Integrated Development System - Figaro User Guide

June 2002

Welcome to the Integrated Development System (IDS) from Atmel Corporation. This versatile system works with a variety of CAE platforms, providing a range of design entry and simulation options when designing an Atmel field programmable gate array (FPGA). ViewLogic's Workview Office and Powerview products, and OrCAD Express for Windows are supported for schematic entry and simulation. Simulation platforms supported also include Vital VHDL systems from Model Technology, and the Verilog-XL simulator.

Synthesis tools from Synopsys, Viewlogic, Everest Design Systems and Exemplar Logic are integrated into the system to provide optimum results. Atmel's IDS lets designers create fast, efficient, and predictable designs with AT40k Series FPGAs.

IDS has integrated an HDL planning environment to create efficient, technology independent VHDL and Verilog designs on Atmel FPGAs. Currently Synopsys, Exemplar, and Everest synthesis software is supported.

IDS also supports designs in the Xilinx XNF format. It provides the user with the ability to translate a Xilinx XC3000, XC4000, or XC5200 family design into an AT40k device. Most of the XBLOX components in a XNF design are automatically mapped to Atmel *Macro Generator* components. XNF import is integrated into the Viewlogic design flow.

Operation of the IDS is controlled by a single graphical interface, the Figaro Desktop. Figaro integrates the programs, links them through a unified data base, and provides a seamless working environment that allows the user to move easily among the programs. While the system is configured to help the user design and layout the chip by invoking the appropriate design modules, the user can also run programs independently from the Shell Window command-line.

It is recommended that the user also browse through the *AT40k IDS Tutorial*. The manual details steps to set up and implement sample designs using Figaro and the supported CAE interfaces. It also discusses ways to optimize designs for the AT6000 and AT40k architecture, so the user can create the chip in the most efficient 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*>
- Buttons to execute functions are shown as pictograms, e.g. Run

Product Updates

Updates are made available to users during the maintenance period at no charge to the user. Each update comes with its own installation program, release notes, and any special instructions that might be necessary to make the transition to the new software version.

Sales Representatives

Atmel sales representatives are ready to assist with pre-sales questions, product literature, price information, and product availability. To contact a local sales representative, please call Atmel at 408.441.0311 during normal business hours.

Customer Service

Atmel Corporation Customer Service provides software and hardware support and assists customers in uploading and downloading files.

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

1. Calling Customer Service at 408.436.4119 between 9 am and 5 pm, Pacific Standard Time
2. Sending electronic mail to fpga@atmel.com
3. Accessing Atmel’s web site at http://www.atmel.com
4. Faxing to 408.436.4200
5. Writing to:
   Atmel Customer Service
   2325 Orchard Parkway
   San Jose, CA 95131
   USA

System Basics

The Integrated Development System (IDS), otherwise known as Figaro, is designed to provide push-button capability in the design of Atmel FPGAs. This allows the user to either automate the design process, or optimize the design by invoking manual placement and routing options.

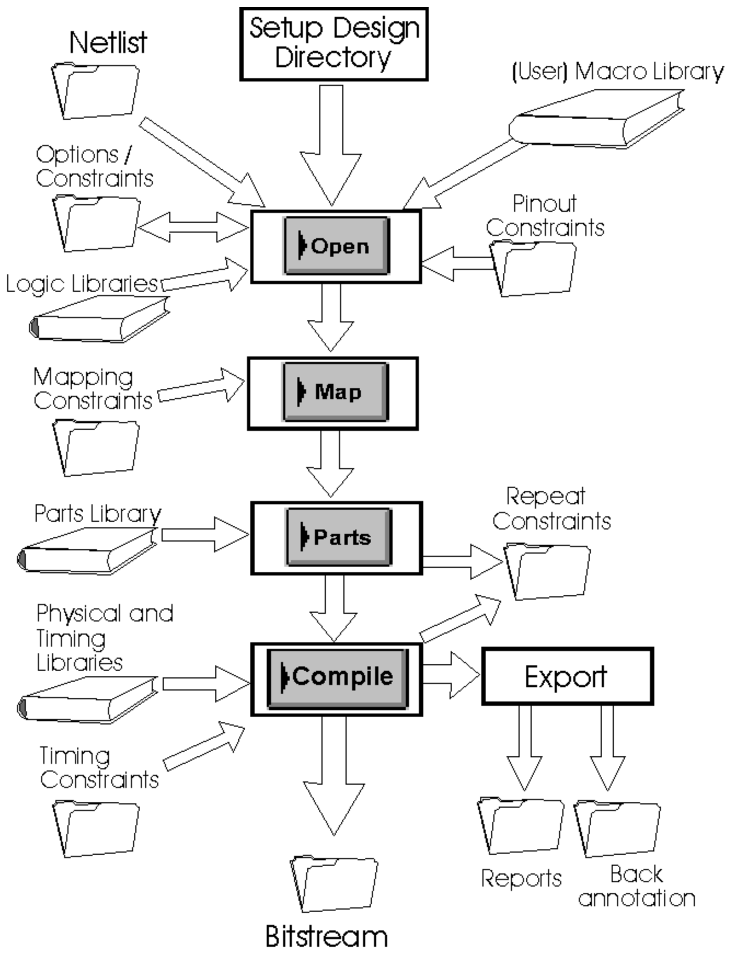
IDS supports many forms of design entry which include schematics, equation entry, and high-level language design. The system then automatically places, routes and optimizes the design to fit into the target device. Once that is completed, the user has a choice of either downloading the data onto the FPGA, Serial Configuration Memory, or creating a library of user defined macros (UDMs) for future applications. Expanded design analysis features provide delay estimates across the chip, as well as export timing reports and back annotation files for many different simulators.

The general concepts and organization of the design phases are discussed in this chapter, along with the following topics. To learn how to enter, simulate, compile, and back annotate a sample design with different menu options, refer to the *IDS Tutorial* for more information.

- IDS Design Flow
- Designs and User Libraries
- Common Design Information
- On-line Help
- The Transcript Window
- The Shell Window
- Starting and Ending a Session

IDS Design Flow

The following diagram summarizes the IDS design flow functions for bitstream creation.



IDS Design Flow

Designs and User Libraries

The design directory holds all files related to the design. This directory allows grouping of related designs and provides a consistent structure so that Figaro can handle design management tasks for the user. By Viewlogic and Intergraph convention, the design directory is analogous to the project directory.

Design files include all schematic-related and Figaro data base files. A design directory must be set up before schematic entry, simulation, or layout of a design can commence. This directory is the only place searched by Figaro for the design files.

See the *IDS Tutorial* for details on setting up a design. As a part of the process, the directory for storing all the design data must be identified to the system.

User Libraries

IDS supports the application of user libraries. The user library management features are intended to facilitate design reuse and help create high performance circuits. These libraries are repositories for user defined components which have been created by the *Macro Generators*, user schematics, or language-based entry. The system allows the user to specify as many libraries as needed. A common practice is to set up a design-specific library and maintain another that contains common macros shared by different designs.

A library consists of several parts which include the layout files, the CAE system specific files (schematics, symbols, etc.), and any related files that were used to produce the layout. Figaro will maintain all files that go into and out of libraries as well as set up the appropriate control files for the CAE system to access the libraries.

The *Library>Library Setup* dialog box is used to create a user library. Once specified, Figaro will create the layout library file as well as the library sub-directory for storing the rest of the library information. The *Figaro* chapter in the *IDS Tutorial* will provide details on this structure. Also refer to specific platforms under “CAE Interfaces” in the *Tutorial* for details on design related files that are stored in the library.

Sample structures for the PC and Workstation environments are provided below.

For the PC

If a layout library was defined as c:\AtUser\4bitalu*user*.lib the following structure would be created:

```
c:\AtUser\4bitalu\
            user.lib
            user\
                macro1\
                    macro1 design files
                    CAE system files
                macro2\
                    macro2 design files
                    CAE system files
                CAE system files\
```

For the Workstation

```
/AtUser/4bitalu/
            user.lib
            user/
                macro1/
                    macro1 design files
                    CAE system files
                macro2/
                    macro2 design files
                    CAE system files
                CAE system files/
```

Common Design Information

Figaro.ini

The user can customize projects by configuring such items as pin package specifications, design check rules, automatic place and route performance, design analysis parameters, bitstream format, and numerous ancillary screen output functions. The user can determine the default setting each time when beginning a new design.

In the design settings, the user can define such things as libraries for design data, and where output is to be stored for each design.

Figaro utilizes two configuration files, the figaro.ini and *DesignName*.ini files, to store user and design settings.

The user environment is defined in the figaro.ini file. This file contains such information as individual user defined (or default) display threshold and color assignments, and tracks all designs from a single location. The contents of the figaro.ini file can be changed under the *Options>Options* and *Options>Display Options* menus. This file is stored in the directory where Figaro is run from. The user should ensure that the program is started from the same directory each time so the required setup information can be reloaded.

DesignName.ini

The *DesignName*.ini file contains design settings that directly affect the design data, independent of the user's environment. This file is maintained as needed. It is updated when changes to any design specific setting, such as defining a user library, are made.

The *DesignName*.ini file is stored in the design directory. It will be loaded whenever the design is selected via the *File>Design Setup* or *Open Design* menu options. The file is given the same name as the design. For the example design called “4bitalu”, there will be a file named 4bitalu.ini in its design directory.

The following output files contain information that reflect the operations of the software:

DesignName.log

The *DesignName*.log file contains a list of all operations initiated for a design. Diverse information about the design setup, design checks done when a design is opened, placement and routing statistics and many other details are stored in this file. All information that is displayed in the transcript window will be stored in the log as well. A toolbar button is available for viewing of the log file. This file should be reviewed when any problem is encountered during execution of the design flow.

Program Output Files

The Integrated Design System is composed of many different programs, and the user will encounter a good variety of files in the design directory. A detailed list of all program output files will be found in the following documentation:

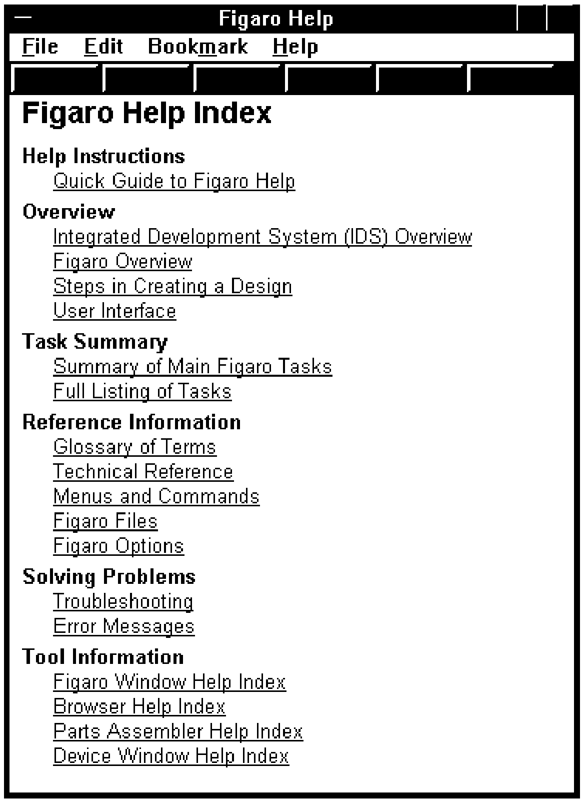
- *IDS Tutorial*, Figaro - Figaro Files.
- *Technical Reference & Release Notes,* Design Files.

Backup Files

Programs that modify the data base files generate a backup file. When a new copy of most any file in the design is produced, the previous copy is saved, with a "~" character as the last character of the file extension. For example 4BITALU.LOG, would be saved as 4BITALU.LO~.

On-line Help

The *Help* function contains detailed information on all parts of the IDS. This function is context sensitive, but can also be invoked directly from the menu bar. The "Figaro" section of the *IDS Tutorial* contains detailed information on how to invoke and get *Help* on specific topics. An example of the *Help* window is shown below.



Figaro Help Index

Help Instructions
Quick Guide to Figaro Help

Overview
Integrated Development System (IDS) Overview
Figaro Overview
Steps in Creating a Design
User Interface

Task Summary
Summary of Main Figaro Tasks
Full Listing of Tasks

Reference Information
Glossary of Terms
Technical Reference
Menus and Commands
Figaro Files
Figaro Options

Solving Problems
Troubleshooting
Error Messages

Tool Information
Figaro Window Help Index
Browser Help Index
Parts Assembler Help Index
Device Window Help Index

The Help Window

The Transcript Window

All information about processes within IDS is displayed in the transcript window. The information displayed here is also stored in the design log file. Normal design information will be displayed in a standard font in black. All warnings are displayed in a bold font in blue. All errors are displayed in the same bold font in red. The user should check this window first to determine the progress of a design.

Refer to the “Figaro” section of the *IDS Tutorial* for more details on the transcript window.

The Shell Window

The *New Shell Window* button in the vertical menu bar will create a window for manipulation of files in the design directory. Common operations such as editing simulation command files or creating constraint files can easily be done, as the shell is invoked in the design directory.

Starting a Session

Before beginning a session, verify that the appropriate environment variables are set for Figaro and the selected CAE system. These variables are discussed in the “Installation Guide” as well as the Tutorial for the specific CAE System.

To invoke Figaro from within Windows, double click on its icon, or enter the command “figaro” if in UNIX. Set the active design by using the icon or *File>Design Setup* menu option.

Subsequent chapters will discuss each design phase or activity represented by the Flowbar buttons.

Ending a Session

Select the *File>Exit* menu option to end the Figaro session. A prompt will be displayed to confirm exiting from the Desktop. If a design has been modified in any way, an additional prompt will be issued to save the design.

Design Entry

The Integrated Development System (IDS) offers a wide array of interfaces to work with design input processes such as schematic capture, equation entry, macro generation, and HDL planning and entry. Once the design is entered, it can be simulated. A netlist can then be created and the layout completed based on the specified CAE system.

Figaro supports Viewlogic tools for schematic capture. This platform is fully integrated with IDS so the interface is transparent to the user when moving between Figaro and the third party system during all phases of the design.

