

---

# Ambit and Envisia Tutorial

## Product Version 4.0 August 2000

© 1999-2000 Cadence Design Systems, Inc. All rights reserved.  
Printed in the United States of America.

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

**Trademarks:** Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 1-800-862-4522.

All other trademarks are the property of their respective holders.

**Restricted Print Permission:** This publication is protected by copyright and any unauthorized use of this publication may violate copyright, trademark, and other laws. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. This statement grants you permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used solely for personal, informational, and noncommercial purposes;
2. The publication may not be modified in any way;
3. Any copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement; and
4. Cadence reserves the right to revoke this authorization at any time, and any such use shall be discontinued immediately upon written notice from Cadence.

**Disclaimer:** Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence's customer in accordance with, a written agreement between Cadence and its customer. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

**Restricted Rights:** Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

---

# Contents

---

<u>Ambit BuildGates</u> .....	5
<u>Envisia Timing Analysis</u> .....	6
<u>Envisia Test Synthesis</u> .....	7
<u>The CPU Example</u> .....	8
<u>Defining Environment Variables</u> .....	9
<u>More Information</u> .....	9

## 2

<b><u>Synthesizing a Design from the Top Down</u></b> .....	10
<u>Invoking the Synthesis Tool</u> .....	10
<u>Reading a Technology Library</u> .....	10
<u>Reading the Design Modules</u> .....	11
<u>Building a Generic Netlist</u> .....	11
<u>Setting Timing Constraints</u> .....	12
<u>Defining Data Arrival and Required Times</u> .....	13
<u>Optimizing the Design</u> .....	14
<u>Generating a Timing Report</u> .....	15
<u>Saving the Netlist</u> .....	17
<u>Exiting from Ambit BuildGates</u> .....	17

## 3

<b><u>Creating a Flattened Netlist</u></b> .....	19
<u>Invoking the GUI</u> .....	19
<u>Reading a Technology Library</u> .....	20
<u>Reading the Design Modules and Building the Generic Netlist</u> .....	21
<u>Defining the Timing Constraints</u> .....	24
<u>Optimizing the Netlist</u> .....	28
<u>Flattening the Netlist</u> .....	30
<u>Generating the Timing Report</u> .....	31
<u>Saving the Netlist</u> .....	32
<u>Exiting from the GUI</u> .....	33

# Ambit and Envisia Synthesis Tutorial

---

## **4**

<b><u>Synthesizing a Design from the Bottom Up</u></b> .....	34
<u>Preparing for Synthesis</u> .....	34
<u>Setting the Ideal Clock</u> .....	35
<u>Synthesizing Individual Design Blocks</u> .....	35
<u>Generating a Netlist for the Top Module in the Design</u> .....	37

## **5**

<b><u>Inserting a Scan Chain</u></b> .....	38
<u>Preparing for Synthesis</u> .....	38
<u>Setting Test Synthesis Assertions</u> .....	39
<u>Adding the Scan Logic</u> .....	39
<u>Setting Timing Constraints and Optimizing the Design</u> .....	40
<u>Connecting the Scan Chain</u> .....	40
<u>Saving the Netlist and Exiting</u> .....	41
<u>Viewing the Scan Chain File</u> .....	42

<b><u>Glossary</u></b> .....	44
------------------------------	----

---

## Introduction

---

*Synthesis* is the process by which you convert a design written at the register-transfer level (RTL) into a gate-level netlist. The RTL specification is written in Verilog or VHDL, using high-level constructs such as `for` loops and `case` statements. The synthesis tool transforms this RTL specification into a set of logic gates, such as AND, OR, and BUF, that are connected in a network.

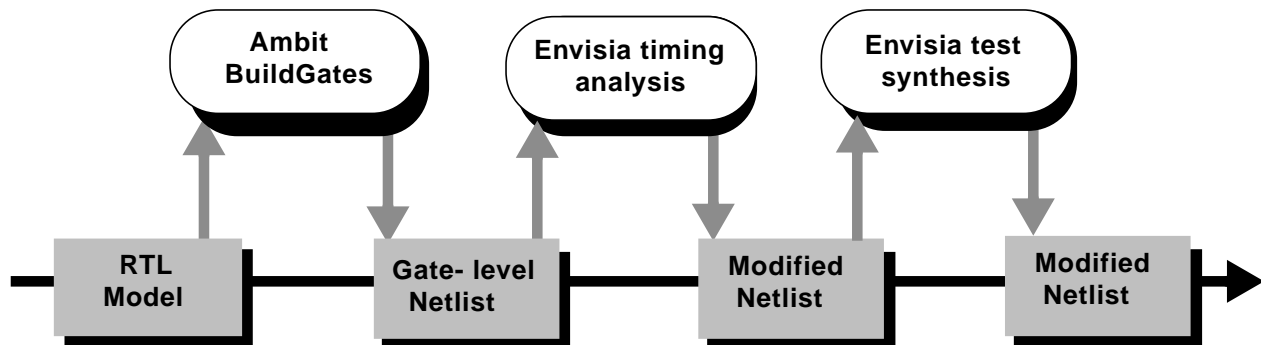
To specify the gates that the synthesis tool uses to build a netlist, you need to choose a technology from a specific vendor. The vendor that you have chosen to fabricate your chip or system supplies a technology library for you to use in synthesis. The technology library defines the physical properties of the gates, including the amount of time that is required for a signal to pass through each gate.

In addition to creating a gate-level netlist, the synthesis tool can perform the following functions:

- Analyze the timing of the netlist to ensure that no timing errors can occur.
- Optimize the design for either the best performance or the smallest size.
- Automatically insert a chain of scan elements or test signals into the netlist.

Figure 1-1 on page 5 shows how you can use the Ambit® and Envisia® synthesis tools to develop a design, from an RTL description through test insertion. These are the steps that are covered in this tutorial. However, you can also use the Ambit and Envisia tools during the back-end development process. Layout and floor planning tools, for example, are also supported by the Ambit and Envisia tools.

**Figure 1-1 Design Stages from Synthesis through Scan Insertion**



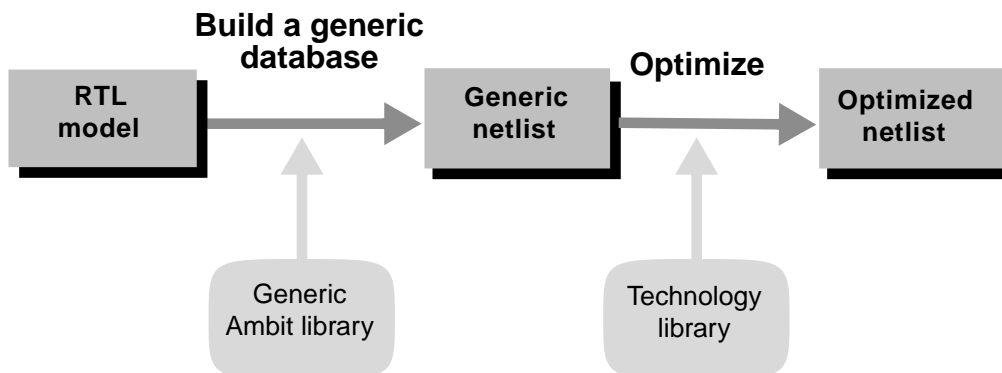
## Ambit BuildGates

You can use Ambit® BuildGates® to generate optimized gate-level netlists from your RTL models, as follows:

1. Read your technology library into the synthesis database.
2. Read the HDL source code for your design, written in Verilog or VHDL, into the synthesis database.
3. Generate a generic netlist based on the generic Ambit library.
4. Map the generic netlist to cells in the technology library and optimize the netlist.

These steps are illustrated in [Figure 1-2](#) on page 5.

**Figure 1-2 Synthesis Steps**



# Ambit and Envisia Synthesis Tutorial

## Introduction

---

Ambit BuildGates has both a command-line interface and a graphical user interface (GUI). Both provide the same synthesis functions. The GUI provides the following additional features:

- **Module browser**—Displays the design hierarchy. You can navigate through the hierarchy, and perform operations on the hierarchy, such as setting the top module or dissolving modules and branches in the hierarchy.
- **Source code editor**—Gives you access to your HDL source files. You can load any changes that you make to the source files back into the synthesis tool, and generate a new netlist with those changes.
- **Schematic viewer**—Displays your design in schematic form. You can pan and zoom, display fanin and fanout cones, and display critical paths and timing values. You can group instances, dissolve instances, or change the reference point of an instance.
- **Report viewer**—Displays timing reports, area reports, and other reports that are generated during your synthesis session.
- **TCL editor**—Lets you create, edit, save, and source your TCL scripts.
- **ac\_shell console**—Lets you use the command-line interface from within the graphical user interface.

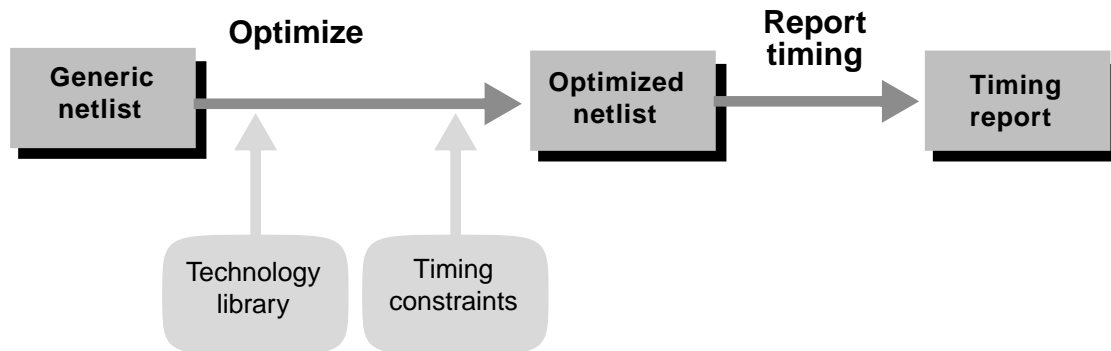
## Envisia Timing Analysis

Envisia® timing analysis is tightly integrated into Ambit BuildGates. It analyzes the timing of your design, as follows:

1. It determines which paths need to be optimized to ensure that the design meets the timing constraints that you have provided.
2. It generates a timing report, so that you can verify that your design meets your constraints.

These steps are illustrated in [Figure 1-3](#) on page 7.

**Figure 1-3 Timing Analysis Steps**



## Envisia Test Synthesis

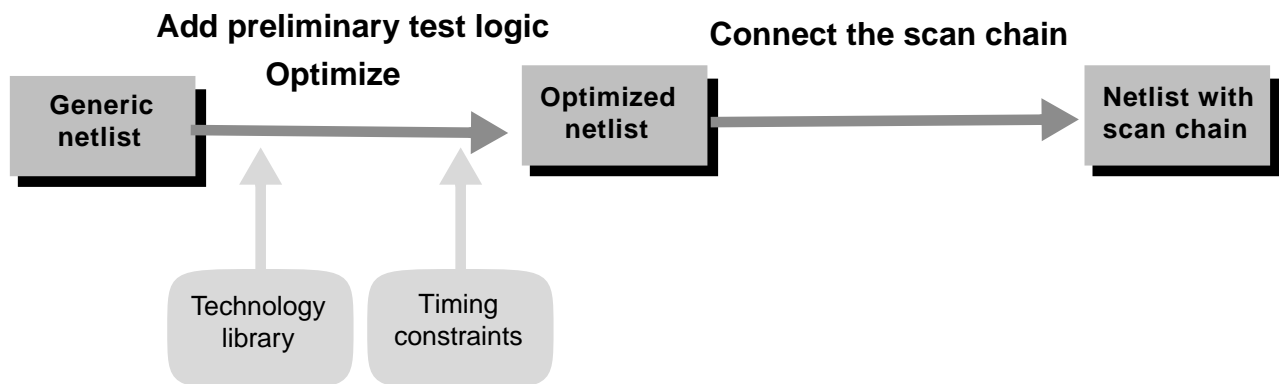
Envisia® test synthesis automates the process of adding design-for-test (DFT) logic to your designs. This test logic, or *scan chain*, does not affect the intended function of the chip. Rather, it lets the foundry verify that the chip works properly.

Envisia test synthesis can perform *one-pass scan insertion*, as follows:

1. Given a set of DFT assertions, it adds preliminary test logic to the design.
2. It generates a netlist that contains the preliminary test logic based on your technology library, and it optimizes the netlist to meet your timing constraints.
3. It connects the scan chain into the optimized netlist.

Figure 1-4 on page 7 illustrates these steps.

