified display makes them much easier to use.

Using Manual Editing to Improve Timing on a Path

Experienced Figaro users with a detailed knowledge of the AT40K/AT6K architecture can use the following manual editing techniques to improve the timing along a particular path in the design:

- Search for areas where route-throughs have been used and see if you can swap ports on the macros to get better connections.
- Look for adjacent instances using bus routing to connect, and try swapping ports to get direct connects.
- Reduce the number of bus turns being used by moving elements so that they are in the same row or column.
- For IO routing, align within the IOs that connect to a particular row or column.
- Manually move instances closer together to reduce the required routing. Remember though, that once you get to this stage, you will probably be affecting more than just the net you're working on.

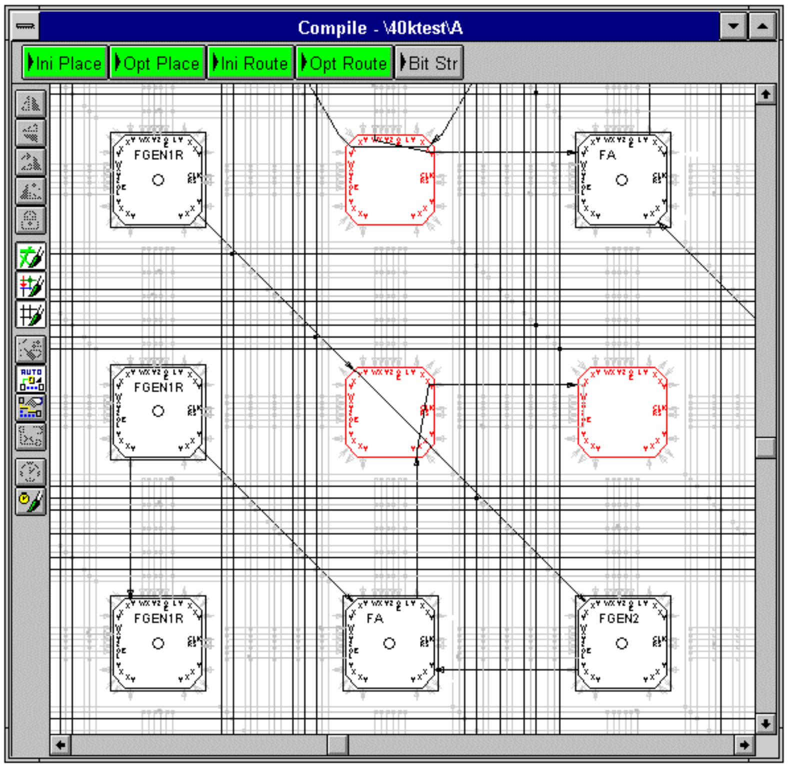
Changes to the timing of a path caused by these manual operations are reported in the transcript immediately. This lets you see straight away if you have actually improved the situation. If you are changing critical paths, always re-run timing analysis to see the changes in the dialog - they will not be updated automatically. Running timing analysis can take some time on larger devices, so it is advisable to make a few changes at a time and then re-test them all together.

Examining the Placed and Routed Design (Optional)

In this section, you will examine the placed and routed design in more detail. Zoom in on an area of the device. There are two main changes to the display:

- A number of green squares have appeared. These are routed locations, i.e. logic locations whose resources used for routing only and not for design instances themselves. It may also be that the cell is rendered unusable for logic because of the exe cution of a bus turn nearby. (For full details refer to your IDS documentation.).
- Lines appear representing the nets in the design. Use *View>Zoom to Area* to focus on a small area of the device to see these.

The following picture shows routing in one area of the device:



Part of the Device after Optimized Routing

Looking at Nets in the Design (Optional)

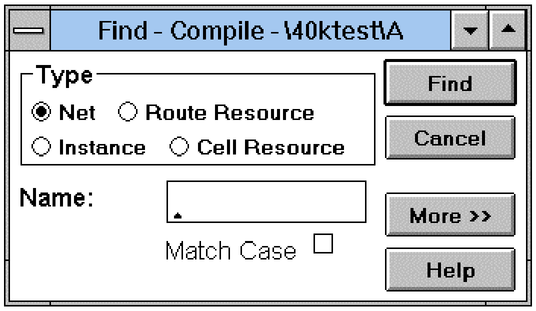
The routed design is now ready for you to produce a bitstream file for eventual downloading to an FPGA. First, though, take a look at the nets in the design.

Finding Named Nets (Optional)

To learn how to locate nets, you'll look for the nets W1 to W4 from one of the adders in $1I69 (ADD4) (for AT6K devices) or READ3 and WRITE3 (for AT40K devices) from the FIFO component of the design: So far you have used the Browser and cross-highlighting to select items in other windows, but you can't find nets in this way. You have to use the *Edit>Find* command.

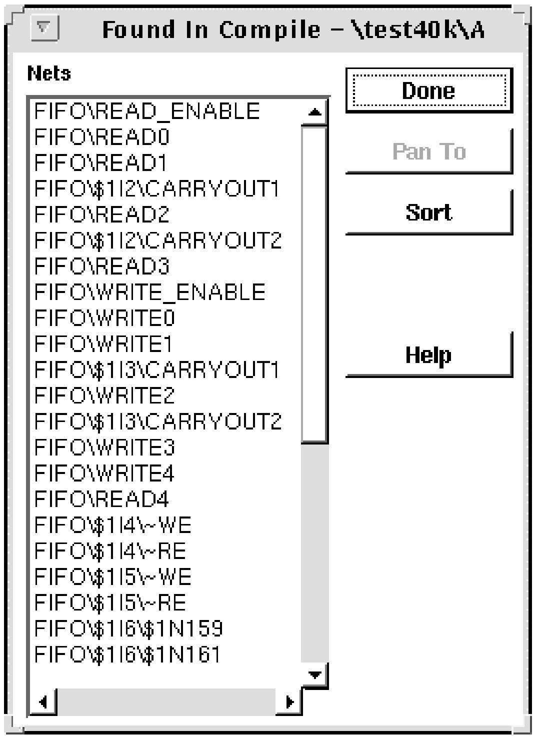
Do the following:

1. Pull down *Edit>Find* (CTRL+F) to display this dialog:



The Find Dialog Box

2. With the “Net” radio button selected, try to find the first net. Type “*W1” (for AT6K devices) or "FIFO*" (for AT40K devices) in the “Name” text box (the “*” is a wildcard) and click *Find*.
3. A Found in Compile dialog appears, listing nets that match your entry:



FIFO\READ_ENABLE
FIFO\READ0
FIFO\READ1
FIFO\$1I2\CARRYOUT1
FIFO\READ2
FIFO\$1I2\CARRYOUT2
FIFO\READ3
FIFO\WRITE_ENABLE
FIFO\WRITE0
FIFO\WRITE1
FIFO\$1I3\CARRYOUT1
FIFO\WRITE2
FIFO\$1I3\CARRYOUT2
FIFO\WRITE3
FIFO\WRITE4
FIFO\READ4
FIFO\$1I4\~WE
FIFO\$1I4\~RE
FIFO\$1I5\~WE
FIFO\$1I5\~RE
FIFO\$1I6\$1N159
FIFO\$1I6\$1N161

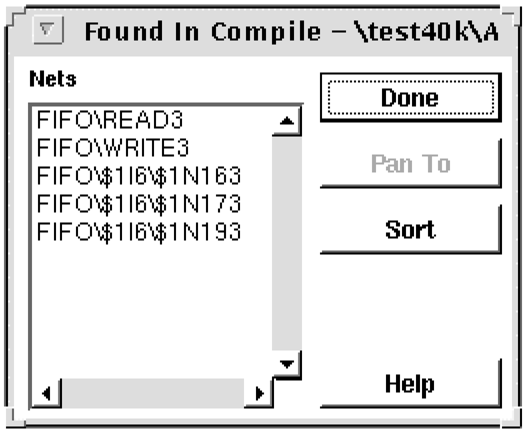
The Found in Compile Dialog Box

In this case, Figaro lists over 37 nets matching the search criteria. You can either scroll down the list and find the entries you want, or use the *Find* command again and try to specify the nets more precisely.

4. Pull down *Edit>Find*, enter the search string "*3" and click *Find*.

   This time Figaro lists the 31 nets in the design that end with the character "3".
5. Narrow down the search still further by entering "*FIFO*3" and clicking *Find*.

   This time, only five nets are listed and you can quite easily see the two you wanted:



FIFO\READ3
FIFO\WRITE3
FIFO\$1I6\$1N163
FIFO\$1I6\$1N173
FIFO\$1I6\$1N193

Five Entries in the Found in Compile Dialog Box

Note that the "#" wildcard replaces any single character in the string. Shift-click on both items in the list. This selects in yellow the nets in the Compile window. To view them, look at the Compile window.

Keep the nets selected in the Compile window, click in it to make it the current window and click *View>Zoom to Selected* to focus on the area containing the selected nets.

NOTE

You can click *More* in the Find dialog to select items on the basis of more parameters than just their names.

Using Display Options to Identify Net Types

You used *View>Display Options* earlier to find what the colors of instances meant. The zoomed display now shows several types of net:

- Normal, clock and reset nets.
- Direct routed, local bus and express bus nets.

To find out which is which, make the Compile window the current window and pull down *View>Display Options*.

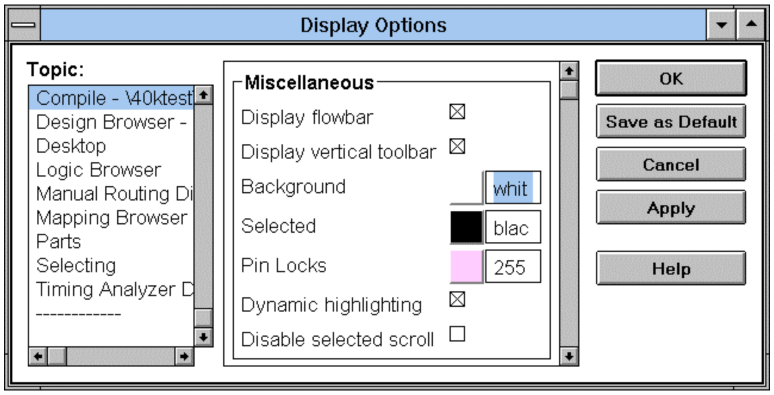
Drag the dialog box to one side of the screen so that you can see the Compile window too. This time scroll down the list of objects until you reach nets. Compare the colors shown with the nets in the Compile window. Note that the list shows all possible types of net: these may not all be present in your Compile window.

Look at the pattern of direct routed, local bus and express bus nets. There's no need to change any color; just use the list as a color key (note that there are two types of direct routed net). Don't close the dialog, though: you'll use it again in the next section.

Dynamic Highlighting (Optional)

It is not always easy to distinguish a single net from the mass of nets. One way to do this is to click to select the net: it will change color to yellow, and will be the only net of this color. A faster way to see where nets begin and end is to turn on dynamic highlighting. This highlights the element under the cursor in white. To use this, do the following:

1. Pull down *Options>Display Options* you looked at in the last section.
2. Check the box labeled "Dynamic highlighting", click *Apply* and then *OK*.



Display flowbar ☒
Display vertical toolbar ☒
Background whit
Selected ■ blac
Pin Locks 255
Dynamic highlighting ☒
Disable selected scroll ☐

The Dynamic Highlighting Check Box

Return to the Compile window and move the cursor over the dis play. As you do this, individual items - instances, sections of routing resource, or nets - are highlighted.

Bitstream and Export

Outputting a Bitstream

This section shows how to produce a bitstream file with the fin ished design for downloading to the FPGA. Do the following:

1. Open the Options dialog and select "AT40K Bitstream" from the "Topic" list. The options, which control the bitstream format, appear in the dialog box.
2. Don't change any options now: just take a look at any you think you'll want to set in future for your own designs. Note that the option "Reset Addresses" is set by default. When you finish looking at the options, click *OK*.
3. In the Compile window, click the *Bit Str* button. (You can only do this after Optimize Route has completed successfully.)

Bitstream Results

Bitstreaming writes the bitstream file 4BITALU.BST (for AT6K devices) or TEST40K.BST (for AT40K devices) to the Figaro directory. The first part of the file name comprises the design name and, where the implementation comprises more than one part; this is followed by the letter used to designate that part. (For example, the bitstream file for the third part added would be called *design*_C.BST.) PC users will see file names exceeding eight characters in length truncated appropriately.

This is the file that you would download to an FPGA. Please re fer to your IDS documentation for a description of this procedure.

Note that the *Compile* button on the Figaro window now changes color, because you have completed all its component stages in the Compile window.

Exporting a Design for Back-Annotation

This section shows how to produce back-annotation files for use by ViewSim, one of the simulators Figaro supports. (For details of the files for other simulators, *Search* the on-line help for “Back-annotation” then read its sub-topic “Files for Back-annotation”.) You can only produce the files after you have routed the design and removed all contentions.

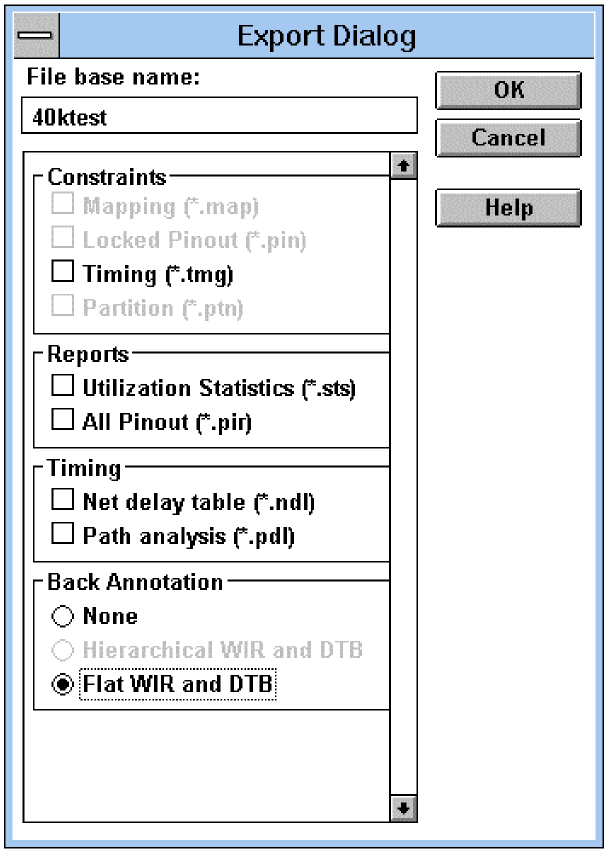
ViewSim requires two types of file, which you export from Figaro using *File>Export*. These are:

ViewLogic WIR files	A set of files similar to those you read in from the WIR directory. The export files are different from those in the netlist WIR directory, however, because back annotation is done at the core cell level and not the original functional macro level. Thus the only parts of the design that match the original are the primary input and output nets.
ViewLogic delay file	A single file with post-layout delay timing information. This is in Delay Table Back-annotation (DTB) format and has the extension ".DTB". (For full details of the file, see the on-line help or your ViewLogic documentation.)

Running ViewSim is outside the scope of this exercise, but it lets you inspect the timing behavior of your design, then modify Figaro's placement and routing on the basis of this.

To output the files, do the following:

1. Make the Compile window the current window.
2. Pull down *File>Export* to display this dialog. (Note that different parameters will be grayed out if you are not in the Compile window with a fully routed device.)



File base name: 40ktest

Constraints
- ☐ Mapping (*.map)
- ☐ Locked Pinout (*.pin)
- ☐ Timing (*.tmg)
- ☐ Partition (*.ptn)

Reports
- ☐ Utilization Statistics (*.sts)
- ☐ All Pinout (*.pir)

Timing
- ☐ Net delay table (*.ndl)
- ☐ Path analysis (*.pdl)

Back Annotation
- ○ None
- ○ Hierarchical WIR and DTB
- ● Flat WIR and DTB

Exporting Back-Annotation Files

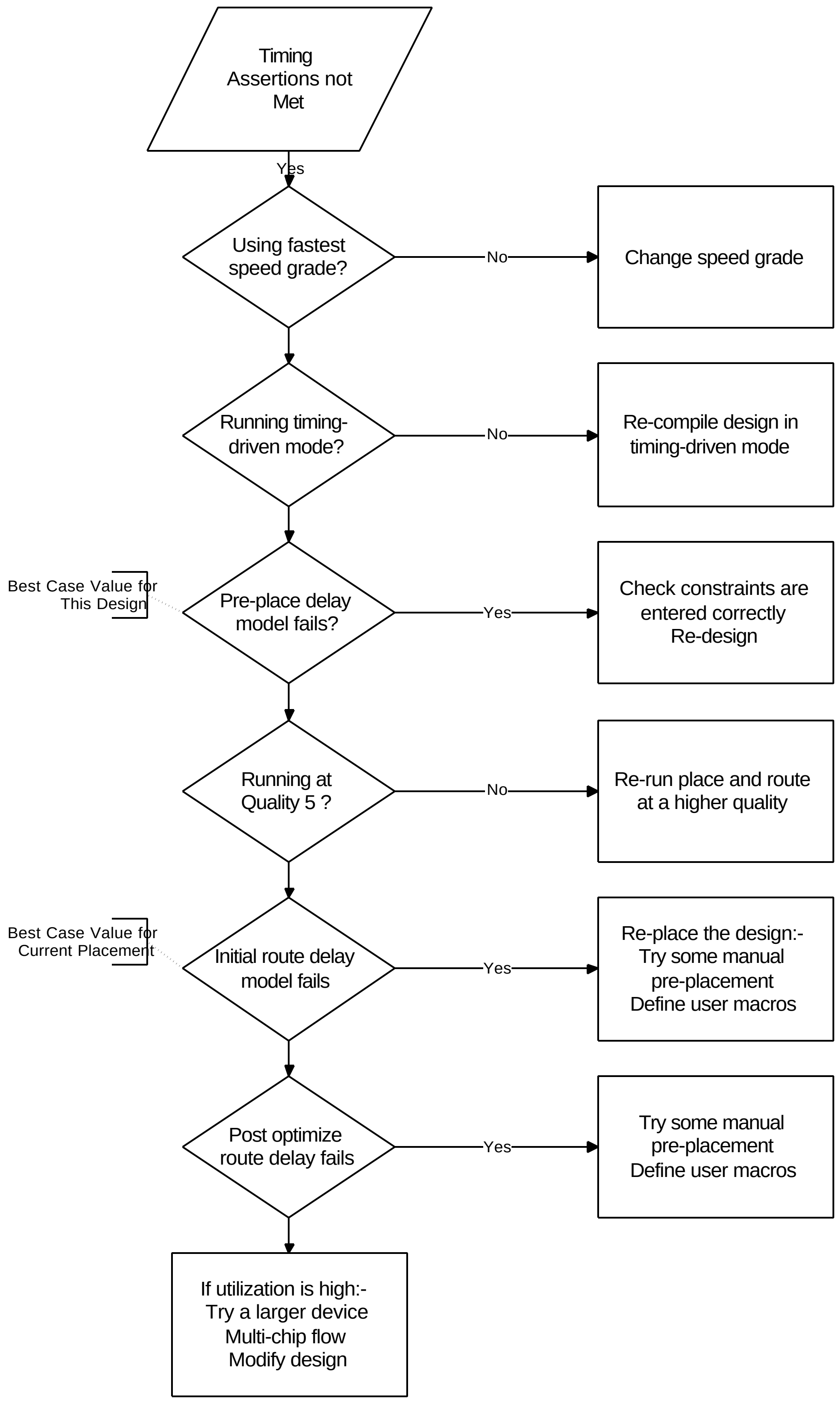
3. Leave "4BITALU" or "TEST40K" as the file base name. A new WIR directory holding the exported WIR files is created in a new directory FIGBA under the design directory.
4. Check the "Flat WIR and DTB" radio button as shown above and click *OK*. (You can export only flat WIR and DTB back-annotation files when using the AT40K configuration.)

NOTE

Before you can use ViewSim, change the directory statements in your VIEWDRAW.INI file to point to the exported WIR files; Search the on-line help for VIEWDRAW.INI and read its sub-topic "Modifying the VIEWDRAW.INI File".

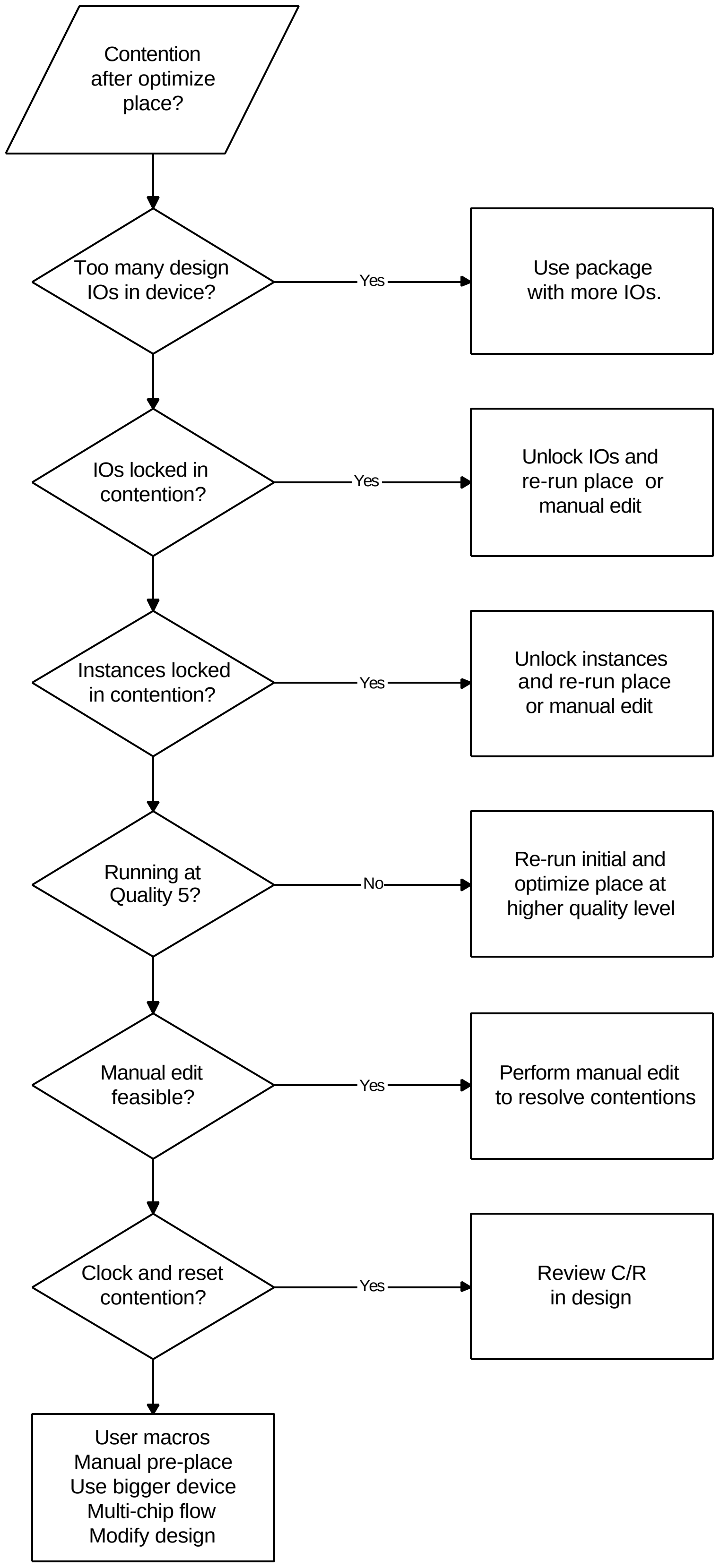
Troubleshooting Timing-Driven Compilation

If your timing assertions have not been met, check the flow chart below for your best course of action:


Timing Assertions not Met
Yes
Using fastest speed grade?
No
Change speed grade
Running timing-driven mode?
No
Re-compile design in timing-driven mode
Best Case Value for This Design
Pre-place delay model fails?
Yes
Check constraints are entered correctly Re-design
Running at Quality 5 ?
No
Re-run place and route at a higher quality
Best Case Value for Current Placement
Initial route delay model fails
Yes
Re-place the design:- Try some manual pre-placement Define user macros
Post optimize route delay fails
Yes
Try some manual pre-placement Define user macros
If utilization is high:- Try a larger device Multi-chip flow Modify design

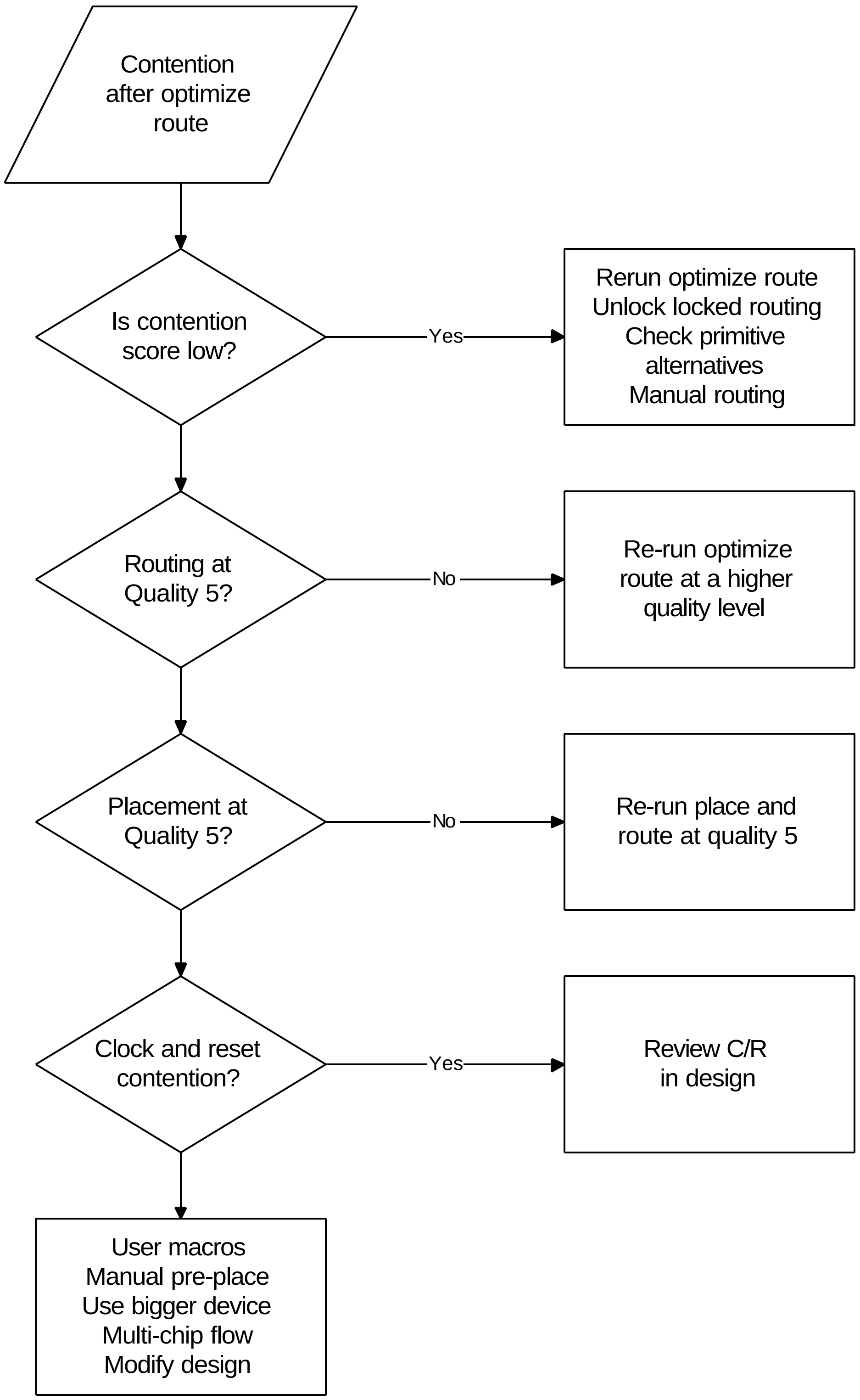
Resolving Contention after Placement

If your design contains contention even after optimize placement has run, check the flowchart below for the best course of action:


Contention after optimize place?
Too many design IOs in device?
Yes
Use package with more IOs.
IOs locked in contention?
Yes
Unlock IOs and re-run place or manual edit
Instances locked in contention?
Yes
Unlock instances and re-run place or manual edit
Running at Quality 5?
No
Re-run initial and optimize place at higher quality level
Manual edit feasible?
Yes
Perform manual edit to resolve contentions
Clock and reset contention?
Yes
Review C/R in design
User macros
Manual pre-place
Use bigger device
Multi-chip flow
Modify design

Resolving Contention after Routing

If your design contains contention even after optimize routing has run, check the following flowchart for the best course of action:


Contention after optimize route
Is contention score low?
Yes
Rerun optimize route Unlock locked routing Check primitive alternatives Manual routing
Routing at Quality 5?
No
Re-run optimize route at a higher quality level
Placement at Quality 5?
No
Re-run place and route at quality 5
Clock and reset contention?
Yes
Review C/R in design
User macros Manual pre-place Use bigger device Multi-chip flow Modify design

Summary

This tutorial showed you how to:

- Add a part.
- Apply timing constraints.
- Place and route the design in timing-driven mode.
- Use the Design Browser to select objects in a window.
- Identify objects by their color.
- Find objects in a window.
- Export constraints and back-annotation files and timing reports.
- Resolve placement and routing contention in your design.

If you want to learn how to place and route manually, or produce user macros, work through the next tutorial as well. In either case, look at the contents of “Additional Features”. This describes some other tasks you might want to perform, but which are outside the scope of the tutorials.

You should now be able to compile a design of your own using the techniques described here. If you have problems, refer to “Troubleshooting and Support”, or from the on-line help Contents page jump to “Troubleshooting”.

Manual Editing and User Macros

Introduction

Work through this tutorial if you intend to use the manual editing functions or produce user macros for your design.

Once again, you will use components of the 4BITALU or TEST40K design used in the previous tutorials to learn about manual placement and routing techniques, particularly in relation to the creation of user macros. You will check the completed user macro into a library, where it becomes available for use with other designs. Finally, you will incorporate your user macro in the overall design and examine some of the different ways in which you can manipulate it in the final implementation.

Summary of Topics

This tutorial contains the following topics:

1. When to Use Manual Editing
2. When to Define User Macros
3. Creating a User Macro Manually
4. Checking a User Macro into a Library
5. Creating a User Macro Automatically
6. Incorporating User Macros in a Design

Functions already used in previous tutorials are not covered in detail again here except where considered relevant. Please refer to the appropriate chapters for more detailed information.

NOTE Do not confuse the terms 'macro' and 'user macro'. The instances used here may contain macros from the vendor library, but the overall groupings you lay out in this exercise are user macros.

When to Use Manual Editing

Manual editing allows you to specify how design instances are placed and routed. You might use manual editing immediately after opening a design to optimize some sections and lock the results before running the automatic tools to complete the task; or you may prefer to run the automatic tools first, study the results and then improve layout and routing using the manual functions.

Manual editing is particularly useful in the following situations:

In the user macro flow

- During placement.
- When permuting ports to obtain better connections using low-cost nets.
- In manual routing of direct connects through cells (if there is a one-to-one direct connect, the automatic router will find it).

Note that bus routing from user macros is not preserved in the final design.

In the design flow

- To allow preliminary placement of part of the design.
- In resolving contention.
- When adjusting timing to meet constraints.

NOTE You can use any of the manual tools before or after any of the automatic steps, however the automatic steps must always be performed in the correct sequence.

When to Define User Macros

User macros allow expert users to exploit the design hierarchy to obtain better and more predictable results in terms of timing and routability. You can optimize a commonly used part of a design, then save it as a user macro and check it into a user macro library where it can be used in other designs. To get the best from your user macros, you need a detailed knowledge of the target architecture.

There are three main reasons for creating user macros:

- To place and route a frequently used piece of functionality and store it for use in other designs.
- To optimize placement and routing of a block of structured logic in your design that you know should be grouped together closely, or in a specific way.
- To guarantee the timing of the user macro.

There are four main steps in producing a user macro:

1. Open the netlist, which must consist of **only** logic and memory functions. I/O macros cannot be included in the netlist for a user macro.
2. Wait for Figaro to select an appropriate part that will be used as a template for the design space. The selection is made based on the size of the macro netlist.
3. Place and route the design in a small area of the device.
4. Export the design to an external library that will be loaded when the macro is instantiated in the design later.

The following sections show you how to produce a user macro using Figaro's manual editing techniques and automatic placement and routing tools.

Creating a User Macro Manually

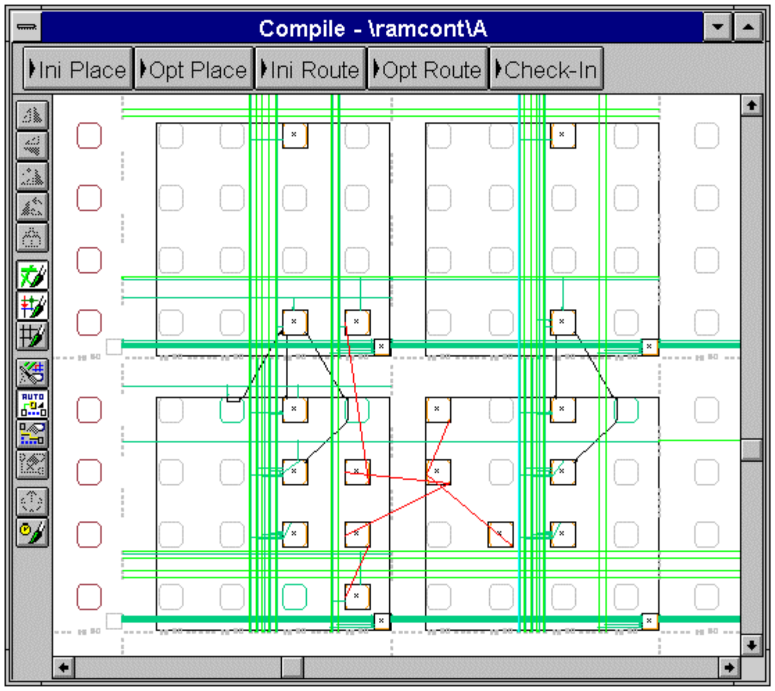
In this section, you will create a user macro for the JKFP2 (for AT6K devices) or RAMCONT (for AT40K devices) component of the 4BITALU/TEST40K design we have used throughout these tutorials. You will get to know many of the manual placement and routing tools and will find out how to get information on selected objects in the design.

Setting up the Design Session

To set up this design session, start up Figaro as described in Chapter 3 and proceed as follows:

1. Pull down File>Design Setup and click New Design.
2. Select JKFP2.1 from \SYSTEMDESIGNER\EXAMPLES\VLOGIC\4BITALU (for AT6K devices) the RAMCONT.1 netlist from \SYSTEMDESIGNER\EXAMPLES\AT40K\VLOGIC\TEST40K (for AT40K devices) and set the Tools Flow to "ViewLogic WorkView Office". Click OK in both dialogs to return to the desktop.
3. Pull down File>Open as Macro and set "Files of Type" to "ViewLogic Wir (*.1)". Macro name JKFP2 (for AT6K devices) or RAMCONT (for AT40K devices) is entered automatically in the appropriate field.

4. Click *OK* to load the macro netlist. Figaro reads in the netlist, adds a suitable part and opens a Compile window with the view zoomed in to show a section of open core cells surrounded by closed core cells:



Compile Window Showing Open Locations

Figaro has assessed the size of the netlist and limited the space available for logic and memory placement by closing surplus locations on the device template added. Pads and I/O logic have also been closed so that no design instances can be placed on them.

The section left open should be large enough to accommodate placement and routing for the macro, however you can change its size and shape as required using the *Open Location* and *Close Location* commands from the *Edit* menu.

NOTE When loading macro netlists, you may encounter some transcript and log file warnings about nets that will ultimately connect to the final design but are presently unconnected. For the purposes of this tutorial, you can ignore any messages, although ordinarily you would check carefully any nets specified before proceeding.

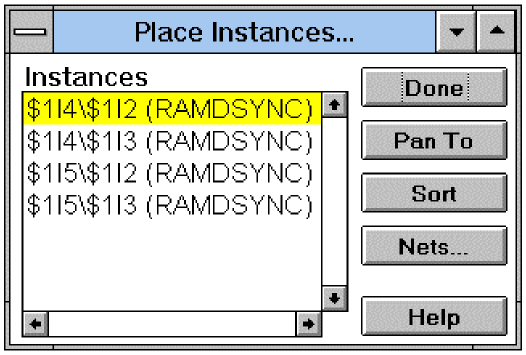
Placing Instances in a Selected Hierarchy for AT40K Designs

You can place an instance manually by selecting it in the Design Browser, dragging it over the Compile window and dropping it in the required cell location.

In addition to this, Figaro also provides a *place by hierarchy* feature that allows you to place all the instances contained in particular level of hierarchy. This function is particularly useful with large and complex hierarchical designs, and is activated automatically when you drag one or more hierarchical instances from the Browser to the Compile window:

1. Make the Design Browser the current window and use SHIFT-click to select instances $1I4 and $1I5 and drag them over the Compile window.

   Figaro opens a Place Instances dialog, listing the instances contained in the hierarchies you selected.



$1I4\$1I2 (RAMDSYNC)
$1I4\$1I3 (RAMDSYNC)
$1I5\$1I2 (RAMDSYNC)
$1I5\$1I3 (RAMDSYNC)

Components of $1I4 and $1I5

2. Rearrange the windows so that you can see the Compile window and then move the cursor over it.

   The first instance in the Place Instances dialog is "attached" to the cursor. You might have to click once to activate the Compile window.)
3. Move the cursor over sector (3,4) and click once to "drop" the instance at this location (the status bar shows the sector in the bottom right corner of the screen).
4. Move the cursor over the Compile window again. The instance is now attached to the cursor - place it in sector (3,3).
5. Repeat this procedure to place the remaining instances as follows:

$1I4/$1I53	Grid (12,13)
$1I4/$1I55	Grid (12,9)
$1I5/$1I2	Sector (4,4)
$1I5/$1I3	Sector (4,3)
$1I5/$1I53	Grid (16,13)
$1I5/1I55	Grid (16,9)

   When all the instances have been placed, the Place Instances dialog closes automatically. You have now placed the hierarchical instances in the macro.
6. Look at the Design Browser again: the placed instances have a filled rectangle in the box that represents them. The rectangle is colored magenta to indicate that the placement is locked.

Placing Instances in a Selected Hierarchy for AT6K Designs

Figaro also provides a place by hierarchy feature that allows you to place all the instances contained in particular level of hierarchy. This feature is similar to Place By Net and is particularly useful with large and complex hierarchical designs.

It is activated automatically when you select and drag a hierarchical instance from the Design Browser to the Compile window. In this example there is only one hierarchical instance, JKFP2:

1. Make the Design Browser the current window.
2. Select the top level JKFP2 instance and drag it over the Compile window.

   Figaro displays the Place Instances dialog containing all the instances contained within the hierarchical instance you selected. This dialog works in the same way as the Place by Net dialog you just used.
3. Use the dialog to place only the remaining unplaced instances as follows:

   $1I10 in cell location (10,10)

   $1I15 in cell location (8,12)

   $1I16 in cell location (9,10)

   $1I6 in cell location (11,10)

   Please do not move any of the instances you have already placed.
4. Click *Done* to close the Place Instances dialog.

You have now placed all the instances on the device without contention, so the Ini Place and Opt Place buttons have turned green.

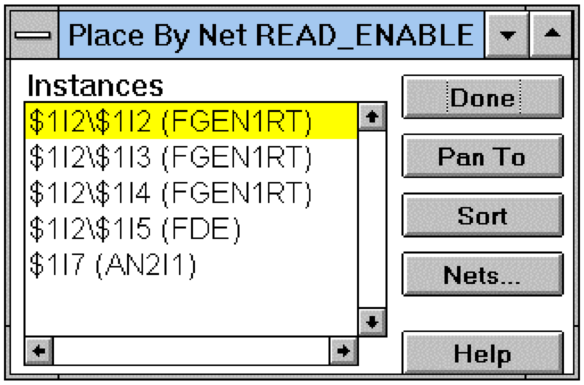
Placing Instances Associated with a Particular Net AT40K Designs

With larger designs or macros, you may find it useful to place instances associated with one or more selected nets.

You can do this using the *Edit>Place By Net* command:

1. Make sure the Compile window is the current window and that nothing in it is selected (use *Edit>Deselect All*).
2. Pull down *Edit>Place By Net.*

   Figaro displays the Place By Net dialog listing all the nets in the RAMCONT macro.
3. Scroll down the list and select the net READ_ENABLE. Figaro displays the Place By Net READ_ENABLE dialog, listing all the unplaced instances associated with the net you selected. The first instance in the list is already highlighted:



Place By Net READ_ENABLE Dialog Box

   The Place By Net dialog functions in the same way as the Place Instances dialog you looked at earlier.
4. Rearrange the windows so that you can see the Compile window and then move the cursor over the Compile window.

   Again the first instance, $1I2\$1I2, is attached to the cursor. (PC users click once to activate the Compile window.)
5. Move the cursor over cell location (11,10) and click once to drop the instance at this location.

6. Move over the Compile window again.

   $1I2\$1I3 - the next instance in the dialog - is now attached to the cursor. Place this in cell location (11,11).

7. Repeat the above procedure to place the remaining instances as follows:

   $1I2\$1I4 in cell location (11,12)

   $1I2\$1I5 in cell location (11,13)

   $1I7 in cell location (11,14)

   The Place By Net READ_ENABLE dialog is now empty so it closes automatically and the "READ_ENABLE" entry is removed from the Place By Net dialog.

8. Now select WRITE_ENABLE in the Place By Net dialog and repeat the above procedure for its associated instances:

   $1I3\$1I2 in cell location (15,10)

   $1I3\$1I3 in cell location (15,11)

   $1I3\$1I4 in cell location (15,12)

   $1I3\$1I5 in cell location (15,13)

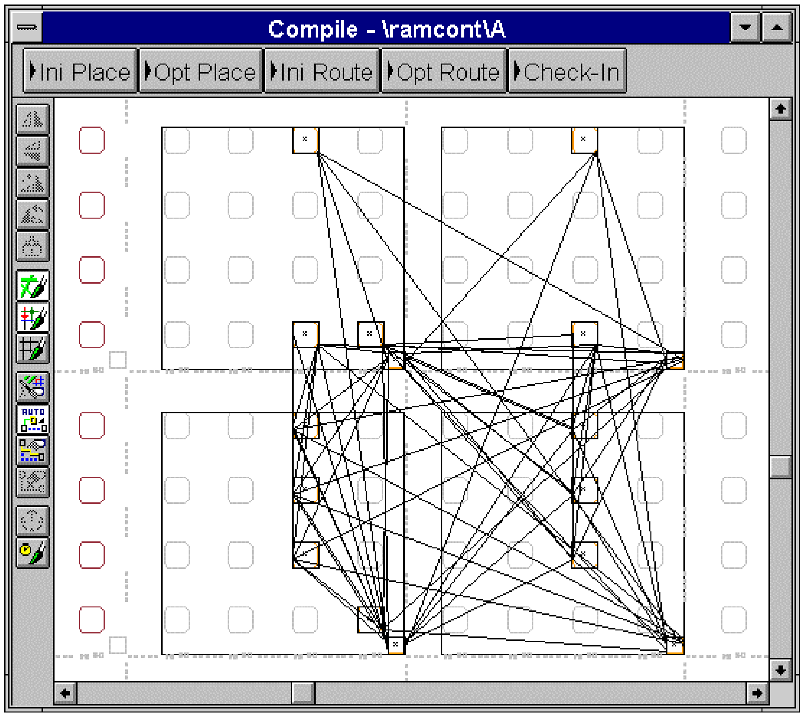
   $1I8 in cell location (15,14)

   The Place By Net WRITE_ENABLE dialog is now empty and closes automatically and the "WRITE_ENABLE" entry is removed from the list in the main Place By Net dialog.

9. Click *Done* to close the Place By Net dialog box.

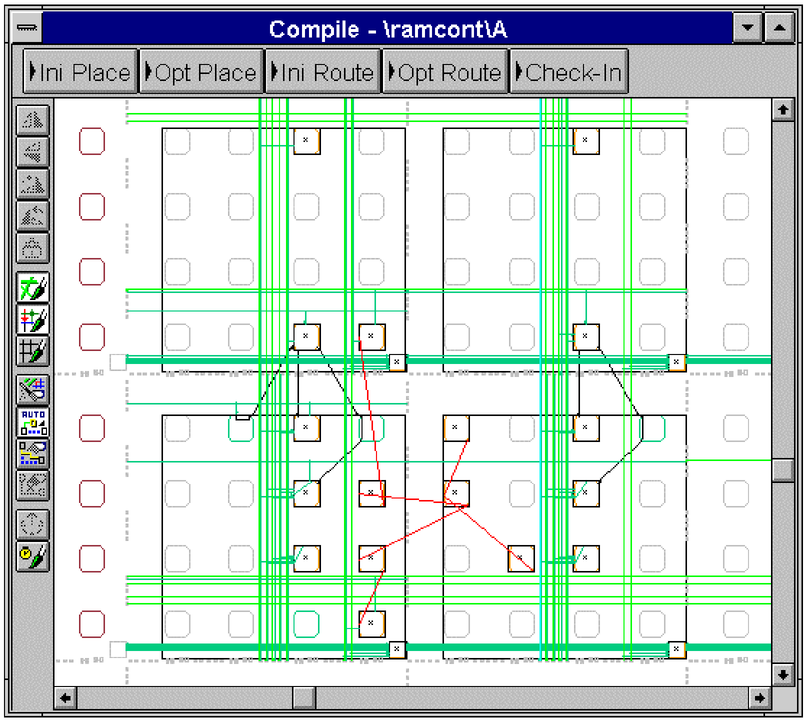
   There are now only three unplaced instances in the Design Browser. You will use the automatic tools to complete the macro. Using *Edit>SelectinArea* and *Edit>CloseLocations*, close off the 5 unused sectors which are still open. The remaining open area on the device should be the 4 sectors containing the RAM IO.

The Compile window now looks like this:



Components of RAMCONT Placed

10. Pull down *Edit>Route Unrouted Nets* to route the pre-placed sections of your design.



Components of RAMCONT Routed

You now have the choice of locking these before running the automatic tools.

Placing Instances Associated with a Net for AT6K Designs

You can place an instance manually simply by selecting it in the Design Browser, dragging it over the Compile window and dropping it in the required cell location. However, with larger designs or macros, you may find it useful to place instances associated with one or more selected nets.

You can do this using the *Edit>Place by Net*:

1. Make sure the Compile window is the current window and pull down *Edit>Place By Net.*

   Figaro displays the Place by Net dialog listing all the nets in the JKFP2 macro.
2. Select $1N19, the first net in the list.

   Figaro displays the Place by Net $1N19 dialog. This lists all the unplaced instances associated with that net. The first instance in the list is highlighted.
3. Rearrange the windows so that you can see the Compile window and then move the cursor over the Compile window.

   You will see the first instance, $1I14, is "attached" to the cursor.
4. Move the cursor over cell location (10,12) and click once to "drop" the instance at this location (the status bar shows the cell location in the bottom right corner of the screen.)
5. Click on the Compile window again. You see that $1I9 - the next instance in the dialog - is now attached to the cursor. Place this instance in cell location (9,12).

   The Place by Net $1N19 dialog is now empty so it closes automatically and the "$1N19" entry is removed from the Place by Net dialog.
6. Select net $1N26 in the Place By Net dialog and repeat the above procedure for its associated instances:

   $1I2 in cell location (12,9)

   $1I4 in cell location (11,12)
7. Close the Place By Net dialog box.

Look at the Design Browser. You have placed four of the eight instances in the JKFP2 macro and the placed instances now have a filled rectangle in the box that represents them. The rectangle is colored magenta to indicate that the placement is locked.

Port Permutations

The unique design of the AT40K core cell means that many of the functions performed by macro instances can be implemented using different permutations of the various ports available in the core cell. Together with the ability to connect directly to both orthogonally and diagonally-adjacent cell locations, this allows you significant freedom when swapping nets between ports of a similar type.

Figaro's placement and routing tools take full advantage of this flexibility by automatically permuting ports in order to achieve optimal results in the compiled design. feature is controlled by the place and route option *Allow auto-pin swap on locked macros*, which allows Figaro to permute ports on manually-placed or otherwise locked macros in order to achieve direct connects.

NOTE The auto-pin swap option is defaulted to on and it is recommended that you keep it activated at all times. If you switch it off, Figaro would be able to perform port permutations only on unlocked macros, significantly reducing the effectiveness of the automatic tools.

Manual Port Swapping

This example is for AT40K devices only.

You can swap ports manually on placed macro instances in routed, partially routed and unrouted designs. When you swap a port, the macro instance in question is automatically locked and the new port configuration for that instance will be retained if you subsequently move it to a new location.

This section shows you how you can swap ports to optimize the connections between adjacent instances:

1. Go to the Compile window and select instance $1I4\$1I53 (INV).
2. Move the selected instance up one location so that it is immediately right of $1I7.

   Figaro automatically re-routes the net connecting the two instances using local buses and the delay, as reported in the transcript, shows a significant increase:

   The net was re-routed automatically because the place and route option *Auto re-route after manual move* is switched on.

   However, this feature does not incorporate auto-pin swap, so Figaro was forced to use the existing port configuration and use local buses instead.
3. Select the arrowhead entering the X-input of instance $1I4\$1I53 (INV) and hold down the left mouse button to drag the port.

   When you move the cursor, you see the name of the port, X(A\$1I4\$1I53\A), written next to it. Figaro also displays a number of other resources in blue: these are the legal destinations for this port.
4. Holding down the left mouse button, drag the port to the Y-input entering at the left of the same cell, and release the mouse button to "drop" the port there (use the middle section of the status bar to see which port you are at).

   Figaro swaps the port and re-routes the net between $1I4\$1I53 and $1I7 on a direct connect:

   Note also that the delay reported in the transcript shows a significant reduction.

Changing the Placement

If you want to make any changes to the way a particular instance is placed, use the left mouse button to select the instance in question in the Compile window, hold it down to drag it to its new location and let go to drop it there.

If you change your mind while still "holding" the instance, use the right mouse button to cancel the action and return it to its original location. If you have already dropped the instance, pull down *Edit>Undo* to reverse the previous action.

If you want to drag the instance to a position not currently displayed in the window, pick up the instance, move the cursor to the edge of the window and watch the display scroll automatically. Pressing the SHIFT key at the same time means Figaro scrolls a window at a time. You can also use the *Zoom* and *Pan* functions to get to the required location.

If you want to move more than one instance, use the SHIFT or CTRL keys and click the left mouse button to pick up all the objects you want.

You will get to know some of the other manual placement commands later when you incorporate the user macro in a full design. For now, try out some of the basic operations described above and then return the instances to their original locations.

Before You Proceed

Once you have finished experimenting with manual placement, click the *Opt Place* button to place the remaining logic on the device and then click *Ini Route* to route the nets in the design.

Manual Routing Display

To route nets manually, you have to enter *manual routing display mode*. This feature simplifies manual routing tasks using special graphics and colors to help you find your way through the diverse routing resources available in the device.

To enter manual routing display mode, select a net in the Compile window and pull down *Edit>Edit Net Routing*. The Compile window display changes, highlighting the resources associated with the selected net and suppressing everything else:

Source and Destination Instances

The source of the net is displayed in blue, while its destination is colored red (there may be more than one destination).

Working Net

The net you selected is called the working net and is displayed in green. If you have already run initial routing, this net will be connected, so you'll have to discard the existing connection using *Edit>Discard Route* before you can route it manually.

Interconnect

The interconnect between the two instances is denoted by an *angled flightline* between the source and destination. This flightline remains until you find a valid route from the source to the destination, or until you pull down *Edit>Finish Net Routing*.

Current Point

If you zoom in on your source instance you will see some orange-colored outputs from the cell. The place you are routing from at any time is called the *current point* and its default color is *orange*.

After you discard an existing route, but before you click anywhere, the current point reverts to the source. When you start manual routing, the current point is always the last point you clicked on.

Dynamic Highlighting

As you move the cursor around the Compile window, you will see Figaro dynamically highlight different resources. This facility shows which resource will be used when you click the mouse button. Use it to make sure you are selecting what you intended.

There are also many internal connection possibilities within the cell that you can also use. These are not highlighted, however: simply select the bus or net you want to use and see what happens. Consult the relevant data book for guidelines.

Routing Manually

When you start manual routing, the net will start from the source, and connect to any routing or cell resource you click in the device. You don't have to click on every resource you want to use: Figaro works out intermediate stations automatically.

If you make a mistake, *Edit>Undo* removes the last connection you added. You can also use *Edit>Discard From* to step back along the route to the last correct point, or you can click on a point along the route and discard the rest of the net from there.

If you select a resource that would require ports to be swapped on either the source or destination, Figaro will make the required swap automatically, provided this still allows a legal configuration of the cell.

NOTE It is advisable to have the data book diagram of core cell connectivity to buses at hand when routing manually. You would normally use these functions only if you are completely familiar with the architecture being used.

You can now either go on and experiment with the techniques described to route more of the nets in the macro manually, or have Figaro complete the routing for you using *Edit>Route Unrouted Nets*.

When you have finished, click the *Opt Route* button to optimize the routing in the macro and remove any remaining contentions.

Contention and Locked Routing

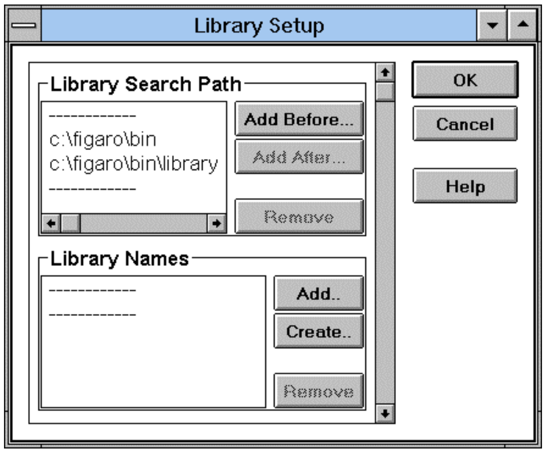
Bear in mind that any nets you route manually are automatically locked and cannot be changed by the automatic tools. This constrains the operation of these tools, and can lead to some unorthodox routes or even failure at optimize routing stage. If you do encounter any contention problems at this stage, you may have to unlock some of the manual placement and routing you performed to allow the automatic tools to operate more freely.

Checking a User Macro Into a Library

When you've finished compiling your macro, you have to check it into a library where it is made available for use with other designs. The check-in procedure **moves** the design files related to the macro into the specified library directory and closes all the windows on your desktop.

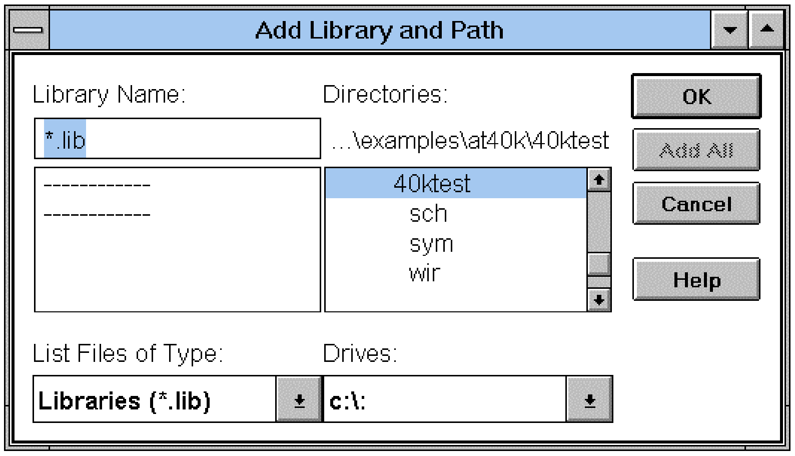
You can set up the library either right at the beginning of the flow, using *Library>Library Setup* and steps 2 and 3 below, or directly from the Check-In User Macro dialog:

1. Click the *Check-In* button on the Compile window flowbar. Figaro opens the Check-In User Macro dialog.
2. Click *Library Setup* to display the following dialog:



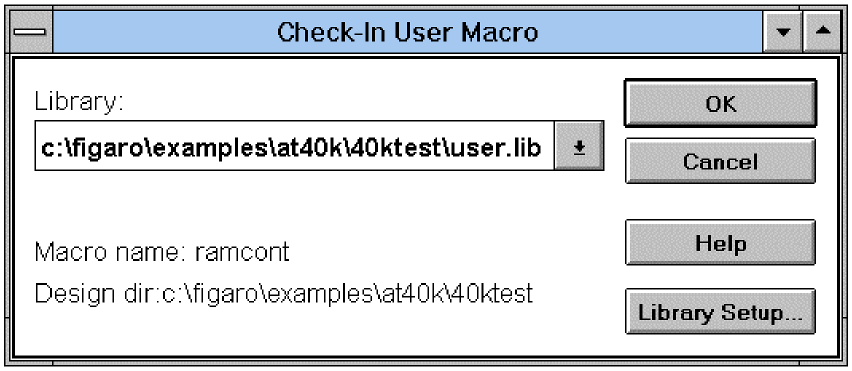
Library Setup Dialog Box

3. Click *Add Before* and set up the design directory ...\SYSTEMDESIGNER\EXAMPLES\AT40K\VLOGIC\TEST40K.



Add Library and Path Dialog Box

4. Enter "user.lib" in the "Library Name" field, click *OK* and confirm that you want to create the library "user.lib".
   The library name and search path are added to the Library Setup dialog.
5. Click *OK* to return to the Check-In User Macro dialog. The new library is entered in the field provided:

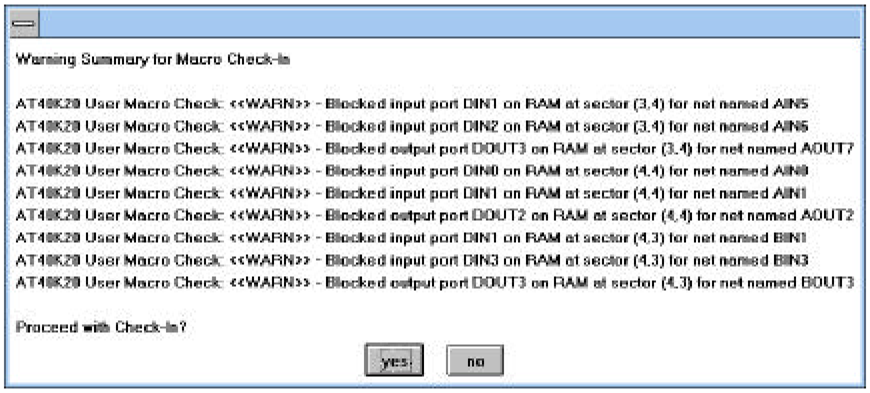

Check-In User Macro

Library:
c:\figaro\examples\at40k\40ktest\user.lib

Macro name: ramcont
Design dir:c:\figaro\examples\at40k\40ktest

Check-In User Macro Dialog

6. Click *OK* again to check-in the user macro.
   Figaro checks the macro and raises a dialog warning you of any problems:


```
Warning Summary for Macro Check-In

AT40K20 User Macro Check: <<WARN>> - Blocked input port DIN1 on RAM at sector (3,4) for net named AIN5
AT40K20 User Macro Check: <<WARN>> - Blocked input port DIN2 on RAM at sector (3,4) for net named AIN6
AT40K20 User Macro Check: <<WARN>> - Blocked output port DOUT3 on RAM at sector (3,4) for net named AOUT7
AT40K20 User Macro Check: <<WARN>> - Blocked input port DIN0 on RAM at sector (4,4) for net named AIN0
AT40K20 User Macro Check: <<WARN>> - Blocked input port DIN1 on RAM at sector (4,4) for net named AIN1
AT40K20 User Macro Check: <<WARN>> - Blocked output port DOUT2 on RAM at sector (4,4) for net named AOUT2
AT40K20 User Macro Check: <<WARN>> - Blocked input port DIN1 on RAM at sector (4,3) for net named BIN1
AT40K20 User Macro Check: <<WARN>> - Blocked input port DIN3 on RAM at sector (4,3) for net named BIN3
AT40K20 User Macro Check: <<WARN>> - Blocked output port DOUT3 on RAM at sector (4,3) for net named BOUT3

Proceed with Check-In?
```

Warning Summary for Macro Check-In

Here, several of the RAM ports in the macro are reported as "blocked": this is because the macro contains no data connections to the RAM cells, and Figaro has chosen to use some of these resources for macro routing. Although not critical at the moment, these blockages will cause difficulties when you incorporate the macro in the final design.

To resolve the problem here would result in some complex and inefficient manual routing, so unless you are an experienced user of the AT40K architecture, it is advisable to proceed with macro check-in and then *soften* the macros at design level later (see "Softening Library Routing").