Designs requiring equation entry are also supported by IDS. Although ABEL and CUPL users must run these tools on PCs outside of Figaro they can bring the resulting *.tt1, *.tt2, or *.pla, formatted files into IDS for optimization and/or conversion into schematics or components for the user library in the PC. The macros created can optionally be simulated with any of the supported CAE tools.

The user can also take advantage of the *Macro Generators* in IDS to create efficient and optimized building blocks for the design. These functions range in complexity from simple gates to complex parallel pipelined multipliers. The vast array of *Generators* will provide more opportunities and flexibility when implementing a design.

Another possible method of design input is via high level design language (HDL) entry. HDLPlanner offers an environment that fosters code portability and facilitates the use of optimized macros. The user can take advantage of functional modules previously optimized for the AT40k architecture and apply them to current or future designs.

To support HDL Entry, the Atmel macro library has been translated for third party vendors of synthesis tools such as Viewlogic, Synopsys, Everest, and Exemplar Logic. These synthesis tools are integrated into Figaro to facilitate the high level design entry process. As a result, performance can be optimized through operator inferencing and use of the Atmel *Macro Generators* .

The AT40k library of components can be divided into 2 types of macros: functional and dynamic. Functional macros are components with fixed functions, such as the 2 input AND gate. Dynamic macros are designed to allow user specification of any desired function, to be attached as an attribute via an equation string, on the symbol. This should be used only when a specific function for an AT40k core cell is required. Designs targeted to AT40k can use a mix of dynamic and functional macros. Please refer to the *AT40k Macro Library* for more details on the list of all macros and the attributes available.

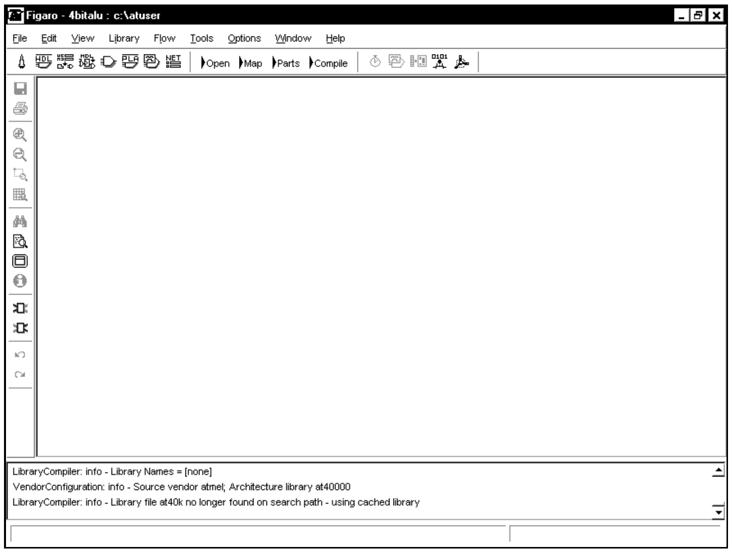
Once a design has been entered and synthesized with optional tools, a netlist can be generated and used for placement and routing.

Schematic Entry

The Integrated Development System is set up to provide the user direct access to a variety of schematic capture tools from the Figaro Desktop. The products supported are:

- Viewlogic PC products Workview Office, PROSeries, and WorkviewPlus; Sun workstation products Powerview, and Workview
- OrCAD Express for Windows

To start schematic entry from Figaro, click on the *Schematic Entry* button on the Flowbar as shown. The CAE tool of choice will be brought up within the Figaro environment.



```
LibraryCompiler: info - Library Names = [none]
VendorConfiguration: info - Source vendor atmel; Architecture library at40000
LibraryCompiler: info - Library file at40k no longer found on search path - using cached library
```

Schematic Entry

System Setup

Figaro automatically sets up the paths and directories needed for the Atmel libraries, user libraries, and the design directory to interface with the CAE system specified.

The Atmel FPGA library is used to reference logical and timing models for every cell in the library, while the user libraries contain custom macros created for functions specific to current or prior designs. The design directory houses all the output files from both the schematic tool and Figaro that are related to the design.

Figaro also automatically calls up the CAE tools in the IDS environment and integrates the process into the design flow for netlist creation.

Viewlogic

IDS controls the complete process of entering schematics with the Viewlogic tool. Once a design has been set up via the icon or *File>Design Setup* dialog box, the *Schematic Entry* button can be used to bring up the schematic editor on the current design.

As part of the process, Figaro will update the project.vpj, registry, and viewdraw.ini files for Workview Office, or the project.lst and viewdraw.ini files for other Viewlogic systems. It will also invoke the appropriate Viewlogic tool for schematic entry. The following table lists the platform and the tool invoked:

CAE Platform	**Tool**
Workview Office	Viewdraw
PROSeries	PROcapture
WorkviewPlus	Viewdraw
Powerview	Viewdraw
Workview	Workview

Once in the schematic tool, all macros used must be: a) Atmel FPGA library components, b) user library components, or c) hierarchical blocks that have been created with the Atmel FPGA library. Viewlogic BUILTIN library elements cannot be used as the Figaro placement and routing tools will not recognize them.

Once the design has been captured, the user can proceed to simulate or compile. For compilation, the Viewlogic Wir files are read directly by Figaro using the *Open* button.

OrCAD Express for Windows

IDS controls the complete process of entering schematics with the OrCAD tool. Once a design has been set up via the icon or *File>Design Setup* dialog box, the *Schematic Entry* button can be used to bring up the schematic editor on the current design.

Once in the schematic tool, all macros used must be: a) Atmel FPGA library components, b) user library components, or c) hierarchical blocks that have been created with the Atmel FPGA library.

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.

Once the design has been captured, the user can proceed to compile (Pre-layout Functional Simulation is not supported for designs using the dynamic macros). For compilation, the OrCAD EDIF files are read directly by Figaro using the *Open* button.

A VITAL VHDL library is supported for Post-layout Simulation of OrCAD designs. For more details please refer to the OrCAD tutorial.

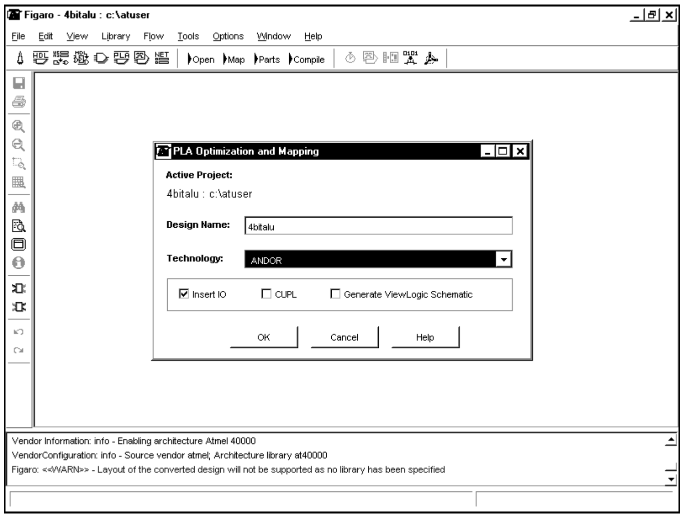
PLA Optimization

The Integrated Development System provides the interface to PLA formatted designs from the Figaro Desktop. Before invoking this function the user needs to enter, edit, and translate their equation based designs to the PLA format. Equation entry provides a compact way of transferring logic descriptions to the Atmel FPGA design tools. The PLA optimization function allows the user to perform logic minimization, technology mapping, and layout optimization from Figaro. The resulting output can be converted into a hard macro or a schematic for use in the design and simulated as needed. The PLA compilers supported are:

- DATA I/O ABEL
- CUPL

The ABEL and CUPL compilers are PC based products, and the resulting files can be set up to interface directly with the Figaro software on either the PC or the workstation.

Click on the *PLA Optimization* button on the Flowbar as shown to initiate the optimization and conversion process.



Active Project:
4bitalu : c:\atuser

Design Name: 4bitalu

Technology: ANDOR

[✓ Insert IO] CUPL Generate ViewLogic Schematic

```
Vendor Information: info - Enabling architecture Atmel 40000
VendorConfiguration: info - Source vendor atmel; Architecture library at40000
Figaro: <<WARN>> - Layout of the converted design will not be supported as no library has been specified
```

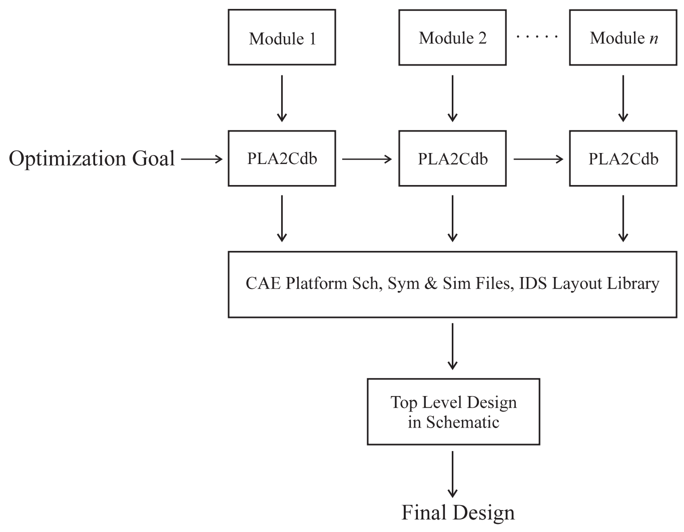
PLA Optimization Dialog Box

Design Flow

This module is invoked as another means of design entry when an optimized equation-based function is needed for a design.

The Figaro design flow allows a mixed method of design entry. The design sub-modules are specified in ABEL or CUPL format, optimized individually, and become sub-blocks or complete designs. The individual sub-modules are then instantiated in a top level schematic with the appropriate I/Os added to form a complete design. Optionally the PLA description can constitute a complete design with I/Os automatically inserted by this tool.

The design flow utilizing PLA entry is illustrated and described in detail below.


Module 1
Module 2
Module n
Optimization Goal
PLA2Cdb
PLA2Cdb
PLA2Cdb
CAE Platform Sch, Sym & Sim Files, IDS Layout Library
Top Level Design in Schematic
Final Design

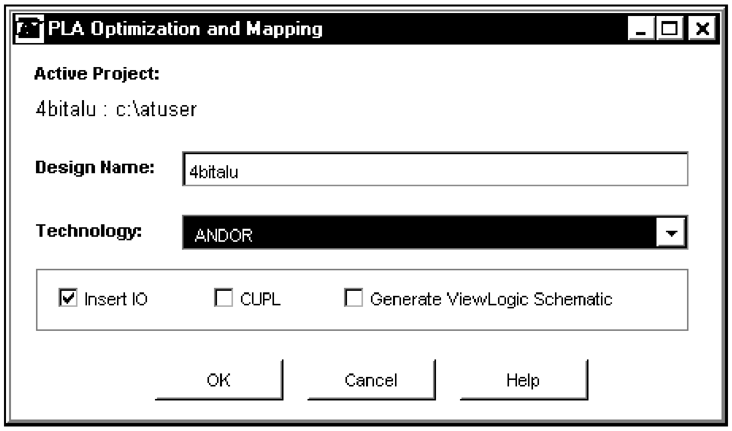
PLA Design Flow in IDS

System Setup

PLA Optimization is tailored for compiling optimized user-defined blocks. To ensure that the ABEL or CUPL PLA outputs are sent to the correct directories, the design files should be in place prior to initiating *Optimize PLA*. Refer to the *CAE Interfaces* section in the *IDS Tutorial* for a detailed description of the Setup process.

Conversion and Optimization

The process of creating a design starts with the design and library setup as already mentioned. The design is partitioned into sub-modules and an ABEL or CUPL description is written for each of them. This is done independent of the Figaro environment. The user should run the appropriate program and options to generate the files needed. These files should be stored in the design directory. When the *Optimize PLA button* is selected, a dialog box appears in the Figaro Desktop which looks like the following.


PLA Optimization and Mapping

Active Project:
4bitalu : c:\atuser

Design Name: 4bitalu

Technology: ANDOR

[✓] Insert IO [ ] CUPL [ ] Generate ViewLogic Schematic

The PLA Optimization and Mapping Dialog Box

The options in the dialog box are explained in detail below.

InsertIO The user can invoke the automatic pad insertion algorithm by selecting this option. By default, ITTL, OD and ODEN pad buffers are inserted. However, if the file *designName*.pin is placed in the design directory, it can be specified to override the default pad types. This file should contain a signal name and a pad type, each on a separate line. An example .pin file is given below.

```
IN1 ITTLP
IN2 ITTLP
OUT1 ODF
OUT2 ODF
```

Generate ViewLogic Schematic This switch is used to create a ViewLogic Schematic.

CUPL This switch is used to read *.pla files produced by the CUPL compiler. By default, the *.tt1 or *.tt2 files produced by ABEL compilers are used.

Technology If *Technology* is set to AND XOR, pre-optimized PLA files (*.tt1 for ABEL and *.pla for CUPL) are taken as input and AND-XOR optimization is performed

Some designs can be more efficiently optimized as AND-OR equations instead of AND-XOR equations. Although AND-OR minimization is not done using PLA optimization, it will still accept files that are optimized within the ABEL environment. After such optimization in ABEL, a PLA file with the extension *.tt2 will be created and should be placed in the design directory. This file will be used for technology mapping of AND-OR equations.

Note to Users of ABEL 5.X Software

The ABEL compiler version 5.1 onwards does not produce a *.tt1 file. Therefore, a *.tt1 file cannot be used for AND-XOR optimization. If AND-XOR optimization is needed, users can copy and rename the *.tt2 file as *.tt1 and perform AND-XOR optimization.

