Integrated Development System
Technical Reference

June 2002

This Tutorial

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 with an Atmel field programmable gate array (FPGA). Design entry vendors supported include Viewlogic, Mentor, OrCAD, Synario, VeriBest, Cadence, Data 1/O and CUPL. Synthesis tools from Cadence, Everest Design Systems, Exemplar Logic, Mentor Graphics, Synario, Synopsys, VeriBest and Viewlogic have been integrated into the system to provide optimum results. Simulation support for Mentor Graphics, Model Technology, Viewlogic, and Synopsys have been added. Atmel's IDS works in conjunction with these tools to let designers create fast, efficient, and predictable designs with Atmel FPGAs.

The operation of IDS is controlled by a single graphical interface, the *Figaro Window*. Figaro integrates all programs and links them through a unified database. Figaro 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.

This *Technical Reference*, includes more details on the various parts of the Integrated Development System which may not be covered in the *User Guide* or on-line help.

The Command Reference and Program Messages cover the entire Integrated Development System and the tools supported. Details on the *Macro Generators*, DesignWare Library and IDS interface (including the *Macro Generators Interface)* to other CAE systems are also included. Information on the new QuickChange software design flow and Cache Logic is given as well. Additionally, guidelines for PLA, Verilog, and VHDL coding are provided. Finally a list of files that may be found in the design directory is given in the Appendix.

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"

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 the PSLI Hotline at 408-436-4119 between 9 am and 5 p.m., Pacific time.
2. Sending electronic mail to fpga@atmel.com.

Command Reference

The Command Reference describes the function of each program and the commands associated with its use. Programs are listed alphabetically. The *Macro Generator* programs are not currently listed here. Any other program run by IDS that is not contained here will be a CAE platform specific program and the appropriate documentation should be consulted. The following information is provided for each program:

- Short description
- Usage statement
- Explanation of program specific command line options.

Usage statements (shown below) give the program name, version, and any command line arguments. All programs in the IDS share common command line syntax and a number of standard arguments. Instead of describing these common arguments under every program, they are described here for reference.

```
NETOUT Version 6.00
Copyright (c) 19911998, Atmel Corp. All rights reserved
Usage: NETOUT [Design] [/Options]
Design    : DesignName or ProjectDir\DesignName
/d name   : Design configuration file name
/g        : Go using the default user settings
/i name   : Technology input file name
/o name   : Technology output file name
/u name   : User initialization file name
```

Syntax

PROGRAM [Design] [/Options] [Keywords]
[Other Arguments]

The command line syntax for IDS software is composed of the program name followed by various command line arguments. These arguments are divided into a few categories: options, keywords, and other arguments.

Options are single character arguments preceded by a '/'.Some options require no arguments, others can be followed by one or more, such as /i *filename*.

Keywords are words that the program recognizes as a command to do something. For example, the keyword "Initial" makes the detail router perform initial routing.

Other arguments are variables that can change each time a program is run. For example, *Design* name is a variable that all programs recognize.

Design

The *Design* variable specifies the active design. If *Design* is specified, it must be the first command line argument following the program name.

The *Design* variable must match one of the projects and designs already listed in the [ProjectDefinitions] section of the user initialization file (atmel.ini). It is not necessary to include the project name when specifying the *Design* variable because the program will search all project specifications to find a match. If there is more than one match, the program will select the project specification with a "*" in front, or ask for clarification.

Options

The following options are common to all IDS programs except where it does not make sense to have a particular option (e.g., specifying the input file for the Design Manager).

/d name: Design configuration file name

Programs use the *design*.cfg file in the design directory by default. The /d option is used to specify an alternative configuration file. The file name given must include all directory information.

/g: Go using default user settings

The /g option is used to start a program using default user and design settings. If any command line options are used, the /g option can be omitted.

/i name: Technology input file name

Programs use the *design*.cdb file in the active *project**design* directory by default. The /i option is used to specify an alternative data base file. The file name given must include all directory information.

/o name: Technology output file name

Programs write output to a *design*.* file in the active *project**design* directory by default. The /o option is used to specify an alternative name for technology output files.

If the program produces more than one technology output file, then the extension can not be specified using the /o option. For example, NetOut produces three output files: *.pyo, *.sig, and *.1. The following example gives an incorrect and correct invocation of NetOut.

- Incorrect: NETOUT /o C:\SomeDir\SomeFile.Ext
- Correct: NETOUT /o C:\SomeDir\SomeFile

The name given must include all directory information.

/u name: User initialization file name

Programs use the atmel.ini file in the current directory as the user initialization file by default. The /u option is used to specify an alternative initialization file. The name given must include all directory information.

AtErc

AtErc is intended to run against the *.cdb file and produce a report detailing all errors found.

Two types of rules are checked by AtErc. The first type, called structural rules, is mainly concerned with the legality of the chip configuration bits.

The second type, called functional rules, is concerned with the performance of the programmed device.

```
ATERC Version 6.00
Copyright (c) 1991-1998, Atmel Corp. All rights reserved
Usage: ATERC [Design] [/Options]
Design    : DesignName or ProjectDir\DesignName to process
/d name   : Design configuration file name
/g        : Go using the default user settings
/i name   : Technology input file name
/u name   : User initialization file name
```

Syntax

ATERC [Design] [/Options]

Bstr

The Bit Stream program generates an ASCII file containing 1's and 0's of a design's *.cdb file, which is then used to program Atmel's family of field programmable gate arrays.

```
BSTR Version 6.00
Copyright (c) 1991-1998, Atmel Corp. All rights reserved
Usage: BSTR [Design] [/Options]
Design    : DesignName or ProjectDir\DesignName to process
/a nnnnn  : DMA address in the range 0 to 1FFFF
/c        :Cache-in new configuration information
/d name   : Design configuration file name
/g        : Go using the default user settings
/i name   : Technology input file name
/m n      : Programming mode in the range 1 to 6
/n        : Blank do nothing file
/o name   : Technology output file name
/p cccc   : Programming order
/r nn     : Configuration register in the range 0 to 255
/s        : Generate randomly scrambled windows
/t        : Generate test address bit stream file
/u name   : User initialization file name
```

Syntax

BSTR [Design] [/Options]

Options

/a nnnnn: DMA address in the range 0 to 1FFFF

This DMA address specification in the header allows the user to store configuration files as a continuous stream, or as a pointer-based linked list. The address is only used in modes 1, 2, and 5.

Example: BSTR /a 1234

/c: Cache-in new configuration information

This option is used by Figaro and causes the BSTR program to implement specified Cache Logic design changes in the resulting configuration bitstream.

Example: BSTR /c

/m n: Programming mode in the range 1 to 6

This option specifies the mode needed to program a part. Different modes may contain different header and file formats.

Example: BSTR /m 4

/n: Blank do nothing file

This option specifies that the program is to output a bit stream file that contains only the preamble and postamble appropriate for the current design configuration settings.

/p cccc: Programming order

This option can be used to specify the order in which the various sections of the chip are programmed. There are four main sections of the chip: cells ('C'), repeaters ('R'), IOs ('I'), and corner cells ('J'). The IO section can be divided into four sub-sections as necessary. These four are the North IO ('N'), East IO ('E'), South IO ('S'), and West IO ('W'). The default programming order is 'CRIJ' which is cells, repeaters, IOs, and corner cells. The default order of the IOs is 'SNWE'.

The order can be modified by mixing up the sequence of the letters in the string 'CRIJ' or 'CRSNWEJ'.

Example: BSTR /p JNSCREW

/r nn: Configuration register in the range 0 to 255

This is an 8-bit value that is used to control various configuration sequence parameters:

B0: Determines whether the device's memory address counter is reset after each configuration sequence (default), or if it retains its last value. When this bit is 0 the DMA address in modes 1 and 5 will be set to 00000, but will be set to 1FFFF in mode 2 configuration. If the bit is 1, it retains its last value, and the user can store multiple designs in sequential configuration files.

B1: Determines whether the DMA address in the header field(s) used in modes 1, 2, and 5 is ignored or loaded into the device's memory address counter.

B2: This disables the /CEN, DATAOUT and CLKOUT functions for a minimum pin-count configuration circuit.

B3: Disables the configuration function of the /ERR pin and disables the data compare capability provided by the /CHECK pin. This is useful for design security and minimum pin-count configuration.

B4: Prevents configuration data from being written into the device following the initial configuration sequence. Reset this bit by rebooting the device.

B5: This bit disables the global clock when the CONN and CSN pins are set low. If the bit is clear (0), the clock continues to run during configuration.

B6: Determines whether the diagnostic clock mode is enabled. In this mode, the clock is stopped for both the configuration process and the operation mode. The internal global clock signal is pulsed once every time the CONN and CSN pins are brought low.

B7: This bit disables the internal configuration clock and puts the part in the lowest power state.

Example: BSTR /r 123

/s: Generate randomly scrambled windows

This option is related to the "/p" option. Instead of the user specifying the order, the program will randomly pick an order. This option will make it more difficult to decode a bit stream file and provide a simple means of security.

/t: Generate test address bit stream file

When this option is selected, addresses are also produced along with the bit stream data within the bit stream file for reference.

BstrWin

The program will compare two specified bit stream files and produce a "windowed" output file. A bit stream output file is compared against the 'baseline' file. If the character of the data reflects a change of 5 bytes or more, the difference is "windowed" and placed in the bit stream output file. When the baseline file is undefined, BstrWin will compress the bit stream output file. In either case, a maximum of 256 windows can be produced.

```
BSTRWIN Version 6.00
Copyright (c) 1991-1998, Atmel Corp. All rights reserved
Usage: BSTRWIN [Design] [/Options]
Design    : DesignName or ProjectDir\DesignName
/b name   : Batch control file name
/d name   : Design configuration file name
/g        : Go using the default user settings
/i name   : Technology input file name
/o name   : Technology output file name
/u name   : User initialization file name
/w name   : Window 'baseline' file name
/y        : Answer all prompts with 'Yes'
```

Syntax

BSTRWIN [Design] [/Options]

Options

/b name: Batch control file name

This option directs BSTRWIN to use the file specified as command-line input. Use this option when the command-line length exceeds that allowed by DOS. All options must be specified on a single line.

Example: BSTRWIN *design* /b *design*.bat

/w name: Window 'baseline' file name

This is the "baseline" file. If this name is omitted, BstrWin will compress the bit stream output file. The baseline file is the bit stream file that is assumed to be the initial or original design. The input bit stream file is the design that has been modified.

Example: BSTRWIN /w c:\at*user**project**design**design*.bas

/y: Answer all prompts with 'Yes'

BstrWin will assume an affirmative response to all questions asked.

Cascade

Cascading is an optional step that creates a concatenated bitstream file from one or more bitstream programming files.

```
CASCADE Version 6.00
Copyright (c) 1991-1998, Atmel Corp. All rights reserved
Usage: CASCADE [Design] [/Options]
Design    : DesignName or ProjectDir\DesignName to process
/c name   : Cascade or 'list' file name
/d name   : Design configuration file name
/e size   : Size of the configuration data EPROM
/g        : Go using the default user settings
/i name   : Technology input file name
/o name   : Technology output file name
/s        : Sequence program files
/u name   : User initialization file name
/y        : Answer all prompts with 'Yes'
```

Syntax

CASCADE [Design] [/Options]

Options

/c name: 'list' file name

This is a file containing a list of bitstreams to be cascaded. Bitstreams are cascaded together in an order specified in the list file.

Example: CASCADE /c design.bsl

The first line in the list file should contain the keyword [BitStreamList], followed by the name of the master bitstream. Each subsequent line should have the names of the slave bitstreams that are to be cascaded together.

Example: [BitStreamList]
c:*atuser**project**design1**design1*.bst
c:*atuser**project**design2**design2*.bst
c:*atuser**project**design3**design3*.bst

/e size: Size of the configuration data EPROM

This option is used to let the program know the size of the target EPROM used to store the cascaded file. The program can then check to ensure the bitstreams fit in the specified device. It is also used to determine the padding required. The EPROM size units are in K bytes.

Example: CASCADE c:*atuser*\4bitalu -c 4bitalu.bsl -o 4bitalu.bst -e 8

/s: Sequence program files

This option controls how the program will cascade the various bitstream files together. Without this option the files are combined into one large bitstream file with a single preamble and postamble. With this option the files are combined into a single file but still retain their individual identities (i.e. each has its own preamble and postamble).

Example: CASCADE /c design.bsl /s

/y: Answer all prompts with 'Yes'

Cascade will assume an affirmative response to all questions asked. This option should always be used when running the program in batch mode.

CF

The CF program is a software utility for programming, verifying, and reading serial memory devices using the ATDH2200 Configurator Memory prototype board.

```
CF Version 6.00
Copyright (c) 1991-1998, Atmel Corp. All rights reserved
Usage          : CF [/Options]
/A             : Program and verify AT17Cxxx from an Altera file
/B             : Convert an Altera file to an Atmel .BST file
/C             : A2 high when programming reset; low otherwise
/D port        : Parallel port: [LPT1|LPT2|LPT3] (default is LPT1)
/E             : Program and verify AT17Cxxx from a Xilinx .hex file
/F format      : Output file format. Can be .hex or .BST (default is .BST)
/G             : Data is for an Atmel AT40K FPGA
/H             : Convert a Xilinx .hex file to an Atmel .BST file
/I name        : Input file name
/J polarity    : Specify A2 bit polarity, H (high) or L (default)
/K checksum    : Compare checksum for device read & verify operations
/L             : Append hexadecimal values to each .BST output file line
/M [enable|disable]:AT17CxxxA clock generator enable/disable control
/O name        : Output file name
/P             : Program and verify AT17Cxxx from an Atmel .BST file
/Q             :Do not program reset level
/R             : Read AT17Cxxx and save to a .BST file (program mode)
/S size        : Size of device, 65K, 128K or 256K Bit
/T speed       : Operating speed. 'speed' can be SLOW, MEDIUM, or FAST
/U             : Verify any 17Cxxx contents against an Atmel .BST file
/V             : Verify AT17Cxxx against a .BST file (program mode)
/Z [H|L]       : Device RESET level, L (active low) or H (default)
/?              Help
```

Syntax

CF [/Options]

Options

/A: Program and verify AT17Cxxx from an Altera file

This option allows programming from an Altera .pof, .rbf, or .mcs (Altera .hex format) file. In order for the input file to be correctly processed, the appropriate input file name extension must be used. The input file is specified using the /I option.

An output file is also generated which contains the converted input file data in the Atmel .BST format. The output file name can be specified using the /O option. If the /O option is not used, the output file name will be the same as the input file name with either a .BST or .hex extension added.

The /F option can be used to specify the format of the output file to be in either .hex or Atmel .BST format. If the output file is in the latter format, the /L option can be used to force hexadecimal byte values to be appended to each output file line.

The size of the target device can be specified with the /S option (default size is 65). If the size of the input file data exceeds that specified using the /S option, the user can partition the input file data across multiple devices by following the on-screen prompts. Where the input file data is partitioned across multiple AT17Cxxx devices, a separate output file is generated for the contents of each device. On-screen messages indicate the names of each of these files.

The RESET polarity for the target AT17Cxxx device(s) can be specified using the /Z option. By default, the RESET polarity is high.

Example: CF /A /I *altera*.pof /Z H /S 65 /F bst /L

/B: Convert an Altera file to an Atmel .BST file

This option converts the Altera format (.pof, .mcs, or .rbf) input file, specified using the /I option, into an Atmel format .hex or .BST file. The output file name can be specified using the /O option. The particular format of the output file can be set using the /F option. The default output file format is the Atmel .BST file format.

The output file can be split into multiple output files by using the /S option. In this case, each output file will contain data corresponding to a single 65, 128, or 256 Kbits AT17Cxxx device.

If the output file is in the Atmel .BST format, and the /L option is specified, each line of the output file will contain a hexadecimal byte value in addition to the normal binary value.

/C: A2 high when programming reset; low otherwise

This option can be specified if it is required that the AT17Cxxx command byte A2 bit is set low for data programming, and high for RESET programming. By default, the A2 bit is set low for both data and RESET programming.

/D port: Parallel port: [LPT1|LPT2|LPT3] (default is LPT1)

By using the /D option, the PC parallel port that is used can be selected as LPT1, LPT2, or LPT3. By default, LPT1 is used.

Example: CF /D LPT2 /A /I *altera*.mcs

/E: Program and verify AT17Cxxx from a Xilinx .hex file

This option allows programming from a Xilinx .hex (.mcs) format file. The input file name is specified using the /I option.

An output file is also generated which contains the converted input file data in the Atmel .BST format. The output file name can be specified using the /O option. If the /O option is not used, the output file name will be the same as the input file name with either a .BST or .hex extension added.

The /F option can be used to specify the format of the output file to be in either .hex or Atmel .BST format. If the output file is in the Atmel .BST format, the /L option can be used to force hexadecimal byte values to be appended to each output file line.

The size of the target device can be specified with the /S option (default size is 65). If the size of the input file data exceeds that specified using the /S option, the user can partition the input file data across multiple devices by following the on-screen prompts. Where the input file data is partitioned across multiple AT17Cxxx devices, a separate output file is generated for the contents of each device. On-screen messages indicate the names of each of these files.

The RESET polarity for the target AT17Cxxx device(s) can be specified using the /Z option. By default, the RESET polarity is high.

Example: CF /E /I *xilinx*.hex /Z H /S 128 /F bst /L

/F format: Output file format. Can be .hex or .BST (default is .BST)

The format of the output files can be set to .hex or Atmel .BST using this option. Output files are created when using the /A, /B, /E, /H, and /P options, or splitting the input file over multiple target AT17Cxxx devices.

Example: CF /R /O *atmel*.hex /F hex

/G: Data is for an Atmel AT40K FPGA

When the configurator is used to program an Atmel AT40K series FPGA, this option ensures that the data is correctly formatted.

Example: CF /I *at40k*.*bst* /G /S 128 /Z L

/H: Convert a Xilinx .hex file to an Atmel .BST file

A Xilinx .hex (.mcs) file can be converted to an Atmel .BST format file using this option. The input Xilinx .hex file is specified using the /I option. The output file can be specified using the /O option.

The output file can be split into multiple output files by using the /S option. In this case, each output file will contain data corresponding to 65, 128, or 256 Kbits AT17Cxxx devices.

If the output file is in the Atmel .BST format, and the /L option is specified, each line of the output file will contain a hexadecimal byte value in addition to the normal binary value.

Example: CF /H /I *xilinx*.hex /O *atmel*.BST

/J polarity: Specify A2 bit polarity, H (high) or L (default)

This option is used to set the A2 bit polarity used by CF when programming or reading an AT17Cxxx device.

Example: CF /P /I *atmel*.BST /J H

/K checksum :Compare checksum for device read & verify operations

Whenever an AT17Cxxx device is programmed using CF, a checksum (sum of bytes written to the target device) is reported when programming is complete. When reading or verifying an AT17Cxxx device contents, the /K option can be used to compare a number with the sum of the bytes contained in the target device. Any difference will be reported by CF.

Example: CF /R /O *atmel*.BST /K 32456

/L: Append hexadecimal values to each .BST output file line

The Atmel .BST file is in ASCII format where each file data byte is represented by a binary string on each file line. When CF is used with command-line options to generate an output file in the Atmel .BST format, the /L option can be used to append data byte values to each line in hexadecimal format.

Example: CF /A /I *altera*.pof /O *atmel*.BST /F bst /L

/M [enable|disable]: AT17CxxxA clock generator enable/disable control

This command-line option is used to enable or disable the internal oscillator in AT17C512A and AT17C010A devices.

Example: CF /A /I *altera*.pof /M enable

/P: Program and verify AT17Cxxx from an Atmel .BST file

This option allows programming from an Atmel .BST format file. The input file name is specified using the /I option.

The size of the target device can be specified with the /S option (default size is 65). If the size of the input file data exceeds that specified using the /S option, the user can partition the input file data across multiple devices by following the on-screen prompts. Where the input file data is partitioned across multiple AT17Cxxx devices, a separate output file is generated for the contents of each device. On-screen messages indicate the names of each of these files.

In cases where output files are created, the /F option can be used to specify the format of the output file to be in either .hex or Atmel .BST format. If the output file is in the Atmel .BST format, the /L option can be used to force hexadecimal byte values to be appended to each output file line. By default, the format of output files is .BST.

The RESET polarity for the target AT17Cxxx device(s) can be specified using the /Z option. By default, the RESET polarity is high.

Example: CF /P /I *atmel*.BST /Z H /S 256 /F hex /L

/Q: Do not program reset level

Normally, CF will program the reset polarity on the target AT17Cxxx device. If the /Z command-line option is not used with CF then the default reset polarity is active high. By specifying the /Q option, CF will not program the reset polarity in the target device. After programming, the reset polarity will be whatever level it was prior to programming.

Example: CF /P /I atmel.BST /Q

/R: Read AT17Cxxx and save to a .BST file (program mode)

When this option is specified, the contents of the AT17Cxxx device are read and stored in a file. The output file name must be specified using the /O option, and the format of the output file can be in .hex or Atmel .BST, as set by the /F option. The default file format is the Atmel .BST format.

The volume of data read back can be set to 65, 128, or 256 Kbits using the /S option. By default, 65 Kbits of data is read.

If the /L option is used, and the specified output file is in the Atmel .BST file format, each line of the output file will have a hexadecimal byte value appended.

Example: CF /R /O *atmel*.BST /S 128 /F bst /L

/S size: Size of device, 65K, 128K, or 256K bit

The size of the target device can be specified as 65, 128, or 256 Kbits using this option.

Example: CF /P /I *atmel*.BST /S 128

/T speed: Operating speed. 'speed' can be SLOW, MEDIUM, or FAST

This option can be used to set the operating speed during device programming, verification, and read operations. By default, the operating speed is FAST.

Example: CF /P /I *atmel*.BST /T MEDIUM

/U: Verify any 17Cxxx contents against an Atmel .BST file

With the ATDH2200 board in the standard READ mode, this option can be used to verify the device contents against a reference file. The reference file must be specified using the /I option.

The RESET polarity of the device can be specified using the /Z option.

Example: CF /U /I *atmel*.BST /Z L

/V: Verify AT17Cxxx against a .BST file (program mode)

This option verifies the contents of an AT17Cxxx device against a reference file while the ATDH2200 board is in the PROGRAM mode. The reference file must be specified using the /I option.

Example: CF /V /I *atmel*.BST

/Z [H|L]: Device RESET level, L (active low) or H (default)

Specifies the RESET polarity of the target device for programming (ATDH2200 PROGRAM mode) or read (ATDH2200 READ mode) operations.

Example: CF /P /I *atmel*.BST /Z H

Downld

The download utility consists of 2 programs: downld and downld40 for the AT6K and AT40K devices respectively.

The syntax for downloading AT6000 series FPGA bit stream data through the specified parallel port is as follows.

```
DOWNLD Version 6.00
Copyright (c) 1991-1998, Atmel Corp. All rights reserved
Usage: DOWNLD [/Options] filename[.BST]
/p LPTn   : Use parallel port specified (must be LPT1, LPT2, or LPT3)
```

Syntax

DOWNLD [/Options] filename[.BST]

Additional Command-Line Arguments

/p LPTn: Use parallel port specified (must be LPT1, LPT2, or LPT3)

Set to the port where the device programming board is attached.

Example: DOWNLD LPT3 *Design*.BST

Downld40

This program will download AT40K series FPGA bit stream data through the specified parallel port.

```
DOWNLD40 Version 6.00
Copyright (c) 1991-1998, Atmel Corp. All rights reserved
Usage: DOWNLD40 [/Options] filename[.BST]
/p LPTn   : Use parallel port specified (must be LPT1, LPT2, or LPT3)
```

Syntax

DOWNLD40 [/Options] filename[.BST]

Additional Command-Line Arguments

/p LPTn: Use parallel port specified (must be LPT1, LPT2, or LPT3)

Set to the port where the device programming board is attached.

Example: DOWNLD40 LPT3 *Design*.BST

MakeKey

The program should be run from a project directory that has been created by the Design Manager. By default it will update all files in both the "sch" and "sym" directories. By using the /i command-line option, a specific schematic or symbol can be specified.

```
MAKEKEY Version 6.00
Copyright (c) 1991-1998, Atmel Corp. All rights reserved
Usage: MAKEKEY [Design] [/Options]
Design    : DesignName or ProjectDir\DesignName to process
/d name   : Design configuration file name
/g        : Go using the default user settings
/i name   : Technology input file name
/u name   : User initialization file name
/w        : Update wir files only
```

Syntax

MAKEKEY [Design] [/Options]

Options

/i: Technology input file name

This option instructs MakeKey to update the schematics and symbols that match the given name.

Example: MAKEKEY /i test

This command will update the files sch/test.* and sym/test.*.

/w: Update wir files only

This option instructs MakeKey to update the wir files in the current project directory. It can also be used in conjunction with the /i option to update specific wir files

Example: MAKEKEY /w /i test

This command will update the files wir/test.* in the current project directory.

PLA2Cdb

The PLA2Cdb program is a technology-mapping program to translate PLA design files optimized using ABEL and CUPL compilers. This program is useful for creating a soft macro of a PLA file. There is also a provision to insert I/O pads if PLA file is to be directly placed and routed using Figaro.

PLA2Cdb is to be used with PROgen schematic generation utility and macro generation utilities of Atmel's Integrated Development System.

```
PLA2CDB Version 6.00
Copyright (c) 1991-1998, Atmel Corp. All rights reserved
Usage: PLA2CDB [Design] [CUPL] [/Options]]
Design    : DesignName or ProjectDir\DesignName
CUPL      : Read pla file produced by CUPL
/c        : Insert I/O pads to design pins
/d name   : Design Configuration file name
/g        : Go using the default user settings
/u name   : User initialization file name
/x        : Map AND-XOR PLA
```

Syntax

PLA2CDB [Design] [CUPL] [/Options]

Options

/c: Insert I/O pads to design pins

This option connects I/O pads to the design pins. By default, ITTL, OD and ODEN are used. The user defined pad types can be specified in the *.pin file which contains the pin name and pad type on each line.

Example: PLA2CDB count /x /c

The command will direct the PLA2Cdb program to read count.ttx file produced by REEDMUL and insert I/O pads to the pins of the design.

A sample count.pin file for the design is shown below.

```
Q0 ODF
Q1 ODF
Q2 ODF
Q3 ODF
COUT ODF
```

/x: Map AND-XOR PLA

This option reads the *.ttx file produced by the REEDMUL program and maps it to FPGA gates.

CUPL: Read Output of a CUPL compiler

Specifying this keyword on the command-line directs PLA2Cdb to read a Berkeley PLA formatted file generated by a CUPL compiler. With this keyword specified, PLA2Cdb looks for a file with .pla extension.

Example: PLA2CDB new2cnt /t CUPL
Optimize new2cnt.pla file generated by CUPL compiler.

Reedmul

REEDMUL is a logic minimization program used to optimize logic using AND-XOR optimization. The program reads the *.tt1 file generated using DATA I/O's ABEL or *.pla file generated using the CUPL compiler.

This program should be used very carefully as AND-XOR optimization will produce good results only on arithmetic intensive designs. For other types of designs, the PLA optimization capability provided by CUPL and DATA I/O ABEL compilers should be used.

```
REEDMUL Version 6.00
Copyright (c) 1991-1998, Atmel Corp. All rights reserved
Usage: REEDMUL [Design] [CUPL]
Design    : DesignName or ProjectDir\DesignName
CUPL      : Read .pla file produced by CUPL
/d name   : Design Configuration file name
/g        : Go using the default user settings
/u name   : User initialization file name
```

Syntax

REEDMUL [Design] [CUPL]

Options

CUPL: Read Output of a CUPL compiler

Specifying this keyword on the command-line directs the REEDMUL program to read a Berkeley PLA formatted file generated by the CUPL compiler. This keyword directs REEDMUL to look for a file with the *.pla extension.

Example: REEDMUL new2cnt CUPL

Program Messages

The Program Messages section of the Technical reference manual is divided into two parts. The first part, called General Messages, contains common messages that may be displayed by any program. The second part contains program -specific messages. Entries in the General Messages section are sorted by message number for ease in locating the appropriate message. The program specific messages are stored under each program's section. Within that program's section, the messages are also sorted by message number. Messages displayed by Figaro can be found either in the IDS Messages section or in the on-line Help. These messages do not contain message numbers.

Except for the IDS program messages, message numbers are of the form [XNNN] where X is a letter from "a-z" and NNN is a three-digit number. The number will be displayed at the end of each warning, error, or fatal error message. Status or note messages will not generally have message numbers associated with them. To locate a message refer to the following table to determine which program section it will be listed under

b1	Bstr	**d0**	General1
b2	BstrWin	**g0**	General2
c0	Cascade	**m0**	MakeKey
c2	AtErc	**p6**	PLA2Cdb
c8	Cf	**u4**	DownLd

Messages displayed by programs fall into four categories that are FATAL ERROR, ERROR, WARNING, and NOTE. Except for the NOTE type of message, all messages are preceded by one of the above four prefixes. The same message may apply to several categories depending on the actual problem. The following is an explanation of what should be done when a program displays each of these types of messages.

General Messages

This section is a listing of all of the messages that are common to many of the programs in the IDS. They are listed alphabetically. All messages will contain the text of the message, an example, a description and a solution. Italics are used to specify variables that occur in the messages. The following is a brief description of what these common variables mean:

(Col,Row): used to indicate a location in the array.

Command: command text specified to run a program.

Component: the library type of the symbol or macro.

Count: used when counting a number of items.

Day: the day used in displaying dates.

Design: the name of the design.

Directory: the name of a directory path.

Element: refers to an entity inside the design database, could be a component, instance, or primitive.

Filename: the name of a file can be complete (all directory information specified) or simple (no directory information specified).

Hour: the hour used in displaying dates.

Instance: the name of an instance.

Location: used to indicate an internal pin location or direction.

Minute: the minute used in displaying dates.

Month: the month used in displaying dates.

Netname: the name of a net.

Nettype: one of Clock, Reset, Tri.

Number: numeric values that are not used as counts.

Percent: used to indicate a percentage of completion of a task.

Pinname: the name of a pin either physical or logical.

Problem: the text of some problem with a program.

Program: the name of a particular program.

Project: the name of the project.

Project\Design: the complete path to the directory where design information is stored.

Second: the second used in displaying dates.

TableName: the name of an internal program table.

Text: any miscellaneous string that may be output from a program.

Type: either user initialization or design configuration file or can be used for CELL, IO, BUS, CLOCK, RESET elements.

Value: the value that a particular variable should be set to.

Year: the year used in displaying dates.

Program Messages

Found and fixed number instance(s) of an invalid version of component "EN83". Read the full explanation of this message in the General Message section of the Technical Reference & Release Notes guide [d038]

Example

Found and fixed 10 instances(s) of an invalid version of component "EN83". Read the full explanation of this message in the General Message section of the Technical Reference & Release Notes Guide

Description

This message is displayed if the design uses a version of the component "EN83" that was found to have a problem. Discovered after the version 1.0 libraries were distributed, the problem has been fixed for the 1.1 release, and the software enhanced to look for and repair the problem.

Solution

This is just an informational message to notify the user that the design had been using an invalid version of the component. The Incremental Design Change program should be run to update any database that may be using this component. Any bit streams created using the invalid component should also be re-generated.

Removed unsupported turn at (Col,Row) [d040]

Example

Removed unsupported turn at (10,10)

Description

Please refer to the description of message [d042].

Solution

Please refer to the solution for message [d042].

Removed unsupported bus connection at (Col,Row) [d041]

Example

Removed unsupported bus connection at (10,8)

Description

Please refer to the description of message [d043].

Solution

Please refer to the solution for message [d043].

Found and removed number unsupported "turns" in the design. Read the full explanation of this message in the General Message section of the Technical Reference & Release Notes guide [d042]

Example

Found and removed 10 unsupported "turns" in the design. Read the full explanation of this message in the General Message section of the Technical Reference & Release Notes.

Description

Due to an enhancement in the architecture, the ability to make any and all connections from a turn cell has been changed. The turns supported are between the north and west buses and between the south and east buses. The software will determine which turns in the design are allowed and will convert them as required. If a turn is not allowed it will be removed and will leave a dangling wire. The program list file will display the location of each turn that was removed.

Solution

The user should either run the automatic router or the interactive editor to re-connect the open nets. The program list file will contain the location of each removed turn to assist in interactive editing.

Found and removed number unsupported bus connections in the design. Read the full explanation of this message in the General Message section of the Technical Reference & Release Notes guide [d043]

Example

Found and removed 2 unsupported bus connections in the design. Read the full explanation of this message in the General Message section of the Technical Reference & Release Notes guide

Description

Due to an enhancement in the architecture the ability to make any and all connections in a repeater has been changed. The bus connections not supported are those which allow the bus to change directions (i.e. make a "U" turn). For example, the following situation is no longer allowed. A local bus connects to an express bus to the East of it. That express bus in turn connects to an express bus to the West of it, thus changing direction. The software will determine which bus connections in the design are not allowed and will remove them. This will leave a dangling wire. The program list file will display the location of each connection that had to be removed.

Solution

The user should either run the automatic router or the interactive editor to re-connect the open nets. The program list file will contain the location of each removed connection to assist in interactive editing.

See Page 56 for more [dxxx] messages

"Instance" has no Nettype pin for use by its child element "element" [g001]

Example

"$1I1" has no CLK pin for use by its child element "$1I1.$E1"

Description

If a component contains a primitive that is a flip-flop it must have a clock and a reset pin. If the clock and reset are missing, this message is displayed.

Solution

If this message is displayed for a component that is in the layout or symbol library, report the problem to Customer Service. If this message is displayed for a user-created layout or symbol, be sure to add the needed clock and reset pins.

"Instance" missing pin "Pinname" used by component "component" [g002]

Example

"$1I1" missing pin "Q0" used by component "CRP4"

Description

This message is displayed when an instance of a component from the schematic does not use all of the pins that are available. It is also displayed when the symbol does not match the component layout or when a schematic symbol has unconnected pins.

Solution

If the pin is missing on the symbol, add it. If the pin should not be in the layout, remove it. Please contact Customer Service if the symbol from the IDS macro or layout library is missing a pin, or has an extra pin.

"Project\Design" directory cannot be found and is not part of a defined project [g003]

Example

"C:\AtUser\Project\Design" directory cannot be found and is not part of a defined project

Description

This message is displayed when the current project\design as specified by the atmel.ini file or by command-line options does not exist or is not a usable directory.

Solution

Check to see that the current project name in the atmel.ini file is spelled correctly. The current project is the one with the "*" in front of it in the [ProjectDefinitions] section of the atmel.ini file. If this name is correct, then make sure the directory which this current project points to exists. If it does not exist, create the appropriate directories.

This problem should not be encountered if the Design Manager is used to create the design.

"Text" is not a valid number. Valid numbers contain 0-9, '-', or '.' [g004]

Example

"1A0" is not a valid number. Valid number contain 0-9, '-', or '.'

Description

This message is displayed when a program encounters a string that does not contain only digits, '-', or '.'.

Solution

If this message appears at the very beginning of a program, it is probably due to an incorrectly specified command-line option. Run the program with no command-line arguments to see the proper syntax for that program's options. If this message appears later in a program run, the source of the error will be a part of the message.

Sequential backslashes '\' are not allowed [g005]

Description

More than one back slash '\' was found in a file specification.

Solution

Remove the extra back slashes from the file specification.

Sequential colons ':' are not allowed [g006]

Description

More than one colon ':' was found in a file specification.

Solution

Remove extra colon(s) from the file specification.

Sequential periods '.' are not allowed [g007]

Description

More than one period '.' was found in a file specification.

Solution

Remove the extra period(s) from the file specification.

A general permit failure has occurred. Please contact Customer Service. [g020]

Example

A general permit failure has occurred. Please contact Customer Service.

Description

The permit file is either invalid or missing.

Solution

Contact Customer Service.

A VGA board is required [g021]

Description

The computer running IDS does not use a VGA screen.

Solution

Upgrade the current video system to a VGA monitor and graphics card.

Access violation reported by DOS [g023]

Description

An access violation occurred while loading a program.

Solution

If on a network, make sure the access rights to the specified drive are set correctly. If not on a network, contact Customer Service.

Active project is not set in the user initialization file [g024]

Description

If the user initialization file, atmel.ini, does not have an active project specified, this message appears.

Solution

To specify an active project place an asterisk "*" at the beginning of one of the projects in the [ProjectDefinition] section of the initialization file. Such as in the following example:

[ProjectDefinitions]

*C:\AtUser\Project\Design

[LayoutEditor]

Check to see that there is an asterisk on one of the projects in the atmel.ini file as in the above example. If a project\design is being specified from the command-line it should match one that is already in the atmel.ini file.

Attribute "text" found in Pathfind command file "filename" is not allowed during Timing Estimation.

Please modify your command file and rerun Timing Estimation [g026]

Example

Attribute "WIRE_CAP" found in Pathfind command file "Design.pcf" is not allowed during Timing Estimation.

Please modify your command file and rerun Timing Estimation

Description

The specified file contains the attribute listed, but the attribute can not be used for Timing Estimation.

Solution

Comment out (by using the '/*' comment syntax) the attribute listed or delete the attribute line from the file.

All components in the design have not been placed. Routing will be incomplete [g027]

Description

The design currently being routed does not have all components placed. Any net that is connected to an unplaced component will not be routed.

Solution

Place all components that need to be placed and routed by using the automatic or interactive placement programs before using the router.

Cannot access the design log file, program aborting [g040]

Description

If there is something wrong with the specification of the project\design directory the program will not be able to access the log file and will abort.

Solution

All programs create a log file entry that specifies the program being run on the design, how it was run, and what the final results were. The log file is normally located in the project\design directory and is named design.log. Be sure that a current project is defined in the user initialization file and that the project\design directory specified by it exists and has the correct write permission.

Cannot access the program's list file, program aborting [g041]

Description

If there is something wrong with the specification of the project\design directory the program will not be able to access the list file and will abort.

Solution

All programs will create a list file that specifies the program being run on the design, how it was run, any messages that the user should know regarding the run of the program, and what the final results were. The list file is called *program*.lst and is located in the project\design directory. Be sure that a current project is defined in the user initialization file and that the project\design directory specified by it exists and has the correct write permission.

Cannot update file "filename" [g042]

Example

Cannot update file "C:\AtUser\Project\Design\Design.bak"

Description

Programs that produce data base files (*.cdb) will try to create backup files. If an existing backup is present (*.bak) the program will attempt to delete it. If it fails to delete the file, the above message is displayed.

Solution

It is possible that the file or directory that contains it is protected. Check the protection of the file or directory.

Cannot find active project\design setting [g043]

Description

The program could not determine the active project\design.

Solution

Select the active project\design in the Design Manager Setup.

Cannot find file "filename" [g045]

Example

Cannot find file "SMP.EXE"

Description

The program cannot find the specified file name.

Solution

Check the DOS PATH environmental variable for the specified file name.

Cannot find user initialization file "atmel.ini" [g046]

Example

Cannot find user initialization file "atmel.ini"

Description

The program could not open the user initialization file (atmel.ini).

Solution

If the atmel.ini file is not in the current directory, make sure the AtmelDIR environmental variable specifies the directory where the file is stored

Cannot open file "filename" for reading [g047]

Example

Cannot open file "C:\AtUser\Project\Design\Design.cdb" for reading

Description

The program cannot read from and write to the specified file.

Solution

Make sure the specified file is not read and write protected if it does exist. Otherwise make sure the directory is not write protected, so the specified file can be created.

Cannot open file "filename" for writing [g048]

Example

Cannot open file "C:\AtUser\Project\Design\Design.cdb" for writing

Description

This message is displayed when a program is unsuccessful in creating a file that is used to store program output. This problem is usually caused by an existing write-protected file with the same name. The directory intended to contain the file may also be write-protected or may not exist.

Solution

Check the paths specified on the command-line, the atmel.ini file, or the *design*.cfg file to make sure they are all spelled correctly.

Cannot process the command "text" [g049]

Example

Cannot process the command "AtNet C:\AtUser\Project"

Description

The program could not execute the specified process.

Solution

Check the DOS PATH environmental variable for the specified process.

Cannot process the package file. No package library was specified [g050]

Description

This message is displayed when an incorrect package library is specified.

Solution

There are a number of places where the package file to use is provided to the IDS software. Before the build database function, the user can specify the default package in the atmel.ini file (DefaultPackage keyword in the [Miscellaneous] section). Another specification is on the top-level symbol for the schematic. It is derived from the concatenation of the CLI_PART and CLI_PKG attributes. After the build data base function, the package is specified by the *.cdb file. Be sure that the correct package library is specified.

Colon must follow drive letter [g052]

Description

A colon does not follow the drive letter.

Solution

Place a colon after the drive letter.

Component "component" missing pin "Pinname" used by "instance" [g053]

Example

Component "CRP4" missing pin "Q0" used by "$1N1"

Description

This message is displayed when an instance from the schematic has a pin that does not exist on the component from the layout library.

Solution

If the component is a user-created symbol or layout, examine the symbol in Workview to determine if it has too many pins. If the symbol is correct, then add this pin to the component layout. If this component is from the macro and layout libraries, contact Customer Service.

Cannot find the design configuration file "design.cfg" [g055]

Description

The program could not open the design configuration file, design.cfg.

Solution

If on a network, make sure the access rights to the current location are set correctly. If not on a network, contact Customer Service.

Cannot move file "filename" to "filename" [g056]

Example

Cannot move file "design.yod" to "design.net"

Description

The program could not move a file from the specified source to the specified destination.

Solution

Make sure the destination file is unlocked. Also check that the source file exists. If is does not, contact Customer Service.

Cannot open design configuration file "design.cfg" [g057]

Description

The program could not open the design configuration file, design.cfg.

Solution

Make sure there is sufficient room on the current drive. Delete any unnecessary files. If on a network, make sure the access rights to the current location are set correctly.

Cannot open user initialization file "atmel.ini" [g058]

Description

The program could not open the user initialization file, atmel.ini.

Solution

Make sure there is sufficient room on the current drive. Delete any unnecessary files. If on a network, make sure the access rights to the current location are set correctly.

Cannot open "viewdraw.ini" [g059]

Description

The program could not open the Viewlogic initialization file VIEWDRAW.INI.

Solution

The program uses the environmental variable WDIR to find the configuration file. Make sure that WDIR points to the correct location.

Cannot write to file due to: text [g060]

Example

Could not write to file due to Write Failed

Description

This message is displayed when the program has a problem writing its output to a file. The reason for the problem is given in the message but is not always self-explanatory.

Solution

"Write Failed" is a common problem, typically caused by a full disk. Clear up enough space for the program to write its various outputs. One megabyte is usually enough disk space for the output of most programs.

Current data base "filename" not backed up, problem moving to file "filename" [g062]

Example

Current data base "C:\AtUser\Project\Design\Design.cdb" not backed up, problem moving to file "C:\AtUser\Project\Design\Design.bak"

Description

If an old backup file is protected in some manner so that the original input database cannot be renamed to design.bak, this message is displayed.

Solution

Most programs that produce data base files (Design.cdb) will first write their output to a temporary file (xxxxxxxx.tmp). Upon successful completion of whatever it is that the program does it will try to rename the original input database from its current name to design.bak. Then it will try to rename the temporary file to design.cdb. The user should determine if the backup file design.bak file is protected in some manner so that it cannot be renamed. Once the problem is cleared up the user should re-run the affected program because it would not have generated the needed output file(s).

Cannot find directory "Directory" [g065]

Description

The program cannot find the specified directory.

Solution

Check the DOS PATH environmental variable for the specified directory.

Cannot open file "filename" for reading and writing [g066]

Description

The program cannot read from and write to the specified file.

Solution

Make sure the specified file is not read and write protected if it does exist. Otherwise make sure the directory is not write protected, so the specified file can be created.

Cannot update file "filename" [g067]

Description

The program cannot update the specified file.

Solution

Make sure the file and directory are not write protected.

Cannot find environmental variable WDIR [g068]

Description

The program could not find the environmental variable 'WDIR'.

Solution

Set the environmental variable to point to the Workview standard location. Both Integrated Design System and Viewdraw use this value.

Cannot create directory "Directory" [g069]

Example

Cannot create directory C:\AtUser\Project\Design

Description

The program could not create the path specified.

Solution

Make sure there is enough room on the current drive. Delete any unnecessary files. If on a network, make sure the access rights to the current location are set correctly.

Cannot rename "filename" to "filename" [g070]

Example

Cannot rename "tmp1.$$$" to "atmel.ini"

Description

The program could not rename the file names specified.

Solution

If on a network, make sure the access rights to the current location are set correctly. If not on a network, contact Customer Service.

Cannot create file "filename" [g072]

Example

Cannot create file "TEMP.MAC"

Description

The program could not create the specified file name.

Solution

Make sure there is sufficient room on the current drive. Delete any unnecessary files. If on a network, make sure the access rights to the current location are set correctly.

Cannot delete file "filename" [g078]

Example

Cannot delete file "TEMP.MAC"

Description

The program could not delete the specified file name.

Solution

This message should only be displayed if the specified file exists but the program cannot delete it. Make sure the file, directory, or disk that contains the file is not write protected.

Design contains count components that exceeds the maximum of number [g082]

Example

Design contains 5000 Cells which exceeds the maximum of 3136.

Design contains 150 IO's,, which exceed the maximum of 108.

Design contains 65 Clocks and Resets which exceeds the maximum of 56.

Description

When the program is calculating the design complexity rating it also determines the number of different types of elements used in the design. If any of these types of elements: cells, IO's, or clocks and resets exceed the maximum available. This message is displayed.

Solution

Modify the schematic so that it does not use more than the resources available, or, if this is not possible, use a larger device or partition the design to fit on more than one device.

Directory does not exist "Directory" [g085]

Example

Directory does not exist "C:\AtUser\Project\Design"

Description

Because all of the inputs and outputs to the programs in the IDS system require data to be stored and retrieved from directories, it is important that all of these directories be created for use by the programs. The Design Manager will create all the needed directories as requested by the user and update the entries that are required in the user initialization and design configuration files.

Solution

Check to see that the directory exists and that it is specified correctly. Look in the user-initialization file (atmel.ini) under the [ProjectDefinitions] section to make sure that the paths specified there are correct. The design configuration file (*design*.cfg) also contains paths that may have been specified incorrectly.

Disk full while writing to file "SCREEN.CAP" [g086]

Description

The disk is now full.

Solution

Remove all unnecessary files from the disk and retry.

DMA address is not within range of 0 to 131071 [g087]

Description

The address entered is not in the range 0 to 131071.

Solution

Specify an address within the 0 to 131071 range.

Missing preamble byte in file "filename" [g088]

Example

Missing preamble byte in file "design.bst"

Description

The program is checking the specified file to make sure that it is a bit stream file. Each such file should have header information that identifies it as a bit stream file. The preamble is the first part of the header.

Solution

Make sure the specified file name is correct and re-specify as needed. If the name is correct, the file has probably been corrupted and needs to be regenerated.

Missing postamble byte in file "filename" [g089]

Example

Missing postamble byte in file "design.bst"

Description

The program is checking the specified file to make sure that it is a bit stream file. Each such file should have ending information that identifies it as a bit stream file. The postamble is the last part of the file.

Solution

Make sure the specified file name is correct and re-specify as needed. If the name is correct, the file has probably been corrupted and needs to be regenerated.

Drive letter first in specification [g090]

Description

The drive letter was not first in the file specification.

Solution

Enter the file specification with a drive letter at the beginning.

Drive letter missing in project directory specification [g091]

Description

The drive letter is not in the project directory specification.

Solution

Re-enter the full file specification for the project directory in the Design Manager Setup.