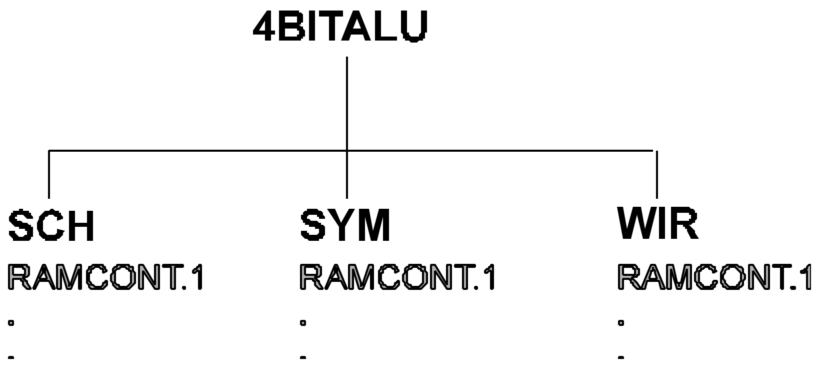
NOTE If routing at the design level still fails to produce a satisfactory result, you may want to consider checking-in the macro with no routing at all and allowing Figaro to allocate all the routing at the design level.

7. Click *Yes* to proceed with the check-in and clear the desktop.
   To edit this macro again, you must first check it out of the library. This moves all the relevant files back and restores the desktop to allow you to work. Scroll back in the transcript to see details of the files that were moved during check-in. This is explained in more detail in "User Macro Files and Directories" below.

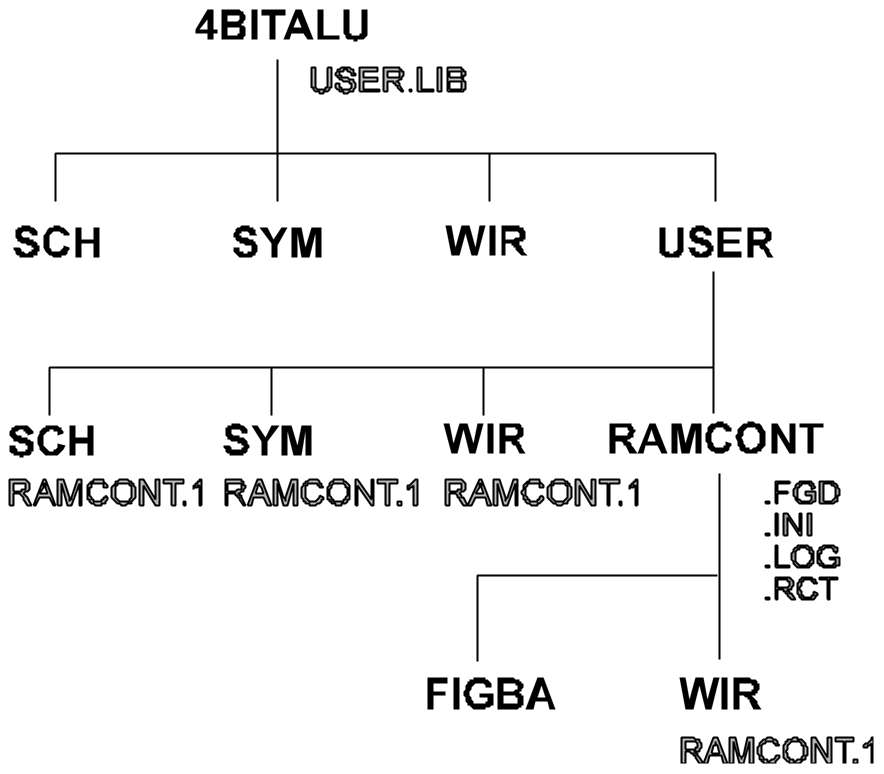
User Macro Files and Directories

Creating a user macro changes your file setup, creating some directories and files and moving others to new locations. In the above example, the file structure changes as outlined below (directories are shown in bold text, files in lighter text).

Before compiling the user macro, the file structure is as follows:



The following diagram shows the new files and directories created when you produced the macro RAMCONT:



The sequence is:

1 When you set up the library, Figaro creates the USER directory and the user macro library file USER.LIB.

The directory will contain the macro design files while the .LIB file contains information on the macros that have been created and checked into this library. Its format is the same as that of design FGD files, although it contains different information, used when an instance of the macro is incorporated in a design: ports, instances and nets along with placement and routing data for each macro cell.

2 When you check macro JKFP2 (AT6K) or RAMCONT (AT40K) into the library, subdirectories SCH, SYM and WIR are created. Directory JKFP2 or RAMCONT is also created to hold the files that contain and control the user macro.

3 Under JKFP2 or RAMCONT, a new WIR sub-directory is created and the files from the original design directory (previous diagram) are **moved** there. These files are not changed in any way and are moved rather than copied to ensure only one copy of the component exists, either in a design directory, or in a library.

NOTE If you use *Library>Delete Macro* to delete the user macro, Figaro deletes all the files associated with the user macro, including any original design files.

4 If you subsequently check other user macros into the USER library, Figaro creates a separate sub-directory to hold all the files for this macro. Again, a new sub-directory WIR is created and the files from the original design directory are **moved** there. The existing SCH, SYM, and WIR directories are updated as required.

Hierarchical macros

If the macro has a hierarchy of design cells, Figaro does not move the entire hierarchy, in case the lower-level cells are used elsewhere in the design. This kind of macro is non-portable: it cannot be checked into a library from one design directory and then checked-out for editing when used in another design directory.

Creating a User Macro Automatically

Although you will generally use manual editing when producing user macros, you can of course also use Figaro's automatic functions. For example, you might use the automatic tools

- To obtain a preliminary layout as a starting point before optimizing the design manually
- To finish off compilation after you have placed critical sections by hand.

To use the automatic tools, open the macro netlist in the usual way and wait for Figaro to add a part to be used as a device template. Then, simply click the Compile button in the Figaro window and respond to the questions Figaro asks.

In some circumstances, the automatic tools place instances on the closed locations and report these as being in contention. This is usually because there is too much congestion within the boundaries of the open locations, or with large multi-cell macros, the placer only ensures that the origin is in an open location. For either case you can resolve the contention by

a) opening the locations concerned.
b) moving the offending instance to one of the open cell locations.
c) locking the instance.

If you manually place a cell on a closed location, then it will not be reported as contention, as it is taken to be a user constraint that overrides the automatically closed location. It is still best to open the locations you wish to use, along with a few surrounding them. If the router is required to go through cells, it will have difficulty, as it tries not to use closed locations. This may result in some unorthodox routes.

Incorporating User Macros in a Design

In this section you will re-open the your design and incorporate the user macro you have just created. You will learn about some of the Compile window functions you can use to make manual adjustments to this and other instances, before using the automatic tools to compile the design.

Opening the Design

To open a design containing user macros, do the following:

1. Pull down *File>Design Setup* and enter 4BITALU (AT6K) or TEST40K (AT4-K) in the Design Name field (use *New Design* to set this up if necessary).

   Click *OK* to return to the desktop.
2. Pull down *Library>Library Setup* and set up the path to the library you created in the previous section, USER.LIB.

   Click *OK* to return to the desktop
3. Open the options dialog, go to the "Design Constraints" topic and check the box marked "Auto-import Repeat Constraints".

   When you saved this design at the end of the previous tutorial, Figaro saved a record of the part you used in the file 4BITALU.RCT (AT6K) TEST40K.RCT (AT40K). This is called a repeat constraints file. When you open a design for which there is a .RCT file, Figaro automatically adds the part specified in the file and makes locks for any pins defined. Using the constraints file can save time, especially when you've locked pins.

. (If the file 4BITALU.RCT (AT6K) or (If the file TEST40K.RCT(AT40K) has been moved or deleted, you'll have to add an AT6003-2JC (AT6K) or AT40K05-2JC (AT40K) part as before. Refer to “Specifying the Target Device” in Chapter 3.)

4. Pull down *File>Open as Design*, set "Files of Type" to "ViewLogic Wir (*.1)" and click *OK* to load the design.

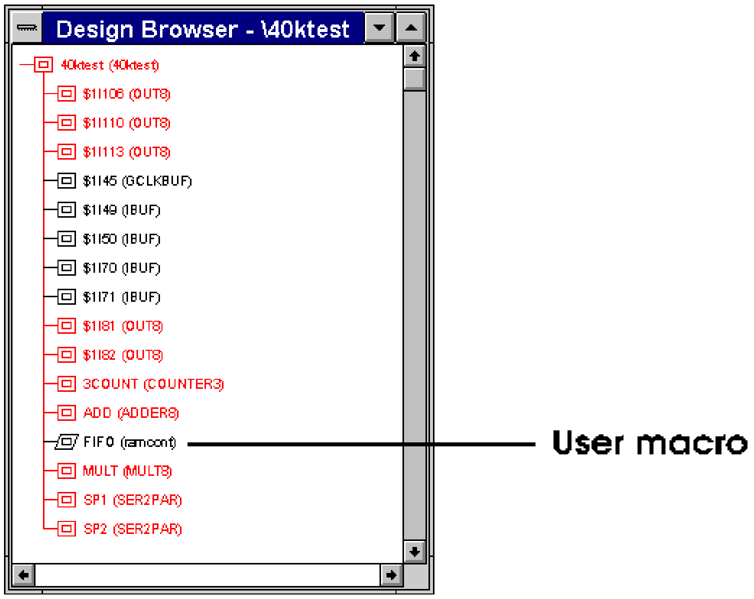
   Figaro loads the design and adds the part specified in the repeat constraint file.

5. Click *Partition*.

User Macros in the Design

1. In the Design Browser, you should see the hierarchical instance $1I72(CONTROL) (AT6K) or FIFO(RAMCONT) (AT40K).

   Note the different shape used to represent user macros in the Design Browser:



```
40ktest (40ktest)
  $1I106 (OUT8)
  $1I110 (OUT8)
  $1I113 (OUT8)
  $1I45 (GCLKBUF)
  $1I49 (IBUF)
  $1I50 (IBUF)
  $1I70 (IBUF)
  $1I71 (IBUF)
  $1I81 (OUT8)
  $1I82 (OUT8)
  3COUNT (COUNTER3)
  ADD (ADDER8)
  FIFO (ramcont)
  MULT (MULT8)
  SP1 (SER2PAR)
  SP2 (SER2PAR)
```

User Macros in the Design Browser

2. Open a Compile window, select $1I46(JKFP2) (AT6K) or FIFO(RAMCONT) (AT40K) from the Design Browser and, with the left mouse button held down, drag it over the Compile window. You see the shape of the macro you created earlier.

3. Release the mouse button over the center four sectors of the device to drop the macro.

   The macro cannot legally be placed in this location, so it automatically snaps to the nearest legal location.

4. Zoom in to get a better view of your user macro and look at the toolbar on the left-hand side of the Compile window. Move the cursor over the individual buttons and look in the status bar at the bottom of the desktop to see what each one does.

5. With the JKFP2 (AT6K) or RAMCONT (AT40K) macro selected, press the toolbar button to flip it horizontally.

   Figaro issues an error message in the transcript informing you that it failed to change the orientation of the RAMCONT macro. This is because the RAM locations in the AT40K architecture have alternating addressing layout and therefore components can only be placed on a region with an appropriate layout. Because the proposed manual move is illegal, Figaro rejects it outright.

   Try moving the RAM one sector to the right. It does not move as this is not a legal location.

   Now try moving it up two sectors. Here, the device layout is suitable to accommodate the macro, so Figaro completes the manual move.

Compiling the Design

Experiment further by placing a few of the non-I/O instances in the Design Browser by hand. Once you have placed an item, look at the box next to it in the Design Browser: note that it now contains a filled rectangle, allowing you to see which ones have already been placed.

With smaller designs, it is advisable to try and place the instances reasonably close together so that you can zoom in to a higher magnification yet still work with them all in view:

1. Make the Compile window the current window.
2. Find the toolbar button used to Display or Hide Unrouted Nets and set it to Display. (When it is set to Display it is a lighter gray color.)

   You will see green flightlines showing the interconnect between the instances you placed.
3. Zoom in on this area using *View>Zoom to Area*.

   Try moving the instances around relative to each other and watch the flightlines change.
4. Pull down *Edit>Route Unrouted Nets* and see if there are any bus connections that could become direct connects with different port permutations.
5. When you have finished experimenting, press *Ini Place*.

   This will place all the remaining instances along with the design I/Os. Instances placed by Figaro will be displayed in a different color because they are not locked.

NOTE You must use automatic placement if you want to retain any pin locks you assigned your I/Os in the Parts window. If you place the I/Os manually, Figaro assumes you want to override these constraints.

6. Press *Opt Place*, which will move the logic and I/Os closer together. Note that none of the manually placed instances have been moved.
7. Now press *Opt Route*, to route the design.

   Figaro is unable to remove all of the contention in the design because of the blocked RAM ports you were warned about in the "Checking a User Macro Into a Library" section. To solve this problem, see "Softening Library Routing" below.

Softening Library Routing

Macro nets often have routes predefined for them in user or vendor macro libraries. Figaro uses this library description as a blueprint for creating locked routing for these macros in the implementation. If the routes of two such nets coincide, the automatic tools will soften the library routing.

To soften all macro routing in the design, switch on the place and route option "Auto-soften conflicting hard routing in Optimize Route". Nets are softened at the start of the optimize route step. You can also soften library routing manually as described. .

Softening macro routing involves unlocking the predefined routing for user and/or library macros to afford the automatic tools more freedom to resolve route contention in the final implementation.

You soften macro routing for selected macros using the *Edit>Soften Library Routing* command:

1. Switch off the options "Auto-select input nets" and "Auto-select output nets" in the desktop toolbar.
2. Select each of the resources showing contention, either in turn or using SHIFT-click to select them all at once.
3. Pull down *Edit>Soften Library Routing*. You could then rerun the automatic routing.

You can still perform manual placement after routing if you wish and you can also swap ports (as described in the macro section) at the design level.

If your manually-placed instances make the design impossible to place and/or route, you may have to unlock some or all of these to allow the automatic tools to operate more freely. Use *Edit>Find* to create a list of locked instances and unlock these as required using *Edit>Unlock*.

Summary

Below is a summary of the main steps in manual editing:

1. Place any critical sections of your design manually.
2. Route any unrouted nets.
3. Use manual port permutations as required to achieve the best connections.

4. Move instances again if necessary.
5. Use automatic placement and routing to compile the rest.
6. Make any final manual improvements required.

Mapping

Mapping is the process of optimizing design logic and adapting it to a specific vendor architecture.

You can use mapping:

- To reduce the area required to accommodate a design netlist created using the same library as the target device. You should run mapping for all AT40K designs. The only exception is when you are using dynamic macros to specify the function of each core cell and do not wish the mapper to make any changes to these.
- To read in a design created using a different library from the target device, for example when implementing an AT6K design on an AT40K part. In this situation, you **have to** map the design to convert the device library information to be compatible with the new device, so Figaro automatically enables the Mapping step.

In both cases, mapping takes the instances from the design netlist and performs some reduction of the area required by the design, and then converts them to instances that are specific to the selected device.

You can use the Map Browser to view the results of this process. Note that many of the names will have been changed by mapping, but that associations between the original and mapped versions are indicated by cross-highlighting between the two browsers.

This tutorial describes when to use the mapping function, introduces the cross-highlighting feature and highlights the potential benefits of using the various preservation features. If you are using an AT40K device, this tutorial uses the TEST40K design stored under \SYSTEMDESIGNER\EXAMPLES\AT40K\VLOGIC\TEST40K in your main Figaro directory. If you are using an AT6K device, use the 4BITALU design stored under \SYSTEMDESIGNER\EXAMPLES\VLOGIC\4BITALU in your main Figaro directory

Use *File>Design Setup* to set up a **new** version of this design now, using the AT6K or AT40K configuration - depending on the device you are using- and "Viewlogic-Workview Office" tools flow.

When To Map the Design

Design and Device use the Same Library

If your design uses the same library as the target device, mapping is optional but nevertheless **strongly encouraged**. If you do not want to map the design, you have to disable the function manually in the Options dialog.

Example: Mapping from ViewLogic. For a more detailed example of mapping a design from ViewLogic, see below.

The benefits of mapping in this situation depend largely on the quality of the original design.

For a rough design, mapping will improve the implementation: this might be the case if the designer is unfamiliar with the architecture, or has devised the schematic with no particular device in mind.

Even if the original design is well-planned and has been conceived specifically for the current architecture, the *Map* step may still be able to find scope for improvement by performing some local optimization of the functions defined for any FGEN cells used. For example, if the design contains the equation:

```
A*B*!C + A*!B + A*B*C
```

Then the mapper will optimize away the redundant logic driving the B and C inputs and reduce the equation to the more optimal A.

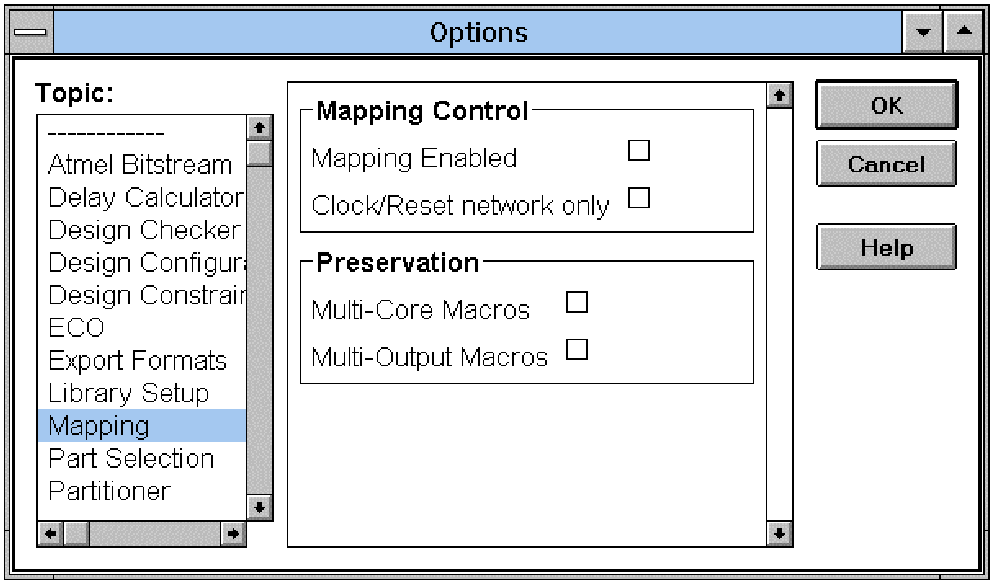
The sections following cover the steps in mapping a ViewLogic design that uses the same library as the target device. It shows you how to set mapping options and map the design, before examining the effects of preservation in more detail.

Design and Device use Different Libraries

There may be some cases where the design you want to import uses a library different from that of the target device (for example, when implementing AT6K designs on AT40K parts). In cases like this, the design must be mapped: Figaro detects this situation on import and automatically enables the mapping option for you.

Mapping Options

Before you can map the design, you have to enable mapping in the Options dialog. To open this dialog, pull down *Options>Options* and select *Mapping* from the topic list. Figaro displays the dialog below:



Mapping Control
Mapping Enabled ☐
Clock/Reset network only ☐
Preservation
Multi-Core Macros ☐
Multi-Output Macros ☐

Default Mapping Options

Mapping Control

The "Mapping Enabled" checkbox enables and disables the Map step within the design flow. If you load a design using a library different to that used by the target device, Figaro enables this option automatically.

The second checkbox is labeled "Clock/Reset network only", meaning that Figaro will optimize only inversions on the clock/reset distribution network. This prevents inversions that could be implemented directly on these resources from being implemented as derived clocks/resets.

Preservation

The options in the "Preservation" pane allow you to "protect" certain features of the original design through the map step Checking a box means the relevant feature is preserved, clearing it allows the mapper more freedom.

In addition to using these global options, you can also preserve items:

- Using the *Edit>Constraints* command for selected objects within Figaro.
- Using Atmel library defaults to preserve certain types of cell.
- Using attributes in the original schematic or design file.

These are described in more detail below.

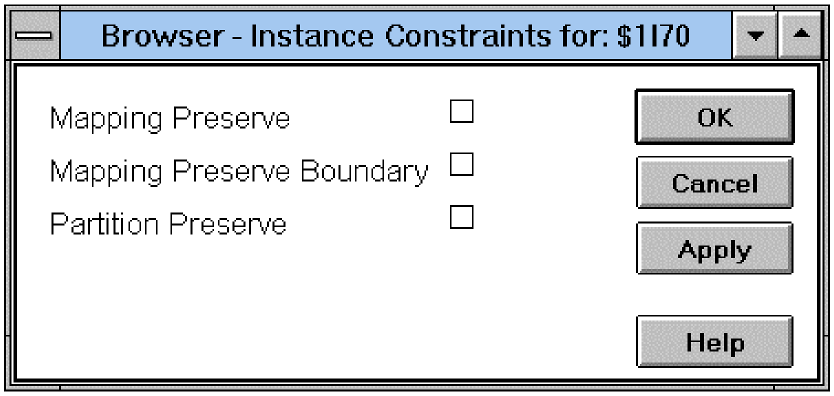
Setting and removing constraints on selected instances

While the mapping options described above are applied globally to all the instances in the design, you can preserve individual or selected hierarchical or macro instances using the *Edit>Constraints* command:

1. Open the Options dialog, select "Design Constraints" from the Topic list and switch off the "Auto-import Repeat Constraints" option. (This means that when you import the test design, Figaro will not import any existing .RCT file from a previous session.)
2. Use *File>Open as Design* to import the 4BITALU netlist (AT6K) or TEST40K netlist (AT40K).

3. Make the Design Browser the current window, select one of the hierarchical instances and pull down *Edit>Constraints*.

   Figaro displays the Constraints dialog with the following checkboxes (for non-hierarchical instances, only the "Mapping Preserve" constraint is displayed):


Browser - Instance Constraints for: $1I70
Mapping Preserve
Mapping Preserve Boundary
Partition Preserve
OK
Cancel
Apply
Help

Setting Constraints for Hierarchical Instances

   "Mapping Preserve" preserves the selected instances through the *Map* step, while in multichip designs, the "Partition Preserve" constraint tells Figaro that the component parts of the selected cell or instance are not to be split between the different devices in the implementation.

   "Mapping Preserve Boundary" has no meaning in the AT40K configuration.

4. To set constraints, check the relevant boxes and click *Apply*. The transcript reports the constraints applied.
5. When you have finished, simply click *OK* to close the dialog.

You can also use the *Edit>Constraints* command to **remove** constraints placed on selected instances, including those set by library defaults (see below). Simply select the instances, call up the dialog and clear the checkbox. You'll see how this is done in the section "Overriding the Defaults".

Before you go on, make sure you remove any constraints you set above.

Atmel library defaults

There are several types of macros to which Figaro does not map in AT40K, most notably multi-output macros. These macros would always be mapped to a series of single-output equivalents, leading to an overall increase in the area required to accommodate your design.

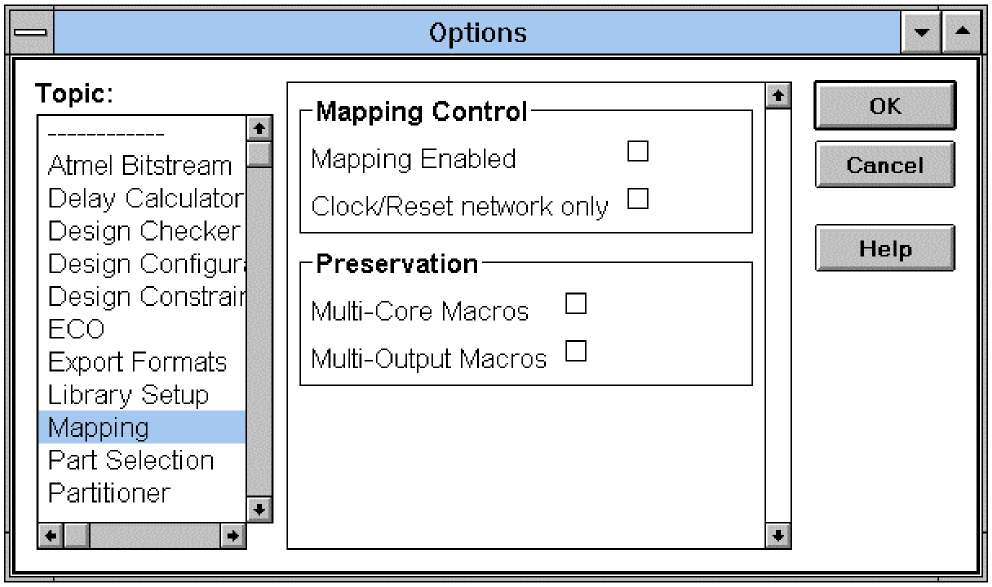
To avoid this, Atmel library definitions for generic multi-output macros default the mapping preservation constraint to "On". This means that when you import a design containing an instance of a multi-output macro, the mapping preservation constraint is set automatically for that instance.

You can, of course, still map these macros by removing the default constraint using the *Edit>Constraints* command described above.

Mapping a Design

1. Load the netlist \SYSTEMDESIGNER\EXAMPLES\VLOGIC\4BITALU\4BITALU.1 (AT6K) or \SYSTEMDESIGNER\EXAMPLES\TEST40K\TEST40K.1 (AT40K) if you haven't already done so. (Again, make sure the RCT import option is switched off before you do so.)
2. Pull down *Options>Options* and select "Mapping" from the Topic list.

   Enable mapping, but leave the other options switched off:



Mapping Control
Mapping Enabled ☐
Clock/Reset network only ☐

Preservation
Multi-Core Macros ☐
Multi-Output Macros ☐

Mapping Options Dialog Box

3. Click *OK* to confirm your selection and return to the desktop.

   You see that the *Map* flowbar button has been activated.
4. Click the *Map* button.

   Figaro maps the design and opens a Map Browser where you can view the results.

The Map Browser

The Map Browser shows the logic elements of your design, taking account of changes to the original after mapping.

Associations and Cross-Highlighting

Wherever possible, Figaro maintains *associations* between the instances and nets displayed in the Design Browser and their counterparts in the Map Browser. When you select an instance in one browser, Figaro automatically highlights and scrolls to its associated instances in the other one. This allows you to trace original components in the current implementation of your design. These associations might be:

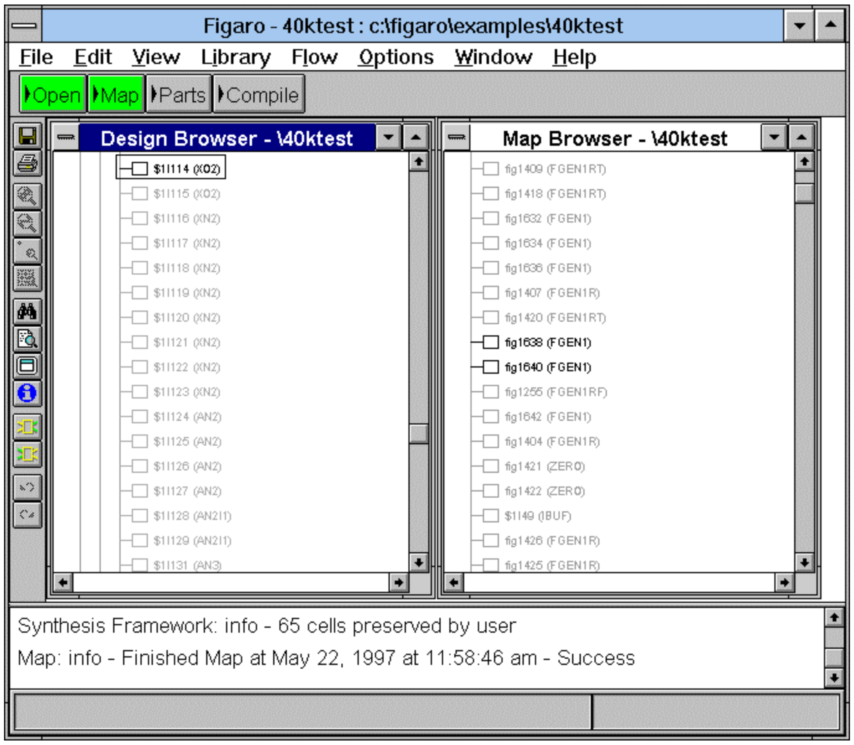
One-to-One	Single design cell to single mapped cell.
One-to-Many	Single design cell to multiple mapped cells.
Many-to-One	Multiple design cells to single mapped cell.
Many-to-Many	An instance broken down into several components which are mapped to several larger instances, which may also include other instances from the original design.
One-to-Zero	Here, an instance is mapped out of the design entirely (for example, a double inversion).

Clearly, using the Design Browser to place components of a mapped design could lead to several, one or no mapped instances being dragged from the Map Browser. To avoid confusion, you should **always** use the Map Browser when performing manual partitioning and placement. It represents the physical implementation of your design, and each instance displayed there can be placed individually in the target device.

Using Cross-highlighting

1. Find instance FIFO(RAMCONT) in the Design Browser and double-click on it to expand the hierarchy.
2. Expand instance $1I70(MULT4)\SUM3(ADD8)\$1I18(FULL_ADD) (AT6K) or $1I6(ENABLE) and the select instance $1I131 (AT40K).

   Figaro highlights two instances in the Map Browser. This is an example of a one-to-many association:


Figaro - 40ktest : c:\figaro\examples\40ktest
File Edit View Library Flow Options Window Help
Open Map Parts Compile
Design Browser - \40ktest
Map Browser - \40ktest

$1I114 (XO2)
$1I115 (XO2)
$1I116 (XN2)
$1I117 (XN2)
$1I118 (XN2)
$1I119 (XN2)
$1I120 (XN2)
$1I121 (XN2)
$1I122 (XN2)
$1I123 (XN2)
$1I124 (AN2)
$1I125 (AN2)
$1I126 (AN2)
$1I127 (AN2)
$1I128 (AN2I1)
$1I129 (AN2I1)
$1I131 (AN3)

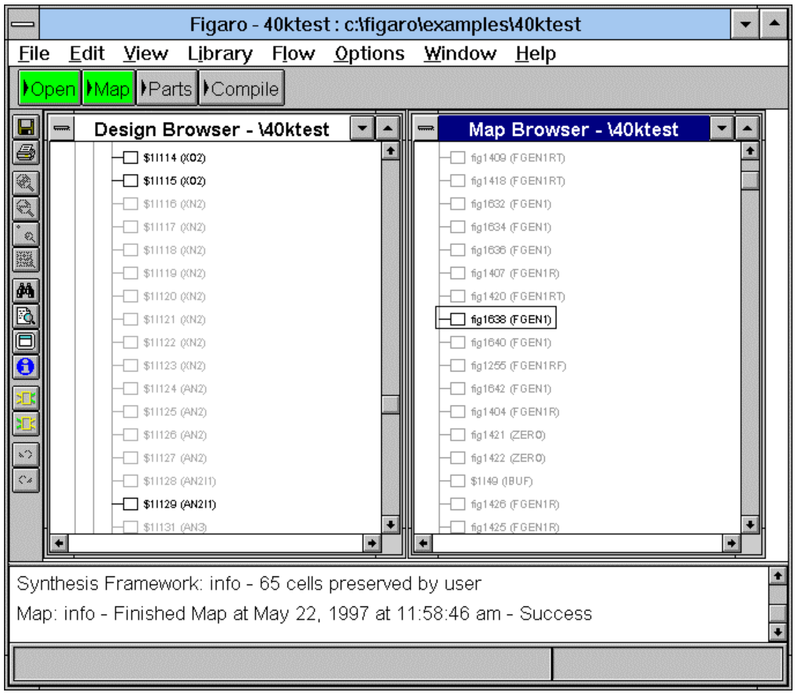
fig1409 (FGEN1RT)
fig1418 (FGEN1RT)
fig1632 (FGEN1)
fig1634 (FGEN1)
fig1636 (FGEN1)
fig1407 (FGEN1R)
fig1420 (FGEN1RT)
fig1638 (FGEN1)
fig1640 (FGEN1)
fig1255 (FGEN1RF)
fig1642 (FGEN1)
fig1404 (FGEN1R)
fig1421 (ZERO)
fig1422 (ZERO)
$1I49 (IBUF)
fig1426 (FGEN1R)
fig1425 (FGEN1R)

```
Synthesis Framework: info - 65 cells preserved by user
Map: info - Finished Map at May 22, 1997 at 11:58:46 am - Success
```

One-to-Many Mapping Association

3. In the Map Browser now, click on one of the highlighted instances.

   Figaro highlights three leaf instances in the Design Browser. This is an example of a many-to-one association:


Figaro - 40ktest : c:\figaro\examples\40ktest
File Edit View Library Flow Options Window Help
Open Map Parts Compile
Design Browser - \40ktest
Map Browser - \40ktest

$1I114 (XO2)
$1I115 (XO2)
$1I116 (XN2)
$1I117 (XN2)
$1I118 (XN2)
$1I119 (XN2)
$1I120 (XN2)
$1I121 (XN2)
$1I122 (XN2)
$1I123 (XN2)
$1I124 (AN2)
$1I125 (AN2)
$1I126 (AN2)
$1I127 (AN2)
$1I128 (AN2I1)
$1I129 (AN2I1)
$1I131 (AN3)

fig1409 (FGEN1RT)
fig1418 (FGEN1RT)
fig1632 (FGEN1)
fig1634 (FGEN1)
fig1636 (FGEN1)
fig1407 (FGEN1R)
fig1420 (FGEN1RT)
fig1638 (FGEN1)
fig1640 (FGEN1)
fig1255 (FGEN1RF)
fig1642 (FGEN1)
fig1404 (FGEN1R)
fig1421 (ZERO)
fig1422 (ZERO)
$1I49 (IBUF)
fig1426 (FGEN1R)
fig1425 (FGEN1R)

```
Synthesis Framework: info - 65 cells preserved by user
Map: info - Finished Map at May 22, 1997 at 11:58:46 am - Success
```

Many-to-One Mapping Association

Mapping Information in the Log File

This portion of the tutorial only refers to examples for AT40K devices

For more detailed information on what has happened to your design, consult the log file:

1. Pull down *Window>New Viewer>Log File* to open the log viewer. Figaro scrolls to the end of the file.
2. Use *Edit>Find* to search for the term "Mapping Report" (remember, the search utility is case-sensitive).

   Figaro scrolls to the first mapping report (see extract below), listing the different types of cells used and summarizing the number of instances used and the area required on the device:

```
Cells Used

#Insts  #Ins  #Outs  #Cost  #Cores  Cell Name (Description)
10      2     1      1      1       FGEN1 (Macro )
4       1     1      1      1       INV (Combinational Inverter Macro )
2       0     1      1      1       ZERO (Combinational Ground Macro )
1       1     1      1      1       RSBUF (Input Pad Macro )
1       3     1      1      1       AN3I1 (Combinational Macro )
2       2     1      1      1       AN2I1 (Combinational Macro )
4       2     1      1      1       AN2 (Combinational Macro )
4       17    4      1      1       RAMDSYNC (Sequential Macro )
1       3     1      1      1       AN3 (Combinational Macro )
1       1     1      1      1       GCLKBUF (Input Pad Macro )
8       2     1      1      1       XN2 (Combinational Macro )
3       3     1      1      1       FGEN1RF (Sequential Macro )
                                    ## PRESERVED ##
6       4     2      1      1       FGEN2RF (Sequential Macro )
                                    ## PRESERVED ##
41      1     1      1      1       OBUF (Output Pad Macro )
15      3     2      1      1       FGEN2 (Macro ) ## PRESERVED ##
49      4     2      1      1       MGEN (Macro ) ## PRESERVED ##
18      1     1      1      1       IBUF (Input Pad Macro )

Total #Instances: 170
Total Area: 170
```

   Note also that several cell types are marked as being ##PRESERVED## (like FGNEN2 and MGEN above). This is the default, taken from the Atmel library definition for the cell type.

3. In the Log Viewer, scroll down to the next "Mapping Report", giving you the post-mapping figures:

```
Cells Used
#Insts  #Ins  #Outs  #Cost  #Cores  Cell Name (Description)
41      1     1      1      1       OBUF ( Output Pad Macro )
18      1     1      1      1       IBUF ( Input Pad Macro )
4       17    4      1      1       RAMDSYNC ( Sequential Macro )
18      4     1      1      1       FGEN1 ( Macro )
6       4     2      1      1       FGEN2RF ( Sequential Macro )
                                    ## PRESERVED ##
3       3     1      1      1       FGEN1RF ( Sequential Macro )
                                    ## PRESERVED ##
2       0     1      1      1       ZERO ( Combinational Ground Macro )
15      2     2      1      1       FGEN2 ( Macro ) ## PRESERVED ##
49      4     2      1      1       MGEN ( Macro ) ## PRESERVED ##
1       1     1      1      1       RSBUF ( Input Pad Macro )
1       1     1      1      1       GCLKBUF ( Input Pad Macro )

Total #Instances: 158
Total Area: 158
```

   Many of the cell types used in the implementation are now different, but those preserved by default have remained unchanged. The total number of instances in the design and the area required to accommodate them have both been reduced.

   All the AN2, INV, and XN2 etc. have been mapped to FGEN1. There is only a small reduction in area in this case because many of the components were generated using the *Macro Generators*.

Overriding the Defaults

This portion of the tutorial only refers to examples for AT40K devices

You can remove mapping constraints set automatically via the cell definition in the Atmel library using the *Edit>Constraints* command. This is known as "unpreserving" the cells in question:

1. Leave the Log Viewer open and make the Design Browser the current window.
2. Expand the hierarchical instance $1I134(MULT8) and select the 7 “Product” terms at the top..
3. Pull down *Edit>Constraints*, clear the "Mapping Preserve" check-box and click *OK* to confirm the change.
4. Now click the *Map* button and confirm that you want to re-run mapping.
5. When Figaro is finished, go back to the Design Browser and select one of the “Product” instances from the $1I134(MULT8) hierarchy.

   Figaro highlights two instances in the Map Browser.
6. Try selecting any of the other product instances. Overriding the Atmel preservation defaults has effectively doubled the number of instances in this section of your design. This is because the AT40K mapper cannot map to multi-output macros, only to single-output macros.
7. Go back to the log file and pull down *File>Update* to load the latest version.

   Scroll back up to the latest mapping report and note the increase in the total number of instances and the area required to accommodate them.

Of course, the increase seen over the whole design will not necessarily be double, and you may even find an overall decrease if the registers/logic can be merged with any of the other components.

Summary

In this tutorial you saw:

- The purpose of the map step and when you might consider using it.
- How to set mapping options.
- How to map the design.
- How to use the Map Browser.
- How associations are maintained between the original and mapped designs.
- The default preservation constraints defined for certain cell types in the Atmel library.
- How to place and remove constraints on selected design instances and effects this can have on your implementation.

Mapping AT6K Designs to AT40K

If you want to implement older designs on parts belonging to the AT40K family, you can do so using the "AT6K mapped to AT40K" configuration.

This retargets your original design so that its components are compatible with the AT40K architecture, and at the same time performs some local optimization of the functions defined for any FGEN cells used in your design. For example, if the design contains the equation:

```
A*B*!C + A*!B + A*B*C
```

then the mapper will optimize away the redundant logic driving the B and C inputs and reduce the equation to the more optimal A.

NOTE You cannot map an AT40K design to the AT6K architecture.

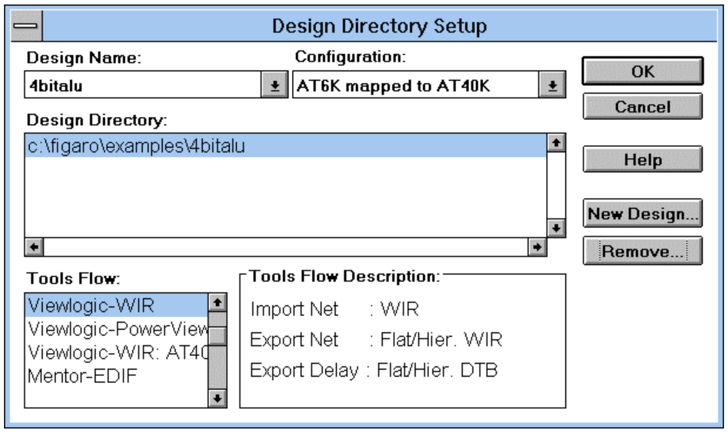
This chapter shows you how to set up your design session for this flow and how you can find information on your retargeted design. It also highlights some of the potential pitfalls of using older designs with the new architecture.

Setting up the AT6K to AT40K Mapped Flow

1. Pull down *File>Design Setup* to open the Design Setup dialog.
2. Set up the 4BITALU design.

NOTE If you've already completed the macro section in Chapter 5, you should use a new version of the design for this tutori

3. Under "Configuration", select "AT6K mapped to AT40K".



Design Name: 4bitalu

Configuration: AT6K mapped to AT40K

Design Directory: c:\figaro\examples\4bitalu

Tools Flow: Viewlogic-WIR, Viewlogic-PowerView, Viewlogic-WIR: AT4(, Mentor-EDIF

Tools Flow Description:
Import Net : WIR
Export Net : Flat/Hier. WIR
Export Delay : Flat/Hier. DTB

Design Directory Setup Dialog Box

The map step is compulsory in this configuration, so when you click OK and return to the desktop, you will see that the Flowbar button labeled *Map* has been enabled.

Before Going Further

Before loading your design, you should consider the following:

Part and package attributes

If you attempt to add an AT6K part to the AT40K flow, Figaro will report an error. So if your Viewlogic design uses the part and package attributes in the symbol file to add a part automatically, this should be disabled before you load the design.

To do this, open the *Options* dialog and select the topic "Viewlogic Import". Clear the checkbox labeled "Use part/package attributes" and click *OK* to save the new settings.

Repeat constraints files

Similarly, if you auto-import an old repeat constraints file specifying an AT6K part, Figaro will reject the design on import. Again, open the *Options* dialog, select the topic "Design Constraints", and make sure the check-box labeled "Auto-import Repeat Constraints" is clear.

Pin assignments

If you try to transfer existing pin assignments made for an AT6K to an AT40K part, you are likely to encounter problems as the pin layouts of the two families are not entirely compatible.

Figaro will report any inconsistencies it finds and you can resolve these by manually editing the constraints file. However, for improved design layout it is recommended that you either allow Figaro to re-assign pins for the new architecture automatically, or make your own AT40K-specific pin assignments using the procedure outlined in Chapter 8.

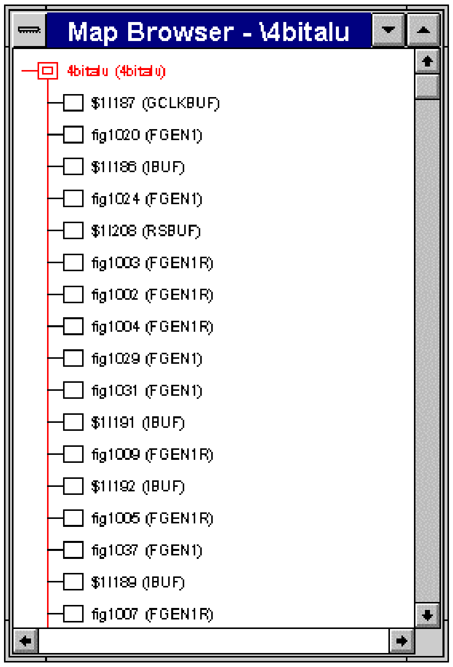
Running the AT6K to AT40K Mapped Flow

The AT6K to AT40K mapped flow is run in the same way as any other:

1. Pull down *File>Open as Design* and set "Files of Type" to Viewlogic Wir (*.1).

   Figaro enters the file 4bitalu.1 in the "Existing Design File" field.
2. Click *OK* to open the design. Figaro opens a Design Browser.
3. Click the *Map* button to map the design to the AT40K architecture.

   Figaro opens a Map Browser displaying the components of the mapped design:



```
4bitalu (4bitalu)
    $1I187 (GCLKBUF)
    fig1020 (FGEN1)
    $1I186 (IBUF)
    fig1024 (FGEN1)
    $1I208 (RSBUF)
    fig1003 (FGEN1R)
    fig1002 (FGEN1R)
    fig1004 (FGEN1R)
    fig1029 (FGEN1)
    fig1031 (FGEN1)
    $1I191 (IBUF)
    fig1009 (FGEN1R)
    $1I192 (IBUF)
    fig1005 (FGEN1R)
    fig1037 (FGEN1)
    $1I189 (IBUF)
    fig1007 (FGEN1R)
```

Map Browser

4. Select one or two instances in the Design Browser and use the cross-highlighting feature to see how these are represented in the mapped version of the design.

Restrictions

The following restrictions apply when mapping AT6K designs to the AT40K architecture.

Multi-output cells
The mapper is designed to map to single-output cells only. Any multi-output AT6K cells (for example, FDHA) will be mapped to two separate single-output AT40K macros

AT6K pads with no AT40K equivalent
There are a number of AT6K pads that cannot be mapped to an AT40K equivalent:

BCOCEN	BTPFEN
BCOCFEN	OOCEN
BCPEN	OOCFEN
BCPFEN	OPEN
BTOCEN	OPFEN
BTOCFEN	

To map the design successfully, you will have to remove instances of these pads from your design.

Optimization
As mentioned previously, the mapper performs only local optimization of the functions defined for the different FGEN cells. It does not perform any cross-cell optimization for your design.

User macros and generated macros
AT6K user macros and generated macros cannot be mapped directly to multi-cell equivalents. However, in the case of *generated macros*, you can use the *Library>Translate Library* command to search through an existing AT6K user library, find all generated components, and invoke the AT40K generator with similar options.

For *user macros*, each macro must be checked out of its library and recompiled using the "AT6K mapped to AT40K" flow, before being checked back into a new library containing only AT40K components. To re-import the design, remove the old library from the search path and add the new library containing the AT40K components. The design will now map successfully using the "AT6K mapped to AT40K" flow.

Finding Information on the Mapped Design

The Log File

The best source of information on the changes made to the design by mapping is the log file.

Pull down *Window>New Viewer>Log File* to open the Log Viewer for this design. If you scroll backwards through the file, you will find two mapping reports. The first of these summarizes the pre-mapping design, comprising only AT6K components:

```
Mapper: info - Mapping Report
Cells Used
  #Insts    #Ins     #Outs    #Cost    #Cores    Cell Name (Desc.)
  24        0        1        1        1         ZERO (Comb Gnd Mac)
  19        1        1        1        1         INV (Comb Inv Mac)
  1         1        1        1        1         CLKBUF (Inp Pad Mac)
  1         1        1        1        1         RSTBUF (Inp Pad Mac)
  27        3        1        4        4         OR3 (Comb Macro)
  6         2        1        3        3         OR2 (Comb Macro)
  3         2        1        1        1         AN2L (Comb Macro)
  105       2        1        1        1         AN2 (Comb Macro)
  8         3        1        1        1         AN3 (Comb Macro)
  19        3        1        1        1         FD (Seq Macro)
  52        2        1        1        1         XO2 (Comb Macro)
  2         0        1        1        1         ONE (Comb Pwr Mac)
  8         1        1        1        1         OD (Output Pad Mac)
  9         1        1        1        1         ITTL (Inp Pad Mac)

Total #Instances: 284
Total Area: 377
```

The second report shows the mapped version, comprising only AT40K components:

```
Mapper: info - Mapping Report
Cells Used
 #Insts    #Ins     #Outs    #Cost    #Cores    Cell Name (Desc.)
 8         1        1        1        1         OBUF (Outp.Pad Mac)
 9         1        1        1        1         IBUF (Inp Pad Mac)
 35        4        1        1        1         FGEN1 (Comb Macro)
 19        4        2        1        1         FGEN1R (Seq Macro)
 1         1        1        1        1         RSBUF (Inp Pad Mac)
 1         1        1        1        1         GCLKBUF (InpPad Mac)

Total #Instances: 73
Total Area: 73
```

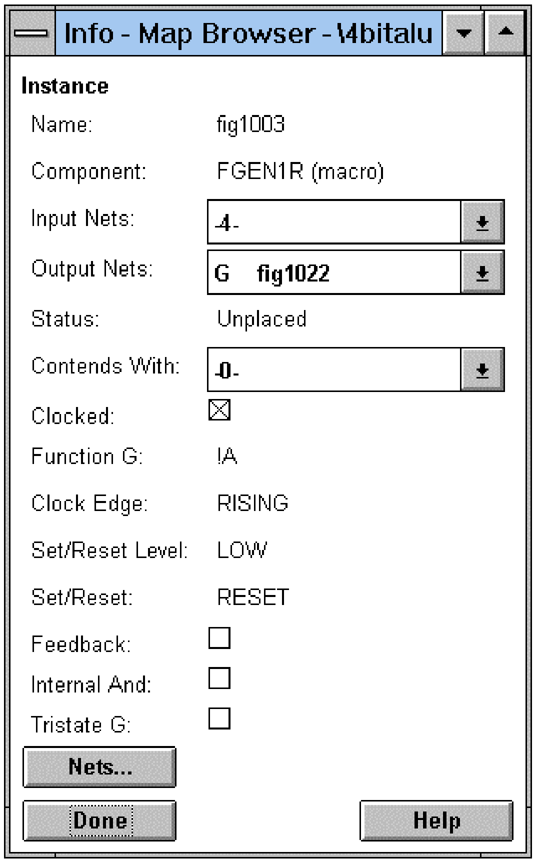
Note the significant reduction both in the number of instances required for the design (284 to 73) and the area required to accommodate it on the device (377 to 73).

The Info Window

You will find more detailed information on individual components using the Info window.

For example, you might look at the Map Browser and note the number of FGEN1 and FGEN1R components: these are basic components of the AT40K architecture and can be used to implement any 4-input function. To find out more about the actual function being performed:

1. Select instance fig1003 in the Map Browser.
2. Pull down *Edit>Info* to open an Info window on the selected object:


Info - Map Browser - \4bitalu

Instance

Name:	fig1003
Component:	FGEN1R (macro)
Input Nets:	-4-
Output Nets:	G fig1022
Status:	Unplaced
Contends With:	-0-
Clocked:	☒
Function G:	!A
Clock Edge:	RISING
Set/Reset Level:	LOW
Set/Reset:	RESET
Feedback:	☐
Internal And:	☐
Tristate G:	☐

Info Window on fig1003(FGEN1R)

The field labeled "Function G" shows the Boolean equation for the combinational part of the output.

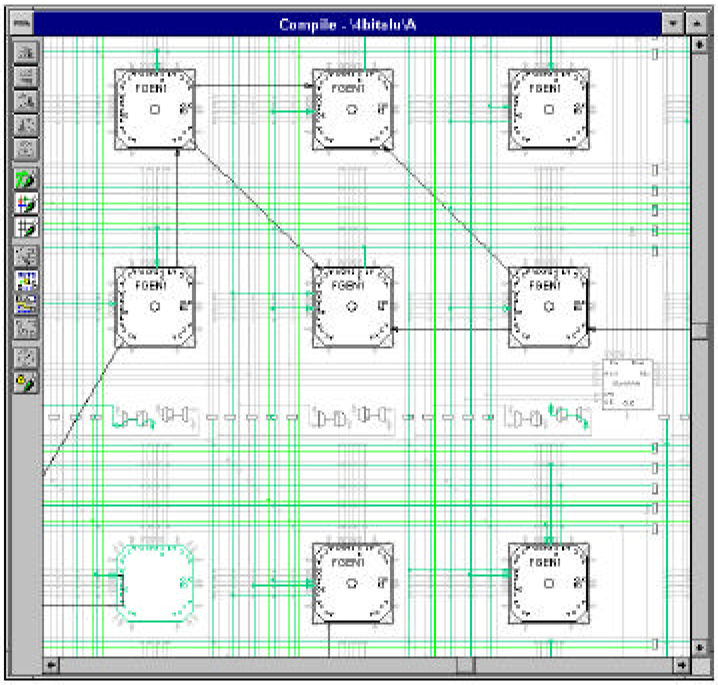
If the equation overflows the right hand edge of the window, either resize the Info window or position the cursor somewhere on the equation and use the direction keys to scroll left and right.

Compiling the Mapped Design

To compile the mapped AT6K design, proceed as follows:

1. Use the Part Select dialog to add a single AT40K05-2QC device and then partition the logic into the part.

   You will see some transcript warnings informing you that some incompatible pin locks have been removed.
2. Click the *Compile* button to compile the design.
3. When Figaro is finished, pull down *Window>New Compile Window* to open a Compile window.
4. Use *View>Zoom to Area* to magnify an area of nine core cells:



Compile Window Showing X and Y Direct Connects

The red and blue lines indicate diagonal and orthogonal direct connects. The addition of diagonal direct connects in the AT40K architecture means that a single core cell can now be connected directly to any of the eight cells surrounding it.

Timing

When performing timing analysis or compiling a retargeted design in timing-driven mode, timing assertions and constraints referring to external signals (for example external clock constraints or IO assertions) can be re-used in the AT40K flow.

However, since mapping will have changed the names of many of the internal components, any derived clock or asynchronous delay assertions you try to apply to the design will result in errors being reported to the transcript.

If you import a TMG file that contains both legal and illegal constraints, Figaro will apply the legal ones correctly and issue warnings drawing your attention to the others. You should check and modify any assertions rendered invalid by mapping.

Partitioning

Partitioning is the process of allocating your design logic to parts on the board. You can partition in two ways:

- Manually

  Drag and drop design instances into individual parts from the Design Browser. The emphasis here is on user control: you can specify certain logic functions to be performed by a particular part or keep particular components of your design together.
- Automatically

  Figaro allocates design logic for you. Use this method if there are no constraints on which part instances are placed in.

You can, of course, combine the two methods. Automatic partitioning respects any manual partitioning you have performed.

This tutorial comprises the following sections:

1. Adding Multiple Parts
2. Assigning Pin Locks
3. Partitioning Design Logic
4. Exporting Partitioning and Pinout Information
5. Compiling a Design with Multiple Devices

Although you will use the same design as before, you need not have completed the previous tutorials before starting this one. However, functions already used in previous tutorials are not described in detail again here, except where considered particularly relevant. Please refer back to the appropriate sections for detailed information.

Before You Start

This section describes preliminary steps to check before embarking on the tutorial itself.

Checking the Design Setup

If Figaro is not already running, open it as described in "Starting up Figaro", then do the following to set up the desktop for this tutorial:

1. Pull down *File>Design Setup* to display the Design Directory Setup dialog box. The Design Directory list contains all the directories you have worked with so far.
2. Make sure that the "Tools Flow" is set to "Viewlogic Workview Office and select \SYSTEMDESIGNER\EXAMPLES\VLOGIC\4BITALU (AT6K) or \SYSTEMDESIGNER\EXAMPLES\AT40K\VLOGIC\TEST40K (AT40K) from the design directory list.
3. Click *OK* to accept these entries.

Disregarding an Existing Repeat Constraints (.RCT) File

When you saved your design at the end of the previous tutorial, Figaro saved a record of the part and any pin locks you made in a file named *design*.RCT. This is called the repeat constraints file.

When you open a design for which a repeat constraints file exists, and the "Auto-import Repeat Constraints" option is enabled, Figaro automatically adds the part and makes any pin locks specified in that file. Using this file can often save time, especially when you have locked pins.

However, in this tutorial you want to run the design as if it were new, so you have to make sure that the existing repeat constraints file is disregarded during import:

1. Open the *Options>Options* dialog and select "Design Constraints".
2. Make sure "Auto-import Repeat Constraints" is **not** checked.
3. Click *OK* to accept these options.

   Figaro will now ignore any existing repeat constraint files when reading in design netlists.

You will find more information on the repeat constraint file in Appendix B "Figaro Files".

Opening the Design

To read in the design netlist, proceed as follows

1. Click on the *Open* button.

   Figaro asks whether you want to open the design as a design or as a macro.
2. Click *Design* to display the Open as Design dialog box.
3. Select \SYSTEMDESIGNER\EXAMPLES\VLOGIC\4BITALU (AT6K) or \SYSTEMDESIGNER\EXAMPLES\AT40K\VLOGIC\TEST40K (AT40K) from the pull-down design directory list. (If you can't see it, set it up again using *New Design*.)
4. Make sure the "Files of Type" is set to "Viewlogic Wir (*.1)" and then select 4BITALU (AT6K) or TEST40K (AT40K) in the "Design Name" field.

   Figaro displays 4BITALU.1 or TEST40K.1 in the "Existing Design File" field.
5. Click *OK* to open the design.

   The Design Checker writes information and warning messages to the transcript depending on the options set. If there is an existing repeat constraints file, Figaro issues a message to the transcript informing you that it was ignored.

Mapping is enabled, and you are now ready to start partitioning.

Adding Multiple Parts

As you saw in Chapters 3 and 4, the Parts window is a representation of the PCB that holds the FPGA and is designed to help you visualize the orientation of the part.

You use the Part Select dialog to specify the parts you want for your design and these are then displayed along with the design's I/O signals in the Parts window. The part is represented by a line drawing of the device while I/O connections to the outside world are represented by stylized drawings of pins, with input functions to the left and outputs to the right. If you add multiple parts, the window shows all of these and the connections between them.

To add a part, proceed as follows:

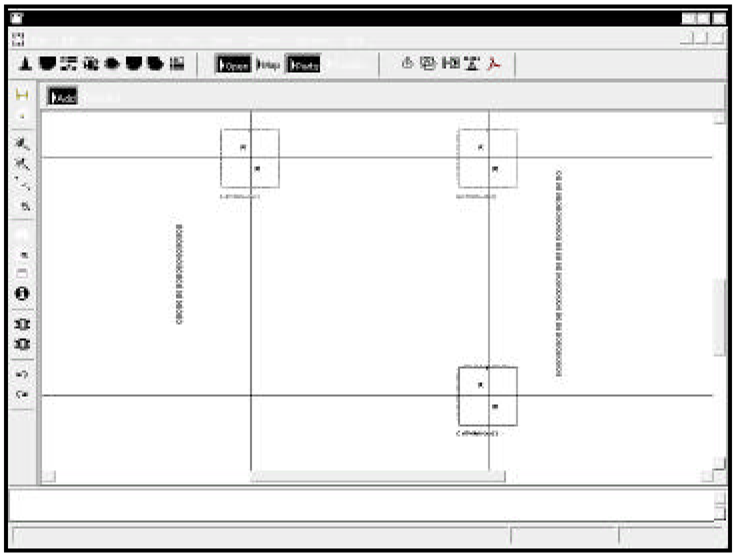
1. Click on the *Parts* button. Figaro opens an empty Parts window and the Part Select dialog box.
2. Select "AT6003-2JC" (AT6K) or "AT40K05-2AJC" (AT40K) in the "Part Name" list and click *Add*.

   Figaro adds the selected part to the Parts window and displays a green bar below each of the gray bars at the base of the Part Select dialog.

   The green bars represent the *capacity* of the selected part (logic capacity above and memory capacity below). The gray bars illustrate the (logic and memory) capacity *required* for the design. The relative lengths of the bars indicate whether the part is large enough to accommodate the design. In this example, the design will fit on to a single part, but we'll add more logic to demonstrate partitioning.
3. In the Part Select dialog, select "AT6003-2JC" (AT6K) or "AT40K05-2AJC" (AT40K) again.

   You see that the capacity of the parts (green bars) now exceeds design size (gray bars), even though Figaro has not yet added the part to the Parts window. This allows you to check whether the capacity of the selected part will meet your requirements.
4. Now click the *Add* button twice to add two further parts to the Parts window.
5. Click *OK* to close the Part Select dialog box.

   The Parts window now contains three parts as shown:



The Parts Window with Three Parts Added

Changing the P&R Utilization for a Part

You can also the use *P&R utilization* to change the percentage of a part's resources made available for the placement and routing of logic and memory. For example, you might lower the utilization to prevent your devices becoming too congested and therefore more difficult to place and route, or you might increase the utilization of a particular part to allow it to accommodate a design that is otherwise just too large.

You change the P&R utilization in the Options dialog under "Partitioner". Use the slider to set the figure as required and click *OK* to accept settings and return to the Parts window.

Change the value for a single part by selecting the part in the Parts window and using *Edit>Constraints*.

Assigning Pin Locks

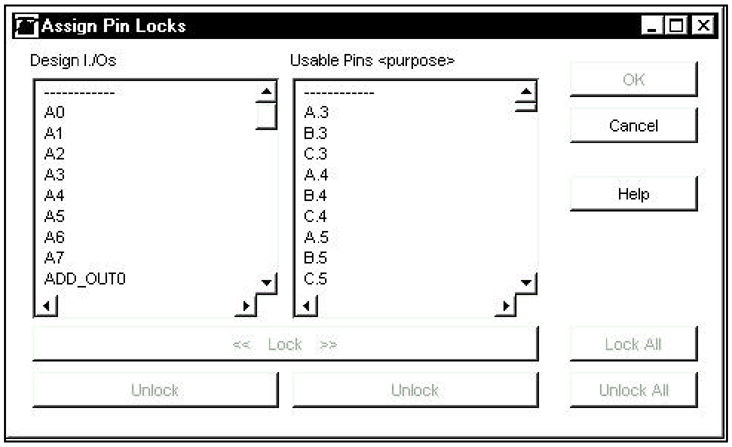
In this tutorial, you are not producing the FPGA in isolation, but need to take account of its position relative to the other FPGAs on the board. This means you need to specify what pins on the device are to perform what I/O functions. This is called *assigning pin locks*.

Locking pins restricts Figaro during placement and routing, so for your own designs you should only do this as the PCB layout dictates, For example if the board design specifies that certain I/Os (for example, a clock input) must be associated with particular package pins, or that a number of input or output functions must be handled by adjacent pins.

This section describes how to assign pin locks for the three parts you selected for the design. You'll use the Parts window to assign different output functions to specified pins on the devices.