Macro Generators

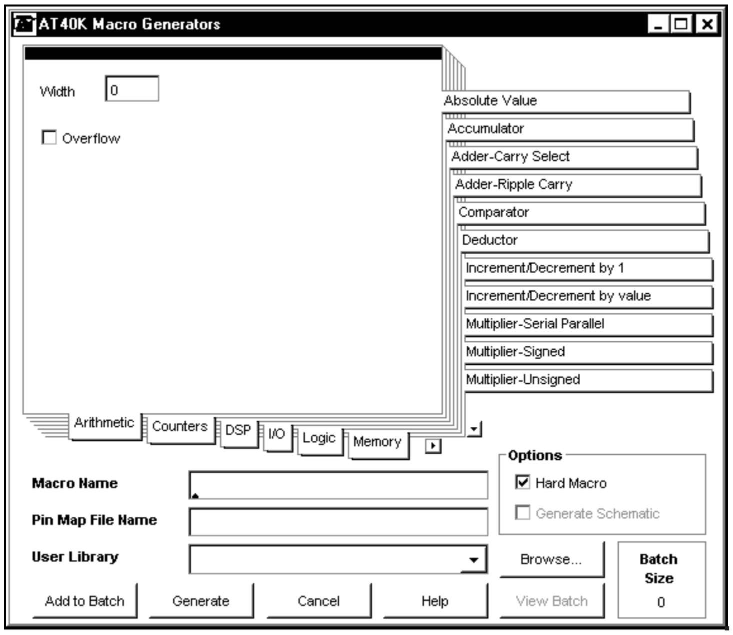
The *Macro Generators* module provides the user with the ability to construct highly efficient counters, adders, and other structured blocks. It is designed to allow easy inclusion of new *Generators* from Atmel and other third party vendors in the future. The core set of *Generators* is based on the EIA Library of Parameterized Macros (LPM) standard. This core set has been enhanced to achieve a superset to this specification.

A complete explanation of each Generator available at the time of release is provided in the *Technical Reference & Release Notes*. Because of continuous additions to the *Macro Generators*, some new functions may not be covered in the manual. On-line *Help* provides the most current information, and is discussed in this chapter. Both on-line *Help* and the *Technical Reference* manual provide information on the parameters available, pin type descriptions, and truth tables. Statistics on speed, delay, size, gates per cell, and power consumption are also addressed.

The basic user interface and details of how to get more information about a specific Generator are explained below. A step by step description on how to run a Generator is also available in the *AT40K IDS Tutorial* manual.

Design Flow

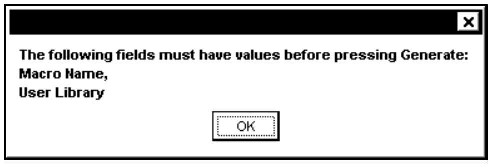
A design and its associated user library must be set up prior to the initiation of this module. Once the *Macro Generators* button is selected, the following dialog box will appear in the Figaro Desktop as shown below.



Macro Generators Dialog Box

The tabs along the bottom represent the various categories of *Macro Generators* available. The arrow buttons along the bottom allow the user to scroll through the entire selection. The tabs along the side display the *Generator* functions. A function may be available for more than one category. The arrow buttons along the side allow the user to scroll through the entire selection.

The user must enter the name of the macro to be created before the Generators can put it into the library. Specify the appropriate values for the parameters such as the input or output widths of the function. If any required field is not filled in prior to selecting the Generate button an error message will be displayed.


The following fields must have values before pressing Generate:
Macro Name,
User Library

Macro Generators Error Message

To get details on component functionality and the associated parameters, press the Help button.

Verify the library path in the *User Library* list box or specify a new library by pressing the *Browse* button.

When *Macro Generators* is run, IDS will typically create a schematic automatically for the specified CAE system. However, for certain CAE platforms, a schematic is not required and so schematic generation is an optional step which is controlled through a check box in the *Macro Generators* dialog box.

The *Macro Generators* create a fixed layout hard macro for Figaro and the corresponding CAE systems symbol and simulation models. In the *Macro Generators* dialog box, the user has an option to generate just a schematic and simulation model for the desired macro by clearing the *Hard Macro* check box. This allows the user to edit the schematic and change the logic of the macro generated. However, the layout for the macro is not generated. It is most efficient to use the output of the *Macro Generators* as a hard macro.

The default pin names on the components generated by the *Macro Generators* can be changed by specifying the user defined pin names in a file and providing the name of that file as the value for the *Pin Map File Name* option in the *Macro Generators* dialog box. The optional pin map file allows alternate pin names to be specified.

Press the Help button in the dialog box to get information on the default pin names of the component. The pin map file syntax is as follows:

```
map default_pin_name user_defined_pin_name
```

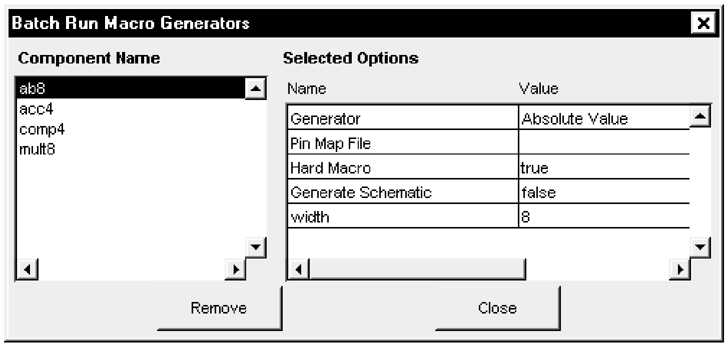
As an example, the default pin names generated for a 4 bit adder component are DATAA[3:0], DATAB[3:0], SUM[3:0], CIN, COUT. To change these pins to A[3:0], B[3:0], Q[3:0], RCI, RCO respectively the corresponding pin map file should contain the following lines.

```
map DATAA A
map DATAB B
map CIN RCI
map COUT RCO
```

map SUM Q

Several macros can be generated in batch mode by using *the Add To Batch* function in the *Macro Generators* dialog box. To use the Macro Generators *Add To Batch* function follow the steps below.

- Store the macro settings in the batch by pressing *Add To Batch* after specifying the parameters for each macro. The *Batch Size* on the lower right corner of the *Macro Generators* dialog box will get incremented by one.
- Review the macros by pressing *View Batch* to bring up the following dialog box. To remove a macro from the batch list, select a macro and press *Remove*. Press *Close* to return to the *Macro Generator* dialog box.



Component Name
ab8
acc4
comp4
mult8

Name	Value
Generator	Absolute Value
Pin Map File	
Hard Macro	true
Generate Schematic	false
width	8

View Batch Dialog Box

- Press Generate when all macros are configured to the appropriate settings and *Batch Size* shows the final count. After each macro is generated and stored in the user library, the *Batch Size* counter is decremented accordingly. The following dialog box appears after successful completion of the *Macro Generators* batch run.

Macro Generator batch run completed succesfully

Completed Batch Run Dialog Box

User Library Structure

Although user libraries are discussed at great length elsewhere, some important information that relate to the *Macro Generators* are highlighted below.

Output Files

Once all of the needed information is specified, *Macro Generators* can be invoked to create a complete user library component with the associated output files as follows:

Layout Except for the I/O modules, all *Generator*s create a hard layout by default which is stored in the Figaro format user library. Clearing the *Hard Macro* option instructs Figaro to leave the circuit as a soft macro.

Schematic The *Generators* produce schematics for the Viewlogic platform. For Viewlogic, the schematic is optional as the actual connectivity information is provided in the netlist (wir) files.

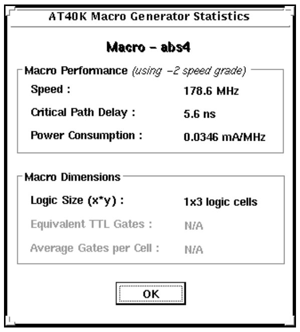
Simulation All *Generator*s produce information which can be used to provide correct functional simulation. For Viewlogic, this is contained in the design specific wir file. For all other tool flows, either a Verilog or vhdl file is produced to provide the functional model.

Symbol To facilitate design entry using the *Generators*, symbols are automatically created for all supported schematic entry tools. The symbols will contain all the interface pins for the macros generated, with inputs on the left and outputs on the right.

HDL Support To support users who enter their designs with VHDL, a template of the macro is created. This template will contain the macro name and a list of the input and output pins in VHDL syntax. It can be found in the library directory inside the component's sub-directory with the extension *.vht. Structural Verilog and VHDL descriptions of the macros are provided for *Tool Flows* that require them. These will be found in the library directory under the verilog and vhdl sub-directories. . The postsim directory contains descriptions with buses flattened in order to facilitate *Post-layout Simulation*.

Statistics

After the *Macro Generator* has completed running, a dialog box will be displayed showing statistics of the macro. It will look like the following.


AT40K Macro Generator Statistics

Macro – abs4

Macro Performance *(using –2 speed grade)*

Speed :	178.6 MHz
Critical Path Delay :	5.6 ns
Power Consumption :	0.0346 mA/MHz

Macro Dimensions

Logic Size (x*y) :	1x3 logic cells
Equivalent TTL Gates :	N/A
Average Gates per Cell :	N/A

Macro Generators Statistics Box

The information displayed in this dialog box can also be found in a file, identically named as the user library directory, with the *.sts extension. For example a library named *user*.lib will have a statistics file named *user/user*.sts. Additionally in the library directory under each component will be a file called *macro*.lst. This file will contain details of the parameters used to generate the macro. It will also contain a file called *macro*.sts with the above statistics for the macro.

HDL Planner

This section presents a design development environment, called HDLPlanner, for planning and creating HDL designs for Atmel FPGAs. Special HDLPlanner features allow the incorporation of technology specific information during the early planning stages so a 100% technology independent design can be maintained. Additionally, HDLPlanner prepares the input data to take advantage of the synthesis software for optimum results.

The software's interface can define and instantiate pre-verified VHDL/Verilog modules in the design files quickly. The modules can be parameterized for bit-widths and clock/reset schemes.

The HDL Planner software is tightly integrated with the back-end layout generation engines. Using the *Macro Generators*, design modules can be automatically translated into hard macros with efficient implementation in Atmel FPGAs. On-line module statistics on area and delays can be accessed easily and used to estimate design performance.

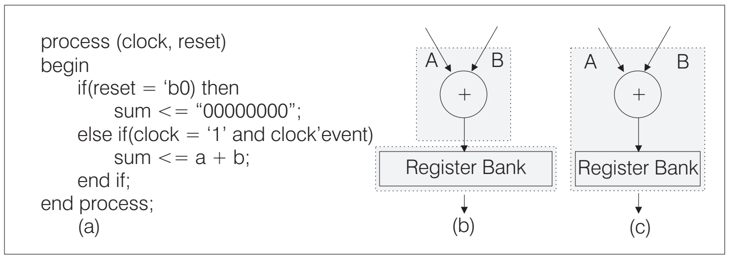
Finally, HDL Planner is an open system. Using its IPEditor user interface, users can integrate their modules within HDL Planner and access them as if macros were supplied from the factory

It is an open knowledge archival system. Previously synthesized modules become an effective resource because they can be reused in future designs.

Synthesis Technology Limitations

The benefits of hardware description languages (HDL) include the ability to parameterize modules and create technology independent designs. Parameterization allows generic definition of a module to be defined once but used multiple times with different parameters. The support for parameterization encourages design reuse and simplifies design maintenance. Technology independence in HDL also allows designs to be written once and then targeted to a large number of FPGAs or ASICs.

However HDL descriptions must be synthesized and optimized to realize their gate level implementation for placement and routing. Often, that means relying on synthesis software capabilities for module inferencing and logic optimization. Unlike manually entered circuits, HDL designs are more inefficient because of the inability of synthesis tools to fully incorporate technology specific information during optimization. A notable improvement is operator inferencing. This methodology identifies arithmetic and boolean operators from a design and links them to their preferred implementation from the technology library. However operator inferencing does not guarantee optimal results, as illustrated below.



```
process (clock, reset)
begin
     if(reset = 'b0) then
          sum <= "00000000";
     else if(clock = '1' and clock'event)
          sum <= a + b;
     end if;
end process;
     (a)
```

(a) VHDL template of registered adder, (b) Optimized implementation, (c) Preferred implementation.

Operator Inferencing Limitations

Additionally, only operators that are supported in the language can be inferred. Macros such as counters and FIFOs cannot be inferred. They must be instantiated using modules defined in a technology specific library, and technology independence is compromised in the process.

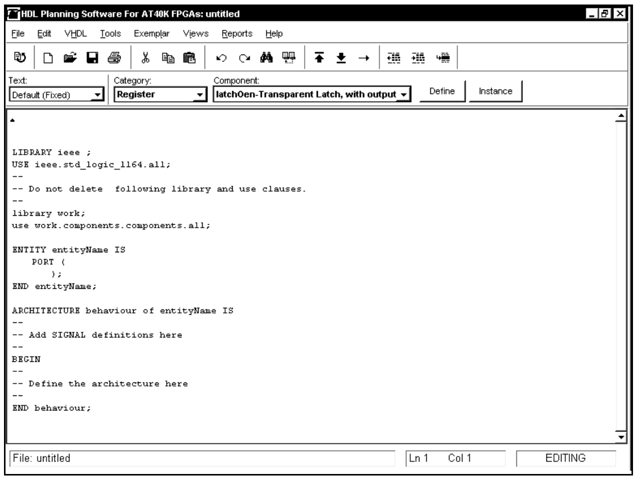
FPGA Technology Specific Considerations

Synthesis tools perform architecture specific optimization without considering the technology contents of the FPGA. Such items include clocks and resets, tri-states, wired logic, I/O buffers, on-chip configurable memory resources and their address decoding circuitry. A cost driven optimization of these resources, unavailable from synthesis tools, is important for achieving optimum performance from the underlying FPGA technology.

HDL Planner's design planning environment encourages the users to follow *meet-in-the-middle* methodology for creating HDL designs. It contains a set of well defined methodologies that can be used to create technology independent descriptions. The graphical interface allows the user to address architecture, technology, and layout specific issues earlier in the design process, resulting in a simplified and shortened design cycle. It also provides area and delay statistics for corresponding performance estimates.

Graphical User Interface

HDL Planner Graphical user interface can be divided into three separate components as outlined below.