Drive not ready [g092]

Description

The program detected that your disk drive is not operational.

Solution

Be sure the floppy disk drive door is closed. If the problem continues, contact Customer Service.

Due to previous errors program cannot proceed, aborting [g093]

Description

This message is given if previous errors make it impossible for the program to continue running.

Solution

Solve the problems reported by accompanying error messages so the program can continue running.

Enter the Pathfind command file name [g110]

Description

Pathfind needs the command file name.

Solution

Enter the file specification to the Pathfind command file.

Environment not setup correctly, program aborting [g111]

Description

The program will abort if the information provided to it in the user initialization (atmel.ini) or design configuration (*design*.cfg) files is not complete.

Solution

Other messages accompany this message. Solving the problems noted by these messages should prevent the program from aborting again. Make sure there is a library directory (normally C:\atmel\lib) and a current project\design directory for reading input and writing output files to.

Exiting DM without changing "DIR [p]" in viewdraw.ini [g112]

Description

The value of "DIR [p]" does not match the project set in the user initialization file, atmel.ini. The program is quitting without modifying the viewdraw.ini file.

Solution

Make sure the value of "DIR [p]" matches the project set in the user initialization file, atmel.ini, before running the Design Manager again.

Expected hex character versus number [g113]

Example

Expected hex character versus 0x

Description

A command-line option requiring a hexadecimal value was given a non-hex character.

Solution

Re-enter command-line value in proper format.

File being added does not exist [g130]

Description

The program could not find the specified file.

Solution

Re-enter the file specification.

File does not exist "filename" [g131]

Example

File does not exist "C:\AtUser\Project\Design\Design.cdb"

Description

If the program cannot find a file, this message is displayed.

Solution

Make sure the file name or directory for the file is spelled correctly. Check the command-line arguments used to start the program, the user initialization (atmel.ini), or the design configuration (*design*.cfg) files for file and directory names.

Floating point value of "number" has been converted to integer number [g136]

Example

Floating point value of "25.123" has been converted to integer "25"

Description

This message is displayed when a program is attempting to read an integer but encounters a string that represents a floating-point number.

Solution

If this message appears at the very beginning of a program, it is probably due to an incorrectly specified command-line option. The user should run the program with no command-line arguments to see the proper syntax for that programs options. If this message appears later in a program run, the source of the error will be a part of the message. If the automatic conversion is not a problem, then the program will to work as needed.

Global "Nettype" net in the corners is blocked, cannot route to core [g150]

Example

Global "CLOCK" net in the corners is blocked, cannot route to core

Global "RESET" net in the corners is blocked, cannot route to core

Description

The cell used to bring a global clock or reset signal into the core of the chip is already dedicated for another purpose.

Solution

Use the interactive editor to relocate the offending cell.

Incorrect command line to run DMSECCHK (Design Manager Security Check) [g160]

Description

This program can only be invoked by the Design Manger system.

Solution

Return to the Design Manager to run the program.

Incorrect library. Expected version "number" versus "number" in "filename" [g161]

Example

Incorrect library. Expected version "2.1" versus "2.0" in "C:\Atmel\Lib\6005.132"

Description

This message is issued when an out-of-date version of a library is found.

Solution

Determine what type of file the program does not recognize. If it is a design data base or layout library then the wrong version of software is being used to process it. Contact Customer Service to request an upgrade to their current software.

If it is a package file or any other library file, update the library files. Contact Customer Service.

It is also possible that the design.cfg file is pointing to a directory that contains old library files.

Insufficient memory available for execution [g163]

Description

There wasn't enough memory to execute either the compression or decompression routines.

Solution

Remove any unnecessary memory resident (TSR) programs from memory (it may be necessary to modify the autoexec.bat file and reboot the computer).

Insufficient memory reported by DOS [g164]

Description

There is not enough main memory available to execute the program.

Solution

Remove all unnecessary memory resident (TSR) programs from memory and retry.

Internal program problem. Returned from text [g168]

Description

The program encountered an internal program problem.

Solution

Note the message number and report to Customer Service.

Library does not contain component "component", shape number [g180]

Example

Library does not contain component "AN2" shape 10

Description

This message is displayed if a program fails to find a component shape in the layout library.

Solution

This message can appear for three reasons. First, the SHAPE attribute of the instance may have been changed in the schematic to refer to a shape that does not exist. Check the SHAPE attribute to be sure it points to an available shape.

Second, the required library may not be referenced correctly. Check the design setting (design.cfg or DM panel) for the correct library path.

Third, the layout for the macro may not have been created yet. Be sure all required macros are complete before running the design through the automatic or interactive layout tools.

Mouse driver not installed [g190]

Description

The program could not find a mouse driver installed in memory.

Solution

Install the mouse driver and retry. Note: Logitech mice need to use the 'PC' option (mouse.com pc)

Moved file "filename" to "filename" [g191]

Example

Moved file "design.yod" to "design.net"

Description

The program has successfully moved a file from source to destination.

Solution

No solution is required.

Multiple projects use the design name "text". Please make one active [g192]

Example

Multiple projects use the design name "MyDes". Please make one active

Description

This message appears when a design name is specified on the command-line that matches more than one project specification in the [ProjectDefinition] section of the atmel.ini file.

Solution

Place an asterisk (*) at the beginning of the project in the [Project Definition] section that should be the design that needs to be processed. For example:

[ProjectDefinitions]

C:\AtUser\Project\Design

*C:\Project\Design

[LayoutEditor]

Net "Netname" is not being driven by any output pin [g200]

Example

Net "$1N1" is not being driven by any output pin

Description

All nets must be driven by at least one output pin.

Solution

Using the schematic, supply an output pin to the net.

Net "Netname" listed in route file not found in the design [g201]

Example

Net "$1N1" listed in route file not found in the design

Description

The detail and global routers both allow the user to specify a file (using the /r switch) which contains the names of nets to be routed. If any of the names in the list are not found in the design, this message is displayed.

Solution

Be sure the specified name is correct and includes the necessary hierarchy information. The /n option on the detail router can be used to get a list of the net names known to the routing programs.

No help available on this topic: "text" [g202]

Example

No help available on this topic: " E x i t "

Description

The program could not locate help for the specified topic.

Solution

Contact Customer Service.

No pins found for element "element" in parent "component" [g203]

Example

No pins found for element "$1I1" in parent "TOP_LEVEL_DESIGN"

Description

This message appears when a component instance or primitive in the database is found to have no pins.

Solution

Find the specified instance in the schematic and attach at least one net to it.

Net "Netname" is already a Nettype net and cannot be a Nettype net for "instance" pin "Pinname" [g207]

Example

Net "$1N1" is already a CLOCK net and cannot be a RESET net for "$1I1" pin "A"

Description

This is an internal program error.

Solution

Contact Customer Service.

No valid net names found in the route list file [g208]

Description

This message is issued by the router when the "/r" command-line option is used and none of the nets specified in the file are found in the design. The program will exit in absence of the required information.

Solution

A check should be made to ensure that the net names are spelled correctly. Also, the file is case sensitive so the case must match that used during design entry.

Package library file was not specified or is incorrect. It is required to initialize an internal map [g220]

Description

If no package file is specified or the specified package is incorrect, the program will not be able to start running, and this message is displayed.

Solution

This message is accompanied by other error messages. Use the solutions for these messages before re-running the program.

Expected a value greater than or equal to zero. Found "text" [g221]

Example

Expected a value greater than or equal to zero. Found "-1"

Description

This message is displayed when a program is attempting to read a positive integer but encounters a string representing a negative integer.

Solution

If this message appears at the very beginning of a program, it is probably due to an incorrectly specified command-line option. The user should run the program with no command-line arguments to see the proper syntax for that programs options. If this problem occurs later in a program run, the source of the error will be a part of the message.

Pin "Pinname" blocked on "instance" at (Col,Row) [g223]

Example

Pin "?" blocked on "$E1" at (10,55)

Description

This message is displayed when the router cannot route an IO, clock or reset element. The coordinate displayed is the cell nearest the element.

Solution

Use the interactive editor to check for routing conflicts. Rearrange the IO pins or the registers that connect to the column clock and reset to eliminate any conflicts.

Press <Pause> to suspend program. To resume press any key [g225]

Description

Pressing the <Pause> key temporarily stops the program . Press any key (except the <Pause> and shift keys) to resume the process.

Solution

No solution is required

Printer out of paper [g227]

Description

The printer is out of paper.

Solution

Put paper in printer.

Program runs on netlist-driven designs. Current design is not netlist-driven [g231]

Description

This message appears when a program that can only be run on netlist-driven designs is asked to run on a non-netlist-driven design.

Solution

If a design is created using the interactive editor or has been modified using the interactive editor, it is no longer possible to run programs that only work with a netlist-driven design.

Program creates more than one output file. Cannot accept an extension on "filename" [g232]

Example

Program creates more than one output file. Cannot accept an extension on "design.lyo"

Description

This message is displayed when an output file extension normally provided by the program was specified in the command-line option.

Solution

Run the program again without specifying the file name extension. For example, if NetOut is run in the following manner it will result in an error.

NetOut -o Design.xyz

Now if it is run the correct way as in:

NetOut -o Design

NetOut will generate the outputs Design.pyo, Design.sig, and Design.1.

Pin "Pinname" appears on line number of the netlist file, but is not defined in the library definition of this element [g233]

Example

Pin 'Q0' appears on line 217 of the netlist file, but is not defined in the library definition of this element

Description

Part of the information read from the timing library file is a list of the names of the pins on each cell type defined in the library. RzLvl has read an instance definition of one of the cell types in the timing library, and found that the instance definition specifies the name of a pin that was not defined in the timing library.

Solution

Attempt to regenerate the file, or contact Customer Service.

Program reads more than one input file. Cannot accept an extension on "filename" [g235]

Example

Program reads more than one input file. Cannot accept an extension on "design.cdb"

Description

This message is displayed when an input file extension normally provided by the program was specified in the command-line option.

Solution

Run the program again without specifying the file name extension. For example, if AtIdc is run in the following manner it will result in an error.

AtIdc -i *Design*.cdb

Now if it is run the correct way as in:

AtIdc -i Design

AtIdc **will read the inputs Design.pdc, and Design.cdb.**

Remove write protection for "filename" [g261]

Example

Remove write protection for "C:\AtUser\Project\Design"

Description

If the program cannot access a directory, this message appears.

Solution

First, make sure the program is allowed to write to the directory, then change its protection to allow write permission.

Routing of net "Netname" incomplete [g265]

Example

Routing of net "$1N1" incomplete

Description

Routing of the specified net is not complete.

Solution

Use the Interactive Editor to either route the net or modify placement and re-run the router to complete the routing.

Removed the routing for partially routed net "Netname" [g266]

Example

Removed the routing for partially routed net "$1N1"

Description

If a net is partially routed and the automatic placement program is started, it will remove the net routing.

Solution

If the routing should not be removed it must be completed using automatic or manual routing before placement is invoked.

Removed the dangling wires for net "Netname" [g267]

Example

Removed the dangling wires for net "$1N1"

Description

This message is given by the Incremental Design Change program to inform the user that the routing for the changed design is different from the old layout.

Solution

A check should be made to ensure that the specified nets have changed or the instances that they connect to have changed. In most cases the removal of the dangling wires is desirable. If this is not the case the "/k" command-line option can be used to keep dangling wires.

Syntax error in command-line. Found extra number of arguments [g283]

Description

The program found too many arguments in the command-line.

Solution

Type in the program name to get the usage statement or refer to the Command Reference section of the *Technical Reference & Release Notes*.

Syntax error in command-line. Found extra number of options [g284]

Description

The program found too many options in the command-line.

Solution

Type in the program name to get the usage statement or refer to the Command Reference section of the *Technical Reference & Release Notes*.

Syntax error in command-line. Insufficient number of arguments [g285]

Description

The program expected more command-line arguments to be entered.

Solution

Type in the program name to get the usage statement or refer to the Command Reference section of the *Technical Reference & Release Notes*.

Syntax error in command-line. Insufficient number of options [g286]

Description

The program expected more command-line options to be entered.

Solution

Type in the program name to get the usage statement or refer to the Command Reference section of the *Technical Reference & Release Notes*.

Syntax error in command-line. Found an unrecognized argument "argument" [g287]

Description

The program found an unrecognized argument in the command-line.

Solution

Type in the program name to get the usage statement or refer to the Command Reference section of the *Technical Reference & Release Notes*.

Syntax error in command-line. Found an unrecognized option "option" [g288]

Description

The program found an unrecognized option in the command-line.

Solution

Type in the program name to get the usage statement or refer to the Command Reference section of the *Technical Reference & Release Notes*.

Syntax error in command-line. Found an unrecognized keyword "keyword" [g289]

Description

The program found an unrecognized keyword in the command-line.

Solution

Type in the program name to get the usage statement or refer to the Command Reference section of the *Technical Reference & Release Notes*.

Syntax error in command-line. Option "/text" requires an argument [g290]

Description

The program expecting an argument to follow the option in the command line.

Solution

Type in the program name to get the usage statement or refer to the Command Reference section of the *Technical Reference*.

Syntax error in command-line. Option "/text" cannot be used with option "/text" [g291]

Example

Syntax error in command-line. Option /c cannot be used with option /t

Description

AtNet will only generate one type of output whenever it is run. This is either the Atmel database (*.cdb), the AtLvs logical netlist (*.lvs), or the Timing analysis logical netlist (*.lyo).

Solution

Do not specify both the /c and /t options at the same time. If both types of output are required, AtNet should be run twice, once with each option.

Syntax error in command-line. Option "/text" requires option "/text" to be specified [g292]

Example

Syntax error in command-line. The /i option requires the /c option to be specified

Description

The /i option is used to specify the name of the input file or files for an IDS program. AtNet gets its input only from the schematic except when the /c option (for layout versus schematic check) is specified. In this case it gets input from both the schematic and the layout. This is the only time the /i option can be used.

Solution

Specify the /i option when the /c option is being used. If a different schematic is to be loaded, the appropriate changes to the viewdraw.ini and atmel.ini file must be made.

Syntax error found in file "filename" [g293]

Description

The program found a syntax error in the specified file.

Solution

Determine which program created the specified file and check the listing file to see if there were any errors listed. Correct any errors before attempting to proceed. If there were no errors, run the program again to see if it still generates a corrupted file. If it does, contact Customer Service for further assistance.

Syntax error found in file "filename". Cannot find attribute "attribute" [g294]

Description

The program cannot find the specified attribute in the file.

Solution

Determine which program created the specified file and check the listing file to see if there were any errors listed. Correct any errors before attempting to proceed. If there were no errors, run the program again to see if it still generates a corrupted file. If it does, contact Customer Service for further assistance.

Syntax error found in file "filename". Cannot find keyword "keyword" [g295]

Description

The program cannot find the specified keyword in the file.

Solution

Determine which program created the specified file and check the listing file to see if there were any errors listed. Correct any errors before attempting to proceed. If there were no errors, run the program again to see if it still generates a corrupted file. If it does, contact Customer Service for further assistance.

Syntax error found in file "filename". Cannot find table "text" [g296]

Description

The program cannot find the specified table in the file.

Solution

Determine which program created the specified file and check the listing file to see if there were any errors listed. Correct any errors before attempting to proceed. If there were no errors, run the program again to see if it still generates a corrupt file. If it does, contact Customer Service for further assistance.

Syntax error found in file "filename". Unexpected value "value" found in line number: "number" [g297]

Description

The program found an unexpected value in the specified line of the file.

Solution

Determine which program created the specified file and check the listing file to see if there were any errors listed. Correct any errors before attempting to proceed. If there were no errors, run the program again to see if it still generates a corrupted file. If it does, contact Customer Service for further assistance.

Syntax error found in file "filename". Unexpected value "value" found for keyword "keyword" [g298]

Description

The program found unexpected value for the specified keyword in the file.

Solution

Determine which program created the specified file and check the listing file to see if there were any errors listed. Correct any errors before attempting to proceed. If there were no errors, run the program again to see if it still generates a corrupted file. If it does, contact Customer Service for further assistance.

Syntax error found in file "filename". Unexpected value "value" found in table "text" [g299]

Description

The program found unexpected value in the specified table.

Solution

Determine which program created the specified file and check the listing file to see if there were any errors listed. Correct any errors before attempting to proceed. If there were no errors, run the program again to see if it still generates a corrupted file. If it does, contact Customer Service for further assistance.

Syntax error found in file "filename". Unexpected value "value" found in ASCII file [g300]

Description

The program found unexpected non-ASCII data in the specified ASCII file.

Solution

Determine which program created the specified file and check the listing file to see if there were any errors listed. Correct any errors before attempting to proceed. If there were no errors, run the program again to see if it still generates a corrupted file. If it does, contact Customer Service for further assistance.

Syntax error found in file "filename". Unexpected value "value" found for attribute "attribute" [g301]

Description

The program found an unexpected value for the specified attribute.

Solution

Determine which program created the specified file and check the listing file to see if there were any errors listed. Correct any errors before attempting to proceed. If there were no errors, run the program again to see if it still generates a corrupted file. If it does, contact Customer Service for further assistance.

Syntax error found in file "filename". Unexpected value "value" found [g302]

Description

The program found unexpected value in the specified file.

Solution

Determine which program created the specified file and check the listing file to see if there were any errors listed. Correct any errors before attempting to proceed. If there were no errors, run the program again to see if it still generates a corrupted file. If it does, contact Customer Service for further assistance.

Syntax error found in file "filename". Found an unrecognized keyword "keyword" [g303]

Description

The program found an unrecognized keyword in the specified file.

Solution

Make sure it is not a misspelled keyword before removing it.

Syntax error found in the file "filename". Cannot find "=" after keyword "keyword" [g305]

Description

The user initialization and design configuration files are composed of various sections introduced by section headers of the form "[SectionHeader]". Keywords under each section set various values for the program to use. To set the value for some keywords, an '=' character is required. If the '=' is missing, this message appears.

Solution

Look at the atmle.ini or design.cfg file that contains the keyword in question. It should be followed by an '=' character and some value. Spacing is not important. For the proper syntax, refer to the example atmel.ini and *design*.cfg files in the C:\Atmel\Etc directory.

Syntax error found in file "filename". Unexpected number of fields number, expecting number, in line number: number [g306]

Description

This message is displayed when an error is found in reading the design database (*.cdb) or layout library. The name of the input file will be given in a "Processing file" statement before this message is displayed.

Solution

Determine which program created the design database and look at the listing file to see if there were any errors listed. Remove any errors before attempting to proceed. If there are no errors, run the program to see if it still generates a corrupt file. If it does, contact Customer Service.

Syntax error found in file "filename". Unexpected keyword "text" conflicts with keyword "text" [g307]

Description

This message is displayed when an error is found reading the specified file. Certain keywords in the file cannot be used in conjunction with other keywords.

Solution

Examine the specified file and determine which of the two keywords should actually be used and delete the other.

Syntax error found in file "filename". Exceeded maximum number of characters allowed in line number: number [g310]

Description

There is a syntax error in the file. The designated line is too long to be properly read by the program. Over-length lines should not be generated by programs in the software system.

Solution

Attempt to regenerate the file, or contact Customer Service.

The license for this product has expired [g323]

Description

The application license expiration date has been reached.

Solution

Contact Customer Service.

The maintenance for this product has expired [g324]

Description

The application maintenance expiration date has been reached.

Solution

Contact Customer Service.

The "permit.id" file contains hostid "number", conflicts with the block [g325]

Example

The "permit.id" file contains hostid "123456a7", conflicts with the block

Description

The permit contains a different host identification number than that found on the security block.

Solution

Make sure the permit.id file is correct for the security block being used. If the problem persists, contact Customer Service.

The "permit.id" file contains unexpected information [g326]

Description

The program found invalid data in the permit file.

Solution

Contact Customer Service.

The "permit.id" file is not valid for this version of the product [g327]

Description

The permit file does not have the correct version number for the application.

Solution

Contact Customer Service.

The pins "D" and "SI" on component "SRPST" may need to be swapped [g328]

Description

This message is displayed when a design database (*.cdb) is converted from a prior release to the 1.00 release. There is an error in the layout of the SRPST component that swapped the D and SI pins.

Solution

If the SRPST component has not been placed yet, nothing needs to be done. If the design has already been placed or routed, bring the design up in the interactive editor. Identify, select and delete SRPST. The affected components need to be replaced and then re-connected as needed.

The product license will expire on mm/dd/yy [g329]

Example

The product license will expire on 04/30/92

Description

The application's license will expire on the specified date.

Solution

No solution is required.

The text file contains errors. Attempting to run with default settings [g330]

Example

The user initialization file contains errors. Attempting to run with default settings

The design configuration file contains errors. Attempting to run with default settings

Description

If any errors are encountered while processing the user initialization file or the design configuration file. This message is displayed as a summary. All defaults used are displayed in the list and log files.

Solution

Refer to the message reference for each error reported to determine the appropriate solution.

This product is not licensed according to the "permit.id" file [g335]

Description

The application has not been licensed to operate.

Solution

Contact Customer Service.

This program must be run from the Design Manager, not as a stand-alone program [g331]

Description

The invoked program can only be run from within the Design Manager.

Solution

Return to the Design Manager to run this program.

This routine is not available to you. Please contact your sales representative. [g337]

Description

The license for this program has expired or was not purchased.

Solution

Contact Customer Service for licensing information.

TIMING has modified Pathfind command file: filename [g338]

Example

TIMING has modified Pathfind command file: Design.pcf

Description

The program has commented out the PRINT attribute definition in the Pathfind command file specified. This option can not be defined during execution of Timing.

Solution

No solution is required.

Top level element not found in design "text" [g339]

Example

Top level element not found in design "Design"

Description

This message is displayed when a program cannot find the top-level element (an internal name for the root record in the design database) for a design or when the program has used an incorrect file as input.

Solution

Check to see that the correct file was specified as input.

Determine which program created the design database and look at the listing file to see if there were any errors listed. Remove any errors before attempting to proceed. If there are no errors, run the program to see if it still generates a corrupt file. If it does, contact Customer Service for further assistance.

Text value "text" should be an integer [g340]

Example

SHAPE attribute on component "$111" value "XYZ" should be an integer.

Description

This message appears when the SHAPE attribute contains a character other than 0-9, + or -.

Solution

Change specified value into a legal integer.

The demonstration version has limited capabilities [g344]

Description

The application has been set up for demonstration only and is not fully functional.

Solution

Contact Customer Service.

There is no library information for element "element" [g348]

Example

There is no library information for element "C4P_2"

Description

Instances of the cell type listed appear in the *.cyo or *.lvs file, but no description of this cell type appears in the timing library file.

Solution

This is only a warning message. The program continues and tries to match layout to schematic.

Two cells have the same name "instance" [g349]

Example

Two cells have the same name "C21\M304"

Description

While reading a netlist file, the program found two cells with the specified name. This is a serious error in the netlist data and the program terminated.

Solution

Attempt to regenerate the file, or contact Customer Service.

Unable to create file: filename [g350]

Example

Unable to create file: Design.bsl

Description

The program could not create the specified file name.

Solution

Make sure there is sufficient room on the current drive. Delete any unnecessary files. If on a network, make sure the access rights to the current location are set correctly.

Unable to execute DMSETUP [g351]

Description

The program could not run the Design Manager Setup program.

Solution

Check the DOS PATH environmental variable for the process cited.

Unexpected value found for the time or date, exceeds two digits [g352]

Description

The program found illegal characters in the permit file.

Solution

Contact Customer Service.

Unknown media type [g353]

Description

This is an internal program error.

Solution

Contact Customer Service.

The filename file needs updating. Change keyword "string" to "string" [g354]

Example

The design configuration file needs updating. Change keyword "UserLibrary1" to "UserLibrary2".

Description

Between release 1.0 and 1.1 a change has been made to some of the keywords in the user initialization and design configuration files. The program will still recognize the old keyword but will issue this message so that the user will update the requested file.

Solution

The user should update the requested file by changing the specified keyword to the string specified in the message. This should be done so the correct information can be provided to the program when the version 1.0 keywords are no longer supported.

Unexpected end-of-file (EOF) found in "filename" [g355]

Example

Unexpected end-of-file (EOF) found in "filename"

Description

This message is displayed if the program has unexpectedly reached the end of the file it was reading.

Solution

Determine if the correct file has been specified to the program being run. Also, make sure that the program that created this file has completed without error. Re-specify or rerun the programs as needed.

Write file error [g370]

Description

The program could not write information to the file.

Solution

Make sure there is sufficient room on the current drive. Delete any unnecessary files. If on a network, make sure the access rights to the current location are set correctly.

"Number" Kbytes EPROM is inadequate to hold cascaded bit stream [d046]

Example

"2" Kbytes EPROM is inadequate to hold cascaded bit stream

Description

This message is displayed if the specified EPROM size is smaller than the sum of the cascaded files.

Solution

Specify a larger EPROM size. It should be at least as big as the sum of each of the individual files that are being cascaded.

Not possible to get from pin "Pinname" on primitive "Instance" at (col,row) [d047]

Example

Not possible to get from pin "A" on primitive "E10" at (5,5)

Description

This message is only displayed by component generators. It signals that there is an error inside the generator.

Solution

Contact Customer Service.

No net was specified to connect to "Component" pin "Pinname" "Pintype" [d048]

Example

No net was specified to connect to PINV pin L Input

Description

This message is only displayed by component generators. It signals that there is an error inside the generator.

Solution

Contact Customer Service.

Cannot assign multiple directions to input pin "Pinname" because it is not a tri-state pin [d049]

Example

Cannot assign multiple directions to input pin "A" because it is not a tri-state pin

Description

This message is only displayed by component generators. It signals that there is an error inside the generator.

Solution

Contact Customer Service.

No core cell placed at location (col,row) [d050]

Example

No core cell placed at location (5,7)

Description

This message is only displayed by component generators. It signals that there is an error inside the generator.

Solution

Contact Customer Service.

Cannot use both the A and the L input for a wire or xwire cell [d051]

Description

This message is only displayed by component generators. It signals that there is an error inside the generator.

Solution

Contact Customer Service.

Cannot connect two Pintype pins together for pins "Pinname" and "Pinname" [d052]

Example

Cannot connect two Input pins together for pins "A" and "B"

Description

This message is only displayed by component generators. It signals that there is an error inside the generator.

Solution

Contact Customer Service.

Core cell at location (col,row) not connected to anything yet [d053]

Example

Core cell at location (10,15) not connected to anything yet

Description

This message is only displayed by component generators. It signals that there is an error inside the generator.

Solution

Contact Customer Service.

Primitive element "Element" does not have a corresponding component. Schematic will be incomplete [d054]

Example

Primitive element "PAN2S" does not have a corresponding component. Schematic will be incomplete

Description

This message is displayed when a program is attempting to create an input file for automatic schematic generation. It will convert all primitives used in the layout to their corresponding component. Some primitives in the Atmel library do not have a corresponding component.

Solution

If possible, change the primitive in the layout to another one that has a corresponding component. Otherwise, contact Customer Service.

Symbol "Component" not found in the macro library [d055]

Example

Symbol "AN2L" not found in the macro library

Description

In order to generate a Viewlogic compatible netlist, the program must be able to locate the specified symbol for the component in the Atmel Viewlogic library. Library paths contained in the Viewlogic initialization file, viewdraw.ini, are used to search for symbols.

Solution

Check that the viewdraw.ini file located in the project directory, or one of the directories pointed to by the WDIR environment variable, contains the correct paths to both IDS symbol libraries, CLiMacro and CLiPrim.

Errors in processing symbol. Schematic wire file will be incorrect [d056]

Description

In order to generate a Viewlogic compatible netlist, the program must be able to match data on the symbol with data from the Atmel layout library. If the data does not match, this message is displayed. This is typically caused by pin names not matching.

Solution

Contact Customer Service.

AtErc

Standard variables in Italics for this program are as follows:

Bus: one of NBUS, SBUS, EBUS, or WBUS to specify which type of bus element has a problem.

Count: is used if a count is performed.

(Col,Row): is used to indicate a location in the array.

Instance: is the name of an instance.

Filename: is a complete or simple file name.

Netname: is the name of a net being used in the design.

Number: is for numeric values that are not used as a count.

Pinname: is for names of pins.

PinType: is either Input or Output.

Text: is a string that can be alphanumeric.

Type: used to specify the lowest level primitive type of an element in the database.

Program Messages

"Type" illegally has a "B" pin [c200]

Example

"ZW10" illegally has a "B" pin

Description

The component has an illegal B connection.

Solution

Remove the B pin.

"Type" illegally has a "L" pin [c203]

Example

"ZW10" illegally has a "L" pin

Description

The component has an illegal L connection.

Solution

Remove the L pin.

"Type" illegally has an "A" pin [c204]

Example

"ZW10" illegally has an "A" pin

Description

The component has an illegal A connection.

Solution

Remove the A pin.

"Type" illegally has an input Pinname with a straight turn [c205]

Example

"ZW10" illegally has an input A pin with a straight turn

Description

A straight turn (north to south, south to north, east to west or west to east) is being defined with an A input.

Solution

Either the turn or the A input has to be removed.

"Type" is missing a required "L" pin [c206]

Example

"ZW10" is missing a required "L" pin

Description

The component needs a L connection.

Solution

Connect the L pin.

"Type" is missing the required "A" input pin [c207]

Example

"ZW10" is missing the required A input pin

Description

The component needs an A connection.

Solution

Connect the A pin.

"Type" is missing the required "B" input pin [c208]

Example

"ZW10" is missing the required "B" pin

Description

The component needs a B connection.

Solution

Connect the B pin.

Bus "PinType" is not connected at (Col,Row): bus [c211]

Example

Bus "INPUT" is not connected at (55,23): NBUS

Description

A bus IO is not connected.

Solution

Make the necessary connections or remove the bus.

Core cell "instance" on fringe (Col,Row) has an illegal text [c215]

Example

Core cell "$E3466" on fringe (55,28) has an illegal B output NORTH

Description

The specified cell connection falls outside the array.

Solution

Modify the instance to make a legal cell connection.

Cross wire cell "instance" is wired incorrectly at (Col,Row) [c216]
Text

Example

Cross wire cell "$E3466" is wired incorrectly at (55,28)
B in does not connect to A out

Description

The cross wire is connected incorrectly.

Solution

A cross wire cell should have an 'A' or an 'L' input hooked to a 'B' output and a 'B' input hooked to an 'A' output. Modify the instance to make a legal cross wire cell connection.

Driving a bus a second time with "instance" at (Col,Row) [c217]

Example

Driving a bus a second time with "$E3466" at (55,28)

Description

The bus specified has two drivers.

Solution

Use only one driver for the bus.

Driving a tri-state bus with a non tri-state core cell in "instance" at (Col,Row) [c218]

Example

Driving a tri-state bus with a non tri-state core cell in "$E3466" at (55,28)

Description

A tri-state bus is being driven by a core cell that is not defined as a tri-state.

Solution

Either replace the cell with a tri-state or remove the connection to the bus.

Pin "Pinname" in "instance" drives more than number type output(s) at (Col,Row) [c219]

Example

Pin "A" in "$1I1" drives more than 1 BUS output(s) at (5,5)

Description

The maximum number of outputs has been exceed.

Solution

Reduce the number of outputs.

Illegal jumper for a "bus" is defined in "instance" at (Col,Row) [c223]

Example

Illegal jumper for a "NBUS" is defined in "$E3466" at (55,23)

Description

A north or west local bus is defined with a jumper.

Solution

Only south and east buses maybe defined with a jumper. Remove the bus and retry using a repeater.

Illegal turn with input "text" and output "text" in "instance" at (Col,Row) [c226]

Example

Illegal turn with input "?" and output "?" in "$E3466" at (55,28)

Description

There is an illegal turn in the instance specified.

Solution

Remove the turn and all bus connections associated with it.

Local bus is illegally defined as both input and output in "instance" at (Col,Row) [c227]

Example

Local bus is illegally defined as both input and output in "$E3466" at (55,28)

Description

The instance specified has a local bus input and output defined.

Solution

Make the cell either input or output to the bus.

Illegal use of both tri-state output and normal output to bus in same cell at (Col,Row) [c230]

Example

Illegal use of both tri-state output and normal output to bus in same cell at (5,5)

Description

If a cell has a tri-state output it cannot use the A output pin to connect to the bus.

Solution

Disconnect the A output from the bus.

Multiple "Pinname" PinType pins specified for "instance" at (Col,Row) [c231]

Example

Multiple "A" input pins specified for "$E3466" at (55,28)

Description

More than one input is being specified for the same location.

Solution

Change the instance input pin to a valid CDB value. Report this problem to Customer Service.

Net "Netname" does not have any input pins [c234]

Example

Net "D0" does not have any input pins

Description

The net should have an input otherwise it should probably not be in the design. Unused outputs on components typically cause this situation.

Solution

If this is caused by an unused component output, this message can be ignored.

Net "Netname" is not being driven by any output or input pin [c235]

Example

Net "D0" is not being driven by any output or input pin

Description

The net needs an input or an output pin.

Solution

Add the necessary input or output pin to the schematic.

Net "Netname" is not being driven by any output pin [c236]

Example

Net "D0" is not being driven by any output pin

Description

The net needs an output driver.

Solution

Add the necessary output pin to the schematic.

Nets "Netname" and "Netname" do not match in "instance" at (Col,Row) [c237]

Example

Nets "D0" and "D1" do not match in "$E3466" at (55,28)

Description

The architecture does not allow a turn and a bus input or output at the same cell for different nets. If the nets are the same, then there must be a straight turn.

Solution

Either the turn or the bus connections must be removed.

Text cell "instance" has no pins [c241]

Example

Core cell "$E3466" has no pins

Description

The instance specified is not connected.

Solution

Make necessary connections.

Text pin "Pinname" is not connected in "instance" at (Col,Row) [c242]

Example

Output pin "A" is not connected in "$E3466" at (55,23)

Description

An IO pin is not connected.

Solution

The necessary connections.

Wire cell "instance" is wired incorrectly at (Col,Row) Text [c245]

Example

Wire cell "$E3466" is wired incorrectly at (55,28)
B in does not connect to B out

Description

The instance specified is wired incorrectly.

Solution

A wire cell should have an 'A' input hooked to an 'A' or 'L' output and a 'B' input hooked to a 'B' output. Modify the instance to make a legal wire cell connection.

Illegal use of text pair to connect to cell and turn at (Col,Row) [c246]

Example

Illegal use of North/West pair to connect to cell and turn at (5,5)

Description

Turns from one local bus to another can only be made at the North/West or South/East corners ("turn dots") of each core cell. A cell cannot read or write to the pair of buses associated with a "turn dot" once that connector is activated.

Solution

The illegal cell should be located and either have the A output to the bus, L input from the bus, or turn removed. This problem should be reported to Customer Service.

Illegal use of text pair for input and output to bus at (Col,Row) [c247]

Example

Illegal use of North/West pair for input and output to bus at (5,5)

Description

Turns from one local bus to another can only be made at the North/West or South/East corners ("turn dots") of each core cell. The chip architecture will not allow simultaneous input and output to a cell from buses connected to the same turn dot. An input from a North or West bus can only be output to a South or East bus, or vice versa.

Solution

The illegal cell should be located and the A output to the bus or the L input from the bus removed. This problem should be reported to Customer Service.

Illegal use of text turn to connect to the text pair at (Col,Row) [c248]

Example

Illegal use of North/West turn to connect to the South/East pair at (5,5)

Description

Turns from one local bus to another can only be made at the North/West or South/East corners ("turn dots") of each core cell. Turns between either the North and East, or South and West buses around a cell are prohibited.

Solution

The illegal turn should be located and disconnected. It can either be routed through the cell or re-routed to use a different pair of buses. This problem should be reported to Customer Service.

Type bus is being driven from 2 pins at (Col,Row): bus [c249]

Example

Express bus is being driven from 2 pins at (0,5): NBUS

Description

The express buses along the perimeter of the array are driven by the core cells in every other row or column. It is illegal to drive these express buses with a repeater. This message will report that bus and its anchor location that is being driven on both ends.

Solution

The express bus should be disconnected from the repeater and re-routed. This problem should be reported to Customer Service.

Bus at (Col,Row) has only pintype pins [c250]

Example

WBUS at (0,0) has only output pins

Description

It is an electrical violation to have only input or output connections to a bus. Under such circumstances, the data is being written and not read or vice a versa.

Solution

Connect the input or output of the bus as required. Otherwise remove the bus from the design if it is not needed.

Bstr

Standard variables in Italics for this program are as follows:

(Col,Row): used to indicate a location in the array

Filename: the name of a file can be complete (all directory information specified) or simple (no directory information specified)

Instance: the name of an instance

Number: numeric values that are not used as counts

Pinname: the name of a pin either physical or logical

Text: any miscellaneous string that may be output from a program

Type: used to specify the lowest level primitive type of an element in the database

Program Messages

"Type" is missing the required "B" bit [b100]

Example

"ZW10" is missing the required "B" bit

Description

The type listed is missing a required "B" connection.

Solution

Add the "B" connection.

"Type" illegally has a "B" bit [b101]

Example

"ZW10" illegally has a "B" bit

Description

The type listed has an illegal "B" connection.

Solution

Remove the "B" connection.

"Type" illegally has a "L" bit [b104]

Example

"ZW10" illegally has a "L" bit

Description

The type listed has an illegal "L" connection.

Solution

Remove the "L" connection.

"Type" illegally has an "A" bit [b105]

Example

"ZW10" illegally has an "A" bit

Description

Remove the "A" connection.

Solution

Please modify the instance specified to disconnect the specified connection.

"Type" is missing a required "L" bit [b106]

Example

"ZW10" is missing a required "L" bit

Description

The type listed is missing a required "L" connection.

Solution

Add the required "L" connection.

"Type" is missing the required "A" bit [b107]

Example

"ZW10" is missing the required "A" bit

Description

The type listed is missing a required "A" connection.

Solution

Add the required "A" connection.

Multiple "Pinname" input pins specified for "instance" at (Col,Row) [b121]

Example

Multiple "A" input pins specified for "$E3466" at (55,28)

Description

More than one input has been specified for the instance at the designated coordinates.

Solution

Make sure the instance input pin is a valid IDS database (*.cdb) value.

Reprocessing bit stream for programming mode number [b124]

Example

Reprocessing bit stream for programming mode 2

Description

The program is reprocessing the bit stream file according to the specified mode.

Solution

No solution is required.

Respecify configuration register to be within 0 and 255 [b127]

Example

Respecify configuration register to be within 0 and 255

Description

The configuration register specified is not within the defined range.

Solution

Restart the program with the configuration register set within the defined range.

Respecify DMA address to be within 0 and 131071 [b128]

Description

The specified DMA address is not within the defined range.

Solution

Restart the program with DMA address set within the defined range.

Respecify programming mode to be within 1 and 6 [b129]

Description

The programming mode specified is not within the defined range.

Solution

Restart the program with mode set within the defined range.

Type cell "instance" has no pins [b130]

Example

Core cell "$E3466" has no pins

Description

The specified type cell for the instance specified isn't connected.

Solution

Make the necessary connections.

Expected hex character versus 0xnumber encountered [b133]

Example

Expected hex character versus 0x09 encountered

Description

The program was expecting a legal hexadecimal number (0-9, A-F) and encountered an illegal character.

Solution

Make sure all specified hex numbers contain legal hex digits (0-9, A-F). This problem can only occur in mode 2 when generating hex format bit stream files.

Illegal use of both tri-state output and normal output to bus in same cell at (Col,Row) [b134]

Example

Illegal use of both tri-state output and normal output to bus in same cell at (10,5)

Description

There should not be both a tri-state output and an A output to a bus in the same cell.

Solution

The illegal cell should be located and the A output to the bus removed. This problem should be reported to Customer Service.

Illegal use of text pair to connect to cell and turn at (Col,Row) [b135]

Example

Illegal use of North/West pair to connect to cell and turn at (5,5)

Description

Turns from one local bus to another can only be made at the North/West or South/East corners ("turn dots") of each core cell. A cell cannot read or write to the pair of buses associated with a "turn dot" once that connector is activated.

Solution

The illegal cell should be located and either the A output to the bus, L input from the bus, or turn removed. This problem should be reported to Customer Service.

Illegal use of text pair for input and output to bus at (Col,Row) [b136]

Example

Illegal use of North/West pair for input and output to bus at (5,5)

Description

Turns from one local bus to another can only be made at the North/West or South/East corners ("turn dots") of each core cell. The chip architecture will not allow simultaneous input and output to a cell from buses connected to the same turn dot. An input from a North or West bus can only be output to a South or East bus, or vice versa.

Solution

The illegal cell should be located and the A output to the bus or the L input from the bus removed. This problem should be reported to Customer Service.

Illegal use of text turn to connect to the text pair at (Col,Row) [b137]

Example

Illegal use of North/West turn to connect to the South/East pair at (5,5)

Description

Turns from one local bus to another can only be made at the North/West or South/East corners ("turn dots") of each core cell. Turns between either the North and East, or South and West buses around a cell are prohibited.

Solution

The illegal turn should be located and disconnected. It can either be routed through the cell or re-routed to use a different pair of buses. This problem should be reported to Customer Service.

Pin "Pinname "in "Instance" drives more than number bus output(s) at (Col,Row) [b138]

Example

Pin "A" in "$1I1" drives more than 1 bus output(s) at (5,5)

Description

The chip architecture allows only a single bus output for each core cell.

Solution

The specified cell should be found and the extra bus outputs removed. They should be re-routed using neighboring wire cells or turns. This problem should be reported to Customer Service.

Bus "PinType" is not connected at (Col,Row) [b139]

Example

Bus "INPUT" is not connected at (5,8)

Description

The program has found a disconnected bus in the design. All connections must be complete for the design to work correctly.

Solution

The specified bus should be located and either removed or connected as needed.

BstrWin

Standard variables in Italics for this program are as follows:

Filename: is a complete or simple filename.

Number: numeric values that are not used as a count.

Text: any miscellaneous string that may be output from a program.

Program Messages

Bit stream output file and 'baseline' file sizes differ, unable to continue [b200]

Description

The bit stream output and base files sizes are not the same. Both files must be programmed under the same basic conditions (i.e. package and mode).

Solution

Check the input file names for proper data and retry.

File headers are different, use bit stream output file header [b201]

Description

The bit stream file headers are different. The header from the current bit stream output file will be used.

Solution

No solution is required.

Text file already windowed, unable to continue [b202]

Example

Bit stream output file already windowed, unable to continue

Description

The file is windowed. The program is unable to window a windowed file.

Solution

Check the input file names for proper data and retry.

No file name was specified for 'baseline' file. Compressing bit stream file [b203]

Description

A base file with standard bit stream data has not been named. The program will now attempt to compress the file into a maximum of 255 windows.

Solution

To prevent data compression, restore the design bit stream file and provide the program with a base file name.

Header number of windows does not match [b205]

Description

The number of windows specified in the header of the baseline file differs from that specified in the input bit stream file.

Solution

Check to be sure that the input files (bit stream and baseline) have been specified correctly.

Header registers different [b206]

Description

The register data specified in the header of the baseline file differs from that specified in the input bit stream file.

Solution

Verify that the input files (bit stream and baseline) have been specified correctly. Also check that register settings are identical for both the baseline and input bit stream files.

Header DMA address different [b207]

Description

The DMA address specified in the header of the baseline file differs from that specified in the input bit stream file.

Solution

Verify that the input files (bit stream and baseline) have been specified correctly. Also check that the DMA address settings are identical for both the baseline and input bit stream files.

The addresses in window #number do not match [b208]

Example

The addresses in window #5 do not match

Description

The number of windows and the addresses within the windows should match between the baseline and the input bit stream files in order to generate a bit stream window output file.

Solution

Verify that the input files (bit stream and baseline) have been specified correctly. Also check that the settings used to generate both bit stream files are the same.

Cascade

Standard variables in Italics for this program are as follows:

Filename: the name of a file can be complete (all directory information specified) or simple (no directory information specified).

Program Messages

No file name was specified to cascade to [c002]

Description

The bit stream list file name has not been specified.

Solution

Use the Design Manager Cascade program to generate the bit stream list file.

The configuration register Cascade bit (B2) is clear but should be set. [c003] Set bit (Yes/No)?

Example

The configuration register Cascade bit (B2) is clear but should be set.
Set bit (Yes/No)?

Description

The second - least - significant bit, B2, should be set but is not.

Solution

If B2 should be set, answer 'Y'. If not, answer 'N'.

The configuration register Cascade bit (B2) is set but should be clear [c004]. Clear bit (Yes/No)?

Example

The configuration register Cascade bit (B2) is set but should be clear.
Clear bit (Yes/No)?

Description

The second - least - significant bit, B2, should be clear but is not.

Solution

If B2 should be cleared, answer 'Y'. If not, answer 'N'.

No file name was specified to sequence to [c006]

Description

A file containing the list of bit streams is not specified.

Solution

Specify a list file containing names of the bit streams.

Sequence requires first bit stream file be in mode 1, 2 or 5. Mode 3 or 4 found. Truncating preamble [c007]

Description

Sequencing requires the first bit stream to be in Mode 1, 2 or 5. If the first bit stream is found in Modes other than 1, 2 or 5, the sequencing is done by truncating additional preamble bytes found in the Mode 3 or 4 bit stream.

Solution

No special action is required.

Sequence requires first bit stream file be in mode 1, 2 or 5. Mode 6 found. Truncating preamble [c008]

Description

Sequencing requires the first bit stream to be in Mode 1, 2 or 5. If the first bit stream is found in Modes other than 1, 2 or 5, the sequencing is done by truncating additional preamble bytes found in the Mode 6 bit stream.

Solution

No action is required.

Bit stream "filename" is already cascaded and cannot be used. Regenerate this bit stream before running CASCADE [c009]

Example

Bit stream "*Design*.bst" is already cascaded and cannot be used. Regenerate this bit stream before running CASCADE

Description

This message is displayed when an illegal attempt is made to cascade an already cascaded bit stream.

Solution

Regenerate the bit stream and run Cascade program again.

CF

Standard variables in Italics for this program are as follows:

Address: is for numeric values that are used as a memory address.

Filename: is a complete or simple file name.

Number: is for numeric values that are not used as a count.

Text: is a string that can be alphanumeric.

Program Messages

ADDR sense_ack timed out [c800]

Description

This error occurs if an acknowledgment is not received when trying to send an address to the chip.

Solution

Try downloading the file again to see if the message still occurs. If so check to see that the board is plugged in correctly to the power supply and the PC. If problems persist, please contact Customer Service.

Syntax error in command-line [c801]

Description

The command-line options specified to the program are not correct.

Solution

Run the program without any command-line options to determine the correct usage. Make the appropriate changes and run again.

Input filename required [c802]

Description

The program was expecting an input file name to be specified on the command-line.

Solution

Run the program without any command-line options to determine the correct usage. Make the appropriate changes and run again.

Invalid polarity option specified [c803]

Description

The reset polarity must be specified as an H for high or an L for low. Any other characters will result in the above error.

Solution

Run the program without any command-line options to determine the correct usage. Make the appropriate changes and run again.

CMD ack_sense timed out [c804]

Description

This error occurs if an acknowledgment is not received when trying to send a command to the chip.

Solution

Try downloading the file again to see if the message still occurs. If so check to see that the board is plugged in correctly to the power supply and the PC. If problems persist, please contact Customer Service.

Input file "Filename" corrupted [c805]

Example

Input file "4bitalu.hex" corrupted

Description

The program has found inconsistent data in the input file.

Solution

Check to make sure the correct file name has been specified on the command-line. Also look at the file to determine if it is of the correct format. If problems persist, please contact Customer Service.

DATA ack_sense timed out for address: Address [c806]

Example

DATA ack_sense timed out for address: 1001

Description

This error occurs if an acknowledgment is not received when trying to send data to the chip.