Assign Pin Locks Dialog

Make sure the Parts window is the current window and pull down *Edit>Assign Pin Locks* to display the following dialog:



Assign Pin Locks Dialog

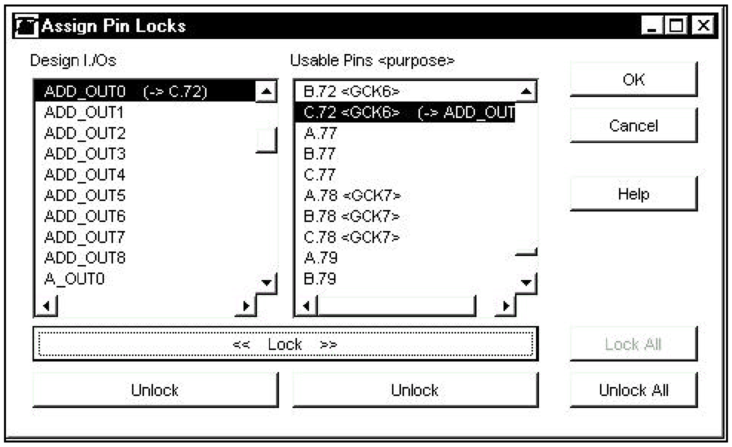
Hint: You can enlarge this dialog vertically to display the full list of Design I/Os. This makes the subsequent tasks much easier to perform.

The “Design I/Os” box lists the design's I/O functions, while the “Usable Pins” list shows the pins available for these I/Os: this list excludes pins reserved for other functions (e.g. VDD).

The pin names are prefixed by the letters "A", "B", and "C" indicating which part they belong to. Some pins have text in brackets after their name, showing the pin's dedicated function.

Use this dialog to lock the output I/O "O0" to pin C.56 (AT6K) or "ADD_OUTPUT0" to pin C.72 (AT40K). Do the following:

1. Select this entry. Select "O0" 56 (AT6K) or "ADD_OUTPUT0" (AT40K), from the I/O list.
2. Select the entry "C.56" (AT6K) or "C.72" (AT40K) in the "Usable Pins" list.
3. Click *Lock*. Package pin "C.56" will now carry the "O0" (AT6K). 72 on part C now carries the "ADD_OUTPUT0" function (for AT40K). The name of the associated pin or I/O appears in parentheses after each of the items, like this:



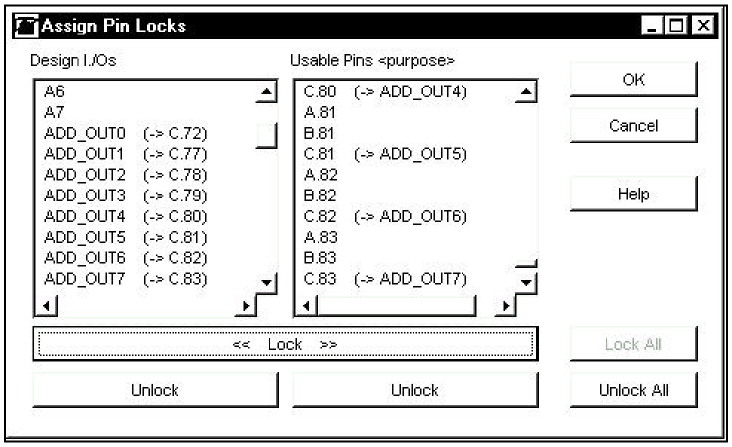
Output ADD_OUTPUT0 and Pin C.72 Locked

The Parts window provides visual feedback on the actions performed in the Assign Pin Locks dialog: as you selected the "O0" board I/O and pin 56 of part C (AT6K) or "ADD_OUTPUT0" board I/O and pin 72 of part C (AT40K), these were highlighted in the Parts window. When you clicked *Lock*, the two were connected by a line indicating the pin lock constraint.

4. Repeat the procedure until the following I/Os have been assigned:

Lock design I/O...	To usable pin...	Device
O1	C.57	AT6K
O2	C.58	AT6K
O3	C.59	AT6K
O4	C.62	AT6K
O5	C.63	AT6K
O6	C.64	AT6K
O7	C.65	AT6K
ADD_OUT1	C.77	AT4K
ADD_OUT2	C.78	AT4K
ADD_OUT3	C.79	AT4K
ADD_OUT4	C.80	AT4K
ADD_OUT5	C.81	AT4K
ADD_OUT6	C.82	AT4K
ADD_OUT7	C.83	AT4K
ADD_OUT8	C.84	AT4K

The dialog list will then look like this:



```
A6                          C.80   (-> ADD_OUT4)
A7                          A.81
ADD_OUT0   (-> C.72)        B.81
ADD_OUT1   (-> C.77)        C.81   (-> ADD_OUT5)
ADD_OUT2   (-> C.78)        A.82
ADD_OUT3   (-> C.79)        B.82
ADD_OUT4   (-> C.80)        C.82   (-> ADD_OUT6)
ADD_OUT5   (-> C.81)        A.83
ADD_OUT6   (-> C.82)        B.83
ADD_OUT7   (-> C.83)        C.83   (-> ADD_OUT7)
```

Assigned Pin Locks

Click *OK* to accept the pin locks you have just assigned and return to the Parts window.

In the Parts window, you'll see lines connecting the IOs and pins: these reflect the pin lock **constraints** you've placed on the design and were updated each time you clicked *Lock*. The package pins themselves remain unchanged until after partitioning (see next section), when they will also change color to indicate that they are locked. The colors used depend on the settings in *Options>Display Options.*

Undoing Pin Locks

The dialog also incorporates two *Unlock* buttons, allowing you to unlock individual pins without first selecting the items in both lists. Just select one and click the *Unlock* button under that column.

If you are unhappy with the pin locks you have made, you can undo all the changes since opening the dialog by pressing *Cancel*. To save the pin locks, press *OK*.

Drag and Drop Pin Lock Assignment

The above method is useful if your board layout is strictly predefined. However, you can also use Figaro's drag and drop facility to reposition parts and IOs and assign pin locks directly in the Parts window.

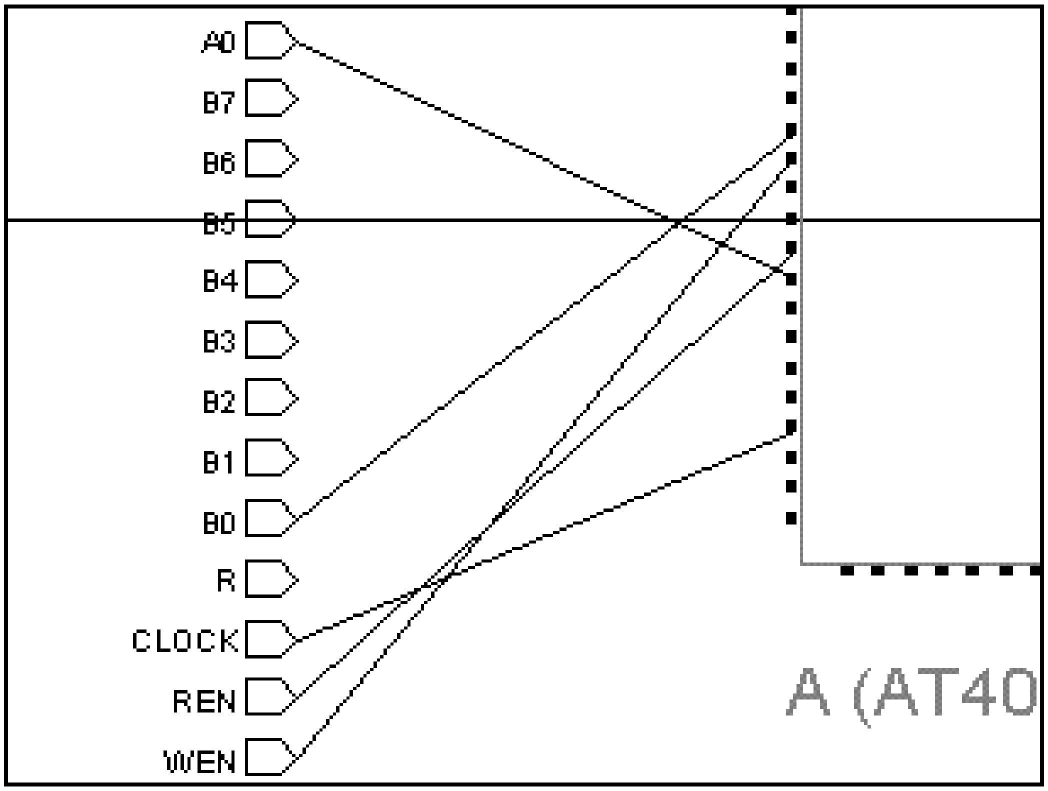
To move parts:

1. Select part C and click-hold the left mouse button to drag it so that the parts are arranged in a horizontal row in the sequence A, B and then C.
2. Release the mouse button to drop the part at your desired location.

   Figaro uses the cursor position as a reference when positioning the part. If the cursor happens to be positioned over some other resource (i.e. another part or IO) when you release the mouse button, Figaro will return the part to its original location and report the error in the transcript.
3. The board IOs are now badly positioned. Pull down *Edit>Arrange IOs* to redraw them.

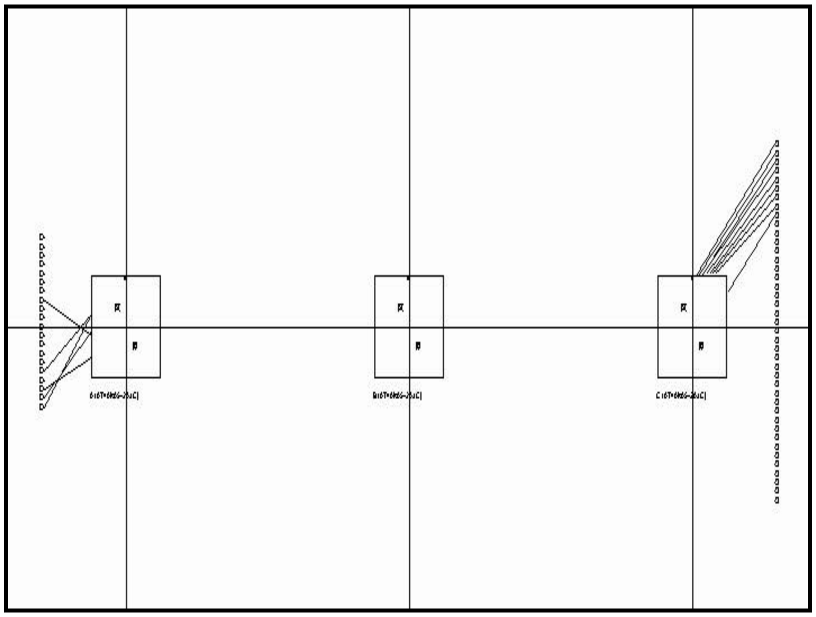
To assign pin locks:

1. Use *View>Zoom to Area* to zoom in on the left side of device A and the input board Ios.
2. Find pin A.30 on part B (AT6K) 24 on part A (AT40K) (when you move the cursor over a pin, the status bar displays its name and type).
3. Select (AT6K) and drag it over to pin A.30. Select IO "A0" (AT40K) and drag it over pin 24. The line joining the two instances indicates the constraint.
4. Use this method to lock IB[2:0] to pins A.29 through A.27 and IA[0:3] to pins A.24 through A.21 (AT6K) or "B0" to 19, "WEN" to 20, "REN" to 23 and "CLOCK" to pin 29 (or GCK2, shown in green) (AT40K).



Pin Lock Constraints on Part A

5. Pull down *View>Zoom Fit* to view the final board layout:



PCB Layout with Assigned Pin Locks

Partitioning Logic

In this section, you will use manual and automatic partitioning to allocate design logic to the three parts on to the board. You will get to know the Design Info dialog and the Sub-Design Browser, and learn how to interpret the information provided by Figaro.

Design Info Dialog Box

Make the Parts window active, select *Edit>Design Info*, and click the *More>>* button. This opens the extended Design Info dialog, providing information while you are carrying out manual partitioning.

Using Manual Partitioning

1. Select part B (AT6K) or part A (AT40K) in the Parts window.
2. Pull down *Window>New Sub-Design Browser*. This lists all the logic instances currently allocated to the selected part. At the moment it is empty, so only the part name is shown.
3. Open a sub-design browser for part B as well, and then pull down *Window>Tile* so that you can view all your tool windows.
4. Make the main Design Browser the current window and select instance $1I72(CONTROL) (AT6K) or ADDER8 (AT40K). Click-hold the left mouse button, drag the instance over part B (AT6K) A (AT40K) in the Parts window then release.

   The Sub-Design Browser for the selected part is updated to include these instances as is the information displayed in the Design Info dialog. Figaro adds extra pads as required to carry intermediate signals.
5. You can also drag and drop directly between browsers. Using either method, perform the following partitions:

Partition instance...	To part...	Device
REGISTR4 (both)	A	AT6K
MULT8	B	AT40K

If, at any stage, you are unhappy with the way your efforts are developing, just pull down *Edit>Unpartition Design* and Figaro will remove all partitioning information and you can start again.

You cannot partition IO buffers as these will be automatically partitioned to go with the appropriate logic.

Modifying the Partition

1. Go to the Parts window and look at the green arrows showing the interconnect between parts A and C.

   The thickness of the arrows reflects the number of signals being carried, with the exact number displayed next to it.
2. Use drag and drop from either browser to move the "CONTROL" (AT6K) "ADDER8" (AT40K) component from part B to part A (AT6K) or from part A to part B (AT40K).

   Note that the interconnect line between A and C disappears.
3. The Sub-Design Browser for the selected part displays only the part name, indicating that the part is now empty.
4. Use either *Edit>Remove Unused Parts* or select it and use *Edit>Remove Part* to remove it from the Parts window.
5. Use the auto-partitioning function to complete the rest of the partitioning of your design.

NOTE Some manual moves might be restricted, for example, if you have already locked an I/O that connects directly to the instance you are moving. To unlock pin assignments like these, use the Assign Pin Locks dialog described earlier.

Using Automatic Partitioning

Once you have completed the required manual partitioning, click *Partition* (or pull down *Edit>Auto-Partition Design into Parts* in the Parts window) and Figaro will complete the process.

You can exert control over the automatic partitioning function using the following parameters contained in the partitioner's *Options* dialog:

Quality:	Determines the relative emphasis given to resolving pin contention and minimizing chip-to-chip interconnect during partitioning. The higher the quality, the harder Figaro tries to minimize interconnect.
P&R utilization:	Specifies the percentage of the part's logic to be made available for placement of design logic.
Dual function pins:	Specifies whether any dual function pins on the part are to be made available for placement.
Pinout contention:	Controls whether Figaro takes account of pinout contention during partitioning
Part swapping:	Controls the management of swap files in multi-chip designs.

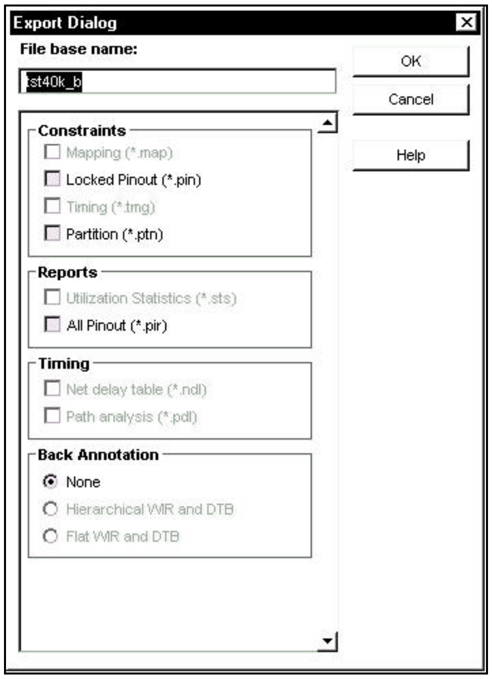
Manually partitioned logic is locked and remains unaffected during automatic partitioning. You can also use *Edit>Constraints* from the Design Browser and the Parts window to place constraints on particular levels of hierarchy to ensure these are kept together.

NOTE You can also use the auto-partitioner to achieve preliminary results if you do not already have a firm plan of how you want to partition your design. You can then modify the results as required using the manual techniques described.

Exporting Partitioning and Pinout Information

Once you have completed partitioning and pinout, you can export this information in several forms. This section shows you how to export the information from Figaro and describes the content of the different outputs:

1. Make sure the Parts window is the active window.
2. Check that none of the parts is selected (if there is, click *ESC* to deselect).
3. Pull down *File>Export* to display the following dialog box:


Export Dialog

File base name: st40k_b

Constraints
- ☐ Mapping (*.map)
- ☐ Locked Pinout (*.pin)
- ☐ Timing (*.tmg)
- ☐ Partition (*.ptn)

Reports
- ☐ Utilization Statistics (*.sts)
- ☐ All Pinout (*.pir)

Timing
- ☐ Net delay table (*.ndl)
- ☐ Path analysis (*.pdl)

Back Annotation
- ● None
- ○ Hierarchical WIR and DTB
- ○ Flat WIR and DTB

Exporting a Pinout Constraints File and Report

4. Leave the file base name as it is.
5. Check the boxes for the information you want to export:
   a) Constraints

      "Locked pinout (*.pin)" creates a constraints file listing the pinout locks for use in subsequent sessions (this is a subset of the .RCT file you looked at earlier).

      "Partition (*.ptn)" produces a constraints file listing the design instances and the part to which they have been allocated.
   b) Reports

      "All pinout (*.pir)" is a record of the full pinout, whether locked or not.
6. Click *OK* to export the selected files.

   Figaro automatically adds the appropriate extensions to the file base name and creates the files requested.

NOTE

Note that only *locked pinouts* will be used as constraints the next time you run the design. After partitioning the design, you can use the *Lock All* button in the Assign Pin Locks dialog box to ensure that *all* pin assignments are repeated.

Any manual partitioning information is also written to the RCT file and is used the next time you run the design.

For more information on the individual file types, refer to Appendix B "Figaro Files".

Compiling a Design with Multiple Devices

After partitioning, simply click *Compile* to run the automatic placement and routing functions. These are described in more detail in Chapters 3 and 4.

With a multi-chip implementation like this one, Figaro compiles one device at a time. When the first one is complete it is "swapped out" to release memory resources and obtain better performance. Figaro creates a .SWP file for each part swapped out. These files are written to a directory FGSWAP under the design directory. When it needs to, Figaro moves the part back into memory.

You can control part swapping using the Partitioner options in the *Options>Options* dialog box:

Figaro Controlled	Figaro swaps parts out of memory as required
On	Figaro always swaps parts out
Off	Figaro never swaps parts out

When compilation is complete and as long as swapping is **not** active (i.e. *Figaro Controlled* or *Off*), you can view both devices at the same time by opening a Compile window for each. If swapping is *On*, you will be able to view only one device at a time.

Summary

This tutorial showed you how to:

- Disregard an existing repeat constraints file for a design.
- Adjust the P&R utilization for selected parts.
- Add multiple parts using the Part Select dialog.
- Assign pin locks.
- Partition the design manually.
- Partition the design automatically.
- Export partitioning and pinout information.

You should now be able to add multiple parts and partition your design logic between them using the techniques described here.

If you have problems, refer to Chapter 10 “Troubleshooting and Support”, or from the on-line help Contents page jump to “Troubleshooting”.

NOTE Before moving on to any of the other tutorials contained in this guide, open the Partitioner options dialog again and ensure the P&R utilization is reset to 100%.

Additional Features

This chapter tells you about a few areas not covered by the tutorials, but which you might need:

- How to abort a task.
- How to save and restart a design session, or undo changes made during a session.
- How to incorporate design changes.
- How to change Figaro options.

Aborting a Task

If you mistakenly start an operation which takes some time to run, you can stop it by pressing CTRL+C keys simultaneously. This stops the current operation and displays one of two dialogs:

Stop dialog	This asks whether you really want to stop the current process. Click Yes or No.
User Interrupt dialog	Click Proceed if you want to continue the process, or Abort if you want to stop it.

In the Compile window this has different effects depending on the task being performed: some operations cannot safely be interrupted and will stop at the first safe point. Figaro writes a message to the transcript if this is the case.

For more details, search for the on-line help on “Aborting an Operation”.

Saving and Restarting a Session

You can save a Figaro design session to a file. The saved session includes a record of tasks you've performed during the session and the desktop state, for example, the number, type and size of open windows.

There are two ways of saving a design session:

- Keep a single file, updated regularly.
- Make a series of saves to different files, for example one after each design step. This means you can restore the design to its state at any intermediate stage.

Saving a Design Session

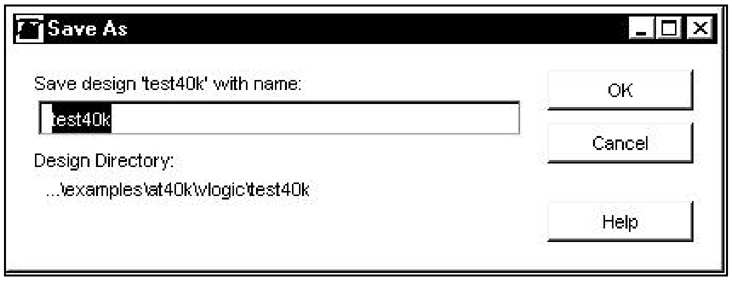
To save your design session, pull down *File>Save*. Figaro creates a file with the extension FGD (Figaro design), which you can re-open in the future. The file is stored in the design directory.

Each time you use *File>Save*, Figaro automatically creates a backup copy of the original save file to a file with extension .FG~. It then overwrites the existing FGD file with the latest state of the design.

Saving a Series of Snapshots of a Session

To make a series of saves within a single design session, use *File>Save As* and enter a different file name each time. The file is saved in the design directory.

Enter the file name you want in the dialog box shown below:



Save design 'test40k' with name:

test40k

Design Directory:

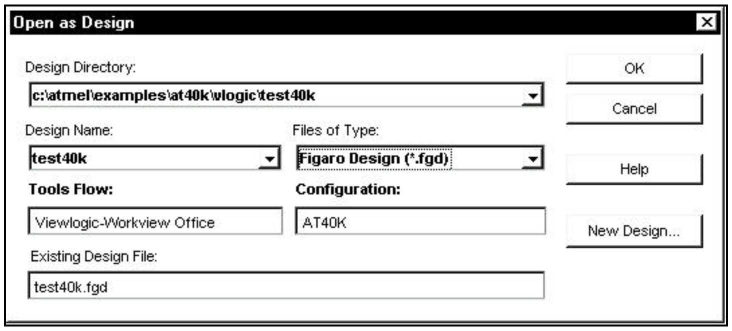
...\examples\at40k\vlogic\test40k

Saving a Design Session with a New Name

Using *File>Save A*s lets you restart the session at a number of different points. It's best to save to a new file after major steps like assigning pin locks or performing placement.

Restarting a Saved Session

To restart a saved design session, click the *Open* button to display the Open as Macro dialog or the Open as Design dialog (shown here):


Open as Design

Design Directory: c:\atmel\examples\at40k\vlogic\test40k

Design Name: test40k

Files of Type: Figaro Design (*.fgd)

Tools Flow: Viewlogic-Workview Office

Configuration: AT40K

Existing Design File: test40k.fgd

Opening a Saved File

Do the following:

1. Select the directory containing the saved design file from the “Design Directory” list (or use *New Design* to add it to the list).
2. Specify “Figaro Design (*.fgd)” in the “Files of Type” box.
3. Specify the name of the saved design session in "Design Name". Figaro enters the appropriate name in the field "Existing Design File".
4. Click *OK* to restore the design session at the required stage.

Undoing Changes made during a Session

To go back to the stage at which you last saved the design session, pull down *File>Revert to Saved.* You are asked “Do you want to discard your changes to design “yourdesign?”. Click *Yes.*

If you saved the session to different files at different times and want to go back to an earlier save than the last one, click the *Open* button. You are asked whether you want to save any changes to the current design, before Figaro displays the Open as Design dialog. Use the dialog to specify the name of the file you want. (The dialog is described in the previous section, “Restarting a Saved Session”.)

Incorporating Engineering Change Orders (ECO)

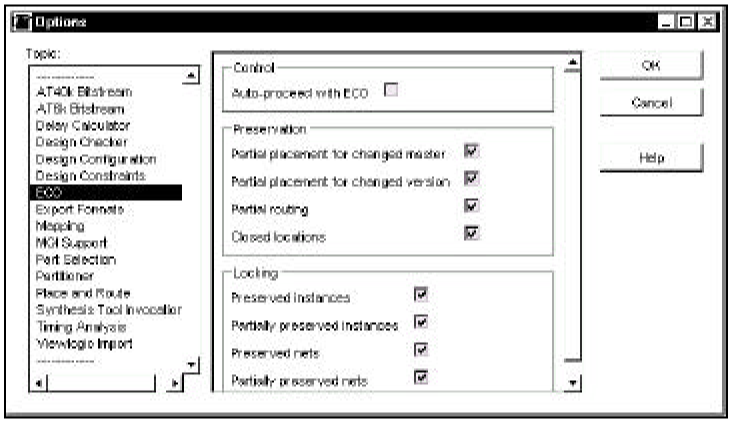
After you have compiled a design using Figaro, you may find you need to go back and make some modifications to the original netlist. Rather than having to rerun all the steps again for the changed design, Figaro supports a function for making incremental design changes called Engineering Change Order (ECO).

You can use ECO with single-chip, multi-chip and mapped designs of all netlist formats. Simply specify the extent to which unchanged instances and routing are to be preserved in the new version (see "ECO Options" below) and read in the modified netlist using the *Open as ECO* function.

Figaro compares the new design with the one it previously implemented, establishes what changes have been made and preserves and locks any unchanged elements as stipulated in the ECO options. You can then place and route any altered sections either manually or automatically. Pressing *Compile* will compile the remainder of the design for you.

ECO Options

Before running ECO, take a closer look at the ECO options: open the *Options>Options* dialog and select the entry "ECO" from the Topic list. Figaro displays the ECO options as illustrated below:



Control
Auto-proceed with ECO ☐

Preservation
Partial placement for changed master ☑
Partial placement for changed version ☑
Partial routing ☑
Closed locations ☑

Locking
Preserved instances ☑
Partially preserved instances ☑
Preserved nets ☑
Partially preserved nets ☑

Default ECO Options

The "Auto-proceed" option is switched off, meaning that after the modified netlist has been loaded, Figaro will prompt you to start ECO. This provides you with an opportunity to study the comparison summaries in the transcript before proceeding.

The "Preservation" options specify how much of the original implementation is to be preserved during ECO. Generally the more you preserve, the faster Figaro will be able to complete any subsequent compilation of the design.

The "Locking" options control whether preserved instances and nets are automatically locked in the new version of the design.

Make sure the options are set as in the picture above i.e. with the maximum level of preservation and all preserved objects locked.

ECO and Mapping

You can use ECO on mapped designs of all formats, however it is important to note the following:

- Running ECO on mapped designs may not always achieve the expected level of preservation because mapping is not always able to maintain associations between netlist design instances and Figaro's implementation of them. As a result, ECO cannot preserve the implementation of some instances, even if they are identical in the original and new versions. As the level of instance preservation degrades, so does the preservation of nets and routing.
- When running ECO on a mapped design, ensure the mapping options match those used when the design was originally compiled.

ECO and User Macros

You can also run ECO with user macros. Do the following:

Step
Compile the original user macro and check it into a library
Open and compile the design containing the user macro
Check the macro out of the library then make the necessary changes to the netlist
Open the new macro netlist as ECO
Compile the changed macro and check it back into the library

When you restore the original design containing the user macro, Figaro automatically uses the latest version of the macro and places it in the same orientation as the old one. It also issues a message warning you that it has used a changed version of the macro.

If the new version of the macro is much bigger than or a different shape from the old one, you should check the instantiation of the macro for contention.

Changing Figaro Options

You can change many of the parameters that control Figaro's operation. Many of these control window displays, some control the default entries in dialogs, and others control the way Figaro imports and exports files.

These options can be split into two groups, accessed using different commands in the *Options* menu. The first of these is the *Options* command, which accesses the following parameters:

Option Name	Controls
AT40K Bitstream	Bitstream format
AT6K Bitstream	Bitstream format
Delay Calculator	Delay range, input slew rate, output load
Design Checker	Error checks for opened design
Design Configuration	Rebuild hierarchy
Design Constraints	Auto-import repeat constraints
ECO	ECO flow, preservation, locking
Export Formats	Default selections in export dialog
HDL Planner	Parameterization of clock/reset schemes and design customization
Mapping	Mapping control and preservation options
MGI Support	Automatic macro generation as specified by the EDIF netlist
MGL Editor	MGL Editor setup
Part Selection	Sets defaults for the filters in the Part Select dialog
Partitioner	Partitioner quality and control
Place and Route	Compile quality and other options
Synthesis Tool Invocation	Tool flow and interface options for running synthesis
Timing Analysis	Graph type, paths listed, paths to ignore, report format
Viewlogic Import	Processing and use of attributes from ViewLogic netlists
Xilinx	Opening, processing and exporting of the XNF netlists

Parameters Set in the Options Dialog Box

The *Display Options* command allows you to view and change the display characteristics of the windows listed in the table below:

Option Name	Controls
Browser	Design and Map Browser windows.
Compile	Compile window(s) (see below).
Desktop	Message colors, toolbar display etc.
Parts	Parts windows (see below)
Selecting	Selection of nets or ports with an instance.

Parameters Set in the Display Options Dialog Box

The list is adapted dynamically depending on the windows open on your desktop. When you first open the dialog, the entry for the current window is always highlighted.

Window options control the colors and *display thresholds* of tool windows. The display threshold is the magnification at which a particular item is shown in the window.

Changing window options changes the settings for the specified window and not for all the windows of that type that are open. It only changes the default settings if you click the *Save As Default* button.

Changing Figaro Options: Examples

The following sections show how to change two sets of Figaro options:

- The design checks carried out when a design is loaded (under *Options)*
- The Compile window display (under *Display Options)*.

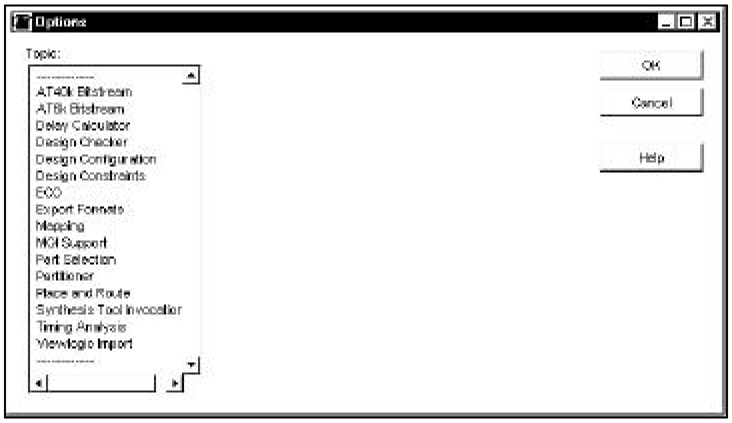
The procedures for changing other options are similar. Use on-line help for detailed instructions on a particular set of options.

Changing the Design Checker Options

This section shows how to use the Options dialog to change the checks that are run when you open a design. For example, you might want to instruct Figaro on what to do when it finds a net without a source in the design.

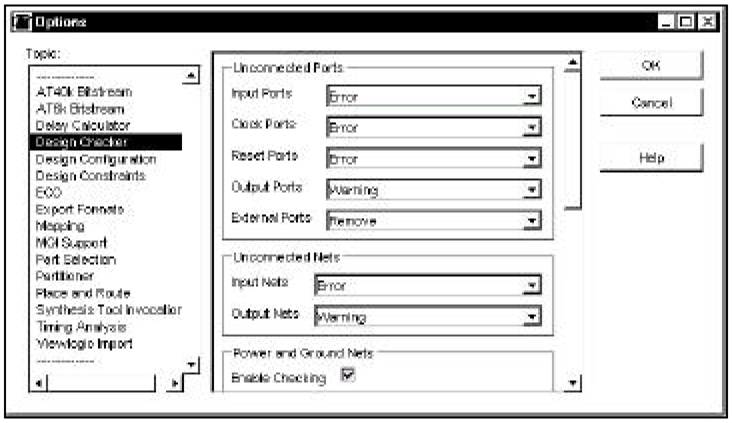
This example shows how to make Figaro issue a warning if it finds an unconnected input port in the design. Do the following:

1. Call up the Options dialog as displayed below:



Specifying the Options to Change

2. Select “Design Checker" from the "Topic" list. The current settings are shown, as follows (this is not the full list):



Unconnected Ports
Input Ports: Error
Clock Ports: Error
Reset Ports: Error
Output Ports: Warning
External Ports: Remove

Unconnected Nets
Input Nets: Error
Output Nets: Warning

Power and Ground Nets
Enable Checking ✓

Default Design Checker Options

3. Under “Unconnected Ports” click on the "Input Ports" pull-down list and change the setting from “Error" to "Warning”.

   The Design Checker will now report unconnected input ports as a warning, write the warning to the transcript and continue. Previously it would have reported an error and stopped opening the design.
4. Click *OK* to effect the change.

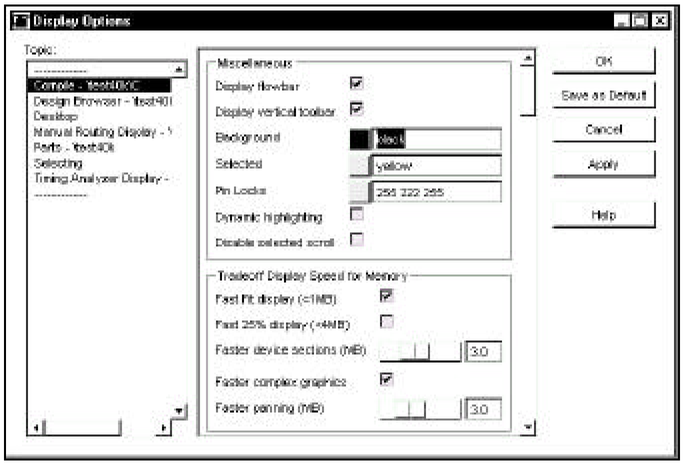
   This comes into effect immediately and will remain in effect in future sessions.

When you open the design, Figaro’s view of it will be modified by the Design Checker in the way these options specify. For more information, see the on-line help on "Design Checks".

Changing the Compile Window Display

This example shows how to change the color of nets using express buses in the current Compile window. (You might do this , for example, to distinguish these nets from others displayed in the same color.) The procedure is the same for all window display options. Do the following:

1. Pull down *Options>Display Options* and select "Compile" in the topic list to display this dialog box:



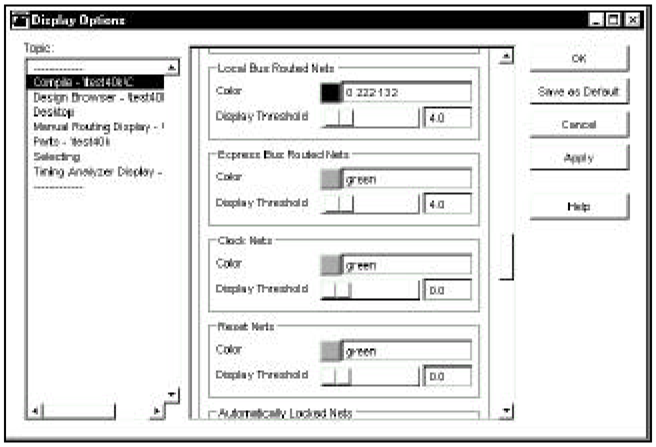
Miscellaneous
Display flowbar ✓
Display vertical toolbar ✓
Background: black
Selected: yellow
Pin Locks: 255 222 255
Dynamic highlighting ☐
Disable selected scroll ☐

Tradeoff Display Speed for Memory
Fast Fit display (<1MB) ✓
Fast 25% display (<4MB) ☐
Faster device sections (MB): 3.0
Faster complex graphics ✓
Faster panning (MB): 3.0

Compile Window Display Options

After the miscellaneous parameters, "Unused Locations" is the first instance parameter. Its entry shows a color entered as a sequence of three numbers in the range 0-255. These are the values for red, green and blue respectively. This is the alternative to typing in a color by name.

2. Scroll down the list until you reach “Express Bus Routed Nets”:



Topic:
- ------------
- Compile - \test40\C
- Design Browser - \test40\
- Desktop
- Manual Routing Display - \
- Parts - \test40\
- Selecting
- Timing Analyzer Display -
- ------------

Local Bus Routed Nets
- Color: 0 222 132
- Display Threshold: 4.0

Express Bus Routed Nets
- Color: green
- Display Threshold: 4.0

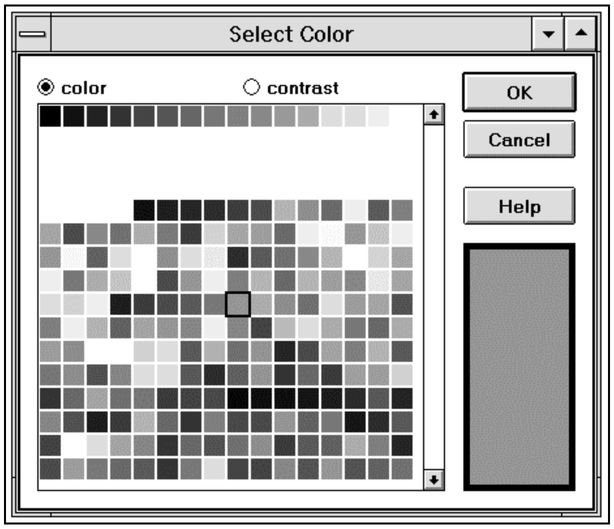
Clock Nets
- Color: green
- Display Threshold: 0.0

Reset Nets
- Color: green
- Display Threshold: 0.0

Automatically Locked Nets

Color and Display Threshold for Express Bus Routed Nets

3. Click on the colored button. This displays the palette of colors you can choose from (the “color” radio button will be selected here):



[✓ color] contrast

Color Palette

4. Use the cursor to select a color that is easily recognizable and is not being used by another type of net. A black outline appears around the button and the preview rectangle on the right of the dialog box changes to the selected color.
5. If you want to see what this color looks like in relation to another one, click the “contrast” radio button then choose the other color. This appears as a border to the main color in the preview box, so you can check the contrast and visibility of the combination.

   You might use this to test the contrast between two colors used for different routing resources, for example. The “contrast” color has no effect on the display once you leave this dialog.
6. Click *OK* (or ENTER) to return to the Display Options dialog.
7. Click *Apply* to view the new color in the Compile window before you accept it. Click on the Compile window: any express bus routed nets will change to the color selected.
8. Click *OK* to accept the change and leave the dialog. (To change settings for future Compile windows as well as this window, click *Save as Default* in the Display Options dialog - this new setting will be added to the FIGARO.INI file and used the next time you start up Figaro.)

Resetting Figaro Options

If you make changes to Figaro options and then decide you preferred the original settings, you can retrieve these by removing or editing the file FIGARO.INI stored in your Figaro BIN or startup directory.

Troubleshooting and Support

This chapter tells you:

- Answers to some frequently asked questions.
- What to do when you see an error message.
- What to check before you report a problem.
- How to report a problem.
- How to install an update (patch) to Figaro.

Using the Transcript and Log File

During a session messages are output to the transcript and log file. Look here first if you have problems: refer to the on-line help on "Error Messages". Use *Window>New Viewer>Log File* to open a separate window displaying the log file: this is easier to use than the transcript, shows more information and provides useful search and edit functions.

Introduction

This section lists some common problem areas.

Performance

If Figaro runs slowly in interactive mode, empty memory may still be allocated. Pull down *Help>Compact Memory* to use memory more efficiently.

Mapping

If a design is not mapped as you'd expected, check the help on "Mapping" or "Retargeting".

Placement, Routing and Timing

The most common problem is failure to complete placement and routing of a design. For details see the help on "Troubleshooting", particularly the jumps to "Design cannot be placed" and "Design cannot be routed".

Colors

Figaro uses colors to provide much of its information. If your system has few colors available, this may cause problems, particularly in the Compile window display. For example, if your system only has one shade of green, express buses may be shown in gray, which normally shows unused routing.

To avoid problems pull down *Options>Display Options* and select "Compile". Click the first color button in the list. This shows the color palette available on your system. If there are very few colors or they are badly distributed (for example, lots of reds but no greens), do one of the following:

As a short-term measure, use the color button to change the colors of any resources that look confusingly close. (For a list of the defaults, see the help on "Colors in Figaro".)

Consider reconfiguring your system:

- If you are on a PC running Windows it is a good idea to use a display driver with at least 256 colors.
- If you're running X-Windows on a UNIX platform, close down Figaro then use one of the following (see the `xtdcmap` "man page" for details.):

```
"xstdcmap -all"  or  "xstdcmap -default".
```

Restart Figaro, which should now have a wider selection of available colors.

Display

If a PC has problems displaying graphics in the Compile window, pull down *Options>Display Options* and select "Compile", then uncheck the "Faster complex graphics" option.

Frequently-Asked Questions

This list gives answers to some frequent questions asked by Figaro users. (This list is also available on-line: from the Figaro Overview help screen, click on Troubleshooting, then on Frequently Asked Questions.)

Q. The menu option I want to use is grayed out. Why?

A. This is probably because you are not in the correct tool window for the command, or because you have not selected the object you want to work on. Click-hold the mouse on the command and look at the status bar for a simple explanation; press SHIFT+F1 for further details.

Q. Elements of the design seem to be hidden from view. Why?

A. Check the setting of *Options>Display Options* for the tool window. You may have turned off the display of some wiring, net, port or other element.

Q. Figaro is "hanging". The last thing I was doing was using a dialog, which has now disappeared from view. What should I do?

A. This occurs when a window other than the dialog becomes the current window. For example, this happens if you click in a window belonging to a different application. Use your PC facility for listing open tasks to return to the open Figaro dialog.

Q. I have several dialogs open, but they keep disappearing below the windows. Why?

A. Use *Window>Dialogs to Front* to bring all your dialogs back into view.

Q. I have selected an object in the *Edit>Find* dialog box and it doesn't show up in the Compile window. Why not?

A. Not all objects are drawn to prevent the display becoming unnecessarily cluttered. For example the global clock/reset resources are not shown since they are attached to all core cells.

Q. I displayed information on the same instance in the Design Browser and Parts (or Compile) windows. However, the instance and net names shown in the two dialogs are different.

A. This is because only the Design Browser shows the original design. Other browsers and windows show Figaro's implementation of the design: for details, see the on-line help on "Association".

Q. My design structure is not maintained after compilation. Why?

A. You can compile individual blocks of your design separately by compiling them as user macros and storing them in a user macro library. The macros will be used when you compile the whole design.

Q. I can't export the type of file I want. Why not?

A. The files you are able to export depend on which is the current window. For example, you can only export a pinout constraints file if the Parts window is the current window and none of the parts in the window is selected. See the help on *File>Export* for more details.

Q. My design has one error on import but it's perfectly all right. Why?

A. Scroll up the transcript window or check the log file to see if there are other errors. The design checker gives a summary of the number of errors it finds. If you opened as a macro a design which has pads, there will be a single error.

Q. Multiple design instances are in contention in the Compile window. How can I select just one of them to move it?

A. Select the location in the Compile window. All the objects placed there will be highlighted in the Design Browser. Now select just one of them in the Design Browser and drag it to a new location in the Compile window. This leaves the other object(s) behind. Repeat this as necessary until there's just one object at the location in question.

Q. I clicked the mouse in the transcript and a small menu appeared. What's this for?

A. You only need to use this if you're reporting a problem. It's for pasting from the transcript into the report form.

Q. Since I mapped my AT6K design, Figaro's performance has slowed down. Why?

A. Mapping large AT6K designs can greatly increase the size of the footprint left by Figaro on your computer's memory, making performance sluggish. Before proceeding, save your mapped design and exit Figaro to free up this extra memory. **This does not apply for AT40K designs**.

Before you Report a Problem....

If you have a problem, work through this section, point by point, before you report it:

Have you installed Figaro properly?

Check against the Installation section of this manual. You should have all the files listed there.

Is the system set up correctly with all the necessary files? If you're using a PC does it have enough memory?

Did you try to load an illegal or corrupted netlist?

Check the on-line help on "Netlists as Input".

Check the transcript or log file for messages beginning "Design Checker:". Some of these mean that the design cannot be read in by Figaro: see the section "Design Check Error Messages" for details.

Is the file you're looking for really in the directory you think?

Quick checks

Is there an easy answer to the problem? Check against the list of Frequently Asked Questions (above). This shows some places where it's possible to misunderstand Figaro.

Has the problem already been reported?

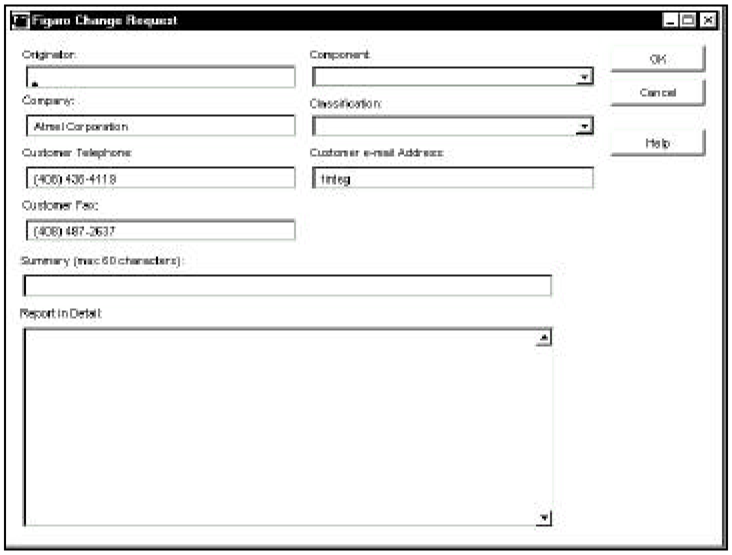
The Figaro *Known Problems and Solutions* document contains descriptions of problems which Figaro technical support knows about. Methods of avoiding these are given, allowing you to work around a problem.

This document can be accessed on-line using the *Help>Review KPS* pull down.

Reporting a Problem

Once you're sure your problem is not something you can solve yourself (see above), pull down *Help>Report a Problem.* This displays a Report Form for sending to technical support.

The layout of the form, shown below, is straightforward, but if you do need more information, click on its *Help* button.



Originator:

Component:

Company: Atmel Corporation

Classification:

Customer Telephone: (408) 436-4118

Customer e-mail Address: tinteg

Customer Fax: (408) 487-2637

Summary (max 60 characters):

Report in Detail:

The Problem Report Form

Enter your details and a summary of the problem. In the "Report in Detail" text box, give a fuller description, including:

- What you were doing when the problem appeared.
- How the problem looked.
- A copy of the log file, *design*.LOG.

  To paste in the log file, click the right mouse button and select *paste file* from the list, then type in the name of the log file and press ENTER.
- A copy of the Figaro stack file (FIGARO.DMP) which contains a record of previous functions executed.
- Any relevant information from the transcript.

  To paste from the transcript, move the cursor over the transcript space, click the right mouse button and select *copy* from the menu. Then click in the report form, click the right mouse button and select *paste* from the menu.)

Once you've supplied all the information, click *OK*. Clicking *OK* produces a report file FCR.*n*, where *n* is the number of this report. If it's your first report, it will be FCR.0. This file is written to the directory FIGARO\REPORTS and to the current directory. Fax or e-mail a copy of this report to technical support.

Installing an Update to Figaro

Installing an update means patching your version of Figaro by applying a patch file sent by Technical Support to solve a problem. You can patch permanently or just for one session (so that you can try out a patch).

For details of product updates, contact your supplier.

Installing an Update

To update your Figaro version permanently, do the following:

1. If Figaro is open and you have performed any operations since opening it, close it down. (You cannot patch Figaro after opening a design.)
2. Copy your FIGARO.INI file to save it from being overwritten.
3. Restart Figaro, but don't open a design.
4. Pull down *Help>Install Update*, enter the patch file name and click *OK*.

   Figaro installs the patch and displays the Install Upgrade dialog asking whether you want to update the release.
5. Click *OK* to update the release: the patch applies for this and all future Figaro sessions.

NOTE

If you want to make a permanent update on a workstation, you should only install it from Figaro's home directory. If you install an upgrade from any directory other than the Figaro `bin` directory, you will create a new FIGARO.IM file in the current directory but the source FIGARO.IM will remain unchanged with the previous version.

To update the Figaro version just for one session to evaluate the patch, do the following:

1. If Figaro is open and you have performed any operations since opening it, close it down. (You cannot patch Figaro after opening a design.)
2. Restart Figaro, but don't open a design.
3. Pull down *Help>Install Update*, enter the patch file name and click *OK*.

   Figaro installs the patch and displays the Install Upgrade dialog asking whether you want to update the release.
4. Click *Cancel*: the patch applies for the current session only. When you close Figaro down, the patch will be removed.

Installing Library Updates

There is a parallel function for updating the package and vendor libraries used by Figaro. Pull down *Help>Update Library* and proceed in a similar manner as when installing an update.

A Summary of Menu Commands

This appendix describes the Figaro menu commands and the shortcuts defined for frequently-used commands. For full information on any command, use the on-line help. There are two easy ways to display help on a command:

- Pull down the menu, move the cursor onto the command and press the SHIFT and F1 keys simultaneously.
- Pull down *Help>Search for Topic* and type in the first few characters of the command name (on a workstation, this field is case-sensitive: note that most words begin with a capital letter). Select the command name from the list and click *Show Topics*. (If the help window is already open, click the *Search* button at the top of the window.)

There are also buttons on the toolbars for commands you might use a lot, and keyboard shortcuts for many more commands. These are listed in full next to the respective menu commands. If you hit an incorrect key, or a command is not available, Figaro beeps.

NOTE

The window name in parentheses after the headings in the following list shows which tool "owns" the menu. The entries in some menus, like *Edit* and *View*, change depending on which tool window is the current window.

Menu Commands and Shortcuts

File Menu (Figaro Window)

Design Setup	Specify the design directory.
Open as Design	Open a design to compile a bitstream.
Open as Macro	Open a design to compile a macro.
Open as ECO	Read in new version of a previously- implemented netlist for quick update.
Close	Close the current window.
Run Batch	Open a Figaro batch file (*.fbf) to be run.
Save Batch File	Produce a Figaro batch file for the currently loaded design.
Import Constraints	Read in a constraints file.
Export	Export a board report, back-annotation file or timing report files.
Save	Save the design.
Save As	Save the current design to a specified file.
Revert to Saved	Discard all changes made to the design since the last save.
Print	Print the current window.
Exit	Close down Figaro.

File Menu (Figaro Window)

Command	Key	Button
File>Open as Design	Ctrl+O	
File>Close	Ctrl+F4	
File>Export	Ctrl+E	
File>Save	Ctrl+S	💾
File>Print	Ctrl+P	🖨
File>Exit	Ctrl+Q	

Edit Menu

Undo	Return your design to the state it was in before you performed the last operation.
Redo	Restore changes reversed using Undo.
Select in Area	Select all the objects in an area you specify with the cursor.
Deselect in Area	Deselect all the objects in a specified area.
Deselect All	Deselect all selected items.
Timing Constraints	Open the timing constraints editor dialog.
Constraints	Edit browser, mapping or part constraints.
IO Pad Attributes	Open the IO pad attributes editor dialog.
Find	Find a specified object.
Info	Display information on a selected object.
Design Info	Display statistics on a design.

Edit Menu (Parts Window only)

Add Part(s)	Add or replace the target device.
Change Part Speed or Application	Replace the target device with another from the same family, but with a different speed or application.
Remove Part(s)	Cut selected part(s).
Remove Unused Parts	Cut parts which hold no design logic.
Arrange I/Os	Display I/Os neatly around part(s).
Rotate Parts	Rotates the selected part(s) clockwise through 90 degrees
Assign Pin Locks	Define pinout.
Auto-Partition Design into Parts	Partition automatically.
Unpartition Design	Undo partitioning.

Edit Menu

Command	Key	Button
Edit>Undo	Ctrl+Z	↶
Edit>Redo	Ctrl+Y	↷
Edit>Select in Area	Ctrl+A	
Edit>Deselect All	Esc	
Edit>Constraints	@	
Edit>Find	Ctrl+F	🔍
Edit>Info	I	ℹ
Edit>Design Info	Ctrl+F1	

Edit Menu (Compile Window)

Place by Net	Place instances attached to a net.
Next Macro Alternative (not AT40K)	Replace the selected macro(s) with a functionally equivalent macro.
Next Primitive Alternative (not AT40K)	Replace the selected primitive (on a single logic block) with an equivalent.
Rotate R90	Rotate selected items 90° clockwise.
Rotate L90	Rotate selected items 90° counterclockwise.
Rotate 180	Rotate selected items 180°.
Flip Horizontal	Reflect selected items on vertical axis.
Flip Vertical	Reflect selected items on horizontal axis.
Close Location	Prevent logic being placed in a location.
Open Location	Re-open a closed location.
Route Unrouted Nets	Route placed instances' nets.
Route Selected Nets	Route the working net automatically.
Discard Route	Discard the route of the working net.
Discard From (manual route)	Discard the routed part of the working net (from the selection to destination(s)).
Tidy Net (manual route)	Remove antennae from the working net.
Edit Net Routing (normal mode)	Mark the selected net as the working net and enter manual routing mode.
Finish Net Routing (manual route)	End manual routing and return to normal display.
Soften Library Routing	Unlock all locked library routing associated with the selected nets.
Revert to Library Routing	Discard current routing and reinstate library routing for the selected nets.
Lock	Fix the selected object so automatic placement and routing cannot change it.
Unlock	Free the selected object so automatic placement and routing can change it.

Edit Menu (Compile Window)

Command	Key	Button
Edit>Place By Net	N	
Edit>Next Macro Alternative	Ctrl+N	
Edit>Next Primitive Alternative	T	
Edit>Rotate R90	R	▣
Edit>Rotate L90	L	▣
Edit>Rotate 180	U	
Edit>Flip Horizontal	H	▣
Edit>Flip Vertical	V	▣
Edit>Close Location	]	
Edit>Open Location	[	
Edit>Route Unrouted Nets		▣
Edit>Route Selected Nets	Ctrl+R	
Edit>Discard Route	D	
Edit>Discard From	Ctrl+D	
Edit>Tidy Net	C	
Edit>Edit Net Routing	W	▣
Edit>Finish Net Routing	E	▣
Edit>Lock	K	🔒 (toggle)
Edit>Unlock	J	🔒 (toggle)

View Menu

Zoom Fit	Show the entire contents of the window.
Zoom In	Zoom in at x2 on the center of the window.
Zoom Out	Zoom out at x0.5 on the center of the window.
Zoom to Previous	Return to the previous zoom setting.
Zoom to Area	Zoom to a given area.
Zoom to Selected	Zoom in on the selected object(s).
Pan to Selected	Pan to the selected object.
Scale	Zoom in at a particular scale factor.
Center	Center the view on the next place you click.
Refresh	Update the window contents.
Push Window	Display the contents of this object in a new window.
Push	Display the contents of this object in this window.
Return to Previous	Display the level of hierarchy above this one (undo the last Push).

View Menu (Browsers only)

Expand One Level	Expand a single level below this node.
Expand Branch	Expand all levels below this node.
Collapse Branch	Collapse all levels below this node.

View Menu

Command	**Key**	**Button**
View>Zoom Fit	F4	▦
View>Zoom In	F7	⊕
View>Zoom Out	F8	⊖
View>Zoom to Previous	F3	
View>Zoom to Area	F9	⬚
View>Zoom to Selected	F2	
View>Pan to Selected	P	
View>Scale	S	
View>Center	F6	
View>Refresh	F5	
View>Push Window	Ctrl+F7	
View>Push	Shift+F7	
View>Return to Previous	Shift+F8	

View Menu (Browsers only)

Command	Key	Button
View>Expand One Level	+	
View>Expand Branch	*	
View>Collapse Branch	-	

Timing Menu (Compile Window only)

Show Analyzed Paths	Display delays on long/short/critical paths.
Show Net Delays	Display delay information in the transcript.
Measure Delay	Measure delays between two ports.
Timing Display	Change Compile window to timing display.

Options Menu

Transcript	Toggle display of the transcript on/off.
Flowbars	Toggle flowbar display on/off.
Toolbars	Show or hide Figaro and Compile window toolbars (dialog).
Options	Display or change Figaro options.
Display Options	Set display options for the current window.

Library Menu (Figaro Window)

Library Setup	Change the library search path.
Check-In Macro	Check user macro into user macro library.
Check-Out Macro	Check macro out from user macro library.
Delete Macro	Delete macro from a user macro library.
Translate Macro	Translate macros from an AT6k to AT40k user macro library.

Timing Menu (Compile Window only)

Command	Key	Button
Timing>Show Net Delays	=	
Timing>Measure Delay	M	⏱
Timing>Timing Display		⏱ (toggle)

Options Menu

Command	Key	Button
Options>Flowbars	F10 (not PC)	
Options>Options	O	
Options>Display Options	F11 (PC only)	

Flow Menu

Open as Design or Macro	Open a design to compile a bitstream or user macro.
Map	Implement design mapped to selected vendor architecture.
Parts (sub-menu)	Access commands in the Parts sub-menu.
Compile (sub-menu)	Access commands in the Compile sub-menu.

Parts Sub-Menu

The *Parts* sub-menu contains the following entries.:

All Steps	Run all the steps in the Parts sub-menu.
Add Part(s)	Add or replace the target device.
Auto-Partition Design into Parts	Partition automatically.

Compile Sub-Menu

The *Compile* sub-menu contains the following entries. The placement, routing and bitstreaming commands are part-specific.

All Steps	Run all the steps in the Compile sub-menu.
Auto Compile All	Automatically program all parts.
Initial Placement	Run initial placement.
Optimize Placement	Run optimized placement.
Initial Route	Run initial routing.
Optimize Route	Run optimized routing.
Bitstream	Output the design as a bitstream (bitstream mode only).
Check-In Macro	Check user macro into user macro library (user macro mode only).

Compile Sub-Menu

The *Compile* sub-menu contains the following shortcuts. Both are part-specific.

Command	Key	Button
Compile>Optimize Placement	Shift+F10 (not PC)	
Compile>Optimize Route	Shift+F11 (PC only)	

Window Menu (Figaro Window)

Arrange Icons	Arrange the icons on the Figaro desktop.
Cascade	Reorganize a stacked window display.
Tile	Display all windows without any overlap.
New Design Browser	Open a Design Browser.
New Parts Window	Open a Parts window.
New Map Browser	Open a browser displaying the logic elements of the mapped design
New Sub-Design Browser	Open a browser displaying the logic partitioned to a selected part
New Compile Window	Open a window on the target device.
New Viewer	Open viewers for log files, path analysis reports, net delay tables, design statistics.
New Shell Window	Open a DOS, Xterm or cmdtool window.
Dialogs to Front	Make dialog boxes for a tool visible.
Close Dialogs	Close all dialog boxes.
List of open windows	Toggle between windows currently open

Help Menu (Figaro Window)

How to Use Help	Show how to use on-line help.
Figaro Overview	Show the on-line help contents page.
Current Window	Show on-line help tailored to the window.
Search for Topic	Search for specific help information.
Review KPS	Open known problems viewer.
Report a Problem	Report a problem to Technical Support.
Install Update	Install a new version of Figaro.
Update Library	Install a library update.
Compact Memory	Tidy up unused memory.
About Figaro	Display version no. and copyright notice.

Window Menu (Figaro Window)

Command	**Key**	**Button**
Window>Arrange Icons	Shift+F6	
Window>Cascade	Shift+F5	
Window>Tile	Shift+F4	
Window>Dialogs to Front	Shift+F3	
Window>New Viewer>Log File		🔍
Window>New Shell Window		🗔

Help Menu (Figaro Window)

Command	Key	Button
Help>Figaro Overview	F1	
Help>Using the Current Window	?	
Help>Search for Topic	Ctrl+H	

Additional Toolbar Buttons

Figaro also provides toolbar buttons for the following options and functions:

Function	Button
Select input nets of instances (Display Options: Selecting)	
Select output nets of instances (Display Options: Selecting)	
Auto re-route after manual move (Options: Place and Route)	
Show or hide unrouted nets	
Show or hide routed nets	
Show or hide routing resources	

Standard Function Keys

The function keys labeled "F1 to "F11" do the following (F11 is relevant for PCs only, F12 is not used at all):

Key(s)	Action
F1	Opens the help window
Ctrl+F1	Edit>Design Info
Shift+F1	Shows help on highlighted command in menu.
F2	View>Zoom to Selected
F3	View>Zoom to Previous
F4	View>Zoom Fit
Ctrl+F4	File>Close
Shift+F4	Window>Tile
F5	View>Refresh
Shift+F5	Window>Cascade
F6	Center in window.
Shift+F6	Window>Arrange Icons
F7	View>Zoom In
Ctrl+F7	View>Push Window
Shift+F7	View>Push
F8	View>Zoom Out
Shift+F8	View>Return to Previous
F9	View>Zoom to Area
F10	Options>Flowbars (not PC)
Shift+F10	Flow>Compile>Optimize Placement (not PC)
F11	Options>Display Options (PC only)
Shift+F11	Flow>Compile>Optimize Route (PC only)

Figaro Files

This appendix describes the following files:

- The startup files FIGARO.INI and *design*.INI.
- The log file *design*.LOG.
- The constraints files *design*.**RCT**, *design*.**PTN**, *design*.**PIN** and *design*.**TMG**.
- The pinout report *design*.**PIR**.
- The statistics file *design*.**STS**.
- The nets timing report file *design*.**NDL** or *design*.**NDS**.
- The paths timing report file *design*.**PDL** or *design*.**PDS**.
- User Macro library files *.**LIB**.