HDL Planning Software For AT40K FPGAs: untitled
File Edit VHDL Tools Exemplar Views Reports Help
Define
Instance
EDITING

Text: Default (Fixed)
Category: Register
Component: latchOen-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

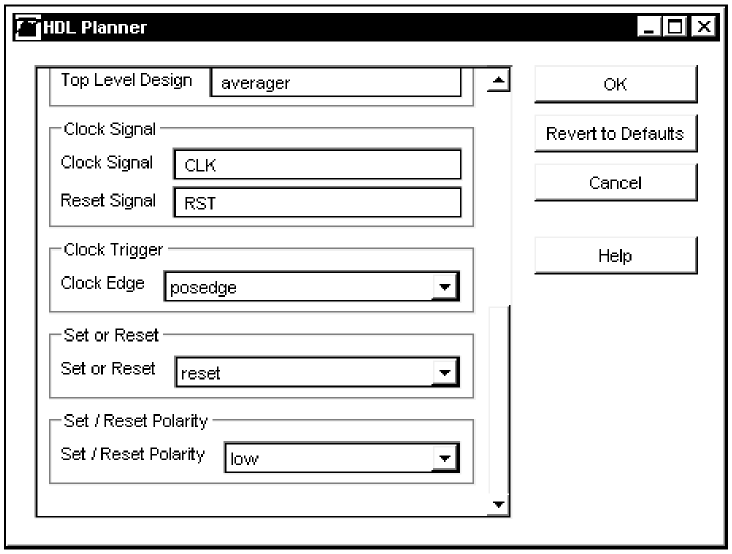
HDL Planner Graphical User Interface

Design Editor

The built-in editor contains buttons to create and save projects, as well as support basic text editing operations such as cut, copy, paste, search, and find etc. This UI specification conforms to the Windows 97 standard.

Module Definition and Instantiation Panes

Special list boxes and buttons are provided to select, define, and instantiate modules. (Refer to the Graphical User Interface above). Select a module and press the *Define* button. This inserts a generic definition of the module in the text window. Use the *Instance* button to instantiate a macro selected from the *Component* list box. Once the macro is instantiated, its parameters can be set by modifying the instantiation statement in the file.



Top Level Design: averager

Clock Signal
Clock Signal: CLK
Reset Signal: RST

Clock Trigger
Clock Edge: posedge

Set or Reset
Set or Reset: reset

Set / Reset Polarity
Set / Reset Polarity: low

HDL Planner Dialog Box

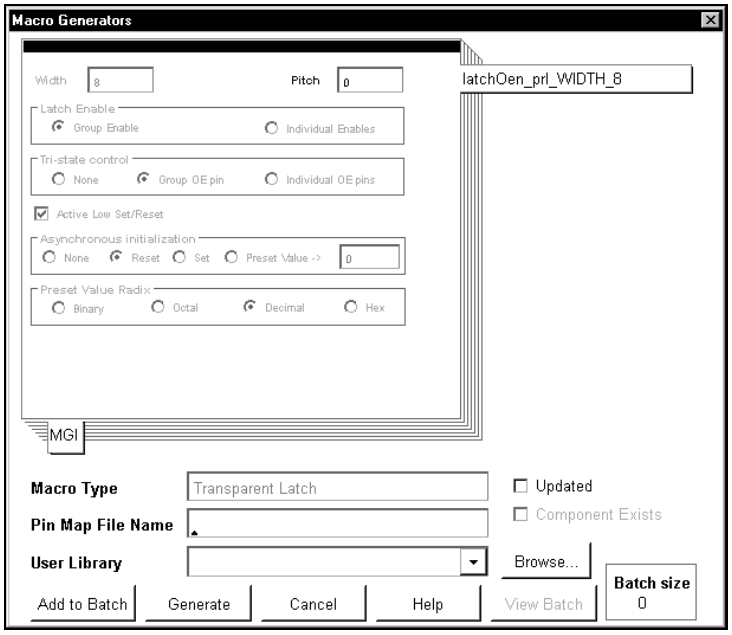
A module can be parameterized to account for the clock edge (positive or negative), as well as the set or reset pins and their polarities (high or low). The appropriate options should be selected before the component is defined or instantiated.

Resource Estimation and Automatic Macro Generation

Menu buttons are provided to access area and macro statistics of the component selected. Refer to the Table *Description of Important Menu Buttons* for further details.

The menu button *Invoke Macro Generators* is provided to create layouts for all components instantiated in the design. The *Macro Generators* dialog box is shown below.

NOTE Only those options that determine the physical properties can be supplied by the user. All other options are grayed out.



latchOen_prl_WIDTH_8

Width 8 Pitch 0

Latch Enable: [✓ Group Enable] Individual Enables

Tri-state control: None [✓ Group OE pin] Individual OE pins

[✓ Active Low Set/Reset]

Asynchronous initialization: None [✓ Reset] Set Preset Value -> 0

Preset Value Radix: Binary Octal [✓ Decimal] Hex

Macro Type: Transparent Latch

Pin Map File Name:

User Library:

☐ Updated

☐ Component Exists

Batch size 0

Macro Generators Dialog Box

Menu Item	**Menu Button**	**Brief Description**
VHDL[1]	Entity	Add HDL template for VHDL entity
	Architecture	Add HDL template for VHDL architecture
	Comp Declaration	Add component declaration statement
	Comp Instantiation	Add component instantiation statement
	Process	Add a process Statement
	Clocked Process	Add a clocked process statement
	If	Add if statement
	Case	Add case statement
	While Loop	Add while loop statement
	For Loop	Add for loop
	Signal	Add signal definition statement
	Variable	Add variable definition statement
	Constant	Add constant definition statement
	Type	Add type declaration statement
	library	Add library statement
	Package	Add package statement
Verilog	Module Definition	Add a module definition statement
	Module Instantiation	Add module instantiation statement
	Always	Add an always statement
	Clocked Always	Add clocked always statement
	If	Add if statement
	Case	Add case statement
	CaseX	Add casex statement
	Casez	Add casez statement
	For	Add for loop
	Repeat	Add repeat loop

[1] This item will be overlaid with Verilog if Verilog HDL is selected

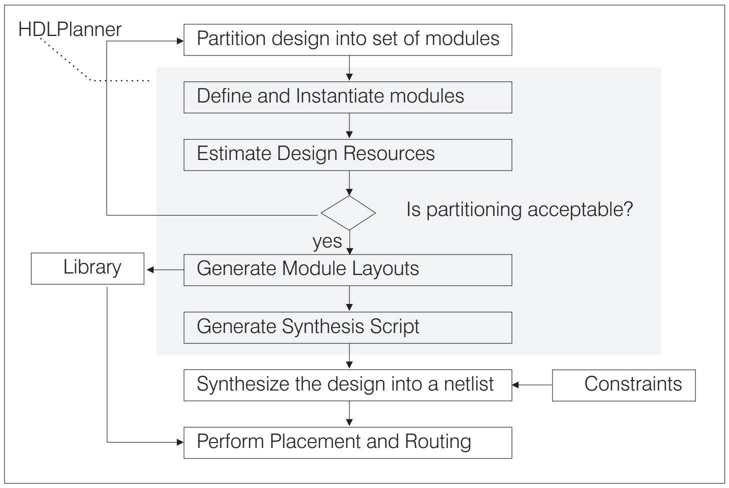
Menu Item	**Menu Button**	**Brief Description**
	While	Add while loop
	Continuous Assignment	Add continuous assignment statement
	Blocking Assignment	Add blocking assignment statement
	Non Blocking Assignment	Add non blocking assignment statement
	Register	Add reg statement
	Wire	Add wir statement
	Tristate	Add tri statement
	Define	Add define statement
	Parameter	Add parameter statement
	Defparam	Add defparam statement
	include	Add include statement
Design	Invoke Macro Generators	Invoke layout generator GUI
	Report Macro Information	Report information on modules used in design
	Generate Synthesis Script	Generate synthesis script
Exemplar[2]	Synthesis Tips	Access synthesis experience
	Integrate a User Macro	Invoke a software to integrate macro in HDL PLanner
Views	Behavioral	Display a behavioral description of a module
	Structural	Display structural description of a module
	Layout	Display a layout of a module layout in MGL
Reports	Area	Display area information for a module
	Delay	Display delay information for a module

Description of Important Menu Buttons

[2] Item Synopsys will be displayed if Synopsys synthesis software is selected

Planning HDL Designs

The process begins with a well thought out partitioning of a system into a set of modules as illustrated below. HDL Planner integrates the design planning philosophy into the process so the user can write a modular and hierarchical design description. Upon synthesis, a gate level netlist that is optimized for the target technology will be created. HDL Planner allows the user to estimate design resources and help avoid assumption changes late in the design cycle.


HDLPlanner
Partition design into set of modules
Define and Instantiate modules
Estimate Design Resources
Is partitioning acceptable?
yes
Library
Generate Module Layouts
Generate Synthesis Script
Synthesize the design into a netlist
Constraints
Perform Placement and Routing

HDL Planner Design Flow

HDL Planner Features and Benefits

Important features of the HDL Planner software and its built in methodology are outlined below.

Design Entry Specific

Design Editing The software has an editing environment for planning, entering, and maintaining HDL descriptions. Its comprehensive set of pre-verified templates of complex HDL constructs can be used to speed up design entry. Refer to the Table *Description of Important Menu Buttons* above. HDL templates can also be used to facilitate the learning of VHDL and Verilog language syntax.

Technology Independent Design Entry   Designs created in HDL Planner are 100% technology independent, conform to vendor laid out synthesis guidelines, and contain complete simulation models (so there are no black boxes).

Design Reuse   HDL Planner has a User Interface to easily define and instantiate pre-verified functional modules. These modules can be parameterized for bit-widths as well as clock and reset schemes.

Technology Specific

Links to Layouts   The *Macro Generators* interface of HDL Planner translates functional modules into layouts that are highly optimized for the architecture and technology.

Management of Clock and Reset Resources   HDL Planner simplifies the task of managing the vast clock, set/reset resources on the FPGA. This feature is especially useful as no known synthesis system supports module parameterization around clocks and resets.

Synthesis Tools Specific

Overcomes Synthesis Technology Limitations   HDL Planner can set up the data for synthesis to obtain the best output.

Tightly integrated with Synthesis Tools   Synthesis scripts generated by HDL Planner do not require user knowledge of technology specific directives that are needed for efficient synthesis.

Productivity Specific

Performance Estimation   On-line reports and statistics of reusable modules allow quick generation of performance estimates.

Shorter Design Cycle   Pre-verified, reusable components and automatic template generation minimizes the design cycle.

Software Architecture Specific

Completely Transparent   HDL Planner is an open system. Users can integrate their components and use them in their design process as if they were shipped from the factory.

Knowledge Archival HDL Planner has an open help system. The user can take advantage of previously synthesized modules and reuse them in future projects.

HDL Entry

With significant improvements in the quality of designs produced by synthesis tools, increasing numbers of circuit designers are adopting a top down methodology based on Hardware Description Languages (HDL) over the traditional design methodology of design entry with schematic capture systems. The top down design methodology (also called high level or textual design methodology), consists of working towards the physical implementation, by specifying the design behavior in HDL, and allowing synthesis tools to automatically translate it to optimized gate level connectivity under a set of design constraints. The gate level connectivity produced by synthesis tools is compiled by placement and routing programs.

This top down design methodology, which allows working at a higher level of design abstraction, has many advantages. A few of the important advantages are shorter design cycles, exploration of many architectural alternatives, ease of design maintenance and debugging, design re-use, and generation of hardware which is correct by construction.

Today's synthesis tools have the ability to synthesize designs under a wide range of design constraints. By specifying technology specific constraints, layout related information can be incorporated during synthesis to produce designs meeting desired performance criteria. Therefore, the ability of a synthesis tool to perform optimization under technology constraints is an important component of top down methodology. In this fashion, the constraint directed synthesis combines advantages of the top down approach as well as bottom up approach, an approach in which layout related issues are given more importance.

One important aspect of the latest synthesis products is the ability to support operator inferencing. With this feature, the tools can recognize functions such as adders, multipliers, comparators, etc. in a design. The process allows the synthesis tools to identify specific components that can be better optimized by the FPGA layout software.

This technique is fully supported in Figaro through the *Macro Generators Interface* (MGI) and specific libraries for the supported synthesis tools. HDL synthesis tools from Viewlogic, Everest, Synopsys (currently without MGI support), and Exemplar Logic can be used with Figaro and subsequent designs imported via a netlist interface.

Viewlogic

Viewlogic synthesis tools are supported via a technology specific library. These libraries are stored in the Atmel library directory and must be copied to the appropriate Viewlogic locations before synthesis is invoked. The currently supported tools include VHDLDES, PROSyn, Vsyn, and Aurora. All of these programs read as input the *.sml technology files and produce as output wir files. These files can then be read in directly by Figaro.

In addition to the *.sml files, other files are needed to run MGI. See the tutorial on Viewlogic synthesis for more details.

The basic design flow for synthesis is to first define the design via the Figaro *File>Design Setup* dialog box. This will set up the Viewlogic environment. Next, push the *Synthesis* button on the Flowbar to invoke the synthesis tool. Use the program to create various hierarchical levels of the design mapped to the Atmel FPGA architecture. Through operator inferencing, adders, multipliers, and comparators can be automatically placed in the output netlist. For any other structural components, the *Macro Generators* should be used in the design to provide the best possible performance for the FPGA.

Automatic pad insertion using the Viewlogic synthesis tools for the Atmel architecture is available. An outline of the process can be found in the *Tutorial*.

Refer to the “CAE Interfaces” section in the *AT40K IDS Tutorial* for design entry with the Viewlogic synthesis tools using the “averager” example. This will describe in detail the set up of the design, library, and technology files; hierarchical synthesis, creation and use of the *Macro Generators* components, MGI flow, and the final preparation of the design for placement and routing.

Synopsys

The Synopsys FPGA/Design Compiler is a powerful synthesis and optimization environment which can perform design synthesis under the user-specified constraints. It can be used with the Integrated Development System to optimize the layout and design of Atmel FPGAs.

Synopsys synthesis is supported via a technology specific library. These libraries are stored in the atmel/lib/synopsys directory, and must be set as a *Technology Library* before synthesis can be performed. The user should follow the appropriate Synopsys flow to synthesize the design and produce an EDIF netlist which can be read into Figaro.

The basic design flow for synthesis is to first define the design via the Figaro *File>Design Setup* dialog box. This will execute the appropriate design directory setup. Next, the synthesis tool is run using the *Synthesis* button on the Flowbar and can be used to create the various hierarchical levels of the design mapped to the Atmel architecture.