Solution

Try downloading the file again to see if the message still occurs. If so check to see that the board is plugged in correctly to the power supply and the PC. If problems persist, please contact Customer Service.

Cannot open file "Filename" for reading [c807]

Example

Cannot open file "4bitalu.bst" for reading

Description

The program needs to load a file that contains the programming for the Serial Configuration Memory. If it does not find the file specified, this message will be displayed.

Solution

Run the program without any command-line options to determine the correct usage. Make the appropriate changes and run again.

Cannot reopen file "Filename" for reading [c808]

Example

Cannot reopen file "4bitalu.bst" for reading

Description

In order to verify that the specified data has been read into the chip correctly, the program will reopen the input file. If a problem occurs when trying to open the file again this message will be displayed.

Solution

Check to verify that the specified file was not accidentally deleted.

Expected Number but found Number at bit Address addr Address(Address) during verify [c809]

Example

Expected 0 but found 1 at bit 3 addr ff(255) during verify

Description

One of the options of the program is to verify that the data has been written correctly to the chip. If there is a bit mismatch between the input data and the data on the chip this message will be displayed.

Solution

Try downloading the file again to see if the message still occurs. If so check to see that the board is plugged in correctly to the power supply and the PC. If problems persist, please contact Customer Service.

Found Number incorrect bits during verify [c811]

Example

Found 20 incorrect bits during verify.

Description

This is a summary message of the total number of incorrect bits found during the verification phase of the program. Prior messages should indicate the exact errors.

Solution

Check to make sure that a) the power is on, and b) the chip and board are plugged in properly. Try downloading the bitstream again.

Cannot open file "Filename" for writing [c812]

Example

Cannot open file "4bitalu.bst" for writing

Description

The specified command-line option tells the program to create an output file for storing the data from the chip. If there is a problem with accessing and writing to this file, the above message will be displayed.

Solution

Run the program without any command-line options to determine the correct usage. Make the appropriate changes and run again. Also, check to make sure that the specified file does not already exist and is somehow protected.

Insufficient Memory Available [c813]

Description

DOS memory is allocated at run-time to hold programming, readback, and verification data. If there is insufficient base memory available in DOS for the current operation, the above message will be displayed.

Solution

Try to allocate more base memory in DOS. This can be achieved by removing unnecessary TSRs and device drivers.

Invalid Line (line) in hex file "Filename" [c814]

Example

Invalid Line (345) in hex file "4bitalu.hex"

Description

Each line of a hex format file applied as input to the program should consist of a record field following a specific structure. If any line in the input file does not follow this structure, the message above will be displayed.

Solution

Verify that the input file contains the necessary hex data in the correct format. Check to make sure the input file is not corrupted.

Checksum mismatch at line line in file "Filename" [c815]

Example

Checksum mismatch at line 245 in file "4bitalu.hex"

Description

Each line in a hex format file has a checksum appended. This checksum is used to validate the integrity of the line. If, when reading the contents of a hex format file, any line fails a checksum test, then the above message will be reported.

Solution

Check that the hex format file has not been corrupted.

Output filename required [c816]

Description

This message is displayed if an output filename, which is needed in the context of the other options that have been specified, is missing.

Solution

Run the program without any command-line options to determine the correct usage. Make the appropriate changes and run again.

Input file must be .POF, .HEX, or .RBF [c818]

Description

When programming or converting Altera files, the input file must have an extension of .POF, .HEX, or .RBF corresponding to the file format. The message given above will be displayed if the input file does not have the appropriate filename extension.

Solution

Ensure that the input file format is in one of the supported formats, and that the filename has an appropriate extension.

Downld

Standard variables in Italics for this program are as follows:

Filename: the name of a file can be complete (all directory information specified) or simple (no directory information specified).

Text: any miscellaneous string that may be output from a program.

Program Messages

The part was not correctly configured

The ERR pin was low at the end of configuration [u400]

Example

The part was not correctly configured

The ERR pin was low at the end of configuration

Description

The program detected an error from the download board..

Solution

Check board connections and retry

IDS

Standard variables in italics for this program are as follows:

(Col,Row): used to indicate a location in the array.

Component: the library type of the symbol or macro.

Design: the name of the design.

Directory: the name of a directory path.

Element: refers to an entity inside the design database, could be a component, instance, or primitive.

Filename: is a complete (all directory information specified) or simple (no directory information specified) file name.

Number: is for numeric values that are not used as a count.

Text: is a string that can be alphanumeric.

Macro Generators

"Component" already exists in Atmel Library and cannot be overwritten. Please choose a different Macro Name.

Example

"SR8" already exists in Atmel Library and cannot be overwritten. Please choose a different Macro Name.

Description

The Atmel Library contains a number of pre-defined macros. Figaro will select a macro in the Atmel library over any macro in the user library or design

Solution

Change the name so that it is not the same as an Atmel Library component.

Macro Name must be number characters or less.

Example

Macro Name must be 8 characters or less.

Description

The component name of the Macro Generators must be 8 characters or less on the PC platform, or 10 characters or less when used for Cadence Concept.

For PC users, the name is used as a file name and a directory, and must conform to the DOS limits of 8 characters. Cadence users need to meet the 10-character limit for Concept names.

Solution

Change the name to be 8/10 characters or less.

Macro Name "Component" does not make a legal file name.

Example

Macro Name "my macro" does not make a legal file name.

Description

The macro name provided is also used as the file and directory names for storing information about the component. As such, it must be a legal file name.

Solution

Change the name to conform to the operating systems requirements.

The following fields must have values before pressing Generate: Text.

Example

The following fields must have values before pressing Generate:

Width

Description

Certain fields of the *Macro Generators* dialog box must be entered before the macro can be created.

Solution

Enter values for the specified fields and then press *Generate*.

The following fields must have values before pressing Run: Text.

Example

The following fields must have values before pressing Run: Width.

Description

The listed fields are required before a macro generator can be run.

Solution

Supply values for the specified fields to the generator dialog box.

Generator file Text could not be found - unable to run generator

Example

Generator file adder.mgl could not be found - unable to run generator

Description

AT40K *Macro Generators* use the MGL files from the $FIGARO_HOME/mgl directory to create the specified hard macro.

Solution

Make sure the $FIGARO_HOME environment variable is pointing to the correct IDS installation tree. Also check that the MGL file reported in the error is available under the $FIGARO_HOME/mgl directory.

Error in Macro Generator program

Example

Error in Macro Generator program

Description

AT40K *Macro Generators* use the MGL files from $FIGARO_HOME/mgl directory to create the specified hard macro. This error indicates that the program failed due to a problem in the *Macro Generators* source file.

Solution

Contact Atmel technical support to get an updated *Macro Generators* source file.

Value for Text must be Text than Number.

Example

Value for Width must be greater than 1.

Description

Some of the parameters for the *Macro Generators* can only support a limited range of values. The interface checks for compliance and informs the user when this restriction has been violated.

Solution

Re-specify the value for the named field to be greater or less than the number specified by this message.

Design Entry, Simulation, and Synthesis Integration

ATMELDIR environment variable is pointing to a directory that does not exist: Directory.

Example

ATMELDIR environment variable is pointing to a directory that does not exist: /atmel/library.

Description

The ATMELDIR environment variable helps the IDS software determine where various libraries and related files are located. It should be set to the directory one level below where Figaro was installed, such as /atmel/etc.

Solution

Make sure that the location ATMELDIR points to is the directory specified above and that there are not misspellings.

Simulation integration is not currently supported for CAE System tool flow .

Example

Simulation integration is not currently supported for Everest-Verilog tool flow.

Description

The IDS provides full support for schematic capture and simulation for the Viewlogic, Mentor, and Cadence Concept platforms. Though provisions for libraries and netlist input for other systems are available, the Flow Bar buttons for schematic capture and simulation, etc. cannot be used on these partially supported platforms.

Solution

Check to make sure *Tools Flow* is specified correctly in the *File>Design Setup* dialog box.

Schematic integration is not currently supported for CAE System tool flow .

Example

Schematic integration is not currently supported for Everest-Verilog tool flow.

Description

The IDS provides full support for schematic capture and simulation for the Viewlogic, Mentor, and Cadence Concept platforms. Though provisions for libraries and netlist input for other systems are available, the Flow Bar buttons for schematic capture and simulation, etc. cannot be used on these partially supported platforms.

Solution

Check to make sure *Tools Flow* is specified correctly in the *File>Design Setup* dialog box.

Netlisting integration is not currently supported for CAE System tool flow .

Example

Netlisting integration is not currently supported for Everest-Verilog tool flow.

Description

The IDS provides full support for schematic capture and simulation for the Viewlogic, Mentor, and Cadence Concept platforms. Though provisions for libraries and netlist input for other systems are available, the Flow Bar buttons for netlisting cannot be used on these partially supported platforms.

Solution

Check to make sure *Tools Flow* is specified correctly in the *File>Design Setup* dialog box.

Synthesis integration is not currently supported for CAE System tool flow .

Example

Synthesis integration is not currently supported for OrCAD tool flow.

Description

The IDS provides full support for synthesis for the Viewlogic, Mentor, Synopsys, Everest, Exemplar and Cadence Synergy platforms. Though provisions for libraries and netlist input for other systems are available, the Flow Bar buttons for invoking the synthesis tool cannot be used on these partially supported platforms.

Solution

Check to make sure *Tools Flow* is specified correctly in the *File>Design Setup* dialog box.

Design must be run through Initial Placement before Post Mapping Simulation can be run.

Example

Design must be run through Initial Placement before Post Mapping Simulation can be run.

Description

For AT40K designs containing dynamic macros (such as FGEN1) functional simulation cannot be performed without reading the netlist into Figaro and exporting a simulatable netlist. With the Post Mapping simulation users can simulate their designs after mapping but before the design is completely placed and routed. In order to do the Post Mapping simulation the design has to be first run through Initial placement.

Solution

Run the design through initial placement and restart the simulation using *Tools>Simulation>Post Mapping Simulation*

Cannot check in a macro that has its data directory the same name as the library directory.

Description

The macro name cannot be the same as the user library name. This is because Figaro needs to store data about the macro in a directory with the same name during the check-in process. This directory must then be moved to the user library. If the library and the macro directory have the same name, then it will be difficult to maintain files correctly.

Solution

Rename the macro or the library so there is not a conflict.

Cannot find environmental variable WDIR.

Example

Cannot find environmental variable WDIR.

Description

Viewlogic software requires that the variable WDIR be set to invoke its tools.

Solution

Check to make sure that this variable is set before invoking Figaro. Refer to the installation section of the *User's Guide* or the Viewlogic Tutorial for more information.

Cannot find file "filename".

Example

Cannot find file "*design*.vsm".

Description

The IDS design management features run many different programs automatically for the user as part of the design flow. Many steps require input files that are created by prior programs in the flow. Checks are made along the way to ensure that all files are created as needed. If a file does not get created this message is displayed.

Solution

Refer to the log file or the transcript window to determine why the required file was not created.

Cannot find or create project.lst file. Check WDIR environment variable.

Description

The project.lst file is used by Viewlogic software to determine which project/design directory is currently active. It should be located in a directory named "vf" which should be found in the path specified by the WDIR environment variable.

Solution

Check to make sure the WDIR environment variable is set. At least one of the paths specified must point to a directory that is writeable as the project.lst file will be updated as needed.

Cannot write to directory “directory”. WVPlus program will not be able to locate design files.

Example

Cannot write to directory “e:\wvplus”. WVPlus program will not be able to locate design files.

Description

WVPlus requires that the viewdraw.ini file be placed in the Windows “working directory”. The viewdraw.ini file is used to specify where design and library files are located. As Figaro tries to maintain this file, it must have write access to the working directory.

Solution

Check that the working directory (look at the Atmel IDS icon’s properties) is set to a directory which the user has write access to. Change to a new directory or provide write access as needed.

Command not found, could not invoke program.

Description

Many of the programs that are invoked from IDS are assumed to be in the user’s path.

Solution

Verify that the program is in one of the directories specified by the PATH environment variable. Update the path and the environment and then restart Figaro. DOS has a limit of 128 characters in the path so check to ensure that the PATH setting does not exceed this limit.

Component cannot be generated using Atmel Macro Generators due to mismatch in LPM properties.

Example

XYZ cannot be generated using the Atmel *Macro Generators* due to mismatch in LPM properties.

Description

When an LPM EDIF netlist is read into Figaro, the LPM components are automatically identified for generation and an MGI dialog box is brought up. If the LPM component in the EDIF netlist cannot be created by one the Atmel’s *Macro Generators*, then the above warning message appears.

Concept required directory "/usr/valid" does not exist. It should be a link to the Cadence directory.

Description

Concept requires that the directory /usr/valid exist on the system it is run on. It will use this directory to find all of its libraries and setup files.

Solution

This directory is often a link to the actual directory where the Cadence software is installed. The user should create this link before trying to run any of the IDS design flow tasks for the Concept platform.

Could not find a viewdraw.ini template file in either the WDIR or current directory. Viewlogic environment cannot be setup correctly.

Description

In order to set up the environment properly for the Viewlogic CAE tools, IDS must have a template file called viewdraw.ini. This file is typically installed with the Viewlogic tools in a directory pointed to by WDIR.

Solution

Check if the WDIR environment variable is set correctly. Verify that at least one of the paths specified by the WDIR environment variable points to the "standard" directory under the Viewlogic installation directory. This should normally contain a template viewdraw.ini file. Otherwise, try to find a viewdraw.ini file from a previous project and copy it to one of the WDIR paths.

Could not find file <AtmelDir>/lib/"Text" Check that environment variable ATMELDIR is set to the Directory directory or there is a Directory directory.

Example

Could not find file *<AtmelDir>/lib/concept.* Check that environment variable ATMELDIR is set to the *AtmelDir/etc* directory or there is an *<AtmelDir>/lib* directory.

Description

IDS uses the environment variable AtmelDir to determine where many of the files it uses are located. AtmelDir should point to the /atmel/etc directory. From there IDS will be able to find all the libraries (/atmel/lib) and program files (/atmel/bin).

Solution

Check to make sure that the ATMELDIR environment variable points to the /atmel/etc directory. From that directory go up one and over to the lib directory (/atmel/lib) and check to see if the specified library exists.

EDIF file was not created by cdb2edif program. Cannot proceed further.

Description

For VeriBest integration, IDS uses the cdb2edif program to create the netlist. If the program fails to produce an EDIF output file this message will be displayed.

Solution

Review the transcript or log file to check for messages explaining why the program failed to produce the needed outputs. Also, try running the program in a shell window if there are no obvious errors in the log file. Make the needed changes and run again.

Enread failed. Mentor Design object for component Component is not created. Please refer to the log file for details.

Example

Enread failed. Mentor Design object for component mult4new is not created. Please refer to the log file for details.

Description

The *Macro Generators* flow for Mentor will output an EDIF file for the creation of a design object with Enread. If this operation fails, the above message will be displayed.

Solution

Make sure that the Enread program is available and licensed. It is required as part of the Macro Generators flow. Also, check the log file or the transcript window for details of why the program failed to create the design object.

Environment variable ATMELFPGA not set.

Description

This environment variable is used by VeriBest tools and Figaro to locate the Atmel FPGA symbol, simulation and synthesis libraries. An error will be reported if this environment variable is not set before invoking Figaro.

Solution

The value of this variable is the path to the *vendor* directory in the VeriBest software tree. For example if it is installed at /sw/igraph, then this variable is set as:

```
% setenv ATMELFPGA /sw/igraph/vendor
```

Environment variable MGC_LOCATION_MAP not set.

Description

Mentor software requires that a location map file exist to specify where the various design libraries used are located. IDS will update this file with the location of the Atmel libraries and any user libraries. This variable must be set before invoking Figaro.

Solution

Set the variable as needed and re-invoke Figaro.

Error setting up registry entries for Workview Office.

Description

Workview Office uses the Window's Registry to store the location of the current project and project file. IDS will try to keep this information updated as needed. If there are problems accessing this information, this message is displayed. The Registry Editor can be used to review the ActiveProject and ActiveProjectDir settings in HKEY_CURRENT_USER/Software/Viewlogic/Workview_Office.

Solution

Check to ensure that Workview Office has been installed correctly. Run the Workview Office Project Manager to fix any problems in the Registry setup. After that IDS should be able to maintain the settings.

Error found in the location map at line Number.

Example

Error found in the location map at line 10.

Description

The IDS software will update the Mentor location map file with the paths for the Atmel schematic, simulation, synthesis and user libraries. Errors encountered while reading the file or trying to update it will result in the above message.

Solution

Look at the location map file pointed to by the environment variable $MGC_LOCATION_MAP. Check to make sure it contains valid syntax and correct as needed.

Errors occurred in verilog generation. Please correct the errors and restart simulation.

Description

IDS will invoke the needed VeriBest tools to generate a Verilog file from the schematic for use in simulation. Any error preventing the Verilog file from being generated during the process will cause this message to be displayed.

Solution

Review the transcript or log file to see if any messages are displayed about why the program failed to produce the needed outputs. Also, try running the program in a shell window if there are no obvious errors in the log file. Make the needed changes and run again.

Exemplar tool cannot be invoked in command mode from Figaro. Please run "fpga -help" from the shell.

Description

The specified Exemplar synthesis tool must be run from a shell window and not from Figaro.

Solution

Follow the specified directions.

Viewlogic synthesis tool cannot be invoked in command mode from Figaro. Please run "aurora_e -help" from the shell.

Description

The specified Viewlogic Workview Office synthesis tool must be run from a shell window and not from Figaro.

Solution

Follow the specified directions.

Figaro back annotation directory Directory was not created. Cannot proceed with macro check in.

Example

Figaro back annotation directory c:\atuser\4bitalu\user\mult4new\figba was not created. Cannot proceed with macro check in.

Description

To build a macro, the *Macro Generators* must create all the files needed for design entry, simulation, and layout of the component. Part of this process creates a simulation netlist that is used for *Post-layout Simulation*. This is commonly stored in the library directory under the macro name in the “figba” sub-directory. This directory and its associated files may not be created due to a problem with the *Macro Generators*.

Solution

Check any prior error messages in the transcript window or the log file during the processing of the macro. Otherwise contact Atmel technical support for further assistance.

File "filename" does not exist. Check log for problems in creating this file.

Example

File "mult4new.vhd" does not exist. Check log for problems in creating this file.

Description

The IDS design management features run many different programs automatically for the user as part of the design flow. Many steps require input files that are created by prior programs in the flow. Checks are made along the way to ensure that all files are created as needed. If a file does not get created this message is displayed.

Solution

Refer to the log file or the transcript window to determine why the required file was not created.

Keyword after MGC_LOCATION_MAP_1 not understood. Expected “force”.

Description

The IDS software will update the Mentor location map file with the paths for the Atmel schematic, simulation, synthesis and user libraries. Errors encountered while reading the entry for MGC_LOCATION_MAP in the file will result in the above message.

Solution

Look at the location map file pointed to by the environment variable $MGC_LOCATION_MAP. Check to make sure it contains valid syntax for the MGC_LOCATION_MAP line and correct as needed.

LPM_Hint property not supported at this time. Ignoring LPM_Hint.

Description

When an LPM EDIF netlist is read into Figaro, the LPM components are automatically identified for generation and an MGI dialog box is brought up. Figaro does not support the property LPM_Hint and so ignores it while issuing the above warning message.

Solution

Disregard the message and proceed with macro generation.

LPM property Component is not supported for component Component.

Example

LPM_Avalue is not supported for component LPM_DFF.

Description

When an LPM EDIF netlist is read into Figaro, the LPM components are automatically identified for generation and an MGI dialog box is brought up. Figaro does not support some LPM properties for certain components and so ignores these properties while issuing the above warning message.

Solution

Disregard the message and proceed with macro generation.

Mentor design for the macro Component does not exist at Directory.

Example

Mentor design for the macro mult4new does not exist at /atuser/4bitalu/user/mult4new.

Description

During the checkin process of an user library macro under the Mentor tool flow, Figaro tries to move the Mentor EDDM design object for that macro into the user library directory and will automatically update the references of that user macro in all other EDDM objects in the current design directory. Problems encountered during the above mentioned process will result in an error.

Solution

Make sure the Mentor design object for the user macro is available in the current design directory before proceeding with the checking process.

MGC_LOCATION_MAP_1 keyword not found in the location map.

Description

The Mentor location map file must start off with the keyword MGC_LOCATION_MAP_1 as the first entry in order to be valid.

Solution

Check the $MGC_LOCATION_MAP environment variable to find the location map file. Make sure this file contains the correct syntax.

No design to perform power calculation on.

Description

A design must be loaded into IDS before a power calculation can be performed.

Solution

Open a design before invoking the power calculator.

Please select a part before invoking this command.

Description

Many of the tasks in the IDS environment must be performed on a single part. If the design has been partitioned, the user should specify which part they want to perform the operation on.

Solution

Select one of the parts in the Parts Assembler window before invoking the command.

Please set the Tools Flow to VeriBest-Verilog-Schematic to invoke VeriBest Design Capture tool.

Description

This error message is displayed when *Tools Flow* is set to *VeriBest-Verilog-Synthesis* and the *Schematic Entry* button is pushed. *VeriBest-Verilog-Schematic* and *VeriBest-Verilog-Synthesis* are two different CAE systems in Figaro.

Solution

Using *File>Design Setup*, please change the *Tools Flow* selection for the current design from *VeriBest-Verilog-Synthesis* to *VeriBest-Verilog-Schematic.*

Pre-simulation file filename was not created. Cannot proceed with macro check in.

Example

Pre simulation file c:\atuser\4bitalu\user\mult4new\mult4new.v was not created. Cannot proceed with macro check in.

Description

The *Macro Generators* will create simulation netlists for both *Functional* (presim) and *Post-layout* (postsim) *Simulation*. If a problem was detected in the generation of the functional netlist, this error message will be displayed.

Solution

Check the transcript window or the log viewer for details of the problem encountered.

Program is already running, must exit before invoking again.

Description

Certain graphical programs allow only a single copy to be run at a time. If the user tries to invoke such programs more than once, this message is displayed.

Solution

Check if the program that IDS is trying to invoke is already running. If so, try to use it as is, or exit and then invoke the program from Figaro again. This message is sometimes also displayed for other problems encountered on invoking the program. Try launching it from the Windows program manager or desktop.

Program hdlPinlist did not create expected output.

Description

In order to support creation of symbols for Macro Generators and user defined macros for Concept, IDS uses the program hdlPinlist. This program uses a Verilog description of the design to create the symbol. It works in conjunction with the bodygen program.

Solution

Make sure that the hdlPinlist program exists in the path. Check the log file or the transcript window for any errors that may have been produced while running the program.

Program redifnet did not create output directory.

Description

The redifnet program is used to create a connectivity description of a *Macro Generator*. The generator outputs an EDIF file and this is supplied as input to the redifnet program. It is expected to create a directory with the name of the macro in the design directory.

Solution

Check to make sure that the redifnet program is in the path. Check the log file or the transcript window for any errors that may have been produced while running the program.

Program was not invoked successfully, exited with status: Number.

Example

Program was not invoked successfully, exited with status: 2

Description

IDS is responsible for invoking the various Design Entry and Simulation tools that it has integrated. Certain status messages are returned to IDS and if they are understood will be explained in the appropriate error message. If not, this message will be displayed.

Solution

Check to ensure that the program being invoked has been correctly installed, licensed, and is in the path. Try to invoke it using other means and see if there are further explanations on the problem. Typically, this type of message is related to licensing issues.

Project file for the design Design does not exist at Directory.

Example

Project file for the design 4bitalu does not exist at c:\atuser\project.

Description

The VeriBest tools use a project file that contains design and configuration information to manage the design data. IDS expects this file to be in the project directory when it tries to invoke functional simulation.

Solution

Check that Figaro is pointing to the correct project directory and design name. Change as needed. If the file is still missing, run the appropriate VeriBest tool to generate the project file again.

Schematic for the design Design does not exist at Directory. If you have a verilog file for your design, please run the simulator from the Shell.

Example

Schematic for the design 4bitalu does not exist at c:\atuser\project. If you have a Verilog file for your design, please run the simulator from the Shell.

Description

IDS will invoke the needed VeriBest tools to generate a Verilog file from the schematic for use in simulation. If the design does not have a schematic this message will be displayed.

Solution

Check to make sure the current project and design names have been identified to Figaro. If this is a language-based design be sure the CAE system is set to *VeriBest-Synthesis*.

Schematic Generator failed for the component "Component". Please refer to the log file for details.

Example

Schematic Generator failed for the component "mult4new". Please refer to the log file for details.

Description

The *Macro Generators* flow for Mentor will create a schematic from the design object that has been output by the enread program. If this operation fails, the above message will be displayed.

Solution

Make sure that the SG program is available and licensed. It is required as part of the *Macro Generators* flow. Also, check the log file or the transcript window for details on why the program failed to create the schematic.

Schematic only macros not supported for the specified CAE platform.

Description

The *Macro Generators* flow for certain CAE systems only supports components which are placed in the library and have hard layouts. No IO generators or soft macros can be used with these platforms.

Solution

Verify that the CAE platform is specified correctly under *File>Design Setup*. Choose another generator or check the Hard Macro option so it does not produce only a schematic as output.

Statistics in error. There must be at least 8 numbers to display.

Description

This is an internal error.

Solution

Contact customer support.

Stimulus file testfixture.verilog not found. Empty file will be created.

Description

In order to invoke the Verilog simulator for a Concept design, the user must create a file named testfixture.verilog. This file should contain the stimulus for simulating the design. If it does not exist when either of the *Simulation* buttons are pressed, the program will create a blank file to allow the process to complete.

Solution

Create the testfixture.verilog with the needed stimulus. It should be placed in the design directory.

Support for multiple speed grades across a partitioned design is not currently supported. Intrinsic delays for part Instance will be done at the Number speed grade.

Example

Support for multiple speed grades across a partitioned design is not currently supported. Intrinsic delays for part A will be done at the -2 speed grade.

Description

Within IDS it is possible to partition a design across parts of different speed grades. As such, when performing back annotation for the design, the appropriate speed grade must be used to generate correct delay information. This ability is not supported for Viewlogic back annotation. IDS must choose a single speed grade that is the speed grade of either the first part or the currently selected part.

Solution

Check to make sure that the parts have been chosen correctly. If not, select the incorrect part, and use *Edit>Change Part Speed or Application*.

Symbol creation failed. Symbol for macro Component required before check-in can proceed.

Example

Symbol creation failed. Symbol for macro mult4new required before check-in can proceed.

Description

A symbol is required for any macro that is placed in the library for any CAE platform that supports schematic entry. IDS will verify this when a macro is checked into the library and will use the appropriate software to create the symbol. This message is displayed whenever this operation fails.

Solution

Check the log file or the transcript window for details of why the program failed to create the schematic.

Text could not invoke program

Example

Out of memory, could not invoke program

Description

IDS invokes various design entry and simulation tools. During this process, the operating system will return the status of the invocation. If it signals an error, IDS will interpret this information and report on it.

Solution

Check to ensure that the program being invoked has been installed properly and its environment as well as license are correct. Use Windows Help to determine a cause for other problems not related directly to the tool being invoked.

Text is not supported on the PC platform.

Example

Concept is not supported on the PC platform.

Description

Certain CAE platforms are not supported on the PC.

Solution

Check the setting in the *File>Design Setup* dialog box for the current design. It should be set to the appropriate *Tools Flow.*

The following environment variables must be set before running Figaro: Text

Example

The following environment variables must be set before running Figaro: ATMELDIR

Description

IDS requires that certain environment variables be set in order to find all of the various libraries and files that are a part of the system.

Solution

Check the “Installation Guide” Chapter of the *User’s Guide* for details on the variables that are displayed by this message.

The program reported the following warnings during execution: Text.

Example

The program reported the following warnings during execution:

Warning: Missing xyz parameter.

Description

IDS runs certain programs from the shell of the operating system. These programs do not always supply consistent information on their status. IDS will check the output generated for any Warning messages and display them.

Solution

Look at the warning message and determine if it needs to be fixed. Find the specified message in either the IDS message reference or the message reference for the tool being invoked.

vbdc.sup file was not created by the verilog generator. Please check the log file for problems.

Description

IDS will invoke the needed VeriBest tools to generate a Verilog and vbdc.sup file from the schematic for use in simulation. If any errors occur during this process which prevent the Verilog file from being generated this message will be displayed.

Solution

Review the transcript or log file to check for messages explaining why the program failed to produce the needed outputs. Also, try running the program in a shell window if there are no obvious errors in the log file. Make the needed changes and run again.

Verilog files are not found in Directory.

Example

Verilog files are not found in c:\atuser\project1\genhdl

Description

On the PC, Figaro assumes all the verilog files generated from VeriBest schematic and state diagrams are stored in a directory called genhdl with a .vsh extension in the current design directory. The above message will be displayed if no verilog files are found in that directory due to errors from the VeriBest Verilog Generator.

Solution

Review the transcript or log file to check for messages explaining why the program failed to produce the needed outputs. Also, try running the program in a shell windows if there are no obvious errors in the log file. Make the needed changes and run again

BstrWin

Baseline bitstream is same as the active design bitstream. Please choose different bitstreams.

Description

The bit stream window process requires two different bit streams. It will compare them and generate a new bit stream, which is the difference between the two. As such, the user should not use the same file name for both input bit streams.

Solution

Change either of the input files to be a different file.

No baseline bitstream files are available.

Description

The bit stream window process requires two different bit streams. It will compare them and generate a new bit stream, which is the difference between the two. If a bit stream exists for the current design, but there are no other bit streams in the project directory that can be used as a baseline bit stream file, this message will be displayed.

Solution

Create a bit stream file that can be used as a baseline file, or copy such a file to the current project directory.

No bitstream is available for the current design.

Description

To execute the bit stream window process, a bit stream file must first exist for the current design.

Solution

Create a bit stream file for the current design.

Please select a Baseline Bitstream from the list.

Description

A bit stream file name must be selected for both the current design and the baseline design. This message will be displayed if the user clicks on the OK button without first selecting a baseline bit stream file name.

Solution

Make a selection from the Baseline Bitstream list before clicking on the OK button.

Cascade

At least two bitstream files must be available.

Description

The Cascade process will take 2 or more bit stream files and cascade them into a larger file with the appropriate modifications required by the FPGA device.

Solution

Either generate additional bit stream files, or copy additional bit stream files into the current project directory.

Please select 2 or more bitstreams from the left list.

Description

The Cascade process will take 2 or more bit stream files and cascade them into a larger file with the appropriate modifications required by the FPGA device.

Solution

Choose 2 or more designs from the list on the left so they will be displayed in the list on the right before pushing the OK button.

Compress

Base project is same as the active project. Choose a different base project.

Description

This message is displayed when the bitstream is windowed with reference to itself. Two separate bitstreams should be used during bitstream windowing.

Solution

Select the base project other than the project that is active in Figaro.

Bitstream filename is missing.

Example

Bitstream 4bitalu.bst is missing.

Description

The specified bit stream file was not found in the design directory.

Solution

Check that the bit stream file has been generated before running Compress.

No bitstreams are available for the current design.

Description

The bit stream compression utilities can only be executed if at least one bit stream file exists within the current design

Solution

Generate a bit stream file for the current design.

Please select a bitstream to compress from the list.

Description

This message is displayed if the user clicks on the OK button in the Compress dialog window while no bit stream file is selected in the Active Project Bitstreams list.

Solution

Select one of the bit streams from the list before pressing the OK button

Downld

Bitstream filename is missing.

Example

Bitstream 4bitalu.bst is missing.

Description

The specified bit stream file was not found in the design directory.

Solution

Check that the bit stream file has been generated before running Downld.

No bitstream found for download.

Description

This message is displayed when the download program could not find a bitstream in the project directory.

Solution

Generate bitstream and re-run the download program.

No bitstreams are available for the current design.

Description

The Downld utility can only be executed if at least one bit stream file exists for the current design.

Solution

Generate a bit stream file for the current design.

Please select a bitstream from the list.

Description

In order to down load a bitstream to the FPGA or configuration memory, one of the bitstreams in the list must be chosen.

Solution

Select one of the bit streams before pressing the OK button.

PLA2cdb

Pla File filename is missing.

Example

Pla File sub.tt1 is missing.

Description

The input to the PLA Optimization program is either a *.tt1, *.tt2, or *.pla file depending on which options are chosen in the dialog box. These files are assumed to be in the design directory.

Solution

Verify that the correct options are selected in the dialog box. Also, make sure the appropriate files specified above are in the design directory.

QuickChange

A bitstream needs to be generated for this design before QuickChange data is available.

Description

Before invoking the QuickChange dialog box, the needed input files must be generated.

Solution

Be sure the design has been completely placed and routed with no contentions. Then press either the *Bit Stream* button in the *Compile* window or use the *Flow>Compile>BitStream* menu option. After this has been run, check for any error messages in the log viewer or transcript window. Ensure that the *design*.bst and *design*.d files have been generated from the bit stream process.

File filename is missing or is not readable.

Example

File 4bitalu.d is missing or is not readable.

Description

Each design needs a file that contains design-specific QuickChange information. This file is produced as part of the bit stream generation procedure for the design.

Solution

Execute the *Flow>Compile>Bitstream* menu option for the design.

Incorrect format specification at (Row). Only binary or hexadecimal strings are allowed.

Example

Incorrect specification at 3. Only binary or hexadecimal strings are allowed.

Description

This message indicates that an incorrect entry has been made corresponding to the instance at a particular position on the instance list within the QuickChange dialog window.

Solution

Verify that the *New Value* entry for the specific instance is expressed in either binary or hexadecimal. If the entry is expressed in hexadecimal, ensure that the numeric value has an ‘h’ prefix.

Incorrect Format String Found in the Configuration Information Field.

Description

The *New Value* entries for each instance must be expressed either as a binary string, or as a hexadecimal value. If the entry has an incorrect digit within the expression, this message will be displayed.

Solution

Check that all *New Value* entries are expressed in the correct format.

Length of the new configuration and existing configurations at (Row) are unequal.

Example

Length of the new configuration and existing configurations at 3 are unequal.

Description

When the *New Value* entry for an instance at the indicated position in the instance list contains a binary expressed value that contains more digits than the corresponding *Old Value* entry, this message will be displayed.

Solution

Correct the *New Value* entry so that it contains as many binary digits as the *Old Value* entry.

Length of the new configuration and existing configurations at (Row) are unequal. Unwanted MSB bits will be truncated.

Example

Length of the new configuration and existing configurations at 3 are unequal. Unwanted MSB bits will be truncated.

Description

When the *New Value* entry for an instance at a particular location in the instance list is expressed as a hexadecimal number, the implied length of the number is a multiple of 4-bits. This can mean that the suggested length of the *New Value* entry is longer than that of the *Old Value* entry (which is expressed in binary). When this is the case, the 'Unwanted MSB bits' (excess bits) in the hexadecimal value will be discarded.

Solution

Disregard the message and proceed with QuickChange.

Name of the output bitstream is missing.

Description

The QuickChange software requires an output file to store the resulting bit stream.

Solution

Specify an output file name (different from the input file name) before pressing the OK button.

Names of the base design and output bitstream are identical. Change output bitstream name.

Description

In order to prevent accidental erasure of the input bit stream, the program requires that the input and output bit stream files have different names.

Solution

Specify a file name that is different from the input file name.

No QuickChange data is available for the selected design.

Description

QuickChange looks for the one (ONEONE) and zero (ZEROZERO) cells in a design and displays them in the user interface. If none of these cells exist in a design or there were problems in generating the design.d file this message will be displayed.

Solution

Check the transcript window or the log file viewer for any problems occurring when the bit stream program was run. Re-run the bit stream program to ensure that it is functioning properly. Check to make sure the design uses either ONEONE or ZEROZERO cells as these are the only supported cells for QuickChange.

The QuickChange data for the selected design is corrupt.

Description

Whenever a bit stream file is created for a design, a QuickChange data file is also created. This contains information needed to run the QuickChange software. If any of the data within that file is corrupt, the above message will be displayed.

Solution

Re-generate the QuickChange data file using the *Flow>Compile>Bitstream* menu option.

XNF Import

Included XNF file Text not found.

Example

Included XNF file test.xnf not found.

Description

Figaro will read in the file test.xnf which is specified in parameter FILE=*filename*.xnf of a symbol. If the file does not exist, the above message will be displayed.

Solution

Put the file into the design directory or the directory that is specified by the file path.

Cannot read XNF file: Text.

Example

Cannot read XNF file: test.xnf.

Description

The file is not readable.

Solution

Change the file mode to read.

Symbol Text: Equation parameter only allowed on EQN symbols.

Example

Symbol inst1: Equation parameter only allowed on EQN symbols.

Description

Equation parameter only allowed on EQN symbols.

Solution

Delete the equation parameter.

Part number: Text is not available in Figaro.

Example

Part number: 95108pq100-7 is not available in Figaro.

Description

Figaro supports Xilinx xc3000 and xc40000 families.

Solution

Use the Xilinx xc3000 or xc4000 families.

The Text symbol is not supported.

Example

The OSC4 symbol is not supported.

Description

Some Xilinx library components are not supported by Figaro. Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Replace the component by a supported one that is logically equivalent.

The Text xblox symbol is not supported.

Example

The SRAM xblox symbol is not supported.

Description

Some Xilinx XBLOX cells are not supported by Figaro. Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Replace the component by a supported one that is logically equivalent.

Symbol: Text Invalid combinational logic definition. Too many inputs.

Example

Symbol: inst1 Invalid combinational logic definition. Too many inputs.

Description

The symbols: AND, NAND, OR, NOR, XOR and XNOR are used as combinational logic gates. When using the xc3000 or xc4000 family of parts, combinational gates with more than 5 inputs are not allowed.

Solution

Replace a logic gate with a group of gates that is logically equivalent.

Symbol: Text has mixture of set and reset pins

Example

Symbol: inst1 has mixture of set and reset pins.

Description

Both reset (CLR/RD) and set (PRE/SD) pins are connected for symbol DFF.

Solution

Disconnect one of the two pins.

Symbol: Text Invalid number of inputs.

Example

Symbol: inst1 Invalid number of inputs.

Description

The number of input pins does not match the number of inputs of the symbol defined in the Xilinx library.

Solution

Contact Customer Service.

Illegal polarity for PWD signal.

Example

Illegal polarity for PWD signal.

Description

Polarity must be 0 or 1. Polarity is specified in PWD record.

Solution

Change polarity to 0 or 1.

Cannot read MEM file: Text.

Example

Cannot read MEM file: test.mem.

Description

The memory file of XBLOX PROM is not readable.

Solution

Change the memory file mode to read.

XNF version Text not supported.

Example

XNF version 7.1 not supported.

Description

XNF version number must be smaller than 6.

Solution

Use XNF version 6 or below.

Please run Xilinx program to generate simulation model record, and then import again

Example

Please run Xilinx program to generate simulation model record, and then import again.

Description

Figaro supports CLB and IOB by using the simulation model which is inside the symbol. If there is no model group for a CLB or IOB symbol, the error massage will be displayed.

Solution

Run the Xilinx program to generate a simulation model group for the CLB or IOB symbol, and then import the new design. Refer to the Xilinx manuals for more information.

The bi-directional net: Text in top level has multiple pads

Example

The bi-directional net: *net1* in top level has multiple pads

Description

The bi-directional net is connected to more than one input pad or output pad.

Solution

Delete the extra pads.

The buffers: Text and Text cannot replace bi-directional buffer for net: Text

Example

The buffers: INREG and OBUF cannot replace bi-directional buffer for net: net1

Description

If a bi-directional port connects to a group of input and output pads as a bi-directional buffer, every input or output buffer used must be the one of the following:

IBUF, BUFG, INFF, INLAT, INFF(OldStyle), INLAT(OldStyle), OBUF, OBUFT, OUTFF and OUTFFT.

Solution

Change the unsupported buffer to a supported one that is logically equivalent.

The Memory Definition File is not specified in XBLOX PROM cell: Text

Example

The Memory Definition File is not specified in XBLOX PROM cell: inst1

Description

PROM uses the MEMFILE attribute to specify the name of a Memory Definition File that defines the contents of this PROM. If the file is not specified, the error message will be displayed.

Solution

Use the MEMFILE attribute to specify a memory file. Refer to the Xilinx manual for more information about the Memory Definition file.

User library is not specified

Example

User library is not specified

Description

The XBLOX cells and some library components are mapped to Atmel macros, so user needs to set up the user library.

Solution

Set up the user library and run XNF import again.

The port: Text of XBLOX Text in cell: Text cannot be used

Example

The port: *TERM_CNT* of XBLOX *COUNTER* in cell: *inst1* cannot be used

Description

Some ports of the XBLOX cells can not be connected. Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

ENCODING cannot be ONE_HOT for instance Text

Example

ENCODING cannot be ONE_HOT for instance inst1

Description

Figaro does not support the ENCODING=ONE_HOT attribute for XBLOX cells. Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The ports A and B of XBLOX COMPARE must be connected in cell: Text

Example

The ports A and B of XBLOX COMPARE must be connected in cell: inst1

Description

The ports *A* and *B* of COMPARE must be connected.

Solution

Contact Customer Service.

The XBLOX COUNTER cell: Text, the parameter COUNT_TO can not be set when STYLE attribute is BINARY or LFSR

Example

The XBLOX COUNTER cell: inst1, the parameter COUNT_TO can not be set when STYLE attribute is BINARY or LFSR

Description

When the COUNTER has a STYLE attribute of BINARY or LFSR, the parameter COUNT_TO can not be used. Please refer to XNF Entry in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The XBLOX COUNTER cell: Text, the parameter ASYNC_VAL value must be 0 or unset when STYLE attribute is JOHNSON

Example

The XBLOX COUNTER cell: inst1, the parameter ASYNC_VAL value must be 0 or unset when STYLE attribute is JOHNSON

Description

When the COUNTER has a STYLE attribute of JOHNSON, the port ASYNC_CTRL is mapped to reset, the ASYNC_VAL value must be 0 or unset. Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The XBLOX COUNTER cell: Text, the ports: D_IN, LOAD and UP_DN cannot be connected when STYLE attribute is JOHNSON or LFSR

Example

The XBLOX COUNTER cell: inst1, the ports: D_IN, LOAD and UP_DN cannot be connected when STYLE attribute is JOHNSON or LFSR

Description

Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The XBLOX COUNTER cell: Text, the port: CLK_EN cannot be connected when STYLE attribute is LFSR

Example

The XBLOX COUNTER cell: inst1, the port: CLK_EN cannot be connected when STYLE attribute is LFSR

Description

Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The XBLOX COUNTER cell: Text, the port: SYNC_CTRL cannot be connected when STYLE attribute is JOHNSON or LFSR

Example

The XBLOX COUNTER cell: inst1, the port: SYNC_CTRL cannot be connected when STYLE attribute is JOHNSON or LFSR

Description

Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The XBLOX COUNTER cell: Text, the ports: SYNC_CTRL and LOAD cannot be connected at the same time

Example

The XBLOX COUNTER cell: inst1, the ports: SYNC_CTRL and LOAD cannot be connected at the same time

Description

Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The XBLOX COUNTER cell: Text, the ports: LOAD and D_IN must be connected at the same time when STYLE attribute is BINARY

Example

The XBLOX COUNTER cell: inst1, the ports: LOAD and D_IN must be connected at the same time when STYLE attribute is BINARY

Description

Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The XBLOX COUNTER cell: Text, the port: CLOCK must be connected

Example

The XBLOX COUNTER cell: inst1, the port: CLOCK must be connected

Description

Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The XBLOX DATA_REG cell: Text, the port CLOCK can not be connected and CLK_EN must be connected when STYLE attribute is ILD

Example

The XBLOX DATA_REG cell: inst1, the port CLOCK can not be connected and CLK_EN must be connected when STYLE attribute is ILD

Description

Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The XBLOX DATA_REG cell: Text, the ports CLOCK, D_IN and Q_OUT must be connected when STYLE attribute is CLB or unspecified

Example

The XBLOX DATA_REG cell: inst1, the ports CLOCK, D_IN and Q_OUT must be connected when STYLE attribute is CLB or unspecified

Description

Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The XBLOX DATA_REG cell: Text, the port: SYNC_CTRL can not be connected when STYLE attribute is ILD

Example

The XBLOX DATA_REG cell: inst1, the port: SYNC_CTRL can not be connected when STYLE attribute is ILD

Description

Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The XBLOX SHIFT cell: Text, the STYLE attribute only can be LOGICAL or not specified

Example

The XBLOX SHIFT cell: inst1, the STYLE attribute only can be LOGICAL or not specified

Description

Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

The XBLOX SHIFT cell: Text, the ports: CLOCK and LS_IN must be connected

Example

The XBLOX SHIFT cell: inst1, the ports: CLOCK and LS_IN must be connected

Description

Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The XBLOX SHIFT cell: Text, the ports: LOAD and PAR_IN can not be connected at the same time

Example

The XBLOX SHIFT cell: inst1, the ports: LOAD and PAR_IN can not be connected at the same time

Description

Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The XBLOX SHIFT cell: Text, the ports: MS_OUT and PAR_OUT can not be connected at the same time

Example

The XBLOX SHIFT cell: inst1, the ports: MS_OUT and PAR_OUT can not be connected at the same time

Description

Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The XBLOX SHIFT cell: Text, the ports: SYNC_CTRL and PAR_IN can not be connected at the same time

Example

The XBLOX SHIFT cell: inst1, the ports: SYNC_CTRL and PAR_IN can not be connected at the same time

Description

Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The XBLOX SHIFT cell: Text, the ports: SYNC_CTRL and PAR_IN can not be connected at the same time

Example

The XBLOX SHIFT cell: inst1, the ports: SYNC_CTRL and PAR_IN can not be connected at the same time

Description

Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The XBLOX SHIFT cell: Text, the ports: SYNC_CTRL and PAR_IN can not be connected at the same time

Example

The XBLOX SHIFT cell: inst1, the ports: SYNC_CTRL and PAR_IN can not be connected at the same time

Description

Please refer to *XNF Entry* in this *Technical Reference & Release Notes* for more information.

Solution

Contact Customer Service.

The XBLOX SHIFT cell: Text, the BOUNDS attribute must be specified

Example

The XBLOX SHIFT cell: inst1, the BOUNDS attribute must be specified

Description

For parallel input or parallel output, if a bus width is not specified, the width propagation will determine the BOUNDS attribute. For serial input and serial output, BOUNDS must be specified.

Solution

Use the BOUNDS attribute to specify bus width.

HDLPlanner

MGL file for component “component” does not exist.

Example

MGL file for component “fifo” does not exist.

Description

This message is displayed when the layout file of a component written in the Macro Generator Language does not exist.

Solution

AT6K macros do not have MGL files. If the component is for an AT40K FPGA and is supplied by the factory, contact Atmel technical support. For user macros, add the *component*.cmp file in the appropriate IDS directory.

Set proper package file name in Options window

Description

This message is displayed when the incorrect name of the package file is given.

Solution

Access *Edit>Options>VHDL Package* and the Package File dialog box and specify the correct package file name.

Set proper package name in Options window

Description

This message is displayed when the incorrect name of the package is given.

Solution

Access *Edit>Options>VHDL Package* and the Package File dialog box and specify the correct package name.

Set proper script name in Options window

Description

This message is displayed when the incorrect name of the script file is given.

Solution

Access the *Edit>Options>Synthesis Script* dialog box and specify the correct name of the script. By default, the script is generated and kept in the file named “script”.

Set proper name of the top-level design in Options window

Description

This message is displayed when the incorrect name of the top-level design is given.

Solution

Access the *Edit>Options>Top Level Design* dialog box and specify the correct name of the top-level design. By default, the name “TopLevel” is assigned.

File filename missing or unreadable.