Most files are named *design*.*, where *design* is the name of the design you opened or, if you have saved the session with a different name, the new name of the design. Files normally reside in the design directory, though you can write some files to other locations.

Where the file contains information relevant to only one of the parts in a multi-chip implementation, the naming convention incorporates the part letter as well: for example *design_part*.ext

Other files produced by Figaro are:

- The bitstream file *design*.BST
- Saved design session: *design*.FGD
- Problem report files: FCR.*.

NOTE For most files, when you produce a new copy, the previous copy is saved, with a "~" character as the last character of the extension. 40KTEST.LOG, for example, would be saved as 40KTEST.LO~.

The Startup File: FIGARO.INI

The FIGARO.INI file resides in the directory from which Figaro was invoked (usually the bin directory) and records the following information:

- Any changes you make to default parameters or options using either of the options dialogs. The new values are stored in the FIGARO.INI file for use in future Figaro sessions.
- Design directory locations, and the tools flow and configuration used for each design.

The .INI file is written when you close a session during which you changed default parameters and is read in automatically when you open a new Figaro session.

If you want to revert to Figaro's default values, change the name of the FIGARO.INI file. When there is no FIGARO.INI file in the BIN directory the original defaults are reinstated.

A simple FIGARO.INI file might look like this:

```
; Produced by Figaro version Atmel train2.1.12 on May 8,
  1997 at 2:31:54 pm
;
; Modified Options only
;
[Design Setup]
    Design Data = *c:\figaro\

examples\at40K\vlogic\40ktest<40ktest><viewlogic_wir_macro
    _level><atmel40k>c:\figaro\examples\4bitalu<4bitalu>
    <viewlogic_wir_macro_level><atmel6k>

[Timing Analysis]
     Measure Delay Paths Limit = 5

[Compile, Instances, Routed Locations]
     Color = 204 153 64

[Partitioner, Basic Options]
     P&R Utilization = 6
```

The design.INI File

The initialization file *design*.INI resides in the design directory and records the following information:

Design Flow	Whether the design was compiled for bitstream or user macro.
Library Setup	Library search path and library names.
Mapping Options	Whether mapping is enabled or not.

To reset these options to their defaults, you can either change the name of the file or move it out of the design directory. (When Figaro finds no file with this name, it automatically reverts to the defaults.)

A simple **design.INI** file might look like this:

Example

```
; Produced by Figaro version Atmel train2.1.12 on May 8,
  1997 at 2:31:54 pm
;
; All Design Specific Options
;

[Place and Route]
     Compile for User Macro = false

[Mapping, Mapping Control]
     Mapping Enabled = false
     Clock/Reset network only = false
     Map to Multi-Core Macros = false

[Library Setup]
     Library Search Path = c:\figaro\examples\design
     Library Names = user
```

The Log File: design.LOG

The log file records what happens during a Figaro session. (A subset of these messages is also shown in the transcript.) The file resides in the design directory.

If you encounter any problems with the Figaro flow, look at the log file first. The best way to do this is to pull down *Window>New Viewer>Log File*. The Log Viewer provides basic functions for updating and editing the log file and you can also highlight error and warning messages in the file to make them easier to locate.

Log File Format

For each design step in Figaro, the log file records:

- The name of the step, like this:
```
===================
Name
===================
```
- The values of options used by that design step:
```
------------- Options-------------
[Step options]
```
- The progress of the design step:
```
Step: info - text
```
- Any warnings associated with the design step:
```
Step:<<WARN>> - Text
```
- Any errors:
```
********************************
Step: **ERROR** - Text
********************************
```

Saving the Previous Log File

When you produce a new copy of the file **design**.LOG, the previous copy of the file is saved as **design**.LO~.

Log File Example

This sample shows messages written to the log by the Optimize Placement step:

```
===================================================
Optimize Placement: info - Starting on June 9, 1997 at 5:58:43 pm
===================================================
; --------- Options for Place -----------

[Place and Route]
    Quality = 4
    Timing driven = false
    Route if Place contention = false
    Auto set parameters = false
    Equivalent port swap = true

[Place and Route, Manual Editing]
    Auto re-route after manual move = true

[Place and Route, Atmel 40K]
    Allow global clock/reset signals to use non-global
    resources = true
    Allow auto pin-swap on locked macros = false
    Auto-soften conflicting hard routing in Optimize
    Route = true

; ---- Constraints for Place (device A) ----

Place: info - Clock net $1N41 (external name CLOCK1) assigned to global clock location GCK4
Place: info - Derived clock nets: 3COUNT\Z0, 3COUNT\Z1, CLOCK2
Place: info - Reset net fig1000 (external name R) assigned to global reset location IO46
; ----------------------------------------------

Place: info - The quality setting (4) is greater than the estimated minimum (3)
Optimize Placement: info - Finished optimize Placement at June 9 1997 at 6:23:35 pm
```

The Repeat Constraints File: design.RCT

When you save your design, Figaro writes any pinout and timing constraints to the repeat constraints file **design**.RCT. This file lets you reopen a design with the same pinout or timing, so there's no need to lock pins or import timing constraints again a second time. The constraints are stored in the same format as in the pinout or timing constraints files.

If you want Figaro to import an existing repeat constraints file when you load a particular design, you should enable the auto-import function in the design constraints options. To do this, open the Options dialog and select "Design Constraints", check the box labeled "Auto-import Repeat Constraints" and press *OK* to accept the setting.

You can create a .RCT file from an existing .RCT file or another constraints file, then rename it and use the constraints for other designs.

Producing a design.RCT File

If you saved the design, there will be a **design**.RCT file already. If you did not, there are two other ways to produce this:

- If you produced a pinout constraints file (see below) for a design, but did not save the design, you will not have a **design**.RCT file. You can copy the pinout constraints file **design**.PIN to **design**.RCT, then use this as described above.
- If the .RCT file does not have the part/pinout constraints you want, edit it with a text editor. (Take care to preserve the format.) This may be quicker than setting new pin locks in Figaro. You can copy a different .RCT file to the name you want, then edit this.

Saving the Previous Constraints File

When you produce a new copy of a constraints file from Figaro, the previous copy of the file is saved, with a "~" character as the last character of the file extension: **design**.RC~, *design*.PI~, *design*.TM~ and so on.

The Partition Constraints File: design.PTN

Partition constraints control the partitioning of logic into devices. There are two ways to produce the file:

- Using Figaro, either by partitioning manually or using *Edit>Constraints*. Output the file using *File>Export*.
- Create an ASCII text file using an editor.

If you use the latter method, note that all names must be specified as full hierarchical names which include the top level instance name (i.e. the design name) and that all strings must be enclosed in single quotes (').

Make sure that any partition constraints in the file are preceded by the appropriate part constraints, because partition locks require that the part has already been defined.

Preserving Cells or Instances

You can preserve one or more specified design instances/cells, or **all** design instances/cells.

```
PartitionPreserveInstances ( 'instName' )
    PartitionPreserveInstances ( ALL )


PartitionPreserveCells ( 'cellName')
PartitionPreserveCells ( ALL )
```

For example, this two line file would preserve the two named instances and all the '<cell_name>' cells in the current design:

```
PartitionPreserveInstances ( 'alu/adder/add_1' 'alu/adder/add_2')
PartitionPreserveCells ( '<cell_name>' )
```

While this one line file would preserve all instances:

```
PartitionPreserveInstances ( ALL )
```

Changing Current Settings

If you have used a file with statements like those above to preserve instances, you can reset the mapping parameters by importing a new file with an "unpreserve" statement:

```
PartitionUnpreserveInstances ( [ instName | ALL ] )
      PartitionUnpreserveCells ( [ cellName | ALL ] )
```

NOTE If you apply partition preserve constraints to particular instances *before* mapping the design, you must also apply mapping preserve constraints to the same instances to ensure that they survive the map step unscathed.

Locking or Unlocking Partitioning

To lock partitioning use a PartitionLock statement.

```
PartitionLock ( [instName | cellName ] toPart <letter> )
```

This example specifies that instance '\test\I6' is to be locked in part A:

```
PartitionLock ( '\test\I6' toPart 'A' )
```

NOTE When you use partition lock constraints in your .PTN file, remember to include the relevant part constraints as well. You cannot apply partition lock constraints to pads.

To undo a lock set by a file read in earlier, use the "PartitionUnlock" statement:

```
PartitionUnlock ( '\test\I6' )
```

The Pinout Constraints File: design.PIN

Figaro can read in pinout constraints from a file. The file is named **design**.PIN. It lists the part used in the design along with any manual pin locks (whether read in as constraints or assigned using the Parts window.) The file does not include pin locks assigned by Figaro.

To produce a pinout constraints file, make the Parts window the current window then pull down *File>Export* and check the "Locked Pinout (*.pin)" box. You can rename the exported file then import it for use in a different design. You can also create or edit a file using a text editor.

The file can also be used as a repeat constraints file. To do this, edit the file according to the constraints syntax to add or remove parts and pinouts, then save it as **design**.RCT. Also, you can read in pinout constraints from a .RCT file:

```
AddPartNamed ( 'AT40K10-2RC' 'A'  )
AddPartNamed ( 'AT40K10-2RC' 'A'  )

Pinout ( 'A_OUTPUT1' toPin 'A.152' )
Pinout ( 'A_OUTPUT2' toPin 'A.153' )
Pinout ( 'A_OUTPUT3' toPin 'A.154' )
Pinout ( 'A_OUTPUT4' toPin 'A.155' )
```

The Pinout Report: design.PIR

You can export a report showing the pinout constraints. Use this in documents or reports, or to lay out the PCB. The format is the same as for the pinout constraints file, but the report shows all locks, not just those you lock manually. (If the design has been placed and routed, the file will include the pinout chosen by Figaro.)

The Timing Constraints File: design.TMG

Figaro can read in timing assertions and constraints from a timing constraints file, **design**.TMG using the *File>Import Constraints* command. Figaro uses a sub-set of these constraints in timing-driven compilation and all of them when reporting timing values during interactive timing analysis.

You can create this file in one of two ways:

- Using a standard text editor to create a file from scratch.
- Using the *File>Export* command to output a file of all the timing assertions and constraints currently valid for the design. These may have been set either by importing an existing .TMG file or using the Figaro constraints editor (*Edit>Timing Constraints*).

The following pages show an example timing file and a list of the keywords used. For a full description of the file syntax, open the on-line help and *Search* for the keyword "Syntax" then the topic "Timing Constraints File Syntax".

Example Timing Constraints File Contents

This example shows an entry for each type of constraint:

```
PrimaryClock (
     'CLOCK1'
     dutyCycle (
        edge RISING
        halfPeriod ( 30 )
        period ( 60 )
     )
)
SecondaryClock (
     'CLOCK2'
     dutyCycle (
        edge RISING
        halfPeriod ( 35 45 )
        period ( 120 )
     )
     primaryClock (
        edge RISING
        offset ( 20 )
     )
)
DerivedClock (
     '3COUNT\$1I5 Q'
     dutyCycle (
        edge RISING
        halfPeriod ( 80 )
        period ( 120 )
     )
     otherClock (
        'CLOCK1'
        edge RISING
     )
)
AssertIO (
          'A'
          direction INPUT
          clock 'CLOCK1'
          edge RISING
          rise ( 5 10 )
     fall ( 15 20 )
     )
AsyncDelay (
     from ( '$1I12 Q' )
     to ( '$1I13 A' )
     rise (10 25)
     fall (31)
)
FalsePath (
          '$1I193 Q'
)
```

Valid Keywords Used in the Timing Constraints File

The timing constraints file parser is case-sensitive. It only accepts the following words, valid strings and numbers.

Keyword	Function
AssertIO	Starts definition of an I/O assertion.
AsyncDelay	Start definition of an asynchronous delay assertion.
clock	Starts definition of the clock referenced by an I/O assertion definition.
Cut	Specifies the net to be cut disconnecting macro ports
direction	Defines the direction of an I/O assertion (INPUT or OUTPUT).
dutyCycle	Starts definition of the duty cycle.
edge	Starts definition of an edge (RISING or FALLING).
fall	Starts definition of an I/O assertion's fall delay(s).
FalsePath	Starts definition of a new false path.
halfPeriod	Starts definition of a half period (the rise or fall period).
offset	Starts definition of an offset from the primary clock.
period	Starts definition of a period.
PrimaryClock	Starts definition of the primary clock.
primaryClock	Starts definition of a secondary clock's relationship with the primary clock.
rise	Starts definition of an I/O assertion's rise delay(s).
SecondaryClock	Starts definition of a secondary clock assertion.

The Statistics File: design.STS

The statistics file holds information on placement and routing. It is updated by Figaro at the end of each design step.

```
FIGARO STATISTICS FILE
=====================
Date And Time  : June 9, 1997 at 11:15:47 am
Device Type    : AT40K10-2RC
Figaro Version : Atmel train2.1.19
Design Statistics for " 40ktest "
Design Step : Initial Placement

Number of Macros :                                          359
Number of Nets :                                            386
Number of Pins :                                            1320
Average Pins per Net :                                      3.42
Maximum Pins per Net :                                      11
Number of Nets :                                            386 unrouted
                                                            0 routed
Number of Logic only Macros :                               0 unplaced
                                                            309 placed
Number of Logic Cells :                                     309 used
                                                            0 needed
                                                            267 free
Number of macros with RAM :                                 0 unplaced
                                                            4 placed
Number of RAM Cells :                                       4 used
                                                            0 needed
                                                            32 free
Number of IO Macros :                                       0 unplaced
                                                            46 placed
Number of IO Cells :                                        46 used
                                                            0 needed
                                                            144 free
Number of Flip-Flops :                                      223
Number of Gates :                                           90
Number of Macro Wires :                                     0
Number of Route Wires :                                     0
Number of Buses :                                           0
     Local Buses :                                          0
     Express Buses :                                        0
Number of Clock and Reset Combinations :                    1
Number of Cell Contentions :                                834
Number of Net Contentions :                                 0
```

The fields in the statistics file are as follows:

Number of Macros

The number of macro design instances placed on the device.

Number of Nets

The number of design nets on the device.

Number of Pins

The total number of pins used (except pins on the global clock and reset nets).

Average Pins per Net

The number of pins divided by the number of nets.

Maximum Pins per Net

The number of pins in the most complex case. Multi-pin nets are more difficult to route.

Number of Nets

How many nets have not been routed, and how many have been.

Number of Logic only Macros

How many logic-only macros have not been placed, and how many have been.

Number of Logic Cells

"Used" shows how many locations are used for routing or logic.

"Needed" shows how many will be needed for logic which has still to be placed, if the default shape from the library is used for each macro or primitive.

"Free" shows how many locations are still available.

Number of Macros with RAM

How many of these macros have not been placed, and how many have been.

Number of RAM Cells

"Used" shows how many of the total number of RAM cells have been used. "Needed" shows how many will be required for the design. "Free" indicates how many are still available.

Number of I/O Macros

How many of these macros have not been placed, and how many have been.

Number of I/O Cells

"Used" shows how many of the total number of I/O cells are used for routing or logic. "Needed" shows how many will be needed. "Free" shows how many are still available.

Number of Flip-Flops

The number of logic locations that are being used as registers. (That is, the number of locations which are configured as flip-flops and use the flip-flop outputs.)

Number of Gates

The number of logic locations that contain design logic (not used for flip-flops or purely for routing).

Number of Macro Wires

The number of logic locations which are part of a macro but also used for routing.

Number of Route Wires

The number of routed locations.

Number of Buses

The total amount of routing resource used (both internal and external to macros). Local buses and express buses are shown separately.

Number of Clock and Reset Combinations

The number of different combinations of clock and reset lines used by the flip-flops in the design. (For example, if the only combinations used were Clock 1 and Reset 1, Clock 3 and Reset 1, this would have a value of 2). The larger the number, the more difficult the design is to place and route.

Number of Cell Contentions

The number of logic locations which currently have multiple design instances placed on them.

Number of Net Contentions

The total number of nets currently competing for routing resource.

The Net Delay Table: design.NDL or .NDS

The net delay table gives a list of the net delays in the placed and routed design. The table contents are controlled by the "Timing Analysis" options. You can produce a normal report (.NDL) or a report in spreadsheet format for Microsoft Excel (.NDS).

The net delay table shows the following information:

- Time of creation.
- Design name.
- Part.
- Netlist file(s).
- Environmental settings, such as delay range, derating factor, etc.
- List of nets in order of delay (longest first). Net information in the normal report is shown as follows:

  Net-name

  From: *source-port*

  To: *destination-port rise-delay (R minimum maximum)*

  fall-delay (F minimum maximum)

NOTE

Delay values are shown in nanoseconds.

The spreadsheet format presents the data in tabular form, with "R" and "F" as column headers; see the following examples.

Examples

These examples show truncated files.

Normal Format (.NDL)

```
Net Delay Table June 6, 1997 at 3:37:56 pm
Design: adder
Part: A (AT40K10-2RC)
Imported from: #('c:\figaro\examples\adder\wir\adder.1')

       Delay calculator
       ------------------------
       Mode             = #MinMax
       derating factor   = (1.0 1.0)

     $1I69\W2
       From: $1I69\A1\G5 Q
         To: $1I69\A2\GO A   R (6.58 15.24) F (6.49 15.47)
         To: $1I69\A2\G4 B   R (2.03 4.5)   F (2.39 4.98)
         To: $1I69\A2\G3 B   R (2.03 4.5)   F (2.39 4.98)
```

Spreadsheet Format (.NDS)

The spreadsheet format looks like this:

```
Net Delay Table June 6, 1997 at 3:37:56 pm
Design: adder
Part: A (AT40K10-2RC)
Imported from: #('c:\figaro\examples\adder\wir\adder.1')

Net       From    To        R(min) R(max)  F(min) F(max)
$1I69\W1  A0\G5 Q A1\GO A   4.83   11.69   5.55   12.66
$1I69\W1  A0\G5 Q A1\G3 B   4.43   10.69   5.06   11.56
$1I69\W1  A0\G5 Q A1\G4 B   3.90    9.17   4.59    9.89
```

The Path Analysis Report: design.PDL or .PDS

The Path Analysis report lists the path delays in the placed and routed design. Its contents are controlled using the "Timing Analysis" options. Again, you can export a normal report (.PDL) or a report in spreadsheet format for Microsoft Excel (.PDS).

The default values are the long and short path slack values for the 10 most critical paths, the 10 longest paths and the 10 shortest paths. Negative slack values always signify that the timing constraints set for the design have not been met.

You can select whether the times reported are rise/fall values or the maximum delay. The default is for timing analysis to choose the values most appropriate for the design. If you have included clock assertions in the timing constraints file, the delays obtained for the relevant paths are also reported.

The path analysis report is divided into three main sections:

- A *general section*, including the date and time the file was created, the design name, the part used and the timing analysis and delay calculator options valid when the report was produced.
- A *clock report section*, reporting statistics on each asserted clock defined in the timing constraints file.

  The clock arrival times show the minimum and maximum times for the clock signal to all the ports controlled by the clock. Also listed are the asserted (required) period for the design and the current period obtained by the implementation. You then see a summary of how these figures are computed followed by the individual delays for each path segment and absolute and relative delays through the logic and routing for the overall path.
- *Path report sections* summarizing the required and actual arrival times for each path followed by the delays along the individual sections of the path and absolute and relative delays through the logic and routing.

Example

This truncated example shows the header section at the start of the file and a truncated longest path example, but no clock report section:

```
Path Analysis June 6, 1997 at 3:37:56 pm
Design: adder
Part: A (AT40K10-2RC)
Imported from: #('c:\figaro\examples\adder\wir\adder.1')

Timing analysis options
-----------------------
Delay Range = MaxMax
Long critical path analysis  -> Trace 10 most critical
Short critical path analysis -> Trace 10 most critical
Longest path analysis        -> Trace 10 longest paths
Shortest path analysis       -> Trace 10 shortest paths

Delay calculator options
------------------------
Mode             = MinMax
derating factor   = 1

Clock Report Section
--------------------
 Clock: CLK

   Min clock arrival time  = 3.42ns
   Max clock arrival time  = 3.42ns

   Total number of setup checks violated = 2 (out of 2)

   Asserted period = 2.0ns (500.0MHz)
   Current period  = 25.83ns (38.71MHz)

Worstcase path:

   Path #1

     Slack = -23.83ns
     Type  = Input -> Output ('ICMS_1 A' -> 'OD_1 Q')

     Clock Edge: 'CLK' on 'ICMS_2 A'          _/  0.00ns
     Asserted Reqd Time: 'OUT' on 'OD_1 Q'    \_  2.00ns
                                                 -------
     Required Arrival Time:                   \_  2.00ns
     Clock Edge: 'CLK' on 'ICMS_2 A'          _/  0.00ns
      Asserted Arr. Time: 'IN' on 'ICMS_1 A'  \_  2.00ns
      Data Path: 'ICMS_1 A' -> 'OD_1 Q'       \_ 23.83ns
                                                 -------
      Actual Arrival Time:                    \_ 25.83ns


Data Path:
        ICMS_1 A                     \_  0.00
          MACRO ICMS R 2.40 F 1.90
        ICMS_1 Q                     \_  1.90
          NET   N1   R 5.52 F 6.09
        AN2_1 B                      \_  7.99
          MACRO AN2  R 1.10 F 1.10
        AN2_1 Q                      \_  9.09
          NET   N5   R 4.26 F 4.84
        OD_1 A                       \_ 13.93
          MACRO OD   R 8.70 F 9.90
        OD_1 Q                       \_ 23.83

        logic delay = 12.90ns (54%)
        route delay = 10.93ns (46%)
Short critical path analysis
--------------------------
   Path #1

     Slack = 5.0ns
     Type  = Flop -> Flop ('FD_1 CLK' -> 'FD_2 D')

     Clock Edge:      'CLK'      on 'ICMS_2 A' _/ 0.00ns
     Clock Delay:     'ICMS_2 A' -> 'FD_2 CLK' _/ 3.42ns
                                                  ------
     Required Arrival Time:                    \_ 3.42ns

     Clock Edge:      'CLK'      on 'ICMS_2 A' _/ 0.00ns
     Clock Delay:     'ICMS_2 A' -> 'FD_1 CLK' _/ 3.42ns
     Data Path:       'FD_1 CLK' -> 'FD_2 D'   \_ 5.00ns
     Hold             'FD_2 D'                 \_ 0.00ns
                                                  ------
     Actual Arrival Time:                      \_ 8.41ns

      Data Path:
        FD_1 CLK                    _/ 0.00
          MACRO FD  R 1.80 F 2.20
        FD_1 Q                      _/ 1.80
          NET   N2  R 0.12 F 0.13
        INV_1 A                     _/ 1.92
          MACRO INV R 2.90 F 3.00
        INV_1 QN                    \_ 4.92
          NET   N3  R 0.07 F 0.07
        FD_2 D                      \_ 4.99

        logic delay = 4.80ns (96%)
        route delay = 0.19ns (4%)
```

For more detailed information on the PDL file and the computation of the values it contains, refer to the on-line help.

The format of the .PDS file makes it unsuitable for reproduction here. For full information, refer to the on-line help.

User Macro Library files: *.LIB

User macro library files have the extension .LIB and contain information on the macros you have been created and checked into this library. Its format is the same as that of design FGD files, although it also contains different information required when an instance of the macro is incorporated in a design: ports, instances and nets along with placement and routing data for each macro cell.

When you create a new macro, use *Library>Check-In Macro* or the *Check-In* button to add the macro to a library using the cell name you specify. If that cell name already exists, you can overwrite the previous definition, although you cannot create a macro using the same name as an instance in the vendor library. You can set up a number of different library files.

For a design which is to use one or more of your macros, use *Library>Library Setup* to tell Figaro where the library file is. This is then cached by Figaro, making your macros available. Macros in a user macro library will be used in preference to netlist symbols/descriptions which have the same name.

For a full description of the files and directory structure used, *Search* the on-line help for "User Macro" and jump to the topic "User Macro Files and Directories".

Orcad

The Integrated Development System (IDS) is set up to provide a seamless interface for the OrCAD Express for Windows product. The user can switch and choose easily between the OrCAD and IDS systems for design entry, circuit implementation, and eventual download of the bitstream to an Atmel FPGA. Full VHDL simulation for OrCAD is supported.

This chapter of the "CAE Interfaces" section will detail the design process using OrCAD Express and the Atmel FPGA (AT6000 or AT40K series) library. For details about the OrCAD tools, check the appropriate OrCAD User's manuals. Please refer to the "Figaro" section of this *Tutorial* for detailed instructions on running Figaro with the example design "test40K".

The Atmel library files in this release will allow the user to work with OrCAD tools and target the design to either the AT6000 or AT40K series FPGAs. The design flow allows the user to invoke Express with the Figaro *Schematic Entry* button for design entry. The design can then be netlisted and placed and routed using Figaro. Specific details on each topic will be discussed later in this chapter.

Design Flow

This section will explain in detail the Figaro flow when integrating OrCAD tools into the design process. Though there are different ways to design a circuit with OrCAD tools in conjunction with the Integrated Development System, the recommended work sequence is as follows:

System Setup(Figaro)

- Sets up the design directory and libraries

Design Entry (OrCAD Express)

- Sets up the software environment
- Enters the schematic for the design

Netlist Generation (OrCAD Express)

- Outputs EDIF netlist for Figaro

Layout Generation (Figaro)

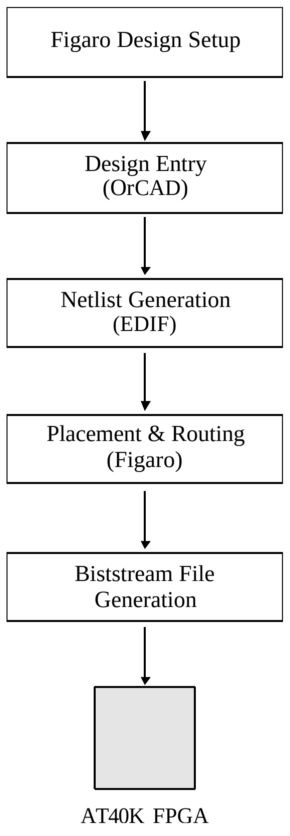
- Invokes automatic/manual placement
- Invokes automatic/manual routing
- Writes delay values for back annotation

Post-layout Simulation

Device Programming (Figaro)

- Creates bitstream output
- Downloads onto Atmel part

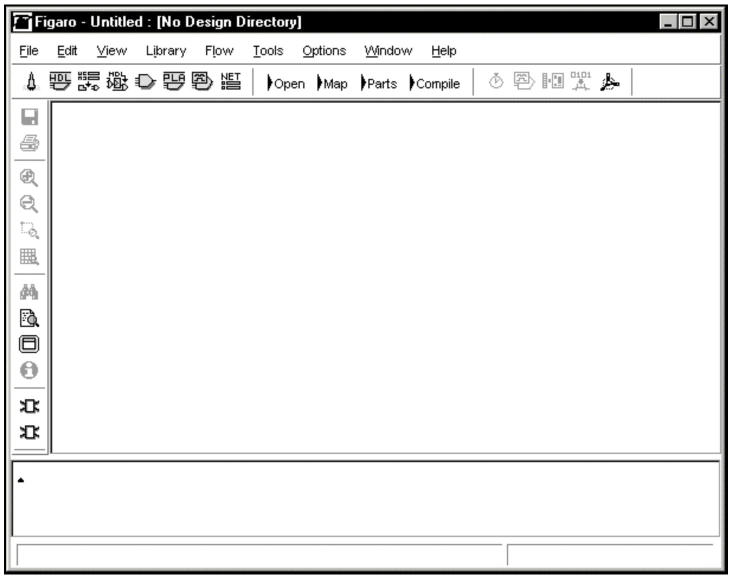
The design flow for the Atmel FPGA with OrCAD tools is graphically shown below.


Figaro Design Setup
Design Entry (OrCAD)
Netlist Generation (EDIF)
Placement & Routing (Figaro)
Biststream File Generation
AT40K FPGA

Design Flow for OrCAD Platform

System Setup

The Figaro interface is built to facilitate user interaction with the required CAE platform from within Figaro. OrCAD tools required in the design flow with an AT6000 or AT40K series FPGAs can be accessed from the buttons provided in the Figaro Desktop.



Figaro Desktop (Main Screen)

Setup Files

In order to use the Atmel FPGA (AT6000 or AT40K Series) libraries, the design template new.dsn, stored in the \SystemDesigner\lib\orcad directory must be copied to the design directory. This step is normally performed by Figaro and is only required if the file is missing on completion of the design setup process.

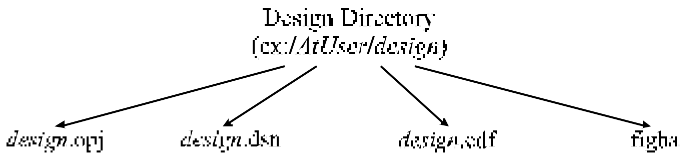
Atmel Library

The Atmel OrCAD library is stored under a directory called orcad in the \SystemDesigner\lib area. IDS will automatically set up the needed pointers to the OrCAD library as specified above. This library is named at6k.olb (for AT6000) or at40k.olb (for the AT40K).

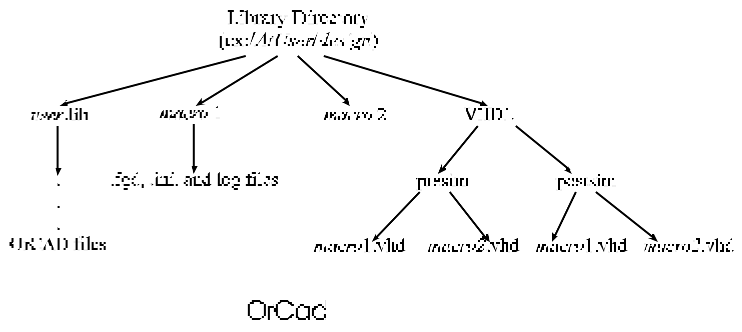
File Structure

IDS can be used to manage all files related to a design and any user libraries that may be associated with it. Most design files will be found in the design directory that is identified to Figaro. Also IDS supports the idea of a user library in which all design files associated with its components are stored in a common sub-directory.

The following two diagrams describe the file structures in the design and library directories respectively.



OrCAD Design Directory Structure

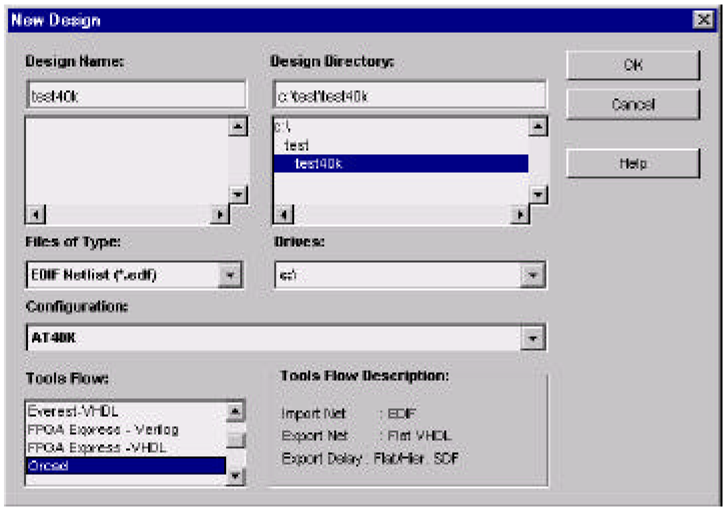


OrCAD Library Structure

Exercise

1) Copy the “test40k” exa mple from the Figaro \SystemDesigner\examples\at40k\orcad\test40k directory to the newly created design directory location, e.g. \SystemDesigner\\at*user*\test40k
2) In Figaro, select the icon or *File>Design Setup* from the main menu and click on the *New Design* button. Inside the *New Design* dialog box:
   - Set *Configuration* to *AT6K or AT40K.*
   - Choose *Orcad* from the *Tools Flow* list box.
   - Set *File of Types* as *EDIF Netlist (*.edf).*
   - Enter the absolute path of the *Design Directory* where the “test40k” example will be copied to such as “c:\at*user*\test40k”.
   - Enter “test40k” as the *Design Name*.

   An example of the dialog box with complete settings is shown below.



Design Name: test40k

Design Directory: c:\test\test40k

c:\
test
test40k

Files of Type: EDIF Netlist (*.edf)

Drives: c:\

Configuration: AT40K

Tools Flow:
Everest-VHDL
FPGA Express - Verilog
FPGA Express -VHDL
Orcad

Tools Flow Description:
Import Net : EDIF
Export Net : Flat VHDL
Export Delay : Flat/Hier. SDF

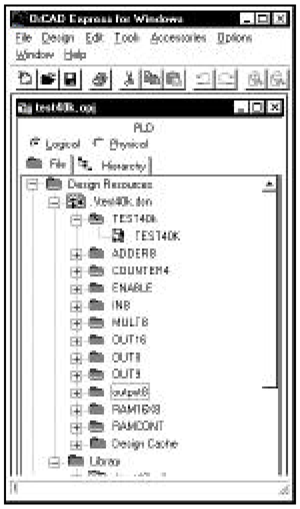
New Design Dialog Box

   - Click OK to return to the main screen.
3) The design set up is now complete.

Design Entry

OrCAD Express for Windows is the tool used for performing schematic entry. This tool can be invoked from the *Schematic Entry* button on the Figaro desktop.

Inside Express, the project file (*.opj) of the design will be created and stated as the header of the *Design Manager* window. Under the *Design Resources* folder in the *Design Manager* window will be shown the design file (test40k.dsn in this example) and the *Library* folder. All the libraries used in the design will be added to this *Library* folder, and all the schematics of the design will be in different folders under the design file.



Design Manager in OrCAD Express

Drawing Creation Guidelines

All components placed in the design sheets should be either selected from the Atmel FPGA library (AT6K or AT40K) or hierarchical blocks that have been created with Atmel FPGA library components.

It is important to attach "hierarchical ports" to all input and output pins at all levels of the design. The user should also change the name and type for each port used. This ensures proper creation of the ports and corresponding directions by the netlist generator for Figaro to perform placement and routing.

Exercise

For the purpose of this exercise, the steps involved in creating three of the hierarchical components, an output pad, an Adder and a Multiplier for the "test40k" example are shown. The output pad is a component in the logic block called out8 in test40k, and is used to demonstrate the general process for entering a schematic with Express using the Atmel FPGA library. The Adder and Multiplier, on the other hand, are used to illustrate the automatic macro generation capability of IDS. The application of "hierarchical ports" to the top level schematic is also discussed below.

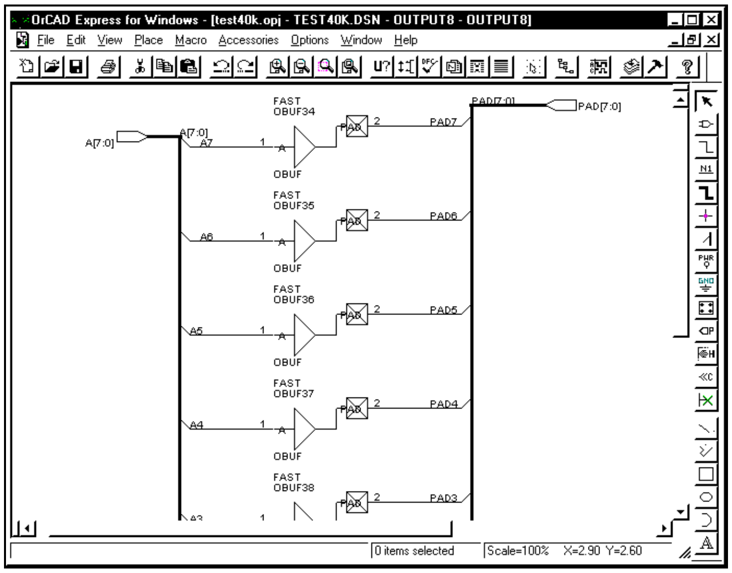
Entering an output hierarchical block

1. Follow the steps in the previous exercise to complete the design set up. The name of the design should be set to "test40k".

2. To start a new schematic:
   - Click on the *Schematic Entry* button from the Desktop to invoke Express on "test40k".
   - Select *test40k.dsn* in the *Design Manager* window. Use *Design>New Schematic*, and enter "output8" as the new schematic name.
   - Select the *output8* folder, use *Design>New Schematic Page* to set the name of new schematic page as "output8".

3. After the schematic page has been set up, output8 can be implemented by following steps:

   - Open *Schematic Page - output8* by double-clicking it, and select *Place>Part* from the menu bar in Express.
   - Select the *AT40K* library from the *Place Part* dialog box. If it is not listed, click on the *Add Library* button and select the file At40k.olb stored in the \SystemDesigner\lib\orcad directory.
   - Select and place the component OBUF in the schematic sheet. Repeat this 7 times. Meanwhile, place all the OBUFs vertically in a column as illustrated below.



The output8 Block Schematic

   - Select *Place>Bus* from the menu and use the mouse to create a bus vertically on each side of the OBUFs. Use *Place>Net Alias* to name buses as A[7:0] on the left and PAD[7:0] on the right.
   - Use *Place>Wire* to draw nets for all pins of OBUFs.
   - Select *Place>Bus Entry* to connect each net to a bus on either the left or right. Use *Place>Net Alias* to name each bus net.

NOTE

Note: *Hierarchical Ports* are required for every net of a component referenced in higher levels of hierarchy in the schematic in order to generate a correct and portable EDIF netlist. *Netlist generation* will be discussed in a later section.

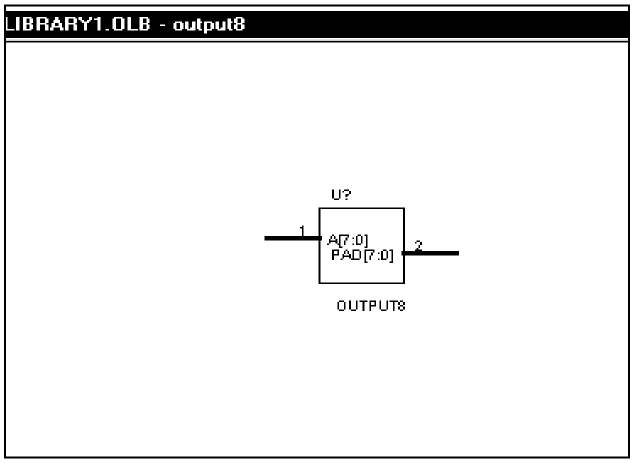
   - Use *Place>Hierarchical Ports* to create the connections to all of the interface pins (two bus pins). Apply *Edit>Properties* to change the labels on these connectors to A[7:0] and PAD[7:0] and the *Type* of pins to Input, and Output respectively. This process will place the necessary information in the EDIF netlist for Figaro to determine the pin direction.
   - Save the schematic by using *File>Save*. Switch to the *Design Manager* window. After selecting the *output8* folder, use *Tools>Annotate* (with the default settings) to update the part references and *Tools>Design Rules Check* to ensure the design has been entered correctly.
   - Finally use *Save* to save the design in the *Design Manager* Window.

4. A library has to be set up for the creation of the symbol for the schematic, output8. To set up the library and generate the symbol, follow the steps below:

- Create a new library by using *File>New>Library*.
- Pick the library file in the *Library* folder. Select *File>Save as* from the menu, and enter the file name (e.g. "*design*.olb"). Set *Files of Type* to *Capture Library*. Click OK.
- Select the *design/design*.olb folder, click on *Design>New Part*, and enter "output8" as the part *Name*.
- Select *Attach Implementation* in the *New Part Properties* window. Inside the *Attach Implementation* dialog box, select *Schematic View* as the *Type of Implementation* and specify output8 as the *Name*. Click OK in the dialog boxes that follow to invoke the symbol editor.
- Create and edit the symbol for the schematic "output8" in the symbol editor

NOTE *Type* and *Pin Number* must be specified for every pin of the component's symbol in order to generate a correct and portable EDIF netlist.

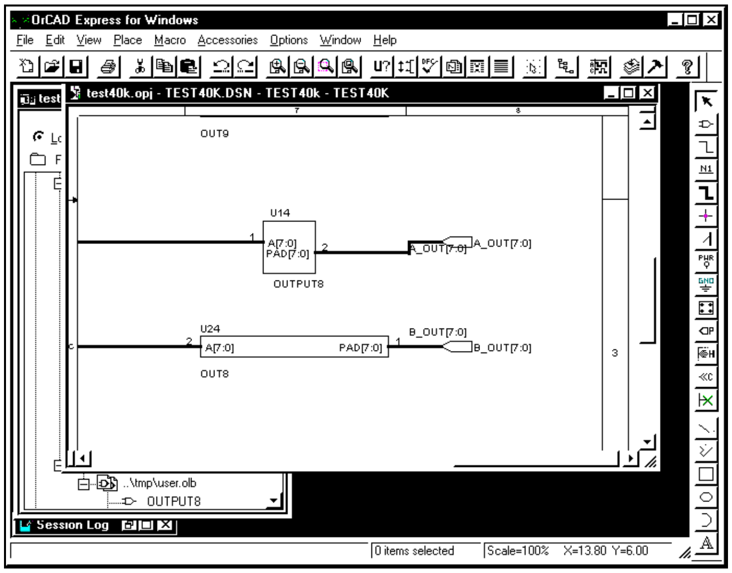
The symbol should have two pins, A[7:0] and PAD[7:0]. Specify the *Type* of pins as "Input" and "Output" respectively, and assign a number to the *Number* field of the pin properties as well. Choose *Save* to save the design. The completed symbol is shown below.



Symbol of Output8

NOTE To maintain a library that can be reused in other designs, copy the folder "dflop" from the design browser to the design library "*design*.olb" browser.

5. Open up the test40k sheet and replace one of the out8 components with the newly created output8 from *design*.olb. Select the output8 and use *Edit>Properties* to change the *Primitive* option from *Default* to *No*.


OrCAD Express for Windows
File Edit View Place Macro Accessories Options Window Help
test40k.opj - TEST40K.DSN - TEST40k - TEST40K
OUT9
U14
A[7:0]
PAD[7:0]
OUTPUT8
A_OUT[7:0]
U24
B_OUT[7:0]
OUT8
..\tmp\user.olb
OUTPUT8
Session Log
0 items selected
Scale=100% X=13.80 Y=6.00

Schematic of Output8

6. Save the schematic by using *File>Save*. Next, switch to the *Design Manager* window and use *Tools>Annotate* to update part references and *Tools>Design Rules Check* to ensure the design has been entered correctly. Finally choose *Save Design* from the *Design Manager Window*.

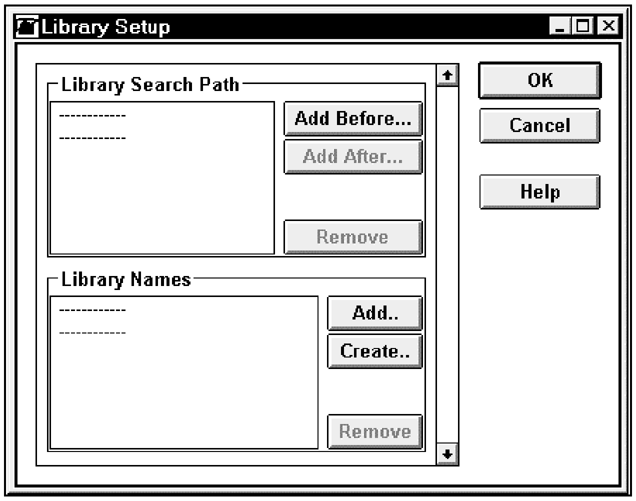
7. Quit Express.

Using the Macro Generators

In this section, the *Macro Generators* will be used to replace a portion of the circuit with functions that have been optimized for the Atmel architecture. Before using the *Macro Generators*, a user library must be first created. The user library is composed of two parts. The first is a library file that stores the layouts created by:
a) the *Macro Generators*, or b) the user converting parts of the design to a hard layout. The second is a directory holding all the files used to create the layout. These include various layout-related files as well as netlists that may be used for simulation.

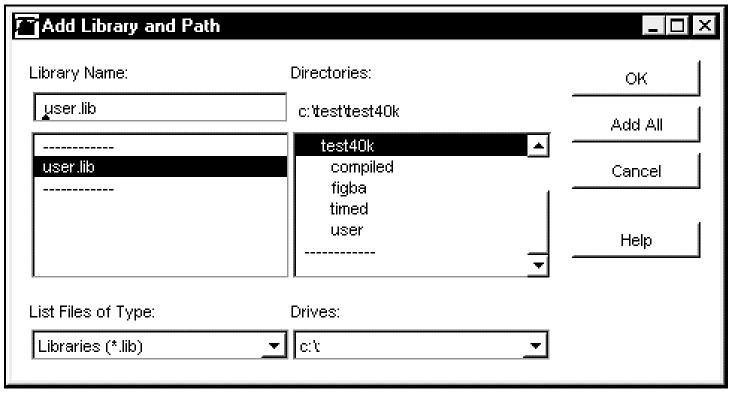
To create a library, follow the procedures below.

- Select *Library>Library Setup* from the Figaro menu. The *Library Setup* dialog box will come up as follows.



Library Setup Dialog Box

- From the *Library Setup* dialog box, select *Add Before* and bring up the *Add Library* dialog box. Fill it in to specify the location and name of the user library. To simplify design archival and file maintenance, a design specific user library should be stored in the design directory. The following is a sample dialog box for this exercise where the *Library Name* is set to "*user*.lib" and *Directories* set to "\SystemDesigner\at*user*\test40k".
- Click OK to finish the library set up.

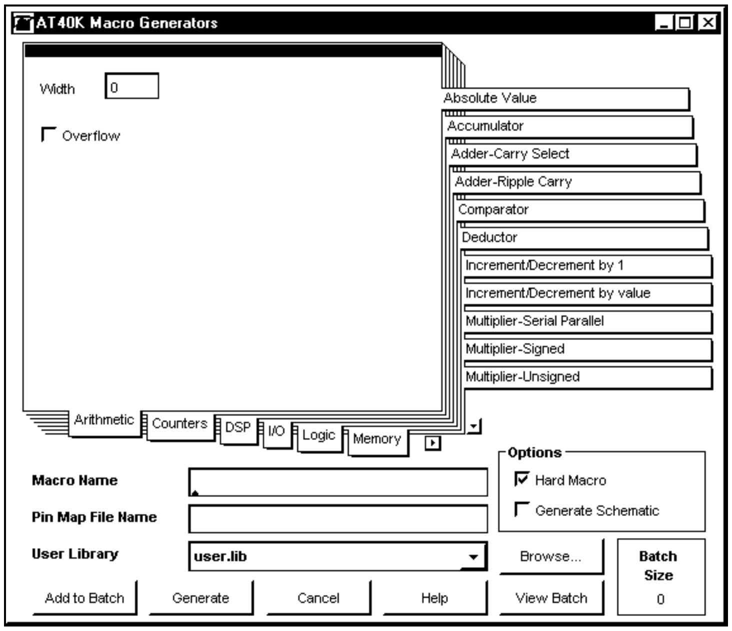


Library Dialog Box

Macro Generation

Two macros, mult8new and add8new, are going to be generated in this section with the use of *Macro Generators*. They will be used to replace the original multiplier, MULT8, and adder, ADDER8, in the test40k design.

Once the library has been set up, press the *Macro Generators* icon on the Flowbar to bring up the *Generators* interface. The notebook dialog box is shown below.



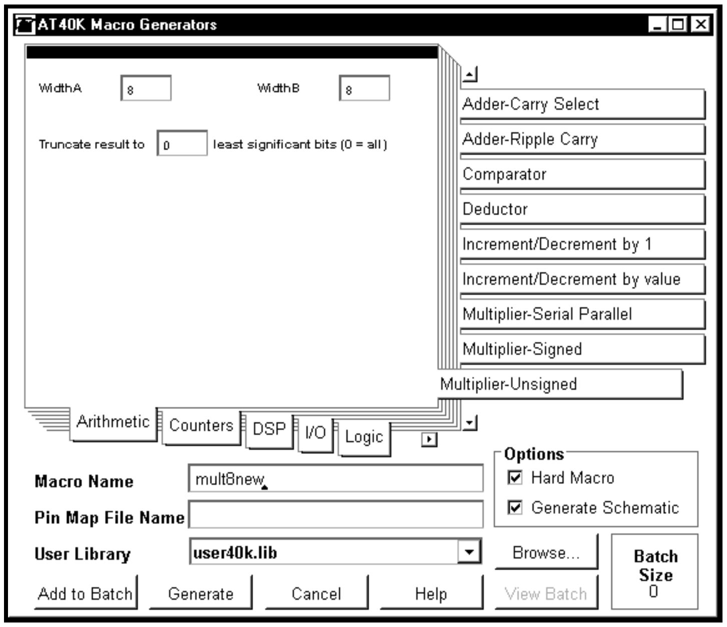
Macro Generators Dialog Box

To generate the mult8new macro,

1. Under the *Arithmetic* category, select the *Multiplier-Unsigned* tab on the notebook.

2 Enter “mult8new” as the *Macro Name*.

3 Enter “8” for both *WidthA* and *WidthB*. A sample of the completed dialog box is shown below.


AT40K Macro Generators

WidthA 8 WidthB 8

Truncate result to 0 least significant bits (0 = all)

Macro Name: mult8new

Pin Map File Name:

User Library: user40k.lib

Options: ☑ Hard Macro ☑ Generate Schematic

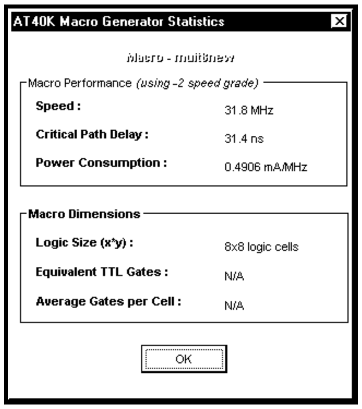
Batch Size 0

Complete setup for mult8new

For details about the macro’s properties, press the ADD TO BATCH button then the VIEW BATCH button. This will provide information on the *Generator* parameters, pin names and functions, as well as either a description of the function or a truth table.

4. Press GENERATE to initiate the process.

The *Macro Generators* will then create a layout, an EDIF netlist, and an OrCAD symbol. Upon completion, a dialog box will be displayed with statistics about the macro that was created. These statistics are stored in a file in the library as *libName*.sts.


AT40K Macro Generator Statistics

Macro - mult8new

Macro Performance *(using -2 speed grade)*

Speed :	31.8 MHz
Critical Path Delay :	31.4 ns
Power Consumption :	0.4906 mA/MHz

Macro Dimensions

Logic Size (x*y) :	8x8 logic cells
Equivalent TTL Gates :	N/A
Average Gates per Cell :	N/A

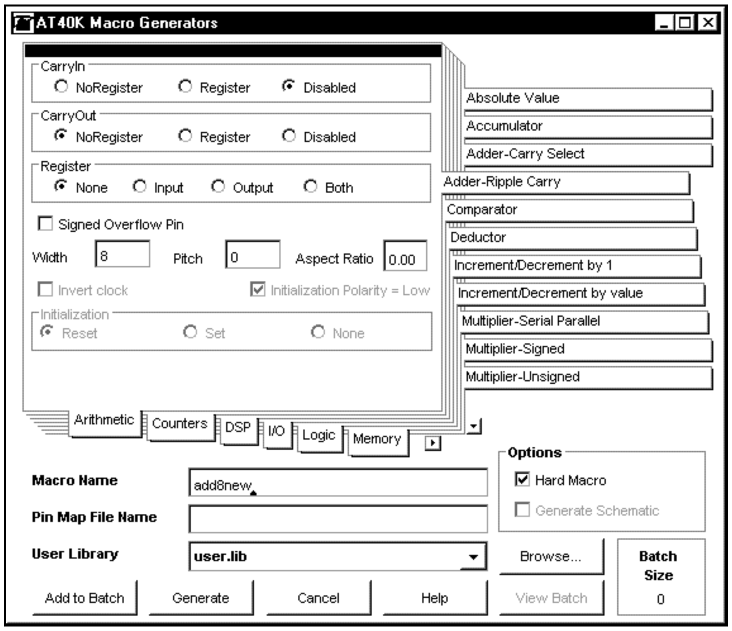
Mult8new Statistics Box

Create the 8-bit adder macro, add8new, in a similar manner.

1. Under the same *Arithmetic* category, select the *Adder-Ripple Carry* tab on the notebook.
2. Enter “add8new” as the *Macro Name*.

3. Select *Disabled* for the *CarryIn* option.
4. Enter “8” for *Width*.

NOTE These statistics only appear after a single macro has been generated. If an entire batch has been generated, statistical information will not be provided.



AT40K Macro Generators

CarryIn: ( ) NoRegister ( ) Register [✓ Disabled]

CarryOut: [✓ NoRegister] ( ) Register ( ) Disabled

Register: [✓ None] ( ) Input ( ) Output ( ) Both

☐ Signed Overflow Pin

Width: 8 Pitch: 0 Aspect Ratio: 0.00

☐ Invert clock ☑ Initialization Polarity = Low

Initialization: [✓ Reset] ( ) Set ( ) None

Macro Name: add8new

Pin Map File Name:

User Library: user.lib

Options: ☑ Hard Macro ☐ Generate Schematic

Batch Size: 0

Complete setup for add8new

5. Click GENERATE in the *Macro Generators* dialog box to initiate the generation process.

6. After both macros have been created, click CANCEL to exit the *Macro Generators* notebook dialog box.

Entering Top-level Schematics

The “test40k” schematic is the top-level schematic in this example. In this part of the exercise, the macros created in the previous section will be used to replace the multiplier and adder that are currently in the schematic. Finally, a netlist containing the design test40k will be generated for placement and routing in Figaro.

Setting up the Design

1. From Figaro use the icon or select *File>Design Setup* from the pull down menu.
2. Select *test40k* as the *Design Name*. Click on OK to return to the main screen.
3. Complete the library set up as outlined above. Check that the *user*.lib created for the *Macro Generators* is specified correctly.
4. Click on the *Schematic Entry* button from the Figaro Desktop. Express will be invoked on the design test40k.

Replacing macros in the Schematic

The multiplier and adder macros, mult8new and add8new, can be placed into the schematic with the following steps.

1. Add the library file “\SystemDesigner*user\user.olb* to the *Library* folder of the design *test40k.* It can be done by selecting the *Library* folder in the *Design Manager* window, clicking on the right button of the mouse and choosing *user*.olb inside the library directory, *user*.
2. Select mult8new from *user*.olb and place it in the schematic.
3. Modify the symbol mult8new according to the section, Modifying Symbols, below.

NOTE To maintain a library that can be reused in other designs, modify the template of the symbols in the library file, e.g. *user.olb* in this example, and instantiate them in the schematics, instead of modifying the same ones after they have been instantiated.

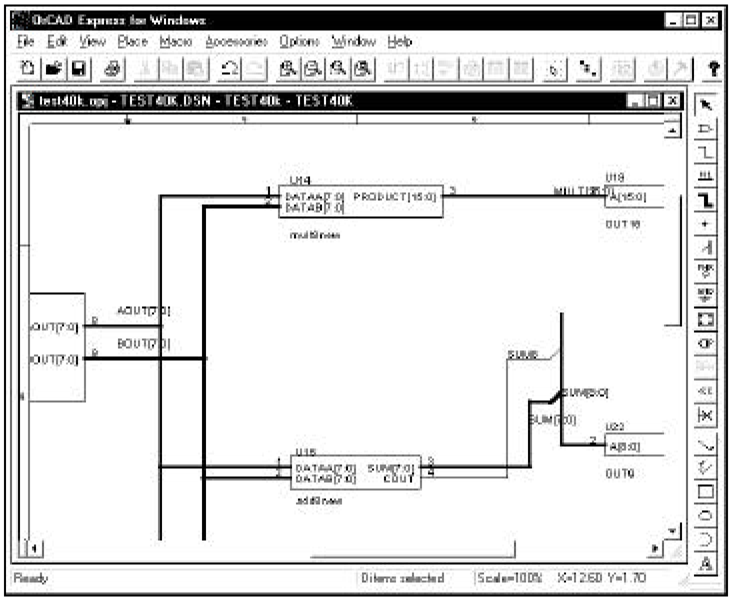
4. Connect the pins of mult8new to the same buses that were hooked up to mult8.
5. Delete mult8.
6. Repeat steps 2 to 5 to replace the 8-bit adder with add8new.

NOTE Since the symbols for the macros are automatically generated according to OrCAD configurations, some modification may be needed for symbols. The symbol can be edited to provide information needed for creating a netlist, or provide correct connectivity for the replacement macros.

Modifying Symbols

To modify the symbols, follow the steps described below:

1. Select the symbol after being instantiated in the schematic and use *Edit>Part*, or double-click on the symbol in the library file to bring up the symbol Editor.
2. Specify the name of the symbol by double-clicking on *{value}* in the schematic and enter the *Value* field.
3. Number each pin of the symbol by selecting the pin, using *Edit>Properties* and entering the number in the *Number* field.
4. Close the symbol Editor and choose *Update Current* in the *Save Part Instance* dialog box. This section of the schematic should look like the following when completed.



Using mult8new and add8new in Schematic

7. Save the schematic by using *File>Save*. Next, switch to the *Design Manager* window, use *Tools>Annotate* to update part references and *Tools>Design Rules Check* to ensure the design has been entered correctly.
8. Choose *Save Design* from the *Design Manager* Window.

Netlist Generation

An EDIF netlist file is needed as input prior to Placement and Routing. This file must be generated from within the OrCAD Express for Windows tool.

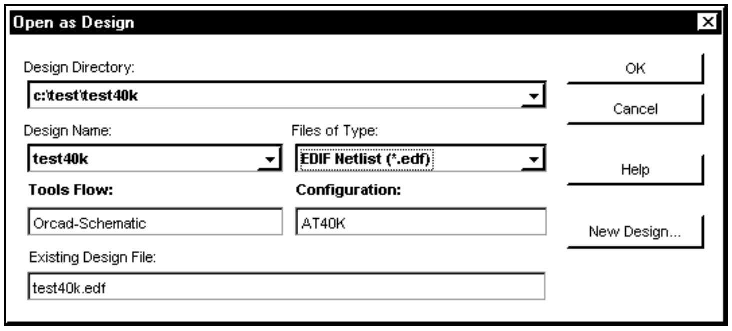
To generate a netlist, test40k.edf, from the OrCAD schematic tool, do the following steps:

1. Switch to the *Design Manager* window and select *test40k.dsn*.
2. Select *Tools>Create Netlist*. A dialog box should be invoked allowing the creation of an EDIF 2.00 netlist. Check the following items in the *Options* field: *Output pin names (instead of pin numbers)* and *Output Part Properties*.
3. Change the *Output File Extension* from "*.edn*" to "*.edf*". Click OK.
4. Use *File>Exit* to quit.

The design "test40k" is now ready to be imported into Figaro for compilation.

Figaro Place & Route

An EDIF netlist produced by OrCAD can be read into Figaro via the *Open as Design* pop-up box. Select the *Open* button or *File>Open as Design* from the menu. Confirm the design directory and design name and specify the *Files of Type* as *EDIF Netlist (*.edf)*. Verify the input file name as *test40k.edf* and click on DESIGN to read in the design.


Open as Design

Design Directory: c:\test\test40k

Design Name: test40k

Files of Type: EDIF Netlist (*.edf)

Tools Flow: Orcad-Schematic

Configuration: AT40K

Existing Design File: test40k.edf

Open as Design Dialog Box

Press the *Compile* button to complete the design from placement and routing to bitstream generation. For step by step instructions on how to select the Atmel part and implement a design in Figaro, please refer to the "Figaro" section of the *IDS Tutorial*.

Post Layout Simulation

After the design has been implemented with the *Compile* function, *Post-layout Simulation* can be performed. OrCAD simulation tools can be invoked with the provided post-layout wire delays, including pin-to-pin delays, setup and hold times, as well as actual wire delays for evaluating the device timing and performance.

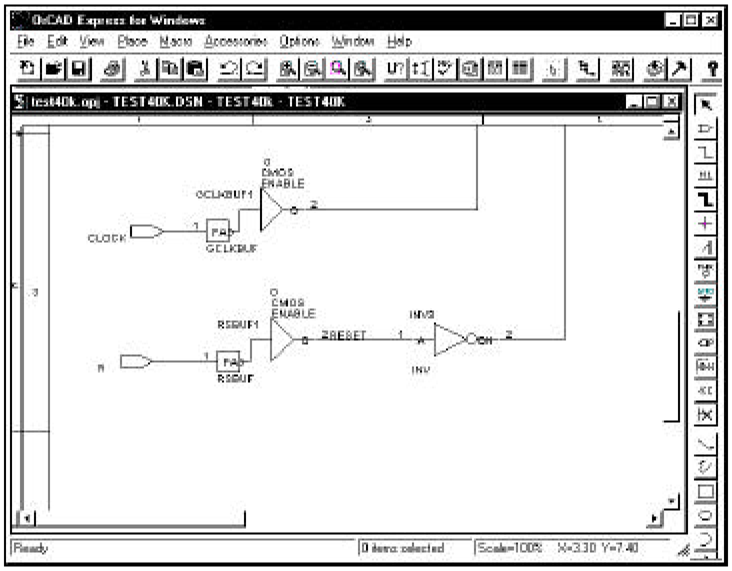
Please refer to the OrCAD tutorial for more details on *Simulation* of this design.

Engineering Change Order (ECO)

Figaro's ECO feature supports revisions to the schematic, while recompiling only the modified areas of the chip. The process allows repeated design changes to be compiled quickly and efficiently, while ensuring minimal changes to the timing of the circuit.

For this exercise, a minor change will be made to the "test40k" schematic using ECO. In this case, the polarity of the RESET signal will be reversed.