**Figure 1-4 Scan Insertion Steps**



# Ambit and Envisia Synthesis Tutorial

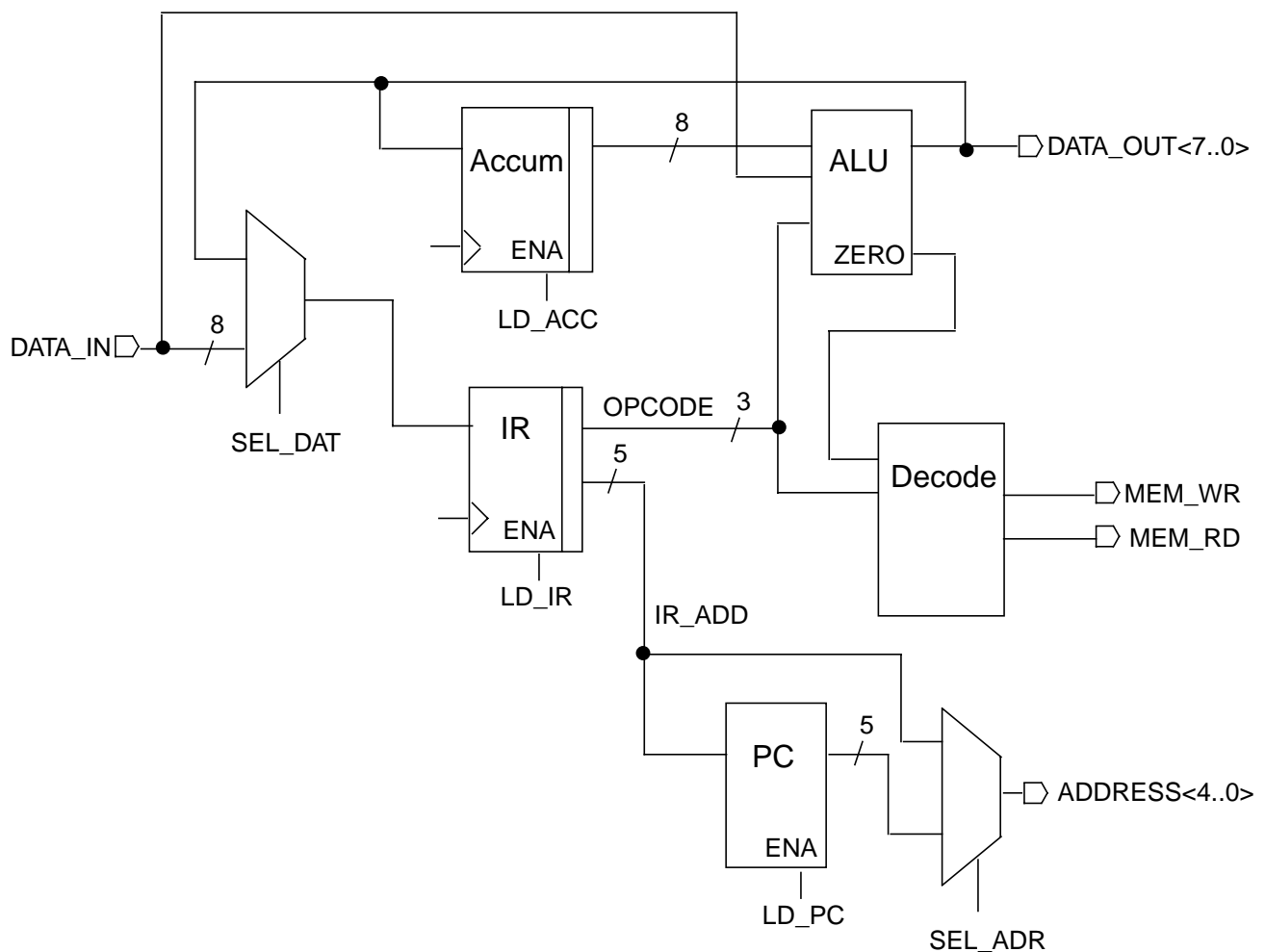
## Introduction

Because it adds test logic prior to and during optimization, Envisia test synthesis can reduce the impact of the added logic on the area and timing of your design.

## The CPU Example

This document takes you through a few synthesis scenarios with a simple CPU design. This design, shown in [Figure 1-5](#) on page 8, is made up of several modules—accumulator, arithmetic logic unit, instruction register, program counter, and decoder.

**Figure 1-5 CPU Design**



The source files for the RTL design, the gate-level netlist, and the library that these designs reference are stored in the *your\_install\_dir/demo/flow* directory, where *your\_install\_dir* represents the top of your Cadence installation hierarchy.

# Ambit and Envisia Synthesis Tutorial

## Introduction

---

If you want to run the examples in this document, you must change to a working directory and copy the example directories, as follows:

```
cp -r your_install_dir/demo/flow .
```

By running the examples in this document, you will see how you can use the Ambit and Envisia tools at many points in the design process. However, please note that this document gives you only a quick introduction to the tool. You can read more about the Ambit and Envisia tools in the Ambit and Envisia online documentation.

## Defining Environment Variables

Before you use the Ambit and Envisia tools, you must define the following environment variables (where *your\_install\_dir* is the top-level directory in which the tools are installed).

Variable	Description
CDS_LIC_FILE	Specifies the path to the Cadence license file on your system.
LD_LIBRARY_PATH (Solaris) or SHLIB_PATH (HPUX)	Specifies the path to the directory in which your Cadence shared libraries have been installed (usually <i>your_install_dir/tools/lib</i> ).
PATH	Specifies the default search path for binary files. This variable must include the path to the directory in which the Ambit and Envisia executable files are installed.
AMBIT_SLIB_PATH	Specifies the search path for technology libraries. If you do not define this variable, you must specify the entire directory path for the libraries that you use.

## More Information

For more information about the Ambit and Envisia tools described here, please refer to the following documents:

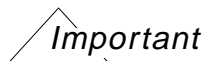
- *Ambit BuildGates User Guide*
- *Envisia Timing Analysis User Guide*
- *Envisia Test Insertion User Guide*

---

## Synthesizing a Design from the Top Down

---

*Top-down synthesis* is the most desirable method of synthesis. Using this method, you can apply optimizations and perform timing verification of the design as a whole. This chapter describes how to synthesize the CPU design from the top down using the command-line interface.



### Important

All of the commands in this chapter assume that you are running from the `flow` directory in your example hierarchy.

## Invoking the Synthesis Tool

To invoke Ambit BuildGates, enter the following command from the `flow` directory:

```
ac_shell
```

After Ambit BuildGates displays a copyright notice, it displays the `ac_shell` prompt, as follows:

```
ac_shell[1]>
```

The number in brackets increments after each command that you enter.

## Reading a Technology Library

A *technology library* defines the characteristics of the gates that you are going to use in your design. All technology library files must have the `.alf` suffix. AMBIT Library format (ALF) libraries contain compacted, optimized and precomputed data that load quickly into the synthesis tool. You can generate these libraries with the Ambit Technology Compiler, `libcompile`. The Ambit BuildGates installation provides several libraries that you can use. This example uses the `lca300k.alf` library.

To read the `lca300k.alf` library into the synthesis database, enter the following command:

```
read_alf lca300k.alf
```

## Ambit and Envisia Synthesis Tutorial

### Synthesizing a Design from the Top Down

---

Ambit BuildGates displays the following messages as it loads the library into its internal database:

```
Info:      Library 'lca300kv [compiled with LIBCOMPILE{v4.0-b004 (Jul 27 2000
          15:32:47)}}] was loaded from file
          'your_install_dir/lib/technology/ambit/alf/lca300k.alf'
          <TCLCMD-701>.
```

```
lca300kv
```

When it loads the library, Ambit BuildGates makes `lca300k` the target technology. Whenever Ambit BuildGates maps a gate in the design to a specific library cell, it uses that technology library.

## Reading the Design Modules

You are now ready to read the design source files into the synthesis tool's internal database. As Ambit BuildGates reads the files, it parses them and reports any syntax errors that it finds. It creates a parse tree that other commands use during synthesis.

To read the CPU design into the synthesis tool, enter the following command:

```
read_verilog "alu_rtl.v count5_rtl.v cpu_rtl.v decode_rtl.v reg8_rtl.v"
```

## Building a Generic Netlist

After Ambit BuildGates has read the HDL source files, you must convert them into generic logic with the `do_build_generic` command. The `do_build_generic` command generates a generic, hierarchical netlist for all of the modules in the design. This netlist uses technology-independent logic gates, defined in the AMBIT Technology Library (ATL) or the Extended AMBIT Technology Library (XATL). Operators such as adders and shifters are instantiated as black boxes at this stage of the synthesis process. That is, their internal implementation is unknown at this time.

To create a generic netlist, enter the following command:

```
do_build_generic
```

Ambit BuildGates displays the following messages as it processes each module:

```
Info:      Processing design 'cpu' <CDFG-303>.
Info:      Processing design 'reg8' <CDFG-303>.
Info:      Processing design 'alu' <CDFG-303>.
```

Each case statement in the design is reported in a table similar to the following:

```
Statistics for case statements in module 'alu' (File alu_rtl.v)
<CDFG-800>.
+-----+
|               Case Statistics Table               |
+-----+
```

## Ambit and Envisia Synthesis Tutorial

### Synthesizing a Design from the Top Down

---

Line	Type	Full	Parallel
20	case	AUTO	AUTO

Each sequential device that the `do_build_generic` command infers is reported in a table similar to the following:

Table for sequential elements								
File Name	Line	Register Name	Type	Width	AS	AR	SS	SR
reg8_rtl.v	10	dataOut_reg	D_FF	8	N	Y	N	N

This table shows the name of the source file and the line number in that file at which the sequential element is defined. The table also shows the name of the register that is associated with the sequential element and the type of generic cell that the synthesis tool has chosen to represent the element. In this example, the synthesis tool selected a D flip-flop that has a width of 8 bits.

The remaining columns describe characteristics of the sequential element. For example, this register does not have an asynchronous set (AS) control. It does have an asynchronous reset (AR) control. It does not have a synchronous set (SS) or a synchronous reset (SR) control.

When it has processed all of the modules in the design, Ambit BuildGates displays the following messages:

```
Finished processing module: 'cpu' <ALLOC-110>.
Info: Setting 'cpu' as the top of the design hierarchy <FNP-704>.
Info: Setting 'cpu' as the default top timing module <FNP-705>.
```

Ambit BuildGates sets the top of the design hierarchy to `cpu`, and it sets `cpu` as the default top timing module. For top-down synthesis, you want the current top module to be at the top of the design hierarchy, and you want the timing constraints to apply to all of the modules in the design, from the top down. Therefore, you do not need to change these settings.

## Setting Timing Constraints

For all sequential logic, you specify timing constraints with respect to an ideal clock. An *ideal clock* lets the logic synthesis process determine the intended relationship between various clocks and clock ports. You define the period and cycle duty for an ideal clock, as follows:

```
set_clock clk1 -period 4 -waveform "0 2"
```

In this example, the `set_clock` command defines an ideal clock named `clk1`. This ideal clock has a period of 4ns, a rising edge of 0ns, and a falling edge of 2ns.

## Ambit and Envisia Synthesis Tutorial

### Synthesizing a Design from the Top Down

---

After defining the ideal clock, you must bind a physical clock pin in the design to this ideal clock. The actual arrival times — rising edge and falling edge — for a clock signal on the clock port of a module may be different from the ideal clock. Therefore, in this example, you must specify how the clock port of the CPU (`clock`) behaves in relation to the ideal clock (`clk1`), including the arrival time of the clock signal to the pins of the sequential elements. You define this relationship with the `set_clock_arrival_time` command, as follows:

```
set_clock_arrival_time -clock clk1 -early -late -rise 0.1 -fall 2.1 clock
```

This command associates the `clock` signal with the ideal clock signal, `clk1`, by establishing a rising edge at 0.1ns and a falling edge at 2.1ns.

## Defining Data Arrival and Required Times

Data arrival times and data required times specify the length of the delay that a signal experiences due to other devices that are connected externally. The arrival time is the amount of time that it takes for data to arrive at the input ports of the top-level module.

You define the data arrival time and associate it with the ideal clock by using the `set_data_arrival_time` command. For example:

```
set_data_arrival_time 1.0 -clock clk1 [find -inputs -noclocks]
```

This command specifies that the data arrives at all input signals at 1.0ns, with respect to the ideal clock, `clk1`. The `find` command locates all of the input ports to which you want to apply the constraints.

The `set_data_arrival_time` command in the previous example applies to both setup and hold times. If you want to specify separate arrival times for setup and hold checks, you need to issue two separate commands using the `-early` and `-late` options. For example:

```
set_data_arrival_time 0.5 -early -clock clk1 [find -inputs -noclocks]
set_data_arrival_time 1.0 -late -clock clk1 [find -inputs -noclocks]
```

The `-early` arrival time setting is associated with the hold timing checks; the `-late` setting is associated with the setup timing checks. The `find` command locates all of the inputs on which you want to apply the constraints.