Example

File /atmel/lib/hdlplan/at40k/admin/macros.dat missing or unreadable.

Description

This message is displayed when a file containing a list of components is missing or unreadable.

Solution

Check for file permissions in the /atmel/lib/hdlpanner directory to ensure read access to all files. If the file is missing, contact Atmel technical support.

Illegal Configuration. Set configuration to AT6K, AT40K or AT6K Mapped to AT40K.

Description

This message is displayed when FPGA configurations other than for AT6K, AT40K or AT6K mapped to AT40K are specified.

Solution

Make the necessary changes to the configuration in the Design Setup dialog box.

ATMELDIR variable points to incorrect or non existent directory.

Description

The message is displayed when the variable ATMELDIR is found to point to an incorrect directory.

Solution

Modify the variable and re-run Figaro.

Design does not have component instantiations.

Description

This message is displayed when components or scripts are generated for a design containing no instances of components supported by the HDLPlanner.

Solution

No action is necessary.

Macro Element is user defined and cannot be generated within HDL Planner tool.

Example

Macro addPipelined is user defined and cannot be generated within HDL Planner tool.

Description

Automatic layouts of components only can be generated for factory supplied macros. This message is displayed when a user-defined component is found in the design during macro generation.

Solution

Generate a component layout using the Atmel *Component Generators* or a general-purpose macro generation utility. Add the layout to a library and make that a design library before importing the net list into Figaro.

MakeKey

Standard variables in Italics for this program are as follows:

Filename: the name of a file can be complete (all directory information specified) or simple (no directory information specified).

Number: numeric values that are not used as counts.

PinType: the type of pin can be input, output, bi-directional, or tristate.

Text: any miscellaneous string that may be output from a program.

Type: either user initialization or design configuration file or can be used for CELL, IO, BUS, CLOCK, and RESET elements.

Value: the value that a particular variable should be set to.

Program Messages

Cannot find any file in "filename" directory [m000]

Example

Cannot find any file in "sch" directory

Description

This message is displayed if MakeKey cannot find any files to update in a specific directory such as "sch" or "sym". It may also be displayed if the /i command-line option is used and no file in the specified directory matches the name appended with a sheet number (sch*name*.* or sym*name*.*).

Solution

Confirm that MakeKey is being run from the project directory and that files are present in "sch" or "sym" directories for updating. Also, verify that the name supplied with the /i option has no extension or path on it.

Cannot find any files to update [m001]

Description

This message is displayed if MakeKey cannot find any files of the format *name*.* in either the "sch" or "sym" directory.

Solution

Confirm that MakeKey is being run from the project directory and that files are present in "sch" or "sym" directories for updating. Also, verify that the name supplied with the /i option has no extension or path on it.

PLA2Cdb

Standard variables in Italics for this program are as follows:

Array: chip size.

Component: the library type of the symbol or macro.

Instance: the name of an instance.

Filename: the name of a file can be complete (all directory information specified) or simple (no directory information specified).

Number: numeric values that are not used as counts.

PinType: the type of pin can be input, output, bi-directional, or tri-state.

Text: any miscellaneous string that may be output from a program.

Type: either user initialization or design configuration file or can be used for CELL, IO, BUS, CLOCK, RESET elements.

Value: the value that a particular variable should be set to.

Program Messages

Incorrect number of PinType labels "number" specified following keyword "value", expecting "number" [p601]

Example

Incorrect number of input labels "7" specified following keyword "i", expecting "8"

Description

This message is displayed if PLA2Cdb found discrepancies between the number of input (output) signals specified in .i (.o), and the number of signals read in .ilb (.olb).

Solution

Correct the problem by modifying either the signal list.

Incorrect number of IO pins "number" specified following keyword PINS, expecting "number" [p602]

Example

Incorrect number of IO pins "7" specified following keyword PINS, expecting "8"

Description

This message is displayed if PLA2Cdb found discrepancies between the number of external pins being specified and read. Note that PINS is not a valid *.pla file keyword and it is embedded in the comment line. This problem can occur if comment lines are modified.

Solution

Correct the problem by modifying either the external I/O pin list, or the I/O pin count for the PINS keyword.

Unsupported PLA type "value" specified [p603]

Example

Unsupported PLA type "*fx*" specified

Solution

Modify the design by using supported extensions for register types. Refer to the "Command Reference, PLA2Cdb" section in this manual for legal register types and dot extensions in this application.

Found illegal PLA character “text” in plane “value” at line “number” [p604]

Example

Found illegal PLA character “&” in plane “OR” at line “84”

Description

This message is displayed if PLA2Cdb encountered unsupported or invalid PLA character in the character matrix. Only characters “-”, “~”, “1” and “0” are supported.

Solution

Change the unsupported character to one of the supported characters.

Number of characters in phase definition "number" is not equal to number of outputs "number" [p605]

Example

Number of characters in phase definition "5" is not equal to number of outputs "4"

Description

This message is displayed if PLA2Cdb found discrepancies between the number of outputs in the keyword .o and the phase vector length specified in the keyword .phase.

Solution

Correct the problem by modifying the phase vector length to correspond to the number of output signals specified in the .ob line.

No PinType to PLA specified in file "filename" [p606]

Example

No input to PLA specified in file "counter.tt1"

Description

This message is displayed if PLA2Cdb encountered a *.pla file having zero primary inputs or outputs.

Solution

Designs must have at least one primary input and output.

No product terms specified in file "filename" [p607]

Example

No product terms specified in file "counter.tt1"

Description

This message is displayed if PLA2Cdb encountered a design having no product terms specified in the character matrix.

Solution

Design must have at least one product term.

Keywords .i and/or .o must come before any PLA file records [p609]

Description

This message is displayed when the order in which PLA file records appear is changed. PLA2Cdb expects keywords .i and/or .o to appear before the character matrix so that various consistency checks can be performed.

Solution

Change the order in which PLA records are arranged such that .i and .o appear before the character matrix.

Unsupported or invalid ABLE HDL dot extension "value" for register "type" found [p615]

Example

Unsupported or invalid ABLE HDL dot extension ".qq" for register "reg01" found.

Description

This message is displayed when unsupported or invalid dot extension is used to specify the input pin of a register.

Solution

Modify the design to use one of the supported register dot extensions.

Assuming DFF type for register "type" [p616]

Example

Assuming DFF type for register "cntl"

Description

This message is displayed when PLA2Cdb cannot accurately determine the type of the registers. All registers that cannot be accurately determined are assumed to be type "D".

Solution

Generally, no action is required if type "D" flip-flop is intended, otherwise attach an extension to the register to uniquely identify it as type "T".

Register “type” does not have an equivalent component in ATMEL macro library [p617]

Example

Register “sum0” does not have an equivalent component in ATMEL macro library

Description

This message is displayed when an unsupported register type is used in the design.

Solution

Modify the design to use one of the supported register types.

Asynchronous reset as well as preset signals were found for register "instance" of type "component" [p622]

Example

Asynchronous reset as well as preset signals were found for register "myreg" of type "dreg"

Description

This message is displayed when the extensions ".ar" and ".as" are found for a register. No register in the Atmel Macro Library contains both asynchronous reset and preset pins.

Solution

Designs must be rewritten to use either asynchronous reset or preset pins.

Net "Netname" was mapped to a tri-state output buffer [p623]

Example

Net "O8" was mapped to a tri-state output buffer

Description

This is a warning message given to inform the users that a particular net has been mapped to a tri-state buffer.

Solution

No action is necessary.

PLA file not conforming to open PLA format was found. Regenerate PLA file using CUPL or ABEL compiler [p625]

Description

Only PLA files generated using CUPL or ABEL compilers are supported.

Solution

Generate PLA files using ABEL or CUPL compiler or modify PLA files to conform to open PLA format.

A component must have at least one external pin [p626]

Description

A component must have at least one external pin.

Solution

Add an external pin to the

Macro Generators

Thank you for your interest in Atmel's AT40K and AT6K series FPGAs, and welcome to Atmel's very powerful and unique design tool, the Automatic Component Generators. Designed to exploit the many innovative features of Atmel's AT40K and AT6K architecture, high-speed custom logic functions can be created and implemented, resulting in significantly improved performance for compute intensive applications.

This software facilitates quick and easy implementation of over 75 logic functions, such as multipliers, adders, accumulators and multiplexers. Users can specify individual DSP parameters, including width, pipelining, and other function-specific options. Within minutes the design tool will automatically create both a "hard layout" with worst case speed and area information, as well as generate a schematic symbol, VHDL or Verilog data for the function. The end result of this process is a fully specified, reusable circuit that can be used for any size AT40K and AT6K FPGA array, in any location or orientation, without affecting its speed or function. The use of these fully deterministic functions in synthesis results in significantly higher silicon efficiency, as well as faster speed and design time.

For specific information on the AT40K macro generators, check the IP Core Generators guide available under SystemDesigner/doc/40k_Generator_guide.pdf. For specific information on the AT6K macro generators, check the IP Core Generators guide available under SystemDesigner/doc/6k__Generator_guide.pdf. For each generator there is an explanation of its function, the various parameters available, a list of its input and output pins, the statistics for 16- and 8-bit versions of the macro (except for Adder and Subtractor Carry Select, which have a minimum bit width of 9), and a screen capture of the graphical user interface.

The statistics for the macro contain a number of figures relating to its performance and area. The second and third columns in the table are about the speed of the component. It should be noted that the speed figures for macros that contain sequential components (registers) exclude the delay time for getting the global clock onto the chip and to the register element. This is around 3 to 4 ns plus a fanout factor. The next two columns deal with the size of the macro. The Cells column is the number of core cells used to build the component. The Size is the width and height of the macro in core cells.

Cache Logic™ FPGAs

The AT40K and AT6K series of co-processor FPGAs are designed specifically for compute-intensive DSP functions. Innovative features include:

- New logic cell that facilitates implementation of array multiplication
- Dedicated SRAM that is distributed throughout the array
- I/O with multiple direct paths to the center of the FPGA array

The AT40K and AT6K device is comprised of a perfectly symmetrical array of octagonal logic cells with direct connects to adjoining cells on all eight sides. The number of connections between cells helps to conserve busing resources and minimizes delays. More important, by providing diagonal connections between cells, the chip allows the efficient implementation of extremely high performance array multipliers as well as excellent support for synthesis-based designs. The array also features five separate planes of high speed busing, each of which contains one local bus and two ultra-fast express buses. When combined with the extensive direct cell connections, the busing network provides the FPGA with excellent routability.

The AT40K and AT6K core cell has two three-input LUTs, plus an upstream AND gate. The two LUTs can be used to form a full-adder which, when combined with the upstream AND gate serves as a multiplier element. Therefore, a multiplier of any size can be constructed using an array of individual cells that are only directly connected.

The embedded *FreeRam*ä on the AT40K and AT6K device is distributed throughout the array in 32 x 4 blocks. These dedicated SRAM blocks are not constructed from logic cells, so implementing SRAM does not use up core cell logic resources. *FreeRam* is completely flexible, integrated with the multi-plane busing system, and can be structured in a variety of widths and depths. Since it is distributed throughout the array, access is extremely fast and minimal routing resources are used.

With *Cache Logic*, DSP and other logic functions can be created quickly and accurately using Atmel's design software, programmed into the FPGA, and then changed as new functions or algorithms are required. Individual LUTs can be reprogrammed in just 33 ns, without loss of register data. In fact, part of the array can be reprogrammed while the remainder continues to operate without interruption. These features will offer significant performance gains for graphics and imaging, network, military instrumentation, and telecommunications applications.

Cache Logic makes adaptive hardware possible, and accelerates system performance by handling logic much the same way a computer's cache memory handles instructions and data. In a computer, the highest speed memory (usually SRAM) stores active data, while the bulk of the data resides in a less expensive DRAM, EPROM, or disk. In most complex applications, only a small fraction of the circuitry is active at any given time. Only active functions are loaded into the FPGA's "logic cache", while unused circuits reside in less expensive external memory. Logic can be added to the "cache" as needed to replace or complement existing functions without physical modification to the hardware.

With *Cache Logic*, a 25,000-gate application may actually only require 5,000 gates at any given cycle. By caching the extra 20,000 gates for later use, a 5,000-gate device can replace a more expensive 25,000-gate device.

Atmel's AT40K and AT6K series boasts an innovative combination of features: a compute-intensive, yet flexible logic cell, diagonal cell-to-cell connections, *FreeRam*ä distributed SRAM, and multiple I/O options for each device pin. Together, these provide the flexibility, density and performance required to implement complex functions such as those used in DSP, image processing, communications and complex datapath applications. They also support the efficient synthesis of high-performance random logic in a minimal amount of silicon.

Atmel's design software, the Integrated Development System, uses a single database to take your work from design entry to configured circuit quickly and efficiently. With exclusive features like system level hard macro functions, component generators, *Synthesis-Gateway*ä software and advanced timing analysis that uses physical wire lengths and loads to predict delays, our software tools let you design circuits that perform exactly to specification.

When your finalized design enters high volume production, the tools allow easy migration to Atmel's masked Gate Arrays for even more cost-effective production.

MGI

This document outlines the methodology for integrating Atmel's *Macro Generators* into synthesis flows using the various synthesis tools' module generation systems. The module generation capability in the synthesis tools is a mechanism that matches certain behavioral constructs, or arithmetic and relational operators with pre-designed implementations. Some tools support simple operator inferencing while others can identify complex structures like multiplexers, decoders etc.

However to take advantage of this module generation capability, some tools require a library that is separate from the regular synthesis library. These inferred module generator components are identified from the EDIF netlist by Figaro's EDIF netlist processor and an intermediate file (.*designName*.mgi) is created. The *.mgi file is then used by Figaro to match the modules inferred by the synthesis tool with the equivalent Atmel *Macro Generators* and a user interface (called the MGI dialog box) is brought up. The MGI dialog box will list all the *Macro Generators* components identified from the EDIF netlist as pages of a notebook.

The *Macro Generators* parameters can be classified as:

- physical implementation
- functional

Physical implementation parameters include layout specific options like fold, pitch, aspect ratio, area or speed optimization etc. Functional parameters include the type of *Generator*, width, size etc. The purpose of the MGI dialog box shown below is to give the user flexibility and control over the physical parameters of the *Macro Generators* which affect the layout directly. The user cannot modify any functional related parameters of the components in the MGI dialog box as they are inferred by the synthesis tool.

The following is a brief description of the various buttons in MGI. The tabs of the notebook pages show the macro name and the *Macro Type* entry box shows the equivalent *Generator* function for that component.

Component Exists

This box will be checked if a macro with the same name already resides in the library. This may happen if the design has gone through one iteration. If the component exists and it matches the displayed parameters, its generation is skipped.

Updated

This box is checked if the default settings of the macro have been changed. By default, the user interface displays options per the following guideline. If the component exists in the library, the options that it was run with are displayed. If the component does not exist in the library, the default settings that minimize the component area will be chosen.

Generate

This button is used to proceed with the generation of the macros.

Cancel

This button is used to skip the generation of the macros. If this button is pressed, the design will not contain the hard layout of the macro. Instead, its soft implementation will be chosen.

The created components will be placed in the library displayed in the *User Library* field.

To automatically generate hard macros for the identified *Macro Generators* components without bringing up the graphical user interface, use the *Options>Options* menu. In the *Global Options* dialog box, check the *Automatically Generate Components* option under *MGI Support*.

Module Generation Library Support

This section explains the operators that are supported by MGI in IDS 4.00 for the different synthesis tools listed below.

- Cadence Design Systems Synergy
- Exemplar Logic Leonardo and Galileo
- Everest Design Systems
- Mentor Graphics AutoLogic II
- Synopsys Design Compiler and FPGA Compiler
- Viewlogic ViewSynthesis

All the above tools are fully integrated and separate documents are available (see the "Synthesis" section of the *IDS Tutorial*) concerning the design flow for each tool. The distinct requirements of each synthesis tool are discussed below.

Cadence

Synergy requires a separate module generation library called SmartBlocks. The SmartBlocks library for the AT6000 series FPGAs contains behavioral descriptions of parameterizable components that can be generated using Atmel's *Macro Generators*. Synergy can map the behavioral constructs from the design description (VHDL or Verilog) to the components supported in the SmartBlocks library. In addition it can also map simple arithmetic and relational operators to the components in the SmartBlocks library. The Atmel SmartBlocks library supports the following parameterizable components:

- Ripple Carry Adder
- Ripple Carry Subtractor
- 2:1 Multiplexer
- 4:1 Multiplexer
- 8:1 Multiplexer
- Loadable, Enable Up/Down ripple counter
- Up/Down ripple counter
- Up/Down loadable ripple counter
- Up counter
- Down counter
- Parallel Serial Out Shift Register
- Comparator

Please refer to the "Synthesis - Cadence" chapter in the *CAE Interfaces* section of the *Tutorial* for details on running an example design using the Atmel SmartBlocks library.

Exemplar Logic

Exemplar requires a separate module generation library called Modgen. The following components are supported by the Atmel Modgen library:

- Ripple carry adder
- Ripple carry subtractor
- Comparator
- Multiplier
- Absolute Value
- Unary minus

Please refer to the "Synthesis - Exemplar" chapter in the *CAE Interfaces* section of the *Tutorial* for details on running an example design using the Atmel Modgen library.

Everest

The Everest synthesis tool does not require any module generation library support. It can directly infer Atmel *Macro Generators* components in designs targeted to the Atmel FPGA technology. The following components are inferred by the synthesis tool.

- Ripple carry adder
- Ripple carry subtractor
- Equality Comparator
- Multiplier
- Multiplexer
- ROM
- Up/Down Ripple Counter

Please refer to the "Synthesis - Everest" chapter in the *CAE Interfaces* section of the *Tutorial* for details on running an example design using the Everest Design Systems synthesis tool.

Mentor Graphics

AutoLogic II does not require a separate module generation library. But in order to take advantage of the DMAG (Dynamic Macro Architecture Generation) capability in AutoLogic, some low-level functions are included to write out LPM properties into the output EDIF netlist for all the components that are instantiated. These functions will be called after synthesis and optimization using the custom recipes provided along with the Atmel FPGA synthesis library, so that the parameters for the *Macro Generators* components will be written out as LPM properties into the EDIF netlist. In this release the following *Macro Generators* will be identified from the synthesized netlist and passed on to the MGI interface.

- Ripple carry adder
- Ripple carry subtractor
- Comparator
- Multiplier
- Absolute Value
- Unary minus

Please refer to the "Synthesis - Mentor" chapter in the *CAE Interfaces* section of the *Tutorial* for details on running an example design using AutoLogic.

Synopsys

A separate DesignWare library is provided along with the Atmel FPGA synthesis library to take advantage of the Synopsys module generation capability. Please refer to the *DesignWare Library* section in the *Technical Reference* for a list of components supported by the Atmel FPGA DesignWare library.

Please refer to the "Synthesis - Synopsys" chapter in the *CAE Interfaces* section of the *Tutorial* for details on running an example design using the Atmel DesignWare library.

Viewlogic

ViewSynthesis uses a module generation capability called ALUCONS. A separate library is provided for ALUCONS along with the Atmel FPGA synthesis libraries. Please refer to the "Synthesis - Viewlogic" chapter in the *CAE Interfaces* section of the *Tutorial* for more details on using the ALUCONS library during synthesis and optimization. The following components are supported in the ALUCONS library.

- Ripple Carry Adder
- Ripple Carry Subtractor
- Equality comparator
- Greater than comparator

MGL

The *Macro Generator Language* (MGL) has been created to allow programmatic generation of user macros for the Atmel AT40K FPGA architecture. The language has been optimized specially for this task, making it relatively straightforward to learn. This document provides a basic overview of MGL, and some instructions for using the MGL compiler and debugging tools. It is divided into four main sections. The first describes the basic language constructs used in MGL, such as conditional expressions, loop structures and so on. This is followed in the second section by an explanation of those parts of the language that allow hard macros to be described and modeled. The third section gives an overview of the process of *design generation* in MGL, which involves instantiating IO components and selecting a specific target device. Finally, a complete MGL example is given in section four.

The MGL compiler is invoked from the *Tools* menu in Figaro, under *MGL Editor*. The *MGL Debugger* can be accessed from the *Viewer* toolbar. For user interface information regarding the *Viewer* and *Debugger*, please refer to the *MGL* chapter in the *User's Guide*.

Basic Language Constructs

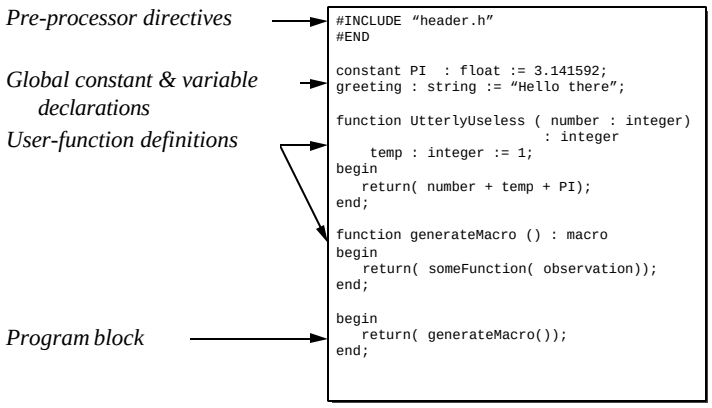
Overview

MGL is a new custom language that allows users to programmatically generate hard macro layouts for AT40K series FPGAs. It is not a general-purpose HDL (Hardware Description Language), and as such cannot be used to describe the behavior of a macro. Instead, it enables designers to use their knowledge of the target FPGA architecture to influence the structure and layout of the generated macro, resulting in a more efficient implementation.

Before describing the fundamentals of MGL, three general observations should be noted:

- MGL is a strongly-typed language (i.e. all objects must be assigned a specific type)
- The MGL compiler is case-sensitive, so that `dosomething()` and `doSomething()` are separate functions.
- Many of the basic language constructs resemble their VHDL equivalents.

The diagram below gives a simplified view of the overall program structure.



MGL Program Structure

At the head of the program are the pre-processor directives, which allow external MGL code to be accessed from the current file. Following this are the global variable and constant definitions. Anything declared here is available throughout the MGL program. The next section contains one or more user-defined functions. These typically do all the work of the program, and the top-level function will usually return a *macro* object that can subsequently be imported into the IDS tools. The final section is the program block. This allows the user to call any of the functions that have been defined in the program, and pass them a suitable set of arguments. When the MGL program is compiled, it is the program block that is executed, and its return value is the object that is imported into the IDS tools.

The following provide a detailed look at the basic MGL constructs.

Comments

Comments in MGL can be indicated in two ways. Two forward slashes together indicate that the rest of the line is a comment, e.g.

```
a := b + c;   // Add b to c and assign to a
```

An alternative method, which is useful for temporarily commenting out blocks of code, is to enclose the statements with the $ character as shown below.

```
    a := b + c;
$   b := dodgyfunction( c + 10);
    c := guaranteed_crash( 16); $
    d := b - c;
```

Variables and Constants

The syntax for variable declarations is as follows.

```
variable1, variable2, .. variableN : type :=    initialValue;
```

When a variable is declared at the top of a program, before any function definitions, it is global in scope (i.e. it can be accessed from anywhere in the program). Variables declared within a function have local scope. If an initial value is not given, the variable will be undefined until assigned a value in the code. Constants can be declared in a similar way.

```
constant constant1 : type := value;
```

Here the initial value is mandatory, and it is not legal to assign the same value to a list of constants (as one can with variables). Some example declarations are shown below.

```
aNumber1, aNumber2 : integer := -1;
result : boolean;
constant PI : float := 3.14159;
```

MGL is a strongly-typed language and it supports a number of built-in object types ranging from simple data types that are common to most programming languages, to more complex ones that are geared specifically towards macro generation.

Numbers

Both floating point and integer number types are supported, with the keywords `integer` and `float` respectively. Floating point literals must include a decimal point as follows.

```
a : float := 1.0;
b : float := 12.;
c : float := .34;
```

Examples of illegal expressions are shown below.

```
a : float := 112;
b : integer := 12.2;
```

Implicit conversion between integers and floats takes place when an arithmetic expression is evaluated. In the following example, `result1` will be assigned the value `4`, whereas `result2` will be `4.66667`.

```
a, b, c, result1 : integer;
result2 : float;
...
a := 2;
b := 3;
c := 4;
result1 := a / b + c;
result2 := a / b + c;
```

Booleans

An object of type boolean is declared using the `boolean` keyword. Boolean values can be assigned the literals `true` and `false`, both of which are also keywords, as shown below.

```
aValue : boolean := true;
```

Strings

String objects are declared with the `string` keyword, as shown in the following.

```
message : string := “Hello world”;
```

String literals must be enclosed in double quotes. They can also incorporate one or more *string expressions*, which allow integer numbers to be easily incorporated into the string as shown below.

```
row : integer := 1;
column : integer := 4;
offset : integer := 6;
locationString : string;
...
locationString := “Row”{row+offset}”Column”{column+offset};
```

The variable `locationString` will have the value `“Row5Column10”`. The expressions in the curly braces are evaluated and cast to an integer value if the result is a float. Then they are converted to a string and concatenated with the preceding string literal. Any number of string literals and string expressions can be chained in this way. This feature is useful for naming instances, ports and nets in a generated macro.

Strings can be concatenated using the + operator, described later. In the following example, aString3 gets the value `“Goodbye world..”`.

```
aString1 : string := “Goodbye “;
aString2 : string := “world..”;
aString3 : string;
...
aString3 := aString1 + aString2;
```

It is also possible to add a number to a string. As shown below, the number (either an integer or a floating point) is converted into a string representation before being concatenated with the other operand.

```
prefix   : string := “Net”;
number   : integer := 3;
netname  : string;

...

netname := prefix + number;
```

There are four pre-defined functions that provide further string manipulation capabilities in MGL as described below.

uppercase(aString) This function translates its string argument such that all characters are uppercase.

lowercase(aString) The function works in a similar fashion as above.

match(aString, target) Basic pattern matching is provided by function. This looks for the target expression in `aString`, and returns a boolean value indicating whether a match has been made. The target string can contain “*” and “?” wildcard characters. These match any characters and any single character respectively.

substitute(aString, aSubstring, aReplacement) This function replaces all occurrences of `aSubstring` within `aString` with `aReplacement`.

Macros

A macro object is declared using the keyword `macro` as shown below.

```
aMacro : macro := getmacro( “FGEN1”);
```

The return value of an MGL program is usually a `macro` object that holds the generated macro. A macro object can contain instantiations of other macro objects, as well as nets and routing data. The use of `macro` objects and the `getmacro()` function is covered later.

Ports

Port objects are used to represent ports on a macro object. They can be used to specify *direct connections* between two macro instances. An example of a port object declaration is shown below.

```
aPort : port := getport(“Cell0”, “G”);
```

The use of `port` objects, and the `getport()` function will be described later.

Connections

Objects of type `connection` are used to describe either a connection between a port and a net, or a connection between two ports. They are used extensively to define the connectivity of generated macros. A connection object can take either of the two forms below.

```
connection1 : connection := “PORT1”->”NET1”;
```

or,

```
aPort : port := getport( “Instance1”, “Q0”);
connection2 : connection := “Port1”->aPort;
```

Connections are compound objects – two string or port objects are connected using the `connection operator`, `->`. The left side’s string or port object is the source, and the right side is the destination. Again, the use of connections will be discussed more in a later section.

RouteNode

`Routenode` objects are used to store routing information for macro nets. Like connections, routenodes are compound objects. They consist of two integer values and a string value enclosed in parenthesis and separated by commas. The integers represent the x and y co-ordinates of the routing node, and the string defines the routing resource being used. The routing node represents the Y output of a core cell at location 2, 3 as illustrated below.

```
aNode : routenode := (2, 3, “yOut”);
```

A summary of the most common routing resource names for core-cell routing on the AT40K is given below.

Routing Resource	Description
“wIn”	Core cell W input
“xIn”	Core cell X input
“yIn”	Core cell Y input
“zIn”	Core cell Z input
“xOut”	Core cell X output
“yOut”	Core cell Y output
“oe”	Core cell OE input
“busLN”{plane}	North local bus on planes 1 to 5
“busLW”{plane}	West local bus on planes 1 to 5
“busEN”{plane}”VHE”	North express bus on planes 1 to 5
“busEW”{plane}”VHE”	West express bus on planes 1 to 5

Certain routing nodes, such as the data and address inputs to RAM cells, require an extra argument to be provided in the routenode object. For example, to specify the “ramDIn:0” node for the RAM cell at the bottom-left corner of the FPGA, the following routenode would be used.

```
(0, 0, “ramDIn:”, 0)
```

This corresponds to the LSB data input for this RAM cell. Routing nodes are discussed in more detail in the section *Complex Routing* later in this chapter.

Files

`File` objects are used to read and write data from external ASCII files. An example of a file variable declaration is given below.

```
aFile : file;
```

File I/O functions are discussed in a later section.

Arrays

`Arrays` in MGL can have any number of dimensions, and can contain objects of any of the built-in types. The syntax for an array declaration is as follows.

```
arrayname : array [dim0][dim1]..[dimN] of type :=  initialValue;
```

The array dimensions must be enclosed in square brackets. Below is an example array declaration.

```
intarray : array [4][6] of integer := 1;
```

This is a 4 by 6 array of integers, whose elements are all initially assigned the value 1. If an initial value is not given, the array elements are undefined until assigned a value. Another way to initialize the array is to list the values that are to be assigned to the individual array elements, e.g.

```
alphabet : array [4] of string := “A”, “B”, “C”,   “D”;
```

Arrays are also accessed using the square brackets notation, as follows

```
i : integer;
..
i := intarray[2][2];
```

In this example, `i` will be assigned the value 1. Note that strict array bounds checking is performed by the compiler, and attempts to access beyond the array dimensions will be flagged as an error. An array variable may also be assigned the value of another array, but the array type and dimensions must match exactly for this to be legal as shown below.

```
a : array[6] of integer := 2;
b : array[6] of integer := 4;
c : array[6] of float;
d : array[8] of integer;
...
a := b;   // Legal
b := c;   // *Illegal*
b := d;   // *Illegal*
```

Arrays in MGL have subscripts that run from 1..size, as opposed to 0..size-1. It is possible to have an array that can hold any type of object. This can be useful as a form of scratch pad as shown below.

```
a : array[3] of anything := 2, 1.2, “Data”;
```

Arithmetic

Operators

MGL has a range of built-in operators for evaluating arithmetic expressions. All operators are left-to-right associative, and operator precedence is shown as in the following table.

Operator	Description	Precedence
- ~ NOT	Unary negation Bitwise (2's complement) NOT Logical NOT	1
* / ** mod	Multiplication Division Raise to the power of Modulo	2
+ -	Addition Subtraction	3
< <= > >=	Less than relation Less than or equal to relation Greater than relation Greater than or equal to relation	4
= !=	Equality Inequality	5
&	Bitwise (2's complement) AND	6
^	Bitwise (2's complement) XOR	7
|	Bitwise (2's complement) OR	8
AND	Logical AND	9
XOR	Logical XOR	10
OR	Logical OR	11
->	Connection (see connection type)	12

Operator precedence can be modified using parenthesis, so that the first example will yield the result 16, while the second will give 36.

```
i : integer := 10 + 2 * 3;
i : integer := (10 + 2)* 3;
```

Built in Arithmetic Functions

MGL contains a number of pre-defined arithmetic functions to complement the built-in operators that have already been described. The table below summarizes these functions.

Function	Return Value
`log(aNumber)`	Log to the base 10 of the argument, which must be > 0
`log2(aNumber)`	Log to the base 2 of the argument, which must be > 0
`ln(aNumber)`	Natural log of the argument, which must be >0
`sin(aNumber)`	Sin of the argument
`cos(aNumber)`	Cos of the argument
`tan(aNumber)`	Tan of the argument
`arcsin(aNumber)`	Arcsin of the argument, which must be between -1 and 1.
`arccos(aNumber)`	Arccos of the argument, which must be between -1 and 1.
`arctan(aNumber)`	Arctan of the argument, which must be between -1 and 1.
`sqrt(aNumber)`	Square root of the argument, which must be positive.
`trunc(aFloat)`	Integer formed by truncating the floating point argument.
`ceil(aFloat)`	Integer formed by rounding up the floating point argument.
`floor(aFloat)`	Integer formed by rounding down the floating point argument.
`abs (aNumber)`	The absolute value of the argument (i.e. the unsigned value)
`largest( number1, number2)`	The larger of the two numbers
`smallest( number1, number2)`	The smaller of the two numbers

Control Structures

MGL supports 4 types of control structure: `if/then/else statements`, `case statements`, `for loops`, and `while loops`.

Conditional Branching

Conditional execution is performed with the `if`, `then` and `else` keywords as shown below.

```
if (expression) then
    statements
else
    statements
end if;
```

The condition to be evaluated can be any expression that evaluates to a `boolean` result. It must be enclosed in parentheses, as shown in the example below.

```
if ( (a = 3) AND (b < c*2) ) then
    DoSomething( a);
else
    DoSomethingElse( b);
end if;
```

Note that the `else` part of the construct is optional. More complex branching can be achieved using the `elseif` keyword, e.g.

```
if (aValue < 3) then
    DoOneThing();
elseif ( aValue = 3 OR aValue = 4) then
    DoAnotherThing();
else
    DoYetAnotherThing();
end if;
```

Case Statements

When a multi-way decision needs to be made based on the value of a variable or expression, a case statement can be used. For example:

```
case (getALetter()) is
    when “A” : print(“A is for apple”);
    when “B” : print(“B is for banana”);
    when “C” : print(“C” is for cherry”);
    when “D” : print(“D” is for...dunno”);
    default : print(“Gimme a break”);
end case;
```

Each branch of the `case` statement is preceded by the `when` keyword and the corresponding value. The statements following the colon are executed and then program flow continues after the `case` statement. If none of the `when` branches are matched, the (optional) `default` action will be executed.

Looping

MGL contains two constructs for looping:

- The `for` statement
- The `while` statement

The following is an example of a `while` statement.

```
while ( a <= 10) loop
    ...
    a := a + 1;
    ...
end loop;
```

As with the `if` statement, the `loop` condition must be enclosed in brackets, as shown. The statements between `loop` and `end loop` are executed while the `loop` condition is true. This also applies to the `for` statement shown below.

```
for i in 1 to 10 loop
    ...
    print( i);
    ...
end loop;
```

The `loop` variable (`i` in the above example) does not have to be declared earlier in the code. It is created implicitly and can be accessed only by the statements within the `loop`. More complex `loop` counts can be created using the `step` keyword, as shown below.

```
for i in 12 to -4 step -2 loop
   ...
   print( i);
   ...
end loop;
```

In the above example `i` will have values 12, 10, 8, 6, 4, 2, 0, -2 and -4. Note that the start value, terminating value and step size must all be integer values.

User-Defined Functions

The syntax for defining a user function is as follows.

```
function FunctionName ( variablelist1 : type1,
             variablelist2 : type2 ..
             variblelistn : typen) : ReturnType
   local variable declarations
begin

   statements

end;
```

The keyword `function` is followed by the name of the function. The `return` type is declared after the formal parameter list. The function body is delimited by the `begin` and `end` keywords. The following is an example of a simple function that returns the lowest of three integer values passed to it.

```
function Lowest( a, b, c : integer) : integer

   lowest : integer;

begin

   if (a < b) then
      lowest := a;
   else
      lowest := b;
   end if;

   if (c < lowest) then
      return( c);
   else
      return( lowest);
   end if;

end;
```

Here the result of the function is declared as an integer. The `return` statement is used to return a value from the function. The value to be returned must be enclosed in parentheses, and its type must match the declared return type. If the function does not return anything, the return type declaration is ommitted. The variable `lowest` declared after the head of the function block is local to this function, and overrides any variables of the same name declared in an outer scope.

Functions are called by giving the name of the function and a list of arguments enclosed in parentheses as shown below.

```
a : integer := 1;
b : integer := 2;
c : integer := 3;
result : integer;
...
result := Lowest( a, b, c);
```

In this case, `result` will be assigned the value 1. Note that the number and type of the arguments must match the parameters declared in the function definition. Array objects can also be passed into and returned from functions as illustrated below.

```
function Offset( a : array of integer) : array of  integer

   constant offset : integer := 10;
begin

   for i in 1 to sizeof(a) loop
      a[i] := a[i] + offset;
   end loop;

   return (a);

end;
```

When an array is passed as an argument to a function, only the array type is declared. The function can then handle arrays of any size by using the `sizeof()` built-in function. Note that `sizeof()` returns the absolute size of the array as a single number, which is less useful for accessing multi-dimensional arrays. To offset this problem, the above function can be modified so the array dimensions are passed in as arguments.

The Program Block

An MGL program typically comprises 4 parts:

- Preprocessor directives
- Global declarations
- Function definitions
- Program block

Note that function definitions must be declared in order. A user function therefore cannot call another user function that has not yet been declared.

The program block is used to invoke the *Macro Generators* function that is currently being worked on.

```
begin
   return( TestGenerator(4));
end;
```

The statements between the `begin` and `end` keywords are exe cuted, and if an object of type `macro` is returned, the user has the option of running the design flow on it. Only global variables and constants can be accessed in the program block. It is intended only for debugging purposes so MGL functions can be invoked with a particular set of arguments. In the final system, MGL functions are called directly from the *Macro Generators* user interface.

The Preprocessor

The MGL compiler includes a pre-processing stage that supports the `#INCLUDE` directive. This allows variables and functions in other files to be referenced from an MGL file. An example is shown below.

```
#INCLUDE “global.h”     // Global header
#INCLUDE “decoder.h”    // Decoder functions
#END
```

Any number of files can be included, and the files that are included can themselves contain `#INCLUDE` directives. Only one copy of each `include` file is prepended to the main source file to avoid issues with circular references.

The `#END` directive indicates to the pre-processor that there are no more pre-processor directives in the rest of the file. Its inclusion is optional, but it speeds up pre-processing since only the first few lines of the file have to be searched for directives. All directives must appear at the start of a line, otherwise they will be ignored.

When a program with `#INCLUDE`s is run, the included files are prepended to the main source file before compilation. If an error occurs in an included file, an *MGL Include File Viewer* window is opened and the position of the error is flagged. The user can then fix the error, save the file, quit the additional viewer and return to the main source file to retry the program.

The MGL Debugger

A debugging environment is provided for MGL programmers. It is possible to set breakpoints and single-step through the code. Breakpoints are set using the `setbreak()` function as illustrated below.

```
for i in 1 to floor(size/2) loop
   setbreak( i = 2);
   DoSomething( i);
end loop;
```

A breakpoint is set whenever the specified condition evaluates to true. In this example, execution will stop when `i` is 2. If no condition is specified, an unconditional breakpoint is set and execution will always stop at that point in the code. Breakpoints are globally enabled and disabled from *the MGL Viewer* toolbar.

When a breakpoint is encountered, an MGL Debugger window is opened. This window comprises of three sections. The top section gives a view of all the variables and constants that are accessible from that point in the code. The value of variables at different scopes (from local to global) can be inspected.

The middle section is a view of the source code, with the breakpoint highlighted in red. Note that this view shows the *pre-processed* source code, which has all included files prepended to it. The text from each individual file is delimited by `**NEWFILE**` to differentiate code sections that come from the main source file and external files.

The bottom section of the window has debug flow controls. The left most and middle buttons are used to *step into* and *over* the next statement respectively, while the right most button will execute the program up to the next breakpoint (or the end of the program). The distinction between stepping *into* vs. stepping *over* a statement applies to function calls, `for` loops and `while` loops, as well as `interface`, `contents`, `instance` and `route` blocks. For more details, please refer to the section on *Macro Generation* below.

When single-stepping through code, the current execution position, along with a description of the current program step, is highlighted in both the *Debugger* window and the *MGL Viewer*. It is possible to single-step through code from the start of the program using the button on the *MGL Viewer* toolbar. This method can be used to debug code without having to set breakpoints.

Printing and Error Handling

Messages can be displayed from within an MGL program using the `print()` built-in function, e.g.

```
a := 1;
b := 2.5;
print( “a: ”, a, tab, “b: ”, b, cr );
```

The `print()` function can be passed any number of arguments of the following types: `integer`, `float`, `string` or `boolean`. All the arguments are converted to a string representation, concatenated and then displayed. The keywords `tab` and `cr` can also be passed to a print function to print tab and carriage return characters respectively.

Display output is directed to the Figaro transcript (and therefore the Figaro log file), as well as the file `macrogen.lst` in the current project directory. This file is overwritten on each run of the MGL compiler. The above print statement produces the following output.

```
Macro generator: info - a: 1 b: 2.5
```

Error handling is provided by the `error()` function. This can be passed the same type of arguments as the `print()` function as shown below.

```
error( “Width of “, width, “ is too big”);
```

When an `error()` function is called, its arguments are displayed as an error message on the Figaro transcript as follows.

```
Macro generator: *ERROR* - Width of 100 is too big
```

In addition, a pop up box containing the error message warns the user. The MGL program is then aborted with a failed status.

File Input and Output

MGL offers support for basic text file I/O operations. To open a text file for reading, declare a variable of type `file` and then call the pre-defined function `fileopen(),` as shown below.

```
aFile : file;

...

aFile := fileopen( “input.txt”, “read”, “File not  found”);
```

The `fileopen()` function takes up to three arguments. The first is the name of the file to be opened. The second is the mode it is to be opened in, i.e. read, write or append modes. Read mode can be specified with `“R”, “r”, “read”` or `“READ”.` A similar situation applies to the other modes. It is not possible to read from and write to the same file from within MGL. The final argument, which is optional, allows the user to provide an error message if the file cannot be opened. If this is not provided, a default message is shown. When a file cannot be opened, an error dialog box is shown and program execution aborted.

Reading from a File

The pre-defined function `fileread()` is used to read tokens from a text file, e.g.

```
aString : string;
aFile  : file := fileopen( “test.txt”,”r”);

...

aString := fileread( aFile);
```

This function reads the next token from the given file. Any white space character delimits tokens. In the following example, successive calls to `fileread()` would return “decimal”, “0”, “21”, “1”, “43” and so on.

```
decimal
0 21
1 43
2 11
3 68
```

A number of built-in functions exist for converting strings that represent a number (such as those that might be retrieved from a text file) into an integer. When reading the decimal numbers in the above file, for example, the following code fragment can be used.

```
test   : boolean;
aNumber : integer;

...

aString := fileread( aFile);
test := aString isdecimal();
if (test) then
   aNumber := string2decimal(aString);
else
   error( “Number no good!”);
end if;
```

The conversion from a string token to a decimal number is done in 2 stages. First, the `isdecimal()` function is called to verify that a conversion can be made. Then the `string2decimal()` function is called on the string. This returns the integer value formed by interpreting the string as a decimal number. If the conversion fails, 0 is returned. Therefore one should always use `isdecimal()` to ensure correct conversion.

Several other pre-defined functions are provided for conversions from other number representations. These are summarized below.

Function	Return value
`isdecimal(aString)`	Does string represent a decimal number?
`string2decimal(aString)`	Integer formed by considering string as a decimal number
`ishex(aString)`	Does string represent a hexadecimal number?
`string2hex(aString)`	Integer formed by considering string as a hexadecimal number
`isoctal(aString)`	Does string represent an octal number?
`string2octal(aString)`	Integer formed by considering string as an octal number
`isbinary(aString)`	Does string represent a binary number?
`string2bin(aString)`	Integer formed by considering string as a signed binary number
`string2ubin(aString)`	Integer formed by considering string as an unsigned binary number

Writing to a file

Text can be written to a file that has been opened in append or write modes, using the `filewrite()` function. This works in a similar way to the `print` function, except that it takes an extra argument, the file to be written to, as shown below.

```
aFile : file := fileopen("test.txt","w");

...

for i in 1 to 10 loop
    filewrite( aFile, "Here's the number", i, cr);
end for;
```

All the arguments that follow the file variable are converted to strings, concatenated and then written out to the file.

Macro Generation

The end product of a complete MGL program is an object of type `macro`. This object contains all the information necessary to place and route the generated macro. A simple ripple-carry counter generator will be used to illustrate the process of defining a macro.

Macros are defined in two main stages:

- The macro interface
- The macro contents

Interface definition

The first step in defining the counter generator is to declare a variable of type macro to hold the generated macro information as shown below.

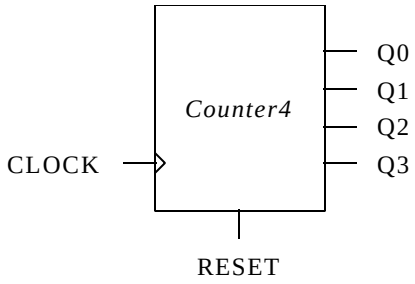
```
counter : macro;
```

Next, the interface of the macro is defined using an *interface block*.

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

If the value of `width` is 4, the above code creates a new macro cell called “Count4” and assigns it to the macro variable counter. The *interface statements* between the keywords `is` and `end interface` completely describe the interface of the new macro. Interface statements can be any valid MGL statements, with the addition of two special interface functions that can only be called from within an interface block. As the names imply, `inputports()` adds input ports of the given name to the macro interface, and `outputports()` adds output ports. A third function, `inoutports()`, is available for specifying bi-directional ports. Because there can be any number of these calls within an interface block, all the ports do not have to be defined from a single call.

The diagram below shows a graphical representation of the macro interface created by the above code.



Macro Symbol

Contents Definition

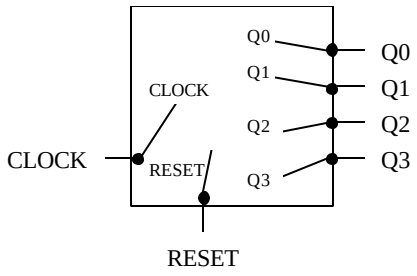
The process of defining the *contents* of a macro (i.e. its underlying implementation) consists of three main tasks:

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

The contents of a macro are defined within a *contents block* as shown in the example below.

```
contents of counter is

    ... statements ...

end contents;
```

When a *contents block* is first entered, a number of nets already exist within the macro contents. They are connected to the external ports previously defined in the *interface block*, and have the same names as those ports. The following nets are present in the counter.



Nets inside Count4

The next step in defining the counter macro is to instantiate some components. There are two ways to instantiate a macro component.

- Call an MGL function to generate the component
- Get the component from a library

Any MGL function that returns a macro object can be reused by other functions. For example, a function that generates FIR filters might call an existing MGL function which generates multipliers, and then instantiate the multipliers within the FIR filter contents. In this way, MGL functions can be reused and macros can be assembled hierarchically.

For this example, the macros will be instantiated from the AT40K library. This is achieved by calling the `getmacro()` function as shown below.

```
aMacro1 : macro := getmacro( “FGEN1RF”);
```

The `getmacro()` function accepts a single string argument corresponding to the name of the component. In the above example, since no library name is specified, the component is checked out of the AT40K vendor library. To get a component from another library, the library name can be specified as follows.

```
aUserMacro : macro := getmacro( “USER:USERCOMP”);
```

The macro component can be instantiated with an *instance block* as shown below.