1. Bring up the "test40k" schematic by pressing on the *Schematic Entry* button. Once in Express, add an inverter to the output of the RSBUF (which is bringing in the RESET signal) as shown below.

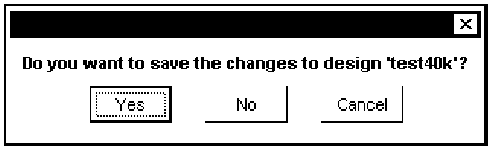

OrCAD Express for Windows
File Edit View Place Macro Accessories Options Window Help
test40k.opj - TEST40K.DSN - TEST40k - TEST40K
CLOCK
GCLKBUF1
GCLKBUF
CMOS ENABLE
RSBUF1
RSBUF
R
RESET
INV3
INV
Ready
0 items selected
Scale=100% X=3.30 Y=7.40

Modified Schematic with Inverter

2. Proceed with *Tool>Annotate* and *Tool>Design Rules Check*. Save the schematic.

3. When completed, write out the EDIF netlist to test40k.edf, the file previously generated. Save the design and then exit Express.

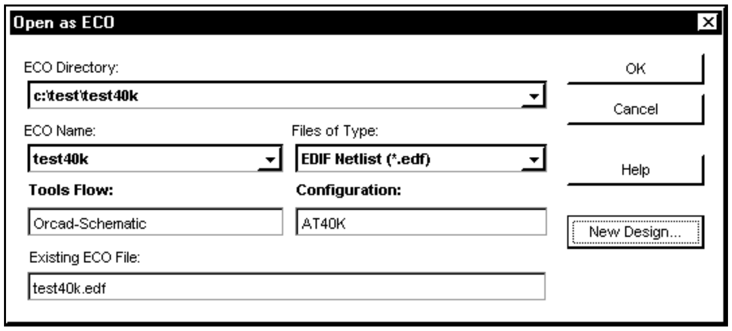
4. To incorporate the change into Figaro use the *File>Open as ECO* option. The existing layout must already be loaded into Figaro. It will then request to save the design as follows.



Do you want to save the changes to design 'test40k'?

ECO Dialog Box

5. Answey Yes and bring up the *Open as ECO* dialog box. Change *Files of Type* to *EDIF (*.edf)* and ensure that *test40k.edf* is specified as the input file name.



Open as ECO

ECO Directory: c:\test\test40k

ECO Name: test40k

Files of Type: EDIF Netlist (*.edf)

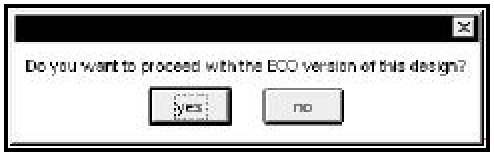
Tools Flow: Orcad-Schematic

Configuration: AT40K

Existing ECO File: test40k.edf

ECO Dialog Box

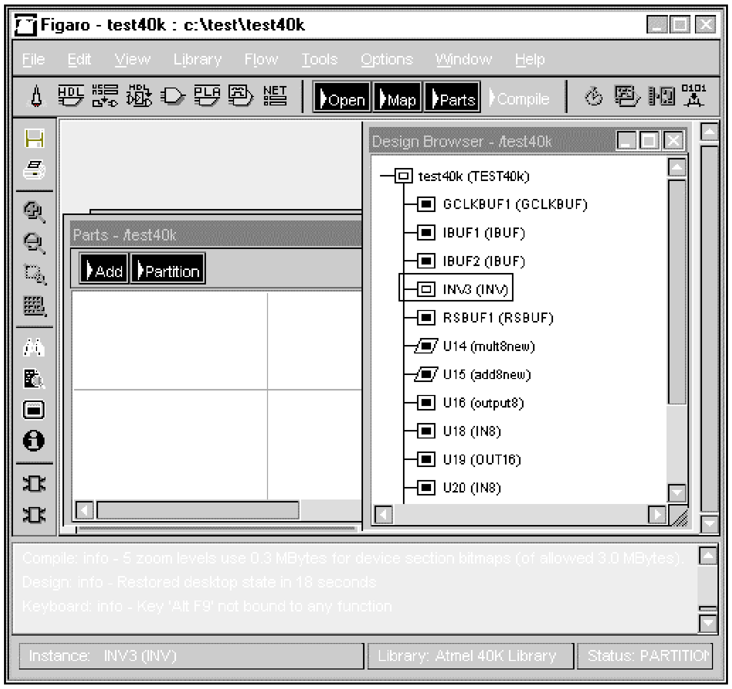
6. Press OK to initiate ECO. During this process, Figaro will report all differences and then request confirmation to proceed.



ECO Pop Up Box

7. Press Yes to execute changes to the design.

   The *Design Browser* below shows that with the exception of the new inverter, which is inside a highlighted box, all original instances are now locked and identified by a solid square preceding their names.



```
Compile: info - 5 zoom levels use 0.3 MBytes for device section bitmaps (of allowed 3.0 MBytes).
Design: info - Restored desktop state in 18 seconds
Keyboard: info - Key 'Alt F9' not bound to any function
```

Changed Instance in Figaro Browser

8. Proceed with *Compile* after this point and the program will place and route the inverter while leaving everything else locked down.

NOTE In some cases placement contention that cannot be resolved automatically, can occur after ECO due to the locking of the unchanged macros in the original design. Use Manual placement to resolve any placement contentions left on the device if the Compile step fails after ECO

Bitstream Generation

Besides performing placement and routing, the *Compile* button will also generate a bitstream file. The file can be found in the design directory and is named test40k.bst.

Viewlogic

Figaro is set up to provide a seamless interface for Viewlogic Workview Office. The user can move easily between the Viewlogic and Figaro tools for design entry, functional verification, circuit implementation, post-layout simulation, and the eventual download of the bitstream to an Atmel device.

This tutorial will step the user through the various phases of entering a design using the Integrated Development System with Viewlogic Workview Office. System setup requirements should be met before working on the tutorial example.

This chapter is a supplement to the Viewlogic documentation. For information on tool functions, check the respective Viewlogic manuals. Please refer to the “Figaro” section of the *AT40K IDS Tutorial* for detailed instructions on running Figaro with the example design “test40k”.

System Setup

It is important that the Design and Library files, with their respective root paths, be in place prior to opening a design. The correct setup of the environment variables is also needed for proper execution of the programs.

Environment Variables

For IDS, there are 2 environment variables which must be set in order for the software to function properly. Also, the programs must be specified in the user’s path. As an example, the following must be set.

```
ATMELDIR = \SystemDesigner\etc
FIGARO_HOME = \SystemDesigner
PATH = \SystemDesigner\bin;....
```

For Workview Office, the environment variables WDIR and LM_LICENSE_FILE must be set for the Viewlogic software to work properly. The path must point to the location where the Viewlogic software is installed, as shown in the examples below.

```
PATH = \SystemDesigner\wvoffice;...
```

WDIR This variable is used by the Viewlogic tools to find important files, such as the license and message files. This variable must be set before invoking Windows. By default, the installation program will set WDIR to point to the directory where Viewlogic was installed, typically this would be:

```
\SystemDesigner\wvoffice\standard
```

The current setting of this variable can be retrieved by typing “set” at the DOS prompt. If it is required to have multiple paths specified in the WDIR variable, a “;” should be used to separate the entries such as:

```
set WDIR=\SystemDesigner\wvoffice\standard;d:\wvoffice\standard
```

LM_LICENSE_FILE This variable is used by the Viewlogic tools to point to the license file. This variable must be set before invoking Windows. By default, the installation program will set LM_LICENSE_FILE to point to the license.dat file in the WDIR directory:

```
\SystemDesigner\wvoffice\standard\license.dat
```

Setup Files

There are two important setup files that Viewlogic uses to access design information. These are the *designName*.vpj and viewdraw.ini files. These files are typically maintained by the IDS software. If the user prefers to maintain these files using the Viewlogic tools, use *Options>Options* and clear the *Figaro Controlled* option under *Viewlogic Import*.

***designName*.vpj** The *designName*.vpj file contains the locations of all design and library paths required to work on the current design (which is analogous to “project” in Viewlogic terminology). This file is maintained by IDS, and updates by the Viewlogic project management utilities, though possible, are unnecessary.

The *designName*.vpj file can be found in the current design directory (analogous to “project directory” in Viewlogic terminology). Upon setup of a new design, a new *designName*.vpj file will be created. The directory section will be updated to contain the primary directory, any user libraries, and then the Atmel libraries. Any other directory settings that were in the initial template will be placed at the end of this file. Any duplicate aliases will be deleted.

viewdraw.ini The viewdraw.ini file contains the locations of all design and library paths required for the current design (which is analogous to “project” in Viewlogic terminology). This file is also maintained by IDS and updating is unnecessary.

The viewdraw.ini file will be located in the current design directory (analogous to “project directory” in Viewlogic terminology). Upon setup of a new design, a new viewdraw.ini file will be created. The initial template will be taken from either the current design directory, or the WDIR, whichever comes first. The directory section (found at the end of the viewdraw.ini file) will be updated to contain the primary directory, any user libraries, and then the Atmel libraries. Any other directory settings that were in the initial template will be placed at the end of this file. Any duplicate aliases will be deleted. The following is an example of this section of the viewdraw.ini file.

Example:

DIR [p] \SystemDesigner*AtUser*\test40k

DIR [r] \SystemDesigner*AtUser*\test40k*user* (user)

DIR [m] \SystemDesigner\lib\at40k (at40k)

Atmel Library

In order to use Viewlogic schematic capture and simulation tools for the Atmel FPGA, a pointer to the Atmel library must be set up in the viewdraw.ini file. This is automatically done when a new design is specified to IDS. This library can be found in the IDS installation directory under the sub-directories lib\at40k. The library files are in the Viewlogic mega file format for ease of management.

This library is where all schematic symbols are found. The alias /at40k should be used when referring to this library in Viewlogic. This is specified by the following entry in the viewdraw.ini file:

DIR [m] \SystemDesigner\lib\macro (at6k) or \SystemDesigner\lib\at40k (at40k)

Example Design

The first step in preparing for the tutorial is to copy the example design from its installation directory to a suitable location.

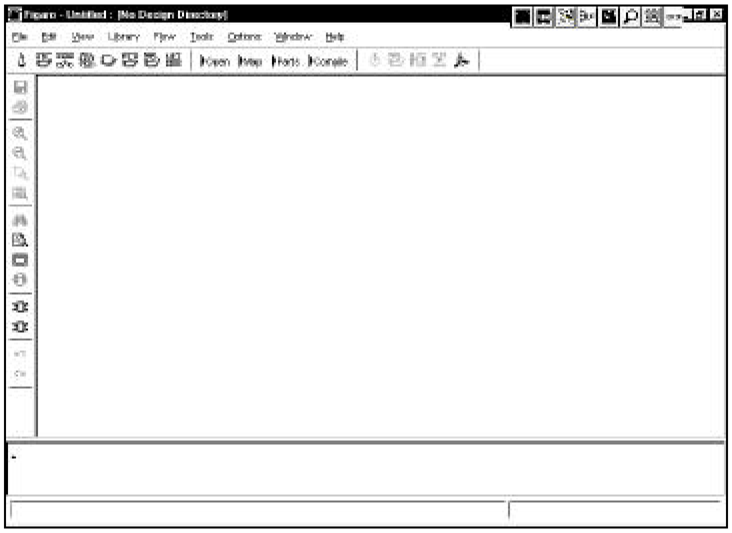
Copying the Example

The example design can be found in the IDS installation directory under the sub-directory examples\vlogic\test40k.

Design Setup

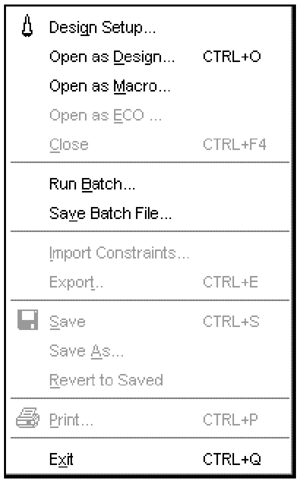
Once everything has been set up, the Figaro program can be invoked. Select START > ATMEL > IDS

If everything has been set up correctly, the Figaro Desktop should be shown as follows.


Figaro - Untitled : [No Design Directory]
File Edit View Library Flow Tools Options Window Help
Open Map Parts Compile

Figaro Desktop

The next step in the tutorial is to set up the design directory for Figaro to archive and retrieve all the files related to the design. The *Design Setup* dialog box can be brought up by clicking on the icon on the Flowbar or under the *File* menu as follows:



File Menu Dialog Box

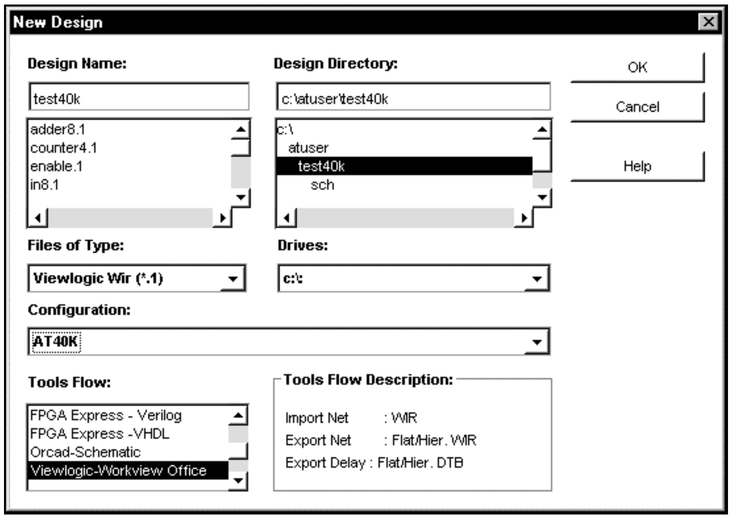
Choose the NEW DESIGN button and bring up the *New Design* dialog box.

In the *Design Directory* input box, enter the directory/path of the “test40k” example (\SystemDesigner*atuser*\test40k). Enter “test40k” as the *Design Name*. The *Files of Type* list box should show the *Viewlogic WIR (*.1)* selection. All the Wir files created for “test40k” will be displayed under the *Design Name* pane. If this is a new design, no files will be shown.

The *Tools Flow* list box will show which CAE system is being used. The *Tools Flow* choice for this tutorial should be Viewlogic-Workview Office. *Tools Flow Description* shows what files are used as input and what types of output will be available from Figaro for back annotation to the simulator.

Once the information has been provided, click on the OK button.

This will bring back the *Design Directory Setup* dialog box to show the following:



Design Name: test40k

adder8.1
counter4.1
enable.1
in8.1

Design Directory: c:\atuser\test40k

c:\
atuser
test40k
sch

Files of Type: Viewlogic Wir (*.1)

Drives: c:\

Configuration: AT40K

Tools Flow:

FPGA Express - Verilog
FPGA Express -VHDL
Orcad-Schematic
Viewlogic-Workview Office

Tools Flow Description:

Import Net	: WIR
Export Net	: Flat/Hier. WIR
Export Delay	: Flat/Hier. DTB

Design Directory Setup Dialog Box

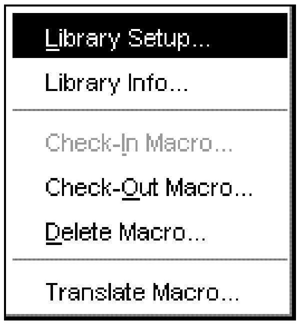
Select OK to execute.

The sch/sym/wir directories in Viewlogic contain the "test40k" schematic, symbol, and netlist files respectively. The viewdraw.ini, viewsim.ini, test40k.var and test40k.vpj files will be set up automatically, accompanied by other files used for running various parts of IDS.

Library Setup

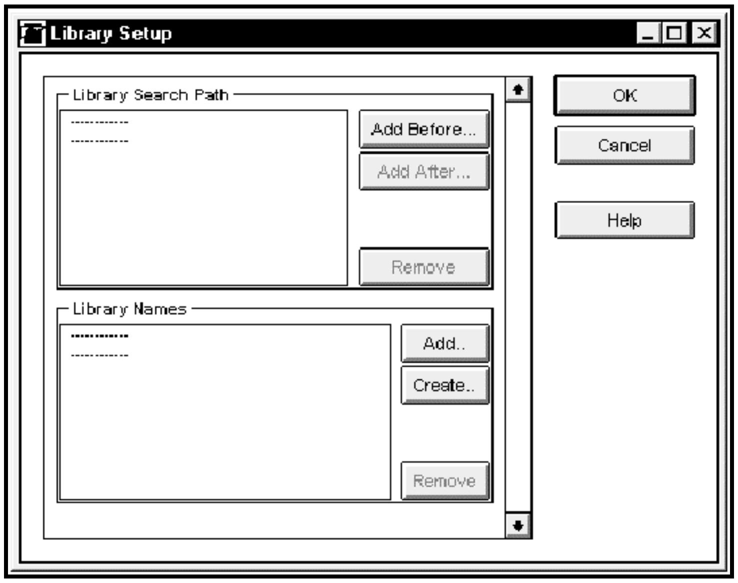
The next item to be set up is the user library. See "System Basics, User Libraries" of the *User's Guide* for a detailed explanation on the topic. A user library consists of a directory and a file where all the data about user-defined macros is stored. These macros can be created with either the schematic entry tool or *Macro Generators* function in the Figaro Desktop. Both subjects will be covered in this tutorial. To facilitate file management and design archival, it is advisable to keep the library in the same directory as the design.

To create a library, the *Library Setup* option must be selected from the *Library* menu.



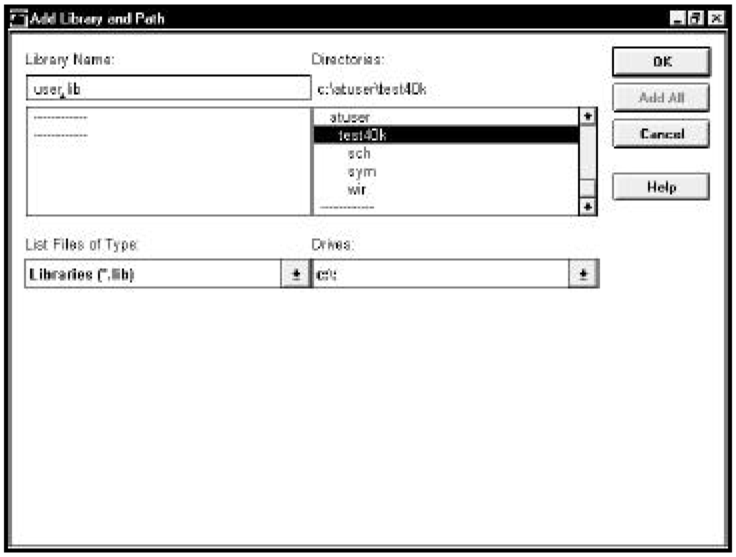
Library Setup Pull Down Menu

Once selected, the *Library Setup* dialog box is displayed.



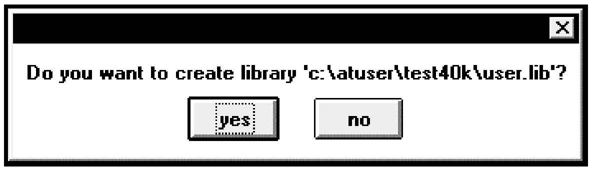
Library Setup Dialog Box

Select the *Add Before* button in the “Library Search Path” pane to specify the path of the library. Enter information in the *Add Library and Path* dialog box by using the path browser to point to the \SystemDesigner*atuser*\test40k directory and set the *Library Name* field to *user*.lib. Press OK.



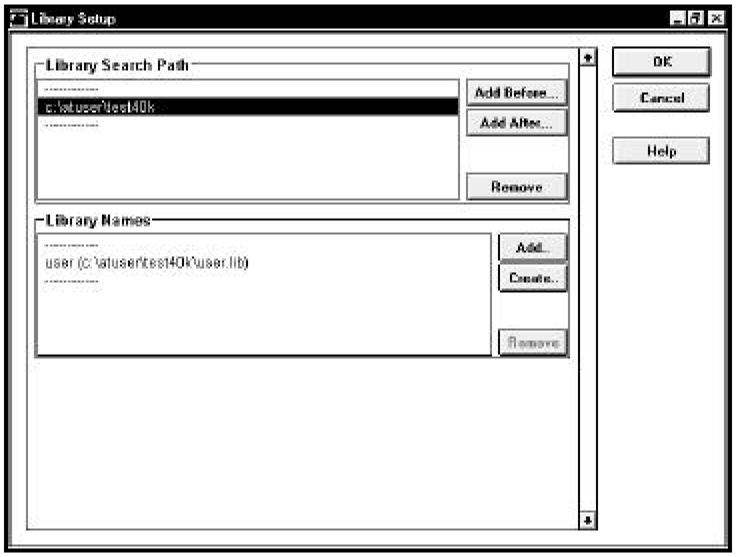
Add Library and Path Dialog Box

At this point, a dialog box should pop up to confirm the creation of the library. Press YES to continue.


Do you want to create library 'c:\atuser\test40k\user.lib'?

Create Library Confirmation Box

The *Library Setup* dialog box is show below.

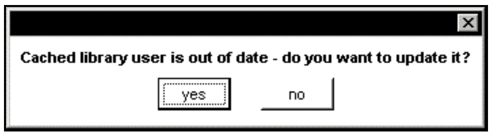


c:\atuser\test40k

user (c:\atuser\test40k\user.lib)

Library Setup Dialog Box

Press OK to have the library structure created. At this point, another dialog box will pop up to confirm updating of the cached library. Press YES to continue.

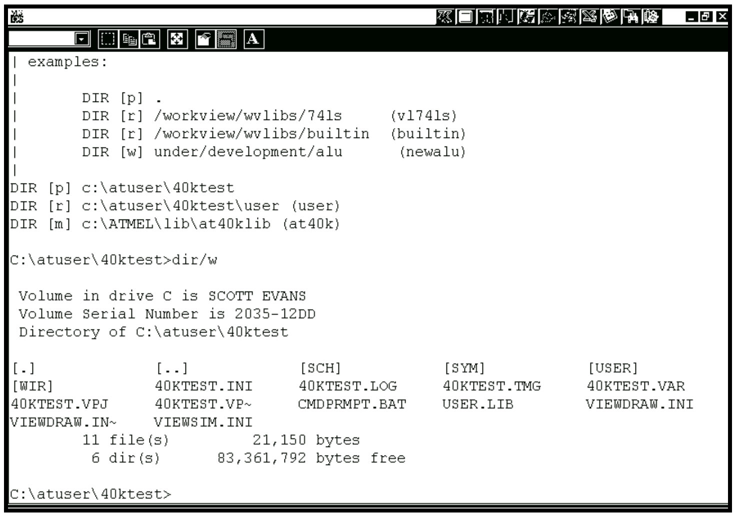


Cached library user is out of date - do you want to update it?

Update Cached LibraryDialog Box

The system should have created a file called *user*.lib in the design directory as well as a sub-directory called *user*. The file *user*.lib is used to store the hard layouts of any macros created. The directory *user* will contain the sch, sym, and wir files for each macro in the library. It will also contain a sub-directory for each macro where important design data is stored for future modifications of the component.

Finally, the viewdraw.ini and the test40k.vpj files will be updated to point to this new library. The Integrated Development System controls the read/write access of all files in the library. The library is assigned a read only attribute [r], as it is unnecessary for the user to update entries in the library.



```
| examples:
|
|       DIR [p] .
|       DIR [r] /workview/wvlibs/74ls     (vl74ls)
|       DIR [r] /workview/wvlibs/builtin  (builtin)
|       DIR [w] under/development/alu      (newalu)
|
DIR [p] c:\atuser\40ktest
DIR [r] c:\atuser\40ktest\user (user)
DIR [m] c:\ATMEL\lib\at40klib (at40k)

C:\atuser\40ktest>dir/w

 Volume in drive C is SCOTT EVANS
 Volume Serial Number is 2035-12DD
 Directory of C:\atuser\40ktest

[.]             [..]            [SCH]           [SYM]           [USER]
[WIR]           40KTEST.INI     40KTEST.LOG     40KTEST.TMG     40KTEST.VAR
40KTEST.VPJ     40KTEST.VP~     CMDPRMPT.BAT    USER.LIB        VIEWDRAW.INI
VIEWDRAW.IN~    VIEWSIM.INI
        11 file(s)         21,150 bytes
         6 dir(s)      83,361,792 bytes free

C:\atuser\40ktest>
```

Viewdraw.ini and Directory after System Setup

Schematic Entry

Figaro works with a number of Viewlogic CAE systems and allows the user to launch into Viewlogic schematic capture and simulation tools. This *Tutorial* is based on the Viewdraw tool from Workview Office. The following section briefly introduces some of the major concepts of using Viewlogic for schematic entry. It covers design hierarchy, file structure, commonly used commands, and illustrates use of the Viewlogic tools by making some modifications to “test40k”.

For best results, it is recommended that the user first go through the Viewlogic *Workview Office Viewdraw* tutorial.

Viewdraw Schematics and Hierarchy

Schematic hierarchy in Viewlogic has the following attributes:

- Every schematic may use several symbols.
- Every symbol may contain a schematic.
- Each schematic level may contain 1 or more sheets.

It is important to plan before starting the design. The way the hierarchy is constructed will have an effect on the hard macros and layout.

Actual DOS File Structure

The file structures for Figaro and Viewlogic are shown in the following table:

Directory	**File Structure**
AT40K Project Directory	\SystemDesigner\AtUser\Project
Symbol Directory	\SystemDesigner\AtUser\Project\Sym
Schematic Directory	\SystemDesigner\AtUser\Project\Sch
Netlist Directory	\SystemDesigner\AtUser\Project\Wir

Figaro File Structures

- Every Symbol and Schematic is stored in the sub-directories Sym and Sch, respectively.
- After every writing of the schematic sheet, that single sheet is checked and an associated wire file (or netlist) is created in the Wir sub-directory.

Viewdraw Commands

The following table contains a list of the most commonly used Viewlogic Viewdraw commands.

Workview Office Commands

Command	Keybd Entry	Function
View Pan	<F6>	Center the view to the cursor.
View In	<F7>	Zoom in one view level.
View Out	<F8>	Zoom out one view level.
View Area Select	<F9>	Zoom in on area as selected.
View Full Out	<F4>	Zoom out to entire sheet.

Add Component	c	Add the component named *name.*
Select Component	Edit>Select> Component	Select component(s) named *name.*
Replace Component	Edit>Replace> Component	Change to named component.
Draw Net	n	Enter a wire (net) drawing mode.
Draw Bus	b	Enter a bus drawing mode.
Copy	^C	Copy currently selected component.
Move		Drag the selected component.
Delete Object	<Del Key>	Remove, erase, delete se-lected item.
Label	Double Click	Label the “clicked on” component.
List Labels	Project>Block> Labels	List labels.
Text	T	Enter random text creation mode.
Attribute	Double Click	Enter Attribute addition mode.
Attributes Visible	Project>Settings >Attributes Dis-play Attributes	Check to make Attributes Visible.
Attributes Invisible	Project>Settings >Attributes Dis-play Attributes	UnCheck to make Attributes Invisible.
Change String	Double Click	Enter String (label,text,attr) mode.
Select Label	Edit>Select>Lab el	Find and select a label(s).
Select Group	Edit>Select> Expression	Enter group selection mode.
Make Global Label	Double Click	Make a net accessible through hierarchy by changing scope to global.

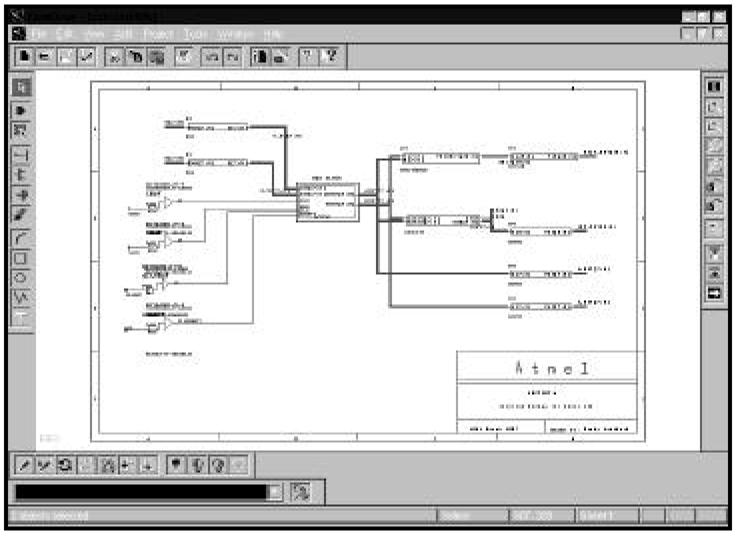
Enter Symbol's Sch	Push Sch Button	Push into the symbol's schematic.
Enter Symbol Edit	Push Sym Button	Push into the symbol for editing.
Enter the next sheet	Next Sheet Button	Push into next schematic sheet.
Return to level	Close Window	Pop back to symbol or schematic level.
Enter any Sch	File>Open	Enter a schematic by name.
Enter any Sym	File>Open	Enter a symbol by name.
Go back to last action	^Z	Undo the last action.
Save or Write file	w	Save+Check the schematic.
Quit Viewdraw	File>Exit	Exit Viewdraw.
Symbol Commands		
Change Sheet Size	Double Click Background	Change avoidance area (symbol) using sheet size in dialog.
Draw Box	B	Enter a box drawing mode (symbol).
Add Symbol Pin	p	Enter a symbol pin drawing mode (symbol).
Special Commands		
Information	Help>Viewdraw Help Topics	Bring up on-line help. Selecting a menu item in Help mode will yield information.
Mouse - LEFT: Select	MIDDLE:	RIGHT: Short Cut Menu

Exercise

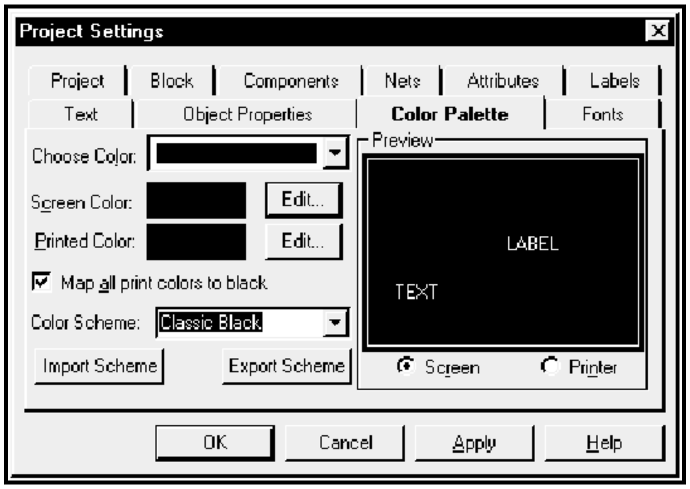
This part of the chapter will show the user how to create both a block of hierarchy (enable circuit), and 2 components (adder and counter) with the *Macro Generators*, and place them in the "test40k" design. The enable circuit is used to demonstrate the general process for entering a schematic with Viewlogic Viewdraw. The second exercise uses the IDS *Macro Generators* to create a counter and an adder.

Entering a Schematic

To enter a design or a macro, click on the *Schematic Entry* button in the Figaro Flowbar. This will invoke Viewdraw on the "test40k" design (see the "Example Design" section in this chapter for setting up the design). The following shows a completed schematic of the "test40k" design.


test40k Schematic

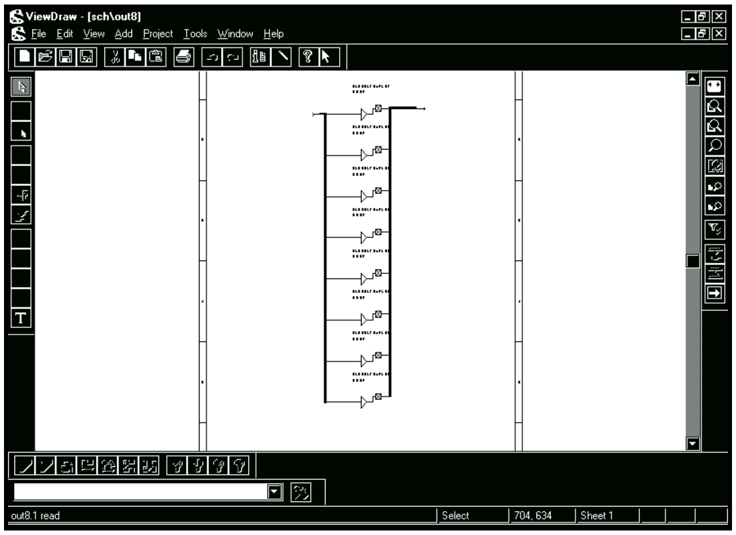
1. The default Viewlogic colors can be changed by choosing the menu option *Project>Settings*. This will bring up a tabbed dialog box. Choose the “Color Palette” tab and the window will show the colors of various display objects that can be changed. For the purposes of this exercise choose the Color Scheme *Classic Black*.



Choose Color:
Screen Color:
Printed Color:
[✓] Map all print colors to black
Color Scheme: Classic Black
[✓ Screen] Printer

Project Settings Dialog Box

2. This exercise will go through the steps to create a schematic for a level of hierarchy in the design. In this example, it will be one of output pads in the “test40k” design. The schematic of the output pad is shown below.



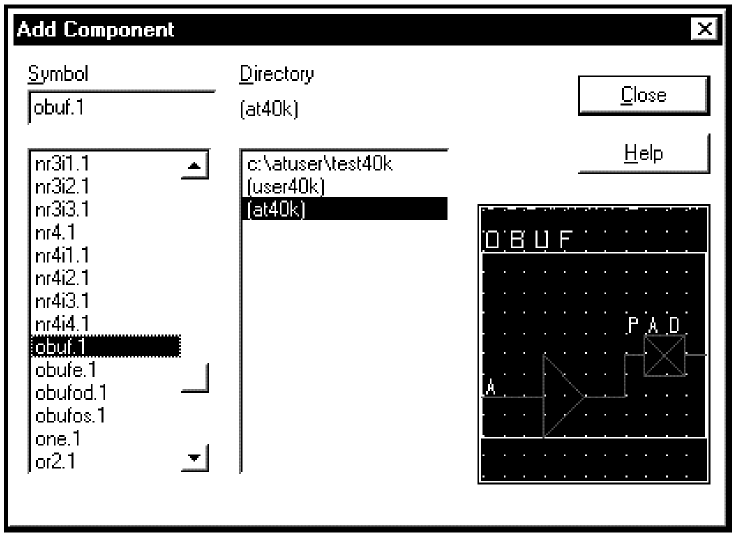
Schematic of output pads out8

3. To create a schematic,
   - Use the *File>New* dialog box and enter the name “outpads”. This should bring up a blank window for use in entering the above schematic.

4. Get a component,
   - Select *Add>Component* from the menu. Choose the FD macro from the **at6k** Library

     or

     Choose the OBUF macro from the **at40k** library as shown below.



Add Component Dialog Box

   - Click first on the symbol in the dialog box and then move the cursor to the blank schematic sheet. The symbol should now be attached to the cursor. Use the mouse and click to place the component in the schematic.
   - Close *Add Component* dialog box.

5. To add a bus,
   - Select *Add>Bus.*

- Use the mouse to add a bus to the 2 pins. Buses are drawn by first clicking (and holding down the mouse button) on one of the macro pins, dragging the mouse to the end point and then releasing the mouse button. Click on the *Cursor* button in the upper left corner of the screen to exit the “bus” mode.

6. To add the labels A[7:0] and PAD[7:0], double click on any net or bus and a dialog box should then be displayed for entering the net or bus name.
   - Type the appropriate string into the label box and then press OK.

7. To add the Array attribute, double click on the OBUF macro, go to the *Attribute* tab and enter the name as “$ARRAY” and the value as “8”. The Array attribute creates 8 instances of the OBUF macro.

8. Save the schematic using “w” or *File>Save+Check*.

9. Create a symbol for the output pads as shown below,
   - Use the *File>New* dialog box and enter the name “outpads”. Also change the “New” field to “Symbol”. This should bring up a blank window for use in entering the following symbol.

10. Draw a box,
    - Type: “B” and use the mouse to drag out a box by first clicking in the upper left hand corner of the symbol block and releasing the mouse in the lower right corner. To modify the symbol border, double click on the background, change the sheet size to “Z”, width to 140, and height to 70.

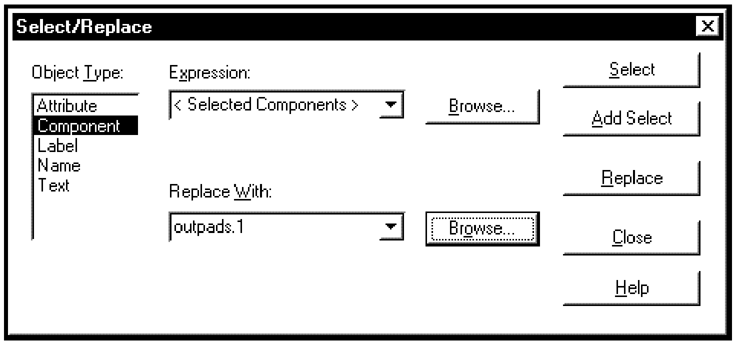
11. To add pins and labels:
    - Type: “p” and use the mouse to define the pins. The pins are lines drawn by dragging the mouse from the start point to the end point.
    - Exit the “pin” mode by selecting the *Cursor* button in the upper left corner of the screen. Select the pin by double clicking on it.
    - Type : “A[7:0]” in the label field to label the pin.
    - Repeat this process for the PAD[7:0] pins. When asked to expand the label, select the YES button.

12. To add text,
    - Type: "T" and then click at the location the text is to be placed.
    - Type: "OUTPUT PADS" in the label field and press OK to enter the text. To move the text block, exit the "text" mode and drag it to the desired location.

13. To save and close the symbol window when done,
    - Use *File>Save*, close the information window, and then use *File>Close*.

14. Save and close the OUTPADS schematic using "w" and *File>Close*.

15. To place the symbol just created into the "test40k" design,
    - Select one or both of the OUT8 symbols.
    - Use the *Edit>Replace* dialog box to replace the component.
    - This is done by setting the *Object Type* field to *Component*, setting the *Expression* field to <*Selected Components*>, setting the *Replace With* field to *OUTPADS*. and then pressing the REPLACE button.


Select/Replace

Object Type: Attribute, **Component**, Label, Name, Text

Expression: < Selected Components >

Replace With: outpads.1

Find or Edit/Replace Dialog Box

16. When complete,
    - Type: "w" to save the sheet.

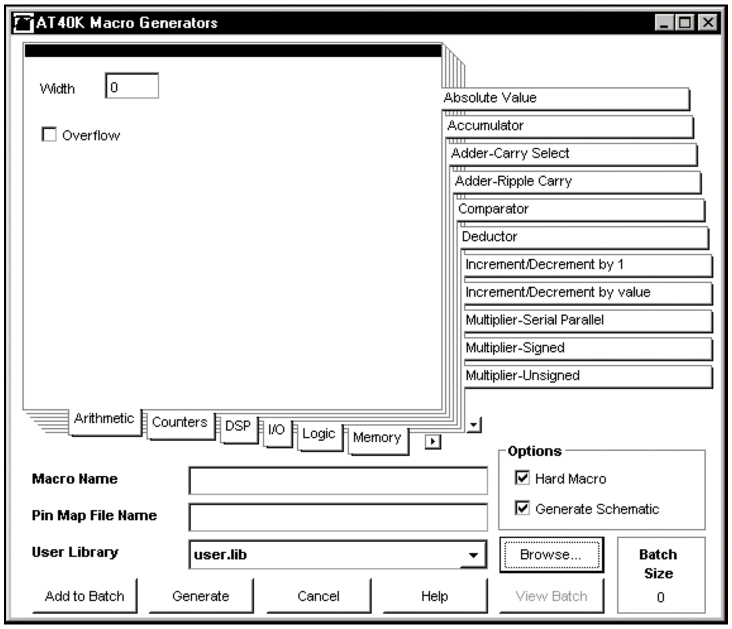
17. To exit Viewdraw,
    - Use *File>Exit*.

Using the Macro Generators

1. This exercise first creates a multiplier, MULT8NEW, with the *Macro Generators*, which is then used to replace the MULT8 component in the "test40k" design. The user should do the same for the ADDER8 component.

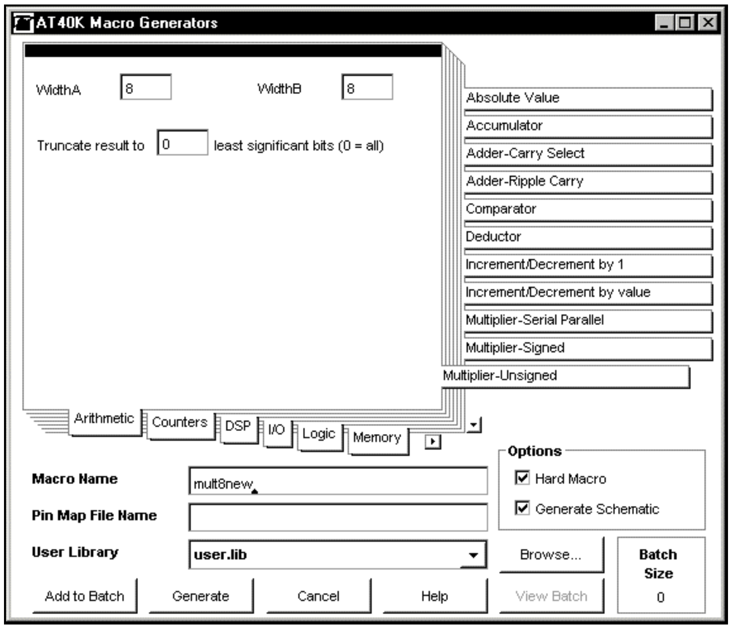
2. The library should be set up in Figaro before proceeding with the *Macro Generators*. See the "Library Setup" section in this chapter for information.

3. Click on the *Macro Generators* button on the Figaro Flowbar, and bring up the following dialog box.



Macro Generators Dialog Box

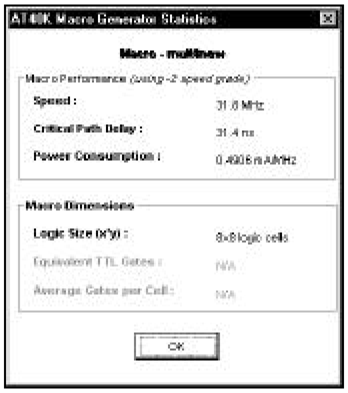
- Select the *Arithmetic* tab on the bottom of the notebook and then select the *Multiplier-Unsigned* tab along the side.
- Type: "mult8new" for *Macro Name*.
- Type: "8" for *WidthA* and *WidthB*. A sample of the completed dialog box is shown below.



Completed "Multiplier" Dialog Box

To get more details about the *Macro Generators*, the HELP button can be used. All *Generators* have *Help* descriptions which describe the parameters of the *Generator*, the pin names and pin functions, and either a description of the function or a truth table.

- Press ADD TO BATCH to initiate the program. On completion, the following statistical data window will be shown. Review the information and press OK to exit.



Macro - mult8new

Macro Performance (using -2 speed grade)

Speed :	31.8 MHz
Critical Path Delay :	31.4 ns
Power Consumption :	0.4906 mA/MHz

Macro Dimensions

Logic Size (x*y) :	8x8 logic cells
Equivalent TTL Gates :	N/A
Average Gates per Cell :	N/A

Statistics Data Window

The same process should be followed for creating an 8 bit adder.

- Select the *Arithmetic* tab on the bottom of the notebook.
- Select the *Adder - Ripple Carry* tab on the side of the notebook.
- Type: "add8new" for *Macro Name*.
- Select the *Disabled* option for the *Carry In*.
- Type: "8" for *Width*.
- Generate the component.
- Click on the OK button in the *Macro Generators* dialog box to exit.

Besides creating a hard layout (fully placed and routed) for the multiplier and the adder, this step has also created three files (symbol, schematic, and WIR) that contain all the functional information needed for simulation.

4. The newly generated macros, MULT8NEW and ADD8NEW can now be used to replace the existing components, MULT8 and ADDER8, in the "test40k" schematic respectively. Press the *Schematic Entry* button on the Figaro Flowbar to bring up the design schematic.

5. Locate the existing MULT8 in the schematic. The new component will have a different shape and the *Edit>Replace* command cannot be used.
   - Select *Add>Component* from the menu.
   - Select the library *(user)*.
   - From the menu select the component *mult8new.1*. Click on the symbol and then place MULT8NEW below the existing MULT8 macro in the schematic.
   - Connect MULT8NEW to the same nets for MULT8.
   - Select the existing MULT8 and press *<Del>* to remove it from the schematic.

6. Repeat step 5 to replace adder8 with add8new.

7. To save and exit,
   - Type: "w" and use *File>Exit*.

Functional Simulation

The design can be verified with Workview Office Simulation using the *Functional Simulation* button from Figaro. *Functional Simulation* executes two programs, check and vsm, before running ViewSim. The check program ensures that all wire files are current, and vsm creates input files for the simulator.

It is highly recommended that the user first go through the Workview Office Simulation on-line tutorials (*Help>Tutorials*) to learn about its event driven digital simulator before proceeding to the following exercise.

Please refer to the "Simulation, Viewlogic Workview Office" chapter in the *CAE Interfaces* section of this *Tutorial* for more details on simu lating this design.

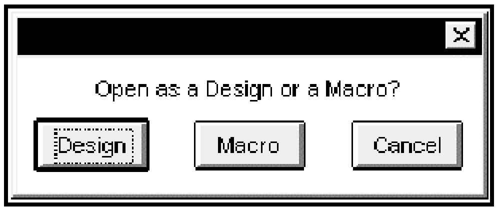
Netlist Generation

The *Netlist* button extracts a net list from the completed Viewdraw schematic for use by the system. By pressing this button, Figaro verifies that all Viewlogic Wir files are up to date prior to loading the design. The net list file created will have the correct format and physical information necessary for proceeding to the place and route phase of the design.

The *Netlist* button will also create an EDIF netlist. This file should be used when *Bus Rippers* are employed in the Viewlogic schematic.

Figaro

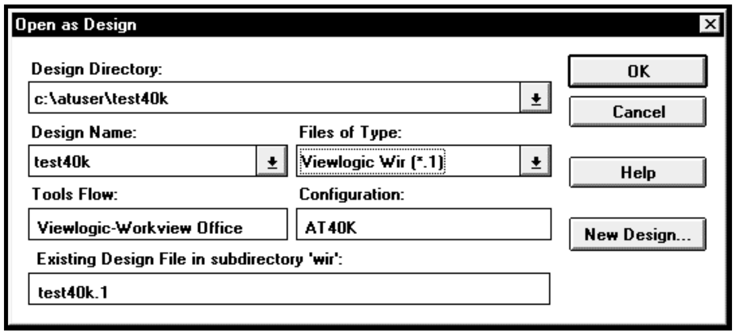
When the user is ready to create the design by automatic or manual placement and routing, click on the *Open* button in the Flowbar and answer the first dialog box with the *Design* option.


Open as a Design or a Macro?

Open as Design or Macro Dialog Box

Once specified, the *Open as Design* dialog box will be displayed. Verify that *Files of Type* is set to *Viewlogic Wir (*.1)* so the files can be processed to create a Figaro database. However, *Files of Type* should be set to *EDIF (*.edf)* when employing *Bus Rippers* in Viewdraw. A correctly set up *Open as Design* dialog box is shown below.

After the design is opened, follow the flow to go through the *Parts* and *Compile* buttons to finish placement, routing, and bitstream generation.


Open as Design

Design Directory: c:\atuser\test40k

Design Name: test40k

Files of Type: Viewlogic Wir (*.1)

Tools Flow: Viewlogic-Workview Office

Configuration: AT40K

Existing Design File in subdirectory 'wir': test40k.1

Open as Design Dialog Box

Details on the Figaro processes as illustrated by the "test40k" example design can be found in the *Figaro* section of the *IDS Tutorial*.

Post-layout Simulation

After the design has been implemented with the *Compile* function, *Post-layout Simulation* can be performed. Viewlogic simulation tools can be invoked with the provided post-layout wire delays, including pin-to-pin delays, setup and hold times, as well as actual wire delays, to predict device timing and performance.

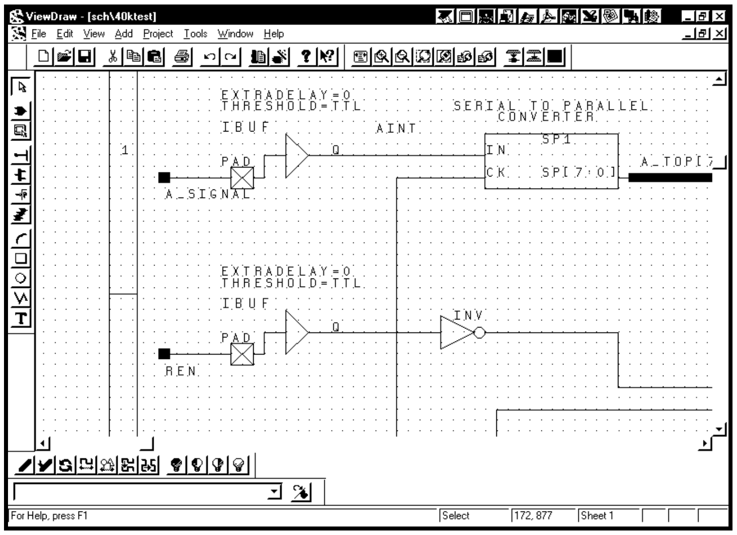
Please refer to the “Simulation, Viewlogic Workview Office” chapter in the *CAE Interfaces* section of this *Tutorial* for more details on *Post-layout Simulation* of this design.

Engineering Change Order (ECO)

Figaro’s ECO feature supports revisions to the schematic, while recompiling only the modified areas of the chip. The process allows repeated design changes to be compiled quickly and efficiently, while ensuring minimal changes to the circuit’s timing.

For this exercise, a minor change will be made to the “test40k” schematic using ECO. In this case, the polarity of the REN signal will be reversed.

1. Bring up the “test40k” schematic by pressing on the *Schematic Entry* button. Once in Viewdraw, add an inverter to the output of the I/O buffer (which is bringing in the REN signal) as shown below.

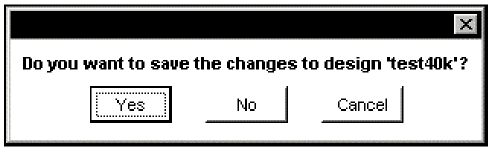


Modified Schematic with Inverter

2. When completed, write the design using “*w*” and then exit Viewdraw.

3. Use the icon on the Flowbar to create an updated netlist.

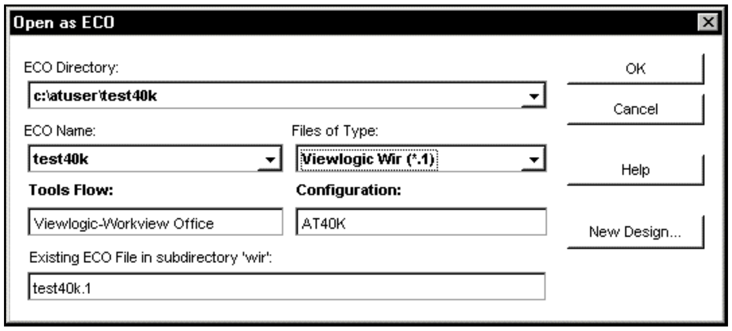
4. To incorporate the change into Figaro use the *File>Open as ECO* option. The existing layout must already be loaded into Figaro. It will then request to save the design as follows.


Do you want to save the changes to design 'test40k'?

ECO Dialog Box

5. Answer YES and bring up the *Open as ECO* dialog box. Change *Files of Type* to *Viewlogic WIR files (*.1)* and ensure that *test40k.1* is specified as the input file name.

   For designs which employ *Bus Rippers*, change *Files of Type* to *EDIF (*.edf)* and verify that *design*.edf is shown as the input file name.


Open as ECO

ECO Directory: c:\atuser\test40k

ECO Name: test40k

Files of Type: Viewlogic Wir (*.1)

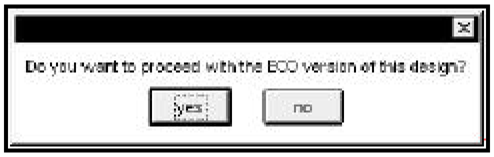
Tools Flow: Viewlogic-Workview Office

Configuration: AT40K

Existing ECO File in subdirectory 'wir': test40k.1

ECO Dialog Box

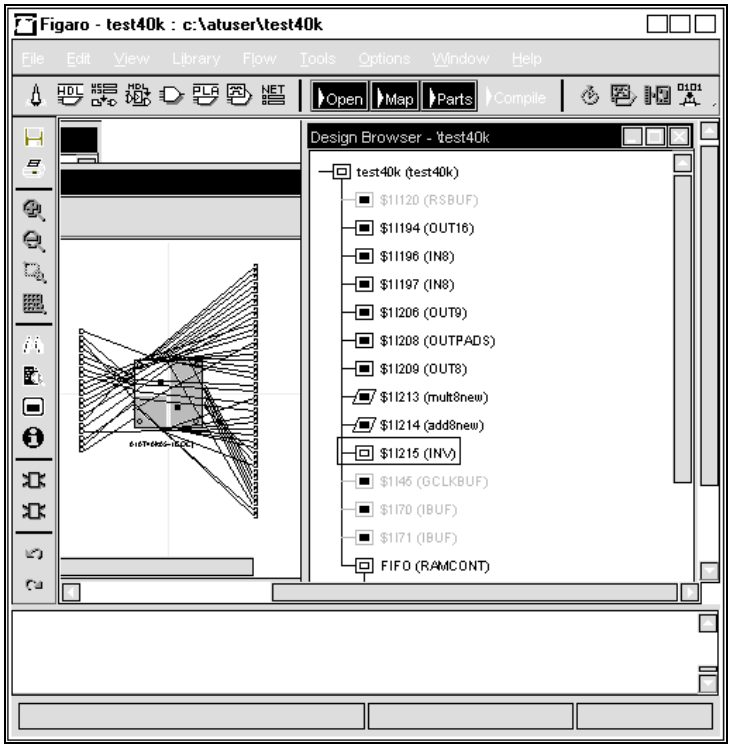
6. Press OK to initiate ECO. During this process, Figaro will report all differences and then request confirmation to proceed.


Do you want to proceed with the ECO version of this design?

ECO Pop Up Box

7. Press YES to execute changes to the design.

   The *Design Browser* below shows that except for the new inverter, which is inside a highlight box, all original instances are now locked and identified by a solid square preceding their names.

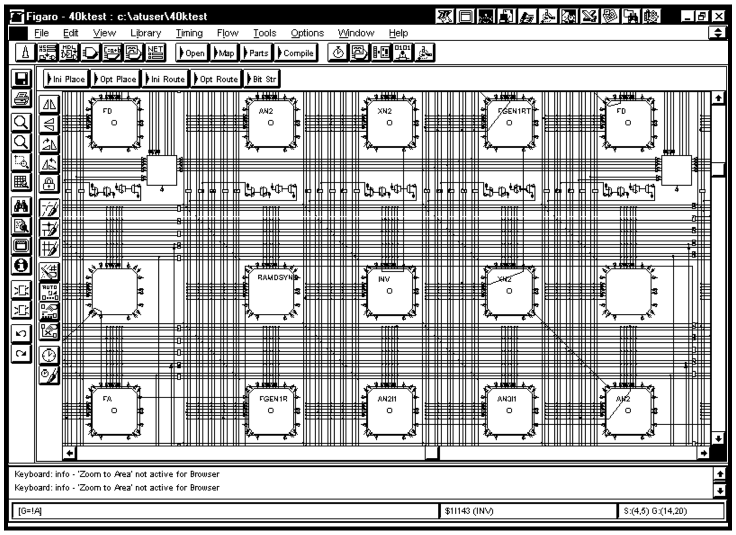

Figaro - test40k : c:\atuser\test40k

Design Browser - test40k

test40k (test40k)
$1I120 (RSBUF)
$1I194 (OUT16)
$1I196 (IN8)
$1I197 (IN8)
$1I206 (OUT9)
$1I208 (OUTPADS)
$1I209 (OUT8)
$1I213 (mult8new)
$1I214 (add8new)
$1I215 (INV)
$1I45 (GCLKBUF)
$1I70 (IBUF)
$1I71 (IBUF)
FIFO (RAMCONT)

Changed Instance in Figaro Browser

8. Proceed with *Compile* after this point and the program will place and route the inverter while leaving everything else locked down. The resulting design is shown below.



```
Keyboard: info - 'Zoom to Area' not active for Browser
Keyboard: info - 'Zoom to Area' not active for Browser
[G=!A]                                   $1I143 (INV)          S:(4,5) G:(14,20)
```

ECO Result

NOTE

In some cases unresolvable placement contention can occur after the ECO compile phase as the tool locks the placement of the unchanged macros from the original design. To resolve these contentions manually move the components or unlock some instances around the contention region and re-run the automatic placement.

HDLPlanner

HDLPlanner is a design entry software for entering VHDL or Verilog designs. It has a user interface to define and instantiate frequently used components such as register banks and counters. There is also a user interface to automatically generate layouts for macros that are instantiated in HDLPlanner. Please refer to the *Design Entry* chapter of the *User's Guide* for details on HDLPlanner capabilities and a description of the user interface options.

In this tutorial, a data averager design is first created in VHDL. The design will then be optimized using Exemplar's Leonardo software.

This tutorial assumes that the user is familiar with Atmel specific synthesis flow using the one of the following software tools: Synopsys, Exemplar or Design Compiler. Refer to the *HDL Entry, Design Entry* chapter in the *User's Guide* and *CAE Interfaces* section in the *IDS Tutorial* for details.

Example Design

For this tutorial, the "averager" design will be used. This design implements a waveform smoothing function represented by:

$$y(n) = \frac{1}{N}\sum_{(i=0)}^{(N-1)} x(n-i)$$

The smoothing function is used in many DSP applications to filter out high frequency spikes. These spikes are the source of noise commonly found in communication channels. They can be eliminated by taking the moving average of sample values arriving at the input of the system.

The design in this example is an "averager" circuit with eight moving points. The hardware specific details of the design are described below.

Interface The "averager" design has eight bit input and output data lines. The design also has a clock pin and a reset pin.

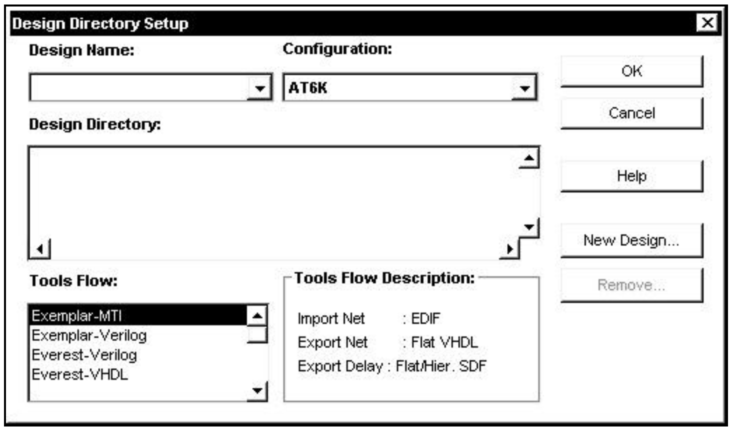
Assumptions and Other Details Only positive values can appear on the input data lines (the input is assumed to be level-shifted). During the implementation of the system, a valid output waveform can be expected on the 9th clock. However, since the outputs of the last and intermediate stages are registered, the actual output waveform starts occurring from the 11th clock cycle onwards.

Invoking Figaro

Figaro can be started by selecting START > ATMEL > IDS

Figaro Design Setup

In the Figaro window, use the icon or *File>Design Setup* menu to set up the design and select its associated *Tools Flow*. The *Design Directory Setup* dialog box will be brought up.



Design Name:

Configuration: AT6K

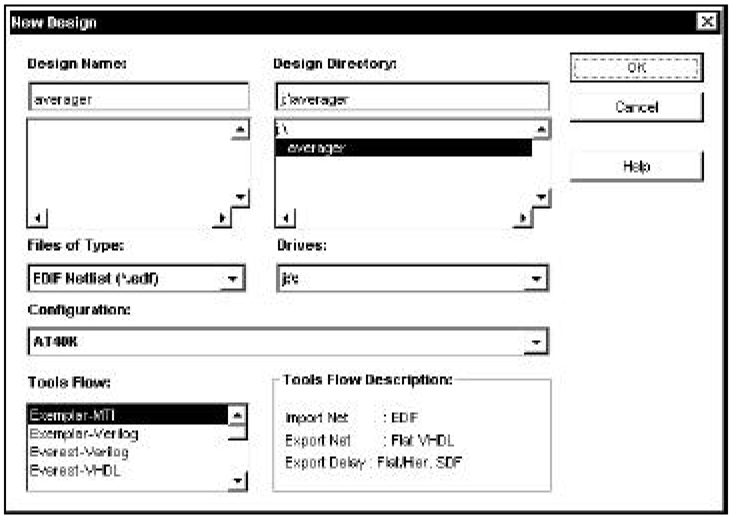
Design Directory:

Tools Flow:
Exemplar-MTI
Exemplar-Verilog
Everest-Verilog
Everest-VHDL

Tools Flow Description:
Import Net : EDIF
Export Net : Flat VHDL
Export Delay : Flat/Hier. SDF

Design Directory Setup Dialog Box

To create a new design click on the *New Design* button. The *New Design* dialog box is shown below.