The *Macro Generators* should be used to replace structures such as adders, multipliers, etc. in the design to provide the best possible performance for the FPGA. Because automatic I/O insertion is available, there is no need to bring the results into a schematic capture tool for final assembly.

Refer to the *CAE Interfaces* section in the *AT40K IDS Tutorial* for design entry with the Synopsys synthesis tools using the "averager" example. This will describe in detail the design and library setups, hierarchical synthesis, creation and use of the *Macro Generators* components, and final preparation of the design for placement and routing.

Synopsys FPGA Express

The Synopsys FPGA Express can be used with the Integrated Development System to optimize the layout and design of Atmel FPGAs.

The user should follow the Synopsys FPGA Express flow to synthesize the design and produce an EDIF netlist which can be read into Figaro.

The basic design flow for synthesis is to first define the design via the Figaro *File>Design Setup* dialog box. This will execute the appropriate design directory setup. Next, the synthesis tool is run using the *Synthesis* button on the Flowbar and can be used to create the various hierarchical levels of the design mapped to the Atmel architecture.

Macro Generator components should be used to replace structures such as adders, multipliers, etc. in the design to provide the best possible performance for the FPGA. Because automatic I/O insertion is available, there is no need to bring the results into a schematic capture tool for final assembly.

Refer to the *CAE Interfaces* section in the *AT40K IDS Tutorial* for design entry with the Synopsys FPGA Express synthesis tools using the “averager” example. This will describe in detail the design and library setups, hierarchical synthesis, creation and use of the *Macro Generators* components, and final preparation of the design for placement and routing.

Everest

The Everest synthesis tool can be used with the Integrated Development System to optimize the layout and design of Atmel FPGAs.

The user should follow the appropriate Everest flow to synthesize the design and produce an EDIF netlist which can be read into Figaro.

The basic design flow for synthesis is to first define the design via the Figaro *File>Design Setup* dialog box. This will execute the appropriate design directory setup. Next, the synthesis tool is run using the *Synthesis* button on the Flowbar and can be used to create the various hierarchical levels of the design mapped to the Atmel architecture.

Everest synthesis tool can infer Atmel *Macro Generator* components for adders, counters, multipliers, comparators and ROM's in a design. Those Atmel *Macro Generator* components are black boxes and hence designs using inferred components cannot be simulated until the design netlist is read into Figaro. Once the design netlist is read into Figaro, the inferred components are automatically identified and the *Macro Generators* (also called MGI) dialog box will be brought up. The MGI dialog box lists all the inferred components and their parameters like width and function. In the MGI dialog box, users can change layout related parameters like area/speed optimization, or layout folding, and create a hard layout for the inferred components.

Because automatic I/O insertion is available, there is no need to bring the results into a schematic capture tool for final assembly.

Refer to the *CAE Interfaces* section in the *AT40K IDS Tutorial* for design entry with the Everest synthesis tools using the "averager" example. This will describe in detail the design and library setups, hierarchical synthesis, creation and use of the *Macro Generators* components, and final preparation of the design for placement and routing.

Exemplar Logic

Exemplar Logic synthesis is supported via a technology specific library. Technology libraries are provided for the Exemplar synthesis tools Leonardo and Galileo Extreme. These libraries are stored in the atmel/lib/exemplar/leonardo and atmel/lib/exemplar/galileo subdirectories and must be copied to the correct location ($EXEMPLAR/lib) before synthesis is invoked. Exemplar's ModGen Library for the Atmel FPGA is also provided along with the synthesis library.

The appropriate Exemplar flow should be followed to synthesize the design and produce an EDIF netlist which can be loaded into Figaro**.** Exemplar's synthesis tools can infer ModGen Components for arithmetic and relational operators in a design from the Module Generation Library provided for the Atmel FPGA. Those 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 and function. In the MGI dialog box, users can change layout related parameters like area/speed optimization, or layout folding, and create a hard layout for the ModGen components.

The basic design flow for synthesis is to first define the design via the Figaro *File>Design Setup* dialog box. This will execute the appropriate design directory set up. Next, the synthesis tool is run using the *Synthesis* button in the Flowbar. It can be used to create various hierarchical levels of the design for mapping to the Atmel architecture.

Although the ModGen Library provides support for the arithmetic and relational operators in the design, the *Macro Generators* should be used to replace other regular structures to provide the best possible performance for the FPGA. The structural vhdl/Verilog files that are created at the time of the macro generation simplify this procedure. Because automatic I/O insertion is available, it is unnecessary to bring the results into a schematic capture tool for final assembly.

Refer to the *CAE Interfaces* section in the *AT40K IDS Tutorial* for design entry with the Exemplar Logic synthesis tools using the "averager" example. This will describe in detail the design and library setups, hierarchical synthesis, creation and use of the *Macro Generators* components, MGI flow, and final preparation of the design for placement and routing.

XNF Entry

Figaro supports designs in the Xilinx XNF format by use of a technology mapper. It allows the user to translate a Xilinx 3000, 4000, 4000E and 5200 family design into an AT40k device.

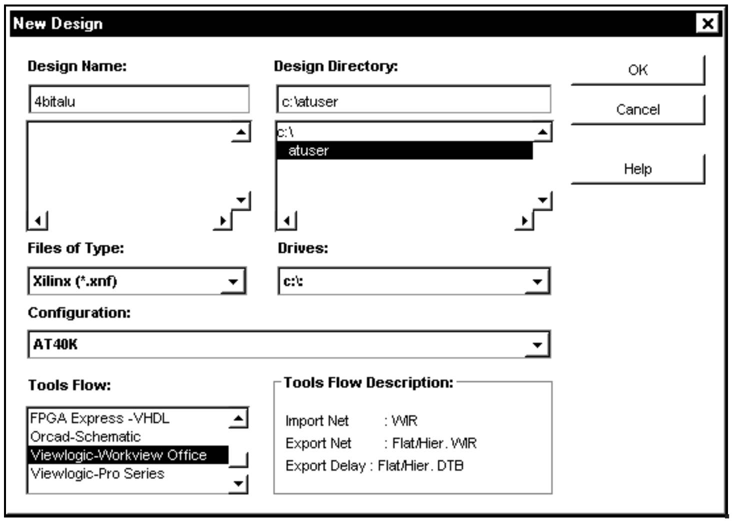
The Xilinx XNF design flow is integrated as part of the synthesis flow and is available for both the PC and Workstation. Viewlogic WIR files, VHDL or Verilog back annotation files can be exported for *Post-layout Simulation* as part of the process.

Most of the Xilinx XBLOX components can be mapped to the Atmel *Macros.* Once a design has been imported, it can be used for placement and routing. A schematic must be first translated into an XNF file with Xilinx software before it can be used by IDS.

XNF import is integrated into the Viewlogic design flow. Basic information on the user interface and design set up is briefly described below. An explanation of each XBLOX cell and library component available is provided in the *Technical Reference & Release Notes.* The manual also provides information on the available parameters and pin map descriptions.

Design Flow

To initiate XNF import, first define the design via the Figaro *File>Design Setup* dialog box. The user must specify the Viewlogic environment, AT40k configuration and XNF import format as shown below. It is recommended that a user library be set up before opening the design as the *Macro Generators* will be called automatically, when applicable, to map the XBLOX cells.


New Design

Design Name: 4bitalu

Design Directory: c:\atuser

c:\
atuser

Files of Type: Xilinx (*.xnf)

Drives: c:\

Configuration: AT40K

Tools Flow:
FPGA Express -VHDL
Orcad-Schematic
Viewlogic-Workview Office
Viewlogic-Pro Series

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

Design Setup Dialog Box

The import process can be started with either the *Open* button or *File>Open as Design* from the menu. The option *Open as Macro* cannot be used for the XNF flow. During XNF import, the *Macro Generators* dialog box will show up as needed. Create the macros by pressing the Generate button. Follow the standard Figaro flow to perform placement and routing.

Although Figaro does not export XNF netlists, the output file can be in Viewlogic, VHDL or Verilog format based on the simulator used for *Post-layout Simulation*.

Netlist Generation

Netlist generation is available via the Figaro Flowbar. This button will invoke the needed processes to generate a netlist that can be read into the system to perform placement and routing. This step needs to be performed before a design is opened.

To ensure the netlist is up-do-date before proceeding to the layout phase of the design, click on the *Create Netlist* button in the Flowbar.

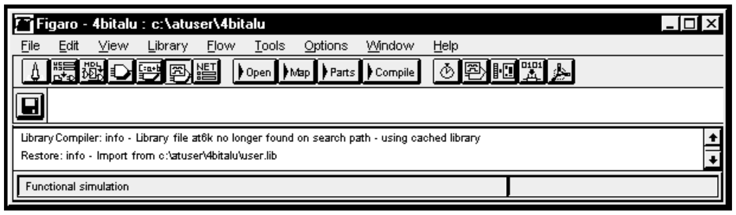
The netlist format varies with the CAE system specified. Please refer to the appropriate "*CAE Interfaces*" section in the *AT40K IDS Tutorial* manual or *Technical Reference & Release Notes* for details. Currently the system takes as input Viewlogic wir files or EDIF netlists from Synopsys, Everest, and Exemplar Logic. Automatic generation of these files is supported for Viewlogic. Synopsys, Everest and Exemplar Logic users must use the appropriate functions in these tools to prepare the netlist for Figaro.

Simulation

The Integrated Development System offers the user both *Functional Simulation* and *Post-layout Simulation*. Functional or pre-layout simulation helps the user ensure circuit validity. This allows the user to identify potential functional problems and rectify them in the circuit before placement and routing.

The *Post-layout Simulation* module performs the same analysis as *Functional Simulation*, except the results reflect the final physical design complete with timing information.

Figaro is designed to interface with simulation tools from CAE systems such as Viewlogic, Model Technology, and Cadence. These platforms can be used to perform *Functional* or *Post-layout Simulation* on a design entered through any of the methods discussed in the “Design Entry” chapter of this *User’s Guide*.



```
Library Compiler: info - Library file at6k no longer found on search path - using cached library
Restore: info - Import from c:\atuser\4bitalu\user.lib
```

Functional simulation

Functional Simulation

Functional Simulation

Functional Simulation provides functional verification of the circuit's characteristics. The simulation predicts real-world behavior by accounting for physical circuit nodes.

Figaro will prepare all configuration files and netlists as well as run all processes needed to invoke the simulator for the specified CAE system. However the user must provide the correct stimulus file or inputs to the simulator to verify circuit functionality.

The AT40k library of components can be divided into 2 types of macros: functional and dynamic. Functional macros are components with fixed functions, such as the 2 input AND gate. Dynamic macros are designed to allow user specification of any desired function, to be attached as an attribute via an equation string, on the symbol. This should be used only when a specific function for an AT40k core cell is required. Designs targeted to AT40k can use a mix of dynamic and functional macros.

Simulating With Dynamic Macros

Designs which use Dynamic Macros cannot be simulated directly from schematics or synthesis. This is because the various simulators do not have models for the Look Up Tables (LUT) which these macros emulate. Therefore any design which contains these components must first be read into Figaro, mapped (as needed), and brought through to initial placement before the appropriate netlist can be generated for simulation.

Figaro Interface

Figaro provides a totally integrated environment so the user can access the simulation tool of the specified CAE system directly from the Desktop. Simulation is supported by the Atmel FPGA library. The appropriate library is automatically set up according to user specification from the Desktop.

To start, press the *Functional Simulation* button on the Flowbar. Figaro will proceed to prepare all inputs and invoke the simulator. The user should refer to the documentation for the CAE system on how to simulate the design before proceeding. For designs containing Dynamic Macros, use *Tools>Post Mapping Simulation* to verify the functionality of the design. *Post Mapping Simulation* can only be performed after initial placement, as explained in the section "Simulating with Dynamic Macros" above.

The next section will describe the processes in general for the respective CAE systems. Refer to the *AT40K IDS Tutorial* and *Technical Reference* manuals for more details.

Viewlogic Simulators

The Viewlogic simulators for the PC and Workstation are fully supported by IDS. The underlying functionalities are similar. In either case, *Functional Simulation* executes two programs, check and vsm, before running PROsim or ViewSim/Fusion. The check program ensures that all wire files are current, and vsm creates input files for the simulation.

The Atmel libraries for the Viewlogic simulators are created using parameterized attributes in order to support the speed bins and the speed ranges. Before vsm can be run successfully, the attributes file must be copied to the current design directory. This is handled by IDS internally. The file, located in the Atmel lib directory, is named spbin40k.var. It is copied to the design directory and renamed *designName*.var.

Model Technology V-System

The Model Technology V-System/VHDL simulator is a VITAL compliant VHDL based simulator. It is integrated into IDS as part of the Exemplar Tools Flow.

Input for *Functional/Post-Mapping Simulation* is a VHDL netlist file generated by the user or as a result of synthesis on the original HDL design.

For VHDL design simulations, VITAL 95 compliant Atmel FPGA libraries are provided along with IDS. The modelsim.ini file has to be modified accordingly to point to the Atmel library. The library is called AT40K and is installed in the atmel/lib/mti directory.

The design VHDL files should first be compiled and then simulated with the command prompt in the *Shell Window*. The V-System simulator cannot be invoked from the push buttons on the IDS desktop.

Cadence Verilog-XL

Verilog-XL is the simulator supported by IDS for the Cadence CAE system. However Atmel Verilog libraries can also be used with other Verilog simulators as well.

Input for *Functional/Post-Mapping Simulation* is a Verilog netlist file generated by the user or as a result of synthesis on the original HDL design..

For Verilog design simulations, Verilog libraries for Atmel FPGA components are provided along with IDS. The library is called AT40K and is installed in the atmel/lib/verilog directory. The file prim.v contains the simulation models for the user-defined Verilog primitives in the Atmel FPGA Verilog library. This file can be specified in the command-line using the -v option.

The Verilog-XL simulator cannot be invoked from the push buttons on the IDS desktop. The design's Verilog files should be simulated with the command prompt in the *Shell Window*.

The command-line syntax should include all user and Atmel libraries, the *min/typ/max Speed Range* specification, and the design's Verilog files.