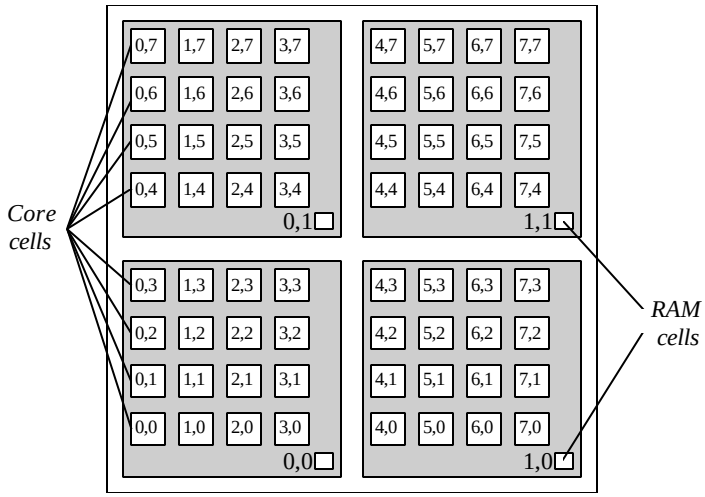
```
instance “Cell0” of aMacro1 is
    location( 0, 0);
    functiong( “FB # B”);
    connections(  “CLK”->“CLOCK”,
                  “RS”->“RESET”,
                  “G”->“Q0” );
    placeports( “G”->“Y”);
    rotation(“R0”);
end instance;
```

An *instance block* contains all the information describing a particular instantiation of the component, such as:

- Instance name
- Location on the FPGA array
- Any properties which are attached to the instance
- How the ports on the instance are connected to nets within the generated macro contents
- How the ports on the instance are placed onto AT40K core cell ports

All this information is specified using `instance` functions that can only be called from within an *interface block*. They do not have to be called in any specific order.

The `location()` function accepts two integer arguments corresponding to the x and y co-ordinates of the location on the FPGA array where the component will be placed. The diagram below illustrates the co-ordinate system that is used. Core cell co-ordinates begin at (0,0) in the bottom-left corner of the array. RAM cells take their co-ordinates from the sector in which they reside. Note that all co-ordinates have 1 added to them when the generated macro is viewed in the IDS Compile View (i.e. the bottom-left core cell is (1,1)). The difference arises because the co-ordinates shown in the Compile View are *absolute* physical co-ordinates, whereas those in MGL are *relative*. Instance co-ordinates in MGL are treated as an *offset* from the bottom-left corner of the *contents block* that contains them. Since there can be multiple levels of hierarchy within a generated macro, the co-ordinates given in the MGL code may have to be translated to arrive at the absolute location relative to the bottom-left of the FPGA array.



MGL Co-ordinate System

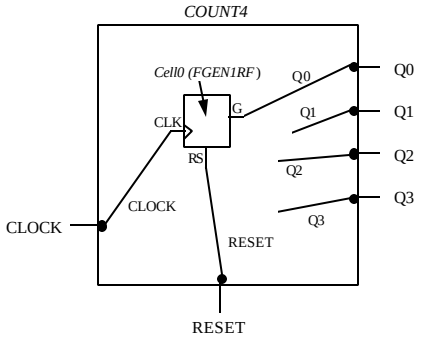
There are several functions that allow properties to be attached to an instance. They are summarized below. Only *dynamic macros*, (AT40K generic macros) can have properties attached to them and only relevant properties can be attached. The compiler will flag an error if a dynamic macro does not expect a particular property.

Function	Description
`functiong(aString);`	**FunctionG equation string, e.g. "A&B+C#D"**
`functionh(aString);`	**FunctionH equation string.**
`clockedge(aString);`	**Clockedge can be either "rising" or "falling".**
`rspolarity(aString);`	**RSPolarity can be either "high" or "low".**
`rsfunction(aString);`	**RSFunction can be either "reset" or "set".**

The format of the equations that are specified using `functiong()` and `functionh()` is exactly the same as those attached to FGEN/MGEN components during design entry. The equation should be expressed in terms of variables A, B, C, D and FB (feedback connection) and can include operators "!" (NOT), "&" (AND), "+" (OR) and "^" (XOR).

Note: If the `functiong()` and `functionh()` equations are not generated within the program, but passed in as arguments from outside, it is necessary to verify the equations' validity. Use `checkfunctiong( aString, errorMessage)` and `checkfunctionh( aString, errorMessage)` to test the relevant equation string. An error message will be returned if the equation is invalid.

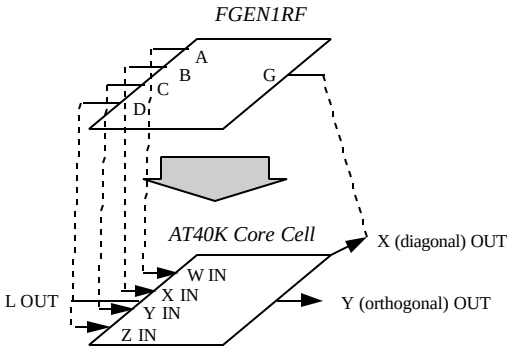
Next, the `connection()` function defines the connectivity of the instance. It accepts `connection` objects as arguments. These arguments define the connections between the nets and each port of the instance. If a net does not already exist, it is created within the generated macro's contents. The diagram below shows how instance `Cell0` is hooked up within the contents of `Count4`.



Instance Block Schematic

The next specialized instance function is called `placeports()`. Like the `connections()` function, it accepts `connection` objects as arguments. These connections specify how the instance's ports are mapped onto the underlying FPGA core cell ports when the instance is placed on the array. Port placements only have to be specified when the default port placements for the macro being instantiated are not suitable.

The diagram below shows the *default* port placements for an FGEN1RF macro.

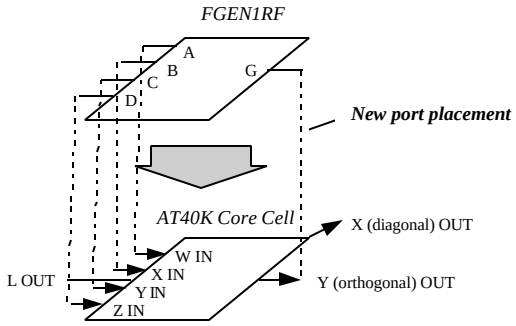


Default Port Placement

Notice that the G output of the macro appears on the X, or diagonal, output of the core cell. To place the instances within the counter macro one above the other, the G output of the FGEN1RF macro must be mapped to a Y core cell output. This is achieved with the following statement.

```
placeports( “G”->“Y”);
```

The resulting macro port to core cell port mapping is shown below.



Macro Port to Core Cell Port Mapping

Finally, the rotation of the instance can be specified using the `rotation()` function. Rotation can only be applied to a macro directly taken from a library. Such a macro will be flat. Macros that contain hierarchy must be rotated explicitly within the code, by altering their internal placement and routing accordingly. Rotation can be specified in 90-degree increments, as either a clockwise (right) rotation or a anti-clockwise (left) rotation. The following calls all produce the same rotation.

```
rotation( “r90”);
rotation( “a270”);
rotation( “c90”);
rotation( “l270”);
```

Net Routing

Once the instances within a *contents block* have been defined, the logical connectivity of the macro is fully described. However, routing information for the macro nets still needs to be specified. This is achieved using a *routing block* as shown below.

```
route of “Q0” is
   nodes( (0, 0, “yOut”), (0, 1, “yIn”));
end route;
```

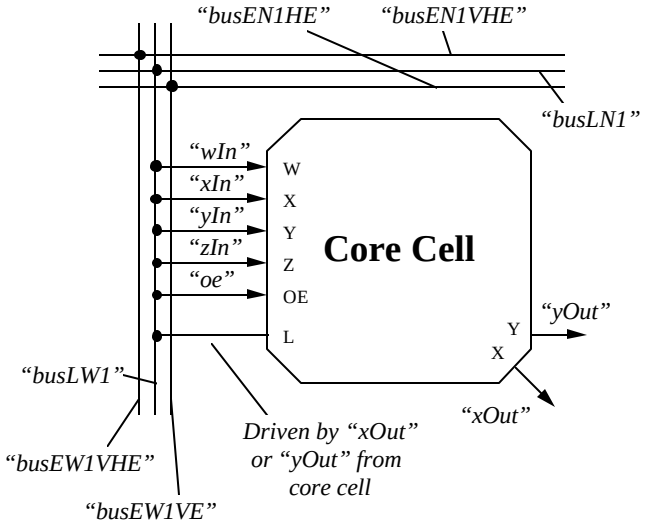
The statements within a routing block specify the physical routing of the named net. The `nodes()` function (a specialized function which can only be called from within a routing block) is used to describe the route the net should take. The `nodes()` function accepts any number of `routenode` objects as arguments. Each `routenode` object describes the location and name of a routing resource that the net should use. The above example is a simple direct connection between the Y output of one core cell and the Y input of the cell above. *Note that the nodes must be listed in order, from source to destination*. For longer, more complex routing, the user has the choice of specifying as many, or as few, routing connections desired. For example, when the user specifies only the source and destination routing nodes, the IDS routing tools will automatically choose a suitable route for the net. On the other hand, if all the nodes are specified, the result is a totally hand-optimized route.

NOTE To route a tri-state net, the function `tristatenodes()` should be used in place of the `nodes()` function shown above.

Complex routing

The example in the previous section illustrated a simple cell-to-cell connection, but routing blocks can also be used to generate more complex routing. Since routing is often the most difficult part of macro generation, it is important to understand the rules that govern how route nodes are applied to a macro net. The diagram below gives an overview of the routing resources that are available to each core cell in the AT40K architecture.

NOTE For simplicity, only the first of the 5 planes of bus routing is shown below.



Core Routing Resources in the AT40K

A core cell has 4 primary inputs - W, X, Y and Z. The W and Z inputs (with node names “wIn” and “zIn”) can only connect to the busing network. The Y input (“yIn”) can connect either to the busing network or to the Y output (“yOut”) of an orthogonally adjacent cell. The X input (“xIn”) can connect either to the busing network or from the X output (“xOut”) of a diagonally adjacent cell.

The OE input controls the tri-state output of the core cell, and connects only to bus plane 5.

The output(s) from a core cell can connect to 3 different resources - the X (diagonal) output, the Y (orthogonal) output or the bus output. A bus connection is made by joining either the X or Y output to the appropriate bus node. For example, if an FGEN1 macro is placed at location (0,0) with its `FUNCTIONG` output placed on the Y output of the core cell, a connection to the vertical local bus is made as follows:

```
nodes( (0, 0, “yOut”), (0, 0, “busLW1”));
```

X and Y outputs cannot be simultaneously connected to a bus.

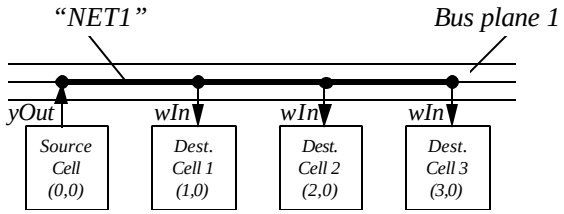
There are two types of local buses available - vertical buses (which lie to the West of each cell) and horizontal buses (which lie to the North). The node names for these are “busLW *x*” and “busLN *x*” respectively, where *x* is the plane number used (from 1 to 5). Each local bus is sandwiched between 2 express buses, as shown in the diagram.

Local buses spans one sector, or 4 core cells. A local bus node can be specified using the co-ordinates of any of the 4 core cells that are associated with it. For example, the plane 1 local bus to the north of cell (0,0) can be specified as (0,0,”busLN1”), (0,1,”busLN1”), (0,2,”busLN1”) or (0,3,”busLN1”). Since express buses span 8 cells, they have 8 cell co-ordinates associated with them.

There are two basic rules that should be followed when routing any net in MGL:

1. Always list routing nodes in order from source to destination.
2. To produce fan-out on a net, route the first branch then *repeat a node that is already attached to the net* before listing the nodes on the second branch. The repeated node is treated as the fan-out point.

   The second rule can be illustrated with the simple example shown below.

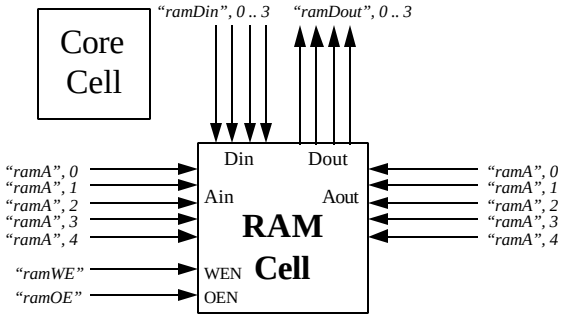


Example of Net Routing with Fan-out

Here the net `NET1` is to be routed from the Source Cell to 3 destination cells. The piece of MGL code that implements this is given below:

```
route of “NET1” is
   nodes( (0, 0, “yOut”));
   for i in 1 to 3 loop
      nodes( (0, 0, “busLN1”), (i, 0, “wIn”));
   end loop;
end route;
```

The bus is repeated on each iteration to show that it is the fan-out point for the net. The routing resources associated with each RAM cell are shown below.

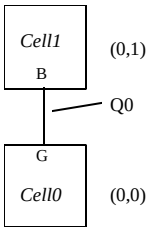


RAM Routing Resources

The co-ordinates of the Ain, Din, Dout, WEN and OEN routing nodes are all derived from the core cell immediately above and to the left of the RAM cell. For example, if the RAM cell location (corresponding to the sector co-ordinate) is (1, 0) the Ain0 node would be (3, 0, "ramA", 0). The co-ordinates of the Aout bus are derived from the core cell immediately above the RAM *one sector to the right*. Using the above example, the Aout1 node would be (7, 0, "ramA", 1). Notice that some of these require a fourth field in the node specification.

Direct Connections

In general, net routing can be specified using *Route Blocks* as illustrated above. Direct cell-to-cell connections, however, are special situations that can be routed in a more concise manner as shown in the following example. The diagram below shows the desired placement and routing for Cell0 and Cell1 of the Count4 macro. Cell1 is to be placed directly above Cell0, with the G output of Cell0 connected to the B input of Cell1 via net Q0. Q0 is to be routed from the Y output of the core cell at location (0,0) to the Y input of the core cell at location (1,1).



Graphical View of Route Block

The standard way to specify the above is shown below.

- Instantiate `"Cell0"`, connect port `"G"` to net `"Q0",` and place port `"G"` on output `"Y"`.
- Instantiate `"Cell1"` and connect port `"B"` to net `"Q0".`
- Route net `"Q0"` with nodes `(0,0,"yOut")` and `(0,1,"yIn").`

However, since this is a direct connection, the two ports can be connected directly, rather than via a net as in the following.

- Instantiate `"Cell0"`, connect port `"G"` to net `"Q0",` and place port `"G"` on output `"Y"`.
- Instantiate `"Cell1"`, connect port `"B"` to port `"G"` on instance `"Cell0",` and place port `"B"` on input `"Y".`

The compiler looks at the port placements specified for the instance ports to be connected, and routes the intervening net accordingly. Port-to-port connections are achieved by calling the `getport()` function as shown below.

```
connections( "B"->getport("Cell0", "G") );
```

The `getports()` function accepts 2 arguments: the first is the name of the instance that contains the port, and the second is the port name. It returns a `port` object.

Design Generation

MGL also supports the programmatic generation of *designs* (which are distinguished from macros by the fact that they can include IOs). The steps in generating a design using MGL are essentially the same as for generating a macro, with the following exceptions:

- A target FPGA device must be specified using the `setdevice()` function.
- One or more IOs must be instantiated.

Selecting a Device

The process of selecting a target FPGA device is illustrated below.

```
fpga : device;
deviceWidth   : integer;

...

fpga := setdevice( “AT40K30-2JI”);
deviceWidth := sizeof( fpga);
```

The first step is to declare a variable of type `device`. This is assigned a value by calling the built-in function `setdevice(),` with a string corresponding to the device (and package) to be used. A by-product of the `setdevice()` call is that the given device is selected as the target FPGA. As the example shows, the `sizeof()` function can be used to get the width of the array. This is useful in cases where the correct number of IOs are required to be instantiated along the edges of a specific package.

Instantiating IOs

IOs are instantiated similar to other macros, except that a number of different properties can be attached to IO macros, and the co-ordinate system used to specify their location is completely different. An example is shown below.

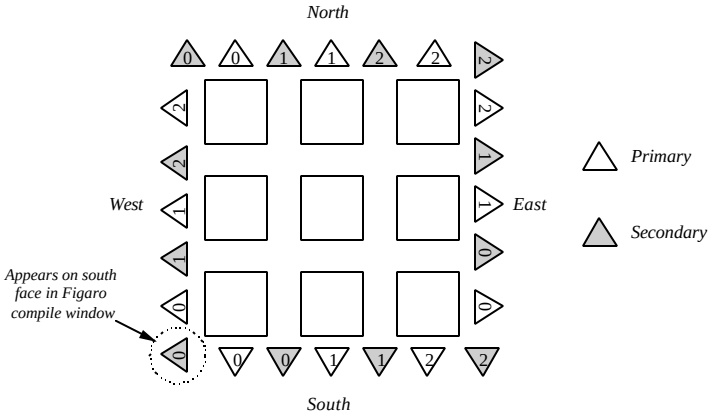
```
anIOB : macro := getmacro( “IBUF”);

...

instance “IN0” of anIOB is
    location( “S”, 0, “Secondary”);
    threshold( “ttl”);
    schmitt( “enable”);
    extradelay( 3);
    connections( “Q”->“INPUT_NET”);
end instance;
```

IO Locations

The location of an IO is uniquely specified with three pieces of information:

1. The array edge it is to be located on. This can be `“n”` (`“north”`), `“s”` (`“south”`), `“e”` (`“east”`) or `“w”` (`“west”`).
2. The offset along that edge, from 0 to (arraysize - 1). This is discussed in more detail below.
3. Whether the instance is a primary (`“p”` or `“primary”`) or secondary (`“s”` or `“secondary”`) IO.

The diagram below shows how IO locations are mapped onto the array edges. It shows a 3X3 array for clarity. Note the discrepancy (in the bottom left corner) between the IO co-ordinate mapping and the FPGA array that is graphically displayed in Figaro.



IO Locations

IO Properties

Properties are attached to IO macros using a number of pre-defined *instance block* functions. These are summarized in the table below.

Function	Description
`slewrate(aString);`	**Sets the slewrate of an output IO. Legal values are "fast" / "f", "medium" / "m" and "slow" / "s".**
`schmitt( aString);`	**For input IOs. Specifies whether Schmitt triggering is enabled. Legal values are "enabled" / "e" and "disabled" / "d".**
`threshold(aString);`	**Threshold of an input IO. Can be "ttl" / "t" or "cmos" / "c".**
`extradelay(aValue);`	**Sets the delay on an input IO. Legal values are 0, 1, 3 and 5, which can be passed to the function either as a string or an integer.**

Testing IO locations

For a given FPGA device and package, IOs may or may not be bonded. It is possible to test whether an IO at a particular location on the currently selected device is bonded by using the `isbonded()` function as shown below.

```
bondtest : boolean;

...

bondtest := isbonded( "south", 3, "secondary");
```

Similarly, it is possible to test if an IO location is bonded on a *specific* part using the `isbondedforpart( location, partName)` function, or on *all* parts using the `isuniversallybonded( location)` function.

In addition to the above, it is possible to test whether the IO at a specified location on the *current* part is a fast clock, a global clock or a global reset, using the following functions:

- isfastclock(location)
- isglobalclock(location)
- isglobalreset(location).

IO Routing

The routing for nets connected to IOs are specified in the same way as for normal macros, except for the node that is attached to the IO itself. This is illustrated in the example below.

```
route of "IOBNet" is
   nodes( ("s", 1, "primary", "out"),
          (0, 1, "yIn") );
end route;
```

The first node on this net has its co-ordinates specified in the same way as IO macros. The wnode names around the edge of the array can be confusing and inconsistent. The user is advised to familiarize oneself with the way the wnodes are organized in the Figaro *Compile* window. For this reason, direct connections should be used to connect to IOs wherever possible by using connections such as that shown below.

```
instance "IN0" of anIOB is
   location( "S", 0, "Secondary");
   connections(getport("G", "CoreCell0"));
end instance;
```

Ripple-Carry Counter Example

The following pages show an example MGL source code listing for a ripple-carry up counter. The generator is parameterized by width and name, and employs two levels of hierarchy.

```
#INCLUDE "global.h"    //  Global constants, useful functions,etc.
#END                   //  Speed up pre-processing a bit

//********************** GLOBAL CONSTANTS *************

constant   CARRYIN     : string := "CARRYIN";
                           // Name of carry-in port
constant   CARRYOUT    : string := "CARRYOUT";
                           // Name of carry-out port
constant   QOUT        : string := "Q";
                           // Counter  stage output

//*****************************************************
// Creates a single up-counter stage, taking into account
// whether it is the first stage, last stage, or a middle stage.
//*****************************************************

function counterCell(  firstCell,
                               // Is this first counter cell?
                       lastCell : boolean
                               // Is this last counter cell?
                   )   :   macro
                               // Return macro for counter cell


   counterCell     : macro;    // Macro to be returned
   fgen            : macro;    // FGEN component that will be
                               // instantiated
   interfaceName   : string;   // Name for the cell's interface


begin

   // First, let's decide what to call the macro that we create
   // (it needs to be different for first, middle or last cells)

   if (firstCell) then
       interfaceName := "FIRST";
   elseif (lastCell) then
       interfaceName := "LAST";
   else
       interfaceName := "MIDDLE";
   end if;


   // Now we'll define the interface for this counter stage. Note
   // that the first stage doesn't have a carry-in, and it only
   // has one output, which we'll call CARRYOUT.
   // Notice also that the last stage cell doesn't have a
   // carry-out port.

   interface interfaceName of counterCell is

       if (NOT firstCell) then
           inputports( "CARRYIN");
           outputports( "Q");
       end if;
       if (NOT lastCell) then
           outputports( "CARRYOUT");
       end if;

       inputports( CLOCK_POSITIVE, RESET_LOW);
       // These constants are found in "global.h"
       // and should be used for all registered macros

   end interface;


   // The contents of the counter stage now has to be defined

   contents of counterCell is


       // First, let's decide whether we need an FGEN2RF or
       // an FGEN1RF. Basically, it'll be an FGEN2RF unless
       // this is the first or
       // last stage (since they only have one output).
```

```
        if (firstCell OR lastCell) then
            fgen := getmacro( "FGEN1RF");
        else
            fgen := getmacro( "FGEN2RF");
        end if;


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

        instance "COUNTINST" of fgen is

            location( 0, 0);    // Relative location within this
                                // contents

            if (firstCell) then // First cell only has 1 output,
                                // which we label CARRYOUT

                functiong( "~FB");
                connections( "G"->CARRYOUT);
            elseif (lastCell) then  // Last cell doesn't have a
                                    // carry out
                functiong( "A#FB");
                connections( "A"->CARRYIN, "G"->QOUT);
            else
                functiong( "A#FB"); // All other cells have 2
                                    // outputs
                functionh( "A&FB");
                connections( "A"->CARRYIN,
                        "G"->QOUT, "H"->CARRYOUT);
            end if;

            // Don't forget to hook up the clock and reset ports

            connections( "CLK"->CLOCK_POSITIVE, "RS"->RESET_LOW);

        end instance;

    end contents;

    return( counterCell);   // We've finished creating the counter
                            // stage, so return it as a macro

end;   // function counterCell()
```

```
//*****************************************************
// Creates a ripple-carry up-counter, parameterized by
// width, and with the given name
//*****************************************************

function UpCounter (name       : string,   // Name of counter
                    width      : integer   // Width of counter
                   )   : macro


   counter :   macro;  // The counter macro that we'll create
   aCell   :   macro;  // Used to hold individual counter stages


begin

   // Kick off by checking that the width isn't too big - we're
   // referring to a global constant stored in "global.h" for
   // the check

   if (width > MAX_ARRAY_WIDTH) then
       error( "Counter width 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
               // single counter stage

           instance "CELL"{i} of aCell is
               location( 0, i);    // Arrange cells vertically

               connections( CLOCK_POSITIVE->CLOCK_POSITIVE,
                       RESET_LOW->RESET_LOW);
                       // Hook up the clock and reset ports

               if (i != width-1) then
                   placeports( CARRYOUT->"Y");                                    // Put the carryout on the core
cell
                   // Y output (ready for direct connection)
               end if;
```

```
                // On all but the first stage, we'll hook up the
                // carry-in port to the carry-out port of the
                // preceding stage, using a getport() call.
                // For this to work correctly, we've got to
                // do a placeports on both the ports being
                // connected

                if (i != 0) then
                    connections( QOUT->QOUT+i);
                    connections( CARRYIN->
                        getport( "CELL"{i-1}, CARRYOUT));
                    placeports( CARRYIN->"Y");
                else
                    connections( CARRYOUT->QOUT + i);
                end if;

            end instance;

        end loop;

    end contents;

    return( counter);

end;   // UpCounter()



//*****************************************************
// PROGRAM BLOCK
// This is where we can call the counter generator for
// testing purposes
//*****************************************************

begin

    return( UpCounter( "count16", 16));
                    // Generate a 16-bit counter

end;
```

MGL Keywords and Built-In Functions

Keywords

AND	anything	are
array	begin	boolean
case	connection	constant
contents	cr	default
device	else	elseif
end	file	float
for	function	if
in	instance	is
loop	macro	mod
NOT	of	OR
return	route	routenode
step	string	tab
then	to	when
while	XOR	

Built-In Functions

Function	Can Be Called From	Description
abs(aNumber)	Anywhere	Returns the absolute value of aNumber
arccos(aNumber)	Anywhere	Returns the arccos of aNumber
arcsin(aNumber)	Anywhere	Returns the arcsin of aNumber
arctan(aNumber)	Anywhere	Returns the arctan of aNumber
ceil(aFloat)	Anywhere	Rounds aFloat up to produce an integer
checkfunctiong(aString, errorMessage)	Instance Block	Checks whether the given function G string is valid for the current instance, and outputs errorMessage if it is not
checkfunctionh(aString, errorMessage)	Instance Block	Checks whether the given function H string is valid for the current instance, and outputs errorMessage if it is not
clockedge(aString)	Instance Block	Sets the clockedge of the current instance (“RISING”/”FALLING”)
connections(connection1, .. connectionN)	Instance Block	Connects instance ports to nets (“PORTNAME” -> “NETNAME”)
cos(aNumber)	Anywhere	Returns the cos of aNumber
error(arg1, arg2, .. argN)	Anywhere	Halts execution of the program and displays an error dialog containing all the arguments (converted to their string representation)
extradelay(aString)	Instance Block	Programs the extra delay (in nanoseconds) for an input IO instance
fileopen(fileName, mode, errorMessage)	Anywhere	Opens file called fileName in given mode (“READ” /”WRITE”/”APPEND”) and outputs errorMessage if the operation fails. Returns file object
fileread(fileObject)	Anywhere	Reads a whitespace-delimited token from fileObject
filewrite(fileObject, arg2, ... argN)	Anywhere	Writes the list of arguments to the given fileObject (using string representation)
floor(aFloat)	Anywhere	Rounds aFloat down to produce an integer
functiong(anEquation)	Instance Block	Attaches the given equation string to an instance, to program its function G LUT
functionh(anEquation)	Instance	Attaches the given equation string to an instance, to program its function H LUT

	Block	
getmacro(aMacroName)	Anywhere	Fetches the given macro from the AT40K library (or a user library if aMacroName is in the form "LIBNAME:COMPNAME"), and returns the corresponding macro object
getport(aCellName, aPortName)	Instance Block	Returns the port object corresponding to port aPortName on instance aCellName
inoutports(portName1, .. portNameN)	Interface Block	Defines the list of port names as In/Out (bi-directional) ports on the macro interface
inputports(portName1, .. portNameN)	Interface Block	Defines the list of port names as input ports on the macro interface
isbinary(aString)	Anywhere	Checks whether aString represents a valid binary number
isbonded(anIOLocation)	Anywhere	Returns whether the IO at the given location is bonded-out on the currently selected part
isbondedforpart(anIOLocation , aPartName)	Anywhere	Returns whether the IO at the given location is bonded-out on part aPartName
isdecimal(aString)	Anywhere	Checks whether aString represents a valid decimal number
isfastclock(anIOLocation)	Anywhere	Returns whether the IO at the given location is a fast clock input on the currently selected part
isglobalclock(anIOLocation)	Anywhere	Returns whether the IO at the given location is a global clock input on the currently selected part
isglobalreset(anIOLocation)	Anywhere	Returns whether the IO at the given location is a global reset input on the currently selected part
ishex(aString)	Anywhere	Checks whether aString represents a valid hexadecimal number
isoctal(aString)	Anywhere	Checks whether aString represents a valid octal number
isuniversallybonded(anIOLocation)	Anywhere	Returns whether the IO at the given location is bonded-out on ALL AT40K parts
largest(num1, num2)	Anywhere	Returns the larger of the two numbers
ln(aNumber)	Anywhere	Returns the natural log of aNumber
location(x, y)	Instance Block	Places the current instance at the given location
log(aNumber)	Anywhere	Returns log to the base 10 of aNumber
log2(aNumber)	Anywhere	Returns log to the base 2 of aNumber
lowercase(aString)	Anywhere	Converts aString to be all lowercase
match(aString, target)	Anywhere	Returns whether aString matches the target expression
nodes(node1, .. nodeN)	Route Block	Attaches the list of routing nodes to the current (non-tristate) net
outputports(portName1, .. portNameN)	Interface Block	Defines the list of port names as output ports on the macro interface
placeports(connection1, .. connectionN)	Instance Block	Specifies the physical pin placement of the given macro ports (e.g. "G"->"Y")

print(arg1, .. argN)	Anywhere	Converts the list of arguments to their string representation, concatenates them and prints them to the IDS transcript window and the design log file
rotation(aString)	Instance Block	Specifies the rotation of the current instance (provided the instance does not contain any hierarchy)
rsfunction(aString)	Instance Block	Programs the RSFUNCTION of the current instance (“RESET”/”SET”)
rspolarity(aString)	Instance Block	Programs the RSPOLARITY of the current instance (“HIGH”/”LOW”)
schmitt(aString)	Instance Block	Programs whether the Schmitt trigger function is enabled for an input IO (“TRUE”/”FALSE”)
setbreak(condition)	Anywhere	Sets a conditional breakpoint in the code. If the (optional) condition is true, execution is halted and the MGL debugger window opened
setdevice(aDeviceName)	Anywhere	Sets the target FPGA device (only used when a design is being generated)
sin(aNumber)	Anywhere	Returns the sin of aNumber
sizeof(anObject)	Anywhere	Returns the size of anObject, where anObject can be a string, an array or a device
slewrate(aString)	Instance Block	Programs the slew rate of an output IO instance (“FAST”/”MEDIUM”/”SLOW”)
smallest(num1, num2)	Anywhere	Returns the smaller of the two numbers
sqrt(aNumber)	Anywhere	Returns the square root of aNumber
string2bin(aString)	Anywhere	Attempts to convert aString to an integer, treating it as a signed binary representation
string2decimal(aString)	Anywhere	Attempts to convert aString to an integer, treating it as a decimal representation
string2hex(aString)	Anywhere	Attempts to convert aString to an integer, treating it as a hexadecimal representation
string2octal(aString)	Anywhere	Attempts to convert aString to an integer, treating it as an octal representation
string2ubin(aString)	Anywhere	Attempts to convert aString to an integer, treating it as an unsigned binary representation
substitute(aString, aSubstring, aReplacement)	Anywhere	Finds all occurrences of aSubstring within aString and replaces them with aReplacement
tan(aNumber)	Anywhere	Returns the tan of aNumber
threshold(aString)	Instance Block	Sets the threshold level of an input IO instance (“TTL”/”CMOS”)
tristatenodes(node1, .. nodeN)	Route Block	Attaches the list of routing nodes to the current tri-state net
trunc(aFloat)	Anywhere	Returns the integer formed by truncating aFloat
uppercase(aString)	Anywhere	Converts aString to be all uppercase

DesignWare Library

This document contains a listing of all components supported by the Atmel DesignWare library. It is organized in a similar fashion as the Synopsys DesignWare Components Databook. This allows users who are already familiar with this environment to easily adapt their design for Atmel AT6000 FPGAs.

For specific information on the macro generators, check the IP Core Generators guide available under SystemDesigner/doc/macrogenerator.pdf

The user should also refer to the “Synthesis-Synopsys” chapter of the *Tutorial* for help in running Synopsys tools using the DesignWare libraries.

AT6K_ABD – Absolut Value

This function creates a macro that generates the absolute value of the parallel input data.

Equivalent Macro Generators Options

Option	Value
Overflow	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	DATA	Width	Parallel input data
Out	RESULT	Width	Absolute value of DATA

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_ABS is
  generic(width : positive);
  port(result : out std_logic_vector(width-1 downto 0);
         data : in std_logic_vector(width-1 downto 0)
       );
end test_AT6K_ABS;

architecture str of test_AT6K_ABS is
begin

-- component instantiation
U0: AT6K_ABS generic map (width => Width)
      port map( RESULT => result, DATA => data);
end str;
```

Verilog

```
module test_AT6K_ABS(result, data);
  parameter width = 16;
  output [width-1:0] result;
  input [width-1:0] data;

  // instantiate AT6K_ABS
  AT6K_ABS # (width) U0 (result, data);
endmodule
```

AT6K_ACC – Accumulator

The Accumulator adds a given number to the registered initial value.

Equivalent Macro Generators Options

Option	Value
Optimize	Area

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	DATA	Width	Parallel input data
In	CIN	1	Carry in
In	ACC	1	Accumulate control
In	CLK	1	Clock
In	R	1	Reset (active low)
Out	SUM	Width	Accumulator output
Out	COUT	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_ACC is
  generic(width : positive);
  port(sum : out std_logic_vector(width-1 downto 0);
         carryout :  out std_logic;
         data : in std_logic_vector(width-1 downto 0);
         carryin, acc, clk, rset : std_logic
        );
end test_AT6K_ACC;

architecture str of test_AT6K_ACC is
begin
  -- component instantiation
  U0: AT6K_ACC generic map (width => Width)
    port map( SUM => sum, COUT => carryout,
                        DATA => data, CIN => carryin, ACC => acc, CLK => clk, R => rset);
end str;
```

Verilog

```
module test_AT6K_ACC(sum, carryout, data, carryin, acc, clk, rset);
  parameter width = 16;
  output [width-1:0] sum, carryout;
  input [width-1:0] data, carryin, acc, clk, rset;

  // instantiate AT6K_ACC
  AT6K_ACC # (width) U0 (sum, carryout, data, carryin, acc, clk, rset);
endmodule
```

AT6K_ACCF – Accumulator

This is a faster version of the Accumulator that adds a given number to the registered initial value.

Equivalent Macro Generators Options

Option	Value
Optimize	Speed

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	DATA	Width	Parallel data input
In	CIN	1	Carry in
In	ACC	1	Accumulate control
In	CLK	1	Clock
In	R	1	Reset (active low)
Out	SUM	Width	Accumulator output
Out	COUT	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_ACCF is
  generic(width : positive);
  port(sum : out std_logic_vector(width-1 downto 0);
         carryout :  out std_logic;
         data : in std_logic_vector(width-1 downto 0);
         carryin, acc, clk, rset : std_logic
        );
end test_AT6K_ACCF;

architecture str of test_AT6K_ACCF is
begin

-- component instantiation
U0: AT6K_ACCF generic map (width => Width)
      port map( SUM => sum, COUT => carryout,
                        DATA => data, CIN => carryin, ACC => acc, CLK => clk, R => rset);
end str;
```

Verilog

```
module test_AT6K_ACCF(sum, carryout, data, carryin, acc, clk, rset);
  parameter width = 16;
  output [width-1:0] sum, carryout;
  input [width-1:0] data, carryin, acc, clk, rset;

  // instantiate AT6K_ACCF
  AT6K_ACCF # (width) U0 (sum, carryout, data, carryin, acc, clk, rset);
endmodule
```

AT6K_ARP – Adder, Ripple Carry

The component is used to create a Ripple Carry Adder. This option creates an area efficient layout.

Equivalent Macro Generators Options

Option	Value
Carryin	Disabled
CarryOut	Disabled
Register	None
Overflow	No
Optimize	Area

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	DATAA	Width	A input
In	DATAB	Width	B input
Out	SUM	Width	Adder output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_ARP is
  generic(width : positive);
  port(sum : out std_logic_vector(width-1 downto 0);
         dataA, dataB : in std_logic_vector(width-1 downto 0)
        );
end test_AT6K_ARP;

architecture str of test_AT6K_ARP is
begin

-- component instantiation
U0: AT6K_ARP generic map (width => Width)
      port map( SUM => sum,
                      DATAA => dataA, DATAB => dataB);
end str;
```

Verilog

```
module test_AT6K_ARP(sum, dataA, dataB);
  parameter width = 16;
  output [width-1:0] sum;
  input [width-1:0] dataA, [width-1:0] dataB;

  // instantiate AT6K_ARP
  AT6K_ARP # (width) U0 (sum, dataA, dataB);
endmodule
```

HDL Usage through Operator Inferencing

VHDL

Use the “+” (addition) operator defined in the Synopsys supplied *std_logic_arith* package to infer an adder. The source listing, *std_logic_arith.vhd,* is located in directory $SYNOPSYS/packages/IEEE/src.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity ADDER is
  generic(Width : POSITIVE);
  port(sum : out std_logic_vector(Width-1 downto 0);
         in1, in2 : in std_logic_vector(Width-1 downto 0)
         );
end ADDER;

architecture impl of ADDER is
  signal in1_signed, ins_signed,
            sum_signed : SIGNED(Width-1 downto 0);
begin
  in1_signed <= SIGNED(in1);
  in2_signed <= SIGNED(in2);

  -- infer the “+” addition operator
  sum_signed <= in1_signed + in2_signed;
  sum <= STD_LOGIC_VECTOR(sum_signed);
end impl;
```

Verilog

Use the standard “+” operator to infer an adder.

```
module ADDER(in1, in2, sum);
  parameter Width = 8;
  input [Width-1:0] in1, in2;
  output[Width-1:0] sum;

  assign sum = in1 + in2;
endmodule;
```

AT6K_ARPCICO – Adder, Ripple Carry

This is a Ripple Carry Adder with carry in and carry out ports.

Equivalent Macro Generators Options

Option	Value
Carryin	NoRegister
CarryOut	NoRegister
Register	None
Overflow	No
Optimize	Area

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	CIN	1	Carry in
In	DATAA	Width	A input
In	DATAB	Width	B input
Out	SUM	Width	Adder output
Out	COUT	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_ARPCICO is
  generic(width : positive);
  port(sum : out std_logic_vector(width-1 downto 0);
         carryout : std_logic;
         dataA, dataB : in std_logic_vector(width-1 downto 0);
         carryin : std_logic
        );
end test_AT6K_ARPCICO;

architecture str of test_AT6K_ARPCICO is
begin

-- component instantiation
U0: AT6K_ARPCICO generic map (width => Width)
      port map( SUM => sum, COUT => carryout,
                        DATAA => dataA, DATAB => dataB, CIN => carryin);
end str;
```

Verilog

```
module test_AT6K_ARPCICO(sum, carryout, dataA, dataB, carryin);
  parameter width = 16;
  output [width-1:0] sum, carryout;
  input [width-1:0] dataA, [width-1:0] dataB, carryin;

  // instantiate AT6K_ARPCICO
  AT6K_ARPCICO # (width) U0 (sum, carryout, dataA, dataB, carryin);
endmodule
```

AT6K_ARPF – Adder, Ripple Carry

This Ripple Carry Adder also has carry in and carry out ports as in AT6K_ARPCICO, but maps to a faster implementation of the layout.

Equivalent Macro Generators Options

Option	Value
Carryin	NoRegister
CarryOut	NoRegister
Register	None
Overflow	No
Optimize	Speed

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	CIN	1	Carry in
In	DATAA	Width	A input
In	DATAB	Width	B input
Out	SUM	Width	Adder output
Out	COUT	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_ARPF is
  generic(width : positive);
  port(sum : out std_logic_vector(width-1 downto 0);
         carryout :  out std_logic;
         dataA, dataB : in std_logic_vector(width-1 downto 0);
         carryin : std_logic
        );
end test_AT6K_ARPF;

architecture str of test_AT6K_ARPF is
begin

-- component instantiation
U0: AT6K_ARPF generic map (width => Width)
      port map( SUM => sum, COUT => carryout,
                        DATAA => dataA, DATAB => dataB, CIN => carryin);
end str;
```

Verilog

```
module test_AT6K_ARPF(sum, carryout, dataA, dataB, carryin);
  parameter width = 16;
  output [width-1:0] sum, carryout;
  input [width-1:0] dataA, [width-1:0] dataB, carryin;

  // instantiate AT6K_ARPF
  AT6K_ARPF # (width) U0 (sum, carryout, dataA, dataB, carryin);
Endmodule
```

AT6K_CMP – Comparator

The function of the Comparator component can be explained as follows:

Equals = (DATAA==DATAB)

LessThan = (DATAA<DATAB)

GreaterThan = (DATAA>DATAB)

Equivalent Macro Generators Options

Option	Value
GreaterThan	Yes
LessThan	Yes
Equality	Yes

Parameters

Parameter	Legal Value	Function
Signed	1	Treat inputs as signed (Default)
	0	Treat inputs as unsigned
Width	Integer > 1	Width of inputs A and B

Pins

Type	Name	Size	Function
In	DATAA	Width	Input data A
In	DATAB	Width	Input data B
Out	AEB	1	1 if A = B, 0 otherwise
Out	ALB	1	1 if A < B, 0 otherwise
Out	AGB	1	1 if A > B, 0 otherwise

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_CMP is
  generic(width : positive);
  port(AeqB, AltB, AgtB :  out std_logic;
         dataA, dataB : in std_logic_vector(width-1 downto 0)
        );
end test_AT6K_CMP;

architecture str of test_AT6K_CMP is
begin

-- component instantiation
U0: AT6K_CMP generic map (width => Width)
      port map( AEB => AeqB, ALB => AltB, AGB => AgtB,
                        DATAA => dataA, DATAB => dataB);
end str;
```

Verilog

```
module test_AT6K_CMP(AeqB, AltB, AgtB, dataA, dataB);
  parameter width = 16;
  output AeqB, AltB, AgtB;
  input [width-1:0] dataA, [width-1:0] dataB;

  // instantiate AT6K_CMP
  AT6K_CMP # (width) U0 (AeqB, AltB, AgtB, dataA, dataB);
Endmodule
```

AT6K_CEQ – Comparator, Equality

The is an Equality Comparator whose output port AEB = 1 when:

DataA == DataB

This function should be used when only Equality is needed as it maps to a much smaller layout than the standard Comparator (AT6K_CMP).

Equivalent Macro Generators Options

Option	Value
None (i.e. use default settings)	None

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Width of inputs A and B

Pins

Type	Name	Size	Function
In	DATAA	Width	Input data A
In	DATAB	Width	Input data B
Out	AEB	1	1 if A = B, 0 otherwise

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_CEQ is
  generic(width : positive);
  port(AeqB :  out std_logic;
         dataA, dataB : in std_logic_vector(width-1 downto 0)
        );
end test_AT6K_CEQ;

architecture str of test_AT6K_CEQ is
begin

-- component instantiation
U0: AT6K_CEQ generic map (width => Width)
      port map( AEQ => AeqB,
                       DATAA => dataA, DATAB => dataB);
end str;
```

Verilog

```
module test_AT6K_CEQ(AeqB, dataA, dataB);
  parameter width = 16;
  output AeqB;
  input [width-1:0] dataA, [width-1:0] dataB;

  // instantiate AT6K_CEQ
  AT6K_CEQ # (width) U0 (AeqB, dataA, dataB);
endmodule
```

AT6K_CJO – Counter, Johnson

This component defines a Johnson Counter.

Equivalent Macro Generators Options

Option	Value
Enable	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	R	1	Reset, active low
In	CLK	1	Clock
Out	Q	Width	Counter output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_CJO is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         clk, rset: std_logic
        );
end test_AT6K_CJO;

architecture str of test_AT6K_CJO is
begin

-- component instantiation
U0: AT6K_CJO generic map (width => Width)
      port map( Q => q,
                      CLK => clk, R => rset);
end str;
```

Verilog

```
module test_AT6K_CJO(q, clk, rset);
  parameter width = 16;
  output [width-1:0] q;
  input clk, rset;

  // instantiate AT6K_CJO
  AT6K_CJO # (width) U0 (q, clk, rset);
endmodule
```

AT6K_CJOEN – Counter, Johnson

This Counter component is equivalent to the Johnson Counter, but also has an enable port to start the counter.

Equivalent Macro Generators Options

Option	Value
Enable	Yes

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	R	1	Reset, active low
In	CLK	1	Clock
In	ENABLE	1	Enable counter
Out	Q	Width	Counter output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_CJOEN is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         clk, rset, enable : std_logic
        );
end test_AT6K_CJOEN;

architecture str of test_AT6K_CJOEN is
begin

-- component instantiation
U0: AT6K_CJO generic map (width => Width)
      port map( Q => q,
                    CLK => clk, R => rset, ENABLE => enable);
end str;
```

Verilog

```
module test_AT6K_CJOEN(q, clk, rset, enable);
  parameter width = 16;
  output [width-1:0] q;
  input clk, rset, enable;

  // instantiate AT6K_CJOEN
  AT6K_CJOEN # (width) U0 (q, clk, rset, enable);
endmodule
```

AT6K_CRC – Counter, Ripple Carry

This is a Ripple Carry Counter component that counts upwards and has a carry out port. See AT6K_CRCD for an equivalent downward counter.

Equivalent Macro Generators Options

Option	Value
Counter	Ripple
Direction	Up
Enable	No
Loadable	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	R	1	Reset (active low)
In	CLK	1	Clock
Out	Q	Width	Counter output
Out	RCO	1	Carry out

sHDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_CRC is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         carryout : out std_logic;
         clk, rset: std_logic
        );
end test_AT6K_CRC;

architecture str of test_AT6K_CRC is
begin

-- component instantiation
U0: AT6K_CRC generic map (width => Width)
      port map( Q => q, RCO => carryout,
                       CLK => clk, R => rset);
end str;
```

Verilog

```
module test_AT6K_CRC(q, carryout, clk, rset);
  parameter width = 16;
  output [width-1:0] q, carryout;
  input clk, rset;

  // instantiate AT6K_CRC
  AT6K_CRC # (width) U0 (q, carryout, clk, rset);
endmodule
```

AT6K_CRCD – Counter, Ripple Carry

This is a Ripple Carry Counter component that counts downwards and has a carry out port. (See AT6K_CRC for an upward counter)

Equivalent Macro Generators Options

Option	Value
Counter	Ripple
Direction	Down
Enable	No
Loadable	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	R	1	Reset (active low)
In	CLK	1	Clock
Out	Q	Width	Counter output
Out	RCO	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_CRCD is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         carryout : out std_logic;
         clk, rset: std_logic
        );
end test_AT6K_CRCD;

architecture str of test_AT6K_CRCD is
begin

-- component instantiation
U0: AT6K_CRCD generic map (width => Width)
      port map( Q => q, RCO => carryout,
                       CLK => clk, R => rset);
end str;
```

Verilog

```
module test_AT6K_CRCD(q, carryout, clk, rset);
  parameter width = 16;
  output [width-1:0] q, carryout;
  input clk, rset;

  // instantiate AT6K_CRCD
  AT6K_CRCD # (width) U0 (q, carryout, clk, rset);
endmodule
```

AT6K_CRCDEN – Counter, Ripple Carry

This is a Ripple Carry Counter component, equivalent to AT6K_CRCD, but with an enable port.

Equivalent Macro Generators Options

Option	Value
Counter	Ripple
Direction	Down
Enable	Yes
Loadable	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	R	1	Reset (active low)
In	CLK	1	Clock
In	ENABLE	1	Enable Counter
Out	Q	Width	Counter output
Out	RCO	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_CRCDEN is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         carryout : out std_logic;
         clk, rset, enable : std_logic
        );
end test_AT6K_CRCDEN;

architecture str of test_AT6K_CRCDEN is
begin

-- component instantiation
U0: AT6K_CRCDEN generic map (width => Width)
      port map( Q => q, RCO => carryout,
                        CLK => clk, R => rset, ENABLE => enable);
end str;
```

Verilog

```
module test_AT6K_CRCDEN(q, carryout, clk, rset, enable);
  parameter width = 16;
  output [width-1:0] q, carryout;
  input clk, rset, enable;

  // instantiate AT6K_CRCDEN
  AT6K_CRCDEN # (width) U0 (q, carryout, clk, rset, enable);
endmodule
```

AT6K_CRC DL– Counter, Ripple Carry

This is a Ripple Carry Counter component, equivalent to AT6K_CRCD, which has parallel data and load ports.