Design Name: averager

Design Directory: j:\averager
j:\
averager

Files of Type: EDIF Netlist (*.edf)

Drives: j:

Configuration: AT40K

Tools Flow:
Exemplar-MTI
Exemplar-Verilog
Everest-Verilog
Everest-VHDL

Tools Flow Description:
Import Net : EDIF
Export Net : Flat VHDL
Export Delay : Flat/Hier. SDF

New Design Dialog Box

In the *New Design* dialog box, type the name of the design as "*averager*". Select the appropriate tool from the *Tools Flow* menu. See the table below:

Tool	Tool Flow Selection
Exemplar VHDL	*Exemplar-MTI*
Synopsys VHDL	*Synopsys-VSS*
Exemplar Verilog	*Exemplar-Verilog*
Synopsys Verilog	*Synopsys-Verilog*

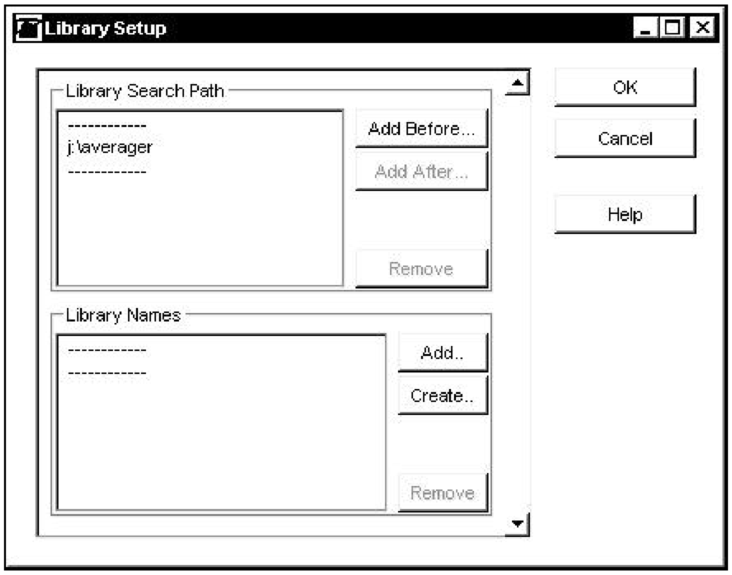
Tools Flow Selection

Dismiss the *New Design* and *Design Directory* dialog boxes by pressing the respective OK buttons.

Library Setup

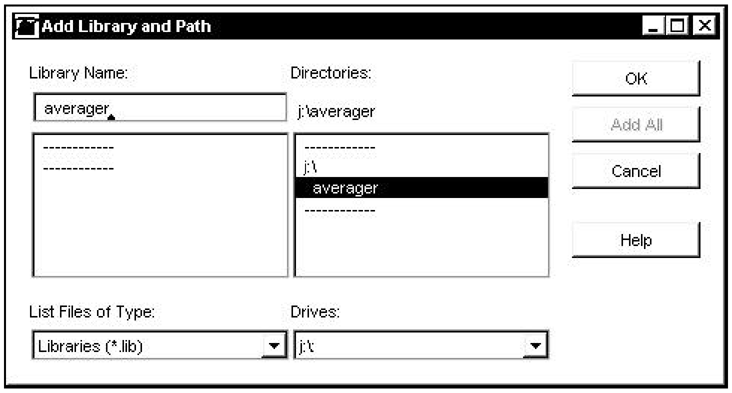
The “averager” design, once synthesized, will contain the macros that are either inferred automatically or instantiated in the HDLPlanner software. These components will then be created automatically with the *Macro Generators.* The associated FPGA layout and simulation models will also be produced. Figaro treats these components as library modules and inserts them in the library. Consequently, a user library must be set up.

A design library can be established by executing the *Library>Library Setup* menu.



Library Setup Dialog Box

In *Library Setup*, press the *Add Before* button to display the *Add Library and Path* dialog box.



Add Library and Path Dialog Box

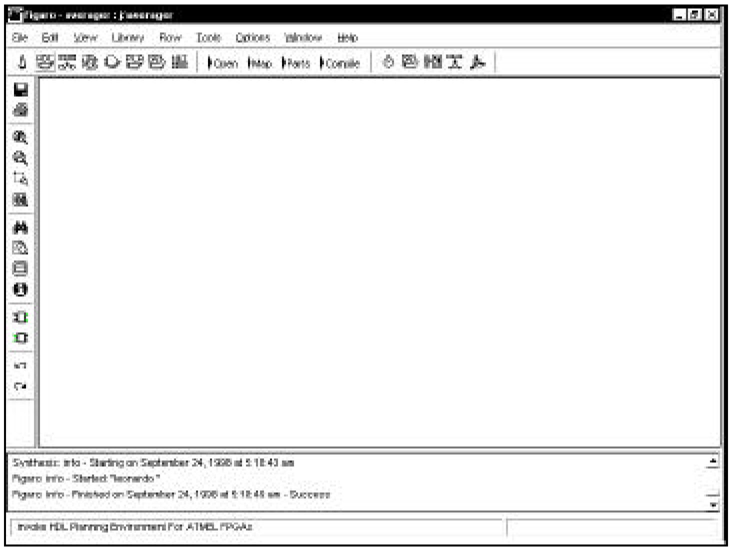
In the *Add Library and Path* dialog box, enter *Library Name* as “*averager*.lib” (or any name of choice). Dismiss the box by pressing OK.

Dismiss the *Library Setup* dialog box by pressing OK.

So far, the design name has been entered as “averager” and the library name as “*averager*.lib”.

Design Entry Using HDLPlanner

After the design and libraries are set up, HDLPlanner can be invoked by pressing the HDL button on the Flowbar.

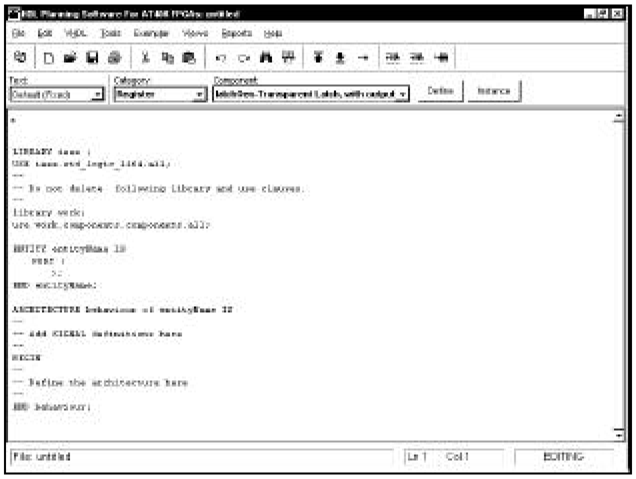

Figaro - avenager : [avenager]
File Edit View Library Flow Tools Options Window Help
Open Map Parts Compile

```
Synthesis: Info - Starting on September 24, 1998 at 5:18:43 am
Figaro Info - Started "leonardo"
Figaro Info - Finished on September 24, 1998 at 5:18:46 am - Success
```

Invoke HDL Planning Environment For ATMEL FPGAs

HDL Button on Figaro Flowbar

The next two figures will be brought up for Verilog designs.

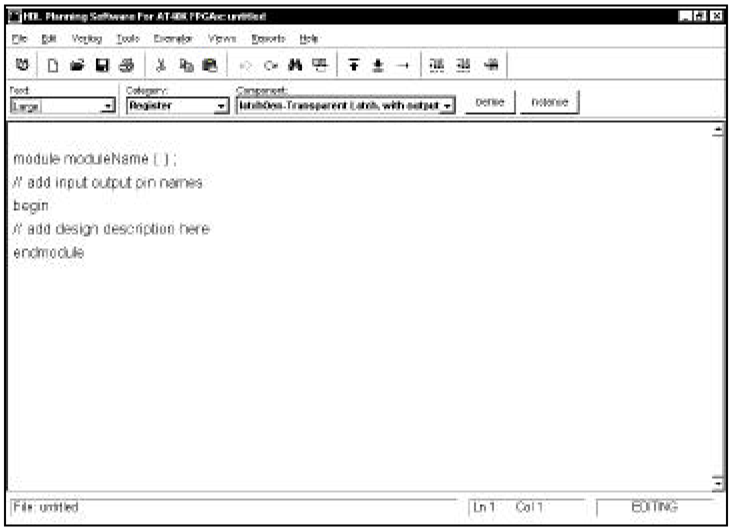

HDL Planning Software For ATMEL FPGAs: untitled
File Edit VHDL Tools Example Views Reports Help
Define
Instance

Text: Default (Fixed)
Category: Register
Component: latch8en-Transparent Latch, with output

```
*


LIBRARY ieee ;
USE ieee.std_logic_1164.all;
--
-- Do not delete following library and use clauses.
--
library work;
use work.components.components.all;

ENTITY entityName IS
   PORT (
        );
END entityName;

ARCHITECTURE behaviour of entityName IS
--
-- Add SIGNAL definitions here
--
BEGIN
--
-- Define the architecture here
--
END behaviour;
```

File: untitled   Ln 1   Col 1   EDITING

HDL Planner User Interface for VHDL


HDL Planning Software For AT40K FPGAs: untitled
File Edit Verilog Tools Exemplar Views Reports Help
Text: Large
Category: Register
Component: latch0en-Transparent Latch, with output
Define
Instance
File: untitled
Ln 1 Col 1
EDITING

```
module moduleName ( ) ;
// add input output pin names
begin
// add design description here
endmodule
```

HDLPlanner User Interface for Verilog

Important menus and interface components are described below:

VHDL Contains predefined templates of commonly used constructs

Verilog Contains predefined templates of commonly used constructs

Tools Allows the user to invoke the *Macro Generators* on instantiated components and collect information on them.

Synopsys This can be used to access Synopsys specific synthesis tips.

Exemplar This can be used to access Exemplar specific synthesis tips.

Define Can be used to insert the definition of the macro displayed in the *Component* list box.

Instance Instantiates a component that is displayed in the *Component* list box.

NOTE A VHDL package called components is automatically created by the HDL Planner software when a design is saved. This is saved in the atmel.vhd file in the current directory and contains all components that are automatically defined using HDL Planner. The default package and file names can be overwritten by using the parameter *VHDL Package Name* in the *Tools>Options* window.

Organization of the Averager Design

The “averager” design is hierarchically organized into the following components.

module initadd This module adds two unit delays to the path of the input signal, which is then output via the dataout port. The delayed signals are added together and sent out on the result port.

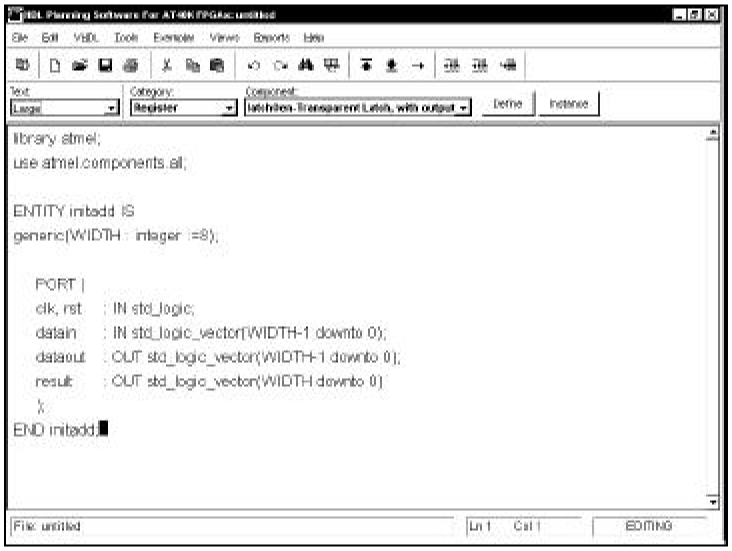
This module uses two register banks and an adder to implement delays and perform the addition.

module averager The averager module contains a serial chain of four initadd modules. The dataout port of the previous module is connected to the datain port of the next module. The outputs of the adjacent modules are added using 9 bit adders. The 2 resulting outputs are then added together using a 10 bit adder, and shifted right 3 bits to divide by eight. The shifted sum is registered, which forms the moving average.

Creating an initadd module

Defining Entity

On bringing up HDLPlanner, the Module/Entity template is displayed on the screen. Modify the template to add port definitions for the initadd module.

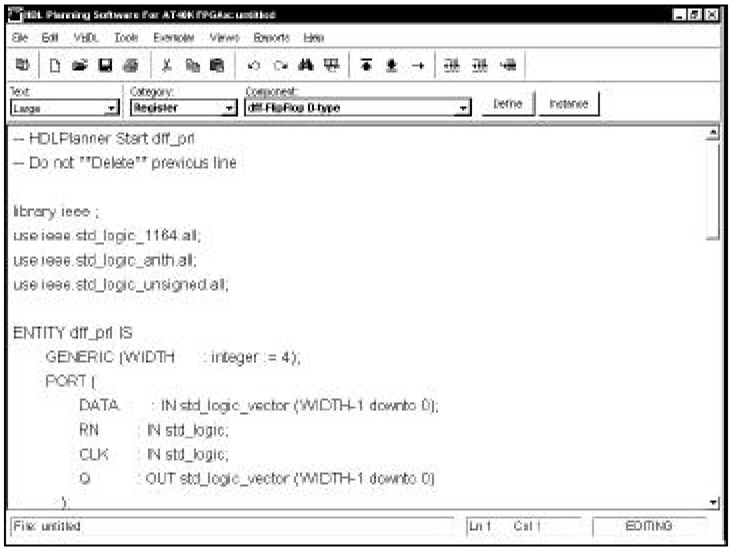

HDL Planning Software For AT40K FPGAs: untitled
File Edit VHDL Tools Examples Views Exports Help
Text: Large
Category: Register
Component: latchrlen-Transparent Latch, with output
Define
Instance
File: untitled
Ln 1 Col 1
EDITING

```
library atmel;
use atmel.components.all;

ENTITY initadd IS
generic(WIDTH : integer :=8);

    PORT (
    clk, rst    : IN std_logic;
    datain      : IN std_logic_vector(WIDTH-1 downto 0);
    dataout     : OUT std_logic_vector(WIDTH-1 downto 0);
    result      : OUT std_logic_vector(WIDTH downto 0)
    );
END initadd;
```

Defining Entity

Defining Components

The initadd module contains two register banks and one adder. To add a register, set *Category* to *Register* and *Component* to *dff*. Click on the *Define* button. Upon pressing *Define*, a parameterized module of the register bank will be inserted into the design file as shown in the following. Use a similar method to insert the adder module.

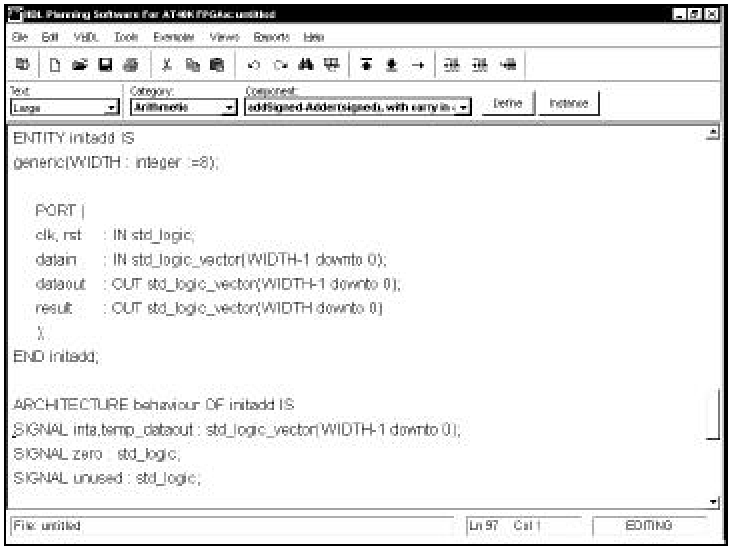

HDL Planning Software For AT40K FPGAs: untitled
File Edit VHDL Tools Examples Views Exports Help
Text: Large
Category: Register
Component: dff-FlipFlop D-type
Define
Instance
File: untitled
Ln 1 Col 1
EDITING

```
-- HDLPlanner Start dff_prl
-- Do not ""Delete"" previous line

library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

ENTITY dff_prl IS
      GENERIC (WIDTH      : integer := 4);
      PORT (
            DATA      : IN std_logic_vector (WIDTH-1 downto 0);
            RN        : IN std_logic;
            CLK       : IN std_logic;
            Q         : OUT std_logic_vector (WIDTH-1 downto 0)
         );
```

Defining Components

Defining Signals

Using *Verilog>Wire* or *VHDL>Signal (as appropriate for your specific design)*, insert the signal templates. Modify them to look like the figure below.

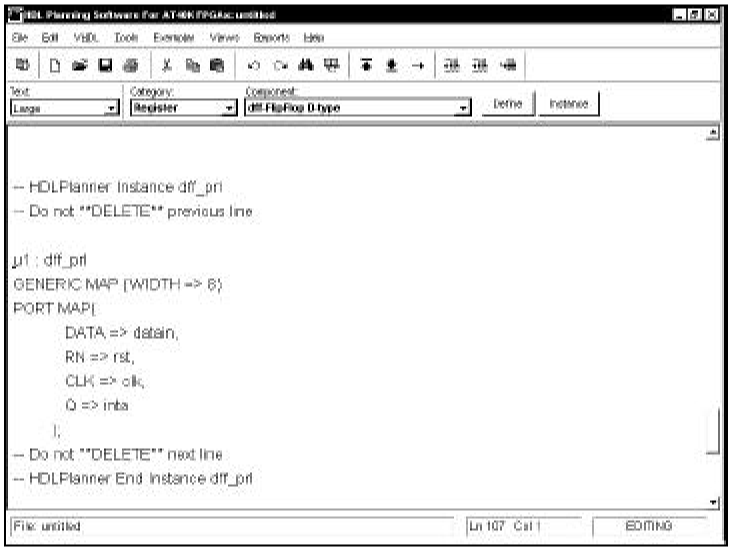

HDL Planning Software For AT40K FPGAs: untitled
File Edit VHDL Tools Exemplar Views Reports Help
Text: Large
Category: Arithmetic
Component: addSigned-Adder(signed), with carry in
Define
Instance
File: untitled
Ln 97 Col 1
EDITING

```
ENTITY initadd IS
generic(WIDTH : integer :=8);

    PORT (
    clk, rst     : IN std_logic;
    datain       : IN std_logic_vector(WIDTH-1 downto 0);
    dataout      : OUT std_logic_vector(WIDTH-1 downto 0);
    result       : OUT std_logic_vector(WIDTH downto 0)
    );
END initadd;

ARCHITECTURE behaviour OF initadd IS
SIGNAL inta,temp_dataout : std_logic_vector(WIDTH-1 downto 0);
SIGNAL zero : std_logic;
SIGNAL unused : std_logic;
```

Defining Signals

Instantiating Components

The components, once defined, can be instantiated in the module/architecture section of the initadd module. Select *dff* from the *Component* list box. Position the cursor at the desired location in the file and press *Instance*. At the click of the button, an instantiation statement for the dff module will be inserted. Modify the instantiation statement to create an 8 bit wide dff register and connect the pins to the nets in the design.


HDL Planning Software For AT40K FPGAs: untitled
File Edit VHDL Tools Exemplar Views Reports Help
Text: Large
Category: Register
Component: dff-FlipFlop D-type
Define
Instance
File: untitled
Ln 107 Col 1
EDITING

```
-- HDLPlanner Instance dff_prl
-- Do not **DELETE** previous line

u1 : dff_prl
GENERIC MAP (WIDTH => 8)
PORT MAP(
        DATA => datain,
        RN => rst,
        CLK => clk,
        Q => inta
      );
-- Do not **DELETE** next line
-- HDLPlanner End Instance dff_prl
```

Instantiating Components

NOTE Comments containing the word “HDLPlanner” have special meaning to the software. Do not delete those lines.

Instantiate another dff register and an adder module. The complete the design using VHDL is shown below:

```
-- Do not delete  following library and use clauses.
--
library atmel;
use atmel.components.all;
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY initadd IS

generic(WIDTH   : integer :=8);
    PORT (
    clk, rst    : IN std_logic;
    datain      : IN std_logic_vector(WIDTH-1 downto 0);
    dataout     : OUT std_logic_vector(WIDTH-1 downto 0);
    result      : OUT std_logic_vector(WIDTH downto 0)
    );
END initadd;

ARCHITECTURE behaviour OF initadd IS

SIGNAL inta,temp_dataout : std_logic_vector(WIDTH-1 downto 0);
SIGNAL zero : std_logic;
SIGNAL unused : std_logic;

BEGIN
zero <= '0';

-- HDLPlanner Instance dff_prl
-- Do not **DELETE** previous line

u1 : dff_prl
GENERIC MAP (WIDTH => 8)
PORT MAP(
           DATA => datain,
           RN => rst,
           CLK => clk,
           Q => inta
        );

-- Do not **DELETE** next line
-- HDLPlanner End Instance dff_prl


-- HDLPlanner Instance dff_prl
-- Do not **DELETE** previous line

u2 : dff_prl
GENERIC MAP (WIDTH => 8)
PORT MAP(
           DATA => inta,
           RN => rst,
           CLK => clk,
           Q => temp_dataout
        );

-- Do not **DELETE** next line
-- HDLPlanner End Instance dff_prl


-- HDLPlanner Instance addSigned
-- Do not **DELETE** previous line

u3 : addSigned
GENERIC MAP (WIDTH => 8)
PORT MAP (
            DATAA => inta,
            DATAB => temp_dataout,
            CIN => zero,
            SUM => result(WIDTH-1 downto 0),
            COUT => result(WIDTH),
            OVERFLOW => unused
        );
-- Do not **DELETE** next line
-- HDLPlanner End Instance addSigned

dataout <= temp_dataout;

END behaviour;
```

The complete the design using VHDL is shown below:

```
module initadd (CLK, rst, datain, dataout, result);

parameter DATAWIDTH  = 8 ;

input CLK;
input rst;
input [DATAWIDTH-1:0] datain;
output [DATAWIDTH-1:0] dataout;
output [DATAWIDTH:0] result;

wire [7:0] inta;
wire [7:0] temp_dataout;


// HDLPlanner Instance dff_prl
// Do not **DELETE** previous line

defparam u1.width = 8;
dff_prl  u1 (.DATA(datain), .RN(rst), .CLK(CLK), .Q(inta) );
// Do not **DELETE** next line
// HDLPlanner End Instance dff_prl


// HDLPlanner Instance dff_prl
// Do not **DELETE** previous line

defparam u2.width = 8;
dff_prl  u2 (.DATA(inta), .RN(rst), .CLK(CLK), .Q(temp_dataout) );
// Do not **DELETE** next line
// HDLPlanner End Instance dff_prl


// HDLPlanner Instance addSigned
// Do not **DELETE** previous line

defparam u3.width = 8;
addSigned  u3 (.DATAA(inta), .DATAB(temp_dataout), .CIN(1'b0),
.SUM(result[DATAWIDTH-1:0]), .COUT(result[DATAWIDTH]) );

// Do not **DELETE** next line
// HDLPlanner End Instance addSigned

assign
    dataout = temp_dataout;

endmodule
```

Listing 1 using Verilog

Creating the Averager Module

Using *Verilog>Module Definition*, or *VHDL>Entity* and *VHDL>Architecture modules*, define the entity and architecture of the averager. In addition to the initadd module, the averager design uses a shift register, registered signed adder, and register bank with enable. Add the following modules to the file:

shiftRegIserOser
addSignedReg
dffIen

Once defined, these modules can be instantiated in the architecture body and connected to each other.

NOTE Components that are user-defined and inserted automatically (such as initadd) must be declared explicitly before they can be used in the architecture. The components defined and instantiated using HDL Planner are declared in the package file atmel.vhd that is created automatically when the design is saved.

Listing 2 using VHDL

```
-- Do not delete  following library and use clauses.
--
LIBRARY atmel;
USE atmel.components.ALL;
library ieee;
use ieee.std_logic_1164.all;

ENTITY averager IS

GENERIC (WIDTH  : integer :=8);
    PORT (
    clk, rst        : IN std_logic;
    datain          : IN std_logic_vector(WIDTH-1 downto 0);
    dataout         : OUT std_logic_vector(WIDTH-1 downto 0)
    );
END averager;

ARCHITECTURE behaviour OF averager IS

COMPONENT initadd
    GENERIC (WIDTH      : integer := 8);
    PORT (
    clk, rst        : IN std_logic;
    datain          : IN std_logic_vector(WIDTH-1 downto 0);
    dataout     : OUT std_logic_vector(WIDTH-1 downto 0);
    result          : OUT std_logic_vector(WIDTH downto 0)
    );
END COMPONENT;

SIGNAL resulta,resultb,resultc,resultd : std_logic_vector(WIDTH    downto 0);
SIGNAL result9adda,result9addb : std_logic_vector(WIDTH+1 downto   0);
SIGNAL dout : std_logic_vector(WIDTH+2 downto 0);
SIGNAL inta,intb,intc : std_logic_vector(WIDTH-1 downto 0);
SIGNAL zero : std_logic;
SIGNAL enable : std_logic;
SIGNAL clkarrived : std_logic;

BEGIN

PROCESS (clk, rst)
BEGIN
    IF (rst = '0') THEN
        clkarrived <= '0';
    ELSIF(clk'event AND clk = '1') THEN
        clkarrived <= '1';
    END IF;
END PROCESS;

u1 : initadd
    GENERIC MAP (WIDTH => 8)
    PORT MAP (
        clk => clk,
        rst => rst,
        datain => datain,
        dataout => inta,
        result => resulta
    );

u2 : initadd
    GENERIC MAP (WIDTH => 8)
    PORT MAP (
        clk => clk,
        rst => rst,
        datain => inta,
        dataout => intb,
        result => resultb
    );

u3 : initadd
```

```
    GENERIC MAP (WIDTH => 8)
    PORT MAP (
        clk => clk,
        rst => rst,
        datain => intb,
        dataout => intc,
        result => resultc
    );

u4 : initadd
    GENERIC MAP (WIDTH => 8)
    PORT MAP (
        clk => clk,
        rst => rst,
        datain => intc,

        result => resultd

    );

-- HDLPlanner Instance addSignedReg_prl
-- Do not **DELETE** previous line

u5 : addSignedReg_prl
GENERIC MAP (WIDTH => 9)
PORT MAP (
            CLK => clk,
            RN   => rst,
            CIN => zero,
            DATAA => resulta,
            DATAB => resultb,
            SUM => result9adda(8 downto 0),
            COUT  => result9adda(9)
          );

-- Do not **DELETE** next line
-- HDLPlanner End Instance addSignedReg_prl


-- HDLPlanner Instance addSignedReg_prl
-- Do not **DELETE** previous line

u6 : addSignedReg_prl
GENERIC MAP (WIDTH => 9)
PORT MAP (
            CLK => clk,
            RN   => rst,
            CIN => zero,
            DATAA => resultc,
            DATAB => resultd,
            SUM => result9addb(8 downto 0),
            COUT  => result9addb(9)
          );

-- Do not **DELETE** next line
-- HDLPlanner End Instance addSignedReg_prl


-- HDLPlanner Instance addSignedReg_prl
-- Do not **DELETE** previous line

u7 : addSignedReg_prl
GENERIC MAP (WIDTH => 10)
PORT MAP (
            CLK => clk,
            RN   => rst,
            CIN => zero,
            DATAA => result9adda,
            DATAB => result9addb,
            SUM => dout(9 downto 0),
            COUT  => dout(10)
          );

-- Do not **DELETE** next line
-- HDLPlanner End Instance addSignedReg_prl
-- HDLPlanner Instance dffIen_prl
-- Do not **DELETE** previous line

u8 : dffIen_prl
GENERIC MAP (WIDTH => 8)
PORT MAP(
           DATA => dout(10 downto 3),
           RN => rst,
           CLK => clk,
           ENABLE => enable,
           Q => dataout
        );

-- Do not **DELETE** next line
-- HDLPlanner End Instance dffIen_prl


-- HDLPlanner Instance shiftRegIserOser_prl
-- Do not **DELETE** previous line

u9 : shiftRegIserOser_prl
GENERIC MAP (WIDTH => 9)
PORT MAP(
           RN      => rst,
           CLK    => clk,
           ENABLE   => zero,
           SHIFTIN  => clkarrived,
           SHIFTOUT => enable
        );

-- Do not **DELETE** next line
-- HDLPlanner End Instance shiftRegIserOser_prl

END behaviour;
```

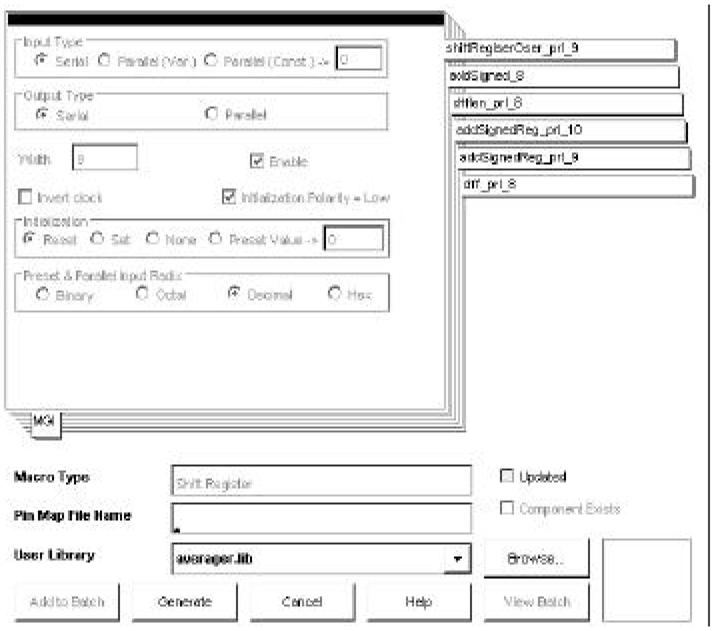
Listing 2 using Verilog

```
module averager (CLK, rst, datain, dataout);

`define DATAWIDTH  8

input CLK;
input rst;
input [`DATAWIDTH-1:0] datain;
output [`DATAWIDTH-1:0] dataout;

reg clkarrived;

wire [`DATAWIDTH-1:0] inta;
wire [`DATAWIDTH-1:0] intb;
wire [`DATAWIDTH-1:0] intc;
wire [`DATAWIDTH-1:0] junk;
wire [`DATAWIDTH:0] resulta;
wire [`DATAWIDTH:0] resultb;
wire [`DATAWIDTH:0] resultc;
wire [`DATAWIDTH:0] resultd;
wire [`DATAWIDTH+1:0] result9adda;
wire [`DATAWIDTH+1:0] result9addb;
wire [`DATAWIDTH+2:0] dout;

always @(posedge CLK or negedge rst)
begin
    if(rst == 'b0)
        clkarrived = 'b0;
    else
        clkarrived = 'b1;
end

initadd u1 (.CLK(CLK), .rst(rst), .datain(datain), .dataout(inta),
.result(resulta));
initadd u2 (.CLK(CLK), .rst(rst), .datain(inta), .dataout(intb),
.result(resultb));
initadd u3 (.CLK(CLK), .rst(rst), .datain(intb), .dataout(intc),
.result(resultc));
initadd u4 (.CLK(CLK), .rst(rst), .datain(intc), .dataout(junk),
.result(resultd));


// HDLPlanner Instance addSignedReg_prl
// Do not **DELETE** previous line

defparam u5.width = 9;
addSignedReg_prl  u5 (.DATAA(resulta),. DATAB(resultb), .CLK(CLK), .RN(rst),
.CIN(1'b0),    .SUM(result9adda[8:0]), .COUT(result9adda[9]));
```

```
// Do not **DELETE** next line
// HDLPlanner End Instance addSignedReg_prl


// HDLPlanner Instance addSignedReg_prl
// Do not **DELETE** previous line

defparam u6.width = 9;
addSignedReg_prl  u6 (.DATAA(resultc),. DATAB(resultd), .CLK(CLK), .RN(rst),
.CIN(1'b0),    .SUM(result9addb[8:0]), .COUT(result9addb[9]));

// Do not **DELETE** next line
// HDLPlanner End Instance addSignedReg_prl


// HDLPlanner Instance addSignedReg_prl
// Do not **DELETE** previous line

defparam u7.width = 10;
addSignedReg_prl  u7 (.DATAA(result9adda),. DATAB(result9addb), .CLK(CLK),
.RN(rst), .CIN(1'b0),    .SUM(dout[9:0]), .COUT(dout[10]));

// Do not **DELETE** next line
// HDLPlanner End Instance addSignedReg_prl


// HDLPlanner Instance dffIen_prl
// Do not **DELETE** previous line

defparam u8.width = 8;
dffIen_prl  u8 (.DATA(dout[10:3]), .RN(rst), .CLK(CLK), .ENABLE(enable),
.Q(dataout) );
// Do not **DELETE** next line
// HDLPlanner End Instance dffIen_prl


// HDLPlanner Instance shiftRegIserOser_prl
// Do not **DELETE** previous line

defparam u9.width = 9;
shiftRegIserOser_prl  u9 (.RN(rst), .CLK(CLK), .ENABLE(1'b0),
.SHIFTIN(clkarrived), .SHIFTOUT(enable)   );
// Do not **DELETE** next line
// HDLPlanner End Instance shiftRegIserOser_prl

endmodule
```

Invoking Macro Generators

After the design is entered, the macros that are instantiated in the design can be generated by accessing the *Tools>Invoke Macro Generators* panel.



Input Type: [✓ Serial] Parallel (Var.) Parallel (Const.) -> 0
Output Type: [✓ Serial] Parallel
Width: 8
[✓ Enable]
Invert clock
[✓ Initialization Polarity = Low]
Initialization: [✓ Reset] Set None Preset Value -> 0
Preset & Parallel Input Radix: Binary Octal [✓ Decimal] Hex

shiftRegisterOser_prl_9
addSigned_8
diffen_prl_8
addSignedReg_prl_10
addSignedReg_prl_9
diff_prl_8

Macro Type: Shift Register
Pin Map File Name:
User Library: averager.lib
Updated
Component Exists

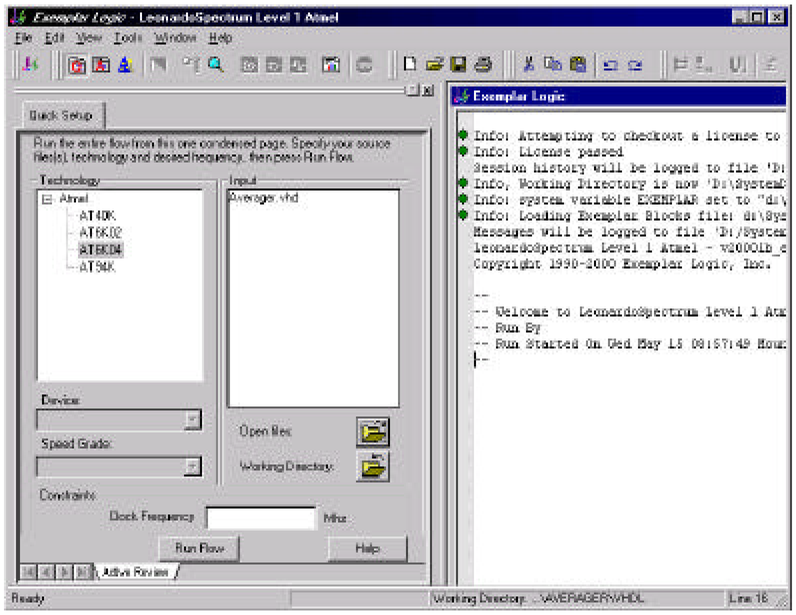
Macro Generators Dialog Box

Press the GENERATE button to create the macros. On completion, Figaro will return to the HDL Planner user interface. Save the file as averager.vhd.

Synthesizing the Design

Once the design is created, it can be synthesized using Exemplar Logic's Leonardo software.

Invoke the Leonardo synthesis tool.



Run the entire flow from this one condensed page. Specify your source file(s), technology and desired frequency, then press Run Flow.

Technology: Atmel — AT40K, AT6K02, AT6K04, AT94K
Input: Averager.vhd
Clock Frequency: Mhz

```
Info: Attempting to checkout a license to
Info: License passed
Session history will be logged to file 'D:
Info, Working Directory is now 'D:\SystemC
Info: system variable EXEMPLAR set to "d:\
Info: Loading Exemplar Blocks file: d:\Sys
Messages will be logged to file 'D:/System
LeonardoSpectrum Level 1 Atmel - v20001b_e
Copyright 1990-2000 Exemplar Logic, Inc.

--
-- Welcome to LeonardoSpectrum level 1 Atm
-- Run By
-- Run Started On Wed May 15 08:57:49 Hour
|--
```

Ready | Working Directory: ...\AVERAGER\VHDL | Line 16

Leonardo Spectrum

Click open files and select averager.vhd

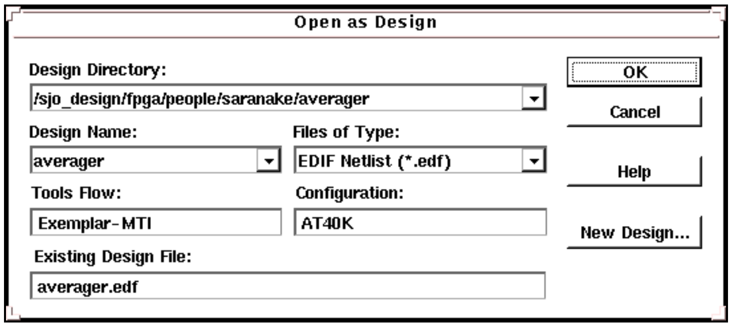
Click Run Flow

An averager.edf file which can be read into Figaro will be created.

Netlist Import

Once the netlist is created by synthesis, it can be imported into Figaro for gate level placement and routing.

On the Flowbar, press the *Open* button. Select *Open as Design* dialog box.



Design Directory: /sjo_design/fpga/people/saranake/averager

Design Name: averager

Files of Type: EDIF Netlist (*.edf)

Tools Flow: Exemplar-MTI

Configuration: AT40K

Existing Design File: averager.edf

Open as Design Dialog Box

Set the *Files of Type* field to *EDIF Netlist (*.edf)*. This will display the string "*averager.edf*" in the *Existing Design File* field. Dismiss the dialog box by pressing the OK button.

Placement and Routing

Press the *Compile* button to automatically complete the placement and routing of the design.

Specify the Atmel part by clicking on the *Parts* button. The Figaro interactive tool allows pin preplacement and pin lock assignments on the device chosen. Select the appropriate part to fit the design.

Initiate placement and routing by clicking on the *Compile* button. The four phases are: *Initial Placement*, *Optimized Placement*, *Initial Route*, and *Optimized Route*. For more detailed instructions on how to select the Atmel part and implement a design in Figaro, please refer to the "Figaro" section of this *Tutorial*.

A complete listing of the averager.vhd and averager.v files can be found in the following directories:

/SystemDesigner/examples/at6k/hdlplan/verilog-sy or /atmel/examples/at40k/hdlplan/verilog-sy   (for synopsys)

/ SystemDesigner /examples/at40k/hdlplan/verilog-ex   (for exemplar)

Exemplar

This *Tutorial* outlines the methodology for optimizing VHDL and Verilog designs using Exemplar Logic's Synthesis System in the workstation environment. A method of integrating components created by the *Macro Generators* and *Module Generators* from Exemplar's synthesis tool is also outlined. The *Macro Generators* instantiated in the VHDL/Verilog design and the components inferred by Exemplar's *Module Generators* are automatically identified by Figaro, which then creates an area or speed efficient hard macro layout. The results are used to create more efficient placement and routing in Figaro.

The design flow suggested in this *Tutorial* spans across two tools: Figaro is responsible for the design management and back end layout, while Exemplar Logic's Galileo Extreme or Leonardo are used for synthesis and optimization of the design. A suitable simulator such as the ModelSim from Model Technology Inc. (MTI) is required if functional simulation and/or post-layout simulation of the optimized netlist is desired.

The following tools must be installed and running before starting this *Tutorial*.

1. Figaro
2. Exemplar Logic's Leonardo with the Atmel Exemplar library
3. Simulator (optional) Model Technology VHDL simulator with the Atmel MTI Vital simulation library

Design Flow

The design flow is briefly summarized below.

System Setup (Figaro)

- Sets up the design directory and libraries

Design Entry (Figaro)

- Enters the design in VHDL or Verilog

Design Synthesis (Exemplar)

- Sets up the software environment for synthesis
- Reads in the design
- Optimizes the design
- Writes the post-synthesis VHDL/Verilog netlist

Functional Simulation (Any available simulator. MTI is used in this *Tutorial*)

- Verifies functionality of the design

EDIF Netlist Generation

- Converts the post-synthesis structural VHDL/Verilog to an EDIF netlist

Layout Generation (Figaro)

- Invokes automatic/manual placement and routing
- Writes delay values for back annotation

Post-layout Simulation (Any available simulator, MTI is used in this *Tutorial*)

- Reads in delay file for simulation

Device Programming (Figaro)

- Outputs Bitstream file
- Downloads onto Atmel part (PC only)

Tutorial Example

To work through the *Tutorial*, a design describing a moving point “averager” circuit is used. Both Verilog and VHDL versions of the design are provided.

The “averager” has input and output ports which are 8 bits wide. The circuit smoothes out a noisy waveform arriving at an input port by averaging the adjacent 8 values of the signal amplitude, and generating the averaged waveform at the output port.

The “averager” example is partitioned into design components, each of which can be generated automatically using the *Macro Generators* module in Figaro. While synthesis will still produce gate level connectivity for the components, Figaro will select the hard layouts of these components from the layout library. In this way, a truly technology independent VHDL design can be written for simulation and retargeting, and the optimized netlist can use the best layout implementations to produce superior results.

Leonardo Spectrum can infer ModGen Components for arithmetic and relational operators in a design from the Module Generation Library provided for the Atmel FPGA. Atmel FPGA ModGen Library components are black boxes and hence designs using ModGen components cannot be simulated until the design netlist is read into Figaro. Once the design netlist is read into Figaro, the ModGen components are automatically identified and the *Macro Generators* (also called MGI) dialog box will be brought up. The MGI dialog box lists all the ModGen components and their parameters like width, function etc. In the MGI dialog box, users can change layout related parameters like area/speed optimization, layout folding etc. and create a hard layout for the ModGen components.

The VHDL files of the example design are located in /SystemDesigner/examples/at6k/exemplar/averager/vhdl or /SystemDesigner/examples/at40k/exemplar/averager/vhdl directory which come as part of the Figaro installation.

The Verilog files of the example design are located in /SystemDesigner/examples/at6k/exemplar/averager/verilog or /SystemDesigner/examples/at40k/exemplar/averager/verilog directory which come as part of the Figaro installation.

Creating and Setting up the Design Directory

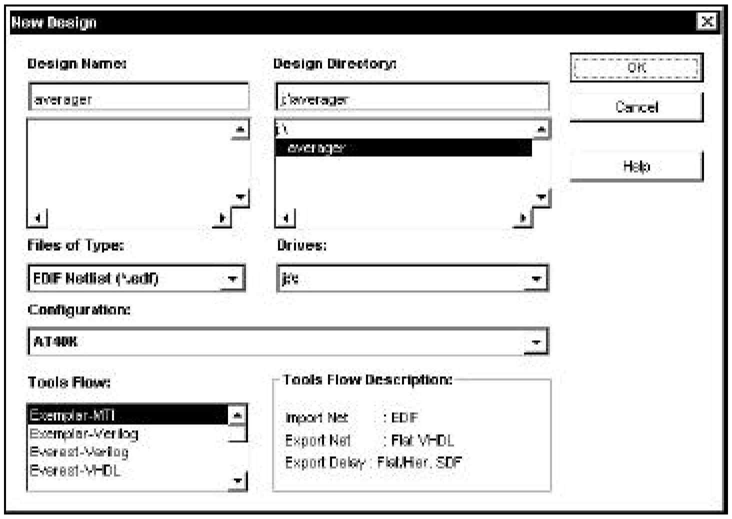
In this first step, the user creates a design directory and assigns a CAE platform to the design so that tasks meaningful to the chosen platform can be applied automatically during processing.

To create a new design, select the icon on the Flowbar or use *File>Design Setup* from the menu.

Select the *New Design* button to open the *New Design* dialog box.

Figaro will input the “averager” design in EDIF format for use. Use the steps outlined as follows to set up the environment. The dialog box will resemble that shown below.

1. Under *Design Directory*, enter the path where the VHDL/Verilog files are to be located.
2. Enter the *Design Name* as “*averager*”.
3. Set *Configuration* to *AT6K* or *AT40K*.
4. Set *Tools Flow* to *Exemplar-MTI* if the design source is VHDL, or to *Exemplar-Verilog* if the design source is Verilog.
5. Under *Files of Type*, select *EDIF Netlist* (**.edf)* and press OK.
6. Press OK again to complete the *Design Setup*.



Design Name: averager

Design Directory: j:\averager

j:\
averager

Files of Type: EDIF Netlist (*.edf)

Drives: j:

Configuration: AT40K

Tools Flow:
Exemplar-MTI
Exemplar-Verilog
Everest-Verilog
Everest-VHDL

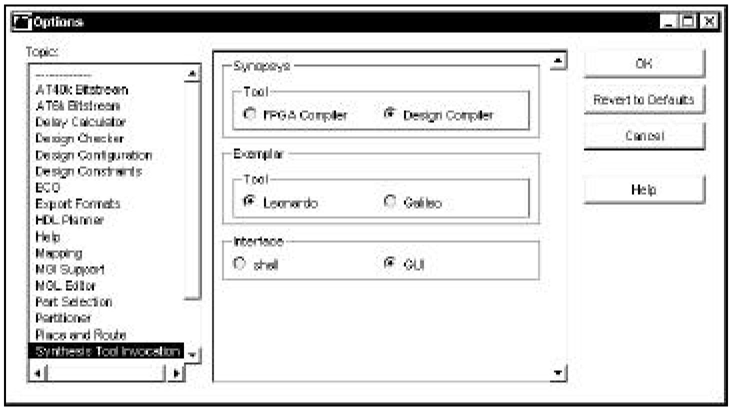
Tools Flow Description:
Import Net : EDIF
Export Net : Flat VHDL
Export Delay : Flat/Hier. SDF

New Design Dialog Box

NOTE Leonardo Spectrum is the default synthesis tool when Figaro is installed.

The synthesis tool Leonardo Spectrum, can be selected using *Options>Options* from the menu.

1. Select *Synthesis Tool Invocation* from the *Topic* list in the *Options* dialog box as shown below.



Topic:

AT40K Bitstream
AT6k Bitstream
Delay Calculator
Design Checker
Design Configuration
Design Constraints
ECO
Export Formats
HDL Planner
Help
Mapping
MGI Support
MGL Editor
Part Selection
Partitioner
Place and Route
Synthesis Tool Invocation

Synopsys
Tool: FPGA Compiler [✓ Design Compiler]

Exemplar
Tool: [✓ Leonardo] Galileo

Interface: shell [✓ GUI]

Options Dialog Box

2. Select the appropriate synthesis tool from the Exemplar group in the dialog box and press OK

Library Setup

Figaro supports application of a user library in which hard macro layouts can be stored for use in current or future designs. The *Macro Generators* create such hard layouts and require a library for storage of the information. Besides the hard layouts, simulation information and other important files are stored in this library directory. A library for the “averager” design can be created as follows:

1. Select the *Library>Library Setup* module to activate the *Library Setup* dialog box.
2. Press the *Add Before* button in the *Library Search Path* panel to activate the *Add Library and Path* dialog box.
3. Enter the *Library Name* as “*user*.lib” and press OK.
4. Proceed to create the library by pressing OK
5. Press OK again to complete the *Library Setup*.
6. Confirm *Updated Cache Library* by pressing OK

Component Generation

The “averager” design makes use of a number of components that can be automatically created using the *Macro Generators* in Figaro. This function produces a component or design that is more compact in size and efficient in speed.

Examination of the averager.vhd file will reveal a very simple method of incorporating the *Macro Generators* in VHDL. The process involves identifying the components that are going to be part of the design and inserting them one level deeper in the design hierarchy. Care must be taken to ensure correspondence between the port names in the VHDL/Verilog designs and the macros created by the *Generators*. It is also imperative to maintain the component names in the VHDL/Verilog design during macro generation.

The following components must be generated for the “averager” design via the *Macro Generators* button on the Flowbar. This will bring up the *Generator Notebook*. Choose the category of macros using the bottom tabs and the desired *Generator* using the right side tabs.

To generate the macros one by one:

1. Choose the *Generator* according the type of macro.
2. Input the parameters for the macro.
3. Click *Generate* to start creating the macro. A statistics dialog box about that macro will appear upon completion.
4. Do the same procedures for all the other macros.
5. Click *Cancel* when all macros have been generated.

The macro names and their appropriate parameters are mentioned below. All other parameters should be left to their default settings.

Generator Register

Register Type = Flip-Flop D-type,

Macro Name= reg8,

Width=8.

Generator Register

Register Type=Flip-Flop D-type,

Macro Name= reg8en,

Width=8,

Register Bank Enable=Group Enable.

The *Macro Generators* are designed to exploit Atmel’s register rich FPGA architecture, and produce a component that is more compact in size and efficient in speed. The hard layouts for the above-generated components are stored in the user defined library. They will be used by Figaro during *Compile*, when the EDIF netlist of the design is being processed.

Design Entry

Once a design has been set up, the VHDL or Verilog files of the design description must be set up in the Design Directory for synthesis to proceed. For the "averager" design, these files have been copied to the Design Directory from /SystemDesigner/examples/at6k/exemplar or /SystemDesigner/examples /at40k/exemplar.

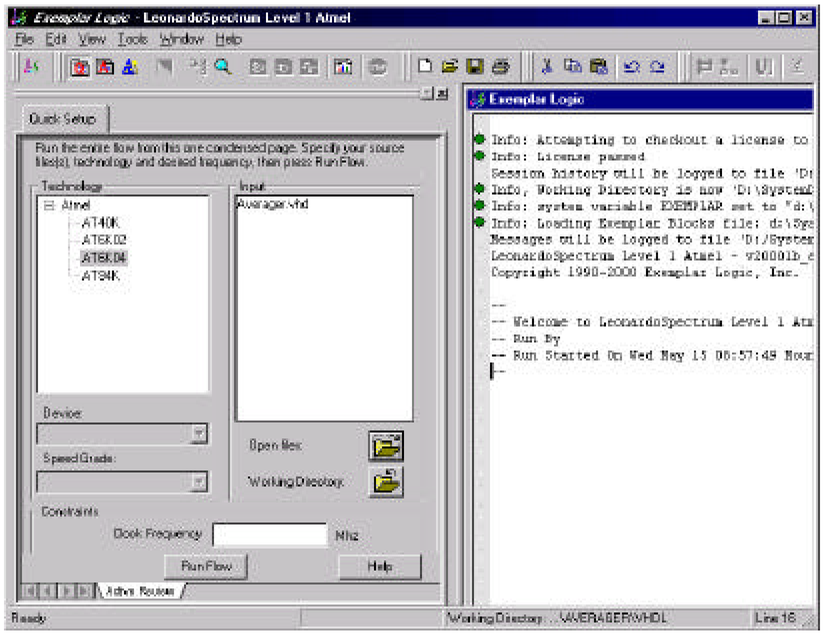
Exemplar's Leonardo can now be brought up by clicking on the *Synthesis Tool Invocation* push button from the Flowbar on the IDS desktop.

Optimization and Netlist Generation

Once the design files are created, optimization of the completed design can proceed hierarchically in a top-down fashion.

Once the design is created, it can be synthesized using Exemplar Logic's Leonardo software.

1. Invoke the Leonardo synthesis tool.



Run the entire flow from this one condensed page. Specify your source file(s), technology and desired frequency, then press Run Flow.

Input: Averager.vhd

```
Info: Attempting to checkout a license to
Info: License passed
Session history will be logged to file 'D:
Info, Working Directory is now 'D:\SystemD
Info: system variable EXEMPLAR set to "d:\
Info: Loading Exemplar Blocks file: d:\Sys
Messages will be logged to file 'D:/System
LeonardoSpectrum Level 1 Atmel - v20001b_e
Copyright 1990-2000 Exemplar Logic, Inc.

--
-- Welcome to LeonardoSpectrum Level 1 Atm
-- Run By
-- Run Started On Wed May 15 08:57:49 Hour
--
```

Leonardo Spectrum

2. Click open files and select averager.vhd
3. Click Run Flow

An averager.edf file that can be read into Figaro will be created.

Functional Simulation

During the synthesis phase a structural VHDL/Verilog netlist can be output from the Exemplar tools. Pre-layout simulation can be done on the post-synthesis Verilog/VHDL netlist generated by the synthesis tool. The Verilog simulation libraries are provided along with IDS and are installed in /SystemDesigner/lib/verilog/at40k. The VITAL libraries provided with IDS are installed in /SystemDesigner/lib/mti/at6k02 and /SystemDesigner/lib/mti/at6k04 or /SystemDesigner/lib/mti/at40k

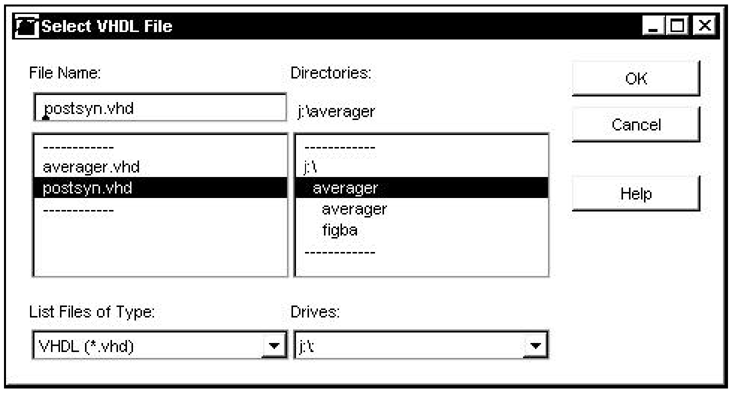
Netlist Creation

The structural VHDL/Verilog netlist created during the synthesis phase of the design flow can be converted into an EDIF netlist using Exemplar's synthesis tool.

1. Click on the *Netlist* icon on the Flowbar.
2. Select the *VHDL or Verilog* (structural VHDL/Verilog netlist from the synthesis tool) file to be converted to EDIF.

NOTE When translating a structural Verilog netlist into EDIF, please note that Exemplar's Leonardo Spectrum will generate errors if any hierarchical modules in the Verilog netlist are not fully defined.

3. Click on OK in the *Select VHDL/Verilog File* dialog box to create the EDIF netlist.



File Name: postsyn.vhd
Directories: j:\averager

```
------------
averager.vhd
postsyn.vhd
------------
```

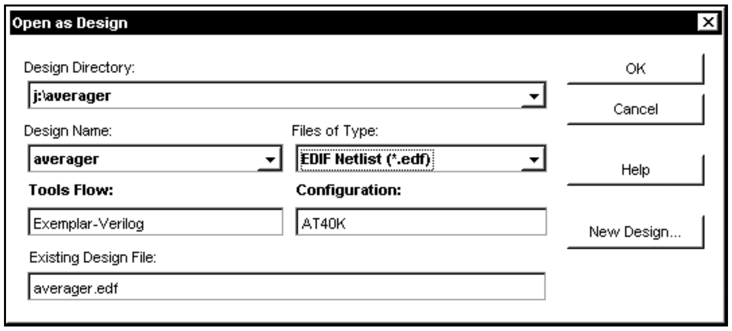
```
------------
j:\
 averager
   averager
   figba
------------
```

List Files of Type: VHDL (*.vhd)
Drives: j:\

Select VHDL File Dialog Box

Netlist Import

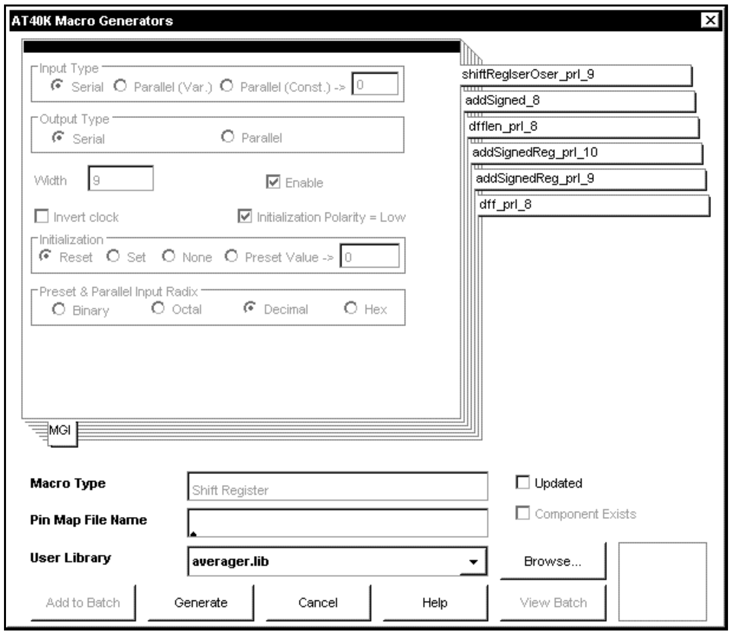
Once the design is verified with *Pre-layout Simulation* and netlisted, it can be imported into Figaro for compilation. To read the averager.edf file into Figaro, click on the *Open* button in the tool bar and proceed with *Open as Design*. Specify *Files of Type* to be *EDIF Netlist (*.edf)* and verify that the *Existing Design File* is *averager.edf.*



Design Directory: j:\averager
Design Name: averager
Files of Type: EDIF Netlist (*.edf)
Tools Flow: Exemplar-Verilog
Configuration: AT40K
Existing Design File: averager.edf

Open as Design Dialog Box for "averager" in VHDL

Click on OK for Figaro to load the EDIF file and generate a database file for the design. While loading the EDIF file, Figaro identifies the ModGen components inferred by the synthesis tool (if any) and brings up a *Macro Generators Interface* dialog box when applicable. The box displays the macro names as notebook tabs. The user can change the layout-related parameters of the ModGen components. For more details on the list of ModGen components supported by the *Macro Generators* and *MGI*, please refer to "Macro Generators Interface" chapter in the *Technical Reference.* The MGI dialog box for the example in this *Tutorial* is shown below.



Input Type: [✓ Serial] Parallel (Var.) Parallel (Const.) -> 0

Output Type: [✓ Serial] Parallel

Width: 9

[✓ Enable]

Invert clock

[✓ Initialization Polarity = Low]

Initialization: [✓ Reset] Set None Preset Value -> 0

Preset & Parallel Input Radix: Binary Octal [✓ Decimal] Hex

Macro Type: Shift Register

Pin Map File Name:

User Library: averager.lib

Updated

Component Exists

MGI Dialog Box

The following is a brief description of the various buttons in the MGI dialog box:

Component Exists

This box will be checked if a macro with the same name already resides in the library. This may happen if the design has gone through one iteration. If the component exists and it matches the displayed parameters, its generation is skipped.

Updated

This box is checked if the default settings of the macro have been changed. By default, the user interface displays options per the following guideline. If the component exists in the library, the options that it was run with are displayed. If the component does not exist in the library, the default settings that minimize the component area will be chosen.

Generate

This button is used to proceed with the generation of the macros.

Cancel

This button is used to skip the generation of the macros. If this button is pressed, the design will not contain the hard layout of the macro. Instead, its soft implementation will be chosen.

Click on GENERATE to create the hard macros for the listed components and proceed with the EDIF netlist processing for the design.

NOTE

To automatically generate hard macros for the identified *Macro Generators* components without bringing up the graphical user interface, use the *Options>Options* menu. In the *Options* dialog box, check the *Automatically Generate Components* option under *MGI Support*.

Mapping

In order to optimize the design and collect better synthesis results, it is recommended that mapping be performed. The netlist from the synthesis tools contains functional macros that can be packed into Look Up Tables by the technology mapper in Figaro. After opening the design, use *Options>Options>Mapping* and activate the *Mapping Enabled* option. Then push the *Map* button on the Flowbar to map the design. Pleases refer to the Mapping section in the “Figaro” chapter of this *Tutorial* for more details about this process and the options available.

Placement and Routing

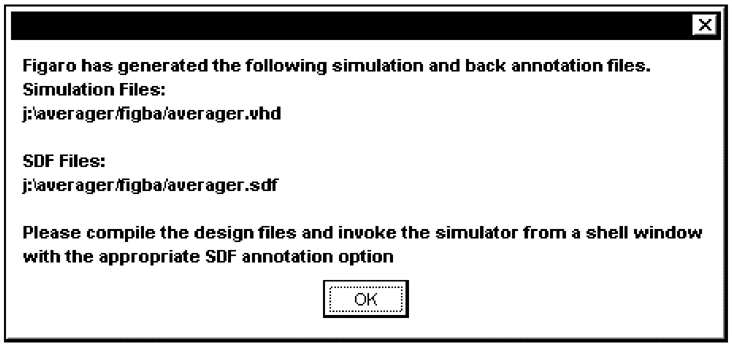
1. Specify an Atmel part by clicking on the *Parts* button. The Figaro interactive tool allows pin preplacement and pin lock assignments on the device chosen. Select the appropriate part to fit the design.
2. Use *Window>New Compile Window* to bring up the *Compile* window. Step-by-step placement and routing can be performed by clicking corresponding buttons in this window.
3. Perform initiate placement and routing by clicking on the *Compile* button. The four phases feature *Initial Placement*, *Optimized Placement*, *Initial Route*, and *Optimized Route*. The last button can be either *Bitstream* (for designs) or *Check-in* (for macros). Depending on the initial design setup, Figaro will create a bitstream file for download to the device, or instantiate the component and store it in the user-specified library.