Post-mapping Simulation

Designs which use Dynamic Macros cannot be simulated directly from schematics or synthesis. This is because the various simulators do not have models for the Look Up Tables (LUT) which these macros emulate. Therefore any design which contains these components must first be read into Figaro, mapped (as needed), and brought through to initial placement before the appropriate netlist can be generated for simulation.

Figaro Interface

Figaro provides a totally integrated environment so the user can access the simulation tool of the specified CAE system directly from the Desktop. Simulation is supported by the Atmel FPGA library. The appropriate library is automatically set up according to user specification from the Desktop.

Figaro will proceed to prepare all inputs and invoke the simulator. The user should refer to the documentation for the CAE system on how to simulate the design before proceeding. To start post-mapping simulation, use *Tools>Post Mapping Simulation* to verify the functionality of the design. *Post Mapping Simulation* can only be performed after initial placement.

Post-layout Simulation

Post-layout Simulation performs post-layout timing and functional verification to provide an accurate estimate of the circuit's input to output timing characteristics. It notes the post-layout wire delays, including pin-to-pin delays, setup and hold times, and actual wire delays, to predict device timing and performance.

Figaro will prepare all necessary configuration files, netlists and delay files, as well as run all processes required to invoke the simulator for the specified CAE system. The user will need to provide the correct stimulus file or inputs to the simulator to verify the circuit functionality.

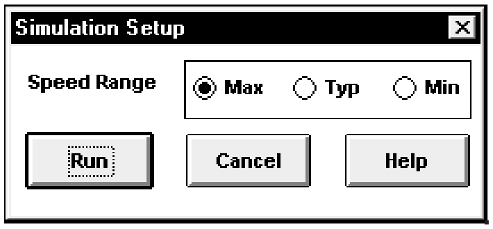
A new simulation netlist is needed whenever a design has gone through placement and routing. This is required to support the design transformations resulting from the compilation process. The netlist is a flattened representation of the design as all hierarchy will have been removed and buses mapped to their scalar components. Because the various components of the design have been mapped into the LUT architecture, the internal signals are often changed. As the internal nodes will be typically unavailable, all stimulus for the circuit should be at the I/O level for correct simulation of the design.

In addition to the simulation netlist, a back annotation file containing information on macro delays intrinsic to the components, as well as routing delays, is generated for *Post-layout Simulation*. This file will be output in a format that matches the flattened netlist created by Figaro as discussed previously.

Figaro Interface

Figaro provides a totally integrated environment so the user can access the simulation tool of the specified CAE system directly from the Desktop. Simulation is supported by the Atmel FPGA library. The appropriate library is automatically set up according to the part chosen..

To start, press the *Post-layout Simulation* button on the Flowbar. The *Post-layout Simulation* dialog box will appear, and the user is asked to set the *Speed Range* as shown below. Select the desired option and initiate the process.



Post-layout Simulation Dialog Box

The *Speed Range* indicates whether the delay values to simulate are *Minimum*, *Typical*, or *Maximum* case scenarios. A *Maximum Speed Range* represents the worst case analysis. The program will default to *Maximum*.

Once the above values have been specified, press the Run button to prepare all inputs and invoke the simulator. The user should refer to the documentation for the CAE system on how to simulate the design before proceeding.

The next section will describe the processes in general for the respective CAE systems. Refer to the *AT40K IDS Tutorial* and *Technical Reference & Release Notes* for more details.

Viewlogic Simulators

For *Post-layout Simulation*, Figaro will generate the back annotation files and needed netlists before running vsm. The same stimulus files used for *Functional Simulation* can be used to test the circuit, provided that no internal nodes are being accessed. The simulators will be invoked as discussed in *Functional Simulation* with the following differences.

A new set of wir files will be created by Figaro to support transformations that result from the mapping of the original circuit to the AT40K architecture. The netlists will be flat as discussed in the *Post-layout* introduction. These new files are found in the figba sub-directory for the design. The system will automatically modify the viewdraw.ini file to point to the correct set of files when *Post-layout Simulation* is invoked.

Along with the above files, a *designName*.dtb file (or separate *.dtb files for the partitioned designs) is created in the figba directory which contains all of the routing delays between the macros in the design. Again, this information is created and accessed during *Post-layout Simulation*.

Model Technology V-System

The Model Technology V-System/VHDL simulator is a VHDL based simulator. It is integrated into IDS as part of the Exemplar Tools Flow .

For *Post-layout Simulation*, both a back annotation file (or separate *.sdf files for the partitioned designs) in the Standard Delay File (SDF) format and a new VHDL netlist are created. These files are output to the design/figba directory. The simulator is then invoked in a similar fashion as in *Functional Simulation.*

The SDF file, which contains routing delay information as well as the intrinsic delays for the macros referenced in the design, will match the new flattened netlist.

NOTE

When a design is flattened, all buses used will be split out into their individual components. The stimulus files should be changed appropriately to take this into consideration before the simulator is invoked by the user.

Cadence Verilog-XL

For *Post-layout Simulation*, both a back annotation (or separate *.dtb files for the partitioned designs) in the Standard Delay File (SDF) format and a new Verilog netlist (if required as outlined in the *Post-layout* introduction) are created. These files are output to the design/figba directory. The simulator is then invoked in a similar fashion as in *Functional Simulation.*

The SDF file contains routing delay information as well as the intrinsic delays for the macros referenced in the design. The SDF file will either match the original input hierarchy or the new flattened netlist as needed.

NOTE When a design is flattened, all buses used will be split out into their individual components. The stimulus files should be changed appropriately to take this into consideration before the simulator is invoked.

Other Simulator Flows

The Flowbar buttons are provided to be used with various supported simulator tools and flows. Simulation with other platforms is still supported but not in the automatic fashion as for the above systems. After the design has been placed and routed (with the EDIF input that is generated by the tools), the user can create the needed back annotation files via the *File>Export* dialog box. This dialog box allows the specification of many different back annotation formats such as SDF, DBA, and DTB for flattened designs. These files can then be read into the simulator as needed.

Design Implementation

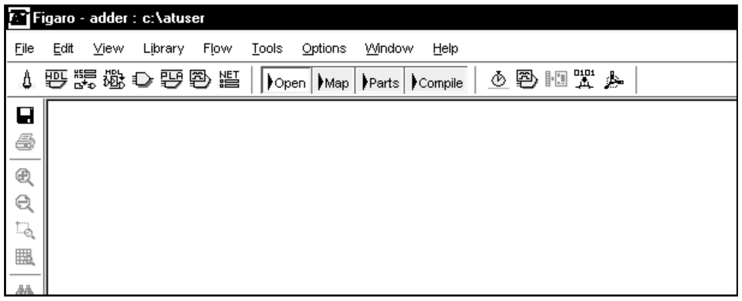
This section serves as an overview to Figaro as it pertains to the design implementation functions of *Open*, *Map*, *Parts*, and *Compile*. Step by step instructions on how to execute a design are discussed in the *Figaro* section of the *AT40k IDS Tutorial*.

The preceding modules in the Flowbar allows the user to perform the preliminary functions of design entry and verification through to netlist generation. Once the netlist is created, Figaro can generate a data base for automatic and/or manual placement and routing.

Open

The *Open* module is used to either create a data base for the design by opening a netlist, or load in a previously saved data base. Figaro takes the netlist created from the *Netlist* button and converts it into a data base file for use as a *Design* or *Macro*. This directs Figaro to either prepare the design for eventual bitstream output, or leave it as a macro for library check-in.

The other application for this module is to provide access to a previously created design. The user can then resume work on manual placement, routing, or other tasks needed for implementation of the layout.



The Open Button

Map

Selection of the *Map* button will cause IDS to search for an open design before proceeding. If none is found, Figaro will bring up the *Open as Design* dialog box. Once a design has been loaded, the optional mapping step can proceed.

Mapping is the process of optimizing design logic and adapting it to the Atmel architecture. It can be used to achieve area optimization for a netlist created with the AT40k library. Although it is an optional step, mapping should be run to achieve the best possible circuit speed and density.

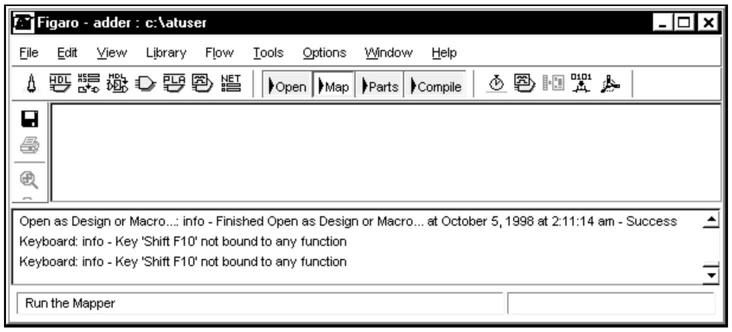
Mapping takes the instances from the design netlist and:

a) Converts the instances to a technology-independent form.

b) Performs area optimization to reduce the space utilization. It will try to pack the input design into the core cells.

c) Generates instances specific to the selected device.

The *Design Browser* and the *Map Browser* can be used to view the results of this process.



```
Open as Design or Macro...: info - Finished Open as Design or Macro... at October 5, 1998 at 2:11:14 am - Success
Keyboard: info - Key 'Shift F10' not bound to any function
Keyboard: info - Key 'Shift F10' not bound to any function
```

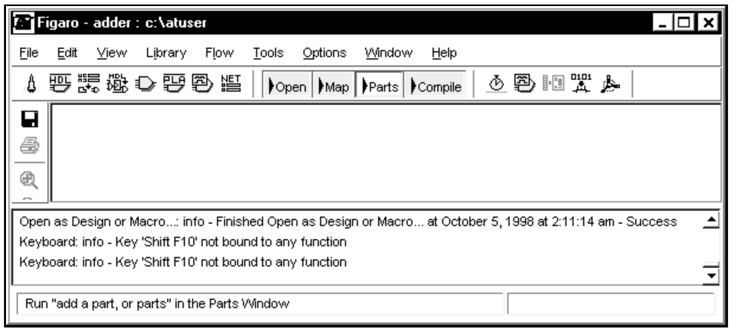
Run the Mapper

The Mapper Button

Selecting Parts

Selection of the *Parts* button will cause IDS to search for an open design before proceeding. If none is found, Figaro will bring up the *Open as Design* dialog box. If mapping has been enabled, the design will first be mapped before *Part Selection* is invoked. Once a design is available, the program will display the *Parts Assembler* window and *Part Select* dialog box. The *Parts Assembler* serves three important functions by: a) allowing the user to select the part or parts needed for the design, b) providing the user with the means to pre-place I/Os by assigning pin locks for the design, and c) partitioning the design.

The user can specify the Atmel part(s) and package(s) as well as select the speed grade. The suitability of a certain part for ease of compiling is presented by the visual display on the screen and in the *Part Selection* dialog box. The amount of logic and RAM resources needed are represented by the left and right bar graphs respectively in *Parts Assembler* window. A high demand on the chip resources will increase the need for manual pre-placement and routing to complete the design.



```
Open as Design or Macro...: info - Finished Open as Design or Macro... at October 5, 1998 at 2:11:14 am - Success
Keyboard: info - Key 'Shift F10' not bound to any function
Keyboard: info - Key 'Shift F10' not bound to any function
```

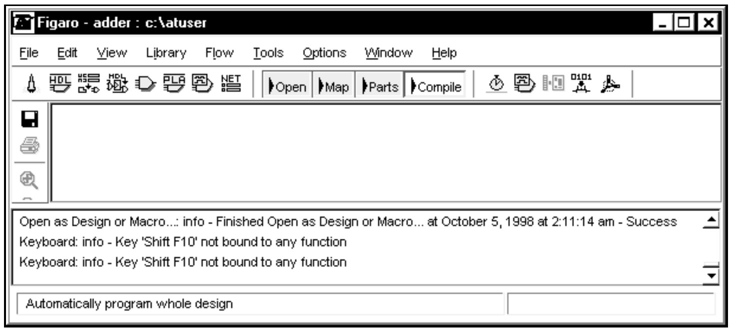
Run "add a part, or parts" in the Parts Window

The Add Parts Button

Compiling

The *Compile* function directs Figaro to place, route, and generate a bitstream (or check-in a macro) automatically. Figaro will automatically call up the *Open*, *Map*, and *Parts* modules in sequence if only the *Compile* button is selected for design implementation.

Depending on the specification of the circuit as *Macro* or *Design*, the result will either be a bitstream for downloading, or a component for the macro library.



```
Open as Design or Macro...: info - Finished Open as Design or Macro... at October 5, 1998 at 2:11:14 am - Success
Keyboard: info - Key 'Shift F10' not bound to any function
Keyboard: info - Key 'Shift F10' not bound to any function
```

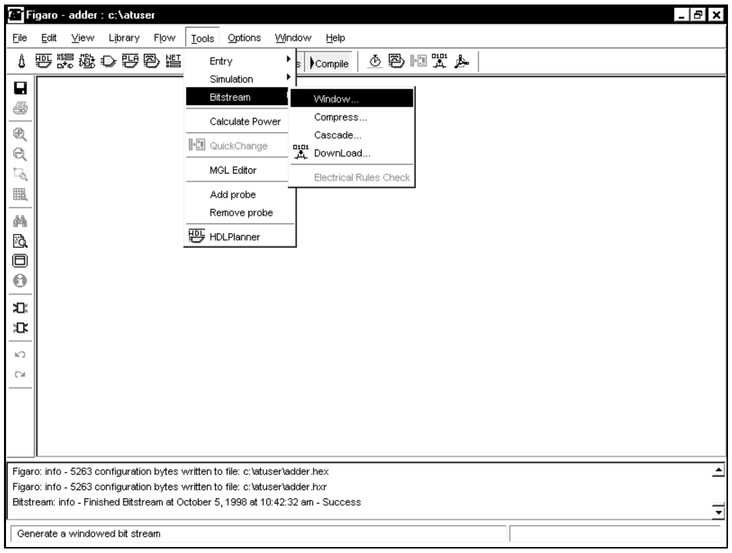
Automatically program whole design

The Compile Button

Device Programming

Following placement and routing, the design is ready for the user to generate the bitstream file for programming the Atmel FPGA device. The user can program and download the design as needed. There are two parts to Device Programming. The first is the actual creation of the bitstream file. This is done as part of the *Compile for Design* process. The second part involves utilities which can manipulate and or check the bitstream.