**Note:** For combinational logic, the data arrival time is independent of the clock. Therefore, you do not include the `-clock` option for a combinational input port.

The `set_external_delay` command models the delay that is associated with designs that are downstream from this design. The external delay must be relative to the ideal clock. For example, if you assume that the downstream device and all interconnecting delays account for a delay of 0.4ns, you can issue the following command:

```
set_external_delay 0.4 -clock clk1 [find -outputs]
```

## Ambit and Envisia Synthesis Tutorial

### Synthesizing a Design from the Top Down

---

The `-late` and `-early` options can define separate delays for setup and hold, just as they do for data arrival times.

## Optimizing the Design

The `do_optimize` command performs logic optimization of the generic netlist. This command maps the resulting logic to the cells in the technology library, and ensures that the resulting logic does not violate any timing constraints.

To map the design to the technology library and optimize it, enter the following command:

```
do_optimize
```

Mapping occurs in a number of steps, as indicated by the following messages:

```
Info: Dissolving AmbitWare instance 'i_337' (cellref 'AWMUX_2_8') in
Info: module 'alu' ... <TCLNL-605>.
Info: Dissolving AmbitWare instance 'i_320' (cellref 'AWMUX_8_8') in
Info: module 'alu' ... <TCLNL-605>.
Info: Dissolving AmbitWare instance 'i_567' (cellref 'AWMUX_2_5') in
Info: module 'count5' ... <TCLNL-605>.
Info: Dissolving AmbitWare instance 'i_566' (cellref 'AWMUX_2_5') in
Info: module 'count5' ... <TCLNL-605>.
Info: Dissolving AmbitWare instance 'i_565' (cellref 'AWMUX_2_5') in
Info: module 'count5' ... <TCLNL-605>.
Info: Dissolving AmbitWare instance 'i_54' (cellref 'AWMUX_2_8') in
Info: module 'reg8' ... <TCLNL-605>.
Info: Dissolving AmbitWare instance 'i_1292' (cellref 'AWMUX_2_5') in
Info: module 'cpu' ... <TCLNL-605>.
Info: Dissolving AmbitWare instance 'i_176' (cellref 'AWACL_UNE_EQ_8')
Info: in module 'alu' ... <TCLNL-605>.
Info: Dissolving AmbitWare instance 'i_536' (cellref 'AWACL_UNE_EQ_5')
Info: in module 'count5' ... <TCLNL-605>.
Info: Duplicated module 'reg8' as 'reg8_0' and bound to instance 'iregl'
Info: in module 'cpu' <FNP-700>.
Info: Duplicated module 'reg8' as 'reg8_1' and bound to instance
Info: 'accum1' in module 'cpu' <FNP-700>.
Info: Propagating constants ... <TCLNL-505>.
Info: Dissolving AmbitWare instance 'i_564' (cellref 'AWACL_UNE_INC_5_
Info: C') in module 'count5' ... <TCLNL-605>.
Info: Structuring module 'reg8_1' ... <TCLNL-500>.
Info: Structuring module 'reg8_0' ... <TCLNL-500>.
Info: Structuring module 'count5' ... <TCLNL-500>.
Info: Structuring module 'decode' ... <TCLNL-500>.
Info: Structuring module 'alu' ... <TCLNL-500>.
Info: Structuring module 'cpu' ... <TCLNL-500>.
Info: Propagating constants ... <TCLNL-505>.
Info: Removing redundancies ... <TCLNL-504>.
Info: Mapping module 'AWACL_UNE_ADD_8_C' ... <TCLNL-501>.
Info: Mapping module 'alu' ... <TCLNL-501>.
Info: Mapping module 'count5' ... <TCLNL-501>.
Info: Mapping module 'decode' ... <TCLNL-501>.
Info: Mapping module 'reg8_0' ... <TCLNL-501>.
Info: Mapping module 'reg8_1' ... <TCLNL-501>.
Info: Mapping module 'cpu' ... <TCLNL-501>.
```

## Ambit and Envisia Synthesis Tutorial

### Synthesizing a Design from the Top Down

---

After it has mapped the cells in the technology library to the gates in your design, Ambit BuildGates optimizes the design. The tool may go through several optimization steps before it completes the entire process. After each optimization step, you may see the late slack time decrease. For example:

```
Info:      Optimizing module 'cpu' to meet constraints(medium effort) ...
          <TCLNL-506>.
```

cpu			
Cell area	Net area	Total area	Late slack
636.50	0.00	636.50	0.0412

```
Critical Begin Point(s): decode1_state_reg_1_Q <TOPT-515>.
Critical End Point(s):  alu1_aluout_reg_0_D <TOPT-516>.
Fixing design rule violations ... <TOPT-505>.
Fixed all design rule violations <TOPT-405>.
```

cpu			
Cell area	Net area	Total area	Late slack
638.50	0.00	638.50	0.2651

```
*** Checking endpoints ...
*** Finished checking endpoints ...
```

When it has completed the optimizations and cell mapping, Ambit BuildGates reports the size of the design and, if timing constraints have been satisfied, any late slack that it detects. In this example, Ambit BuildGates reports a positive slack time. This indicates that the design meets the timing constraints.

## Generating a Timing Report

To generate the timing report, enter the following command:

```
report_timing
```

The first part of the timing report shows the options that you used to generate the report, the version of the tool that you are running, and information about the type of timing analysis that you performed. For example, this report shows the results of a late mode analysis:

## Ambit and Envisia Synthesis Tutorial

### Synthesizing a Design from the Top Down

---

Report	report_timing
Options	
Date	20000808.101153
Tool	ac_shell
Release	v4.0-b004
Version	Jul 27 2000 19:09:27
Module	cpu
Timing	LATE
Slew Propagation	FAST
Operating Condition	NOM
PVT Mode	worst_case
Tree Type	balanced
Process	1.00
Voltage	5.00
Temperature	25.00
time unit	1.00 ns
capacitance unit	1.00 pF
resistance unit	1.00 kOhm

The next part of the timing report shows the critical path of this design. The critical path in this design has a positive slack after optimization, which means that all of the paths in the design have been optimized enough to meet the timing demands. A negative slack indicates that you need to reconsider your optimization strategy, make some design changes at the RTL level, or loosen your constraints—that is, give the logic more time.

**Note:** If you apply new constraints to reduce the slack time to 0, you must regenerate the timing report.

For example, the report shows the beginning and ending points of the critical path, from `iregl/dataOut_reg_6/Q` to `alul/aluout_reg_7/SI`, and it shows the timing results for that path:

```

Path 1: MET Setup Check with Pin iregl/dataOut_reg_1/CP
Endpoint:   iregl/dataOut_reg_1/D (^) checked with leading edge of 'clk1'
Beginpoint: iregl/dataOut_reg_6/Q (^) triggered by leading edge of 'clk1'
Other End Arrival Time      0.10
- Setup                      0.16
+ Phase Shift                4.00
= Required Time              3.94
- Arrival Time               3.67
= Slack Time                  0.27

```

The last part of the report shows the path itself, from pin to pin, including the module or cell through which the signal passed, and the delay, arrival, and required times at each point along the path:

## Ambit and Envisia Synthesis Tutorial

### Synthesizing a Design from the Top Down

Instance	Arc	Cell	Delay	Arrival Time	Required Time
	clock ^			0.10	0.37
iregl	clock ^	reg8_0		0.10	0.37
iregl/dataOut_reg_6	CP ^ -> Q ^	FD2	1.04	1.14	1.40
iregl	dataOut[6] ^	reg8_0		1.14	1.40
decode1	opcode[0] ^	decode		1.14	1.40
decode1/i_417	A ^ -> Z v	IV	0.29	1.42	1.69
decode1/i_12	B v -> Z ^	ND2	0.24	1.67	1.93
decode1/i_756	A ^ -> Z v	MUX21L	0.26	1.93	2.19
decode1/i_822	A v -> Z ^	NR2	0.87	2.80	3.06
decode1	sel_dat ^	decode		2.80	3.06
i_047	S ^ -> Z ^	MUX21SP	0.54	3.34	3.60
iregl	dataIn[1] ^	reg8_0		3.34	3.60
iregl/i_0	B ^ -> Z ^	MUX21SP	0.33	3.67	3.94
iregl/dataOut_reg_1	D ^	FD2	0.00	3.67	3.94

## Saving the Netlist

Ambit BuildGates stores in memory all of the logic synthesis data, including the netlist, constraints, and technology library cells. You can write this information in memory as a Verilog or VHDL netlist, or as an AMBIT database (ADB). You can use the netlist for gate-level verification. You can load an AMBIT database quickly into Ambit BuildGates to perform further synthesis or analysis of the netlist.

To save the netlist for this example design, enter the following command:

```
write_verilog -hierarchical gates.v
```

To save the AMBIT database, enter the following command:

```
write_adb -hierarchical cpu.adb
```

**Note:** The AMBIT database is a binary data file; you should not try to edit or decompile it for any purpose.

## Exiting from Ambit BuildGates

To exit from Ambit BuildGates, enter the following command:

```
exit
```

Ambit BuildGates writes the following files to your run directory. These files give you a record of the synthesis steps that you have performed:

- `ac_shell.cmd` contains all of the commands that you entered during the session. You can use the commands in this file to generate a script with which to rerun this session.

## Ambit and Envisia Synthesis Tutorial

### Synthesizing a Design from the Top Down

---

- `ac_shell.log` contains all of the messages that Ambit BuildGates generated during the session. You can use this file as a record of the results of the synthesis session.
- `time_rpt $n$`  contains the timing report that Ambit BuildGates generated during the session. The number  $n$  is incremented every time you generate another report.

---

## Creating a Flattened Netlist

---

A *flattened netlist* is one in which all of the modules are collapsed into the top level of the hierarchy. For example, if you flatten the CPU design, there is only one module (`cpu`), and all of the submodules are contained within it. Flattened netlists are often necessary at the physical design stage, because many layout and place-and-route tools cannot handle hierarchical netlists.

This chapter shows you how to create a flattened netlist for the CPU design, using the Ambit BuildGates graphical user interface (GUI).

### Invoking the GUI

To invoke the GUI, enter the following command from the `flow` directory:

```
ac_shell -gui
```

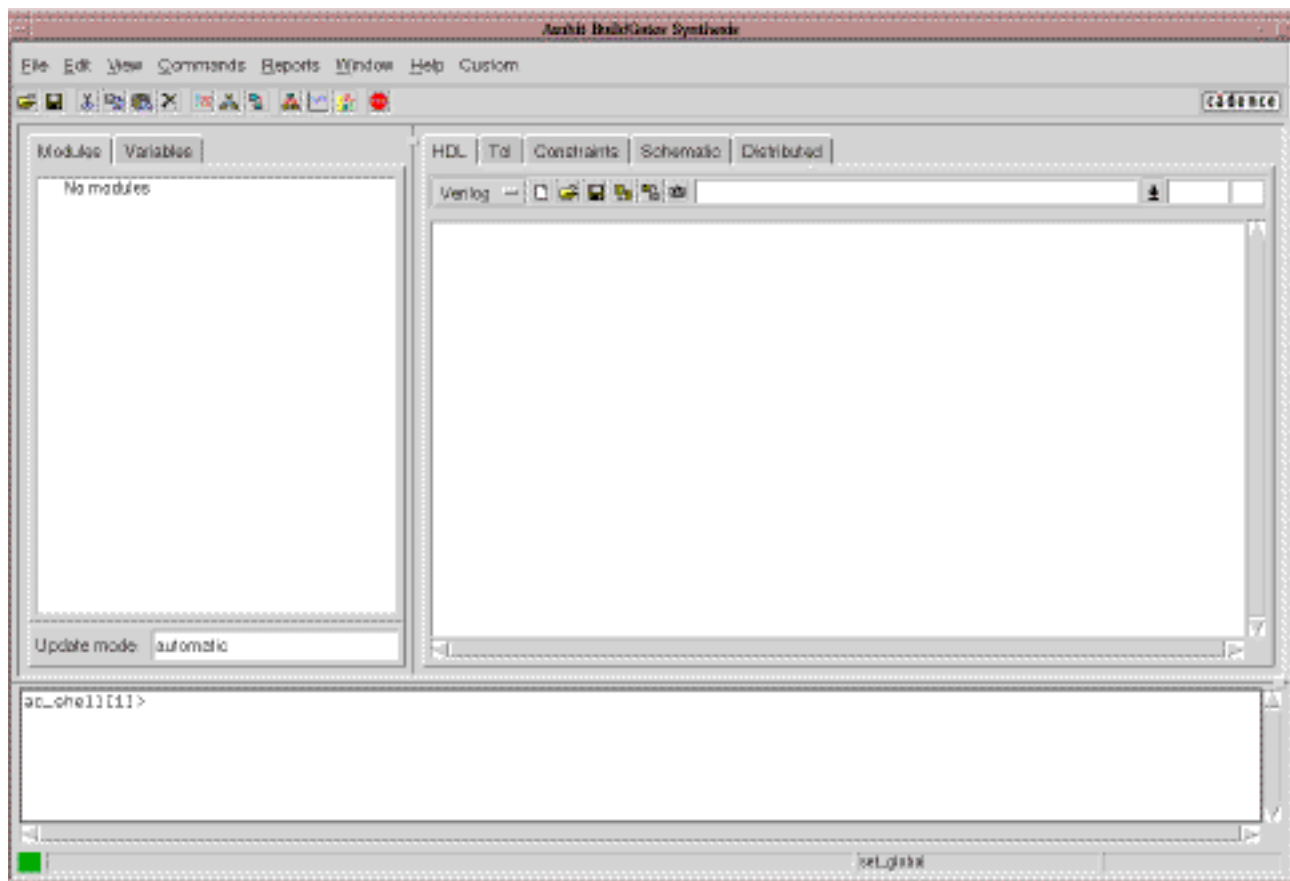
After the copyright notice appears, the GUI main window opens, as shown in [Figure 3-1](#) on page 20.