Details on the *Parts Assembler* and *Compile* modules are covered in the “Figaro” chapter in the *Tutorial*.

Post-layout Simulation

Post-layout Simulation can be performed by using the delay values extracted during *Compile*. The back annotation information is then read into the design in the specified export file format. Figaro determines the file types to be exported based on the selected tool flow and design implementation (whether the design is mapped or not, and if the implementation is for one or more chips). If the design is mapped or partitioned into multiple chips, a flat VHDL (or Verilog) netlist and the corresponding SDF netlist will be created. Otherwise a hierarchical SDF file will be generated and back annotated to the structural post-synthesis VHDL (or Verilog) netlist from the synthesis tool. However even if the design is implemented on a single chip and not mapped, the user can export a flat simulation netlist by using *File>Export*.

Click on the *Post-layout Simulation* button on the Flowbar. This will export the required simulation netlist and a SDF back annotation file into a directory called figba in the current design directory. A pop up dialog box will appear as shown below.



```
Figaro has generated the following simulation and back annotation files.
Simulation Files:
j:\averager/figba/averager.vhd

SDF Files:
j:\averager/figba/averager.sdf

Please compile the design files and invoke the simulator from a shell window
with the appropriate SDF annotation option
```

Exported Netlists Dialog Box

The box will give the name of the exported netlists. These netlists have to be compiled and the simulator invoked on them from a command shell outside Figaro.

NOTE For Verilog users, if *Post-layout Simulation* is to be performed after *Compile* is completed, specify the export format of *flat Verilog* and *flat SDF* using *File>Export*. Figaro will generate a *design*.v file and SDF file containing delay information.

DownLoad Bit Stream

For more information on downloading and bitstream utilities, refer to the “Device Programming” and “Figaro” chapters in the *User’s Guide* and *Tutorial* respectively.

NOTE The *DownLoad* option will only download the bitstream file from a PC only. It is not available to Workstation users.

Synopsis

This tutorial outlines the methodology for optimizing HDL based designs with the Synopsys Design compiler or FPGA Compiler. Atmel supplies a synthesis library, at40k.syn, for the AT40K FPGA series.

Design Flow

Figaro supports two design flows using Synopsys tools to compile the design. In the first flow, Synopsys tools are used to perform synthesis, and a VITAL simulator is used to perform simulations. The second flow is identical, except any Verilog simulator is used to perform design verification. These flows can be specified by selecting the appropriate *Tools Flow* while setting up the design. The output simulation files, generated by Figaro, will depend on the selected flow. A typical flow is given below.

System Setup (Figaro)

- Sets up the design directory and libraries

Design Entry (Command Shell)

- Enters the design in VHDL or Verilog

Functional Simulation (VITAL or Verilog)

- Verifies functionality of the design

Design Synthesis (Synopsys)

- Sets up the software environment
- Reads in the design
- Optimizes the design
- Writes the netlist

Gate Level Simulation (VITAL or Verilog)

- Verifies functionality of the design

Layout Generation (Figaro)

- Invokes automatic/manual placement
- Invokes automatic/manual routing
- Writes delay values for back annotation

Post-layout Simulation (VITAL or Verilog)

- Reads in delay file for simulation

Device Programming (Figaro)

- Creates bitstream
- Downloads onto Atmel part

System Setup

Atmel Library

A library for AT40K FPGA, at40k.syn, is supplied by Atmel. This library is located in the /SystemDesigner/install/lib/synopsys directory.

Environment Variables

The synopsys_dc.setup file contains the environment variables which set up the execution environment during synthesis. A sample template of this file is provided in the directory /SystemDesigner/examples/at40k/synopsys/averager/vhdl, where all other tutorial files are kept. Optionally, this file can also be kept in the directory /SystemDesigner/lib/synopsys. This file can be used by the Design Compiler as well as the FPGA Design Compiler.

To use this file, copy it from either of the above locations. Make substitutions as needed for the paths of the various libraries. Copy the file to either the home directory as .synopsys_dc.setup or the current directory as .synopsys_dc.setup.

Some variables of interest are search_path and link_library, target_library. The following is a description of these variables.

search_path This variable defines the directory search order for the various libraries. It is functionally equivalent to the UNIX path variable. It should be set to:

```
search_path = { Synopsys libraries,
/install/atmel/lib/synopsys }
```

The first directory should point to the libraries that are installed with the Synopsys software, the second corresponds to a path where the Atmel synthesis library (at40k.syn) is located.

link_library This variable is used to link the gate level components in the source HDL to the target technology library so that those components can participate functionally during synthesis.

```
link_library= {at40k.syn}
```

target_library This variable is used to set the target technology in which the output gate level netlist is to be produced.

```
target_library= {at40k.syn}
```

Edifout variables:

These variables are used to produce an EDIF netlist in the desired format.

```
edifout_netlist_only = “true”
edifout_power_and_ground_representation = “cell”
edifout_merge_libraries = “false”
```

Example Design

For this tutorial, the “averager” design will be used. This design implements a waveform smoothing function represented by:

$$y(n) = \frac{1}{N}\sum_{(i=0)}^{(N-1)} x(n-i)$$

The smoothing function is used in many DSP applications to filter out high frequency spikes. These spikes are the source of noise commonly found in communication channels. They can be eliminated by taking the moving average of sample values arriving at the input of the system.

The design in this example is an “averager” circuit with eight moving points. The hardware specific details of the design are described below.

Interface The “averager” design has eight bit input and output data lines. The design also has a clock pin and a reset pin.

Assumptions and Other Details Only positive values can appear on the input data lines (the input is assumed to be level-shifted). During the implementation of the system, a valid output waveform can be expected on the 9th clock. However, since the outputs of the last and intermediate stages are registered, the actual output waveform starts occurring from the 11th clock cycle onwards.

Design Files

The “averager” design consists of several design files:

- Design vhdl files
- Test bench file
- Input stimulus file
- Synopsys simulation and synthesis scripts

The files are kept in the /SystemDesigner/atmel/examples/at40k/synopsys/averager/vhdl directory. (A Verilog version of the “averager” design is also available in the /SystemDesigner/atmel/examples/at40k/synopsys/averager/verilog directory.) Perform the following commands to copy them to the local directory:

```
$ mkdir averager
$ cd averager
$ cp /SystemDesigner/atmel/examples/at40k/synopsys/averager/vhdl/* .
```

The following is a brief description of the design files.

averager.vhd   Contains the top-level design. It has instantiations of the other modules of the design.

averager_tb.vhd   This module contains the test bench file for the “averager” design (the file may not be needed for this tutorial).

averager_tb.vhg   This module contains the test bench file for the “averager” design. This test bench is to be used with Post-layout gatelevel netlist only (the file may not be needed for this tutorial).

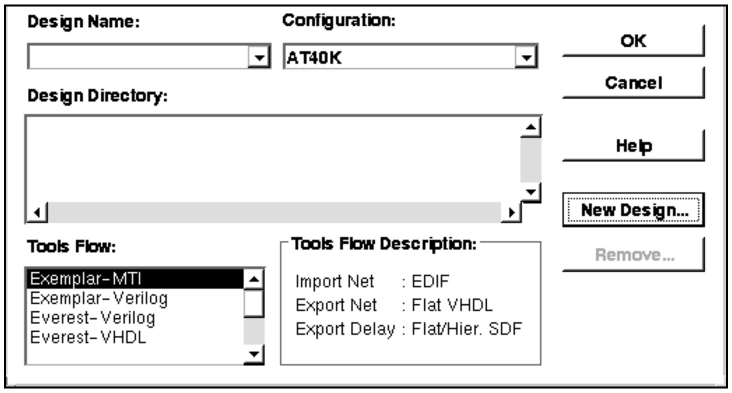
Invoking Figaro

Figaro can be started in the averager directory by typing “figaro” on the command-line.

```
$ cd averager
$ figaro
```

Figaro Design Setup

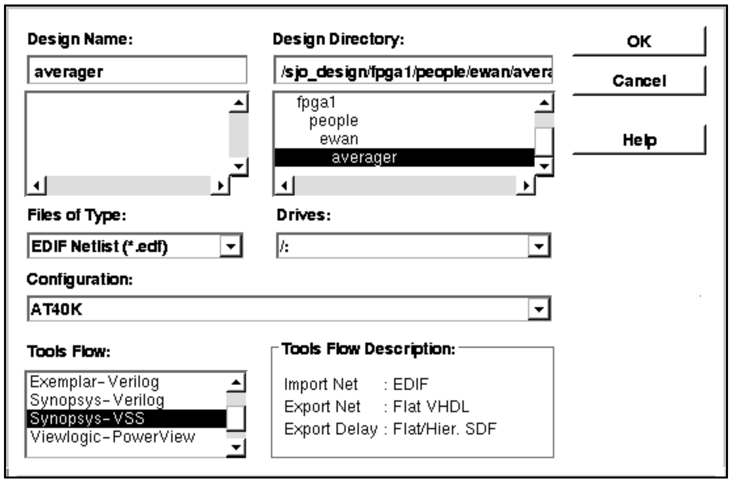
In the Figaro window, use the icon or *File>Design Setup* menu to set up the design and select its associated *Tools Flow*. The *Design Directory Setup* dialog box will be brought up.



Import Net : EDIF
Export Net : Flat VHDL
Export Delay : Flat/Hier. SDF

Design Setup Dialog Box

To create a new design click on the *New Design* button. The *New Design Directory* dialog box is shown below.


Design Name: averager
Design Directory: /sjo_design/fpga1/people/ewan/avera
fpga1
people
ewan
averager
Files of Type: EDIF Netlist (*.edf)
Drives: /:
Configuration: AT40K
Tools Flow:
Exemplar-Verilog
Synopsys-Verilog
Synopsys-VSS
Viewlogic-PowerView
Tools Flow Description:
Import Net : EDIF
Export Net : Flat VHDL
Export Delay : Flat/Hier. SDF

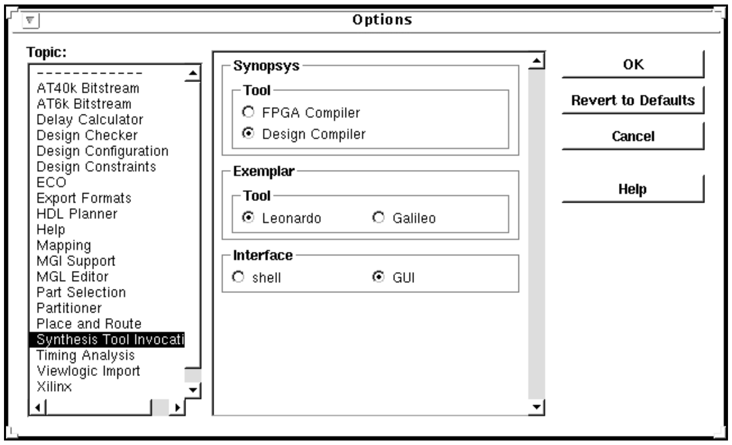
New Design Directory Dialog Box

Inside the *New Design* dialog box:

- Set *Configuration* to *AT40K*.
- Enter the *Design Name* as “*averager*”.
- Select *Tools Flow* as *Synopsys-VSS* or *Synopsys-Verilog* for the VHDL or verilog design respectively.
- Verify *Files of Type* as *EDIF Netlist*.
- Dismiss the *New Design* and *Design Directory* dialog boxes by pressing the respective OK buttons.

The synthesis tool, FPGA Compiler or Design Compiler, can be selected using *Options>Options* from the menu.

1. Select *Synthesis Tool Invocation* from the *Topic* list in the *Options* dialog box as shown below.


Options
Topic:

AT40k Bitstream
AT6k Bitstream
Delay Calculator
Design Checker
Design Configuration
Design Constraints
ECO
Export Formats
HDL Planner
Help
Mapping
MGI Support
MGL Editor
Part Selection
Partitioner
Place and Route
Synthesis Tool Invocati
Timing Analysis
Viewlogic Import
Xilinx
Synopsys
Tool
FPGA Compiler [✓ Design Compiler]
Exemplar
Tool
[✓ Leonardo] Galileo
Interface
shell [✓ GUI]

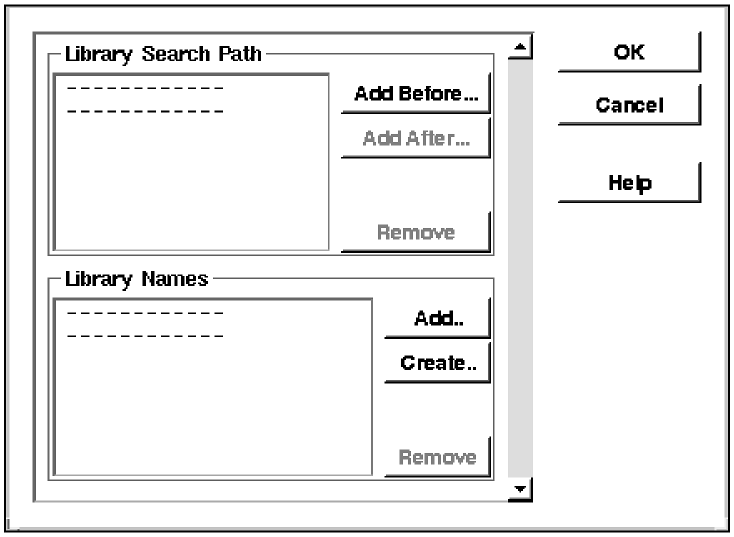
Options Dialog Box

2. Select the appropriate synthesis tool from the Synopsys group in the dialog box and press OK.

Library Setup

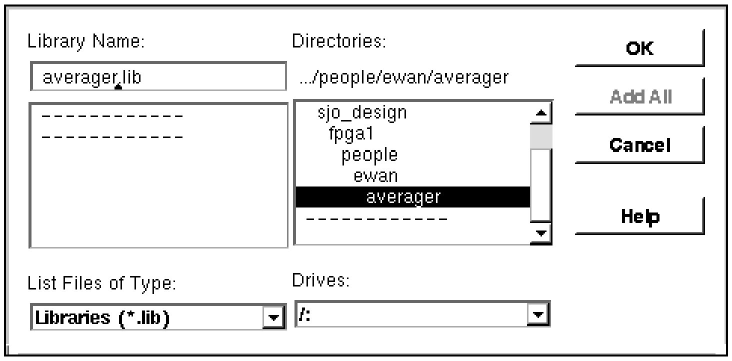
The “averager” design, once synthesized, will contain the macros inferred automatically by the Synopsys Design Compiler. These components are created automatically with the *Macro Generators.* The associated FPGA layout and simulation models will also be produced. Figaro treats these components as library modules and inserts them in the library. Consequently, a user library must be set up.

1. A design library can be established by executing the *Library>Library Setup* menu.



Library Setup Dialog Box

2. In *Library Setup*, press the *Add Before* button to display the *Add Library and Path* dialog box.



Add Library and Path Dialog Box

3. In the *Add Library and Path* dialog box, enter *Library Name* as "*averager*.lib" (or any name of choice).
4. Dismiss the box by pressing OK.
5. Proceed to create the library by pressing OK.
6. Dismiss the *Library Setup* dialog box by pressing OK.

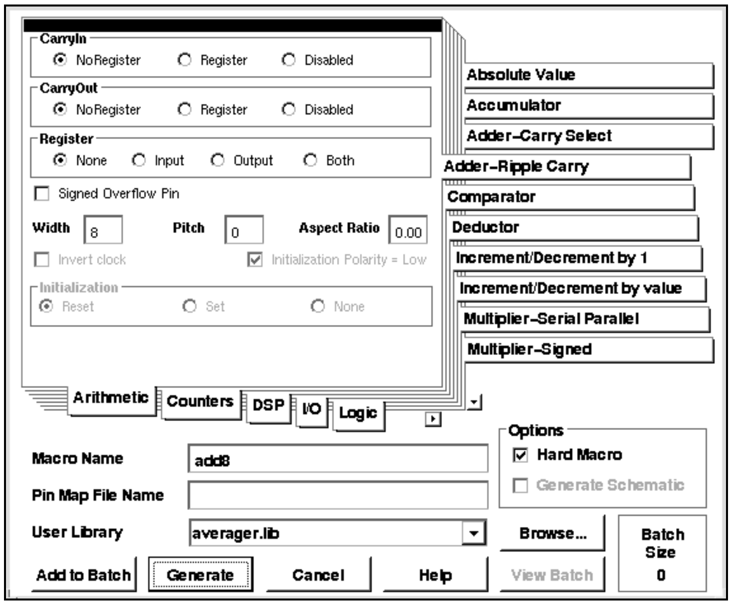
The design name has been entered as "averager" and the library name as "*averager*.lib".

Component Generation

The "averager" design makes use of a number of components that can be automatically created using the *Macro Generators* in Figaro. This macro generation function is designed to exploit Atmel's register rich FPGA architecture, and produce a component that is more compact in size and efficient in speed.

An examination of the averager.vhd file will reveal a very simple method of incorporating the *Macro Generators* in VHDL. The process involves identifying the components that are going to be part of the design and inserting them one level deeper in the design hierarchy. Care must be taken to ensure correspondence between the port names in the VHDL designs and the macros created by the *Generators*. It is also imperative to maintain the component names in the VHDL design during macro generation.

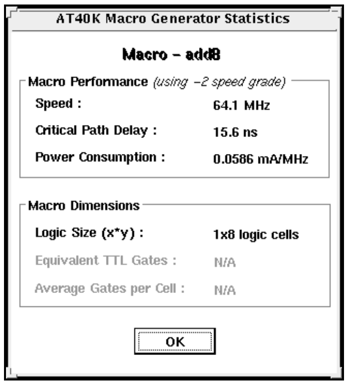
The following components must be generated for the "averager" design via the *Macro Generators* button on the Flowbar. This will bring up the *Generator Notebook* as show below. Choose the category of macros using the bottom tabs and the desired *Generator* using the right side tabs.



Macro Generators Interface Dialog Box

To generate the macros one by one:

1. Choose the *Generator* according the type of macro.
2. Input the parameters for the macro.
3. Click GENERATE to start creating the macro. A statistics dialog box about that macro will appear upon completion.
4. Do the same procedures for all the other macros.
5. Click OK when all macros have been generated.


AT40K Macro Generator Statistics

Macro – add8

Macro Performance *(using –2 speed grade)*

Speed :	64.1 MHz
Critical Path Delay :	15.6 ns
Power Consumption :	0.0586 mA/MHz

Macro Dimensions

Logic Size (x*y) :	1x8 logic cells
Equivalent TTL Gates :	N/A
Average Gates per Cell :	N/A

Statistics Dialog Box for Add8

The macro names and their appropriate parameters are mentioned below. All other parameters should be left to their default settings.

Generator Arithmetic

Macro Type = Adder-Ripple Carry,

Macro Name = add8,

Width = 8.

Generator Arithmetic

Macro Type = Adder-Ripple Carry,
Macro Name = add9reg,
CarryOut = Register,
Register = Input,
Width = 9.

Generator Arithmetic

Macro Type = Adder-Ripple Carry,
Macro Name = add10reg,
CarryOut = Register,
Register = Input,
Width = 10.

Generator Register

Register Type = Flip-Flop D-type,
Macro Name = reg8,
Width = 8.

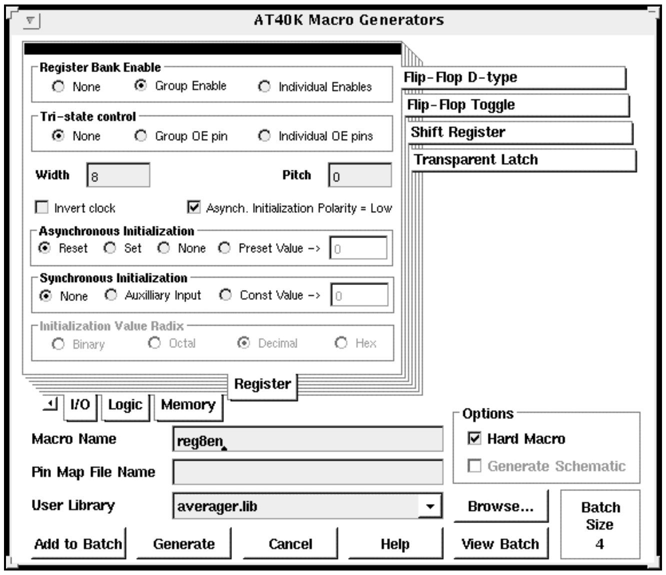
Generator Register

Register Type = Flip-Flop D-type,
Macro Name = reg8en,
Register Bank Enable = Group Enable,
Width = 8.

Generator Register

Register Type = Shift Register,
Macro Name = sr9,
Width = 9.

The hard layouts for the above-generated components are stored in the user defined library. They will be used by Figaro during *Compile*, when the EDIF netlist of the design is being processed.


AT40K Macro Generators

Register Bank Enable: None [✓ Group Enable] Individual Enables
Tri-state control: [✓ None] Group OE pin Individual OE pins
Width: 8 Pitch: 0
Invert clock [✓ Asynch. Initialization Polarity = Low]
Asynchronous Initialization: [✓ Reset] Set None Preset Value -> 0
Synchronous Initialization: [✓ None] Auxiliary Input Const Value -> 0
Initialization Value Radix: Binary Octal [✓ Decimal] Hex

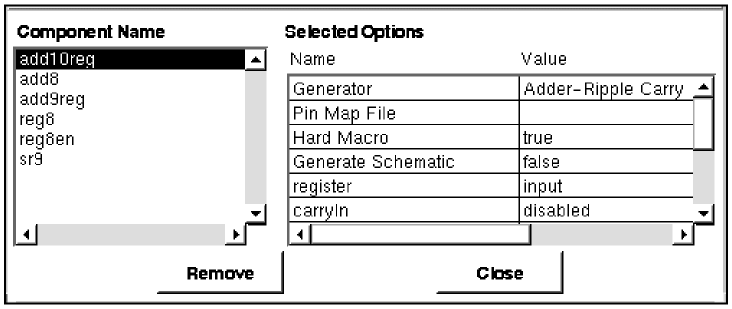
Macro Name: reg8en
Pin Map File Name:
User Library: averager.lib
Options: [✓ Hard Macro] Generate Schematic
Batch Size 4

Setting for Reg8en using Add To Batch Function

The Add to Batch Function

To use the Macro Generators *Add To Batch* function for building several macros at a time, follow the steps below.

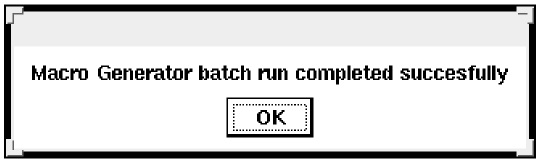
1. Input the parameters for the macro into the *Generator* page.
2. Store the macro settings in the batch by pressing *Add To Batch* after each configuration. The *Batch Size* on the lower right corner of the *Macro Generators* dialog box will get incremented by one.
3. Review the macros by pressing *View Batch* to bring up the following dialog box. To edit the list, select a macro or the properties of the macro, and press *Remove*. Press *Close* to return to MGI.



Name	Value
Generator	Adder-Ripple Carry
Pin Map File	
Hard Macro	true
Generate Schematic	false
register	input
carryIn	disabled

View Batch Dialog Box

4. Press GENERATE to start the process when all macros are configured to the appropriate settings. The *Batch Size* field will show the final count.
5 Upon successful completion, a message box as the following will appear.


Macro Generator batch run completed succesfully

Add To Batch Message Box

6. Click OK to continue.

Design Synthesis

Once the files are copied to the design directory and the library has been set up, the Design Compiler can be brought up. Press the icon on the Flowbar in the desktop to invoke the Synopsys synthesis tool. Depending on the interface chosen in the *Options>Options>Synthesis Tool Invocation* dialog box, the *Command-shell* Design compiler or the *GUI* Design Analyzer will be invoked accordingly. The Design Compiler can also be brought up in a *Shell* window with a dc_shell command.

NOTE All synthesis commands to follow are given in the Synopsys script file synth.script. The user can include that script in the Design Compiler *Shell* to expedite synthesis runs.

For Command-shell users

Use the read command to enter the HDL design into the Design Compiler environment. All design files can be read simultaneously into the Design Compiler using the following command:

```
dc_shell>read -format vhdl averager.vhd
```

NOTE The Design Compiler will issue warnings about the use of generics in the design. These warnings can be ignored. Please consult the Synopsys documentation for more details.

Optimization

Optimization of the previously read in design modules consists of: a) going through the design hierarchy, and b) optimizing the individual modules by setting the appropriate constraints. Some constraints to consider are set_max_area, set_max_delay, max_period, set_min_delay, set_clock, and create_clock. The actual optimization is done using the *Compile* command. During optimization, the Design Compiler examines many design alternatives and chooses one that meets the specified constraints. The amount of effort the Design Compiler uses can be controlled by the map_effort switch of the *Compile* command. If constraints cannot be satisfied, the Design Compiler outputs the best design and issues appropriate warnings. The commands report_area, report_timing and report_constraint are used to collect statistics on the design.

Optimization Recommendations

For inserting clock and reset buffers, the command insert_pads can be used to connect the ports to the pads of the design. The port_is_pad attribute must be attached to the ports for the insert_pads command to work correctly.

To connect global clock and global reset signals to pads, two special buffers GCLKBUF and RSBUF are provided in the libraries. These buffers must be attached to global clock and reset pins of the FPGA.

The Design Compiler can automatically connect GCLKBUF to a global clock signal of the FPGA when the command insert_pads is used. To attach RSBUF to a global reset, the following command must be used: (for the exact command used in this example design, please refer to the section *I/O Pad Synthesis* below.)

set_pad_type -exact rsbuf *GlobalReset*

Attaching GCLKBUF and RSBUF buffers are mandatory to ensure correct back annotation of the delay data during Post-layout verification.

Performing Optimization

Once the design files are read in, optimization of the complete design can proceed hierarchically. Performing optimization hierarchically is not mandatory but recommended. This allows strict control over the cells that are reported in the output netlist (e.g. the user may want to preserve the counter in the output netlist and substitute its hard implementation during Place and Route).

The sequence in which synthesis of the components should proceed is illustrated below.

Optimization of the Leaf Nodes

Use the command current_design to set sr9 as the active design in the Compiler as follows:

```
dc_shell> current_design sr9
```

The sr9 design may now be optionally optimized by specifying the constraints as follows:

1. dc_shell> set_max_area 0.0
2. dc_shell> compile -map_effort high

On completion of optimization, the Design Compiler will store a gate level netlist of sr9 in its internal database.

Compilation of leaf designs add8, add9reg, add10reg, reg8, and reg8en can be performed similar to the sequence mentioned above.

Optimization of the Non-Leaf Nodes

The optimization of the non-leaf nodes can proceed in the same way except the attribute dont_touch should be set on all the leaf components of the design. For the initadd module, the following sequence of commands can be taken.

1. dc_shell> current_design initadd
2. dc_shell> set_dont_touch {add8 reg8}
3. dc_shell> set_max_area 0.0
4. dc_shell> compile -map_effort high

Similarly, optimization of the top-level design "averager" can proceed in the following fashion.

1. dc_shell> current_design averager
2. dc_shell> set_dont_touch {initadd add9reg add10reg sr9 reg8en}
3. dc_shell> set_max_area 0.0
4. dc_shell> compile -map_effort high

I/O Pad Synthesis

The final phase of optimization is to connect I/O pads to the pins of the "averager" design.

1. To do this, the attribute set_port_is_pad must be added to all pins that are connected to the I/O buffers as follows.
   ```
   dc_shell> set_port_is_pad find(port)
   ```
2. After the set_port_is_pad attribute is set, a global reset pin must be connected to the RSBUF buffer using the following command.
   ```
   dc_shell> set_pad_type -exact rsbuf rst
   ```
3. Pads are inserted using the following command.
   ```
   dc_shell> insert_pads
   ```

Netlist Generation

Two netlists must be written: one in EDIF to act as the interface between the Design Compiler and Figaro for placement and routing, and another from a choice of formats to prepare for simulation. The Design Compiler command write can be used to write the design into the output file.

For the "averager" design, these netlists can be generated using the following two commands.

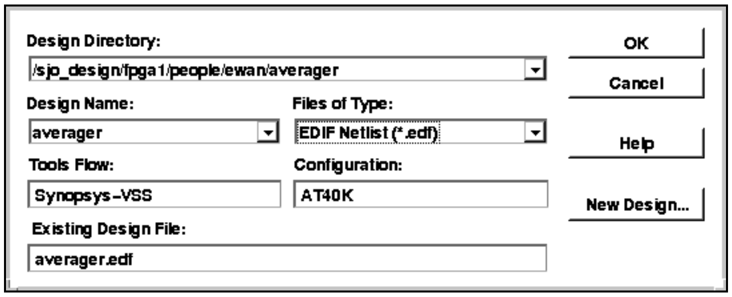
1. To write the netlist for Figaro, the following options must be specified:
   ```
   dc_shell> write -format edif -hierarchy -output averager.edf
   ```
2. To write the netlist for *Functional* and *Post-layout Simulation* the following options must be specified:
   ```
   write -format vhdl -hierarchy -output averager.vhg
   ```

Please refer to the chapter "Simulation-Synopsys" in this *Tutorial* for details on gate level simulation using the VITAL libraries.

Netlist Import

Once the netlist is created by synthesis, it can be imported into Figaro for gate level placement and routing.

On the Flowbar, press the *Open* button and choose to *Open as Design*. This will bring up the *Open as Design* dialog box.



Design Directory: /sjo_design/fpga1/people/ewan/averager

Design Name: averager

Files of Type: EDIF Netlist (*.edf)

Tools Flow: Synopsys–VSS

Configuration: AT40K

Existing Design File: averager.edf

Open as Design Dialog Box

Set the *Files of Type* field to *EDIF Netlist (*.edf)*. This will display the string "*averager.edf*" in the *Existing Design File* field. Dismiss the dialog box by pressing the OK button.

While loading the EDIF file, Figaro recognizes the user defined components that have been generated by the Macro Generators and applies those optimized hard macros from the user library, *averager.lib*, to the design .

For the "averager" design, Figaro has identified the following macros:

```
add10reg
add9reg
add8
reg8en
reg8
sr9
```

Mapping

In order to optimize the design and obtain better synthesis results, it is recommended that mapping be performed. The netlist from the synthesis tools contains functional macros that can be packed into Look Up Tables by the technology mapper in Figaro. After opening the design, use *Options>Options>Mapping* and activate the *Mapping Enabled* option. Then push the *Map* button on the Flowbar to map the design. Pleases refer to the Mapping section in the "Figaro" chapter of this *Tutorial* for more details about this process and the options available.

Placement and Routing

Specify the Atmel part by clicking on the *Parts* button. The Figaro interactive tool allows pin preplacement and pin lock assignments on the device chosen. Select the appropriate part to fit the design. The "averager" design should fit on an AT40K10 or bigger device. Click on the *Add* button and then OK to finish part selection.

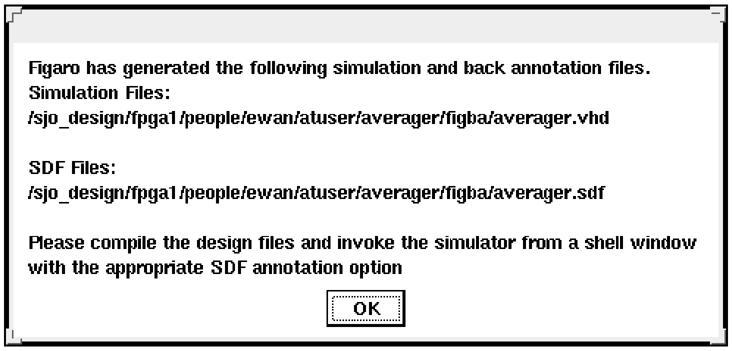
Use *Window>New Compile Window* to bring up the *Compile* window. Initiate placement and routing by clicking on the *Compile* button. The four phases in placement and routing are: *Initial Placement*, *Optimized Placement*, *Initial Route*, and *Optimized Route*. Step-by-step placement and routing can be performed by clicking on the buttons in sequence. The last button can be either *Bitstream* (for designs) or *Check-in* (for macros). Depending on the initial design setup, Figaro will create a bitstream file for download to the device, or instantiate the component and store it in the user-specified library.

For more details on how to select the Atmel part and implement a design in Figaro, please refer to the "Figaro" section of this *Tutorial*.

Post-layout Simulation

Post-layout Simulation can be performed by using the delay values extracted during *Compile*. The back annotation information is then read into the design in the specified export file format. Figaro determines the file types to be exported based on the selected tool flow and design implementation (whether the design is mapped or not, and if the implementation is for one or more chips). If the design is mapped or partitioned into multiple chips, a flat VHDL (or Verilog) netlist and the corresponding SDF netlist will be created. Otherwise a hierarchical SDF file will be generated and back annotated to the structural post-synthesis VHDL (or Verilog) netlist from the synthesis tool. However even if the design is implemented on a single chip and not mapped, the user can export a flat simulation netlist by using *File>Export*.

Click on the *Post-layout Simulation* button on the Flowbar. This will export the required simulation netlist and a SDF back annotation file into a directory called figba in the current design directory. A pop up dialog box will appear as shown below.



Figaro has generated the following simulation and back annotation files.
Simulation Files:
/sjo_design/fpga1/people/ewan/atuser/averager/figba/averager.vhd

SDF Files:
/sjo_design/fpga1/people/ewan/atuser/averager/figba/averager.sdf

Please compile the design files and invoke the simulator from a shell window
with the appropriate SDF annotation option

Exported Netlists Dialog Box

The box will give the name of the exported netlists. These netlists have to be compiled and the simulator invoked on them from a command *Shell* outside Figaro.

NOTE For Verilog users, if *Post-layout Simulation* is to be performed after *Compile* is completed, specify the export format of *flat Verilog* and *flat SDF* using *File>Export*. Figaro will generate a *design*.v file and SDF file containing delay information.

DownLoad Bitstream

After *Post-layout Simulation* with the appropriate simulator, the design is ready for downloading. For more information on downloading and bitstream utilities, refer to the “Device Programming” chapter of the *User’s Guide*, and “Figaro” section of this *Tutorial*.

NOTE The *DownLoad* option is not available to Workstation users as the bitstream file can be downloaded from a PC only.

Synopsys

This *Tutorial* outlines the methodology for optimizing VHDL and Verilog designs using the Synopsys FPGA Express Synthesis System in the PC environment. A method of integrating components created by the *Macro Generators* is also outlined. The *Macro Generators* instantiated in the VHDL/Verilog design are identified by Figaro which then creates an area or speed efficient hard macro layout. The results are used to create more efficient placement and routing in Figaro.

The design flow suggested in this *Tutorial* spans across two tools: Figaro is responsible for the design management and back end layout, while Synopsys FPGA Express is used for synthesis and optimization of the design.

The following tools must be installed and running before starting this *Tutorial*.

1. Figaro
2. Synopsys FPGA Express

Design Flow

The design flow is briefly summarized below.

System Setup (Figaro)

- Sets up the design directory and libraries

Design Entry (Figaro)

- Enters the design in VHDL or Verilog

Design Synthesis (FPGA Express)

- Sets up the software environment for synthesis
- Reads in the design
- Optimizes the design
- Writes the post-synthesis EDIF netlist

Functional Simulation (Any available simulator)

- Verifies functionality of the design

Layout Generation (Figaro)

- Invokes automatic/manual placement and routing
- Writes delay values for back annotation

Post-layout Simulation (Any available simulator)

- Reads in delay file for simulation

Device Programming (Figaro)

- Outputs Bitstream in the PC
- Downloads onto Atmel part

Tutorial Example

To work through the *Tutorial*, a design describing a moving point "averager" circuit is used. Both Verilog and VHDL versions of the design are provided.

The "averager" has input and output ports which are 8 bits wide. The circuit smoothes out a noisy waveform arriving at an input port by averaging the adjacent 8 values of the signal amplitude, and generating the averaged waveform at the output port.

The "averager" example is partitioned into design components, each of which can be generated automatically using the *Macro Generators* module in Figaro. While synthesis will still produce gate level connectivity for the components, Figaro will select the hard layouts of these components from the layout library. In this way, a truly technology independent VHDL design can be written for simulation and retargeting. The optimized netlist can use the best layout implementations to produce superior results.

The VHDL files of the "averager" design can be copied from \SystemDesigner\atmel\examples\at6k\fpgaexp\averager\vhdl or \SystemDesigner\examples\at40k\fpgaexp\averager\vhdl, which come as part of the Figaro installation.

The Verilog files of the "averager" design can be found in \SystemDesigner\Examples\AT40K\FPGAEXP\AVERAGER\VDHL

Or \SystemDesigner\Examples\AT40K\FPGAEXP\AVERAGER\Verilog

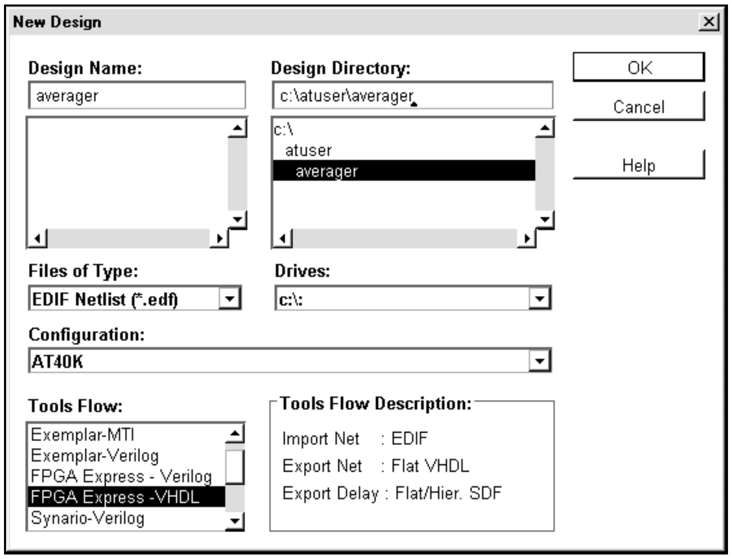
Creating and Setting up the Design Directory

In this first step, the user creates a design directory and assigns a CAE platform to the design so that tasks meaningful to the chosen platform can be applied automatically during processing.

To create a new design, select the icon on the Flowbar or use *File>Design Setup* from the menu.

Select the *New Design* button to open the *New Design Directory* dialog box.

Figaro will input the "averager" design in EDIF format for use. Use the steps outlined as follows to set up the environment. The dialog box will resemble that shown below.

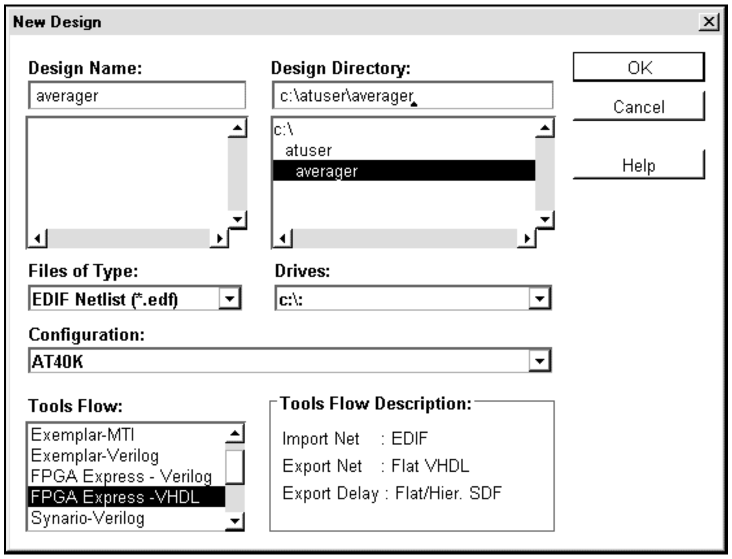


Design Name: averager
Design Directory: c:\atuser\averager
Files of Type: EDIF Netlist (*.edf)
Configuration: AT40K
Tools Flow: FPGA Express -VHDL
Tools Flow Description:
Import Net : EDIF
Export Net : Flat VHDL
Export Delay : Flat/Hier. SDF

New Design Directory Dialog Box

1. Under *Design Directory*, enter the path where the VHDL/Verilog files are to be located.
2. Enter the *Design Name* as "*averager*".
3. Set *Configuration* to *AT6K or AT40K*.
4. Set *Tools Flow* to *FPGA Express - VHDL* if the design source is VHDL, or to *FPGA Express - Verilog* if the design source is Verilog.

5. Under *Files of Type*, select *EDIF Netlist (*.edf)*.
6. Press OK to complete the *Design Setup*.



Design Name: averager

Design Directory: c:\atuser\averager

c:\
atuser
averager

Files of Type: EDIF Netlist (*.edf)

Drives: c:\:

Configuration: AT40K

Tools Flow:
Exemplar-MTI
Exemplar-Verilog
FPGA Express - Verilog
FPGA Express -VHDL
Synario-Verilog

Tools Flow Description:
Import Net : EDIF
Export Net : Flat VHDL
Export Delay : Flat/Hier. SDF

New Design Directory Dialog Box

Figaro supports application of a user library in which hard macro layouts can be stored for use in current or future designs. The *Macro Generators* create such hard layouts and require a library for storage of the information. Besides the hard layouts, simulation information and other important files are stored in this library directory. A library for the "averager" design is created as follows:

1. Select the *Library>Library Setup* module to activate the *Library Setup* dialog box.
2. Press the *Add Before* button in the *Library Search Path* pane to activate the *Add Library and Path* dialog box.
3. Enter the *Library Name* as "*averager*.lib" and press OK.
4. Press OK again to complete the *Library Setup*.

Component Generation

The "averager" design makes use of a number of components that can be automatically created using the *Macro Generators* in Figaro. This function produces a component or design that is more compact in size and efficient in speed.

An examination of the averager.vhd file will reveal a very simple method of incorporating the *Macro Generators* in VHDL. The process involves identifying the components that are going to be part of the design and inserting them one level deeper in the design hierarchy. Care must be taken to ensure correspondence between the port names in the VHDL designs and the macros created by the *Generators*. It is also imperative to maintain the component names in the VHDL design during macro generation.

The following components must be generated for the "averager" design via the *Macro Generators* button on the Flowbar. This will bring up the *Generator Notebook*. Choose the category of macros using the bottom tabs and the desired *Generator* using the right side tabs. The macro names and their appropriate parameters are mentioned below. All other parameters should be left to their default settings.

Generator Flip-Flop D-type, Width=8, Macro Name=reg8.

Generator Adder-Ripple Carry, CarryIn=Disabled, Register=Input, Width=9, Macro Name=add9reg.

Generator Adder-Ripple Carry, CarryIn=Disabled, Register=Input, Width=10, Macro Name=add10reg.

Generator Flip-Flop D-type, Width=8, Macro Name=reg8en, Register Bank Enable=Group Enable.

The *Macro Generators* are designed to exploit Atmel's register rich FPGA architecture, and produce a component that is more compact in size and efficient in speed. The hard layouts for the above-generated components are stored in the user defined library. They will be used by Figaro during *Compile*, when the EDIF netlist of the design is being processed.

Design Entry

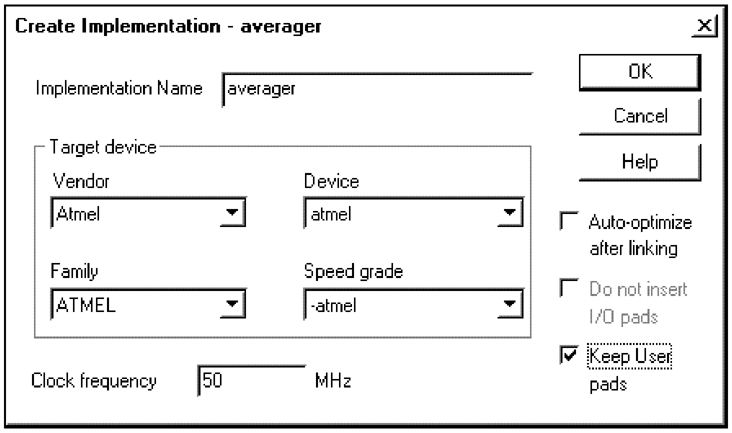
Once a design has been set up, the VHDL or Verilog files of the design description must be set up in the Design Directory for synthesis to proceed. For the "averager" design, these files are located the Design Directory.

Synopsys FPGA Express can now be brought up by clicking on the *Synthesis Tool Invocation* push button from the Flowbar on the IDS desktop.

Optimization and Netlist Generation

Once the design files are created, optimization of the completed design can proceed hierarchically in a top-down fashion. Follow the sequence of commands mentioned below to set the appropriate options during synthesis.

1. Create a new project named *tutorial* using *File>New* from the menu bar in FPGA Express.
2. Specify the source files (VHDL or Verilog) using the menu selection *Synthesis>Identify Sources*. This step will automatically analyze the VHDL or Verilog source files.
3. Select *averager* as the top-level from the *Implementation Name* drop down list and specify *Atmel* as the target device in the *Create Implementation* dialog box as shown below.



Create Implementation - averager

Implementation Name: averager

Target device
Vendor: Atmel
Device: atmel
Family: ATMEL
Speed grade: -atmel

Clock frequency: 50 MHz

☐ Auto-optimize after linking
☐ Do not insert I/O pads
☑ Keep User pads

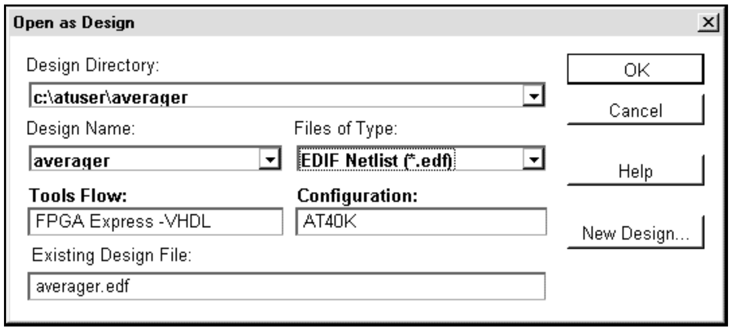
Create Implementation dialog box

4. By default FPGA Express eliminates hierarchy. In order to preserve the hierarchy, select the chip and right click the mouse button to bring up the pop-up menu to edit the constraints.

   In the *Modules* page of the *Constraints* window set *Hierarchy* to *Preserve* by default.
5. Close the *Constraints* window and select the chip. Right click the mouse button again and chose *Optimize chip*.
6. Select the optimized chip and chose *Export Netlist* to write out an EDIF netlist which will be read into Figaro for final implementation in the Atmel device.

Placement and Routing

Once the design is netlisted, it is ready for placement and routing. To read the averager.edf file into Figaro, click on the *Open* button in the tool bar and proceed with *Open as Design*. Specify *Files of Type* to be *EDIF Netlist (*.edf)* and verify that the *Existing Design File* is *averager.edf.*



Design Directory: c:\atuser\averager
Design Name: averager
Files of Type: EDIF Netlist (*.edf)
Tools Flow: FPGA Express -VHDL
Configuration: AT40K
Existing Design File: averager.edf

Open as Design Dialog Box for design "averager"

1. Click OK for Figaro to load the EDIF file and generate a database file for the design. While loading the EDIF file, Figaro identifies the *Macro Generators* that were generated in the previous steps and uses their hard layouts during placement and routing.

   In order to better optimize the synthesis results, it is recommended that mapping be performed. The netlist from the synthesis tools contain functional macros that can be packed into Look Up Tables by the technology mapper in Figaro. After opening the design, use *Options>Options>Mapping* and select the *Mapping Enabled* option. Then push the *Map* button on the Flowbar to map the design. Please refer to the "Figaro - Mapping" section of this *Tutorial* for more details about this process and the options available.

2. Specify an Atmel part by clicking on the *Parts* button. The Figaro interactive tool allows pin preplacement and pin lock assignments on the device chosen. Select the appropriate part to fit the design. The "averager" design should fit on an AT6005 or AT40K05 part but can be run on any device.

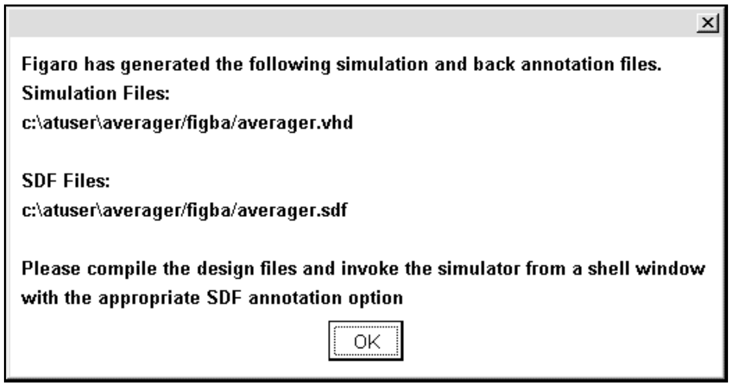
3. Initiate placement and routing by clicking on the *Compile* button. The four phases include *Initial Placement*, *Optimized Placement, Initial Route*, and *Optimized Route*. The last button can be either *Bitstream* (for designs) or *Check-in* (for macros). Depending on the initial design setup, Figaro will create a bitstream file for download to the device, or instantiate the component and store it in the user-specified library.

Details on the *Parts Assembler* and *Compile* modules are covered in the "Figaro" chapter in the *Tutorial*.

Post-layout Simulation

Post-layout Simulation can be performed by using the delay values extracted during *Compile*. The back annotation information is then read into the design in the specified export file format. Figaro determines the file types to be exported based on the selected tool flow and design implementation (whether the design is mapped or not, and if the implementation is for one or more chips). If the design is mapped or partitioned into multiple chips, a flat VHDL (or Verilog) netlist and the corresponding SDF netlist will be created. Otherwise a hierarchical SDF file will be generated and back annotated to the structural post-synthesis VHDL (or Verilog) netlist from the synthesis tool. However even if the design is implemented on a single chip and not mapped, the user can export a flat simulation netlist by using *File>Export*.

Click on the *Post-layout Simulation* button on the Flowbar. This will export the required simulation netlist and a SDF back annotation file into a directory called figba in the current design directory. A pop-up dialog box will appear as shown below.



Figaro has generated the following simulation and back annotation files.
Simulation Files:
c:\atuser\averager/figba/averager.vhd

SDF Files:
c:\atuser\averager/figba/averager.sdf

Please compile the design files and invoke the simulator from a shell window with the appropriate SDF annotation option

Exported Netlists Dialog Box

The dialog box will give the name of the exported netlists. These netlists have to be compiled and the simulator invoked on them from a command *Shell* outside Figaro.

NOTE For Verilog users, if *Post-layout Simulation* is to be performed after *Compile* is completed, specify the export format of *flat Verilog* and *flat SDF* using *File>Export*. Figaro will generate a *design*.v file and SDF file containing delay information.

DownLoad Bit Stream

For more information on downloading and bitstream utilities, refer to the “Device Programming” and “Figaro” chapters in the *User’s Guide* and *Tutorial* respectively.

NOTE The *DownLoad* option will only download the bitstream file from a PC. It is not available to Workstation users.

Model Technology

This tutorial demonstrates the various steps involved in simulating VHDL designs using Model Technology's VSIM V-System VHDL Simulator.

Design simulation is performed at various stages as the design is transformed from a behavioral description to the FPGA layout. This ensures that the functional and timing integrity of the design is within acceptable parameters of the original specifications.

Design Flow

A well-integrated design flow that supports easy verification is provided in the Figaro software. This flow is outlined below.

- Functional (Pre-synthesis) Simulation.
- Synthesis.
- Gate Level Netlist Creation.
- Placement and Routing.
- Post-layout Simulation.

The simulator supports numerous timing checks, recovery and error reporting mechanisms. For details on the default settings in the VITAL libraries, please refer to the "*Simulation - Exemplar*" chapter of the *CAE Interfaces* section in the *Technical Reference*.

This tutorial focuses more on the tools and methodology developed to perform simulation within the Figaro environment.

This tutorial assumes that the user is familiar with synthesis using Exemplar's Leonardo Spectrum. Those unfamiliar with the process are encouraged to go through the separate Exemplar synthesis tutorial.

Example Design

For this tutorial, the "averager" design will be used. This design implements a waveform smoothing function represented by:

$$y(n) = \frac{1}{M}\sum_{(i=0)}^{(M-1)} x(n-i)$$

The smoothing function is used in many DSP applications to filter out high frequency spikes. These spikes are the source of noise commonly found in communication channels. They can be eliminated by taking the moving average of sample values arriving at the input of the system.

The design in this example is an averager circuit with eight moving points. The hardware specific details of the design are described below.

Interface The "averager" design has 8-bit input and output data lines. The design also has a clock pin and a reset pin.

Assumptions and Other Details Only positive values can appear on the input data lines (the input is assumed to be level-shifted). During the implementation of the system, a valid output waveform can be expected on the 9th clock. However, since the outputs of the last and intermediate stages are registered, the actual output waveform starts occurring from the 11th clock cycle onwards.

Design Files

The “averager” design consists of many design files

- Test bench file
- Input stimulus file

The files are kept under /SystemDesigner/examples/at40k/exemplar/averager/vhdl the /SystemDesigner/examples/averager/vhdl directory.

Copy these files to the following directory: /SystemDesigner/examples/mti/averager/vhdl/* .

The averager design is kept in averager.vhd. This file has numerous entity-architecture pairs in them. The following is a brief description of each of them.

averager - Contains the top-level design. It has instantiations of the other modules of the design.

initadd - This module adds two delayed signals. It instantiates reg8 and add8 modules.

reg8 - This module contains an 8-bit register.

add8 - This module contains an 8-bit adder and produces a 9-bit sum.

add9reg - This module adds two 9-bit numbers and produces a 10-bit registered sum.

add10reg - This module adds two 10-bit numbers and produces an 11-bit registered sum.

sr9 - This module is a shift register. It is used to implement a divide by 8 operation.

A description of other vhdl files is given below.

aver_tb.vhd - This module contains the testbench file for the “averager” design. This test bench should be used for functional simulation.

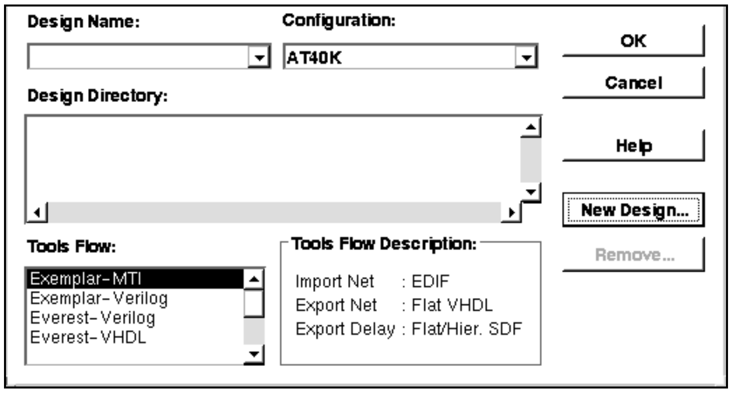
post_tb.vhd - This module contains the testbench file for the “averager” design. This test bench should be used for post-layout gate level simulation.

Invoking Figaro

Figaro can be started in the averager directory by selecting *START > ATMEL > IDS*

Figaro Design Setup

In the Figaro window, use the icon or *File>Design Setup* menu to set up the design and select its associated *Tools Flow*. The *Design Directory Setup* dialog box will be brought up.



Design Name:

Configuration: AT40K

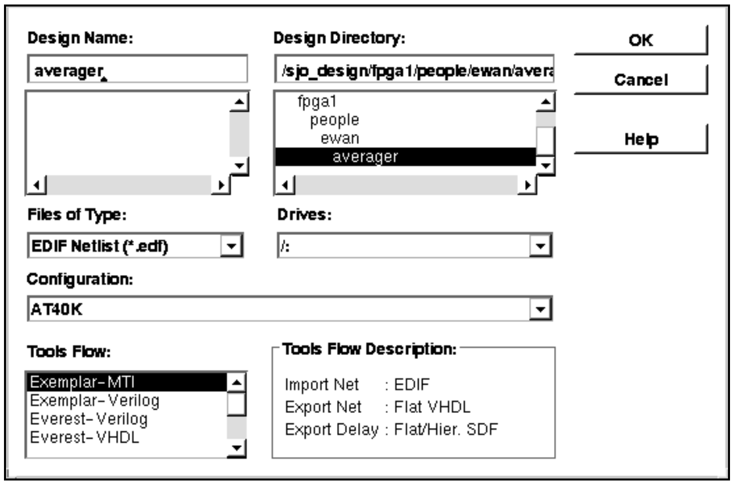
Design Directory:

Tools Flow: Exemplar-MTI, Exemplar-Verilog, Everest-Verilog, Everest-VHDL

Tools Flow Description:
Import Net : EDIF
Export Net : Flat VHDL
Export Delay : Flat/Hier. SDF

Design Setup Dialog Box

1. To create a new design click on the *New Design* button. The *New Design Directory* dialog box is shown below.



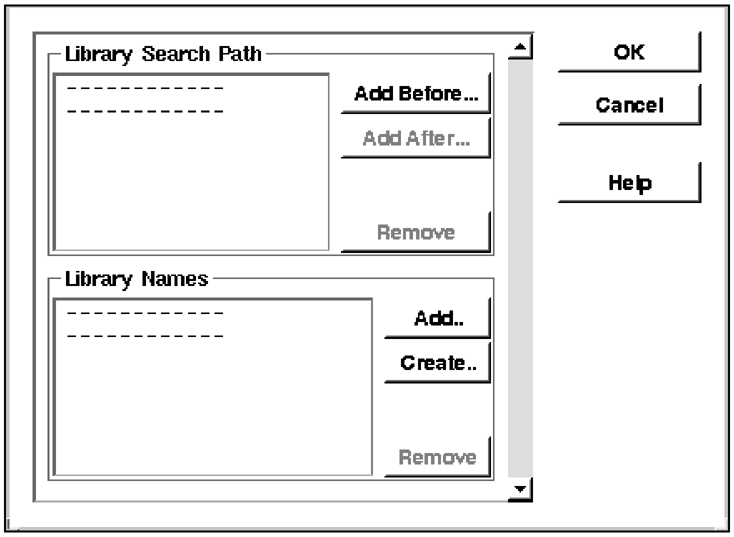
New Design Directory Dialog Box

2. In the *New Design* dialog box, type the name of the design as "*averager*". Set Configuration to *AT6K*/*AT40K*. Select *Tools Flow* as *Exemplar-MTI.*
3. Dismiss the *New Design* and *Design Directory* dialog boxes by pressing the respective OK buttons.

Library Setup

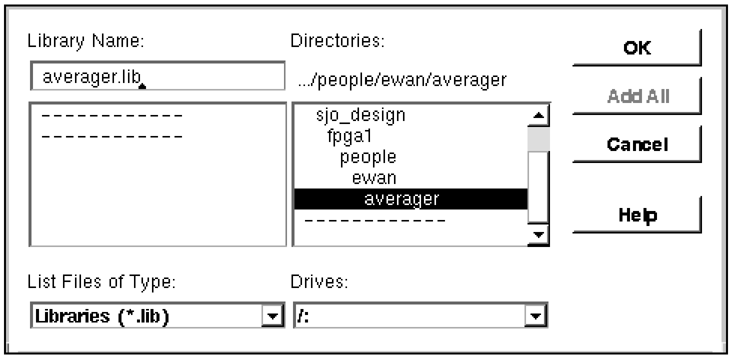
The "averager" design, once synthesized, will contain the macros inferred automatically by Exemplar's ModGen module generation system. These components are created automatically with the *Macro Generators.* The associated FPGA layout and simulation models will also be produced. Figaro treats these components as library modules and inserts them in the library. Consequently, a user library must be set up.

1. A design library can be established by executing the *Library>Library Setup* menu.



Library Setup Dialog Box

2. In *Library Setup*, press the *Add Before* button to display the *Add Library and Path* dialog box.



Library Name: averager.lib

Directories: .../people/ewan/averager (selected: averager)

List Files of Type: Libraries (*.lib)

Drives: /:

Add Library and Path Dialog Box

3. In the *Add Library and Path* dialog box, enter *Library Name* as "*averager*.lib" (or any name of choice). Dismiss the box by pressing OK.
4. Dismiss the *Library Setup* dialog box by pressing OK.

The design name has been set as "averager" and the library name as "*averager*.lib". The next step is to create the environment in the simulation setup files and carry out the simulation at the functional level.

Functional (Pre-synthesis) Simulation

The command vsim is used for simulating the design. Normally, this command will bring up the user interface. However, if this command is invoked with the -c switch, a *Shell* interface will be invoked. For the purpose of this tutorial, the *Shell* interface will be used.

Before starting simulation, a brief discussion of the stimulus and response files used and created by the simulator is provided below.

The test scaffold, specified in the aver_tb.vhd file, reads the file averager.sen that contains the waveform to be smoothed out. Any waveform can be specified in that file. Upon completion of the simulation, a file averager.out is created. This file contains the output values of the waveform.

The simulation script, "run -all; quit", passed via the -do switch, executes the simulation and quits the simulator at its conclusion.