Equivalent Macro Generators Options

Option	Value
Counter	Ripple
Direction	Down
Enable	No
Loadable	Yes

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	R	1	Reset (active low)
In	CLK	1	Clock
In	DATA	Width	Parallel input data
In	SLOAD	1	Load signal (active low)
Out	Q	Width	Counter output
Out	RCO	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_CRCDL is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         carryout : out std_logic;
         clk, rset, load : std_logic
        );
end test_AT6K_CRCDL;

architecture str of test_AT6K_CRCDL is
begin

-- component instantiation
U0: AT6K_CRCDL generic map (width => Width)
      port map( Q => q, RCO => carryout,
                       CLK => clk, R => rset, SLOAD => load);
end str;
```

Verilog

```
module test_AT6K_CRCDL(q, carryout, clk, rset, load);
  parameter width = 16;
  output [width-1:0] q, carryout;
  input clk, rset, load;

  // instantiate AT6K_CRCDL
  AT6K_CRCDL # (width) U0 (q, carryout, clk, rset, load);
endmodule
```

AT6K_CRCDLEN – Counter, Ripple Carry

This Ripple Carry Counter component counts downwards with parallel data, load and counter enable ports.

Equivalent Macro Generators Options

Option	Value
Counter	Ripple
Direction	Down
Enable	Yes
Loadable	Yes

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	R	1	Reset (active low)
In	CLK	1	Clock
In	ENABLE	1	Enable counter
In	DATA	Width	Parallel input data
In	SLOAD	1	Load signal (active low)
Out	Q	Width	Counter output
Out	RCO	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_CRCDLEN is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         carryout : out std_logic;
         clk, rset, load, enable : std_logic
        );
end test_AT6K_CRCDLEN;

architecture str of test_AT6K_CRCDLEN is
begin

-- component instantiation
U0: AT6K_CRCDLEN generic map (width => Width)
      port map( Q => q, RCO => carryout,
                       CLK => clk, R => rset, SLOAD => load, ENABLE => enable);
end str;
```

Verilog

```
module test_AT6K_CRCDLEN(q, carryout, clk, rset, load, enable);
  parameter width = 16;
  output [width-1:0] q, carryout;
  input clk, rset, load, enable;

  // instantiate AT6K_CRCDLEN
  AT6K_CRCDLEN # (width) U0 (q, carryout, clk, rset, load, enable);
endmodule
```

AT6K_CRCEN – Counter, Ripple Carry

This Ripple Counter counts upwards and has an enable option.

Equivalent Macro Generators Options

Option	Value
Counter	Ripple
Direction	Up
Enable	Yes
Loadable	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	R	1	Reset (active low)
In	CLK	1	Clock
In	ENABLE	1	Enable Counter
Out	Q	Width	Counter output
Out	RCO	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_CRCEN is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         carryout : out std_logic;
         clk, rset, enable : std_logic
        );
end test_AT6K_CRCEN;

architecture str of test_AT6K_CRCEN is
begin

-- component instantiation
U0: AT6K_CRCEN generic map (width => Width)
      port map( Q => q, RCO => carryout,
                       CLK => clk, R => rset, ENABLE => enable);
end str;
```

Verilog

```
module test_AT6K_CRCEN(q, carryout, clk, rset, enable);
  parameter width = 16;
  output [width-1:0] q, carryout;
  input clk, rset, enable;

  // instantiate AT6K_CRCEN
  AT6K_CRCEN # (width) U0 (q, carryout, clk, rset, enable);
endmodule
```

AT6K_CRCL – Counter, Ripple Carry

This Ripple Carry Counter component counts upwards and has parallel data and load ports.

Equivalent Macro Generators Options

Option	Value
Counter	Ripple
Direction	Up
Enable	No
Loadable	Yes

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	R	1	Reset (active low)
In	CLK	1	Clock
In	DATA	Width	Parallel input data
In	SLOAD	1	Load signal (active low)
Out	Q	Width	Counter output
Out	RCO	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_CRCL is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         carryout : out std_logic;
         clk, rset, load : std_logic
        );
end test_AT6K_CRCL;

architecture str of test_AT6K_CRCL is
begin

-- component instantiation
U0: AT6K_CRCL generic map (width => Width)
      port map( Q => q, RCO => carryout,
                       CLK => clk, R => rset, SLOAD => load);
end str;
```

Verilog

```
module test_AT6K_CRCL(q, carryout, clk, rset, load);
  parameter width = 16;
  output [width-1:0] q, carryout;
  input clk, rset, load;

  // instantiate AT6K_CRCL
  AT6K_CRCL # (width) U0 (q, carryout, clk, rset, load);
endmodule
```

AT6K_CRCLEN – Counter, Ripple Carry

This Ripple Carry Counter component counts upwards, with parallel data, load and counter enable ports.

Equivalent Macro Generators Options

Option	Value
Counter	Ripple
Direction	Up
Enable	Yes
Loadable	Yes

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	R	1	Reset (active low)
In	CLK	1	Clock
In	ENABLE	1	Enable counter
In	DATA	Width	Parallel input data
In	SLOAD	1	Load signal (active low)
Out	Q	Width	Counter output
Out	RCO	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_CRCLEN is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         carryout : out std_logic;
         clk, rset, load, enable : std_logic
        );
end test_AT6K_CRCLEN;

architecture str of test_AT6K_CRCLEN is
begin

-- component instantiation
U0: AT6K_CRCLEN generic map (width => Width)
      port map( Q => q, RCO => carryout,
                        CLK => clk, R => rset, SLOAD => load, ENABLE => enable);
end str;
```

Verilog

```
module test_AT6K_CRCLEN(q, carryout, clk, rset, load, enable);
  parameter width = 16;
  output [width-1:0] q, carryout;
  input clk, rset, load, enable;

  // instantiate AT6K_CRCLEN
  AT6K_CRCLEN # (width) U0 (q, carryout, clk, rset, load, enable);
endmodule
```

AT6K_CRCUD – Counter, Ripple Carry

This is a Ripple Carry Counter component that has a port to specify the count direction.

Equivalent Macro Generators Options

Option	Value
Counter	Ripple
Direction	UpDown. Up=1
Enable	No
Loadable	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	R	1	Reset (active low)
In	CLK	1	Clock
In	UP_DOWN	1	Up/Down control. Up=1
Out	Q	Width	Counter output
Out	RCO	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_CRCUD is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         carryout : out std_logic;
         clk, rset, up_dn : std_logic
        );
end test_AT6K_CRCUD;

architecture str of test_AT6K_CRCUD is
begin

-- component instantiation
U0: AT6K_CRCUD generic map (width => Width)
      port map( Q => q, RCO => carryout,
                        CLK => clk, R => rset, UP_DOWN => up_dn);
end str;
```

Verilog

```
module test_AT6K_CRCUD(q, carryout, clk, rset, up_dn);
  parameter width = 16;
  output [width-1:0] q, carryout;
  input clk, rset, up_dn;

  // instantiate AT6K_CRCUD
  AT6K_CRCUD # (width) U0 (q, carryout, clk, rset, up_dn);
endmodule
```

AT6K_CRCUDEN – Counter, Ripple Carry

This is a Ripple Carry Counter component with counter enable and direction ports.

Equivalent Macro Generators Options

Option	Value
Counter	Ripple
Direction	UpDown
Enable	Yes
Loadable	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	R	1	Reset (active low)
In	CLK	1	Clock
In	ENABLE	1	Enable counter
In	UP_DOWN	1	Up/Down control. Up=1
Out	Q	Width	Counter output
Out	RCO	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_CRCUDEN is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         carryout : out std_logic;
         clk, rset, up_dn, enable : std_logic
        );
end test_AT6K_CRCUDEN;

architecture str of test_AT6K_CRCUDEN is
begin

-- component instantiation
U0: AT6K_CRCUDEN generic map (width => Width)
      port map( Q => q, RCO => carryout,
                        CLK => clk, R => rset, UP_DOWN => up_dn, ENABLE => enable);
end str;
```

Verilog

```
module test_AT6K_CRCUDEN(q, carryout, clk, rset, up_dn);
  parameter width = 16;
  output [width-1:0] q, carryout;
  input clk, rset, up_dn, enable;

  // instantiate AT6K_CRCUDEN
  AT6K_CRCUDEN # (width) U0 (q, carryout, clk, rset, up_dn, enable);
endmodule
```

AT6K_CRCUDL – Counter, Ripple Carry

This Ripple Carry Counter component has parallel data, load and counter direction ports.

Equivalent Macro Generators Options

Option	Value
Counter	Ripple
Direction	UpDown
Enable	No
Loadable	Yes

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	R	1	Reset (active low)
In	CLK	1	Clock
In	DATA	Width	Parallel input data
In	SLOAD	1	Load signal (active low)
In	UP_DOWN	1	UpDown control. Up=1
Out	Q[Width-1:0]	1	Counter output
Out	RCO	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_CRCUDL is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         carryout : out std_logic;
         clk, rset, up_dn, load : std_logic
        );
end test_AT6K_CRCUDL;

architecture str of test_AT6K_CRCUDL is
begin

-- component instantiation
U0: AT6K_CRCUDL generic map (width => Width)
      port map( Q => q, RCO => carryout,
                        CLK => clk, R => rset, UP_DOWN => up_dn, SLOAD => load);
end str;
```

Verilog

```
module test_AT6K_CRCUDL(q, carryout, clk, rset, up_dn, load);
  parameter width = 16;
  output [width-1:0] q, carryout;
  input clk, rset, up_dn, load;

  // instantiate AT6K_CRCUDL
  AT6K_CRCUDL # (width) U0 (q, carryout, clk, rset, up_dn, load);
endmodule
```

AT6K_CRCUDLEN – Counter, Ripple Carry

This a Ripple Carry Counter component with parallel data, enable, load, and counter direction ports.

Equivalent Macro Generators Options

Option	Value
Counter	Ripple
Direction	UpDown
Enable	Yes
Loadable	Yes

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	R	1	Reset (active low)
In	CLK	1	Clock
In	ENABLE	1	Enable counter
In	DATA	Width	Parallel input data
In	SLOAD	1	Load signal (active low)
In	UP_DOWN	1	Up/Down control. Up=1
Out	Q	Width	Counter output
Out	RCO	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_CRCUDLEN is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         carryout : out std_logic;
         clk, rset, up_dn, load, enable : std_logic
        );
end test_AT6K_CRCUDLEN;

architecture str of test_AT6K_CRCUDLEN is
begin

-- component instantiation
U0: AT6K_CRCUDLEN generic map (width => Width)
      port map( Q => q, RCO => carryout,
                        CLK => clk, R => rset, UP_DOWN => up_dn, SLOAD => load,
                        ENBALE => enable);
end str;
```

Verilog

```
module test_AT6K_CRCUDLEN(q, carryout, clk, rset, up_dn, load, enable);
  parameter width = 16;
  output [width-1:0]q, carryout;
  input clk, rset, up_dn, load, enable;

  // instantiate AT6K_CRCUDLEN
  AT6K_CRCUDLEN # (width) U0 (q, carryout, clk, rset, up_dn, load, enable);
endmodule
```

AT6K_CTR – Counter, Terminal

The TERMCNT pin will go high after the number of clock cycles matches the value present on the DATA pins after the RESET pin is released.

Equivalent Macro Generators Options

Option	Value
None (use default settings)	None

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	R	1	Reset (active low)
In	CLK	1	Clock
In	DATA	Width	Parallel load input
In	SLOAD	1	Load signal (active low)
Out	TERMCNT	1	Terminal Signal

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_CTR is
  generic(width : positive);
  port(count : out std_logic;
         data : in std_logic_vector(width-1 downto 0);
         clk, rset, load : std_logic
        );
end test_AT6K_CTR;

architecture str of test_AT6K_CTR is
begin

-- component instantiation
U0: AT6K_CTRC generic map (width => Width)
      port map( TERM_CNT => count,
                        DATA => data, CLK => clk, R => rset, SLOAD => load);
end str;
```

Verilog

```
module test_AT6K_CTR(count, clk, rset, load);
  parameter width = 16;
  output count;
  input [width-1:0] data, clk, rset, load;

  // instantiate AT6K_CTR
  AT6K_CTR # (width) U0 (count, carryout, clk, rset, load);
endmodule
```

AT6K_DED – Deductor

The Deductor component subtracts a given amount from the register initial value.

Equivalent Macro Generators Options

Option	Value
Optimize	Area

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	CIN	1	Carry in
In	DATA[width-1:0]	Width	Parallel input data
In	ACC	1	Accumulate control
In	CLK	1	Clock
In	R	1	Reset (active low)
Out	SUM	Width	Deductor output
Out	COUT	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_DED is
  generic(width : positive);
  port(sum : out std_logic_vector(width-1 downto 0);
         carryout : out std_logic;
         data : in std_logic_vector(width-1 downto 0);
         carryin, acc, clk, rset: std_logic
        );
end test_AT6K_DED;

architecture str of test_AT6K_DED is
begin

-- component instantiation
U0: AT6K_DED generic map (width => Width)
      port map( SUM => sum, COUT => carryout,
                        DATA => data, CIN => carryin, ACC => acc, CLK => clk, R => rset);
end str;
```

Verilog

```
module test_AT6K_DED(sum, carryout, data, carryin, acc, clk, rset);
  parameter width = 16;
  output [width-1:0] sum, carryout
  input [width-1:0] data, carryin, acc, clk, rset;

  // instantiate AT6K_DED
  AT6K_DED # (width) U0 (sum, carryout, data, carryin, acc, clk, rset);
endmodule
```

AT6K_DEDF – Deductor

This is a faster version of the Deductor component (AT6K_DED). It maps to a larger layout.

Equivalent Macro Generators Options

Option	Value
Optimize	Speed

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	CIN	1	Carry in
In	DATA[width-1:0]	Width	Parallel input data
In	ACC	1	Accumulate control
In	CLK	1	Clock
In	R	1	Reset (active low)
Out	SUM	Width	Deductor output
Out	COUT	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_DEDF is
  generic(width : positive);
  port(sum : out std_logic_vector(width-1 downto 0);
         carryout : out std_logic;
         data : in std_logic_vector(width-1 downto 0);
         carryin, acc, clk, rset: std_logic
        );
end test_AT6K_DEDF;

architecture str of test_AT6K_DEDF is
begin

-- component instantiation
U0: AT6K_DEDF generic map (width => Width)
      port map( SUM => sum, COUT => carryout,
                        DATA => data, CIN => carryin, ACC => acc, CLK => clk, R => rset);
end str;
```

Verilog

```
module test_AT6K_DEDF(sum, carryout, data, carryin, acc, clk, rset);
  parameter width = 16;
  output [width-1:0] sum, carryout
  input [width-1:0] data, carryin, acc, clk, rset;

  // instantiate AT6K_DEDF
  AT6K_DEDF # (width) U0 (sum, carryout, data, carryin, acc, clk, rset);
endmodule
```

AT6K_FFD – Flip Flop, D Type

This is an *n*-bit wide D-type flip-flop register component, where *n* = Width.

Equivalent Macro Generators Options

Option	Value
Enable	No
SetLow	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Widths of parallel input and output data

Pins

Type	Name	Size	Function
In	DATA	Width	Parallel input data
In	CLK	1	Clock Pin
In	R	1	Reset Pin (active low)
Out	Q	Width	Parallel data output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_FFD is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         data : in std_logic_vector(width-1 downto 0);
         clk, rset: std_logic
        );
end test_AT6K_FFD;

architecture str of test_AT6K_FFD is
begin

-- component instantiation
U0: AT6K_FFD generic map (width => Width)
      port map( Q => q,
                      DATA => data, CLK => clk, R => rset);
end str;
```

Verilog

```
module test_AT6K_FFD(q, data, clk, rset);
  parameter width = 16;
  output [width-1:0] q;
  input [width-1:0] data, clk, rset;

  // instantiate AT6K_FFD
  AT6K_FFD # (width) U0 (q, data, clk, rset);
endmodule
```

AT6K_FFDEN – Flip Flop, D Type

This an *n*-bit wide D-type flip-flop register component with parallel data and enable ports, where *n* = Width.

Equivalent Macro Generators Options

Option	Value
Enable	Yes
SetLow	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Widths of parallel input and output data

Pins

Type	Name	Size	Function
In	DATA	Width	Parallel input data
In	CLK	1	Clock Pin
In	ENABLE	1	Enable Data input
In	R	1	Reset Pin (active low)
Out	Q	Width	Parallel data output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_FFDEN is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         data : in std_logic_vector(width-1 downto 0);
         clk, rset, enable: std_logic
        );
end test_AT6K_FFDEN;

architecture str of test_AT6K_FFDEN is
begin

-- component instantiation
U0: AT6K_FFDEN generic map (width => Width)
      port map( Q => q,
                      DATA => data, CLK => clk, R => rset, ENABLE => enable);
end str;
```

Verilog

```
module test_AT6K_FFDEN(q, data, clk, rset, enable);
  parameter width = 16;
  output [width-1:0] q;
  input [width-1:0] data, clk, rset, enable;

  // instantiate AT6K_FFDEN
  AT6K_FFDEN # (width) U0 (q, data, clk, rset, enable);
Endmodule
```

AT6K_FDT – Flip Flop, D Type Tri-state

This an *n*-bit wide D-type flip-flop component with tri-state output, parallel data and enable ports.

Equivalent Macro Generators Options

Option	Value
Enable	No
IndividualEnable	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Widths of parallel input and output data

Pins

Type	Name	Size	Function
In	DATA	Width	Parallel input data
In	CLK	1	Clock Pin
In	OE	1	Individual Output Enable Pins
In	R	1	Reset Pin (active low)
Out	Q	Width	Parallel data output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_FDT is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         data : in std_logic_vector(width-1 downto 0);
         clk, rset, oe: std_logic
        );
end test_AT6K_FDT;

architecture str of test_AT6K_FDT is
begin

-- component instantiation
U0: AT6K_FDT generic map (width => Width)
      port map( Q => q,
                    DATA => data, CLK => clk, R => rset, OE => oe);
end str;
```

Verilog

```
module test_AT6K_FDT(q, data, clk, rset, oe);
  parameter width = 16;
  output [width-1:0] q;
  input [width-1:0] data, clk, rset, oe;

  // instantiate AT6K_FDT
  AT6K_FDT # (width) U0 (q, data, clk, rset, oe);
endmodule
```

AT6K_FDTEN – Flip Flop, D Type Tri-state

This is an *n*-bit wide D-type flip-flop component with Tri-State output, parallel data, data enable and output enable ports.

Equivalent Macro Generators Options

Option	Value
Enable	Yes
IndividualEnable	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Widths of parallel input and output data

Pins

Type	Name	Size	Function
In	DATA	Width	Parallel input data
In	CLK	1	Clock Pin
In	OE	1	individual Output Enable Pins
In	ENABLE	1	Enable data input
In	R	1	Reset Pin (active low)
Out	Q	Width	Parallel data output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_FDTEN is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         data : in std_logic_vector(width-1 downto 0);
         clk, rset, oe, enable: std_logic
        );
end test_AT6K_FDTEN;

architecture str of test_AT6K_FDTEN is
begin

-- component instantiation
U0: AT6K_FDTEN generic map (width => Width)
      port map( Q => q,
                    DATA => data, CLK => clk, R => rset, OE => oe, ENABLE => enable);
end str;
```

Verilog

```
module test_AT6K_FDTEN(q, data, clk, rset, oe, enable);
  parameter width = 16;
  output [width-1:0] q;
  input [width-1:0] data, clk, rset, oe, enable;

  // instantiate AT6K_FDTEN
  AT6K_FDTEN # (width)
 U0 (q, data, clk, rset, oe, enable);
endmodule
```

AT6K_FDTENIEN – Flip Flop, D Type Tri-state

This *n*-bit wide D-type flip-flop component has Tri-State output, parallel data, data enable and individual output enable ports.

Equivalent Macro Generators Options

Option	Value
Enable	Yes
IndividualEnable	Yes

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Widths of parallel input and output data

Pins

Type	Name	Size	Function
In	DATA	Width	Parallel input data
In	CLK	1	Clock Pin
In	OE	Width	Individual Output Enable Pins
In	ENABLE	1	Enable data input
In	R	1	Reset Pin (active low)
Out	Q	Width	Parallel data output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_FDTENIEN is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         data : in std_logic_vector(width-1 downto 0);
         clk, rset: std_logic;
         oe : in std_logic_vector(width-1 downto 0);
         enable : in std_logic
        );
end test_AT6K_FDTENIEN;

architecture str of test_AT6K_FDTENIEN is
begin

-- component instantiation
U0: AT6K_FDTENIEN generic map (width => Width)
      port map( Q => q,
                     DATA => data, CLK => clk, R => rset, OE => oe, ENABLE => enable);
end str;
```

Verilog

```
module test_AT6K_FDTENIEN(q, data, clk, rset, oe, enable);
  parameter width = 16;
  output [width-1:0] q;
  input [width-1:0] data, clk, rset, [width-1:0] oe, enable;

  // instantiate AT6K_FDTENIEN
  AT6K_FDTENIEN # (width) U0 (q, data, clk, rset, oe, enable);
endmodule
```

AT6K_FDTIEN – Flip Flop, D Type Tri-state

This *n*-bit wide D-type flip-flop component has Tri-State output with parallel data and individual output enable ports.

Equivalent Macro Generators Options

Option	Value
Enable	No
IndividualEnable	Yes

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Widths of parallel input and output data

Pins

Type	Name	Size	Function
In	DATA	Width	Parallel input data
In	CLK	1	Clock Pin
In	OE	Width	Individual Output Enable Pins
In	R	1	Reset Pin (active low)
Out	Q	Width	Parallel data output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_FDTIEN is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         data : in std_logic_vector(width-1 downto 0);
         clk, rset: std_logic;
         oe : in std_logic_vector(width-1 downto 0)
        );
end test_AT6K_FDTIEN;

architecture str of test_AT6K_FDTIEN is
begin

-- component instantiation
U0: AT6K_FDTIEN generic map (width => Width)
      port map( Q => q,
                    DATA => data, CLK => clk, R => rset, OE => oe);
end str;
```

Verilog

```
module test_AT6K_FDTIEN(q, data, clk, rset, oe);
  parameter width = 16;
  output [width-1:0] q;
  input [width-1:0] data, clk, rset, [width-1:0] oe;

  // instantiate AT6K_FDTIEN
  AT6K_FDTIEN # (width) U0 (q, data, clk, rset, oe);
endmodule
```

AT6K_FFT – Flip Flop, Toggle

This is an *n*-bit wide toggle flip-flop component.

Equivalent Macro Generators Options

Option	Value
Enable	No
SetLow	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Widths of parallel input and output data

Pins

Type	Name	Size	Function
In	DATA	Width	Parallel input data
In	CLK	1	Clock pin
In	R	1	Reset pin (active low)
Out	Q	Width	Parallel data output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_FFT is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         data : in std_logic_vector(width-1 downto 0);
         clk, rset: std_logic
        );
end test_AT6K_FFT;

architecture str of test_AT6K_FFT is
begin

-- component instantiation
U0: AT6K_FFT generic map (width => Width)
      port map( Q => q,
                      DATA => data, CLK => clk, R => rset);
end str;
```

Verilog

```
module test_AT6K_FFT(q, data, clk, rset);
  parameter width = 16;
  output [width-1:0] q;
  input [width-1:0] data, clk, rset;

  // instantiate AT6K_FFT
  AT6K_FFT # (width) U0 (q, data, clk, rset);
endmodule
```

AT6K_FFTEN – Flip Flop, Toggle

This is an *n*-bit wide toggle flip-flop component with parallel data input and data enable ports.

Equivalent Macro Generators Options

Option	Value
Enable	Yes
SetLow	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Widths of parallel input and output data

Pins

Type	Name	Size	Function
In	DATA	Width	Parallel input data
In	CLK	1	Clock pin
In	ENABLE	1	Enable data input
In	R	1	Reset pin (active low)
Out	Q	Width	Parallel data output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_FFTEN is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         data : in std_logic_vector(width-1 downto 0);
         clk, rset, enable: std_logic
        );
end test_AT6K_FFTEN;

architecture str of test_AT6K_FFTEN is
begin

-- component instantiation
U0: AT6K_FFTEN generic map (width => Width)
      port map( Q => q,
                       DATA => data, CLK => clk, R => rset, ENABLE => enable);
end str;
```

Verilog

```
module test_AT6K_FFTEN(q, data, clk, rset, enable);
  parameter width = 16;
  output [width-1:0] q;
  input [width-1:0] data, clk, rset, enable;

  // instantiate AT6K_FFTEN
  AT6K_FFTEN # (width) U0 (q, data, clk, rset, enable);
endmodule
```

AT6K_GRA – Gray Code

This component serves either as a binary to gray code or gray to binary code.

Equivalent Macro Generators Options

Option	Value
None	None

Parameters

Parameter	Legal Value	Function
ToGray	1	Convert the binary code input to gray-code (default)
	0	Convert the gray-code input to binary code
Width	Integer > 1	Width of input and output data

Pins

Type	Name	Size	Function
In	DATA	Width	Input data to be converted
Out	Q	Width	Converted output data

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_GRA is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         data : in std_logic_vector(width-1 downto 0)
        );
end test_AT6K_GRA;

architecture str of test_AT6K_GRA is
begin

-- component instantiation
U0: AT6K_GRA generic map (width => Width)
      port map( Q => q,
                      DATA => data);
end str;
```

Verilog

```
module test_AT6K_GRA(q, data);
  parameter width = 16;
  output [width-1:0] q;
  input [width-1:0] data;

  // instantiate AT6K_GRA
  AT6K_GRA # (width) U0 (q, data);
endmodule
```

AT6K_LDT – Latch, D Type

This is an *n*-bits wide D-type transparent Latch component with an option to function as a flow-through or latch function.

Equivalent Macro Generators Options

Option	Value
AsynchronousClear	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Width of input and output data

Pins

Type	Name	Size	Function
In	DATA	Width	Data input to the D-type latches
In	GATE	1	Latch enable input (1 = flow-through, 0 = latch)
Out	Q	Width	Data output from D latches

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_LDT is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         data : in std_logic_vector(width-1 downto 0);
         gate : in std_logic
        );
end test_AT6K_LDT;

architecture str of test_AT6K_LDT is
begin

-- component instantiation
U0: AT6K_LDT generic map (width => Width)
      port map( Q => q,
                      DATA => data, GATE => gate);
end str;
```

Verilog

```
module test_AT6K_LDT(q, data, gate);
  parameter width = 16;
  output [width-1:0] q;
  input [width-1:0] data, gate;

  // instantiate AT6K_LDT
  AT6K_LDT # (width) U0 (q, data, gate);
endmodule
```

AT6K_LDTACLR – Latch, D Type

This is an *n*-bits wide D-type transparent Latch component with the AsynchronousClear option.

Equivalent Macro Generators Options

OptionOption	Value
AsynchronousClear	Yes

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Width of input and output data

Pins

Type	Name	Size	Function
In	DATA	Width	Data input to the D-type latches
In	GATE	1	Latch enable input (1 = flow-through, 0 = latch)
In	ACLR	1	Asynchronous clear input
Out	Q	Width	Data output from D latches

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_LDTACLR is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         data : in std_logic_vector(width-1 downto 0);
         gate, aclear : in std_logic
        );
end test_AT6K_LDTACLR;

architecture str of test_AT6K_LDTACLR is
begin

-- component instantiation
U0: AT6K_LDTACLR generic map (width => Width)
      port map( Q => q,
                      DATA => data, GATE => gate, ACLR => aclear);
end str;
```

Verilog

```
module test_AT6K_LDTACLR(q, data, gate, aclear);
  parameter width = 16;
  output [width-1:0] q;
  input [width-1:0] data, gate, aclear;

  // instantiate AT6K_LDTACLR
  AT6K_LDTACLR # (width) U0 (q, data, gate, aclear);
endmodule
```

AT6K_NF – Negate Function

This is a Negate Function component that can be used to generate a two's complement function implemented in a ripple carry manner.

Equivalent Macro Generators Options

Option	Value
Overflow	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output data

Pins

Type	Name	Size	Function
In	DATA	Width	Data input
Out	RESULT	Width	Data output = two's complement of Data

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_NEF is
  generic(width : positive);
  port(result : out std_logic_vector(width-1 downto 0);
         data : in std_logic_vector(width-1 downto 0)
        );
end test_AT6K_NEF;

architecture str of test_AT6K_NEF is
begin

-- component instantiation
U0: AT6K_NEF generic map (width => Width)
      port map( RESULT => result, DATA => data);
end str;
```

Verilog

```
module test_AT6K_NEF(result, data);
  parameter width = 16;
  output [width-1:0] result;
  input [width-1:0] data;

  // instantiate AT6K_NEF
  AT6K_NEF # (width) U0 (result, data);
endmodule
```

AT6K_PSR – Parallel/Serial Register

This is a D-flip flop-based *n*-bits wide Parallel/Serial Register component with shift (for an optional serial input) and shift enable ports.

Equivalent Macro Generators Options

Option	Value
InputOutput	ParallelInSerialOut
Enable	No
SetLow	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Width of parallel input data

Pins

Type	Name	Size	Function
In	DATA	Width	Parallel data input
In	CLK	1	Clock pin
In	R	1	Reset pin (active low)
In	SHIFTIN	1	Serial data input
In	SHIFTEN	1	1 = Enable to function as a shift register, 0 = DFF
Out	SHIFTOUT	1	Serial data output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_PSR is
  generic(width : positive);
  port(shiftout : out std_logic;
         data : in std_logic_vector(width-1 downto 0);
         shiftin, clk, rset, shiftenable : in std_logic
        );
end test_AT6K_PSR;

architecture str of test_AT6K_PSR is
begin

-- component instantiation
U0: AT6K_PSR generic map (width => Width)
      port map( SHIFTOUT => shiftout,
                        DATA => data, SHIFTIN => shiftin, CLK => clk, R => rset,
                        SHIFTEN => shiftenable);
end str;
```

Verilog

```
module test_AT6K_PSR(shiftout, data, shiftin, clk, rset, shiftenable);
  parameter width = 16;
  output shiftout;
  input [width-1:0] data, shiftin, clk, rset, shiftenable;

  // instantiate AT6K_PSR
  AT6K_PSR # (width) U0 (shiftout, data, shiftin, clk, rset, shiftenable);
endmodule
```

AT6K_PSREN – Parallel/Serial Register

This is a D-flip flop -based *n*-bits wide Parallel-to-Serial Register component with shiftin port for an optional serial input and shift enable control port. Additionally it also has a data enable port.

Equivalent Macro Generators Options

Option	Value
InputOutput	ParallelInSerialOut
Enable	Yes
SetLow	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Width of parallel input data

Pins

Type	Name	Size	Function
In	DATA	Width	Data input for InputMode = Parallel
In	CLK	1	Clock pin
In	R	1	Reset pin (active low)
In	ENABLE	1	Data Enable input
In	SHIFTIN	1	Data input for InputMode = Serial
In	SHIFTEN	1	1 = Function as a shift register, 0 = DFF
Out	SHIFTOUT	1	Serial data output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_PSREN is
  generic(width : positive);
  port(shiftout : out std_logic;
         data : in std_logic_vector(width-1 downto 0);
         shiftin, clk, rset, shiftenable, enable : in std_logic
        );
end test_AT6K_PSREN;

architecture str of test_AT6K_PSREN is
begin

-- component instantiation
U0: AT6K_PSREN generic map (width => Width)
      port map( SHIFTOUT => shiftout,
                        DATA => data, SHIFTIN => shiftin, CLK => clk, R => rset,
                        SHIFTEN => shiftenable, ENABLR => enable);
end str;
```

Verilog

```
module test_AT6K_PSREN(shiftout, data, shiftin, clk, rset, shiftenable, enable);
  parameter width = 16;
  output shiftout;
  input [width-1:0] data, shiftin, clk, rset, shiftenable, enable;

  // instantiate AT6K_PSREN
  AT6K_PSREN # (width) U0 (shiftout, data, shiftin, clk, rset, shiftenable, enable);
endmodule
```

AT6K_SPR – Parallel/Serial Register

This is a D-flip flop-based *n*-bits wide Serial-to-Parallel Register component.

Equivalent Macro Generators Options

Option	Value
InputOutput	SerialInParallelOut
Enable	No
SetLow	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Width of parallel output data

Pins

Type	Name	Size	Function
In	CLK	1	Clock pin
In	R	1	Reset pin (active low)
In	SHIFTIN	1	Serial data input
Out	Q	Width	Parallel data output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_SPR is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         shiftin, clk, rset : in std_logic
        );
end test_AT6K_SPR;

architecture str of test_AT6K_SPR is
begin

-- component instantiation
U0: AT6K_SPR generic map (width => Width)
      port map( Q => q,
                      SHIFTIN => shiftin, CLK => clk, R => rset);
end str;
```

Verilog

```
module test_AT6K_SPR(q, shiftin, clk, rset);
  parameter width = 16;
  output [width-1:0] q;
  input shiftin, clk, rset;

  // instantiate AT6K_SPR
  AT6K_SPR # (width) U0 ( q, shiftin, clk, rset);
endmodule
```

AT6K_SPREN – Parallel/Serial Register

This is a D-flip flop-based *n*-bits wide Serial-to-Parallel Register component.

Equivalent Macro Generators Options

Option	Value
InputOutput	SerialInParallelOut
Enable	Yes
SetLow	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Width of parallel input and output data

Pins

Type	Name	Size	Function
In	CLK	1	Clock pin
In	R	1	Reset pin (active low)
In	ENABLE	1	Data Enable input
In	SHIFTIN	1	Serial data input
Out	Q	Width	Parallel data output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_SPREN is
  generic(width : positive);
  port(q : out std_logic_vector(width-1 downto 0);
         shiftin, clk, rset, enable : in std_logic
        );
end test_AT6K_SPREN;

architecture str of test_AT6K_SPREN is
begin

-- component instantiation
U0: AT6K_SPREN generic map (width => Width)
      port map( Q => q,
                    SHIFTIN => shiftin, CLK => clk, R => rset, ENABLE => enable);
end str;
```

Verilog

```
module test_AT6K_SPREN(q, shiftin, clk, rset, enable);
  parameter width = 16;
  output [width-1:0] q;
  input shiftin, clk, rset, enable;

  // instantiate AT6K_SPREN
  AT6K_SPREN # (width) U0 ( q, shiftin, clk, rset, enable);
Endmodule
```

AT6K_SRE – Shift Register

This is a D-flip flop-based *n*-bit Shift Register component.

Equivalent Macro Generators Options

Option	Value
Enable	No
SetLow	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Length of the Shifter

Pins

Type	Name	Size	Function
In	SHIFTIN	1	Serial data input
In	CLK	1	Clock
In	R	1	Reset (active low)
Out	SHIFTOUT	1	Serial data output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_SRE is
  generic(width : positive);
  port(shiftout : out std_logic;
         shiftin, clk, rset : in std_logic
        );
end test_AT6K_SRE;

architecture str of test_AT6K_SRE is
begin

-- component instantiation
U0: AT6K_SRE generic map (width => Width)
      port map( SHIFTOUT => shiftout,
                      SHIFTIN => shiftin, CLK => clk, R => rset);
end str;
```

Verilog

```
module test_AT6K_SRE(shiftout, shiftin, clk, rset);
  parameter width = 16;
  output shiftout;
  input shiftin, clk, rset;

  // instantiate AT6K_SRE
  AT6K_SRE # (width) U0 (shiftout, shiftin, clk, rset);
endmodule
```

AT6K_SREALS – Shift Register

This is a D-flip flop-based *n*-bit Shift Register with active low set mode (where the R port works as Set).

Equivalent Macro Generators Options

Option	Value
Enable	No
SetLow	Yes

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Length of the Shifter

Pins

Type	Name	Size	Function
In	SHIFTIN	1	Serial data input
In	CLK	1	Clock
In	R	1	Asynchronous reset (active low)
Out	SHIFTOUT	1	Serial data output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_SREALS is
  generic(width : positive);
  port(shiftout : out std_logic;
         shiftin, clk, rset : in std_logic
        );
end test_AT6K_SREALS;

architecture str of test_AT6K_SREALS is
begin

-- component instantiation
U0: AT6K_SREALS generic map (width => Width)
      port map( SHIFTOUT => shiftout,
                        SHIFTIN => shiftin, CLK => clk, R => rset);
end str;
```

Verilog

```
module test_AT6K_SREALS(shiftout, shiftin, clk, rset);
  parameter width = 16;
  output shiftout;
  input shiftin, clk, rset;

  // instantiate AT6K_SREALS
  AT6K_SREALS # (width) U0 (shiftout, shiftin, clk, rset);
endmodule
```

AT6K_SREEN – Shift Register

This is a D-flip flop-based *n*-bit Shift Register component with data enable port.

Equivalent Macro Generators Options

Option	Value
Enable	Yes
SetLow	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Length of the Shifter

Pins

Type	Name	Size	Function
In	SHIFTIN	1	Serial data input
In	ENABLE	1	Data Enable input
In	CLK	1	Clock
In	R	1	Reset (active low)
Out	SHIFTOUT	1	Serial data output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_SREEN is
  generic(width : positive);
  port(shiftout : out std_logic;
         shiftin, clk, rset, enable : in std_logic
        );
end test_AT6K_SREEN;

architecture str of test_AT6K_SREEN is
begin

-- component instantiation
U0: AT6K_SREEN generic map (width => Width)
      port map( SHIFTOUT => shiftout,
                        SHIFTIN => shiftin, CLK => clk, R => rset, ENABLR => enable);
end str;
```

Verilog

```
module test_AT6K_SREEN(shiftout, shiftin, clk, rset, enable);
  parameter width = 16;
  output shiftout;
  input shiftin, clk, rset, enable;

  // instantiate AT6K_SREEN
  AT6K_SREEN # (width) U0 (shiftout, shiftin, clk, rset, enable);
endmodule
```

AT6K_SREENALS – Shift Register

This is a D-flip flop-based *n*- shift register component with an enable port and asynchronous active Set control (served by the R port).

Equivalent Macro Generators Options

Option	Value
Enable	Yes
SetLow	Yes

Parameters

Parameter	Legal Value	Function
Width	Integer > 0	Length of the Shifter

Pins

Type	Name	Size	Function
In	SHIFTIN	1	Serial data input
In	ENABLE	1	Data Enable input
In	CLK	1	Clock
In	R	1	Asynchronous reset (active low)
Out	SHIFTOUT	1	Serial data output

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_SREENALS is
  generic(width : positive);
  port(shiftout : out std_logic;
         shiftin, clk, rset, enable : in std_logic
        );
end test_AT6K_SREENALS;

architecture str of test_AT6K_SREENALS is
begin

-- component instantiation
U0: AT6K_SREENALS generic map (width => Width)
      port map( SHIFTOUT => shiftout,
                        SHIFTIN => shiftin, CLK => clk, R => rset, ENABLR => enable);
end str;
```

Verilog

```
module test_AT6K_SREENALS(shiftout, shiftin, clk, rset, enable);
  parameter width = 16;
  output shiftout;
  input shiftin, clk, rset, enable;

  // instantiate AT6K_SREENALS
  AT6K_SREENALS # (width) U0 (shiftout, shiftin, clk, rset, enable);
endmodule
```

AT6K_SBA – Shift, Barrel

This is a Barrel Shifter component that can be used to serve as a combinatorial logic shift/rotate function.

Equivalent Macro Generators Options

Option	Value
Optimize	Area

Parameters

Parameter	Legal Value	Function
Direction	0	Shift/Rotate inputs to the left, lsb to msb (Default)
	1	Shift/Rotate inputs to the right, msb to lsb
Function	1	Shift inputs in direction specified and pads with 0's
	0	Rotate inputs in direction specified (Default)
Width	Integer > 0	Width of input and output data

Pins

Type	Name	Size	Function
In	DATA	Width	Data input to the shifter
In	DISTANCE	Width	Number of positions to shift/rotate
Out	RESULT	Width	Shifted/Rotated data

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_SBA is
  generic(width : positive);
  port(result : out std_logic_vector(width-1 downto 0);
         data : in std_logic_vector(width-1 downto 0);
         distance : in std_logic_vector(width-2 downto 0)
        );
end test_AT6K_SBA;

architecture str of test_AT6K_SBA is
begin

-- component instantiation
U0: AT6K_SBA generic map (width => Width)
      port map( RESULT => result, DATA => data, DISTANCE => distance);
end str;
```

Verilog

```
module test_AT6K_SBA(result, data, distance);
  parameter width = 16;
  output [width-1:0] result;
  input [width-1:0] data, [width-2:0] distance;

  // instantiate AT6K_SBA
  AT6K_SBA # (width) U0 (result, data, distance);
Endmodule
```

AT6K_SUB – Subtractor

This is a Subtractor component with carry-in and carry-out ports.

Equivalent Macro Generators Options

Option	Value
CarryIn	NoRegister
Carryout	NoRegister
Optimize	Area
Register	No
OverFlow	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	CIN	1	Carry in
In	DATAA[Width-1:0]	Width	A input
In	DATAB[Width-1:0]	Width	B input
Out	SUM	Width	Subtractor output
Out	COUT	1	Carry out

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_SUB is
  generic(width : positive);
  port(sum : out std_logic_vector(width-1 downto 0);
         carryout :  out std_logic;
         dataA, dataB : in std_logic_vector(width-1 downto 0);
         carryin : std_logic
        );
end test_AT6K_SUB;

architecture str of test_AT6K_SUB is
begin

-- component instantiation
U0: AT6K_SUB generic map (width => Width)
      port map( SUM => sum, COUT => carryout,
                        DATAA => dataA, DATAB => dataB, CIN => carryin);
end str;
```

Verilog

```
module test_AT6K_SUB(sum, carryout, dataA, dataB, carryin);
  parameter width = 16;
  output [width-1:0] sum, carryout;
  input [width-1:0] dataA, [width-1:0] dataB, carryin;

  // instantiate AT6K_SUB
  AT6K_SUB # (width) U0 (sum, carryout, dataA, dataB, carryin);
endmodule
```

AT6K_SUBF – Subtractor

This Subtractor component is a faster version of AT6K_SUB, but produces a less area-efficient layout.

Equivalent Macro Generators Options

Option	Value
CarryIn	NoRegister
Carryout	NoRegister
Optimize	Speed
Register	No
OverFlow	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	CIN	1	Carry in
In	DATAA[Width-1:0]	Width	A input
In	DATAB[Width-1:0]	Width	B input
Out	SUM	Width	Subtractor output
Out	COUT	1	Carry out (cannot be used with overflow)

HDL Usage through Component Instantiation

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_SUBF is
  generic(width : positive);
  port(sum : out std_logic_vector(width-1 downto 0);
         carryout :  out std_logic;
         dataA, dataB : in std_logic_vector(width-1 downto 0);
         carryin : std_logic
        );
end test_AT6K_SUBF;

architecture str of test_AT6K_SUBF is
begin

-- component instantiation
U0: AT6K_SUBF generic map (width => Width)
      port map( SUM => sum, COUT => carryout,
                        DATAA => dataA, DATAB => dataB, CIN => carryin);
end str;
```

Verilog

```
module test_AT6K_SUBF(sum, carryout, dataA, dataB, carryin);
  parameter width = 16;
  output [width-1:0] sum, carryout;
  input [width-1:0] dataA, [width-1:0] dataB, carryin;

  // instantiate AT6K_SUBF
  AT6K_SUBF # (width) U0 (sum, carryout, dataA, dataB, carryin);
endmodule
```

AT6K_SUBNC – Subtractor

This Subtractor component does not have carry ports, but can be used with the inference operator "-".

Equivalent Macro Generators Options

Option	Value
CarryIn	Disabled
Carryout	Disabled
Optimize	Area
Register	No
OverFlow	No

Parameters

Parameter	Legal Value	Function
Width	Integer > 1	Width of input and output vectors

Pins

Type	Name	Size	Function
In	DATAA	Width	A input
In	DATAB	Width	B input
Out	SUM	Width	Subtractor output

HDL Usage through Component Instantiation

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

enitity test_AT6K_SUBNC is
  generic(width : positive);
  port(sum : out std_logic_vector(width-1 downto 0);
         dataA, dataB : in std_logic_vector(width-1 downto 0)
        );
end test_AT6K_SUBNC;

architecture str of test_AT6K_SUBNC is
begin

-- component instantiation
U0: AT6K_SUBNC generic map (width => Width)
      port map( SUM => sum,
                       DATAA => dataA, DATAB => dataB);
```

end str;

HDL Usage through Operator Inferencing

VHDL

Use the “-” (subtraction) operator defined in the Synopsys supplied *std_logic_arith* package to infer an adder. The source listing, *std_logic_arith.vhd,* is located in directory $SYNOPSYS/packages/IEEE/src.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity SUBTRACTOR is
  generic(Width : POSITIVE);
  port(diff : out std_logic_vector(Width-1 downto 0);
         in1, in2 : in std_logic_vector(Width-1 downto 0)
         );
end SUBTRACTOR;

architecture impl of SUBTRACTOR is
  signal in1_signed, ins_signed,
            diff_signed : SIGNED(Width-1 downto 0);
begin
  in1_signed <= SIGNED(in1);
  in2_signed <= SIGNED(in2);

  -- infer the “-” subtraction operator
  diff_signed <= in1_signed - in2_signed;
  diff <= STD_LOGIC_VECTOR(diff_signed);
end impl;
```

Verilog

Use the standard “-” operator to infer a subtractor.