The main window gives you access to the synthesis functions, such as reading libraries and designs, defining timing constraints, and optimizing the netlist. The main window also gives you access to the module browser, the schematic browser, and all of the online documentation for the Ambit and Envisia tools.

## Ambit and Envisia Synthesis Tutorial

### Creating a Flattened Netlist

Figure 3-1 GUI Main Window



## Reading a Technology Library

The technology library defines the characteristics of the gates that you are going to use in your design. All technology library files must be precompiled and have the `.alf` suffix. These libraries are generated by the Ambit Technology Compiler. They contain compacted, optimized, and precomputed data that you can load quickly into the synthesis tool.

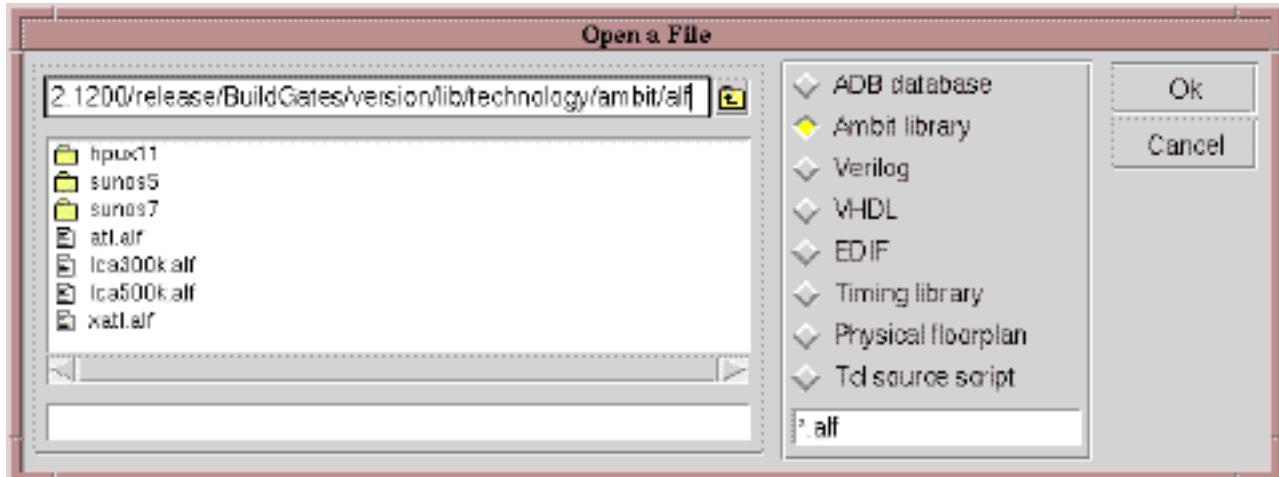
To read a technology library:

1. Click the Open File icon or choose *File–Open* from the menu bar. This opens the Open a File form.
2. Select *Ambit Library* from the list of file types. When you do, the GUI displays the technology libraries in the Ambit installation hierarchy, as shown in [Figure 3-2](#) on page 21.

## Ambit and Envisia Synthesis Tutorial

### Creating a Flattened Netlist

Figure 3-2 Reading an ALF File



3. Select `lca300k.alf` from the list of files and click *OK*.

## Reading the Design Modules and Building the Generic Netlist

You are now ready to read the design source files and build a generic netlist. When it builds the generic netlist, Ambit BuildGates uses the AMBIT Technology Library (ATL) or Extended Ambit Technology Library (XATL) cells to produce a hierarchical gate-level representation of your design.

To read and build the CPU design:

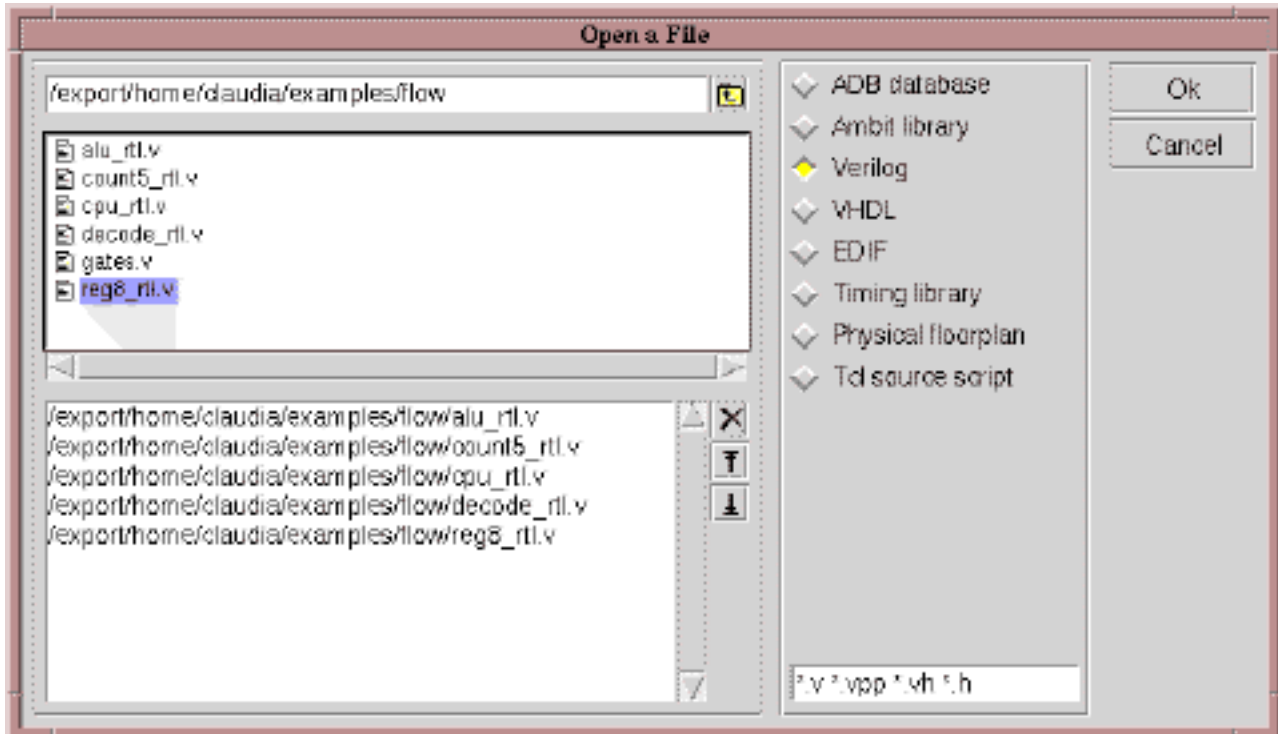
1. Click the Open File icon or choose *File–Open* from the menu bar. The GUI opens the Open a File form.
2. Select *Verilog* from the list of file types, and the GUI displays a list of the Verilog files that you can load from the `flow` examples directory.
3. Select the files that make up the CPU design—`alu_rtl.v`, `count5_rtl.v`, `cpu_rtl.v`, `decode_rtl.v`, and `reg8_rtl.v`.

As you select each file, it appears in the list of files, as shown in [Figure 3-3](#) on page 22.

## Ambit and Envisia Synthesis Tutorial

### Creating a Flattened Netlist

Figure 3-3 Reading Verilog Files

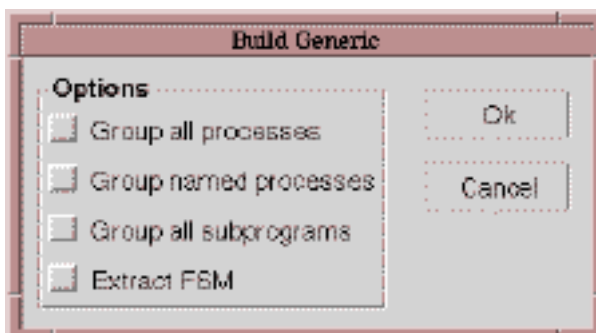


If you select a file that does not belong in the design, you can remove it from the list. First, select the file that you want to remove. Then click on the *X* button to the right of the list.

Click *OK* to read the files into the synthesis database.

4. Click the Build Generic icon or select *Commands–Build Generic* from the menu bar. The GUI pops up the Build Generic form, shown in [Figure 3-4](#) on page 22.

Figure 3-4 Building a Generic Netlist



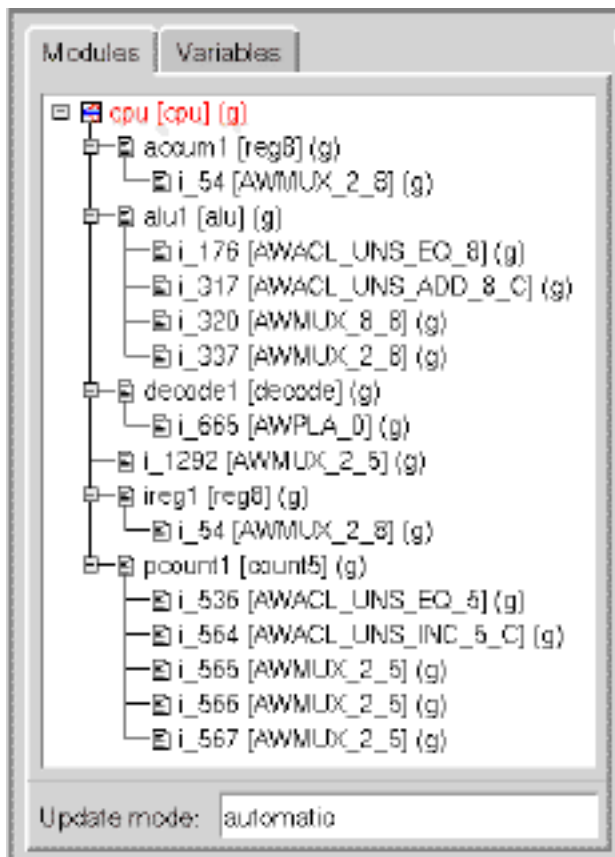
Click *OK* to build the generic netlist.

## Ambit and Envisia Synthesis Tutorial

### Creating a Flattened Netlist

The GUI builds a generic netlist for the design and displays the netlist in the Modules tab, as shown in [Figure 3-5](#) on page 23.