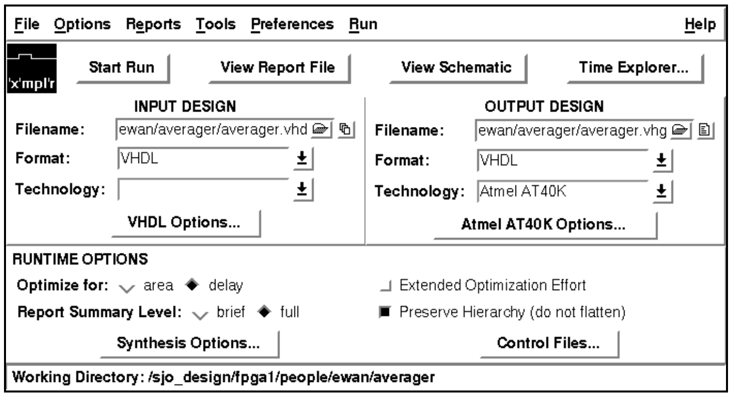
CFG_AVER_TB is the name of the top-level configuration specification, which points to the simulation model that is built into the work directory.

At the end of the simulation, edit the averager.out file and examine the output waveform.

Synthesis

After the design has been validated, it is ready for synthesis. At the end of synthesis, two files will be created in the design directory: the averager.edf and averager.vhg. The former is the EDIF netlist for the synthesized design, and will be imported into Figaro for placement and routing followed by the creation of the needed files for *Post-layout Simulation*. The file averager.vhg will contain the gate level VHDL netlist. This netlist will be used for post-synthesis gate level simulation.

1. Invoke Leonardo from the *Invoke Synthesis Tool* button on the Figaro Flowbar. Refer to the *CAE Interfaces* section in the *IDS Tutorial* on Exemplar Synthesis for more details.
2. In Leonardo , do the following:
   - Set the *Input Design, File Name* and *Format* to "averager.vhd" and "VHDL" respectively.
   - Set the *Output Design*, *File Name* and *Format* to "averager.vhg" and "VHDL" respectively.
   - Set the *Output Design*, *Technology* to "Atmel AT6K02/AT40K".



INPUT DESIGN
Filename: ewan/averager/averager.vhd
Format: VHDL
Technology:

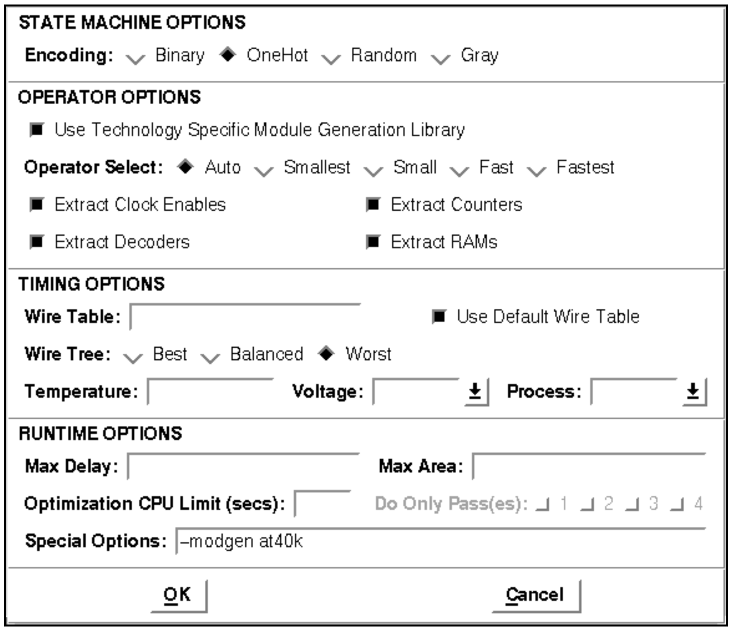
OUTPUT DESIGN
Filename: ewan/averager/averager.vhg
Format: VHDL
Technology: Atmel AT40K

RUNTIME OPTIONS
Optimize for: area [✓ delay]
Report Summary Level: brief [✓ full]
Extended Optimization Effort
[✓ Preserve Hierarchy (do not flatten)]

Working Directory: /sjo_design/fpga1/people/ewan/averager

Leonardo Dialog Box

3. Under *RUNTIME OPTIONS*, check *Preserve Hierarchy (do not flatten)*.
4. Click on the *Synthesis Options...* button of the Leonardo interface.



STATE MACHINE OPTIONS
Encoding: Binary [✓ OneHot] Random Gray

OPERATOR OPTIONS
[✓ Use Technology Specific Module Generation Library]
Operator Select: [✓ Auto] Smallest Small Fast Fastest
[✓ Extract Clock Enables] [✓ Extract Counters]
[✓ Extract Decoders] [✓ Extract RAMs]

TIMING OPTIONS
Wire Table: [✓ Use Default Wire Table]
Wire Tree: Best Balanced [✓ Worst]
Temperature: Voltage: Process:

RUNTIME OPTIONS
Max Delay: Max Area:
Optimization CPU Limit (secs): Do Only Pass(es): 1 2 3 4
Special Options: –modgen at40k

Synthesis Options Dialog Box

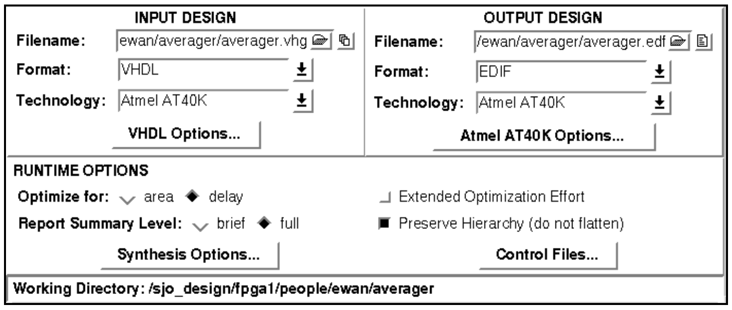
5. In the *Special Options* field, set the switch *-modgen at40k* or *-modgen at6k.*

6. Complete the synthesis session by executing the *Start Run* button of the Leonardo user interface.

Translating Gate level VHDL file to EDIF netlist

Leonardo cannot produce the EDIF and vhdl files simultaneously. To translate from averager.vhg to averager.edf file format, do the following in the main window:

1. Set *Input Design, Technology* and *Output Design, Technology* to "Atmel AT6K02/AT40K".
2. Set *Input Design, File Name* and *Output Design, File Name* to "averager.vhg" and "averager.edf" respectively.
3. Set the *Input Design, Format* and *Output Design, Format* to "VHDL" and "EDIF" respectively.
4. Check *RUNTIME OPTIONS, Preserve Hierarchy (do not flatten).*
5. Complete the file translation by initiating *Start Run* from the Leonardo user interface.


INPUT DESIGN
Filename: ewan/averager/averager.vhg
Format: VHDL
Technology: Atmel AT40K
VHDL Options...

OUTPUT DESIGN
Filename: /ewan/averager/averager.edf
Format: EDIF
Technology: Atmel AT40K
Atmel AT40K Options...

RUNTIME OPTIONS
Optimize for: area [✓ delay]
Report Summary Level: brief [✓ full]
Extended Optimization Effort
[✓ Preserve Hierarchy (do not flatten)]
Synthesis Options...
Control Files...

Working Directory: /sjo_design/fpga1/people/ewan/averager

Translation Setup Dialog Box

6. Alternatively, use the *Create Netlist for Open* button on the Figaro Flowbar. Enter the gate level VHDL netlist file name in the *Select VHDL File* dialog box and press OK. This will convert the VHDL netlist into an EDIF file by invoking the Exemplar tool in the command mode.

Exemplar tools can be dismissed after synthesis and translation is complete.

Gate Level Netlist Creation

The gate level VHDL and EDIF netlists created in the last step cannot be used directly for post-synthesis simulation as they make use of macros which must be created by IDS. In order to complete these netlists certain macros must be created interactively and others automatically from the EDIF netlist via the *Macro Generators Interface* (MGI).

Leonardo must be specified in Figaro as the input source. Access *Options>Options* from the menu and select *Synthesis Tool Invocation*. Under the settings for *Exemplar*, set the *Tool* to *Leonardo*.

Interactive Macro Generation

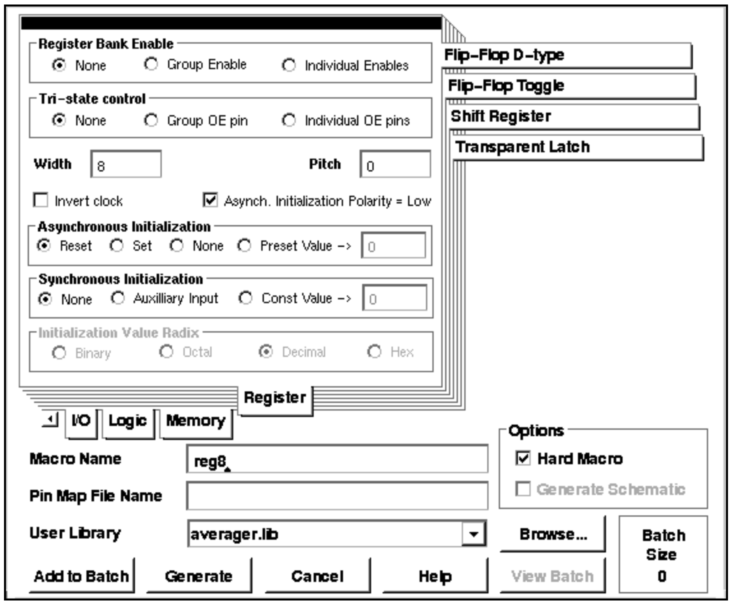
The "averager" design gate level netlist contains components that are treated by the Leonardo as hard macros. These macros are declared as NOMAP components in the averager.vhd design.

reg8:	8 bit register bank
reg8en:	8 bit register bank with enable
sr9:	9 bit shift register

These macros should be created using the *Macro Generators*. Click on the *Macro Generators* button in the user interface.

For the reg8 macro:

7. Select the *Register* tab along the bottom of the *Macro Generators* notebook.
8. Select the *Flip-Flop D-type* tab along the side of the *Macro Generators* notebook.
9. Set *Width* of the macro to “8”.
10. Set *Macro Name* to “reg8”.

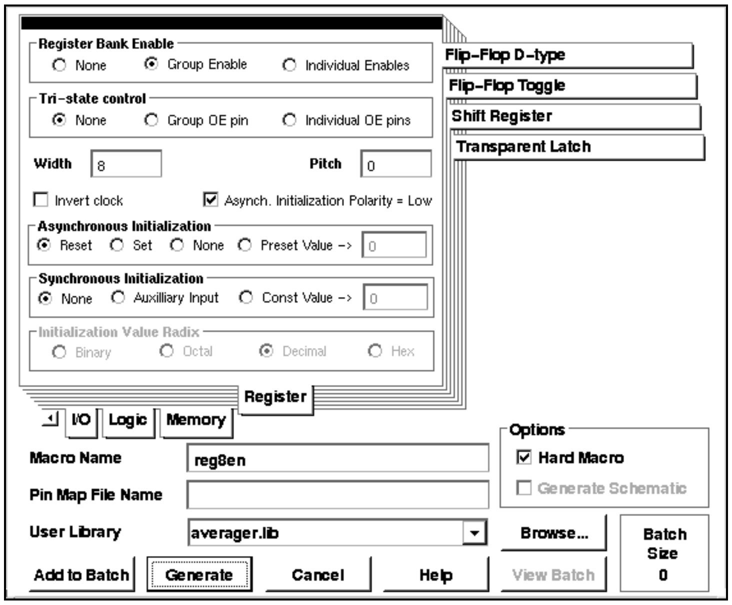


Register Bank Enable: [✓ None] Group Enable Individual Enables
Tri-state control: [✓ None] Group OE pin Individual OE pins
Width 8 Pitch 0
Invert clock [✓ Asynch. Initialization Polarity = Low]
Asynchronous Initialization: [✓ Reset] Set None Preset Value -> 0
Synchronous Initialization: [✓ None] Auxilliary Input Const Value -> 0
Initialization Value Radix: Binary Octal [✓ Decimal] Hex
Macro Name reg8
Pin Map File Name
User Library averager.lib
Options: [✓ Hard Macro] Generate Schematic
Batch Size 0

Generating the reg8 Macro

For the reg8en macro:

1. Use the same notebook page as above.
2. Set *Width* of the macro to “8”.
3. Set the *Register Bank Enable* to *Group Enable*.
4. Set *Macro Name* to “reg8en”.

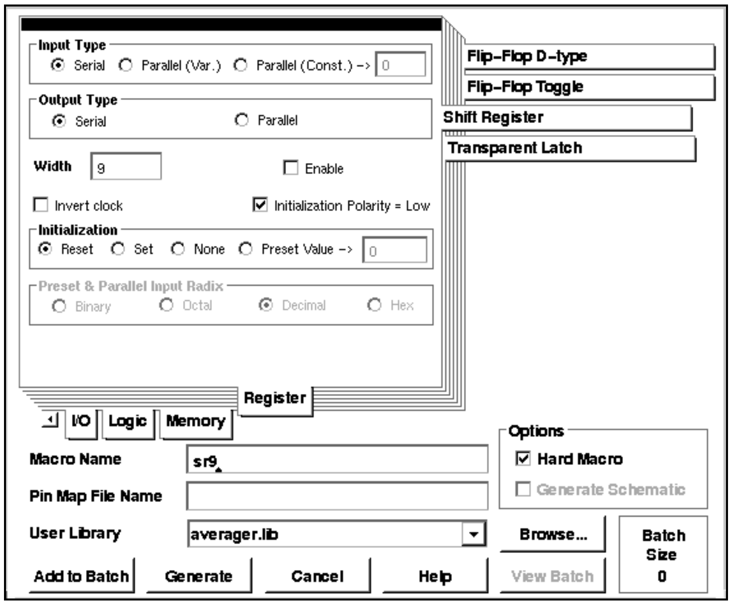


Register Bank Enable: None [✓ Group Enable] Individual Enables
Tri-state control: [✓ None] Group OE pin Individual OE pins
Width 8 Pitch 0
Invert clock [✓ Asynch. Initialization Polarity = Low]
Asynchronous Initialization: [✓ Reset] Set None Preset Value -> 0
Synchronous Initialization: [✓ None] Auxilliary Input Const Value -> 0
Initialization Value Radix: Binary Octal [✓ Decimal] Hex
Macro Name reg8en
Pin Map File Name
User Library averager.lib
Options: [✓ Hard Macro] Generate Schematic
Batch Size 0

Generating the reg8en Macro

For the sr9 macro:

1. Select the “Shift Register” tab in the *Macro Generators* notebook.
2. Set *Width* of the macro to “9”.
3. Set *Macro Name* to “sr9”.



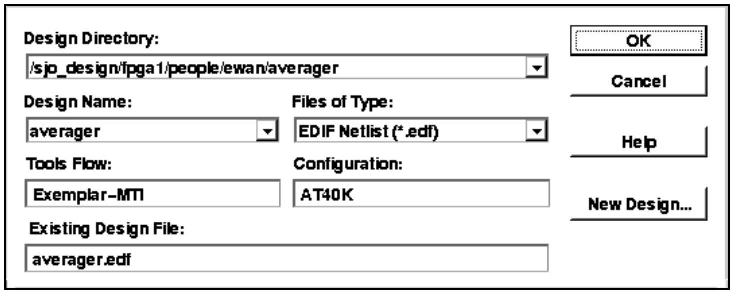
Input Type: [✓ Serial] Parallel (Var.) Parallel (Const.) -> 0
Output Type: [✓ Serial] Parallel
Width: 9
☐ Enable
☐ Invert clock
☑ Initialization Polarity = Low
Initialization: [✓ Reset] Set None Preset Value -> 0
Preset & Parallel Input Radix: Binary Octal [✓ Decimal] Hex
Macro Name: sr9
Pin Map File Name:
User Library: averager.lib
Options: ☑ Hard Macro ☐ Generate Schematic
Batch Size 0

Generating the sr9 macro

The *Macro Generators* user interface can be dismissed now.

MGI Macro Generation

In addition to these components, the synthesized gate level netlist contains components that are inferred automatically by the ModGen module generation system. ModGen does not create a VHDL description of these automatically generated components. Consequently, the gate level models for these components must be created. Figaro automatically creates the gate level models at the time of netlist import, and is described below.

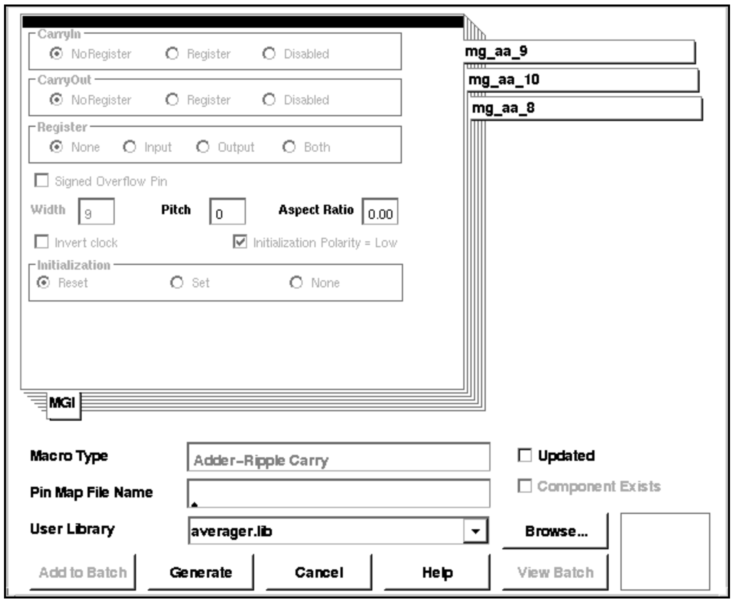


Design Directory: /sjo_design/fpga1/people/ewan/averager
Design Name: averager
Files of Type: EDIF Netlist (*.edf)
Tools Flow: Exemplar-MTI
Configuration: AT40K
Existing Design File: averager.edf

Open as Design Dialog Box

1. On the Flowbar, press the *Open* button. This will bring up the *Open as Design* dialog box.
2. Set the *Files of Type* field, to *EDIF Netlist (*.edf)*. This will display the string “*averager.edf*” in the *Existing Design File* field. Start the import process by pressing the OK button.
3. The automatic generation of inferred macros is undertaken by the *Macro Generators Interface*, and will be discussed below.

Macro Generators Interface (MGI)



CarryIn: [✓ NoRegister] Register Disabled

CarryOut: [✓ NoRegister] Register Disabled

Register: [✓ None] Input Output Both

☐ Signed Overflow Pin

Width 9 Pitch 0 Aspect Ratio 0.00

☐ Invert clock ☑ Initialization Polarity = Low

Initialization: [✓ Reset] Set None

mg_aa_9
mg_aa_10
mg_aa_8
MGI

Macro Type: Adder-Ripple Carry
Pin Map File Name:
User Library: averager.lib

☐ Updated
☐ Component Exists

Macro Generators Interface Dialog Box

During netlist import, Figaro recognizes the functions inferred automatically by the ModGen interface and displays them in the user interface as shown above. For the “averager” design, Figaro has identified the following macros:

mg_aa_9
mg_aa_10
mg_aa_8

These macros can be generated via user specification by supplying the appropriate options, such as *Layout Optimization* used for optimizing the layout. Note that options that determine the component’s functionality cannot be changed.

Press the GENERATE button and proceed with the macro generation.

The created components will be placed in the library displayed in the *User Library* field. In this case, they will be placed in averager.lib.

NOTE To automatically generate hard macros for the identified *Macro Generators* components without bringing up the graphical user interface, use the *Options>Options* menu. In the *Global Options* dialog box, check *Automatically Generate Components* under the topic *MGI Support*.

Placement and Routing

Once the design is verified with *Gate Level Simulation*, and the automatically inferred components are generated, it is ready for placement and routing. Press the *Compile* button to implement the design.

Specify the Atmel part by clicking on the *Parts* button. The Figaro interactive tool allows pin preplacement and pin lock assignments on the device chosen. For this exercise an AT6003/AT40K10 or larger device should be used.

Initiate placement and routing by clicking on the *Compile* button. The four phases are: *Initial Placement*, *Optimize Placement*, *Initial Route*, and *Optimize Route*. For more detailed instructions on how to select the Atmel part and implement a design in Figaro, please refer to the “Figaro” section of the *IDS Tutorial*.

Post-layout Simulation

After placement and routing is complete, a post-layout gate level vhdl netlist and SDF file containing delay information should be generated.

To create these files, press the *Post-Layout Simulation* button on the Flowbar.

Post-layout Simulation can be done along the same lines as *Functional Simulation*. The various steps involved are described below.

Post-layout Simulation Files

To eliminate the possibility of gate level files from being overwritten, Figaro creates a separate directory, called figba, in the design directory and places the post-layout simulation files in it.

In the *Shell Window*, change the directory to figba.

There will be a number of vhdl files in this directory. For the "averager" design, the list of files and a description of them are given below.

averager.vhd

This is the top-level design for the entire circuit. If a design has multiple partitions, this file contains instantiations of the individual designs that make up the circuit. Since the "averager" design has only one partition, only one design will be instantiated in averager.vhd.

NOTE The library statement "library averager" should be deleted from this file before continuing with simulation.

*mg_*.vhd

These are the design files that correspond to the macros identified automatically by the MGI. These components are instantiated in the top-level vhdl file(s). In the case of the "averager", these macros are instantiated in averager.vhd.

reg8.vhd, reg8en.vhd, sr9.vhd

These are the files corresponding to the reg8, reg8en and sr9 components generated using the *Macro Generators*.

In addition to the vhdl files, Figaro also outputs the SDF delay files corresponding to each partition. Since there is only one partition in the "averager", only averager.sdf is created.

Creating the modelsim.ini File

While in the figba directory, follow the procedure for creating the modelsim.ini file as outlined in the section "Simulation Setup" above.

NOTE Please ensure that the modelsim.ini file is deleted from, or renamed in, the project directory. The program vsim searches the modelsim.ini file in all directories in the search path and may pick up the settings from the modelsim.ini file in the project directory.

Copying Files to the figba Directory

Copy the testbench files post_tb.vhd and averager.sen from the design directory to the figba directory.

Analyzing the Design Files

Analyze the gate level design file averager.vhd and the testbench file by typing the following command at the *Shell* prompt.

```
$ vcom mg_* reg8.vhd reg8en.vhd sr9.vhd averager.vhd post_tb.vhd
```

The regular expression mg* will evaluate the vhdl file names of all components generated automatically by the *Macro Generators*. At the end of this command, a simulation model for the "averager" will be built into the work directory.

Simulating the Design

The *Post-layout* netlist is annotated using delays reported in the averager.sdf file. The design can be simulated by executing the following command.

```
$ vsim -c -do “run -all; quit” \
-sdftyp aver_test=averager.sdf CFG_AVER_TB
```

At the end of the simulation run, the output file averager.out will be created.

PLA Optinization

The *PLA Optimization* module translates and optimizes PLD designs described in the PLA format. A design can be described in equations, truth tables, state diagrams or any combination of all three with the equation entry method. DATA I/O's ABEL supports this design methodology, providing a very convenient and fast way of targeting PLA files for the Atmel FPGA. The Integrated Development System offers a well-defined design flow for easy, efficient synthesis and optimization of ABEL designs. This tutorial contains an example of how to create a design using both an ABEL translation and a schematic netlist. Optimization recommendations for ABEL designs using the PLA2Cdb software are included below. Files compiled in CUPL also observe similar processes.

Meeting performance specifications in an optimal way is determined in most cases by the ability to partition a system into its logical sub-components. The system partitioned in this way can then be represented as an interconnection of blocks in which the blocks themselves are specified at the behavioral level.

The design can be synthesized into a technology specific database and provided with multiple options to maximize a design centered on such user criteria as area or speed. *PLA Optimization* generates components with better timing characteristics, superior utilization, and supports a partitioning model that uses a schematic to tie the design together.

Using *PLA Optimization*, the PLA design can be translated into either a soft macro or a complete design. A complete design has its pins connected to I/O buffers. A soft macro can be instantiated in a top-level schematic or optionally translated into a hard macro using the *Macro Check-in* feature of IDS. Once a PLA file is turned into a design, it is ready for placement and routing.

System Setup

It is important that the Design and Library files, with their respective root paths, be in place prior to opening a design. The correct setup of the environmental variables is also needed for proper execution of the programs.

Setup Files

Figaro can optimize the PLA-formatted files generated by ABEL or CUPL. The resulting component can be turned into a soft macro for use in a larger circuit. The user can also translate the PLA file into a complete design, and generate a bitstream within Figaro.

After entering the design with a text editor, the ABEL or CUPL compiler is invoked. Depending on whether the design was optimized or not, the user can have either a *.tt1, *.tt2, or *.pla file as input to IDS. A brief explanation of the extensions is shown in the Table below.

PLA Files for Figaro Interface

PLA File Extension	Description
*.tt1	ABEL unoptimized *.pla file
*.tt2	ABEL optimized *.pla file
*.ttx	PLA file optimized by REEDMUL
*.pla	CUPL *.pla file

Atmel Libraries

Please refer to the Figaro Tutorial for step by step information on setting up a library, creating, using, and managing macros.

Copying the Example

The example files for this tutorial can be found in the directory \SystemDesigner\examples\at40k\abel or \SystemDesigner\examples\at6k\abel for AT40k and AT6k users respectively. There are 3 files that must be copied over once the design directory has been created.

Design Entry

In this tutorial, the design flow is illustrated with the synthesis and optimization of an example digital system. The example is chosen to provide insight into creating a design via Equation Entry using ABEL-HDL, and Schematic Entry with Viewlogic PROcapture. A similar flow can be supported on other platforms as well.

Creating a Design

The example digital system performs arithmetic operations by decoding a string of numeric characters received on its serial port. The numeric operands (undergoing computations) are read from the 2 *n* bit parallel ports of a system. The output is placed on a 1 *n* bit output port.

If string 110 is placed on the control line, the function will add the two 4-bit input values to produce the 4-bit output.

If string 111 is placed on the control line, the function will subtract the two 4-bit input values to produce the 4-bit output.

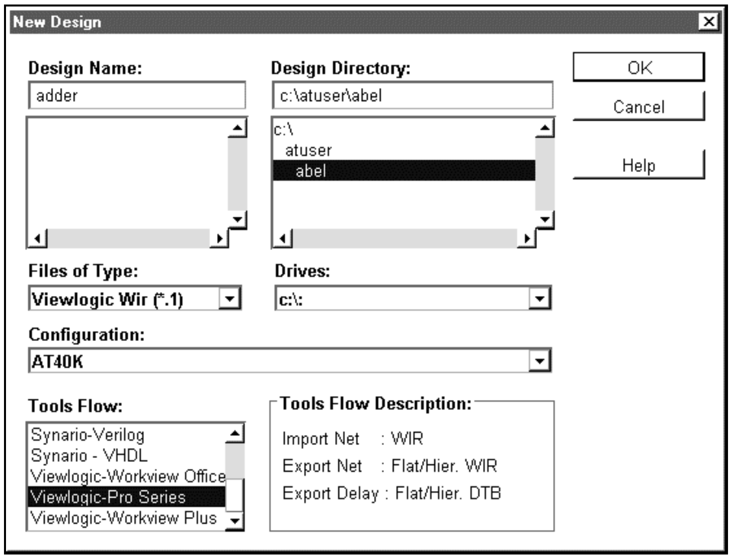
System Partitioning

The system can be divided into three sub-modules.

1. An adder
2. A subtractor
3. A sequence detector to identify 110 and 111 sequences.

The process of creating a design starts with partitioning it into sub-modules and writing an ABEL description for each of them.

1. Set up a Design Directory using *File>Design Setup* and select *New Design*. Fill in the dialog box to look like the following and select OK. Then select OK for the *Design Setup* dialog box as well.


New Design

Design Name: adder

Design Directory: c:\atuser\abel

c:\
atuser
abel

Files of Type: Viewlogic Wir (*.1)

Drives: c:\:

Configuration: AT40K

Tools Flow:
Synario-Verilog
Synario - VHDL
Viewlogic-Workview Office
Viewlogic-Pro Series
Viewlogic-Workview Plus

Tools Flow Description:
Import Net : WIR
Export Net : Flat/Hier. WIR
Export Delay : Flat/Hier. DTB

New Design Directory Dialog Box

NOTE

The user may specify AT40k or AT6k as the target FPGA.

2. To describe a design in ABEL:
   - Click on the *Open Shell Window* button on the vertical toolbar to access the DOS environment in the design directory.
   - Enter the ABEL description of the adder as specified below.
   - Invoke ABEL to translate the description. This is done by starting ABEL, opening up the adder.abl file, and then choosing the *Compile>Compile* menu item. A PLA file with the extension *.tt1 will be created in the design directory.
3. Do the design setup again for the "sub" module. Again, run ABEL to create the *.tt1 file for the "sub" macro.
4. Repeat the same process for the "control" module. Additionally, run the optimization step in ABEL This will produce a *.tt2 file.

The ABEL description of these three modules is given in the figures below.

```
module adder;
title '4 bit adder';

"inputs
a1,a2,a3,a4  pin 1,2,3,4;
b1,b2,b3,b4  pin 5,6,7,8;

"outputs
o1,o2,o3,o4  pin 9,10,11,12;

"sets
A = [a1,a2,a3,a4];
B = [b1,b2,b3,b4];
O = [o1,o2,o3,o4];

equations
        O = A + B;
end
```

ABEL Description of 4-bit Adder

```
module sub;
title '4 bit subtractor';

"inputs
a1,a2,a3,a4  pin 1,2,3,4;
b1,b2,b3,b4  pin 5,6,7,8;

"outputs
o1,o2,o3,o4  pin 9,10,11,12;

"sets
A = [a1,a2,a3,a4];
B = [b1,b2,b3,b4];
O = [o1,o2,o3,o4];

equations
        O = A - B;
end
```

ABEL Description of 4-bit Subtractor

```
module control
title 'Control state diagram for add/sub operation';

"inputs
clock, reset pin 1,2;
sin pin 3;

"outputs
sub, add pin 4,5;

"states
q1, q2 node istype 'reg_d,buffer';
add_sub_control = [q1,q2];

"state values
s1 = 0; s2 = 2; s3 = 3; s4 =4;

equations
   add_sub_control.clk = clock;
   add_sub_control.ar = reset;

state_diagram add_sub_control

State s1: sub = 0;
       add = 0;
       if (sin == 0) then s1 ;
       else if(sin) then s2;
       else s1;

State s2:
       sub = 0;
       add = 0;
       if(sin) then s3;
       else s1;

State s3:
       if(sin) then s1 with add = 0; sub = 1;
       else if(!sin) then s2 with add = 1; sub =0;
end
```

ABEL Description of Sequence Detector

Optimizing in the ABEL Environment

Some designs can be more efficiently optimized as AND-OR equations as opposed to AND-XOR equations. AND-XOR optimization consists of reading a PLA file and optimizing it using AND-XOR optimization techniques. AND-OR minimization is not done in PLA optimization. However, mapping of already optimized AND-OR equations is provided. If AND-OR optimization is chosen, a *.tt2 file must already exist in the design directory.

Notes to ABEL 5.X users

In version 5.1 ABEL, a *.tt2 file can be generated by setting the PLA file option in the *Xfer>Translate Options* dialog box and pressing the *Xfer>Translate* button.

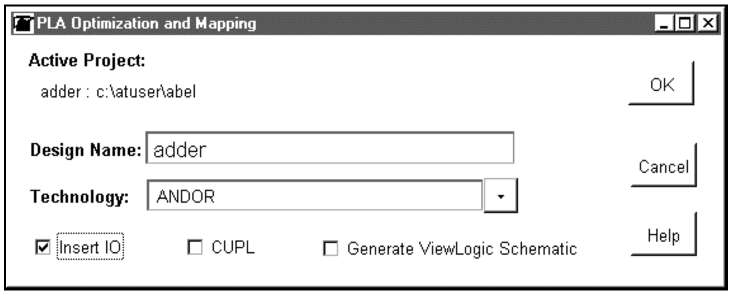
The ABEL compiler from version 5.1 onwards does not produce a *.tt1 file. If AND-XOR optimization is desired, a *.tt2 file can be renamed to *.tt1 before invoking the PLA optimization user interface.

Optimizing with Figaro

The 3 designs will now be brought back into Figaro to minimize logic and perform technology mapping. The first step to minimization consists of setting the optimization criteria. Knowledge of the design will help in this determination. Since adders and subtractors are arithmetic functions, AND-XOR technology may work most efficiently, while the control state machine can work better with AND-OR technology.

Once the optimization criteria is determined, the next step is to minimize and map an ABEL module to the FPGA layout. Steps involved in optimizing and translating the adder are listed below. Similar steps should be taken to optimize the “sub” and “control” modules.

1. Use *File>Design Setup* to select the adder again.
2. Click on the *PLA Optimization* button. The PLA2Cdb window will appear. Complete the screen settings as described below.



Active Project:
adder : c:\atuser\abel

Design Name: adder

Technology: ANDOR

[✓ Insert IO] CUPL Generate ViewLogic Schematic

PLA2Cdb Dialog Box

3. Specify the *Technology* as *AndOr*.
4. Click OK.

The program runs are displayed on screen that shows a summary of all warnings and errors. Details of the process can be found in the file pla2cdb.lst.

For a full explanation of IDS messages, refer to the *Technical Reference* manual.

Complete running pla2cdb on the other 2 modules, specifying the *Technology* to be *AndOr* for the “control” block.

Creating A Top-level Design

Once all sub-modules are created, they can be connected to each other and to I/O pads in the top-level schematic. Steps involved in creating a top-level design are listed below.

1. Use *File>Design Setup* to create a new design called “Toplevel”.
2. Press the *Schematic Entry* button to bring up the Viewlogic schematic editor and create the top-level schematic.
3. Interconnect the modules and I/Os.

Once the modules are connected, a design can be placed and routed using IDS programs.

Recommendations for describing a design in ABEL

1. Partition the system carefully to produce medium sized modules. Medium sized modules produce optimal logic minimization and efficient implementation in a layout.
2. Decide on the optimization criteria first.
3. Avoid using JK flip-flops.
4. Use of tri-state nodes is discouraged. Tri-stating of non-combinatorial nodes, with the exception of a D-type flip-flop, is not supported.
5. The use of multiple clocks and resets is discouraged.

It should be noted that some syntactic conventions related to ABEL extensions must be followed when designs are targeted to AT6000 FPGAs. Refer to the “CAE Interfaces” section of the *Technical Reference* for a list of supported extensions and their meanings.

QuickChange

The QuickChange software is used to access the Cache Logic capability of the Atmel FPGAs.

The design used in the tutorial contains a number of pulse generators. The QuickChange software is used to select the output of one of the pulse generators. This output is fed to a LED, which blinks at a rate equal to the frequency of that pulse generator.

The design created in this tutorial contains 4 pulse generators. The outputs of these pulse generators are multiplexed together using a 4 to 1 MUX macro. The output of this macro is assumed to be connected to an LED, which will blink at an interval equal to the selected frequency. The two select lines of the 4 to 1 MUX are connected to a two-bit array of constants. The constants are modified using QuickChange to make the appropriate selection.

The output of each individual pulse generator should also be connected to its own LEDs. This facilitates observation of the QuickChange LED versus the original setup.

Software Requirements

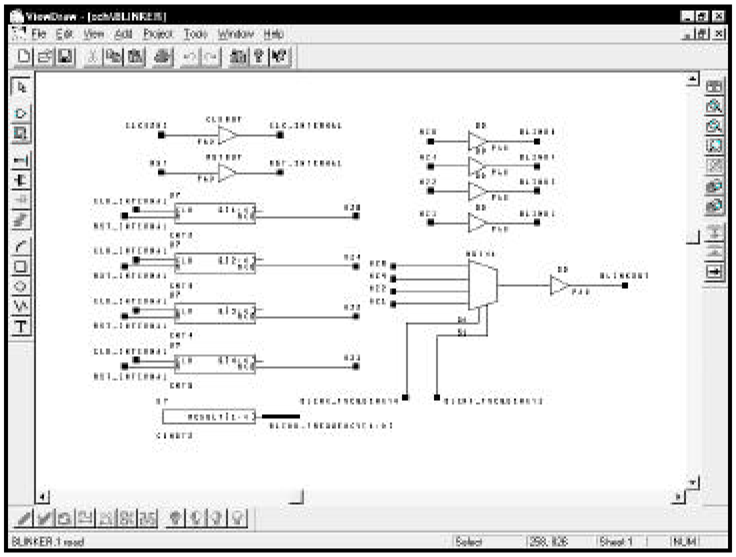
The following software setup is needed to create the demonstration design:

- WorkView Office or other Viewlogic schematic capture software to view the schematic.
- Atmel IDS

Hardware Requirements

In addition to the software setup, the following hardware components are needed:

- 32 HZ square wave frequency source
- 5 LEDs



The Blinker Schematic

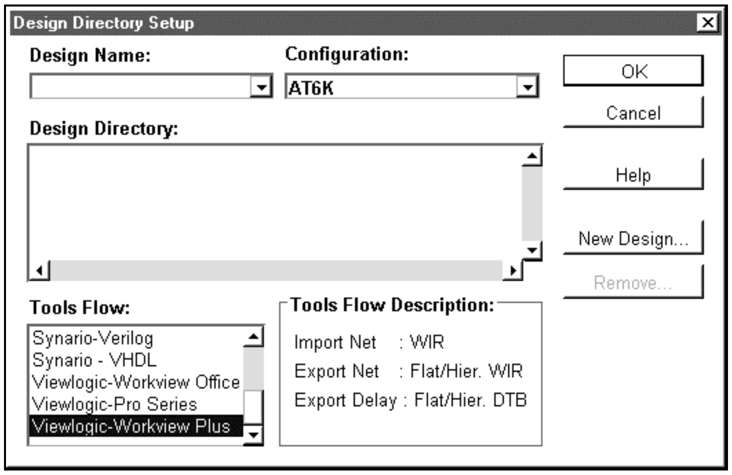
The circuit used in the tutorial is shown in the schematic diagram above. This example uses four counters to divide the input frequency of 32 Hz. cnt2 is a 2-bit counter and its RCO pin toggles every 8 Hz. cnt3 is a 3-bit counter and its RCO pin toggles every 4 Hz. cnt4 is a 4-bit counter and its RCO pin toggles every 2 Hz. Finally, cnt5 is a 5-bit counter and its RCO pin toggles every 1 Hz. The four outputs of these counters are multiplexed using a 4 to 1 MUX macro. The select lines of this macro are controlled by an array of constant cells with the output bus called BLINK_FREQUENCY[1:0].

NOTE The design files for Viewlogic can be copied from the *SystemDesigner*\atmel\examples\at6k\clogic directory or generated by following the steps in this tutorial.

Design Setup

To set up the environment for this exercise, follow the steps below:

1. Start IDS by clicking on the Figaro.
2. In Figaro, use the icon or *File>Design Setup* menu to set up the design and select the associated tool flow for it. The *Design Directory Setup* dialog box will be brought up.

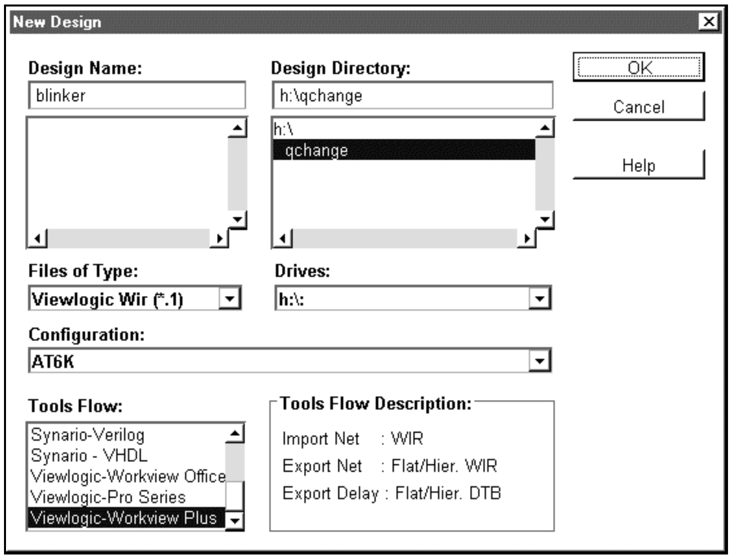


Import Net : WIR
Export Net : Flat/Hier. WIR
Export Delay : Flat/Hier. DTB

Design Setup Dialog Box

NOTE The QuickChange software is supported for AT6k FPGAs only.

3. Click on *New Design* to invoke the *New Design* dialog box. Set the *Design Directory* and *Design Name* fields to *\qchange* and “blinker” respectively. Set *Tools Flow* to *Viewlogic - WorkView Office* or the appropriate tool flow.



Design Name: blinker

Design Directory: h:\qchange

h:\
qchange

Files of Type: Viewlogic Wir (*.1)

Drives: h:\:

Configuration: AT6K

Tools Flow:
Synario-Verilog
Synario - VHDL
Viewlogic-Workview Office
Viewlogic-Pro Series
Viewlogic-Workview Plus

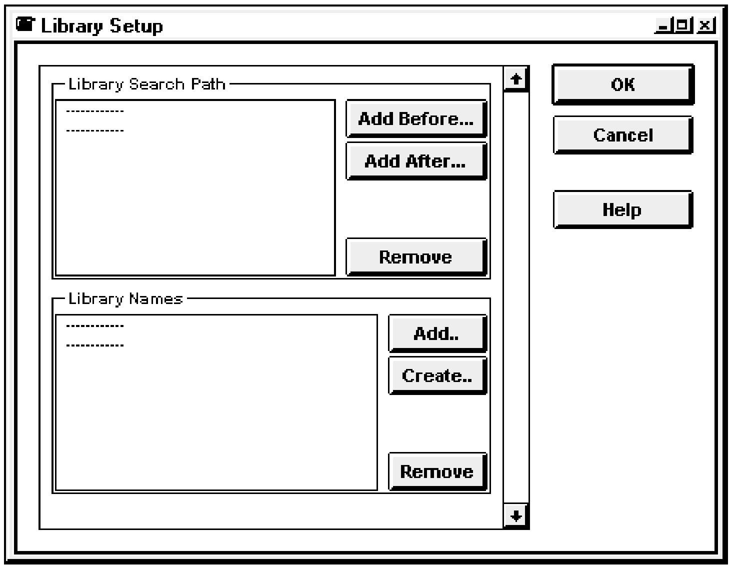
Tools Flow Description:
Import Net : WIR
Export Net : Flat/Hier. WIR
Export Delay : Flat/Hier. DTB

New Design Directory Dialog Box

4. Dismiss *New Design Directory* and *Design Setup* dialog boxes by clicking on OK.

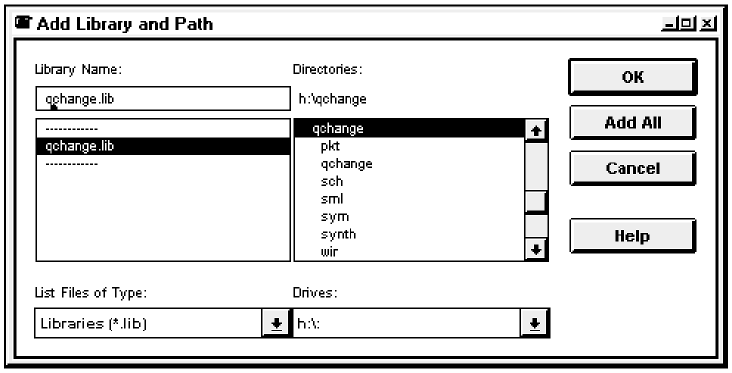
Library Setup

The design consists of components (counters and an array of constants) that are to be created using the *Macro Generators*. Figaro treats these macros as library modules and inserts them into the library. Consequently, a user library must be set up. This library can be established by selecting the *Library>Library Setup* menu.



Library Setup Dialog Box

1. Click on *Add Before* to invoke the *Add Library and Path* dialog box. Enter the *Library Name* as “qchange.lib”.

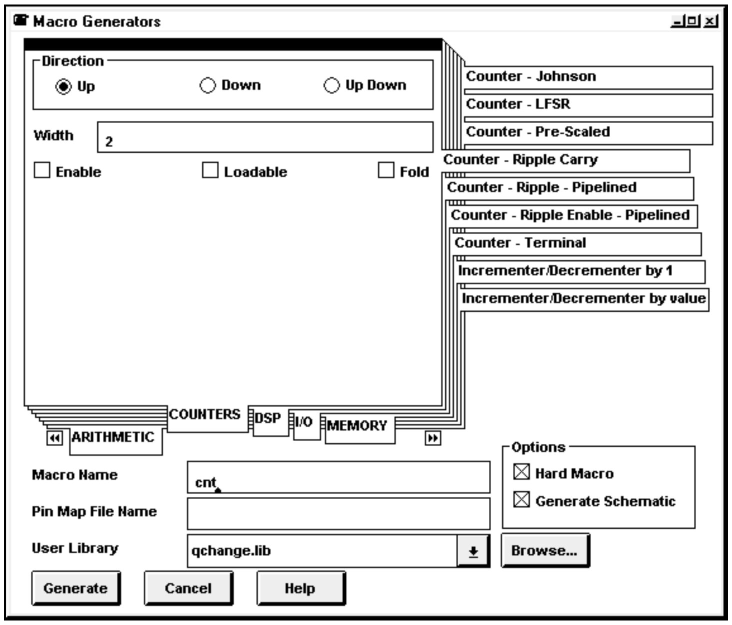


Add Library and Path Dialog Box

2. Dismiss the *Add Library and Path* and *Library Setup* dialog boxes by pressing OK.

Generating Components

Select the *Macro Generators* icon from the Figaro Flowbar to bring up the user interface.



Generating Counters

Generating Counters

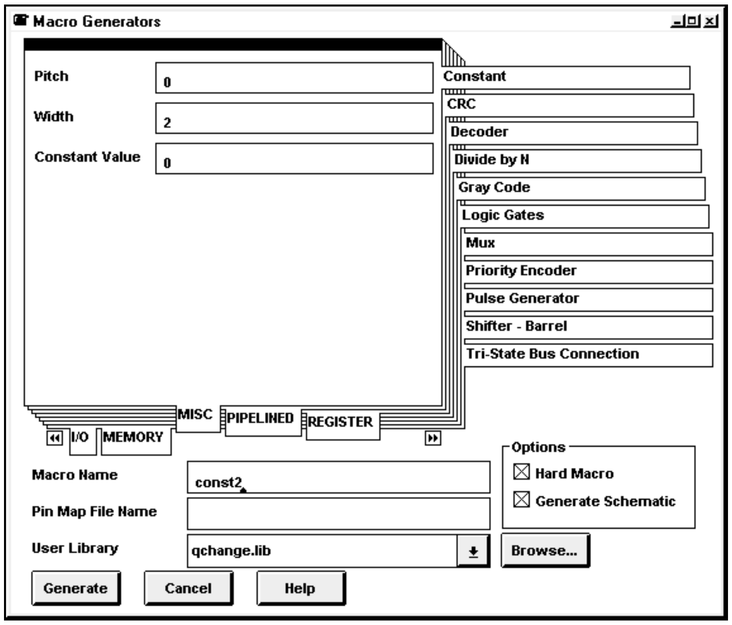
1. Select the *COUNTERS* tab in the *Macro Generators* user interface. Then select the *Counter-Ripple Carry* tab. To generate a 2-bit counter, set the various fields as shown in the figure above (*Direction UP, Width 2, Macro Name cnt2*.) Create the macro by pressing the GENERATE button.

2. Using the same procedure, generate the 3, 4, and 5 bit counters and name them cnt3, cnt4 and cnt5 respectively.

Generating Constants

1. Change the *Generators* page of the *Macro Generators Interface* to *MISC* and select the *Constant* generator.

2. Set the fields of the *Generators* as shown in the figure below (*Width 2* and *Macro Name const2*).



Pitch: 0
Width: 2
Constant Value: 0
Macro Name: const2
Pin Map File Name:
User Library: qchange.lib
Options
☒ Hard Macro
☒ Generate Schematic

Generating Constants

3. Create the macro by pressing the GENERATE button. Dismiss the *Macro Generators User Interface*. Note that the default value of *const2* is set to *00*.

Creating the Schematic

The “blinker” schematic has been provided in the \SystemDesigner\atmel\examples\clogic directory in Viewlogic format. If another CAE system is to be used, the schematic must be created.

Components cnt2, cnt3, cnt4, cnt5 and const2 are accessed from the “qchange” library. The remaining components, ITTL, OD, CLKBUF, RSTBUF and MUX41 are accessed from the “AT6K” library.

Label the output bus name of const2 to *BLINK_FREQUENCY[1:0]*. This bus name will be displayed in the QuickChange user interface.

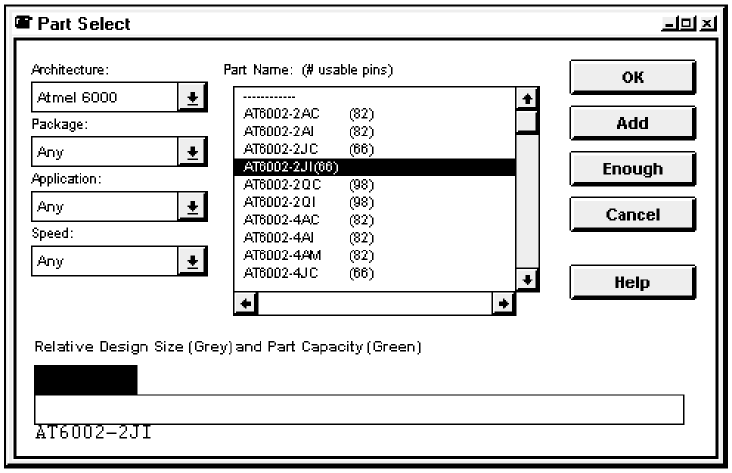
Placement and Routing

The netlist file can be read into Figaro by clicking on the *Open* button on the toolbar. Answer the *Macro or Design* dialog box with the *Design* option. Specify *Files of Type* as *Viewlogic Wir (*.1)* and verify that *Existing Design File* is set to *blinker.1*.

Selecting Parts

1. Press on the *Parts* button on the toolbar.

2. Select AT6002-2JC part and add it to the *Device Browser* by pressing the *Add* button. Dismiss the *Part Select* dialog box by pressing the OK button.

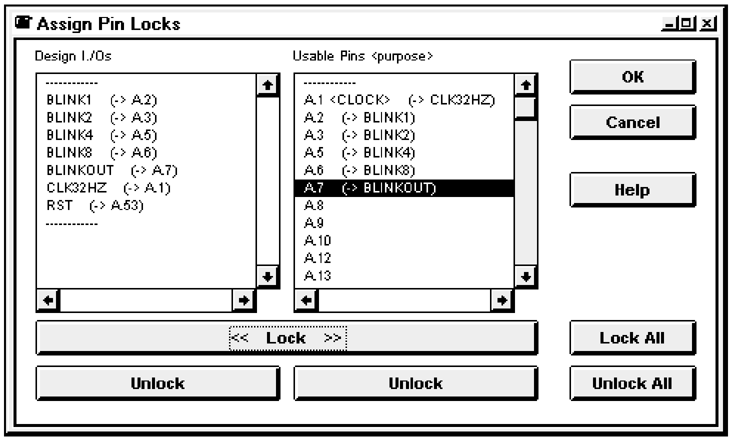


```
Architecture:   Atmel 6000
Package:        Any
Application:    Any
Speed:          Any

Part Name: (# usable pins)
-----------
AT6002-2AC    (82)
AT6002-2AI    (82)
AT6002-2JC    (66)
AT6002-2JI(66)
AT6002-2QC    (98)
AT6002-2QI    (98)
AT6002-4AC    (82)
AT6002-4AI    (82)
AT6002-4AM    (82)
AT6002-4JC    (66)

Relative Design Size (Grey) and Part Capacity (Green)
AT6002-2JI
```

The Part Select Dialog Box

Assigning Pin Locks

1. Invoke the Assign Pin Locks dialog box by selecting *Edit>Assign Pin Locks* from the menu bar.
3. Assign the pins to BLINK1, BLINK2, BLINK4, BLINK8 and BLINKOUT signals as shown below. These pins will be connected to the LED pins on an actual hardware set up.



```
Design I/Os                 Usable Pins <purpose>
-----------                 -----------
BLINK1   (-> A2)            A1 <CLOCK>   (-> CLK32HZ)
BLINK2   (-> A3)            A2   (-> BLINK1)
BLINK4   (-> A5)            A3   (-> BLINK2)
BLINK8   (-> A6)            A5   (-> BLINK4)
BLINKOUT   (-> A7)          A6   (-> BLINK8)
CLK32HZ   (-> A1)           A7   (-> BLINKOUT)
RST   (-> A53)              A8
-----------                 A9
                            A10
                            A12
                            A13
```

Assign Pin Locks Dialog Box

Placement and Routing

Press the *Compile* button on the Flowbar to complete the placement and routing of the design.

Setting Up the Hardware

The hardware should be set up as follows:

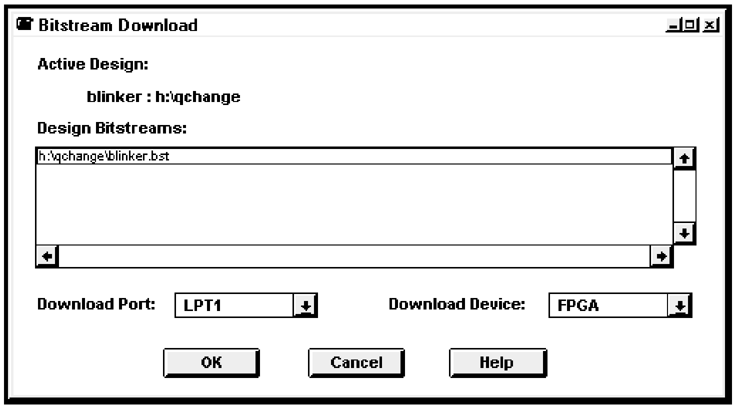
1. Install an AT6002-2JC device on the FPGA prototype board supplied by Atmel.
2. Install 5 LEDs on the board.
3. Connect a pin of each LED to system ground.
4. Connect the other LED pins to pins 1, 2, 3, 4 and 5 of the FPGA (or whichever pins were assigned during the layout process).
5. Connect a 32 Hz clock source to Pin 1 of the FPGA. The user may optionally connect the RST pin (pin 53) to VDD.

Reconfiguring with QuickChange

After the hardware is properly set up, the FPGA can be reconfigured to create the flickering of the LED connected to the net labeled “BLINKOUT”.

The first step in the download sequence is to configure a complete FPGA with the bitstream produced from the design.

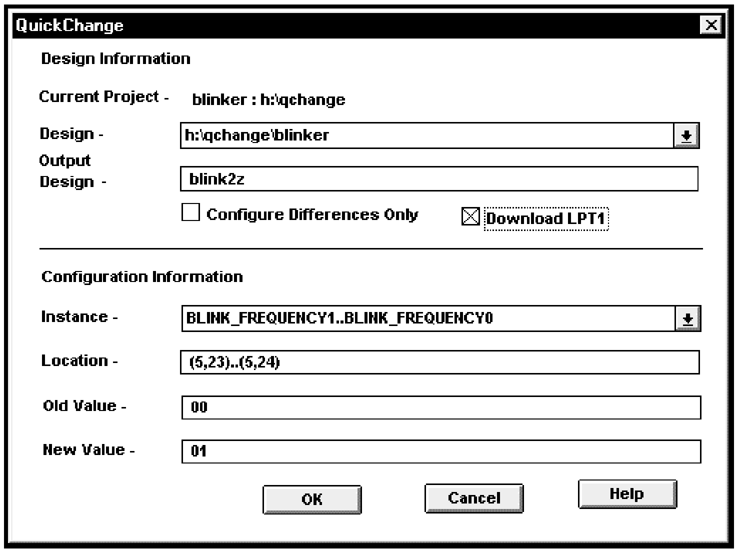
1. Power up the FPGA prototype board.
2. Press the *Download* button provided on the Figaro Flowbar, to display the *Bitstream Download* dialog box.



Active Design:
blinker : h:\qchange
Design Bitstreams:
h:\qchange\blinker.bst
Download Port: LPT1
Download Device: FPGA

Bitstream Download Dialog Box

3. Select the *blinker.bst* bitstream displayed in the *Design Bitstreams* list. Press the OK button to download the initial bitstream.
4. Since the value of the *BLINK_FREQUENCY* is *00*, the *BLINKOUT* LED should flicker with a 1 Hz frequency (tied to the 5 bit or divide by 32 counter).
5. Invoke the QuickChange software by pressing the *QuickChange* button provided on the Figaro Flowbar.



Current Project - blinker : h:\qchange
Design - h:\qchange\blinker
Output Design - blink2z
☐ Configure Differences Only
☒ Download LPT1
Instance - BLINK_FREQUENCY1..BLINK_FREQUENCY0
Location - (5,23)..(5,24)
Old Value - 00
New Value - 01

QuickChange Dialog Box

The settings on the user interface above will create a new bitstream named blink2z, which changes the blinking frequency of the LED from 1 Hz (original blinking frequency) to 2 Hz. This is done by changing the *BLINK_FREQUENCY* constant to select the output of the 4 bit counter. This will divide the 32 Hz clock by 16 to produce a 2 Hz signal. Make sure that the *Download LPT1* box is checked to automatically download the results to the prototype board. Execute the bitstream generation by pressing OK. The *BLINK* LED should blink twice as fast as before.

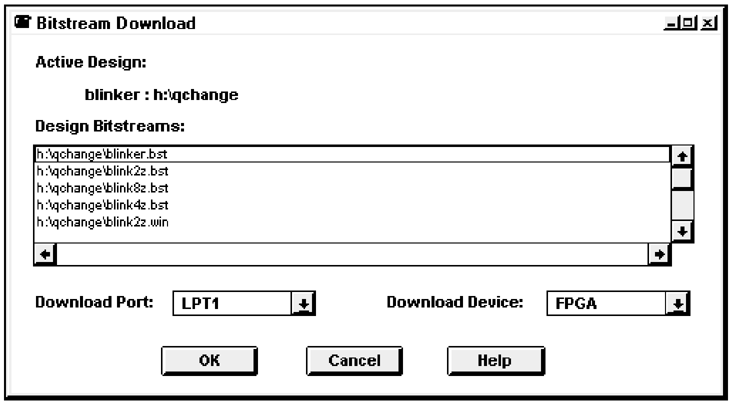
Using the same procedure, generate blink4z (new value 10) and blink8z (new value 11) bitstreams and download them onto the FPGA.

Retrieving and Downloading Old Bitstreams

If required, the bitstreams can be created with QuickChange in one session and downloaded later.

The QuickChange software produces two bitstreams:

1. *OutputDesign*.bst, a complete bitstream which contains the original design with the changed constants.
2. *OutputDesign*.win, a windowed bitstream which contains only the constants (absolute) or only the changed constants (relative).
3. If the bitstreams generated in the steps above are to be downloaded onto an FPGA in a separate session, power down the chip at this point.
4. Power up the device when ready to resume the session and invoke the *Bitstream Download* dialog box.


Bitstream Download

Active Design:

blinker : h:\qchange

Design Bitstreams:

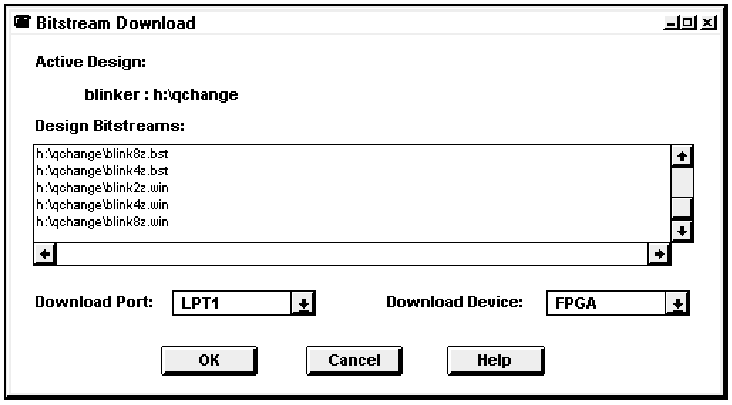
h:\qchange\blinker.bst
h:\qchange\blink2z.bst
h:\qchange\blink8z.bst
h:\qchange\blink4z.bst
h:\qchange\blink2z.win

Download Port: LPT1

Download Device: FPGA

Bitstream Download with *.bst and *.win Files

5. Select blinker.bst from the list and press OK. The bitstream will be downloaded onto the FPGA.
6. Scroll down to the bitstreams with the *.win extensions.


Bitstream Download

Active Design:

blinker : h:\qchange

Design Bitstreams:

h:\qchange\blink8z.bst
h:\qchange\blink4z.bst
h:\qchange\blink2z.win
h:\qchange\blink4z.win
h:\qchange\blink8z.win

Download Port: LPT1

Download Device: FPGA

Bitstream Download with *.win Files

6. Select the blink2z.win bitstream and download it onto the FPGA. Other bitstreams can be selected and downloaded in the same fashion.

Making Relative Bitstreams

The configuration contents of any bitstream that was previously generated by the QuickChange software can be displayed by choosing the appropriate name in the *Design* field of the *QuickChange* dialog box.

For example, to inspect the configuration of blink2z, scroll the design list and select the blink2z design. The information in the *Configuration Information* fields below changes immediately to reflect the newly selected design.

In relative cache logic, when a base line bitstream is selected and displayed in the *Design* field, its associated *Old Value* is shown. Information entered in the *New Value* field is implemented relative to that specified in the *Old Value* field. For absolute cache logic, the *Design* name is irrelevant because the bitstream location is changed in an absolute fashion.