```
module SUBTRACTOR(in1, in2, diff);
  parameter Width = 8;
  input [Width-1:0] in1, in2;
  output[Width-1:0] diff;

  assign diff = in1 - in2;
endmodule;
```

Cadence

Setup Files

Please refer to the tutorial for information on the various setup files used by IDS.

Command Invocation

IDS uses the following Cadence programs and commands for the

various phases of design integration.

Design Entry

concept DesignName

Example

concept 4bitalu

Functional Simulation

vloglink DesignName

verilog SpeedRange -y UserLibraries -y AtmelLibrary +libext+.v

-v AtmelPrimLib testfixture.template vloglink.v

Example

vloglink 4bitalu

verilog +maxdelays -y /AtUser/4bitalu/user/verilog/at6k02 -y

/atmel/lib/verilog/at6k02 +libext+.v

-v /atmel/lib/verilog/at6k02/prim.v testfixture.template vloglink.v

Macro Generator

Symbol Creation

vloglink MacroName

hdlPinList -v vloglink.v -p TmpFileName

bodyGen -p TmpFileName -b body.1.1

Example

vloglink mult4new

hdlPinList -v vloglink.v -p tmp0

bodyGen -p tmp0 -b body.1.1

Connectivity

redifnet -p -pinMapFileName edifFileName

Example

redifnet -p -/atmel/lib/concept/at6kpin.map mult4new.edf

Netlist

wedifnet -p pinMapFileName -o DesignName.edf

Example

wedifnet -p /atmel/lib/concept/at6kpin.map -o 4bitalu.edf

Post-Layout Simulation

verilog SpeedRange -y AtmelLibrary +libext+.v

-v AtmelPrimLib testfixture.template DesignName.v

Example

verilog +maxdelays -y /atmel/lib/verilog/at6k02 +libext+.v

-v /atmel/lib/verilog/at6k02/prim.v testfixture.template 4bitalu.v

Atmel Libraries

The Atmel FPGA libraries for Cadence are found in the install directory under lib/concept and lib/verilog. The Concept library contains all of the symbols of the Atmel macros. Also in this location is the Concept directory file at6k.lib, which contains the list of the macros used in the Concept Component Browser. Another file found in this directory is the at6kpin.map, which contains a list of all of the macros and their pins. This file is used when creating

the various netlists to translate pin names. The Concept library also contains sub-directories underneath each macro that has the Verilog description of the macro. These files are merely place holders for net list generation and should not be used for simulation.`

Mentor Graphics

This section will detail the file structure included in the AT6000 series Mentor library and the various custom scripts used in the design process. This documentation has been written as a supplement to the Mentor Graphics documentation. For detailed questions about tool functionality, please refer to the Bold Browser or the respective Mentor Graphics documentation.

File Structure

$ATMEL_FPGA/des_arch

This directory contains the Ample code for customized Atmel FPGA library menus in Design Architect.

$ATMEL_FPGA/tool_bin

This directory contains the custom userware for different Mentor tools, which Figaro uses to make the IDS design flow.

$ATMEL_FPGA/cells

This directory contains the interface (Symbol and Schematic) for each component in the Atmel FPGA library.

$ATMEL_FPGA/tables/bin

This directory contains the binary files (compiled Quick Part tables) that specify the functionality of each component in the library.

$ATMEL_FPGA/techfiles/at6k_2/bin
$ATMEL_FPGA/techfiles/at6k_4/bin

These directories contain the binary timing files for the -2 and -4 speed grade components respectively.

$ATMEL_FPGA/autologic

This directory contains the Autologic library models for all the cells in the IDS Macro Library.

The following custom scripts are also provided along with the library kit. IDS uses these custom scripts to invoke the different tools from the push buttons in the Desktop.

atmel_da	Customized script to run Design Architect
atmel_dve	Customized script to run Design Viewpoint Editor
atmel_quicksim	Customized script to run QuickSim
atmel_config	Customized script to run the Design Manager configuration window, used to move design objects between different directories

Please refer to the Mentor Graphics chapter in the "CAE Interfaces" section of the *IDS Tutorial* for the software setup and environment variables.

atmel_da

Design Architect is the Mentor Graphics tool used for design entry. It can be invoked from the Mentor Graphics Design Manager, from the IDS desktop, or through a custom invocation script *atmel_da*, located within */atmel/bin*. All the regular Design Architect command-line options can be used with the invocation script but an option, -generatesym , has been introduced to automatically generate a top-level symbol for the design. The Atmel FPGA library custom menus are available to the user when Design Architect is invoked using this script or from the IDS Desktop.

atmel_da command-line options

atmel_da [*DesignName*] *–switches*

Switch	Comments
-help	Prints all the options available with DA
-usage	This will print the usage message for the script
-generatesym *SymbolName*	Creates the symbol with the specified symbol name for the top-level design

All the options of the Design Architect can be used with *atmel_da*. For more details on the other options please refer to the *Design Architect User's Manual*.

atmel_dve

A Design Viewpoint is a data object used to configure a design. The design configuration includes:

a) Setting values (parameters) for variables that are not resolved within the design hierarchy,
b) Evaluating the property values that contain expressions and CASE statements,
c) Setting the lowest level of hierarchy to which a particular Mentor application should descend,
d) To export/import ASCII back annotation files or objects.

To design using Atmel FPGA library components, a valid Viewpoint is required for any QuickSim simulation and also for netlisting. The tool choice is *atmel_dve* which is the custom invocation of the Design Viewpoint Editor. The simulation tool will be using the Design Viewpoint setting of *Technology Values* (-2 or -4 speed grade). The parameter TECHNOLOGY has to be set to *at6k_2* or *at6k_4* for -2 and -4 parts respectively. Command-line changes have been made to enable the user to create Viewpoints that will be compatible with the Atmel library.

The -help switch for atmel_dve provides the user with the command-line options. This script invokes the Design Viewpoint Editor in nodisplay mode. The IDS Desktop uses this script to create valid Viewpoints for *Functional Simulation* and *Post-layout Simulation*. During *Functional Simulation*, IDS creates a Viewpoint named *prelayout* with the technology parameters set according to the information provided by the user in the *Functional Simulation* dialog box. After placement and routing the ASCII back annotation file created by IDS is read into the Viewpoint Editor for *Post-layout Simulation* and a Viewpoint *postlayout* is created.

atmel_dve Command Line Options

atmel_dve [*DesignName*] *-switches*

Switch	Comments
Design_name	Name of the component
-help	Print usage message
-viewpoint_name or -vpt *ViewpointName*	Name of the Viewpoint to be created or opened
-technology at6k_2 | at6k_4	Adds a parameter, at6k_2 for -2 part or at6k_4 for -4 part
-ba *BaName*	Name of the ASCII back annotation file to be imported
-dofile *FileName*	Executes the DVE commands specified in the file

Once a given Viewpoint for a design is created, it is not necessary to recreate it unless there has been a change in the design hierarchy or an update to the library. The Viewpoint must always be associated with the top-level of the design, since Viewpoints of hierarchical components will not be evaluated or used by the simulator.

atmel_quicksim

This is the custom invocation of QuickSim. No changes to the command-line have been made. Therefore, the -help switch can still be used to gather information about the various switches available to the user. For further details on the command-line invocation of QuickSim, refer to *QuickSim II User's Manual*. The Atmel FPGA library components can be simulated in three different timing modes i.e. *Min*, *Typ*, *Max*, of the simulator. IDS uses this script to invoke QuickSim on the design for *Functional* and *Post-layout Simulation*.

Example

To invoke QuickSim on a Viewpoint in the *Typical* timing mode, the following command can be used:

```
% atmel_quicksim DesignName/ViewpointName -tim typ
```

atmel_config

This is a custom script primarily developed for the purpose of moving Mentor design objects. IDS uses this script for user library management. When a component is checked into the user library, this script moves the Mentor design object to the specified library directory and the references to this component in the current design directory are updated as well.

atmel_config command line options

atmel_config *DesignName -Switches*

Switch	Comments
DesignName	Name of the component to be moved
-help	Prints help message
-lib *LibDir*	Absolute path of the destination directory to which the design has to be moved
-ref *SoftPrefix*	Soft prefix to update the reference of the moved design object
-update *DesignName*	Design in which the references of the checked-in component have to be updated

Synopsys Synthesis

The Synopsys synthesis tool can be used with the Integrated Development System to optimize the layout and design of Atmel FPGAs. Design entry, optimization, simulation, and net list generation are all executed with the Synopsys tool outside of Figaro.

This chapter is written as a supplement to the Synopsys documentation. For detailed questions about tool functionality, please refer to the appropriate Synopsys documentation. Refer to the *User Guide* for more information on the Synopsys-Figaro interface.

Library Setup

The requisite software tools must be installed, and environmental variables set up for the proper integration of the Design Compiler and Figaro.

Software Tools

The following software tools must be installed and running before design synthesis can begin.

1. Synopsys Design Compiler for Verilog or VHDL. This tool must be purchased directly from Synopsys. Optionally the Synopsys FPGA Compiler may be used as well.
2. Atmel Figaro Integrated Development System. The system may be purchased directly from Atmel.
3. Atmel AT6000 macro libraries. Two libraries, for two speed grades, are supplied by Atmel. The at6k002.syn library is provided for -2 (fast speed) grade. The at6k004.syn library is provided for -4 (typical speed) grade. The libraries are shipped with IDS and are documented in the Macro Library I & II manuals.

For further information on installing the software, please refer to the appropriate sections of the respective User Documents.

Software Environment

The .synopsys file supplied as a part of the Synopsys FPGA library contains the environment appropriate to Atmel FPGA synthesis. Keeping this file in the HOME or current working directory will automatically set the Design Compiler to Atmel FPGA synthesis needs at it starts up.

The .synopsys file must be edited, and the search_path variable modified, to include the name of the directory in which the Synopsys library is installed.

A sample .synopsys file with the appropriate comments for -2 and -4 libraries is included below.

```
/* modify the search_path variable below to reflect the location of the synopsys and atmel supplied library files. */

search_path={., /sw/synopsys_3.2a/libraries/syn, /sw/lib/synopsys/synopsys_3.0, /sw/lib/synopsys/com,
AtmelLlibraryLocation }

link_library={at6k002.syn}
/* use following entry for -4 libraries */
/* link_library={at6k004.syn} */

target_library={at6k002.syn}
/* use following entry for -4 libraries */
/* target_library={at6k004.syn} */

/* EDIF output variables */
edifout_netlist_only = "true"
edifout_power_and_ground_representation = "cell"
edifout_merge_libraries = "false"

/* It is recommended not to use these component */
/* due to their excessive cost */
```

```
dont_use at6k002.syn/fjk*
dont_use at6k002.syn/ort*
dont_use at6k002.syn/srpst*
```

Describing and Synthesizing HDL Designs

Begin top down design methodology by describing a design in an HDL such as Verilog or VHDL. To maximize the benefits, creation of a well-partitioned hierarchical top down design is recommended. Keeping a design hierarchical helps to provide efficient optimization of the individual modules making up the complete design. Once the design files are prepared, the "*read*" command can be used to read an HDL design into the Design Compiler environment.

Optimization

The second step consists of optimizing the previously read in design modules. This consists of going through the design hierarchy and optimizing individual modules by setting appropriate constraints. Some constraints to consider are max_area, max_delay, min_delay, set_clock, and max_period. The actual optimization is done using the *compile* command. During optimization, the Design Compiler examines many design alternatives and chooses the design that meets the specified constraints. The amount of effort the Design Compiler uses can be controlled by the *map_effort* switch of the *compile* command. If constraints cannot be satisfied, the Design Compiler outputs the best design and issues appropriate warnings. The *report_constraints* command can be used to collect statistics on the design resources.

Design Optimization Recommendations

The following tips will be useful in creating optimal designs.

1. JK flip flops should be avoided due to their excessive area cost. To prevent the Design Compiler from using JK flip flops, set the *dont_use* attribute on all JK flip flops. It is recommended to insert the *dont_use* command for JK flip flops in the .synopsys file(the default .synopsys file has *dont_use* attribute set on JK flip flops).

2. ORT macros should also be avoided as it and some of the other *dont_use* macros display transient characteristics. Refer to the Macro Library I document for more details. As described above, assign the *dont_use* attribute to prevent its use during synthesis (the default .synopsys file has *dont_use* attribute set on ORT components).

3. Inserting clock and reset buffers. The command *insert_pads* can be used to connect the ports to the pads of the design. The *port_is_pad* attribute must be attached to the ports for the *insert_pads* command to work correctly.

   To connect global clock and global reset signals to pads, two special buffers CLKBUF and RSTBUF are provided in the libraries. These buffers must be attached to global clock and reset pins of the FPGA.

   The Design Compiler can automatically connect CLKBUF to a global clock signal of the FPGA when the command *insert_pads* is used. To attach RSTBUF to a global reset, the following command must be used:

   ```
   set_pad_type -exact rstbuf GlobalReset
   ```

   Attaching CLKBUF and RSTBUF buffers are mandatory to ensure correct back annotation of the delay data during post-layout verification.

Net list Generation

This step writes optimized designs as an EDIF net list for placement and routing, and generates the Verilog net list for pre-layout and post-layout simulation if required. The Design Compiler command "write" can be used to write the design in the output file.

When writing an EDIF net list, use the following command-line options to ensure storage of the net list in a single disk file:

```
-format edif -hierarchy -output DesignName.edf
```

The following attributes must be placed in the synopsys file to produce an EDIF output of the appropriate format.

```
edifout_netlist_only=true
edifout_power_ground_representation=cell
```

To write the net list for *Functional* and *Post-layout Simulation*, the -format *NetlistFormatName* -hierarchy options must be specified.

Refer to the *CAE Interfaces* section in the *IDS Tutorial* for design entry with the Synopsys synthesis tool using the "averager" example.

Library Of Components

All components listed in Atmel's *Macro Library I & II* manuals are included in the library. Even though an attempt was made to fully model all components, the functionality of some components could not be modeled due to restrictions in the Library Compiler. These cells were modeled as "black-box" cells. Black box cells can be instantiated in HDL descriptions but cannot fully participate in the technology mapping and optimization phases due to their missing functionality. Given the small number and complex behavior of such cells, it does not appear that this restriction will affect the quality of the final results. Refer to Synopsys Library Compiler Reference for further information on black box cells.

Cells with the following characteristics are modeled as black-box cells.

1. Cells having two outputs, one combinatorial and the other sequential (e.g. FDHA).
2. Cells having XORing of more than four inputs (e.g. larger parity checkers).
3. Cells having multi-state outputs (e.g. counters).

Interconnect Model

Atmel's FPGAs are unique in their interconnect structure, which is largely comprised of local and express buses communicating with each other over the regenerative elements known as repeaters. The core cell itself can be configured as one wire or a pair of wires, and can act as an efficient and cheap interconnect resource. There is no routing model in the Design Compiler that can truly approximate the routing delay of the Atmel FPGAs. Hence, the at6k002.syn and at6k004.syn libraries do not contain explicit definition of the interconnect model.

Timing Ranges The timing data for **typical** process and temperature variation is used in the libraries.

Retargeting designs to Atmel FPGAs Retargeting is a process of translating a design from one technology to another technology. The following procedure is recommended to target the design from source technology to Atmel FPGAs.

1. Set the *link_library* variable to point to the original source technology library
2. Set the *target_library* variable to point to the FPGA technology library (either at6k002.syn or at6k004.syn).
3. Compile the design using the *compile* command
4. Write the design in an EDIF format using the *write* command.
5. Read the EDIF design into Figaro and perform placement, routing, and bit stream generation.

Synopsys Simulation

Atmel supplies fully validated VITAL'95 compliant VHDL simulation libraries for its AT6000 FPGA s. With these libraries, users can perform pre-layout as well as post-layout simulations with the accuracy and flexibility inherently built into VITAL compliant simulators.

This chapter describes the VHDL library and outlines the synthesis-simulation flows using the Synopsys Design Compiler (or FPGA Compiler) and VSS simulator.

For more information on the VITAL standard, please refer to "VITAL ASIC Modeling Specification", Draft, IEEE 1076.4, October, 1995.

Atmel Libraries

Two libraries are provided for two speed grades. The library for the -2 speed grade (fast part) is called AT6K02 () and the library for the -4 speed grade (standard part) is called AT6K04 .

These libraries are installed in the atmel/lib/synopsys/at6k02 and atmel/lib/synopsys/at6k04 directories.

The Atmel Synopsys synthesis libraries consists of components which can participate in technology mapping as well as those that cannot (black box components).

Components which can participate in synthesis and mapping have VITAL compliant VHDL models. A few important components such as FDHA, FDXOAN3 also have VITAL models even if they cannot participate in synthesis. A list of all components having VITAL models is given at the end of this chapter.

Most components which cannot participate in synthesis and mapping have multiple outputs. Correspondingly, no VITAL models are provided for these components. As such, it is recommended not to instantiate these in the design. If these components are needed in a design, the *Macro Generators* should be used to create the them As a result or running the generators, the simulation models will be built automatically. All black box components in the synthesis library can be created using the *Macro Generators*.

Additional information on the FPGA components can be obtained from the *Macro Library I* & *Macro Library II* manuals that can be accessed via the online document mechanism..

VHDL Generics Defaults

VITAL models use the VHDL *generics* to set the default behavior for handling certain aspects of timing violations. A brief description of the *generics* used in VITAL models and their default values is given below. The default behavior can be overridden by specifying a *generic map* specification at the time of component instantiation.

TimingChecksOn is a boolean switch which enables or disables timing checks. The default is "*true*".

XGenerationOn is a boolean switch which controls X generation and propagation. If it is set to *true*, the timing violations cause X to be propagated on the output pins. The default is "*false*".

InstancePath is a string which determines the full path name of the current instance. It is set to "*".

Glitch Detection and Handling

VITAL models support two algorithms for glitch detection, *GlitchOnDetect* and *GlitchOnEvent*. The *GlitchOnDetect* algorithm reports glitches when they are detected (glitches can be detected several time units ahead ofappearing on the node). This algorithm also takes appropriate corrective action (such as X propagation) at the time of glitch detection.

The *GlitchOnEvent* algorithm reports glitches when they are scheduled to happen. This algorithm takes appropriate corrective action (such as X propagation) at the time glitches are scheduled to happen.

AT6000 VITAL components use the *GlitchOnDetect* algorithm for glitch handling and reporting.

The variable *GlitchMode* is used to control glitch reporting and handling. It can take values *MessagePlusX* (report the glitch and propagate X), *MessageOnly* (only report glitches), *XOnly* (only propagate X) and *NoGlitch* (use INERTIAL delay modelling to handle glitches).

AT6000 VITAL components use *MessageOnly* mode for handling glitches.

Library Setup

.synopsys_dc.setup

In addition to other settings the user may have in the .synopsys_dc.setup file, the following *vhdlout* specific variables should be added to the setup file.

```
vhdlout_write_components = "FALSE"
vhdlout_use_packages ={"IEEE.std_logic_1164", "at6k02.VCOMPONENTS.ALL"}
```

By setting *vhdlout_write_components* to "*false*", the Design Compiler will not insert component declaration statements to the gate level VHDL. Instead, it will insert *library* and *use* statements in the output files and these component declarations will be read by the simulator. The name of the library and that of the package containing component declarations are set using the *vhdlout_use_packages* variable.

Additional information on *vhdlout* variables can be obtained by typing the following command in the Design Compiler Shell:

```
list -variables vhdlout
```

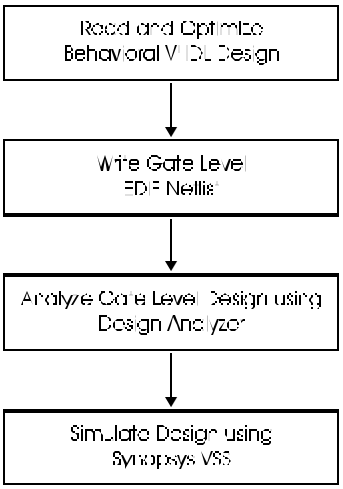
.synopsys_vss.setup

The following two statements should be added to the .synopsys_vss.setup file. They point to the directory locations of the the FPGA libraries (AT6K02 and AT6K04).

```
at6k02 : /atmel/lib/synopsys/at6k02
at6k04 : /atmel/lib/synopsys/at6k04
```

Functional Simulation

The figure below shows the recommended flow for performing Synopsys synthesis and pre-layout simulation of the gate level netlist produced by the synthesis.

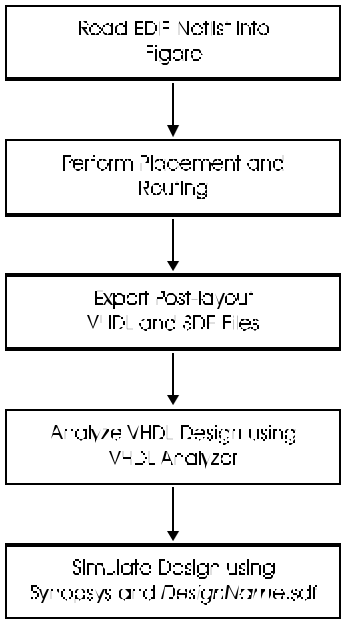


Functional Simulation Flow

Post-layout Simulation

The figure below shows the recommended flow for performing delay annotation and *Post-layout Simulation* using the VSS simulator.

NOTE The post-synthesis netlist used as input for VSS will be different from that generated by Figaro for *Post-layout Simulation*.



Post-layout Simulation Flow

Limitations

The VHDL library is created with typical timing parameters. Therefore, *Pre-layout Simulation* using this library will be restricted to typical timing data. *Post-layout Simulation*, however, is available for minimum, typical, or maximum delay scenarios.

The post-layout netlist is usually flattened in order to support the use use of user or macro generated components, partitioning, and/or mapping. Because of the difference between the post-layout and pre-layout VHDL netlists, post-layout delays cannot be back-annotated to the Design Compiler for design re-synthesis.

Table of Supported Components

Components with VITAL Models				
an2	an2inv	an2l	an2l_2	an2l_3
an2s	an2s_2	an2x	an2x_2	an3
an3_2	an4	an4_2	an5	an5_2
an6	an6_2	an8	an8_2	anxo
ao22	aoi22	bcd7seg	bcmsd	bcmsdf
bcmsdp	bcmsdpf	bcmsoc	bcmsocf	bcmsp
bcmspf	bcocen	bcocfen	bcpen	bcpfen
btocen	btocfen	btpen	btpfen	bttld
bttldf	bttldp	bttldpf	bttloc	bttlocf
bttlp	bttlpf	bufferb	bufz	clkbuf
dc24	dc24h	dc24l	dc38h	dc38l
ec2	ec3	ec4	ec8	en42
en83	en83_2	fa1	fd	fde
fdel	fdmux	fdn	fdor	fdorl
fdr	fds	fdsa	fdsr	fdsr_2
fds_2	fdxo	fdz	fdzq	fjk
fjkq	fjkr	fjkra	fjks	fjksa
fjksr	fjks_2	ft	ftr	ftra
ftr_2	fts	ftsr	fts_2	ha1
hal1	hz	icms	icmsp	inv
invan2	invinv	invz	ittl	ittlp
ld	ldl	ldr	lds	lsrnd
lsrnd_2	lsrnr	lsrnr_2	lz	mc2
mux	mux3	mux4	mux8	mux21
mux21_2	mux31	mux41	mux81	mux_2
nd2	nd3	nd3_2	nd4	nd4_2
nd5	nd5_2	nd6	nd6_2	nd8
nd8_2	ndnd	nr2	nr2l	nr2l_2
nr2_2	nr3	nr3_2	nr4	nr4_2
nr5	nr5_2	nr6	nr6_2	nr8
nr8_2	oai22	oai22_2	od	oden
odf	odfen	odp	odpen	odpf
odpfen	one	oneone	ooc	oocen
oocf	oocfen	op	openb	opf
opfen	or2	or2l	or2l_2	or2l_3
or2_2	or3	or3_2	or4	or4_2
or5	or5_2	or6	or6_2	or8
or8_2	orl	orl_2	ort	ort_2
ort_3	psc1	rstbuf	selbufs	selbufsX
selbufx	selbufxX	srpst	xn2	xn3
xn4	xo2	xo3	xo4	xond
xond3	zero	zeroone	zerozero	fdha
fdnd	fdxoan3	clkedge	clkedgez	lsren
dis7seg				

QuickChange™

The primary motivation behind the development of the QuickChange tools and methodologies is to allow users take advantage of the dynamic reconfiguration or cache logic capabilities of Atmel's AT6000 series FPGAs. These tools and methodologies offer the following advantages.

Reconfigurable FPGA applications can be developed without having to perform most of the time consuming steps needed in traditional application development methodology.

The end application is guaranteed to be correct by construction. It is guaranteed to deliver known timing performance and exhibit identical timing characteristics from one configuration to the next.

The *QuickChange* support of the cache logic capability of the AT6000 device is divided into two parts: static and dynamic. In *Static* cache logic, the part of the design to be changed can be determined ahead of time and is independent of the execution of the design. In *Dynamic* cache logic, the part of the design to be changed cannot be determined ahead of time, but must be resolved as the design progresses.

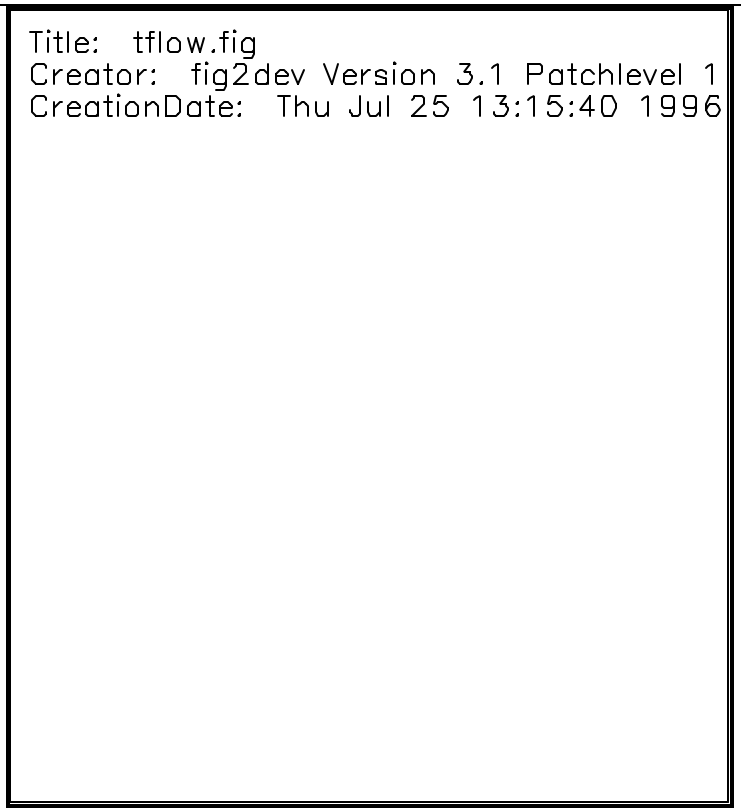
QuickChange and Static Cache Logic

Traditional Design Flow

To understand the benefits of the QuickChange software, it would be useful to review the existing design methodology for design applications involving adaptive computing. Such applications are created in two steps as described below. The user first starts with a base line design and then modifies that base line design to perform a new function.

1. Generation of the configuration bitstream from the base line design.
   - 1.1 Initial design entry.
   - 1.2 Placement and routing of the design entered in step 1.1.
   - 1.3 Generation of the configuration bitstream from the physical data base produced in step 1.2.
2. Generation of the reconfiguration (or partial) bitstream for the new design.
   - 2.1 New design entry (which represents an aspect of the design that needs to be changed by partial reconfiguration).
   - 2.2 Placement and routing of the design entered in step 2.1.
   - 2.3 Generation of the configuration bitstream from the physical database produced in step 2.2.
   - 2.4 Generation of the reconfiguration (or partial) bitstream from bitstreams produced in step 1.3 and 2.3. This partial bitstream only contains the difference between the two bitstreams.

Steps 2.1 to 2.3 can also be carried out by creating a copy of the base line design, making changes to the copied design and performing an ECO on it.



Traditional Design Flow

Unfortunately, the design flow described above has some serious limitations:

- Steps 1.1 to 1.3 consume most of the development effort and have to be duplicated during the creation of the new design. This significantly increases the development effort required to produce the application.
- There is a possibility of accidental errors being introduced when creating the new design. This may cause unintended functional behavior.

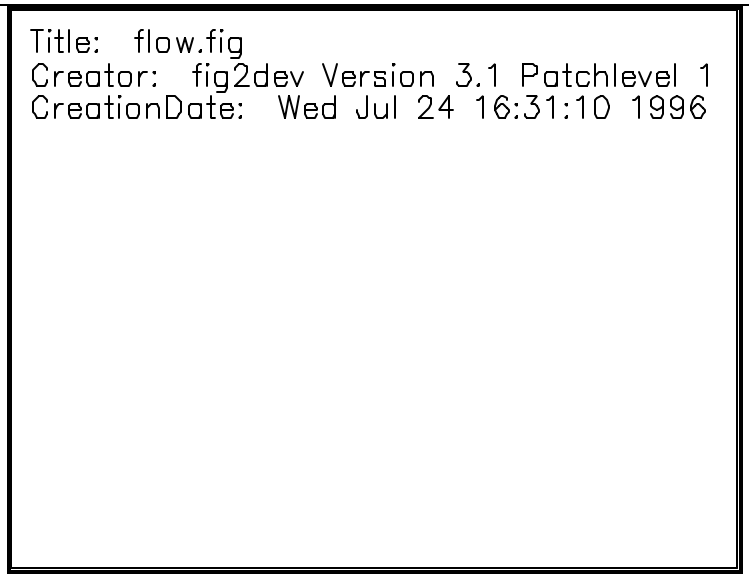
  Re-laying out the new design may alter locations of those components intended to remain unaffected by the reconfiguration. The difference in locations can affect the net lengths between the initial and reconfigured design. This may change timing performance, which is unacceptable in many real time applications.

QuickChange Methodology

Disadvantages of the traditional methodology above are apparent as the designs have to be re-entered to create the reconfiguration bitstream. These disadvantages would disappear if reconfiguration bitstreams could be constructed directly using the *Bitstream Compiler*. The methodology is described and illustrated below.

Design Flow

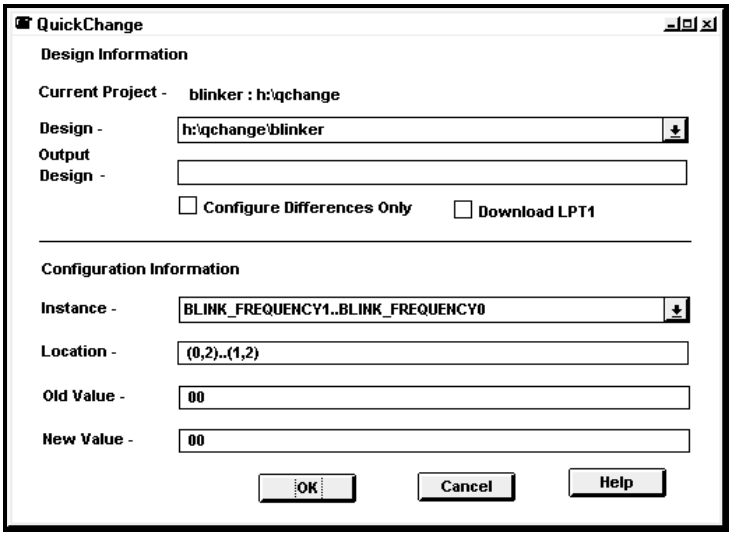
1. Generation of the configuration bitstream from the base line design.
   1.1 Initial design entry.
   1.2 Placement and routing of the design entered in step 1.1.
   1.3 Generation of the configuration bitstream from the physical data base produced in step 1.2.
2. Generation of the reconfiguration (or partial) bitstream.
   2.1 Supplying new configuration information to the bitstream compiler via the graphical user interface (this information represents the changed aspects of the initial design).
   2.2 Compilation of the new bitstream.



QuickChange Design Flow

Graphical User Interface

The figure below shows the graphical user interface designed to call up the *Bitstream Compiler*. Information displayed in the fields *Instance*, *Location*, and *Old Value* (existing configuration) is extracted from the physical data base (layout) of the base line design.



QuickChange User Interface

The new cell configurations should be entered in the field *New Value*. As mentioned previously, the data represents the changed aspects of the initial design.

NOTE

The *New Value* information can be expressed in either binary or hexadecimal format. If specified in hexadecimal, the number must be immediately preceded by an ‘h’.

The name of the reconfiguration (or partial) bitstream should be specified in the *Output Bitstream* field. Once all information is correctly entered, press the [redacted] button and the reconfiguration bitstream will be compiled and placed in the *Output Bitstream* file(with a *.bst file extension added, if not already specified).

Strengths of the QuickChange graphical user interface are:

1. It only allows configuration changes to cell locations that exist in the base line design. Therefore layout re-assembly of the new design is not required.

2. Barring errors from data entry, the reconfiguration bitstream will be correct by construction, ensuring the functional and timing robustness of the reconfigured system.

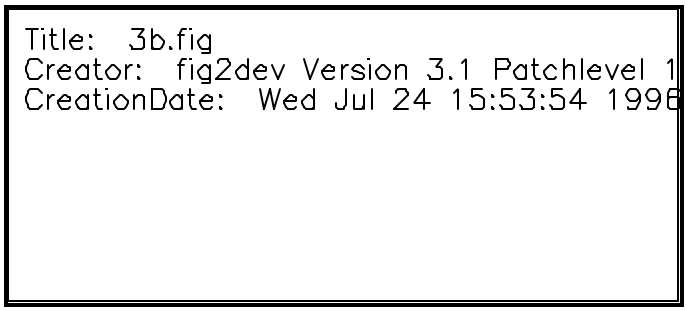
Limitations

Unfortunately, the methodology has some limitations. The user interface can display and modify configurations of constant cells (logic 1 and logic 0) only. Changes to other logic cells are not currently supported. Also the existing user interface does not allow reconfigurations of the routing resources.

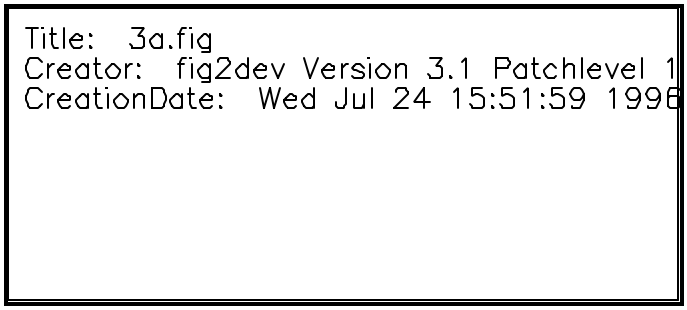
These performance limitations are unrelated to the methodology, and will be addressed in future releases.

Example

The two figures below represent the base line and windowed designs of a DSP system implementing a digital filter. The circuit works as a low pass filter in the initial configuration and reconfigures itself as a high pass filter during system operation. The only difference between these two designs is the values of the coefficients.



Low Pass Filter



High Pass Filter

Assuming that the low pass filter design is already created according to stage 1 in the Design Flow given above, with *QuickChange*, the reconfiguration bitstream for the high pass filter design can be compiled by entering in the new configuration values shown in the *QuickChange* User Interface.

QuickChange and Dynamic Cache Logic

In applications such as adaptive filtering, the filter response must undergo a change at run time, based on the frequency and phase characteristic of the data that is being processed. The operation of such systems, if implemented on an FPGA, would require the software support to create the reconfiguration bitstream on the fly. Additionally, the hardware support must download the constructed bitstream at run time, while the operation of the system is in progress.

The AT6000 series FPGA, being dynamically reconfigurable, is one of few FPGAs capable of supporting such applications. The following outlines the software and hardware support developed for these types of applications.

Software Support

The QuickChange software support for dynamic cache logic (referred to hereafter as dcl software) is intended for use with the C language and is divided into three parts:

- The header file
- The dcl functions
- The bitstream downloading functions

The header file

In order to generate the bitstreams on the fly, the system must have some knowledge of the tie-off cells in the base line design and their coordinates. This information is reported in the header file *design*.h. This file is generated by the bstr program.

This file conforms to C language syntax so that it can be included into C programs written to construct bitstreams.

Listing 1 at the end of this chapter shows a sample *design*.h file. The information contained in this file is described below.

The tie-off cells which are continuous along the X axis are reported as arrays. The tie-off cells which are continuous along the Y axis are reported individually because the memory map of the AT6000 device is discontinuous along that axis. In Listing 1, the X and Y co-ordinates, orientations and values of the tie-off cells are given in the structure BSTR_CONSTS.

The number of tie-off bitstream windows in the design is specified in the variable, WINCNT.

The number of bytes required to store the largest single window in the design is defined in the constant MAXWINSIZE.

The total number of bytes required to store the entire bitstream is defined in BSTRSIZE. This is the maximum storage requirement if the dcl software is to construct the entire bitstream and download it at all once.

For cells that are integrated with bus turn routing, the routing mask information for each cell is stored in the structure CELL_MASKS. Information from this structure is used with the dcl functions to preserve the part of the core cell configuration byte that configures the routing, while allowing changes to be made to that controlling the constant cell.

Other constants are defined to represent the configuration register (CREG), DMA addresses (DMA0, DMA1, DMA2) and configuration mode (MODE). This information is required for constructing the reconfiguration bitstream.

dcl routines

The dcl routines are written in ANSI C language to help in the construction of programs to generate bitstreams. They should be included in the main routine of the program used to create the bitstreams.

A brief description of each of the routines is provided below. Additional details are given in the source code listings under the Atmel software directory /atmel/lib/dcl.

[1]

```
void
window(char* configurations,
unsigned char xStart,
unsigned char yStart,
unsigned char xEnd,
unsigned char yEnd,
unsigned char *win)
```

This routine constructs the bitstream window for the array of tie-off cells starting at (xStart, yStart) and ending at (xEnd, yEnd) in the array win. The storage for win must be pre-allocated. The number of bytes of storage required can be determined using the function winBytes. The new configuration information is passed in the array of characters pointed to by configurations (suc as "101001" etc.). The start and end addresses must correspond to cells which are continuous along the X axis. Also, the row boundaries should not be staggered (the y coordinates should remain constant).

[2]

```
void
window_masked(char* configurations,
unsigned char xStart,
unsigned char yStart,
unsigned char xEnd,
unsigned char yEnd,
unsigned char *mask,
unsigned char *win)
```

This routine is similar to the window routine given above, except the routing mask information is used during bitstream construction. The mask information is supplied as an input to this routine and comes from the mask structure placed in the *design*.h files.

[3]

```
void
oneCell(char tieoff,
unsigned char x,
unsigned char y,
unsigned char *win)
```

This routine constructs the bitstream window for a single cell in win. The storage for win must be pre-allocated. The number of bytes required to be allocated can be determined using the function winBytes. The configuration information is passed in tieoff as either "0" or "1". The cell location is given by the (x,y) co-ordinates.

[4]

```
void
oneCell_masked(char tieoff,
unsigned char x,
unsigned char y,
unsigned char mask,
unsigned char *win)
```

This routine is similar to the oneCell routine, except the routing mask information is used during the bitstream construction. The mask information is supplied as an input to this routine and comes from the mask structure placed in the *design*.h files.

[5]

```
void
header(unsigned char mode,
unsigned char creg,
unsigned char dma0,
unsigned char dma1,
unsigned char dma2,
unsigned char wincnt,
unsigned char *win)
```

This routine constructs the bitstream header in memory win. The storage for win must be pre-allocated. The number of bytes required for storage can be determined using function winBytes.

[6]

```
void
postambles(unsigned char *pambles)
```

This routine constructs the trailer of the bitstream in memory pambles. Two bytes of storage space must be allocated for pambles.

[7]

```
short
winBytes(unsigned char xstart,
unsigned char ystart,
unsigned char xend,
unsigned char yend)
```

This routine returns the number of bytes in a window starting at (xStart, yStart) and ending at (xEnd, yEnd).

Bitstream Downloading Routines

Routines for downloading bitstreams to the FPGA should be written by the user to suit the needs of their application.

dcl software organization

In addition to these routines, a header file dcl.h is provided. This file contains C-function prototypes for all dcl routines, and other additional information such as bitstream header sizes for different configuration modes. This header file is given in listing 2 at the end of this chapter.

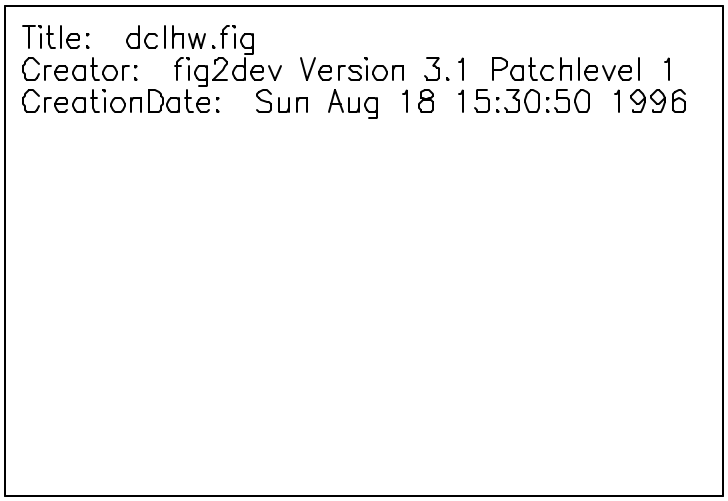
Memory allocation policy is independent of dcl routines. Users can choose their preferred way of allocating memory that is most efficient for their application, hardware or compilers. All dcl routines assume that the memory is pre-allocated (see individual function descriptions).

Routines for creating a one cell window and multi-cell windows and those used to program cell only and cell with routing are separated for efficiency reasons.

Hardware Support

The dcl routines and main program defined by the user (which calls the dcl routines) must be compiled to form the final executable that would then run on the host processor. This processor acts as the configuration controller, the task of which is to construct the bitstream and download it to the FPGA. Any system can be used so long as processor specific compilers exist to compile the dcl routines.

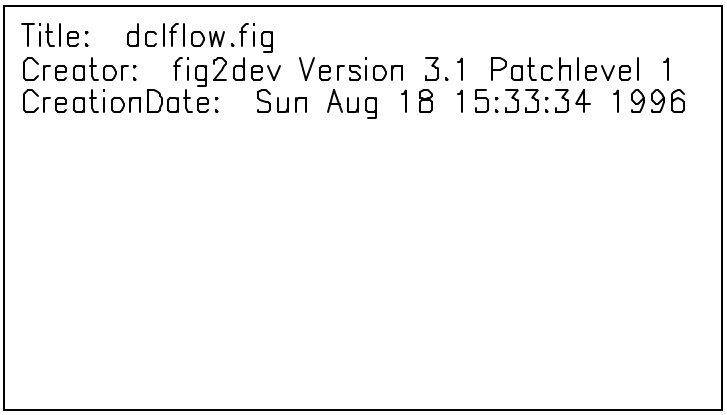
Micro-controllers make an ideal choice. A sample minimal configuration between the AT6000 FPGA (configured in Mode 6) and an AT89C51 micro-controller is shown below.



Micro-controller Interface to AT6000

Information on the various modes of FPGA programming, their associated signals and timing characteristics may be obtained from “Configurable Logic Data Book” published by Atmel.

The figure below shows the hardware-software design flow for creating dynamic configuration bitstreams.



QuickChange support of Dynamic Cache Logic Hardware-Software Design Flow

Example Files

Three sample files are shown in the following pages. Listing 1 shows a *design*.h file created by the bitstream program. Listing 2 illustrates the dcl functional prototypes, while Listing 3 demonstrates one way of incorporating the dcl routines into a typical program using the *design*.h file from Listing 1.