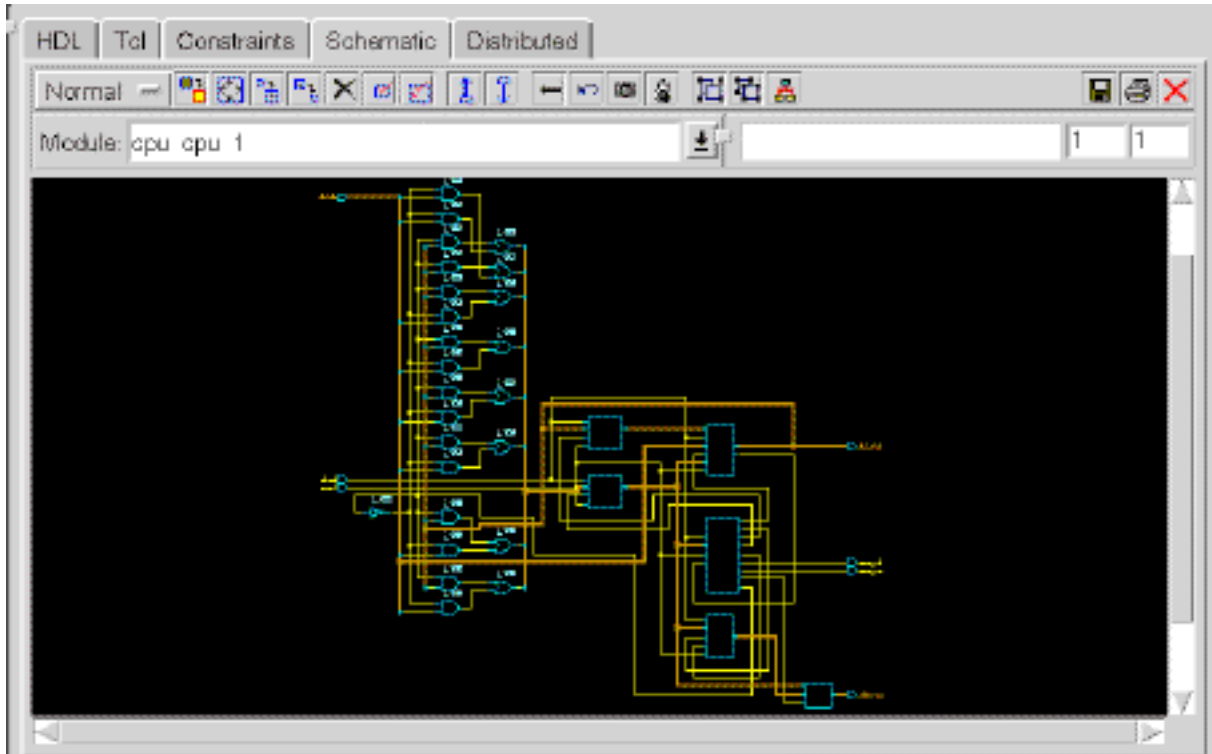
**Figure 3-5 Modules Tab**



The (g) symbols that appear after `accum1`, `alu1`, `decode1`, `ireg1`, and `pcount1` indicate that these instances have been mapped to generic library cells. (You have not yet mapped your design to the technology library.) The ACL in these component names stands for Ambit Component Library. These components are known good architectures that save you time, because they are preoptimized and are used to implement common functions. They are not mapped, however, until you get to the optimization and mapping stage.

5. Open the Schematic tab and double-click on `cpu` in the Modules tab. The GUI displays the schematic with `cpu` as the top-level module, as shown in [Figure 3-6](#) on page 24. You can double-click on other modules in the design to see the schematics for only those portions of the design.

**Figure 3-6 Schematic Tab for the CPU Design**



As you move your mouse cursor over the schematic, the GUI displays the name of the gate or wire to which you are pointing in the status bar at the bottom of the window.

## Defining the Timing Constraints

To specify timing constraints, you need to define an ideal clock, the clock arrival time, and the data setup and hold times for the design.

To define the timing constraints:

1. Open the Constraints tab, shown in [Figure 3-7](#) on page 25.

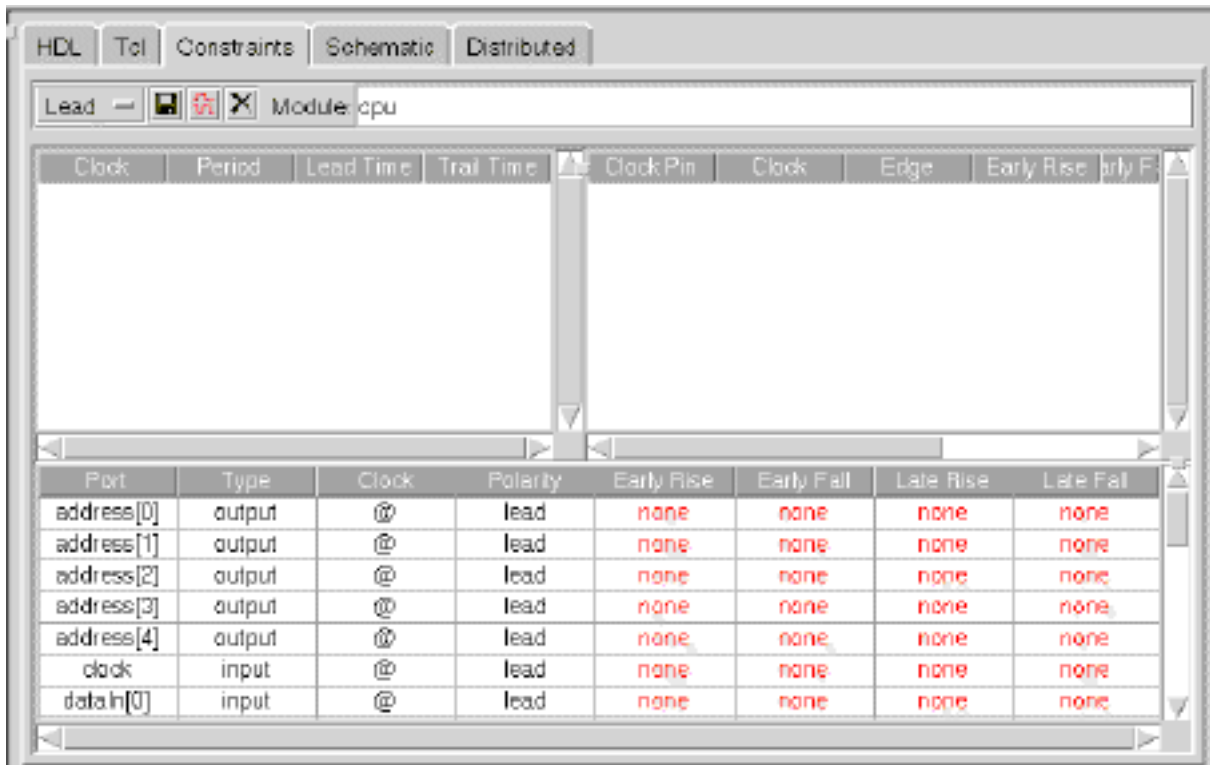
### *Important*

The top portion of this tab contains two tables—one that defines the ideal clock, and one that binds the clock ports of module instances to the ideal clock. If only the ideal clock table appears, use the split-pane slider to make both tables appear.

## Ambit and Envisia Synthesis Tutorial

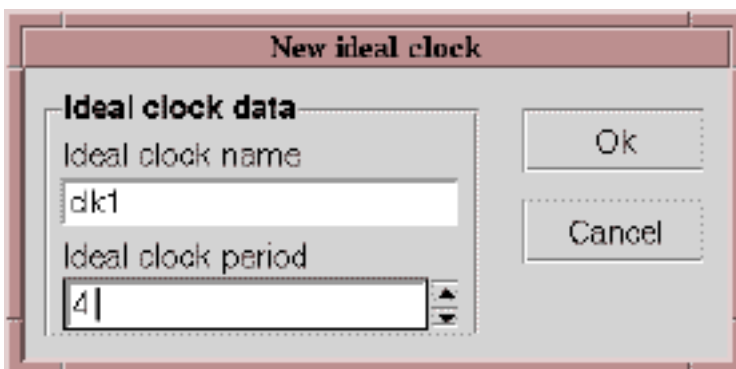
### Creating a Flattened Netlist

Figure 3-7 Constraints Tab



2. Click on the New Ideal Clock icon, or press MB3 inside the ideal clock table and choose *New Ideal Clock* from the pop-up menu. The GUI opens a form in which you define the ideal clock name and clock period, as shown in [Figure 3-8](#) on page 25.

Figure 3-8 Defining a New Ideal Clock



Enter `clk1` in the *Ideal clock name* field, enter `4` in the *Ideal clock period* field, and press Return.

## Ambit and Envisia Synthesis Tutorial

### Creating a Flattened Netlist

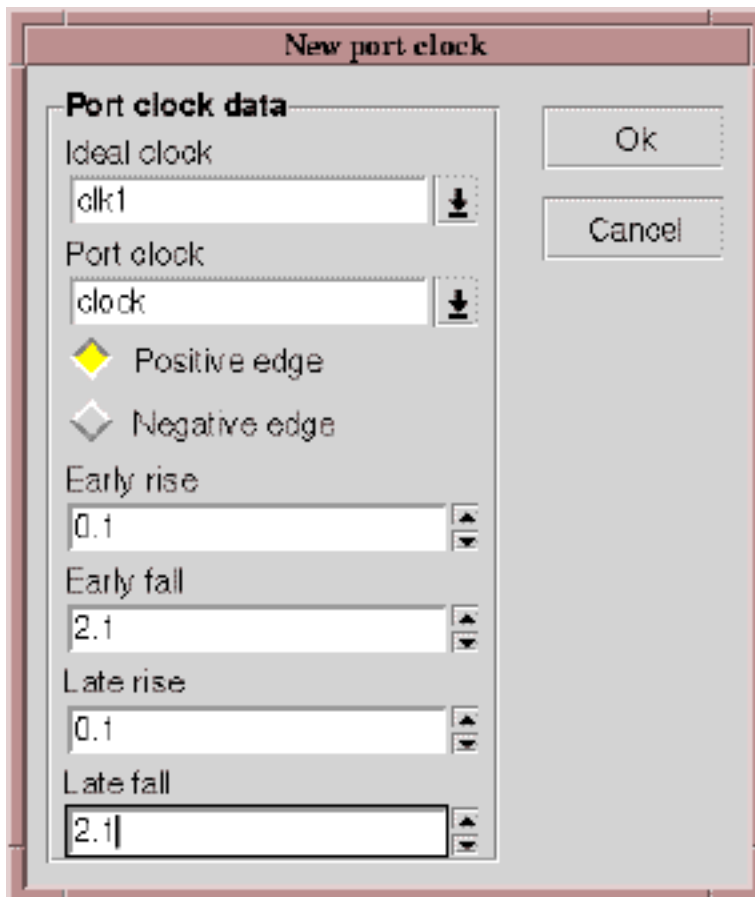
---

The GUI adds the ideal clock to the list of clocks.

3. To bind the ideal clock to the `clock` port of the `cpu` module, press MB3 over the clock port table and choose *New Port Clock* from the pop-up menu.

The GUI opens the New port clock form, as shown in [Figure 3-9](#) on page 26.

**Figure 3-9 Binding a Port to the Ideal Clock**



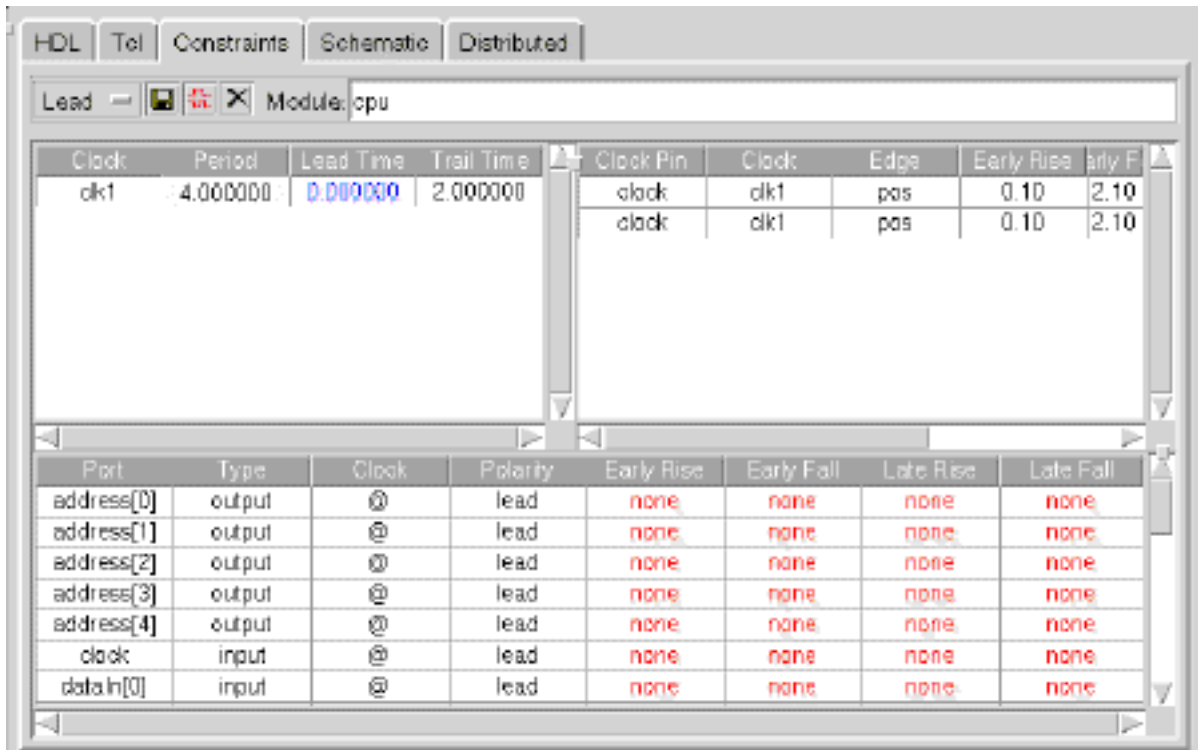
Choose `clk1` from the *Ideal Clock* pull-down menu, and choose `clock` from the *Port clock* pull-down menu. Enter 0.1 in the *Early rise* and *Late Rise* fields. Enter 2.1 in the *Early fall* and *Late fall* fields, and then click *OK*.