Device Programming brings the design from building the bitstream through downloading it to the FPGA or Serial Configuration Memory device on the Prototype Board. Other utilities included allows the user to reduce the output data into a smaller format (*Compress Bitstream*), change the design from one base design to other optional designs (*Window Bitstream*), combine several designs into one bitstream (*Cascade Bitstream*), and download to the FPGA or Serial Configuration Memory (*DownLoad Bitstream*).



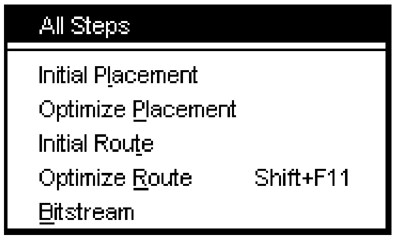
```
Figaro: info - 5263 configuration bytes written to file: c:\atuser\adder.hex
Figaro: info - 5263 configuration bytes written to file: c:\atuser\adder.hxr
Bitstream: info - Finished Bitstream at October 5, 1998 at 10:42:32 am - Success
```

Generate a windowed bit stream

IDS Bitstream Utilities

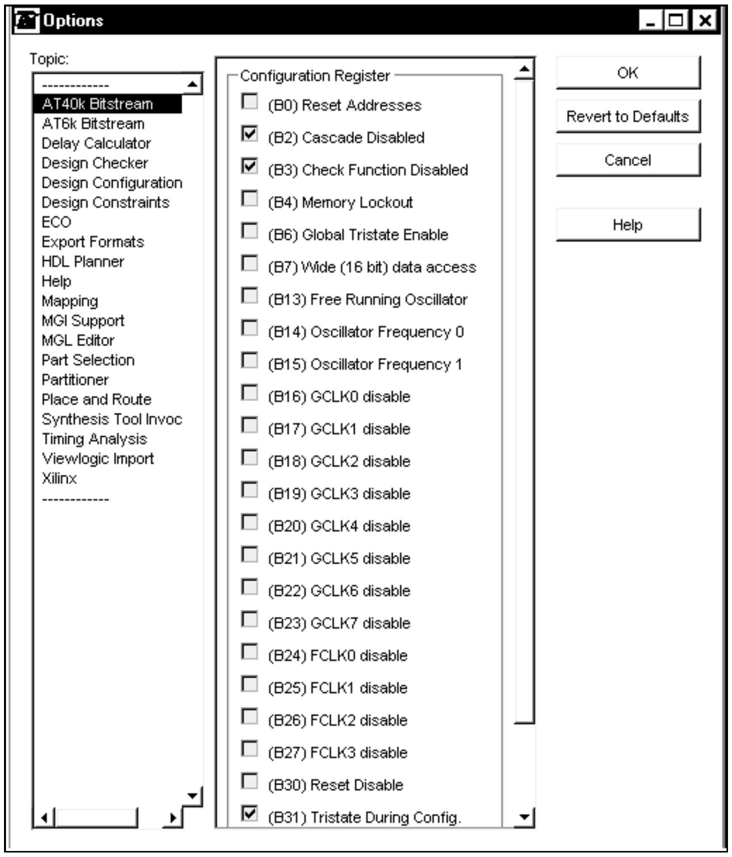
Build Bitstream

Build Bitstream is the final stage of the *Compile for Design* process, where the physical data base is converted into the bitstream file. *Build Bitstream* is invoked after all layout applications are completed. It can be invoked via the *Flow>Compile>Bitstream* menu option or the *BitStr* button on the Device Flowbar.



Compile Menu

The various options for controlling bitstream generation are available under the *Options>Options AT40k Bitstream* dialog box as shown below.



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 Invoc
Timing Analysis
Viewlogic Import
Xilinx

Configuration Register

- ☐ (B0) Reset Addresses
- ☑ (B2) Cascade Disabled
- ☑ (B3) Check Function Disabled
- ☐ (B4) Memory Lockout
- ☐ (B6) Global Tristate Enable
- ☐ (B7) Wide (16 bit) data access
- ☐ (B13) Free Running Oscillator
- ☐ (B14) Oscillator Frequency 0
- ☐ (B15) Oscillator Frequency 1
- ☐ (B16) GCLK0 disable
- ☐ (B17) GCLK1 disable
- ☐ (B18) GCLK2 disable
- ☐ (B19) GCLK3 disable
- ☐ (B20) GCLK4 disable
- ☐ (B21) GCLK5 disable
- ☐ (B22) GCLK6 disable
- ☐ (B23) GCLK7 disable
- ☐ (B24) FCLK0 disable
- ☐ (B25) FCLK1 disable
- ☐ (B26) FCLK2 disable
- ☐ (B27) FCLK3 disable
- ☐ (B30) Reset Disable
- ☑ (B31) Tristate During Config.

Global Options Dialog Box

In order to store the correct setup configuration features and options in the bitstream file, as well as generate the bitstream programming file, the user must determine how configuration data will be loaded into the SRAM of the FPGA. There are thirty-two configuration control register bits for this purpose. Control register option values are drawn from the *AT40k Bitstream Options*. Consult the *AT40k Datasheet* for more information on control register loading.

In addition to the Configuration Register bits, there is a *RAM Initialization* checkbox in the *AT40k Bitstream Options*. If this box is checked, the user-configurable RAM units on the FPGA will be cleared during device programming. If it is un-checked, the RAM contents will retain their previous contents.

Creating a Bitstream

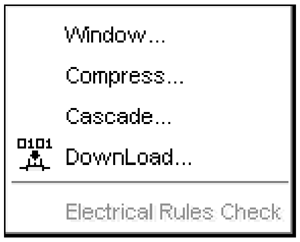
Bitstream creation is the last step in design compilation. Before pressing the *Compile* button on the main Flowbar or the *BitStr* button on the Device Window Flowbar, verify that the options have been set appropriately as discussed in the previous section. A bitstream can only be created after a design has been completely placed and routed. If *Open as Macro* was used to create the lay-out, the bitstream option will be unavailable.

The process of creating a bitstream is one of translating the Figaro based design into an Atmel specific bitstream. Results of the bit-stream process are displayed in the transcript window.

The bitstream creation process generates three files; *DesignName*.bst, *DesignName*.hex, and *DesignName*.hxr. *DesignName*.bst is an Atmel bitstream format file, *DesignName*.hex is an MCS-86 format file, and *DesignName*.hxr is an MCS-86 format file designed for use with third-party programmers for serial configuration memory devices.

Bitstream Utilities

Bitstream Utilities provide the ability to compress, window, cascade, and download files that have been created with *Bitstreaming* from the *Compile* module. An FPGA device can have its changed behavior programmed and re-programmed with data windows in *Window Bitstream*. *Compress Bitstream* compresses the bitstream into windows, programming only areas of the chip which are being used. *Cascade Bitstream* allows the user to generate a bitstream to program multiple cascaded devices. The download process is specifically set up to support the transfer of the bitstream to the Atmel prototype or serial configuration memory boards.

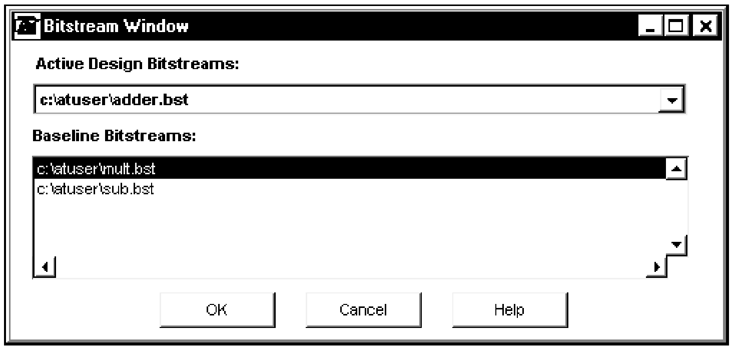


Bitstream Utilities

NOTE The bitstream utilities are not available until at least one bitstream has been created for any of the designs specified in the *File>Design Setup* dialog box.

Window Bitstream

Window Bitstream is used to program or re-program a portion of the FPGA device. This tool is an important step in the support of cache logic as it can determine which parts of the design have changed, and create a bitstream to configure only that part of the chip. A maximum of 255 windows can be produced.



Window Bitstream Dialog Box

The user prepares two similar designs and creates separate bitstream files. The program compares the bitstreams and the resulting differences are placed into a new bitstream file. This new file contains only the information required to transform the baseline design into the new design. The data in this file is in a windowed format, and it will only program the changed portions of the design.

Starting a Session

Pull down the menu option *Tools>Bitstream>Window* to bring up the dialog box. As seen in the above dialog box there is a bitstream for the active or current design. The user must pick among the baseline bitstreams for the one which will be compared against the active design. The operation will result in a new bitstream for the active design. This contains the windows which will transform the selected baseline bitstream into the current design.

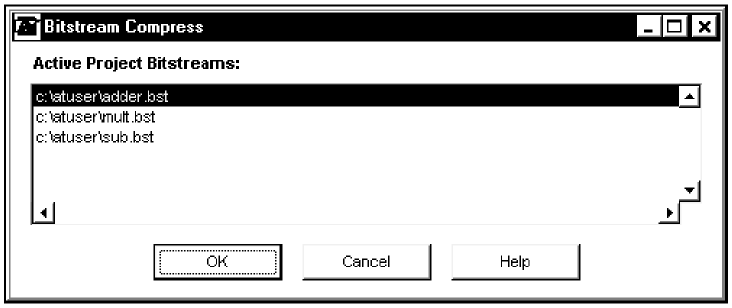
Select OK to initiate *Window*. The process runs to completion and ends without user intervention.

Further details on the Windowing process can be found by selecting the Help button.

Compress Bitstream

The *Compress* program allows the user to compress the bitstream output for faster loading, and to fit into a smaller memory device as needed. For larger devices of the Atmel FPGAs, it is recommended that the bitstream be compressed.

NOTE Use compressed bitstreams to program a device only immediately following power-up, or configuration reset, or when programming two completely independent functions that are to work simultaneously.



Active Project Bitstreams:

c:\atuser\adder.bst
c:\atuser\mult.bst
c:\atuser\sub.bst

Compress Bitstream Dialog Box

Starting a Session

Pull down the menu option *Tools>Bitstream>Compress*. Select the bitstream to compress and click on OK to initiate the program. The process runs to completion and ends without user intervention. The output of this program will be a new bitstream, with the same file name, that is smaller than the original.

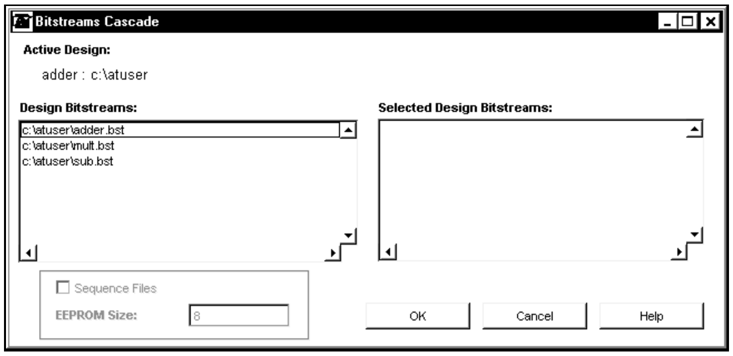
Input to *Compress* must be in BST format but output will be in HEX, HXR and BST format. The HXR format is for use with third party serial configuration memory programmers.

Select OK to initiate *Compress*. The process runs to completion and ends without user intervention.

Further details on the *Compress* process can be found by selecting the Help button.

Cascade Bitstream

The *Cascade* module allows the user to concatenate several bit-stream files into a single file with the appropriate control register settings. This single file can then be downloaded to a master FPGA which will subsequently load other devices in the system.



adder : c:\atuser

c:\atuser\adder.bst
c:\atuser\mult.bst
c:\atuser\sub.bst

EEPROM Size: 8

Cascade Bitstream Dialog Box

Starting a Session

Pull down the menu option *Tools>Bitstream>Cascade*. The *Design Bitstreams* list box displays all design directories in the system that contain bitstream files. To create a list of bitstream files for cascading, click on bitstream file names from *Design Bitstreams* to place them under the *Selected Design Bitstreams* list box. Click on a file in the *Selected Design Bitstreams* box to remove it from the box. Cascading of the bitstream files will be executed in the order shown.

Select OK to initiate *Cascade*. The process runs to completion and ends without user intervention.

NOTE

When creating entries for *Cascade*, the total file number is not allowed to exceed 8, to avoid potential problems when programming the FPGA parts.

Input to *Cascade* must be in BST format but output will be in both HEX and BST format.

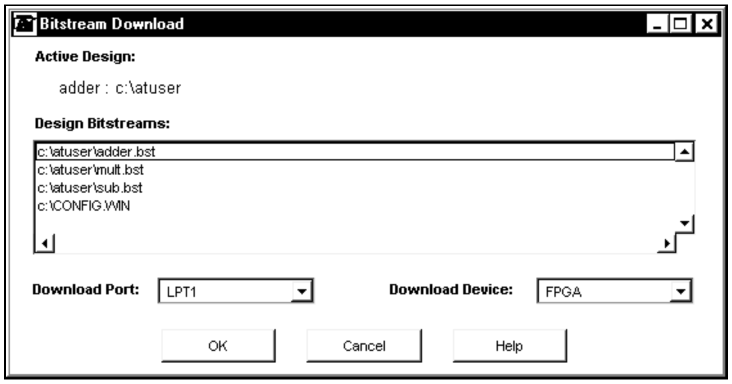
The *Sequence Files* and *EPROM Size* options are not available for AT40k series FPGAs.

Further details on the *Cascade* process can be found by selecting the Help button

.

DownLoad Bitstream

The *DownLoad* option is the last step in designing and laying out a programmable device. This function is available for use when doing design prototyping. The user can download the data through the specified parallel port on a PC to an Atmel prototype board, Atmel serial configuration memory board, or to any download board supplied by the user.