```
************************************************
                                          Listing 1
************************************************

#define HORIZONTAL 0
#define CELL 1

struct BSTR_CONSTS {
unsigned char orientation, xStart, yStart, Xend, yEnd;
char *values;
} bstr_cnsts [] = {
{CELL, 10,7,10,7,"1"},
{CELL, 10,8,10,8,"0"},
{CELL, 10,9,10,9,"1"},
{CELL, 10,10,10,10,"0"},
{HORIZONTAL,15,3,18,3,"1010"},
{CELL,1,14,1,14,"1"},
{CELL,5,13,5,13,"1"},
{CELL,5,24,5,24,"1"},
{CELL,24,5,24,5,"1"},
{CELL,24,13,24,13,"1"},
{CELL,31,7,31,7,"1"},
};
#define WINCNT 11
#define MAXWINSIZE 18
#define BSTRSIZE 147
#define CREG 12
#define DMA0 0
#define DMA1 0
#define DMA2 0
#define MODE 3

struct CELL_MASKS {
unsigned char x, y, mask;
} masks [] = {
{15, 3, 0},
{16, 3, 0},
{17, 3, 0},
{18, 3, 0},
{24, 5, 0},
{10, 7, 0},
{31, 7, 0},
{10, 8, 0},
{10, 9, 0},
{10, 10, 0},
{5, 13, 0},
{24, 13, 0},
{1, 14, 0},
{5, 24, 0},
{1, 14, 0},
{5, 13, 0},
{5, 24, 0},
{10, 7, 0},
{10, 8, 0},
{10, 9, 0},
{10, 10, 0},
{15, 3, 0},
{16, 3, 0},
{17, 3, 0},
{18, 3, 0},
{24, 5, 0},
{24, 13, 0},
{31, 7, 0},
};
```

```
*********************************************

                                        Listing 2

*********************************************

#include <stdio.h>

/* function prototypes */

void oneCell(char tieoff,unsigned char x, unsigned char y, unsigned char *win);

void oneCell_masked(char tieoff,unsigned char x, unsigned char y, unsigned char mask, unsigned
char *win);

void window(char* configurations,unsigned char xStart,
unsigned char yStart, unsigned char xEnd, unisgned char yEnd, unsigned char *win);

void window_masked(char* configurations,unsigned char xStart, unsigned char yStart,
unsigned char xEnd, unsigned char yEnd, unsigned char *mask, unsigned char *win);

void postambles(unsigned char *stg);

void header(unsigned char mode, unsigned char creg, unsigned char dma0, unsigned char dma1,
unsigned char dma2,unsigned char wincnt,
unsigned char *win);

short winBytes(unsigned char unsigned char, unsigned char ystart, unsigned char xend, unsigned
char yend);



/* hash-defines */

#define HSIZE_M1 6
#define HSIZE_M2 6
#define HSIZE_M3 7
#define HSIZE_M4 7
#define HSIZE_M5 6
#define HSIZE_M6 7

#define BSIZE 9
```

```
*********************************************
                                        Listing 3
*********************************************

#include "dcl.h"
#include <stdlib.h>
#include "design.h"
#include <8051.h>

int
main(int argc, char **argv)
{
unsigned char *mem; int oneCellwinSize;
int multiCellwinSize;
int i;
int size = 0;
/* User-define function for downloading FPGA bitstream data */
int downloadm6(unsigned char *mem, int size);

    /* initialize control pins of FPGA */
    /* assumes mode 6 configuration */

    P0_BITS.B3 = 1; /* deactivate CON */
    P0_BITS.B2 = 1; /* deactivate CS */
    P0_BITS.B4 = 0; /* set CCLK to low */

    /* other bits can be initialized */
    /* based on capabilities of the  */
    /* download routine */

    /* Construct and download the header */
    mem = (unsigned char *) malloc(MAXWINSIZE);
    header(MODE, CREG, DMA0, DMA1,DMA2,2, mem);downloadm6(mem, HSIZE_M3);

    /* Construct and download multi cell window */
    multiCellwinSize = winBytes(15,3,18,3);
    window ("1111",15,3,18,3, mem);

    downloadm6(mem, multiCellwinSize);

    /* Construct and download one cell windows */
    oneCellwinSize = winBytes(1,14,1,14);

    oneCell('1',1, 14, mem);

    downloadm6(mem, oneCellwinSize);
    oneCell('1',5, 13, mem);

    downloadm6(mem, oneCellwinSize);
    oneCell('1',5,24, mem);

    downloadm6(mem, oneCellwinSize);
    oneCell('1',24, 5, mem);
    downloadm6(mem, oneCellwinSize);
    oneCell('1',24,13, mem);

    downloadm6(mem, oneCellwinSize);
    oneCell('1',31, 7, mem);

    downloadm6(mem, oneCellwinSize);

    /* postamble */
    postambles(mem);

    downloadm6(mem, 2);
}
```

PLA Optimization

PLA2Cdb reads information embedded in the comments to determine interface pins and pin types during synthesis. Release 6.00 of PLA2CDB can only understand PLA files produced by the ABEL DATA I/O environment, DATA I/O's Synario FPGA synthesis system, and the CUPL compiler. PLA files produced by other compilers are not supported directly. If other PLA formats need to be supported, please modify the PLA file header in accordance with the comment information included in any of the supported PLA formats or contact Atmel customer service for assistance.

IDS can optimize the PLA formatted file generated by ABEL or CUPL according to speed or size specifications. The resulting component can be turned into a hard or soft macro for use in the circuit, or checked into the user library for future applications.

To prepare data for the *PLA Optimize* module, users of ABEL 5.10 should run the Ahdl2Blf compiler with the -pla option to produce a *.tt1 file. There is no provision for specifying the -pla option within the ABEL environment during Ahdl2Blf compilation.

After entering the design with a text editor, the ABEL or CUPL compiler is invoked to produce either a *.tt1, *.tt2, or *.pla file. A brief explanation of the extensions is shown below.

PLA File Extension	Description
*.tt1	ABEL unoptimized *.pla file
*.tt2	ABEL optimized *.pla file
*.pla	CUPL .pla file

PLA Files for Figaro Interface

The following section describes how to specify Atmel FPGA specific features in the ABEL high level descriptions. For more information on the language, refer to the DATA I/O *ABEL Design Software User Manual.*

Notes to ABEL 5.X users

In version 5.1 ABEL, a *.tt2 file can be generated by setting the PLA file option in the *Xfer>Translate Options* dialog box and pressing the *Xfer>Translate* button.

The ABEL compiler from version 5.1 onwards does not produce a *.tt1 file. If AND-XOR optimization is desired, a *.tt2 file can be renamed as *.tt1 before invoking the PLA optimization user interface.

Register Types, Extensions, and their Mapping

Many architecture specific language options of ABEL HDL can be mapped to the components and their pins in Atmel's macro library. The following table outlines the supported options and provides a brief description for each of them. Unsupported features will cause errors when ABEL design files are processed by the PLA2Cdb program. Refer to the "Program Messages" section of the *Technical Reference* manual for more information.

Extension	Description
D	This extension identifies the D input of a D type flip-flop.
T	This extension identifies the T input of a T type flip-flop.
LE	This extension identifies the associated pin as an active low pin of a latch. It is mapped to the EN pin of a latch component.
LH	This extension identifies the associated pin as an active high pin of a latch. It is mapped to the EH pin of a component.
OE	This extension identifies the associated pin as an output enable signal of a tri-state buffer. It is mapped to an OE pin of a D type flip-flop.
CE	This extension identifies the clock enable input of a gated-clock flip-flop. It is mapped to the EN pin of the associated flip-flop.
SR	This extension identifies the associated pin as a synchronous reset of a register. It is mapped to a RN pin of the associated register.
SP	This extension identifies the associated pin as a synchronous preset of a register. It is mapped to a SN pin of the registers.
AP	This extension identifies the associated pin as an asynchronous set pin of a register. It is mapped to the R pin of the associated flip-flop.
AR	This extension identifies the associated pin as a synchronous reset pin of a register. It is mapped to the R pin of the associated flip-flop.
CLK	This extension identifies the associated pin as a clock input of a register and maps to the CLK pin of flip-flops. Use of derived clock signals is not allowed.
PIN	This extension identifies the pin feedback of a register. It is equivalent to FB and COM extensions.
COM	This extension identifies combinatorial feedback. It is equivalent to PIN and COM extensions.
FB	This extension identifies register feedback. It is equivalent to PIN and COM extensions.

Supported ABEL Extensions

Design components supported in ABEL HDL mapping

Registers and latches supported in ABEL to *.cdb conversions are listed below. The PLA2CDB program will exit with error if the register specifications in ABEL designs are not mapped to any of the components listed in the table below. This table should be used as a reference when specifying registers and latches in ABEL designs.

Component	Dot Extension	Macro Name
FD	D, AR	D flip-flop
FDE	D, CE, AR	D flip-flop synchronous with enable
FDZ	D, OE	D flip-flop with tri-state out
FDR	D, SR, AR	D flip-flop with reset low
FDS	D, SP, AR	D flip-flop with set low
FDSA	D, AP	D flip-flop with asynchronous set low
FDSR	D, SP, SR, AR	D flip-flop with set-reset low
LD	D, LH	D latch transparent high
LDL	D, LE	D latch transparent low
FT	T	T flip-flop
FTSR	T, SP, SR, AR	T flip-flop set-reset low
FTR	T, SR, AR	T flop-flop synchronous set low
FTS	T, SP, AR	T type flip-flop synchronous

Supported ABEL extensions

In addition to the extensions mentioned above, extensions Q, PIN and FB can be used to specify flip-flop outputs. The extension PIN is not supported for the macro FDZ.

It is recommended that FD components or components of the same type be used in ABEL designs. Designs created with this guideline will effectively utilize area and clock columns resulting in improved performance.

Logic gates and output polarity

In addition to the register components listed in the table above, there is support for mapping the istype xor language option, which preserves XOR gates without performing their AND-OR expansion. It is strongly recommended that this language option be used whenever possible. These structures are mapped directly to XOR gates that are more efficient in the Atmel FPGA.

The istype reg, invert attribute on register outputs is completely ignored. Instead, the polarity of the register output is largely determined implicitly by the component chosen to map the reg declaration. For example, in register declaration signal reg1 'istype reg', if a register could be mapped to FD (based on other extensions associated with reg1), the output is available in a positive polarity. However, if it could be mapped to FDSA, then the output is available in a negative polarity.

The polarity specification on combinatorial signals is supported completely by IDS.

CUPL

By default, IDS software is set up to read PLA files produced by DATA I/O's ABEL environment. Users have a choice of overriding this default and setting up IDS to read PLA files produced by CUPL. This section briefly describes the compilation and optimization issues involved in supporting CUPL descriptions within IDS.

Compiling CUPL in DOS

There is no direct support to invoke the DOS based CUPL environment from within IDS. The recommended procedure to enter CUPL descriptions is outlined below:

1. Set up *Design* in IDS.
2. Invoke DOS shell by selecting *Open Shell Window* from the vertical menu bar.
3. Enter the CUPL description of the design using a file named *DesignName*.pld. Enter device name as "virtual" in the device statement in the CUPL description. Type the following command to compile the CUPL description to a PLA file.

   CUPL -b *DesignName*.pld

   After successful completion of this command, the *Design Directory* should contain the file *DesignName*.pla.
4. Return to IDS by typing "exit" at the *Shell* prompt.

Specifying *Device Virtual* in the device statement and compiling CUPL with the -b option is mandatory. PLA files produced by options other than -b for other types of devices will result in errors in the program.

Supported CUPL Extensions

The following table shows a list of supported CUPL extensions and their description. Use of extensions other than those specified below will result in unexpected behavior in program execution. It is recommended that users edit the *.pla files to ensure that .ilb and .ob statements contain signal names having the extensions specified below.

Extension	Description
.D	D input of D type flip-flop
.T	T input of T type flip-flop
.DQ	Q output of D type flip-flop
.AP	Asynchronous preset of flip-flop
.AR	Asynchronous reset of flip-flop
.SP	Synchronous preset of flip-flop
.SR	Synchronous reset of flip-flop
.OE	Output enable

Supported CUPL Extensions

Verilog Coding Guidelines

Every synthesis system has a unique way of recognizing the logical modules described using behavioral statements of an HDL design. The task of translating these components into a target technology (a task performed by a technology mapper) can be made more efficient by writing HDL designs with components in the technology library in mind. Writing a design in a technology specific coding style results in synthesis that is very efficient in terms of area and speed, and still provides the technology independence often required for design modification and migration.

This application brief describes a style for writing Verilog statements for components that are most efficient for Atmel FPGAs. Writing the Verilog code as suggested will help the synthesis tools to identify and preserve these components in the output netlist.

For details on an individual synthesis tool, users are encouraged to consult the appropriate sections in the IDS documentation and tutorial sessions provided for the specific system.

General Guidelines

A few important synthesis considerations are listed below. In addition to these guidelines, information specific to a particular synthesis system is given in the appropriate section of the IDS documentation.

- It is strongly recommended to assign Active Low Polarity to a global reset signal and Positive Edge Trigger to a global clock signal.
- Use of JK flip-flops and transparent latches is discouraged due to their excessive area cost.
- Use of many derived clock and reset signals is discouraged. It complicates the placement of flip-flops, resulting in clocks and resets getting routed on the general purpose interconnect, and eventually degrades the performance of the design.
- CLKBUF and RSTBUF pads should be connected to the clock and reset signals which fanout to the maximum number of flip-flops (i.e. identify the best global clock and reset signals and connect CLKBUF and RSTBUF to them).
- It is recommended to preserve the design hierarchy having leaf modules that can be generated using component generators. Making designs hierarchical and using hard macros when possible make them more routable and help in improving their performance.

Verilog Coding Styles

Verilog coding styles for a few important components are given below.

FD: Single Bit D flip-flop

Use of D flip-flops is highly recommended as it is the most efficient in the AT6000 FPGAs.

Assign active low polarity to a reset pin and positive edge trigger to a clock pin.

```
_____________________________________________________

module fd_syn(clk,rst,inp,out);
input clk,rst,inp;
output out;

reg out;

always @(posedge clk or negedge rst)
 begin
    if(rst == 'b0)
        out = 'b0;
    else
        out = inp;
 end

endmodule

_____________________________________________________
```

FDE: D flip-flop with enable

FDE is efficient for designs involving FIFOs or counters with enable capability. It is a good design practice to use them instead of gated clocks with flip-flops.

```
_____________________________________________________

module fde_syn(clk,rst,inp,en,out);
input clk,rst,inp,en;
output out;

reg out;

always @(posedge clk or negedge rst)
 begin
    if(rst == 'b0)
        out = 'b0;
    else if (en == 'b1)
        out = inp;
 end

endmodule

_____________________________________________________
```

FDMUX: Single bit D flip-flop with select

A Verilog description of a D flip-flop with a 2 to 1 MUX in front of it is shown below.

The FDMUX cell is efficient in applications such as load/enable counters, loadable shift registers and applications requiring synchronously settable/resettable register banks.

The FDMUX cell is a one cell implementation of a two input MUX and a D flip-flop.

---

module fdmux_syn(clk,rst,inp1,inp0,sel,out);

```
input clk,rst,inp1,inp0,sel;
output out;

reg out;

always @(posedge clk or negedge rst)
 begin
   if(rst == 'b0)
      out = 'b0;
   else if (sel == 'b1)
      out = inp1;
   else
      out = inp0;
 end

endmodule
```

FDHA: Half adder feeding a D flip-flop

A Verilog description of a D flip-flop with a half adder in front of it is shown below. FDHA is an ideal candidate for implementing registered sum or registered odd bit parity functions. It is also an ideal candidate for implementing various types of counter applications.

Following the coding style below to implement components such as FDHA will create the most efficient synthesis results.

A description similar to this can be written for components FDND (nand gate feeding DFF), FDN (inverter feeding DFF), FDOR (Or gate feeding DFF) and FDORL(Or gate with its one input inverted feeding DFF). Use of all of these components is highly recommended.

```
module fdha_syn(clk,rst,bit1,bit2,out);
input clk,rst,bit1,bit2;
output out;

reg out;

always @(posedge clk or negedge rst)
 begin
   if(rst == 'b0)
      out = 'b0;
   else
      out = (bit1 && ~bit2) || (bit2 && ~bit1);
 end

endmodule
```

FDSA: D flip-flop with asynchronous set capability

FDSA is an efficient component for setting the hot bit of the initial state of a state machine implementing one-hot state encoding.

It is also used for implementing a register bank with active high as reset state.

Use of this cell in place of FD is not recommended as it is more area intensive.

```
module fdsa_syn(clk,rst,inp,out);

input clk,rst,inp;
output out;

reg out;

always @(posedge clk or negedge rst)
 begin
    if(rst == 'b0)
        out = 'b1;
    else
        out = inp;
 end

endmodule
```

FDZ: D type flip-flop with tri-state capability

FDZ is a useful component in applications having buses driven by registered signals.

The tri-state can be used efficiently to implement a wide input multiplexor.

```
module fdz_syn(clk,rst,inp,oe,out);
input clk,rst,inp,oe;
output out;

reg out;
reg out_tmp;

always @(posedge clk or negedge rst )
 begin
    if(rst == 'b0)
        out_tmp = 'b0;
    else
        out_tmp = inp;
 end

always @(oe)
begin
    if(oe == 'b1)
        out = out_tmp;
    else
        out = 'bz;
end

endmodule
```

LDL: Transparent D latch with enable active low

Latches are very expensive and slow so their use is discouraged.

It may be useful as a replacement for flip-flops to reduce clock-reset combinations.

```
module ldl_syn(inp,out,enable);

input inp,enable;
output out;

reg out;

always @(inp or enable)
 begin
    if(enable == 'b0)
        out = inp;
 end

endmodule
```

VHDL Coding Guidelines

Every synthesis system has a unique way of recognizing the logical modules described using behavioral statements of an HDL design. The task of translating these components into a target technology (a task performed by a technology mapper) can be made more efficient by writing HDL designs with components in the technology library in mind. Writing designs in a technology specific coding style results in synthesis that is very efficient in terms of area and speed, and still provides the technology independence often required for design modification and migration.

This application brief describes a style for writing VHDL statements for components that are most efficient for Atmel FPGAs. Writing the VHDL code as suggested will help the synthesis tools to identify and preserve these components in the output netlist.

For details on an individual synthesis tool, users are encouraged to consult the appropriate sections in the IDS documentation and tutorial sessions provided for the specific system.

General Guidelines

A few important synthesis considerations are listed below. In addition to these guidelines, information specific to a particular synthesis system is given in the appropriate section of the IDS documentation.

- It is strongly recommended to assign Active Low Polarity to a global reset signal and Positive Edge Trigger to a global clock signal.
- Use of JK flip-flops and transparent latches is discouraged due to their excessive area cost.
- Use of many derived clock and reset signals is discouraged. It complicates the placement of flip-flops, resulting in clocks and resets getting routed on the general purpose interconnect, and eventually degrades the performance of the design.
- CLKBUF and RSTBUF pads should be connected to the clock and reset signals which fanout to the maximum number of flip-flops (i.e. identify the best global clock and reset signals and connect CLKBUF and RSTBUF to them).
- It is recommended to preserve the design hierarchy having leaf modules that can be generated using component generators. Making designs hierarchical and using hard macros when possible make them more routable and help in improving their performance.

VHDL Coding Styles

VHDL coding styles for a few important components are given below.

FD: Single Bit D flip-flop

Use of D flip-flops is highly recommended as it is the most efficient in the AT6000 FPGAs.

Assign Active Low Polarity to a reset pin and Positive Edge Trigger to a clock pin.

```
_____________________________________________________
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY fd_syn IS
       PORT(inp,clk,rst : in std_logic; op : out std_logic);
END fd_syn;


ARCHITECTURE behv OF fd_syn IS

begin

PROCESS (clk, rst)
begin
       IF(rst = '0') THEN
              op <= '0';
       ELSIF (clk = '1' and clk'event) THEN
              op <= inp;
       END IF;
end process;

end behv;

_____________________________________________________
```

FDE: D flip-flop with enable

FDE is efficient for designs involving FIFOs or counters with enable capability. It is good design practice to use them instead of gated clocks with flip-flops.

```
_____________________________________________________
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY fde_syn IS
        PORT(inp,en,clk,rst : in std_logic; op : out std_logic);
END fde_syn;


ARCHITECTURE behv OF fde_syn IS

begin

PROCESS (rst,clk)
begin
       IF(rst = '0') THEN
              op <= '0';
       ELSIF (clk'event and clk = '1') THEN
              IF( en = '1') THEN
                     op <= inp;
              END IF;
       END IF;
end process;

end behv;

_____________________________________________________
```

FDMUX: Single bit D flip-flop with select

A VHDL description of a D flip-flop with a 2 to 1 MUX in front of it is shown below.

The FDMUX cell is efficient in applications such as load/enable counters, loadable shift registers and applications requiring synchronously settable/resettable register banks.

The FDMUX cell is a one cell implementation of a two input MUX and a D flip-flop.

```
_____________________________________________________
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY fdmux_syn IS
        PORT(inp1,inp0,sel,clk,rst : in std_logic; op : out std_logic);
END fdmux_syn;

ARCHITECTURE behv OF fdmux_syn IS

begin

PROCESS (clk,rst)
begin
       IF(rst = '0') THEN
              op <= '0';
       ELSIF(clk'event and clk = '1') THEN
              IF (sel = '1') THEN
                     op <= inp1;
              ELSE
                     op <= inp0;
              END IF;
       END IF;
end process;

end behv;
_____________________________________________________
```

FDHA: Half adder feeding a D flip-flop

A VHDL description of a D flip-flop with a half adder in front of it is shown below.

FDHA is an ideal candidate for implementing registered sum or registered odd bit parity functions. It is also an ideal candidate for implementing various types of counter applications.

Following the coding style below to implement components such as FDHA will create the most efficient synthesis results.

A description similar to this can be written for components FDND (nand gate feeding DFF), FDN (inverter feeding DFF), FDOR (Or gate feeding DFF) and FDORL(Or gate with its one input inverted feeding DFF). Use of all of these components is highly recommended.

```
_____________________________________________________

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY fdha_syn IS
        PORT(inp1,inp0,clk,rst : in std_logic; op : out std_logic);
END fdha_syn;

ARCHITECTURE behv OF fdha_syn IS

begin

PROCESS (clk,rst)
begin
       IF(rst = '0') THEN
              op <= '0';
       ELSIF(clk'event and clk = '1') THEN
              op <= (inp1 and not inp0 ) or (not inp1 and inp0);
       END IF;
end process;

end behv;

_____________________________________________________
```

FDSA: D flip-flop with asynchronous set capability

A VHDL description of a D flip-flop with asynchronous set capability is shown below.

FDSA is an efficient component for setting the hot bit of the initial state of a state machine implementing one-hot state encoding.

It is also used for implementing a register bank with active high as reset state.

Use of this cell in place of FD is not recommended as it is more area intensive.

```
_____________________________________________________
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY fdsa_syn IS
       PORT(inp,clk,rst : in std_logic; op : out std_logic);
END fdsa_syn;


ARCHITECTURE behv OF fdsa_syn IS

begin

PROCESS (clk, rst)
begin
       IF(rst = '0') THEN
              op <= '1';
       ELSIF (clk = '1' and clk'event) THEN
              op <= inp;
       END IF;
end process;

end behv;

_____________________________________________________
```

FDZ: D type flip-flop with tri-state capability

A VHDL description of a D type flip-flop with tri-state capability is shown below.

FDZ is a useful component in applications having buses driven by registered signals.

The tri-state can be used efficiently to implement a wide input multiplexor.

```
_____________________________________________________
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY fdz_syn IS
       PORT(inp,oe,clk,rst : in std_logic; op : out std_logic);
END fdz_syn;


ARCHITECTURE behv OF fdz_syn IS

SIGNAL op_tmp : std_logic;
begin

PROCESS (clk, rst)
begin
       IF(rst = '0') THEN
              op_tmp <= '0';
       ELSIF (clk = '1' and clk'event) THEN
              op_tmp <= inp;
       END IF;
end process;

PROCESS (oe,op_tmp)
begin
       IF(oe = '1') THEN
              op <= op_tmp;
       ELSE
              op <= 'Z';
       END IF;
end process;

end behv;

_____________________________________________________
```

LDL: Transparent D latch with enable active low

Latches are very expensive and slow so their use is discouraged.

It may be useful as a replacement for flip-flops to reduce clock-reset combinations.

```
_____________________________________________________
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY ldl_syn IS
       PORT(inp,enable : in std_logic; op : out std_logic);
END ldl_syn;


ARCHITECTURE behv OF ldl_syn IS

begin

PROCESS (inp, enable)
begin
       IF(enable = '0') THEN
              op <= inp;
       END IF;
end process;

end behv;

_____________________________________________________
```

Design Files

This section contains information about the various files seen in the design directory. The file names or extensions are given below. A short description of what the files are used for and their importance are provided. More details on the Figaro specific files can be found in the on-line *Help* and the *Figaro* section of the *Tutorial.*

IDS files	Description	Required
.~	Backup of the file with the name *.*	No
*.hxr	Intel MCS86 hexadecimal format of the bitstream suitable for use with third-party programmers for serial configuration memories (AT40k only)	No
*.hx	ASCII hexadecimal file of the bitstream (AT6000 only)	No
*.hex	Intel MCS86 hexadecimal format of the bitstream	No
*.bst	Bitstream file for the specified design	Yes
*.cdb *.cds	Atmel data base of the design used for generating bitstreams (AT6000 only)	No
*.fgd	Figaro data base of the design	Yes
*.ini	Figaro configuration for storing design specific settings, such as the libraries associated with the design	Yes
*.log	It contains information on all activities of the design	Yes

IDS files	Description	Required
*.rct	The repeat constraints file stores information on the package used, and pin locations for the design. It can be used as input to control these functions	Yes
*.sts	Statistics of the design such as number of core cells, macros, nets, etc.	Yes
fgswap/*.swp	When doing a multichip design and if part swapping is enabled, Figaro writes the swapped-out part to the disk	Yes
*.map	Mapping constraints file	No
*.ptn	Partition constraints file	No
*.pin	Pinout constraints file	No
*.pir	Pinout report	No
*.tmg	Timing constraints file	No
*.pdl	Path analysis report file	No
*.ndl	Net delay analysis report file	No
*.lib	User library files	Yes
atmel.ini	The control file for Atmel point tools that make up IDS	No

IDS files	Description	Required
design.cfg	The control file for Atmel point tools that make up IDS	No
fcr.*	Copy of problem report submitted	Yes
figaro.ini	The Figaro control file contains user preference settings as well as design and library settings	Yes

Concept files	Description	Required
figba	Directory used to store all back annotation netlists (*.v) and delay files (*.sdf)	No
*.edf	EDIF netlist for the specified design	Yes
cmplog.dat	The results of compilation of the EDIF netlist or the Verilog netlist	No
compiler.cmd	Control file for the Concept compilation programs	Yes
design.wrk	Concept design directory	Yes
edif2fig.log	Log file of the loading of the EDIF file into Figaro	No
fetsetup/	Concept working sub-directory	Yes
formtool.log	Concept log file	No
global.cmd	Concept library directory specifications	Yes
master.local	Concept library locations	yes
netout.lst	Results file of the Atmel tool used for generating various netlists such as the flat Verilog file	No
sdf.log	Results of the Standard Delay File	Yes

Concept files	Description	Required
testfixture.template	Verilog test fixture created by the netlist program vloglink	Yes
testfixture.verilog	User specified Verilog stimulus file	Yes
verilog.log	The results from running Verilog on the design	No
vloglink.cmd	Control file for the Concept Verilog netlist generator vloglink	Yes
vloglink.lst	The result of generating a Verilog netlist via vloglink	No
vloglink.map	Name mapping file for converting Concept names to Verilog names	No
vloglink.v	Verilog netlist of the design used for *Functional Simulation*	Yes
wedifnet.cmd	Control file for the Concept EDIF netlist generator wedifnet	Yes
wedifnet.lst	Results of the EDIF netlist generation	No
xshadow/	Concept working sub-directory	Yes
xxnedtmp/	Concept working sub-directory	Yes

VeriBest Files	Description	Required
*.sbk	VeriBest Schematic file	Yes
designName.prj	VeriBest project file for the design	Yes
designName.cdb	VeriBest's common design database directory. Can be changed to the desired directory using the Project Settings option in VeriBest tools	Yes
genhdl	Directory where generated HDL files will be stored. Can be changed to the Design Directory using the Project Settings option in VeriBest tools	Yes
hdl	Directory where VeriBest tools look for source HDL files. Can be changed to the design directory using the Project Settings option in VeriBest tools	No
udb	VeriBest's Universal database directory. Can be changed to the design directory using the Project Settings option in VeriBest tools	No
vbdc.sup	Configuration file for the simulator	Yes

VeriBest Files	Description	Required
*.n	VeriBest design database created by the synthesis tool	Yes
*.edf	EDIF netlist for the specified design	Yes
edif2fig.log	Log file of the loading of the EDIF file into Figaro	No
figba	Directory used to store all back annotation netlists (*.v) and delay files (*.sdf)	No
netout.lst	Results file of the Atmel tool used for generating various netlists such as the flat Verilog file	No

Mentor Files	Description	Required
*.edf	EDIF netlist for the specified design	Yes
edif2fig.log	Log file of the loading of the EDIF file into Figaro	No
figba/	Directory used to store all back annotation netlists and delay files (*.dba)	No
design_name/	Mentor design directory where all the mentor schematic and symbol files and other mentor design object related files are stored	No
*.mgc_component.attr	Mentor component attribute file	No

Viewlogic Files	Description	Required
*.var	Parameterized attributes file, contains the values for the macro intrinsic delays used for simulation	Yes
*.vpj	Project file which keeps track of the directories used for the current project and associated libraries.	Yes
*.vsm	Simulation netlist produced by the vsm program	No
*.wfm	Waveform file created by simulation programs and used as input to the Viewlogic Waveform Viewer	No
check.err	Results of running the check program (netlist generator) on the design	No
pv.log	Powerview log file	No
viewdraw.ini	Viewlogic configuration file. Contains various settings for the Viewlogic tools and the location of libraries referenced by the current design	Yes
viewgen.log	Results of running Viewgen, the schematic generator	No
viewsim.log	Results of running the simulation	Yes
Viewlogic Files	**Description**	**Required**
figba/	WIR and DTB files that are to be used for *Post-layout Simulation*	No
pkt/	Viewlogic working directory	No
sch/	Schematic sub-directory	Yes
sym/	Symbol sub-directory	Yes
wir/	Netlist sub-directory	Yes

AT6000 Architecture

The Atmel architecture was developed to provide the highest levels of performance, functional density and design flexibility in a Field-Programmable Gate Array (FPGA). The cells in the Atmel AT6000 array are small, very efficient, and contain the most important and most commonly used logic and wiring functions. The cell's small size leads to arrays with large numbers of cells, greatly multiplying the functionality in each cell. A simple, high-speed busing network provides fast, efficient communication over medium and long distances.

The Symmetrical Array

At the heart of the Atmel architecture is a symmetrical array of identical cells (Figure 1).

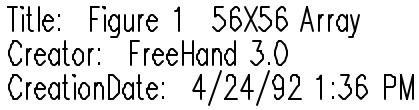


Figure 1. Symmetrical Array Surrounded by I/O

Except for repeaters spaced every eight cells, the array is continuous and completely uninterrupted from one edge to the other (Figure 2).

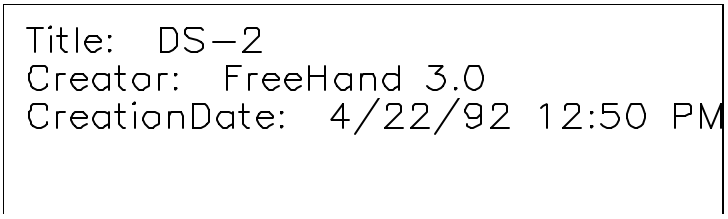


Figure 2. Busing Network

In addition to logic and storage, cells can also be used as wires to connect functions together over short distances, and are useful for routing in tight spaces. Buses support fast, efficient communication over medium and long distances.

The Busing Network

There are two kinds of buses: local and express (Figures 2 and 3).

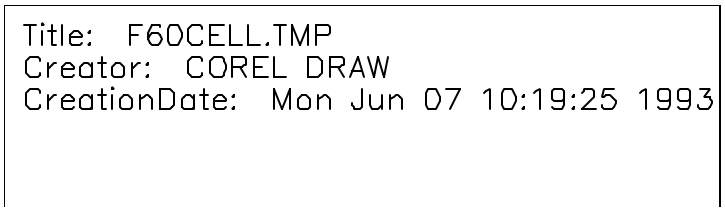


Figure 3. Cell-to-Cell and Cell-to-Bus Connections

Local buses are the link between the array of cells and the busing network. There are two local buses—– North-South 1 and 2 (NS1 and NS2)—– for every column of cells, and two local buses—– East-West 1 and 2 (EW1 and EW2)—– for every row of cells. Each local bus is connected to every cell in its column or row, thus providing every cell in the array with read/write access to two North-South and two East-West buses.

Each cell, in addition, provides the ability to route a signal on a 90-degree turn between the NS1 bus and EW1 bus and between the NS2 bus and the EW2 bus. Express buses are not connected directly to cells, and therefore provide higher speeds. Express buses are the fastest way to cover long, straight-line distances within the array.

Each express bus is paired with a local bus, so there are two express buses for every column and two express buses for every row of cells. Connective units, called repeaters, are spaced every eight cells. Repeaters divide every bus, local and express, into segments spanning eight cells. Repeaters are aligned in rows and columns, partitioning the array into 8X8 sectors of cells. Each repeater is associated with a local/express pair, and each side of the repeater has connections to a local-bus segment and an express-bus segment.

The repeater can be programmed to provide any one of twenty-one connecting functions. These functions are symmetrical with respect to both the two repeater sides and the two types of buses. Among the functions provided are the ability to isolate bus segments from one another, connect two local-bus segments, connect two express-bus segments, and to implement a local/express transfer. In all of these cases, each connection provides signal regeneration and is thus uni-directional. For bi-directional connections, the basic repeater function for the NS2 and EW2 repeaters is augmented with a special programmable connection, which allows bi-directional communication between local-bus segments. This option is primarily used to implement long, tri-state buses.

The Cell Structure

The Atmel AT6000 cell (Figure 4) is simple and small and yet can be programmed to perform all the logic and wiring functions needed to implement any digital circuit. Its four sides are functionally identical, so each cell is completely symmetrical.

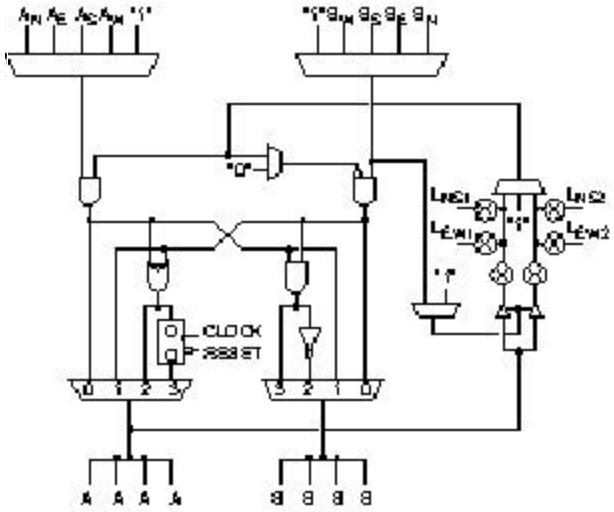


Figure 4. Cell Structure

Read/write access to the four local buses— NS1, EW1, NS2, and EW2— is controlled, in part, by four bi-directional pass gates connected directly to the buses. To read a local bus, the pass gate for that bus is turned on and the three-input multiplexer is set accordingly. To write to a local bus, the pass gate for that bus and the pass gate for the associated tri-state driver are both turned on. The two-input multiplexer supplying the control signal to the drivers permits either: (1) active drive, or (2) dynamic tri-state controlled by the B input. Turning between L_{NS1} and L_{EW1} or between L_{NS2} and L_{EW2} is accomplished by turning on the two associated pass gates. The operations of reading, writing and turning are subject to the restriction that each bus can be involved in no more than a single operation.

In addition to the four local-bus connections, a cell receives two inputs and provides two outputs to each of its North (N), South (S), East (E) and West (W) neighbors. These inputs and outputs are divided into two classes: “A” and “B”. There is an A input and a B input for each neighboring cell and an A output and a B output driving all four neighbors. Between cells, an A output is always connected to an A input and a B output to a B input.

Within the cell, the four A inputs and the four B inputs enter two separate, independently configurable multiplexers. Cell flexibility is enhanced by allowing each multiplexer to select the logical constant “1.” The two multiplexer outputs enter the two upstream AND gates.

Downstream from these two AND gates are an Exclusive-OR (XOR) gate, a register, a downstream AND gate, an inverter and two four-input multiplexers producing the A and B outputs. These multiplexers are controlled in tandem (unlike the A and B input multiplexers) and determine the function of the cell.

- In State 0, which corresponds to the “0” inputs of the multiplexers, the output of the left-hand upstream AND gate is connected to the cell's A output, and the output of the right-hand upstream AND gate is connected to the cell's B output.
- In State 1, which corresponds to the “1” inputs of the multiplexers, the output of the left-hand upstream AND gate is connected to the cell's B output, and the output of the right-hand upstream AND gate is connected to the cell's A output.
- In State 2, which corresponds to the “2” inputs of the multiplexers, the XOR of the outputs from the two upstream AND gates is provided to the cell's A output, while the NAND of these two outputs is provided to the cell's B output.
- In State 3, which corresponds to the “3” inputs of the multiplexers, the XOR function of State 2 is provided to the D input of a D-type flip-flop, the Q output of which is connected to the cell's A output. Clock and asynchronous reset signals are supplied externally. The AND of the outputs from the two upstream AND gates is provided to the cell's B output.

Logic States

The Atmel cell implements a rich and powerful set of logic functions, stemming from 44 cell states. Some states use both A and B inputs. Other states are created by selecting the “1” input on either or both of the input multiplexers.

- There are 20 purely combinatorial states with a range of functions, including NOR, AND, NAND, OR and two-input multiplexer (Figure 5).

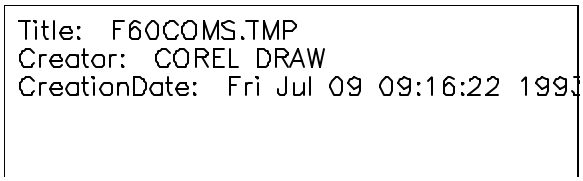


Figure 5. Combinatorial States

- There are 11 register states, ranging from a simple register to a register preceded by a two-input multiplexer (Figure 6).

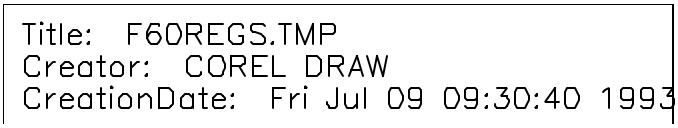


Figure 6. Register States

- Five constant states produce all combinations of constant values at the two cell outputs (Figure 7).

Figure 7. Constant States

- There are five tri-state states.

More complex functions are created by using cells in combination.

A two-input AND feeding an XOR (Figure 8) is produced using a single cell (Figure 9).

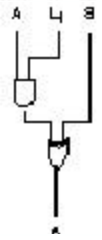


Figure 8. Complex Macro Function

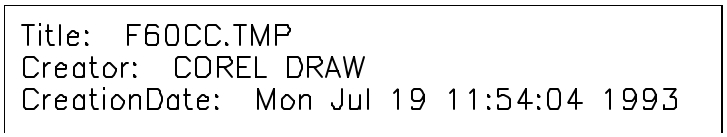


Figure 9. Cell Configuration

A two-to-one multiplexer selects the logical constant “0” and feeds it to the right-hand AND gate. The AND gate acts as a feed-through, letting the B input pass through to the XOR. The three-to-one multiplexer on the right side selects the local-bus input, L_{NS1}, and passes it to the left-hand AND gate. The A and L_{NS1} signals are the inputs to the AND gate. The output of the AND gate feeds into the XOR, producing the logic state (A·L) XOR B.

Clock Distribution

Along the top edge of the array is logic for distributing clock signals to the D flip-flop in each logic cell (Figure 10). The distribution network is organized by column and permits columns of cells to be independently clocked.

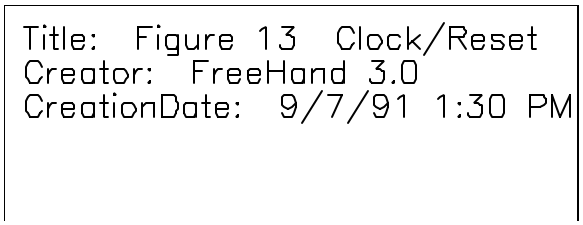


Figure 10. Column Clock and Column Reset

At the head of each column is a user-configurable multiplexer providing the clock signal for that column. It has four inputs:

- Global clock supplied through the CLOCK pin
- Express bus adjacent to the distribution logic
- "A" output of the cell at the head of the column
- Logical constant "1" to conserve power (no clock)

Through the global clock, the network provides low-skew distribution of an externally supplied clock to any or all columns of the array. The global clock pin is also connected directly to the array via the A input of the upper left and right corner cells (AW on the left, and AN on the right). The express bus is useful in distributing a secondary clock to multiple columns when the global clock line is used as a primary clock. The A output of a cell is useful in providing a clock signal to a single column. The constant "1" is used to reduce power dissipation in columns using no registers.

Asynchronous Reset

Along the bottom edge of the array is logic for asynchronously resetting the D flip-flops in the logic cells (see Figure 10). Like the clock network, the asynchronous reset network is organized by column, and permits columns to be independently reset. At the bottom of each column is a user-configurable multiplexer providing the reset signal for that column. It has four inputs:

- Global asynchronous reset supplied through the $\overline{\text{RESET}}$ pin
- Express bus adjacent to the distribution logic
- "A" output of the cell at the foot of the column
- Logical constant "1" to conserve power

The asynchronous reset logic uses these four inputs in the same way that the clock distribution logic does. Through the global asynchronous reset, any or all columns can be reset by an externally supplied signal. The global asynchronous reset pin is also connected directly to the array via the A input of the lower left and right corner cells (AS on the left, and AE on the right). The express bus can be used to distribute a secondary reset to multiple columns when the global reset line is used as a primary reset; the A output of a cell can also provide an asynchronous reset signal to a single column; the constant "1" is used by columns with registers requiring no reset. All registers are reset during power-up.

Input/Output

The Atmel architecture provides a flexible interface between the logic array, the configuration control logic, and the I/O pins. Two adjacent cells, an entrance cell and an exit cell, on the perimeter of the logic array are associated with each I/O pin. There are two types of I/Os, A (Figure 11) and B (Figure 12).

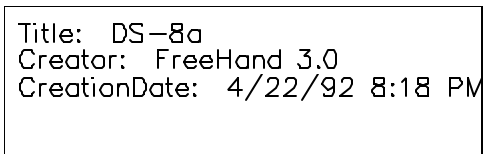


Figure 11. I/O Pin Type A

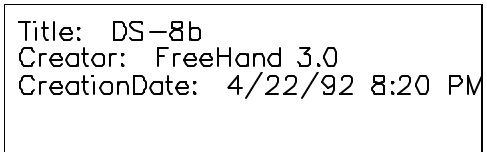


Figure 12. I/O Pin Type B

For A-type I/Os, the edge-facing A output of an exit cell is connected to an output driver, and the edge-facing A input of the adjacent entrance cell is connected to an input buffer. The output of the output driver and the input of the input buffer are connected to a common pin. The B-type I/Os are the same as A-type I/Os, but use the B inputs and outputs of their respective entrance and exit cells. A- and B-type I/Os alternate around the array. Control of the I/O logic is provided by the following user-configurable memory bits:

- **TTL/CMOS Inputs** A user-configurable bit determining the threshold level, TTL or CMOS, of the input buffer.
- **Open Collector/Tri-state Outputs** A user-configurable bit that enables or disables the active pull-up of the output device.
- **Slew Rate Control** A user-configurable bit that controls the slew rate, fast or slow, of the output buffer. A slow slew rate, which reduces noise and ground bounce, is recommended for outputs that are not speed-critical. Fast and slow slew rates have the same DC-current sinking capabilities, but different rates at which each allows the output devices to reach full drive.
- **Pull-up** A user-configurable bit controlling the pull-up transistor in the I/O pin. Its primary function is to provide a logical “1” to unused input pins. When on, it is roughly equivalent to a 25K resistor to V_{CC}.
- **Enable Select** User-configurable bits determining the output-enable for the output driver. The output driver can be static, always on, always off, or dynamically controlled by a signal generated in the array. Four options are available from the array: (1) the control is low and always driving; (2) the control is high and never driving; (3) the control is connected to a vertical local bus associated with the output cell; and (4) the control is connected to a horizontal local bus associated with the output cell. The power-up default is never driving.

In addition to the functionality provided by the I/O logic, the entrance and exit cells provide the ability to register both inputs and outputs. Also, these perimeter cells (unlike interior cells) are connected directly to express buses: the edge-facing A and B outputs of the entrance cell are connected to express buses, as are the edge-facing A and B inputs of the exit cell. These buses are perpendicular to the edge, and provide a rapid means of bringing I/O signals to and from the array interior and the opposite edge of the chip.

Chip Configuration

The Integrated Development System generates the SRAM bit pattern required to configure a AT6000 Series device. A PC parallel port, microprocessor, or EPROM can be used to download configuration patterns

Many factors, including board area, configuration speed, and the number of designs implemented in parallel, can influence the final choice.

Configuration is controlled by dedicated configuration pins and dual-function pins that double as I/O pins when the device is in operation. The number of dual-function pins required for each mode varies.

The devices can be partially reconfigured while in operation. Portions of the device not being modified remain operational during configuration. Simultaneous configuration of more than one device is also possible. Full configuration takes as little as a millisecond, partial configuration is even faster.

Pin Descriptions

This section provides an abbreviated description of the AT6000 Series pins. For more complete information, refer to the AT6000 Series Configuration data sheet.

Power Pins

V_{CC}, V_{DD}, GND, V_{SS} V_{CC} and GND are the I/O supply pins, V_{DD} and V_{SS} are the internal logic supply pins. Pins V_{CC} and V_{DD} should be tied to the same trace on the printed circuit board. Pins GND and V_{SS} should be tied to the same trace on the printed circuit board.

Input/Output Pins All I/O pins can be used in the same way (see "Architecture, Input/Output"). Some I/O pins are dual-function pins used during configuration of the array. When not being used for configuration, dual-function I/Os are fully functional as normal I/O pins. On initial power-up, all I/Os are configured as TTL inputs with a pull-up.

Dedicated Timing and Control Pins

$\overline{\text{CON}}$

Configuration-in-process pin. After power-up, $\overline{\text{CON}}$ remains low until power-up initialization is complete. $\overline{\text{CON}}$ is an open collector signal. After power-up initialization, forcing $\overline{\text{CON}}$ low begins the configuration process.

$\overline{\text{CS}}$

Configuration-enable pin. All configuration pins are ignored if $\overline{\text{CS}}$ is high. The $\overline{\text{CS}}$ pin must be held low throughout the configuration process. The $\overline{\text{CS}}$ pin is a TTL-type input pin.

M0, M1, M2

Configuration-mode pins used to determine the configuration mode. All three are TTL-type input pins.

CCLK

Configuration-clock pin. Whether CCLK is an input or an output depends on the configuration mode selected. It is an output in two modes. The output modes support configuration using the fewest external components. In either of these modes, the configuration time varies from part to part, depending on clock speed. CCLK is a TTL-type input in modes that use it as an input, and a CMOS-type output in those modes that use it as an output. When not in use, CCLK is set low.

CLOCK

Internal logic source used to drive the internal global clock line. Registers toggle on the rising edge of CLOCK. The CLOCK signal is neither used nor affected by the configuration modes. It is always a TTL input.

$\overline{\text{RESET}}$

Array register asynchronous reset. The $\overline{\text{RESET}}$ pin drives the internal global reset. The $\overline{\text{RESET}}$ signal is neither used nor affected by the configuration modes. It is always a TTL input.

Dual-Function Pins

When $\overline{CON}$ is high, dual-function I/O pins act as device I/Os; when $\overline{CON}$ is low, dual-function pins are used as configuration control or data signals as determined by the configuration modes. When using these pins, the user must ensure that configuration activity does not interfere with other circuitry associated with the pin's net.

D0 or I/O

Serial configuration modes use D0 as the serial data input pin. Parallel configuration modes use D0 as the least-significant bit. Input data must meet setup and hold requirements with respect to the rising edge of CCLK.

D1 to D7 or I/O

Parallel configuration modes use these pins as inputs. Serial configuration modes do not use them. Data must meet setup and hold requirements with respect to the rising edge of CCLK.

$\overline{CEN}$ or I/O

During address count-up/down configuration, $\overline{CEN}$ is an output. The $\overline{CEN}$ pin can be used as the output enable of a parallel EPROM. In this case, it should be configured as a constant high, and not used as a configurable I/O pin. The $\overline{CEN}$ pin is available only on the AT6005 device.

A0 to A16 or I/O

During address count-up/down configuration, these pins are outputs and act as the address pins for a parallel EPROM. They eliminate the need for an external address counter if the user wishes to use an inexpensive parallel EPROM to program a device. Addresses change after the rising edge of the CCLK signal.

$\overline{CSOUT}$ or I/O

When cascading devices, $\overline{CSOUT}$ is an output used to configure other devices. $\overline{CSOUT}$ should be connected to the $\overline{CS}$ input of the downstream device. The $\overline{CSOUT}$ function is optional and can be disabled during initial programming when cascading is not used. When cascading devices, $\overline{CSOUT}$ should be dedicated to configuration and not used as a configurable I/O.

$\overline{CHECK}$ or I/O

During configuration, $\overline{CHECK}$ is an input that can be used to enable the data check function at the beginning of a configuration cycle. No data is written to the device while $\overline{CHECK}$ is low. Instead, the configuration file applied to D0-D7 is compared with the current contents of the internal configuration RAM. If a mismatch is detected between the data being loaded and the data already in the RAM, the $\overline{ERR}$ pin goes low. The $\overline{CHECK}$ function is optional and can be disabled during initial programming.

$\overline{ERR}$ or I/O

During configuration, $\overline{ERR}$ is an output. When the $\overline{CHECK}$ function is activated and a mismatch is detected between the current configuration data stream and the data already loaded in the configuration RAM, $\overline{ERR}$ goes low. The $\overline{ERR}$ output is a registered signal. Once a mismatch is found, the signal is set and is only reset after the configuration cycle is complete. The $\overline{ERR}$ pin is also asserted for configuration file errors. The $\overline{ERR}$ function is optional and can be disabled during initial programming.

Control Register

This section provides a general description of the AT6000 Series control register.

The Integrated Development System generates the bit stream file used to configure the FPGA. In addition to the actual data to be loaded into the SRAM, the bit stream loads a control register containing eight bits used to control various configuration sequence parameters (Figure 13. Control Register).

Figure 13. Control Register

B0

B0 controls the value of the device's memory address counter after each configuration sequence. The default resets the value, so subsequent configuration sequences load the configuration file from the same address. In modes 1 and 5, the default address is 0000, in mode 2 it is 1FFFF. When B0 is set, the memory address counter retains its last value, so the user can store multiple designs sequentially in an EPROM.

B1

B1 controls loading of a jump address into the device's memory address counter. The default ignores any jump addresses. With B1 set, the memory address counter jumps to the specified address. Using B1, configuration files can be stored as a continuous stream or as a pointer-based linked list.

B2

B2 controls the operation of the dual-function pins $\overline{\text{CSOUT}}$ and $\overline{\text{CEN}}$. When B2 is set, these two pins are disabled. This is useful when a minimum pin-count configuration is desired.

B3

B3 controls the operation of the dual-function pins $\overline{\text{ERR}}$ and $\overline{\text{CHECK}}$. When B3 is set, both pins are disabled. This is useful when a minimum pin-count configuration is desired, or when design security is a concern.

B4

B4 controls the writing of configuration data after the initialization state. When B4 is set, configuration data can not be written into the device by subsequent configuration cycles. B4 can only be reset by rebooting the device.

B5 and B6

B5 and B6 control the operation of the global clock signal received through the CLOCK pin:

B5	B6	Global Clock Operation
0	0	Normal operation.
1	0	Stops after third rising edge of CLOCK after $\overline{\text{CON}}$ is low. Continues after second rising edge of CLOCK after $\overline{\text{CON}}$ is high.
0	1	Stops after fourth rising edge of CLOCK after $\overline{\text{CON}}$ is high. Each configuration cycle thereafter, it receives one pulse after the third rising edge of CLOCK after $\overline{\text{CON}}$ is low.
1	1	Stops at second rising edge of CLOCK after $\overline{\text{CON}}$ is high. Remains stopped regardless of $\overline{\text{CON}}$.

B7

B7 controls static power consumption. When B7 is set, the internal oscillator used for auto configuration in mode 4 and for generating cascade CLKOUT signal, is disabled. On the AT6005, this function is performed by B4.