The GUI adds `clock` to the list of clock ports, as shown in [Figure 3-10](#) on page 27.

## Ambit and Envisia Synthesis Tutorial

### Creating a Flattened Netlist

**Figure 3-10 Setting the Clock Parameters**

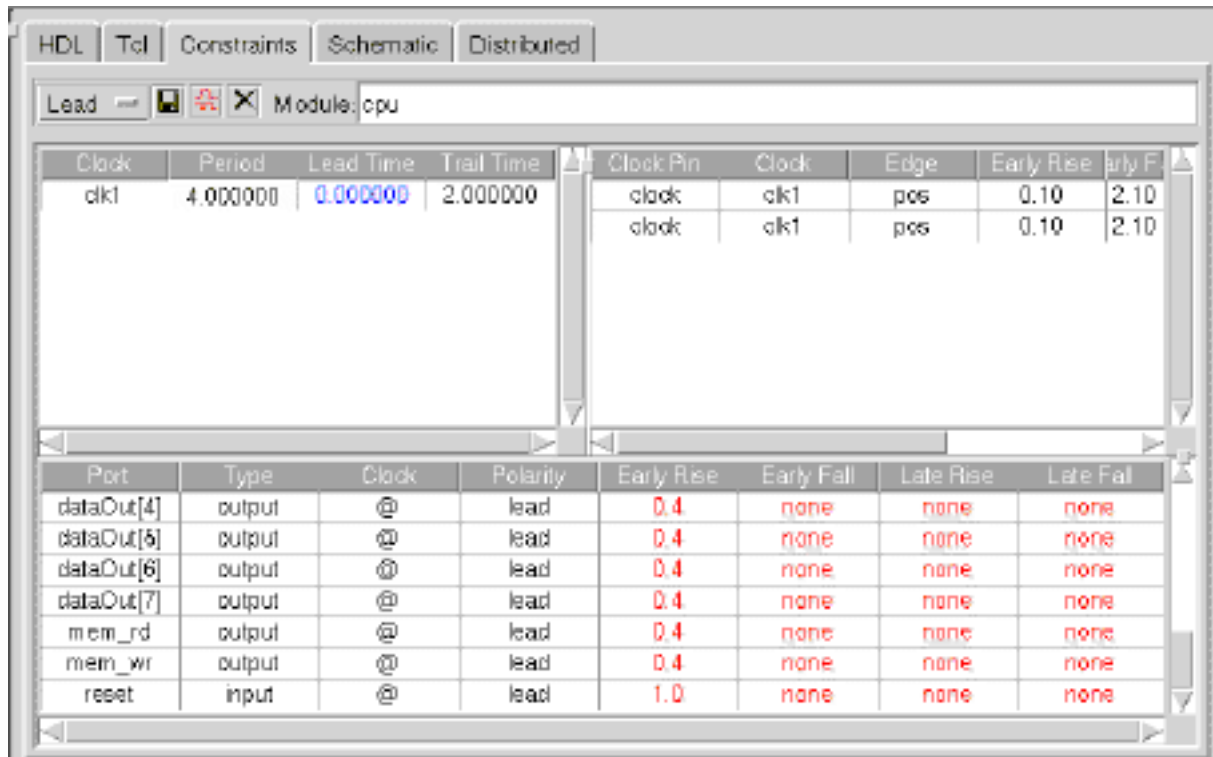


4. To set the data arrival time, place your cursor in the *Early Rise* column of each input port of the design, enter the value 1.0, and press `Return`.
5. To set the data required time, place your cursor in the *Early Rise* column of each output port, enter the value 0.4, and press `Return`. [Figure 3-11](#) on page 28 shows the data arrival times and data required times entered into the table.

## Ambit and Envisia Synthesis Tutorial

### Creating a Flattened Netlist

Figure 3-11 Setting the Data Arrival and Data Required Times



## Optimizing the Netlist

You can now bind your generic netlist to a technology library, check the timing, and optimize the netlist, as follows:

1. Click the Optimize icon or choose *Commands—Optimize* from the main menu. This opens the Optimize form, shown in [Figure 3-12](#) on page 29.

## Ambit and Envisia Synthesis Tutorial

### Creating a Flattened Netlist

---

Figure 3-12 Optimize Form

The screenshot shows the 'Optimize' dialog box with the following settings:

- Effort Level:** Medium (selected)
- Flatten Mode:** Off (selected)
- Priority:** Time (selected)
- Options:** All checkboxes are unchecked.
- Scan file:** scan

You can choose the effort level, flattening mode, and priority (to optimize the netlist for area or timing). The settings shown in the figure are the default settings.

Accept the default settings by clicking *Ok*.

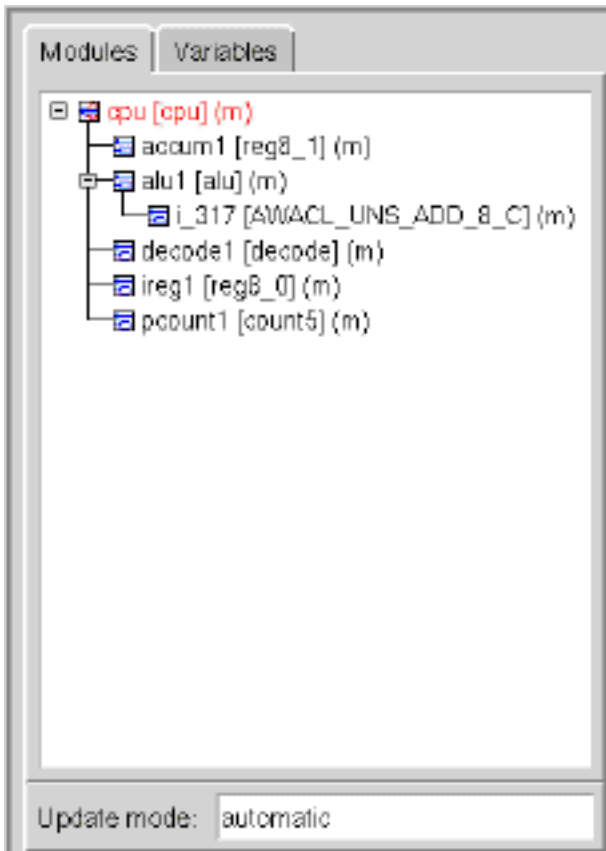
2. When optimization is complete, the GUI displays the optimized netlist, as shown in [Figure 3-13](#) on page 30.

## Ambit and Envisia Synthesis Tutorial

### Creating a Flattened Netlist

---

**Figure 3-13 Modules Tab with an Optimized Netlist**



Now that your design is optimized and mapped, the GUI displays (m) beside each module instance. This indicates that the instances have been mapped to cells in your technology library. You can also see that three instances have been removed during optimization.

## Flattening the Netlist

Flattening a netlist places all of the modules within the top-level module. That is, it removes the hierarchy from the design. This is often necessary before using the netlist with back-end tools, such as a layout or place-and-route tool, because this type of tool cannot usually handle hierarchical netlists. However, you can flatten a design at any time during development, because flattening can let the synthesis tool perform additional optimizations.

To flatten the netlist:

1. In the Modules tab, select `cpu` and choose *Set Current Module* from the pop-up menu.

## Ambit and Envisia Synthesis Tutorial

### Creating a Flattened Netlist

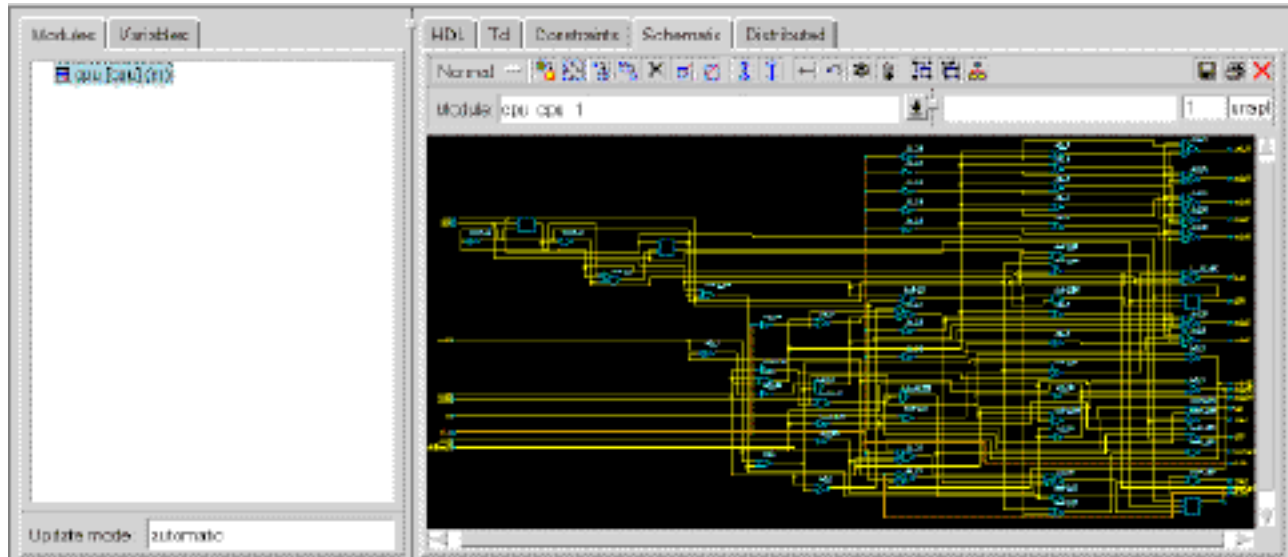
---

2. In the command area, enter the following command:

```
do_dissolve_hierarchy -hierarchical
```

The GUI displays the flattened netlist, as shown in [Figure 3-14](#) on page 31.

**Figure 3-14** Displaying the Flattened Netlist



## Generating the Timing Report

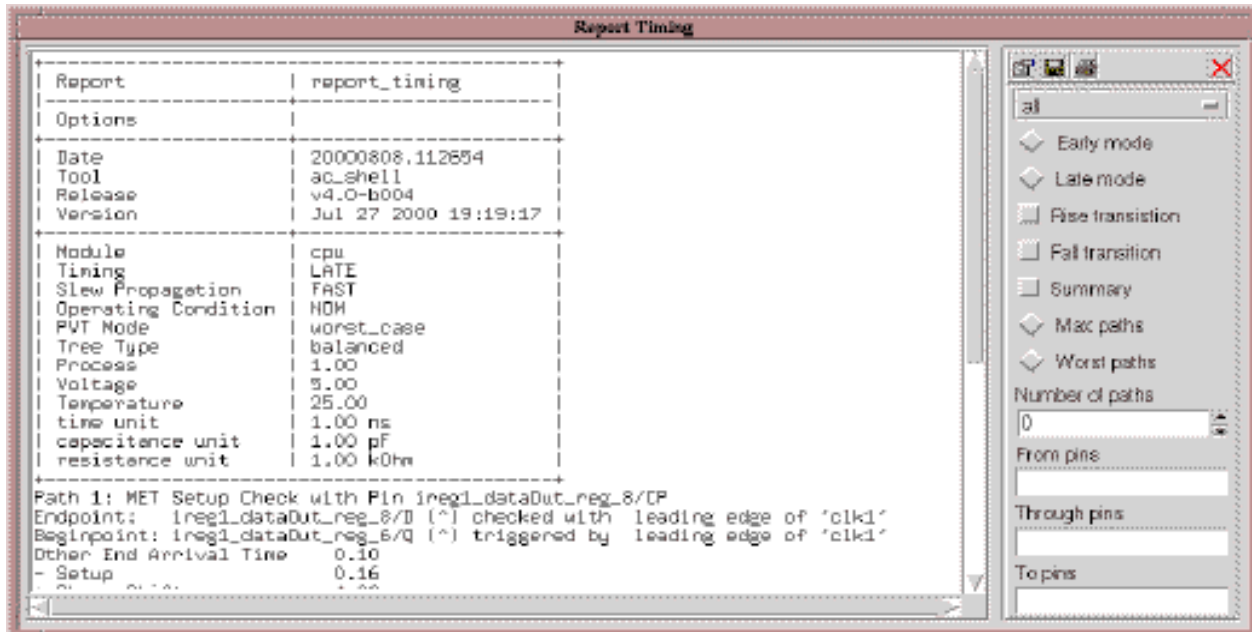
To generate a timing report:

1. Click the Reports icon or choose *Reports–Timing* from the menu bar. The GUI opens the Report Timing form.
2. Click the *Generate Report* icon (the leftmost icon in the top righthand side of the form). The GUI displays the report, as shown in [Figure 3-15](#) on page 32.