Active Design:

adder : c:\atuser

Design Bitstreams:

c:\atuser\adder.bst
c:\atuser\mult.bst
c:\atuser\sub.bst
c:\CONFIG.WIN

Download Port: LPT1

Download Device: FPGA

DownLoad Bitstream Dialog Box

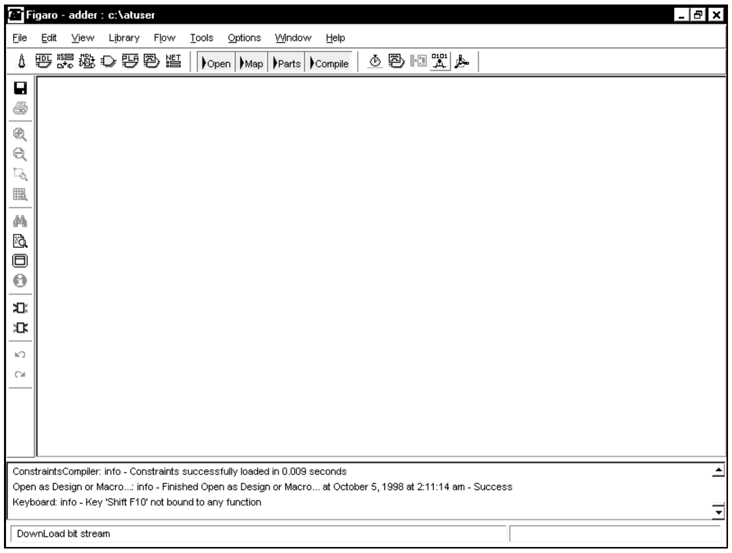
The *DownLoad* process requires a special download cable that is part of the Download Assembly shipped with the Atmel Integrated Development System. To ensure successful execution of the download process, the user should check all hardware connections.

The *DownLoad* button on the Figaro Flowbar is only active on the PC platform. For workstation users, the download program (downld40.exe) can be found in the atmel/bin directory. This program, along with the bitstream file, must be transferred to a PC for downloading. Refer to the Command Reference section of the *Technical Reference & Release Notes* for further details on running the program from the command-line.

Another option is to download the bitstream to Atmel's Serial Configuration Memory. This assumes that the memory prototype board is connected to the PC. The program used for this purpose is cf.exe which is in the atmel/bin directory. Both the program and the bitstream file must be transferred to a PC for downloading. Refer to the Command Reference section of the *Technical Reference & Release Notes* for further details on running the program from the command-line.

Starting a Session

Pull down the menu option *Tools>Bitstream>DownLoad* or select the *DownLoad* button from the Figaro Flowbar.


Figaro - adder : c:\atuser
File Edit View Library Flow Tools Options Window Help
Open
Map
Parts
Compile

```
ConstraintsCompiler: info - Constraints successfully loaded in 0.009 seconds
Open as Design or Macro...: info - Finished Open as Design or Macro... at October 5, 1998 at 2:11:14 am - Success
Keyboard: info - Key 'Shift F10' not bound to any function
```

DownLoad bit stream

DownLoad in the Flowbar

Specify the appropriate parallel port and then select OK to initiate *DownLoad*. The process runs to completion and ends without user intervention.

Stand-alone Session

The stand-alone Downld40 program can be ported to multiple computers. With this feature, the user can carry out the circuit testing process at any site.

In executing Downld40 from the DOS command-line, a typical run is similar to the following example:

```
c:\> Downld40 /p lpt2 DesignName.bst
```

A comparable means to run the cf program for the Serial Configuration Memory is available. In executing cf from the DOS command-line, a typical run is similar to the following example:

```
c:\> cf /p /i /g DesignName.bst
```

MGL

The Figaro software is shipped with a range of *Macro Generators* for the AT40k series FPGAs. (For more information, please refer to the "*Design Entry, Macro Generators"* section in this *User's Guide*). The *Generators* are written in a new language called *MGL (Macro Generator Language)*. The language is specially developed by Atmel to allow the programmatic creation of user macros. Designers can write their own *Macro Generator* programs using this language to produce parameterized macros with hard layout and routing. A compiler, editor, and debugging tools for MGL are included to facilitate this process.

This section provides a brief overview of MGL and its associated tools. For a more thorough description of the language, refer to the MGL section of the *Technical Reference & Release Notes*.

Language Overview

MGL is a highly specialized language, whose principal purpose is the creation of hard user macros. The various language constructs have been optimized specifically for this task, and superfluous features have been avoided as much as possible. The language does, however, support a range of features that are common to most high-level design languages (HDLs) as listed below.

- "If .. then .. else .." and "case" statements for conditional branching
- "For" and "while" loops
- User-defined functions
- Printing and error handling
- Arithmetic operators and built-in arithmetic functions
- File input and output
- Common data types such as integer, float, boolean, string and array

MGL differs from other HDLs in its ability to describe the physical properties of a macro, including its placement and routing. Macro creation is performed in two distinct stages:

- Defining the macro interface.
- Defining the contents of the macro in terms of component instantiations, as well as their logical and physical connectivities.

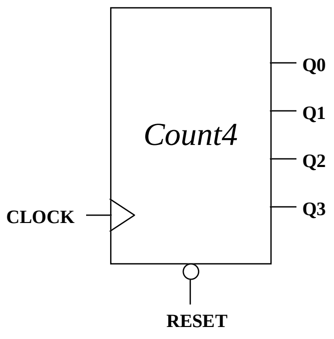
The macro interface is described using an *interface block* as illustrated below.

```
counter : macro;
...

interface “Count”{width} of counter is
   inputports( “CLOCK”, “RESET”);
   for i in 0 to (width-1) loop
      outputports( “Q”{i});
   end loop;
end interface;
```

Interface block

The figure above describes the interface of a variable-width counter macro, with CLOCK and RESET inputs, and an output bus Q[width-1:0]. A graphical representation of the interface produced is shown below.



Macro Symbol

The contents of a macro is described using a *contents block* as in the following example.

```
contents of counter is

   ...  statements ...

end contents;
```

Contents Block

The process of defining the contents of a macro (i.e. its underlying implementation) consists of three main tasks:

- Instantiating components
- Connecting components together via nets
- Specifying the physical routing resources used by those nets

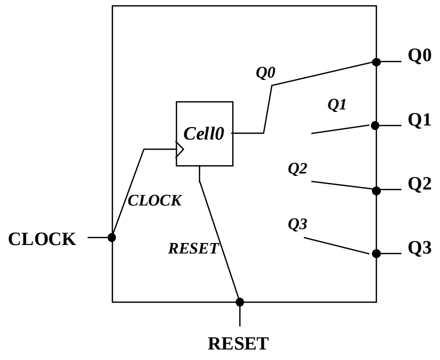
To instantiate a component within the macro contents, the first step is to specify a component from either a library, or another MGL program. An *instance block* is then used to describe the instantiation of that component as shown below.

```
// Get macro from vendor library
aMacro  : macro := getmacro( “FGEN1RF”);


instance “Cell0” of aMacro is
   location( 0, 0);
   functiong( “!FB”);
   connections( “CLK”->“CLOCK”,
                “RS”->“RESET”,
                “G”->“Q0”);
end instance;
```

Instance Block

The instance block creates an instance of an FGEN1RF macro (a generic cell consisting of a 4-input LUT, a register and a feedback connection) called “Cell0”. It places the cell at location (0,0). Next the cell is programmed to negate the feedback connection and feed it into the register. Then the ports on the cell are connected to the appropriate nets within the macro. The schematic of the instance block is shown below.



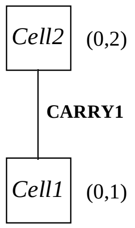
Instance Block Schematic

The routing within a macro is defined using a *route block*, such as the one shown below.

```
route of “CARRY1” is
  nodes( (0, 1, “yOut”),
         (0, 2, “yIn) );
end route;
```

Route Block

The route taken by the net “CARRY1” is described by attaching route nodes to the net. The net is connected from the Y (orthogonal) output of the core cell at location (0,1) to the Y input of the core cell at location (0,2).

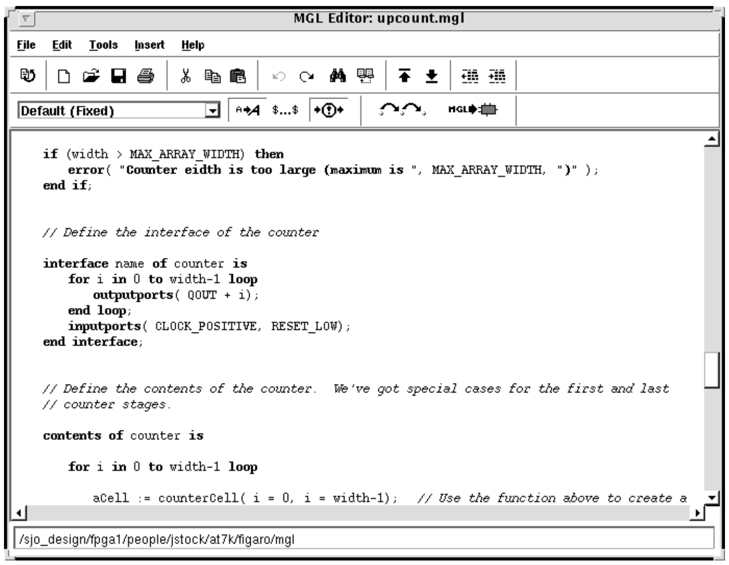


Graphical View of Route Block

The above is a brief outline of how to create a user-customized macro using MGL. Additional MGL features that support the programmatic generation of entire designs (including I/Os) are covered in the *Technical Reference & Release Notes*.

MGL Editing and Debugging tools

An MGL editing environment can be reached from the *Tools* menu in Figaro, under *MGL Editor*. Figaro must be set up with an AT40k design before the *MGL Editor* can be invoked. The editor allows MGL files to be edited, debugged and compiled.



```
if (width > MAX_ARRAY_WIDTH) then
   error( "Counter eidth is too large (maximum is ", MAX_ARRAY_WIDTH, ")" );
end if;


// Define the interface of the counter

interface name of counter is
   for i in 0 to width-1 loop
      outputports( QOUT + i);
   end loop;
   inputports( CLOCK_POSITIVE, RESET_LOW);
end interface;


// Define the contents of the counter.  We've got special cases for the first and last
// counter stages.

contents of counter is

   for i in 0 to width-1 loop

      aCell := counterCell( i = 0, i = width-1);   // Use the function above to create a
```

/sjo_design/fpga1/people/jstock/at7k/figaro/mgl

MGL Viewer

The following buttons are available on the MGL Editor toolbar:

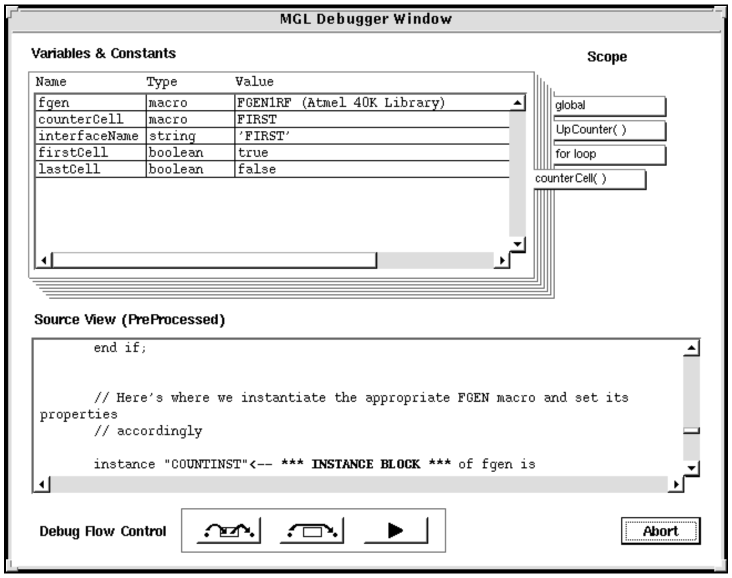
⟲	Updates the file being edited with the latest saved version.
🗋	Creates a new MGL file for editing.
📂	Opens a new MGL file.
💾	Saves the file being edited.
🖨	Prints the current MGL file.

✂	Cuts the selected text and places it in the clipboard.
⎘	Copies the selected text and places it in the clipboard.
📋	Pastes the contents of the clipboard.
↶	Multi-level Undo function.
↷	Multi-level Redo function.
🔍	Searches for a given string in the current file.
⇄	Searches for a given string and replaces it with another.
⤒	Places the cursor at the start of the file.
⤓	Places the cursor at the end of the file.
⇥	Increases the indent level of the selected text.
⇤	Decreases the indent level of the selected text.
Default (Fixed) ▾	Text size menu button - Allows the user to select the font size for displaying the MGL file.
A→A	Switches dynamic syntax highlighting on and off.
$...$	Comments out the selected text.
→!→	Globally switches all user breakpoints (set using the setbreak() function) on or off.
↷↷	Single-steps through the MGL code, bring up the MGL Debugger window after each command is executed.
MGL→▭	Compiles the current MGL file and, if successful, gives the user the option of running the generated macro through the UDM (User-Defined Macro) flow in IDS.

Most of these features can also be accessed through the menu bar. In addition, the *Edit* menu allows the user to *Go To* a selected line and toggle on and off the *Auto Indent* feature. The *Insert* menu provides several of the most common MGL constructs as templates, which can be inserted into the user’s code and then modified as needed.

The Debugger Window

The Debugger window is split into three main sections as shown below. The top allows the user to inspect the value of variables and constants at various scopes in the program. The middle displays the source code with the current execution point highlighted. The bottom provides controls for: a) stepping into the next statement,
b) over the next statement, or c) proceeding with normal code execution.



Name	Type	Value
fgen	macro	FGEN1RF (Atmel 40K Library)
counterCell	macro	FIRST
interfaceName	string	’FIRST’
firstCell	boolean	true
lastCell	boolean	false

```
        end if;


        // Here’s where we instantiate the appropriate FGEN macro and set its
properties
        // accordingly

        instance "COUNTINST"<-- *** INSTANCE BLOCK *** of fgen is
```

MGL Debugger Window