## Ambit and Envisia Synthesis Tutorial

### Creating a Flattened Netlist

Figure 3-15 Timing Report



The timing report is divided into three sections:

- ❑ The first section shows the options that you used to generate the report, the version of the tool that you are running, and information about the type of timing analysis that you performed. For example, this report shows the results of a late mode analysis.
- ❑ The next section shows the critical path that violated the timing constraints. The report gives the beginning and ending points of the path, and the timing results for that path.
- ❑ The last section shows the path itself, from pin to pin, including the module or cell through which the signal passed, and the delay, arrival, and required times at each point along the path.

## Saving the Netlist

You can write the synthesized design as a Verilog netlist or as an AMBIT database. You can use the Verilog netlist for gate-level verification. You can load an AMBIT database quickly into Ambit BuildGates to perform further synthesis or analysis of the netlist.

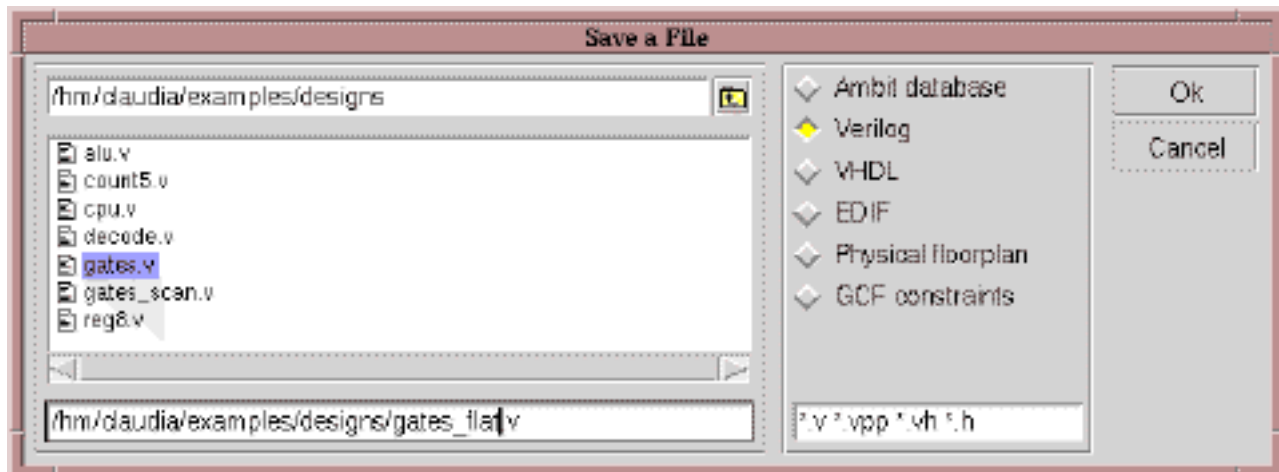
To save the netlist:

## Ambit and Envisia Synthesis Tutorial

### Creating a Flattened Netlist

1. Choose *File–Save* from the menu bar. This opens the Save a File form, as shown in [Figure 3-16](#) on page 33.

**Figure 3-16 Writing the Flattened Netlist**



2. Select *Verilog* from the list of file types.
3. In the *Files* field, enter the name that you want to give to the netlist—`gates_flat.v`—and click *OK*.

## Exiting from the GUI

To exit from the GUI, choose *File–Exit* from the menu bar. The GUI opens a confirmation dialog box, as shown in [Figure 3-17](#) on page 33. Click *OK*.

**Figure 3-17 Exit Confirmation Dialog Box**



---

## Synthesizing a Design from the Bottom Up

---

There are times when top-down synthesis is not possible or practical. For example:

- Your design may be too large for the synthesis tool to handle in its entirety.
- Different people may be working on different parts of the design, and those parts may not be ready for synthesis at the same time.

In these cases, you can perform *block-up* or *bottom-up synthesis*. Using these methods, you synthesize pieces of the design separately. When all of the pieces are complete, you synthesize the top-level module with minimal effort on the lower-level modules, which reduces run time and system resource usage.

Block-up synthesis and bottom-up synthesis differ only in the size of the pieces that you synthesize. In bottom-up synthesis, you synthesize one module at a time. In block-up synthesis, you synthesize functional blocks, which may be made up of several modules.

### Preparing for Synthesis

To start up the GUI and read the design files and library into the database:

1. Invoke the GUI by entering the following command from the `flow` directory:  

```
ac_shell -gui &
```
2. Click on the Open File icon or choose *File–Open* from the menu bar, and load the `lca300k.alf` library, as described in [Reading a Technology Library](#) on page 20.
3. Click on the Open File icon or choose *File–Open* from the menu bar, and select the Verilog design files `alu_rtl.v`, `count5_rtl.v`, `cpu_rtl.v`, `decode_rtl.v`, and `reg8_rtl.v`, as described in [Reading the Design Modules and Building the Generic Netlist](#) on page 21.
4. Click on the Build Generic icon or choose *Commands–Build Generic* from the menu bar to create the generic netlist with the generic Ambit technology library.

## Setting the Ideal Clock

To define the ideal clock, you can use the Constraints tab, as described in [Chapter 3, “Creating a Flattened Netlist”](#), or you can enter the command at the `ac_shell` prompt in the main window, as described in [Chapter 2, “Synthesizing a Design from the Top Down”](#).

- Enter the following command in the command area of the window:

```
set_clock clk1 -period 4 -waveform "0 2"
```

You do not need to repeat this step for any of the module instances, unless they require different clocks.

## Synthesizing Individual Design Blocks

The CPU design is made up of a top-level module (`cpu`) and four submodules. When synthesizing individual blocks, you begin with the lower-level modules. When you have synthesized all of the lower-level modules, you set a don't modify attribute on those submodules so that the synthesis tool does not modify them when you synthesize the top-level module. Each submodule can have its own set of timing constraints, so that you can make the timing of your design as tight as possible.

To synthesize an individual module in the design:

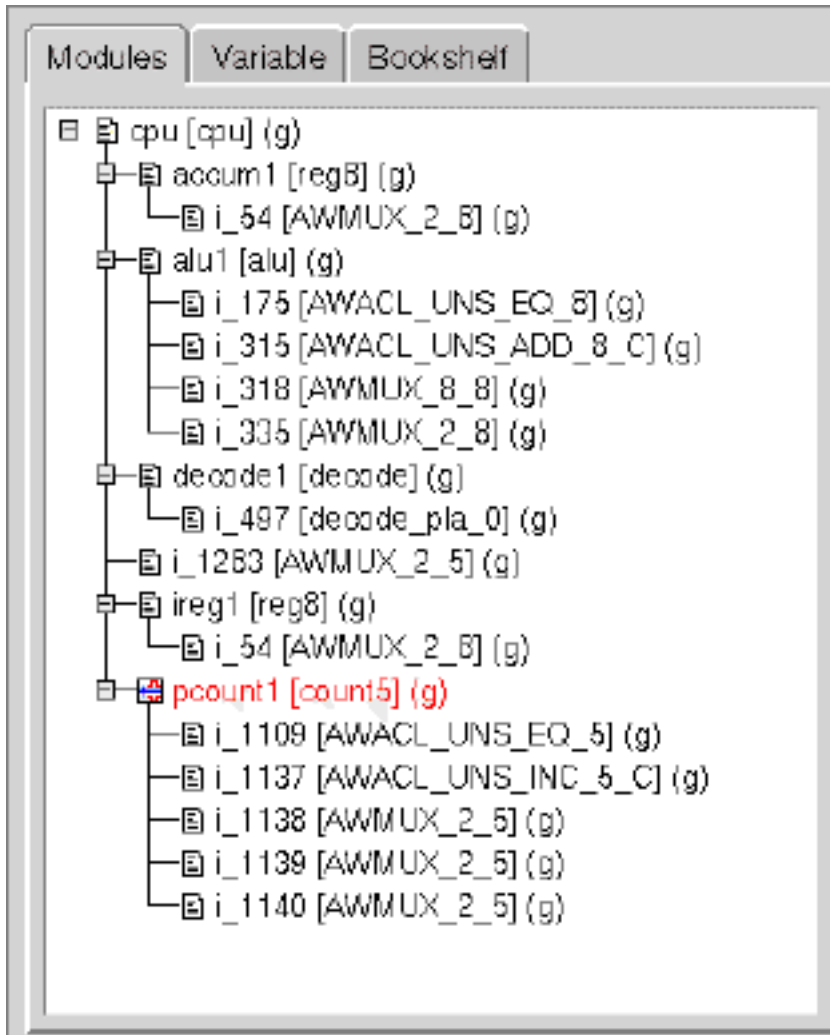
1. In the Modules tab, select the instance that you want to synthesize. For this example, you can begin with the `pcount1` instance.
2. Press `MB3` and choose *Set Current Module* from the pop-up menu.
3. Press `MB3` and choose *Set Top Timing Module* from the pop-up menu.

The GUI displays the module name in red, as shown in [Figure 4-1](#) on page 36.

## Ambit and Envisia Synthesis Tutorial

### Synthesizing a Design from the Bottom Up

Figure 4-1 Setting the Current Module and Top Timing Module



4. Bind the ideal clock to the clock pin of the `pcount` module, by entering the following command in the command area of the window:

```
set_clock_arrival_time -clock clk1 -early -late -rise 0.1 -fall 2.1 clock
```

5. Set the data arrival and data required times, by entering the following commands:

```
set_data_arrival_time 1.0 -clock clk1 [find -inputs -noclocks]
set_external_delay 0.4 -clock clk1 [find -outputs]
```

6. Click on the Optimize icon or choose *Commands–Optimize* from the menu bar. Click *OK* in the Optimize form to generate an optimized netlist for `pcount1` only.
7. Highlight the `pcount1` instance, press MB3, and choose *Set Don't Modify* from the pop-up menu. When you set this attribute, Ambit BuildGates performs no further

## Ambit and Envisia Synthesis Tutorial

### Synthesizing a Design from the Bottom Up

---

optimizations on the module when optimizing it as part of the larger design. The GUI displays (x) after the `pcount1` instance name to indicate that the don't modify attribute is set.

8. Choose *File–Save* from the menu bar. Save the module to a file called `count5_A.v`, and click *OK*.

Repeat these steps for each of the modules in the design, except for the `cpu` module. Save the modules as Verilog files with the names: `count5_A.v` for the `pcount1` instance, `reg8_A.v` for the `ireg1` and `accum1` instances, `decode_A.v` for the `decode1` instance, and `alu_A.v` for the `alu1` instance.

#### *Important*

The clock ports of the `decode` and `alu` modules are named `clk`, not `clock`. Therefore, to bind those clock ports to the ideal clock (Step 4), you must issue this command:

```
set_clock_arrival_time -clock clk1 -early -late -rise 0.1 -fall 2.1 clk
```

## Generating a Netlist for the Top Module in the Design

1. In the Modules tab, select `cpu`. Make this module the current module and the top timing module.

2. Bind the ideal clock to the clock pin of the `cpu` module, by entering the following command:

```
set_clock_arrival_time -clock clk1 -early -late -rise 0.1 -fall 2.1 clock
```

3. Set the data arrival and data required times, by entering the following commands:

```
set_data_arrival_time 1.0 -clock clk1 [find -inputs -noclocks]  
set_external_delay 0.4 -clock clk1 [find -outputs]
```

4. Click on the Optimize icon or choose *Commands–Optimize* from the menu bar. Click *OK* in the Optimize form.
5. Choose *File–Save* from the menu bar, and save this module to a Verilog file named `cpu_A.v`.
6. Choose *File–Exit* from the main menu and Click *OK* in the confirmation dialog box to exit from the GUI.

---

## Inserting a Scan Chain

---

You can insert scan chains at different times during development:

- For *one-pass scan insertion*, you begin with the RTL model for your design. You optimize the design and connect the scan chain in one synthesis run.
- For *two-pass scan insertion*, you begin with an optimized netlist. You analyze the timing of the design again after inserting the scan chain to ensure that your design still meets the timing requirements.

This chapter shows you how to perform one-pass scan insertion in a top-down synthesis run, using the VHDL version of the CPU design.

### Preparing for Synthesis

Before you can add the scan chain, you must start up Ambit BuildGates from the `flow` directory, read the library and design files, and build the generic netlist, as follows:

1. Start Ambit BuildGates from the UNIX command prompt:

```
ac_shell
```

2. Read the `lca300k.alf` library:

```
read_alf lca300k.alf
```

```
Info:      Library 'lca300kv [compiled with LIBCOMPILE{v4.0-b004 (Jul 27 2000
          15:32:47)}}] was loaded from file
          'your_install_dir/lib/technology/ambit/alf/lca300k.alf'
          <TCLCMD-701>.
```

```
lca300kv
```

3. Read the design source files:

```
read_verilog "alu_rtl.v count5_rtl.v cpu_rtl.v decode_rtl.v reg8_rtl.v"
```

4. Build the generic netlist:

```
do_build_generic
```

```
Info:      Processing design 'cpu' <CDFG-303>.
Info:      Processing design 'reg8' <CDFG-303>.
Info:      Processing design 'alu' <CDFG-303>.
```

## Ambit and Envisia Synthesis Tutorial

### Inserting a Scan Chain

---

```
.  
. .  
Info: Setting 'cpu' as the top of the design hierarchy <FNP-704>.  
Info: Setting 'cpu' as the default top timing module <FNP-705>.
```

## Setting Test Synthesis Assertions

Test synthesis assertions define the top-level module for the scan chain, the scan mode, and the scan style, as follows:

- Set the global variable for configuring the scan chain:

```
set_global dft_scan_path_connect tieback
```

This command instructs Envisia test synthesis to connect the scan cells in tieback mode. *Tieback mode* connects the scan cell's scan data output pin back to its own scan data input pin. This gives you a more accurate estimate of the load on the register's output pins than you can obtain if the scan data pins are unconnected, and it speeds up synthesis.

After you have optimized the design, you can connect the scan chain in chain mode.

## Adding the Scan Logic

After you have defined the test assertions, you must check that they are valid. If Envisia test synthesis finds no problems, you can add the scan logic to the design, as follows:

- Check that the DFT rules are correct:

```
check_dft_rules  
Info: Checking for registers with uncontrollable clock ports. <DFT-315>.  
Clock pins for dft_top_module cpu are:clock,  
Info: Checking for registers with uncontrollable set/reset ports.  
<DFT-310>.  
Clear/Preset pins for dft_top_module cpu are:reset,  
TDRC: set PI port reset to logic 1 during scan mode  
Info: Partitioning registers for scan based on clock domain. <DFT-320>.  
Clock Domain 0: PI clock (POSITIVE_EDGE) has 33 f/f  
Total Clock domains: 1 for 33 f/f  
Info: Checking for available scan cells in Library. <DFT-330>.  
Info: Design has no TDRC violation <DFT-343>
```

This command checks for DFT rule violations, such as uncontrollable clocks and uncontrollable asynchronous signals. If it detects any violations, Envisia test synthesis prints a message, and it does not insert scannable elements where the violations occur.

You can set certain global variables to override DFT violations. After you have set one of these global variables, you must invoke `check_dft_rules` again. Otherwise, the

## Ambit and Envisia Synthesis Tutorial

### Inserting a Scan Chain

---

original DFT violation designation remains in effect, and the affected elements are not mapped to scannable elements.

## Setting Timing Constraints and Optimizing the Design

After you have added the scan logic to your design, you can set timing constraints and optimize the design.

### 1. Set the timing constraints:

```
set_clock clk1 -period 4 -waveform "0 2"
set_clock_arrival_time -clock clk1 -early -late -rise 0.1 -fall 2.1 clock
set_data_arrival_time 1.0 -clock clk1 [find -inputs -noclocks]
set_external_delay 0.4 -clock clk1 [find -outputs]
```

### 2. Optimize the design and synthesize the netlist based on the target technology library:

```
do_optimize
Info: Duplicated module 'reg8' as 'reg8_0' and bound to instance 'iregl1'
      <FNP-700>.
Info: Duplicated module 'reg8' as 'reg8_1' and bound to instance
      'accum1' <FNP-700>.
Info: Propagating constants ... <TCLNL-505>.
Info: Structuring module 'alu' ... <TCLNL-500>.
.
.
.
Critical Begin Point(s): iregl1_dataOut_reg_8_Q <TOPT-515>.
Critical End Point(s): alu1_aluout_reg_0_D <TOPT-516>.
Critical Begin Point(s): iregl1_dataOut_reg_8_Q <TOPT-515>.
Critical End Point(s): alu1_aluout_reg_0_D <TOPT-516>.
```

cpu			
Cell area	Net area	Total area	Late slack
18124.80	0.00	18124.80	0.0044

Envisia test synthesis maps the design's registers to scan registers and connects the scan chains as specified by the `dft_scan_path_connect` global variable.

## Connecting the Scan Chain

Because you set the `dft_scan_path_connect` global variable to `tieback`, the scan data output of each scan register is connected back to its own input. After optimization, you must connect and configure the scan chain in chain mode, as follows:

### 1. Set the global variable for configuring the scan chain:

## Ambit and Envisia Synthesis Tutorial

### Inserting a Scan Chain

---

```
set_global dft_scan_path_connect chain
```

This command instructs Envisia test synthesis to connect the scan cells into one (default) or more scan chains. Setting this global variable to `chain` overrides the previous setting of `tieback`.

2. Define the name of the scan enable port and the value that activates the scan chain:

```
set_scan_mode SE 1
```

```
Will create scan enable port 'SE' for scan insertion <DFT-229>.
Info: Scan mode for module 'cpu' set to 'SE = 1'.
```

This command names the scan enable port `SE` and sets the value that activates the scan chain to `1`. By default, the scan enable port is named `BG_scan_enable`, and it is activated when the port is `1`.

3. Define the names of the scan input and scan output ports:

```
set_scan_data SI SO
```

```
Scan input port 'SI' will be created for scan insertion. <DFT-221>.
Scan output port 'SO' will be created for scan insertion. <DFT-224>.
Scan data I/O for module 'cpu' set to: <DFT-225>.
<no clock domain>: {SI SO}. <DFT-227>.
```

This command names the scan input port `SI` and the scan output port `SO`. By default, Envisia test synthesis names the scan input port `BG_scan_in`, and it names the scan output port `BG_scan_out`.

4. Connect the scan chain:

```
do_xform_connect_scan -hierarchical
```

```
Info: Connecting scan chains (mode = 'chain')... <DFT-026>.
      No scan connections to process in module 'AWACL_UNNS_ADD_8_C'
<DFT-011>.
      No scan connections to process in module 'AWMUX_8_8' <DFT-011>.
      1 chain (9 regs. total) connected in module 'alu'. <DFT-025>.
      1 chain (5 regs. total) connected in module 'count5'. <DFT-025>.
      1 chain (3 regs. total) connected in module 'decode'. <DFT-025>.
      1 chain (8 regs. total) connected in module 'reg8_0'. <DFT-025>.
      1 chain (8 regs. total) connected in module 'reg8_1'. <DFT-025>.
      Top-level chain 1 (SI -> SO) has 33 registers. <DFT-024>.
```

Envisia test synthesis connects the scan registers into one scan chain and creates a scan chain report file in your run directory. By default, Envisia test synthesis generates a file name based on the name of the top-level module and the `scan` extension. In this example, the report file is named `cpu.scan`.

## Saving the Netlist and Exiting

You can save the netlist and exit Ambit BuildGates, as follows:

1. Write the netlist:

## Ambit and Envisia Synthesis Tutorial

### Inserting a Scan Chain

---

```
write_verilog -hierarchical gates_scan.v
--> WARNING: Verilog continuous assignments written (port-to-net connection).
        Type 'help multiport' for more information <VLOGWR-004>.
```

The `write_verilog` command results in a warning because Envisia test synthesis has added scan ports. This warning does not indicate a problem.

#### 2. Exit from Ambit BuildGates:

```
exit
```

## Viewing the Scan Chain File

Envisia test synthesis writes two files to your run directory. These files describe the scan chains that it has created — a hierarchical description (named `module.scan`) and a flat description of the scan chain (named `module.scan.flat`).

The hierarchical scan file is organized by module. Scan chain numbers and scan bit numbers restart from 1 in each module. For scan register cells, the file lists the scan cell pins that are contained in each module. For example, the `cpu.scan` file contains the following description of the scan cell pins for the `decode` module:

```
module decode
begin_chain 1 port SI
bit 1 state_reg_0/Q
bit 2 state_reg_1/Q
bit 3 state_reg_2/Q
end_chain 1 port SO
```

The `begin_chain` statement gives the number of the scan chain and the scan data input port name, `SI`. The `end_chain` statement gives the name of the data output port, `SO`. Between the `begin_chain` and `end_chain` statements, the file shows which bits of the register are connected to the scan chain.

In this example, the file shows that the `decode` module supplies the scan pins for scan chain number 1. These scan pins are named `state_reg_0/Q`, `state_reg_1/Q`, and `state_reg_2/Q` for pins 1, 2, and 3, respectively.

For the top-level module, the `cpu.scan` file shows how the bits of the scan chain correspond to registers in the module instances. For example:

```
module cpu
begin_chain 1 port SI
bit 8 accum1/ { SI : SO }
bit 17 alu1/ { SI : SO }
bit 20 decode1/ { SI : SO }
bit 28 ireg1/ { SI : SO }
bit 33 pcount1/ { SI : SO }
end_chain 1 port SO
```

## Ambit and Envisia Synthesis Tutorial

### Inserting a Scan Chain

---

In this example, bit 20 of the scan chain is connected to the last register in the `decode1` instance; it is connected to scan bits 18 through 20.

Flat scan files are organized by scan chain. Each scan chain is listed along with all of the scan bits that belong to the chain. The scan register that corresponds to each scan bit is identified with its full hierarchical name. Unlike hierarchical scan files, the bit numbering of flat scan files is absolute. For example, the `cpu.scan.flat` file contains the following description of the scan chain:

```
module cpu
begin_chain 1 port SI
bit 1 accum1/dataOut_reg_1/Q
bit 2 accum1/dataOut_reg_2/Q
bit 3 accum1/dataOut_reg_3/Q
bit 4 accum1/dataOut_reg_4/Q
bit 5 accum1/dataOut_reg_5/Q
bit 6 accum1/dataOut_reg_6/Q
bit 7 accum1/dataOut_reg_7/Q
bit 8 accum1/dataOut_reg_8/Q
bit 9 alu1/aluout_reg_0/Q
bit 10 alu1/aluout_reg_1/Q
bit 11 alu1/aluout_reg_2/Q
bit 12 alu1/aluout_reg_3/Q
bit 13 alu1/aluout_reg_4/Q
bit 14 alu1/aluout_reg_5/Q
bit 15 alu1/aluout_reg_6/Q
bit 16 alu1/aluout_reg_7/Q
bit 17 alu1/zero_reg/Q
bit 18 decode1/state_reg_0/Q
bit 19 decode1/state_reg_1/Q
bit 20 decode1/state_reg_2/Q
bit 21 ireg1/dataOut_reg_1/Q
bit 22 ireg1/dataOut_reg_2/Q
bit 23 ireg1/dataOut_reg_3/Q
bit 24 ireg1/dataOut_reg_4/Q
bit 25 ireg1/dataOut_reg_5/Q
bit 26 ireg1/dataOut_reg_6/Q
bit 27 ireg1/dataOut_reg_7/Q
bit 28 ireg1/dataOut_reg_8/Q
bit 29 pcount1/qout_reg_1/Q
bit 30 pcount1/qout_reg_2/Q
bit 31 pcount1/qout_reg_3/Q
bit 32 pcount1/qout_reg_4/Q
bit 33 pcount1/qout_reg_5/Q
end_chain 1 port SO
```

This file lists all 33 bits in scan chain 1 and the hierarchical names of the scan registers.

# Glossary

---

### **ALF**

Stands for Ambit Library Format, which was developed for Ambit BuildGates. This format can represent library information for synthesis and timing.

### **black box**

During static timing analysis, represents a block for which the functionality is not known, but for which the I/O pin-level timing is available.

See also *gray box*.

### **CTLF**

A compiled version of the information that is stored in a TLF file.

See also *TLF*.

### **DCL**

Stands for Delay Calculation Language, which is defined by the IEEE 1481 standard. This language is used to represent timing and power.

### **DCM**

Stands for Delay Calculation Module, which is obtained by compiling a DCL description.

See also *DCL*.

### **full scan**

A technique in which every storage element in the design is part of a scan chain.

### **gray box**

During static timing analysis, represents a block for which the functionality is not known, but for which the timing of I/O and internal pins are available.

### **hold**

The amount of time for which a signal on the data pin must remain stable after the active clock edge.

## Ambit and Envisia Synthesis Tutorial

### Glossary

---

#### **IPO**

Stands for *In-Place Optimization*, which is performed by timing-driven physical design tools.

#### **one-pass scan insertion**

A scan insertion technique in which scan chains are inserted in the design during RTL synthesis.

#### **partial scan insertion**

A scan insertion technique in which only some of the storage elements are part of the scan chain.

#### **SDF file**

A file that contains delay information in the Standard Delay Format. You use this file as input to Envisia timing analysis.

#### **setup**

The amount of time that a signal on the data pin must be stable before the active clock edge.

#### **static timing analysis**

A technique for analyzing the timing behavior of a design without using test vectors.

#### **timing constraints**

Specifications of the ideal clock and of the data arrival times for a design.

#### **TLF**

Stands for Timing Library Format, which represents timing information of a technology library.

#### **two-pass scan insertion**

A scan insertion technique in which scan chains are inserted after an optimized netlist has been obtained during the logic design phase.