Module 26

Next Steps

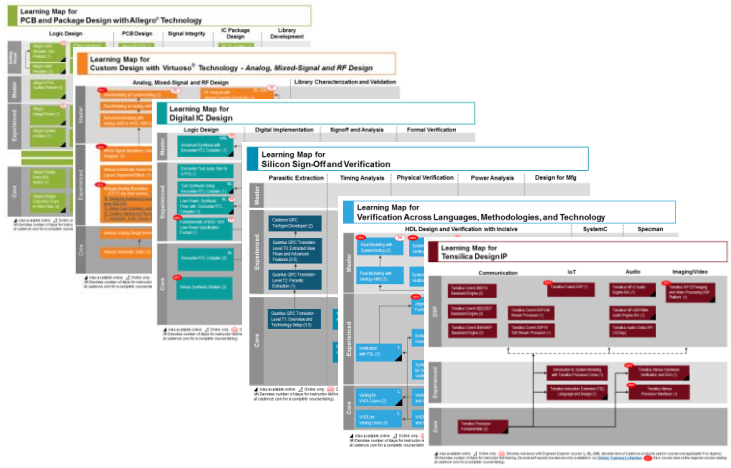
cadence

This page does not contain notes.

Learning Maps

Cadence® Training Services learning maps provide a comprehensive visual overview of the learning opportunities for Cadence customers.

Click here to see all our courses in each technology area and the recommended order in which to take them.



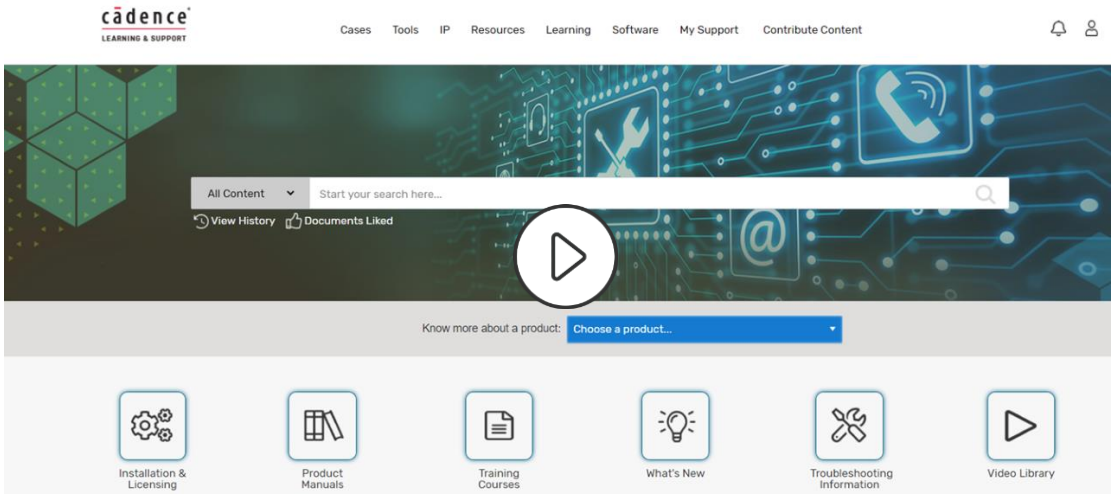
503 © Cadence Design Systems, Inc. All rights reserved.

cadence

Go here to view the learning maps:

http://www.cadence.com/Training/Pages/learning_maps.aspx

Cadence Learning and Support



Cadence Support now includes over 2000 product/language/methodology videos ("Training Bytes")!

504 © Cadence Design Systems, Inc. All rights reserved.

cadence

Click here to view the demo.

Wrap Up

- Complete Post Assessment, if provided
- Complete the Course Evaluation
- Get a Certificate of Course Completion

Thank you!

cadence

This page does not contain notes.

Appendix A

Configurations

cadence

This module examines Verilog-2001 configurations. Verilog configurations provide a standard way to select design and testbench components from among multiple available representations.

Configurations Introduction

Why?

- As you develop and refine the design or testbench you accumulate multiple representations of the components:
  - Finer levels of abstraction
  - Before and after incorporating new features
  - Before and after incorporating change orders
- For performance reasons or merely for convenience you may want to “keep around” already-compiled versions of these components.

How?

- The Verilog-2001 update introduced the “library map file”
  - For compilation – Maps source locations to a symbolic library
  - For elaboration – Provides rules for resolving module instances

cadence

As you develop and refine the design or testbench, you accumulate multiple representations of the components, perhaps with more and less detail, before and after incorporating new features, or before and after incorporating change orders. For performance reasons or merely for convenience, you may want to “keep around” already-compiled versions of these components.

The Verilog-2001 update introduced the “library map file”, which for compilation, maps source locations to symbolic libraries, and for elaboration, provides rules for resolving module instances from those symbolic libraries. The standard calls this resolution process a “binding”. The mechanism for mapping symbolic libraries to file system locations is still vendor dependent. A library is a logical collection of design units and configurations. The standard refers to these design units and configurations as “cells”. A cell has no more than one representation in a given library, and you configure the design by providing rules to bind instances to different representations of the cell residing in different libraries.

The elaborator starts with one or more top-level modules not instantiated elsewhere, and for each top level module builds its hierarchy by binding instances to library cells as it works down each hierarchy.

Example Library Map File

The library map file:

- Maps each Verilog source path to a library name.
- Maps each cell instance to a library name.

include *pathname* ; – Includes another library map file

library *libname pathname* [{ , *pathname* }] – Declares a library

config *configname*; ... endconfig – States a configuration

```
include ${HOME}/libmap.txt;
library mylib myfile.v;
library techlib cell_lib/;
library worklib .../;
config top_rtl;
  design worklib.top;
  default liblist worklib;
  instance cpu1 use mylib.cpu:config;
  cell ADD use techlib.ADD;
endconfig
```

This example library map file includes a library map file from the home directory, maps three sets of Verilog source to libraries, and defines a configuration called *top_rtl* that binds the *cpu1* instance to the *mylib.cpu* configuration and instances of the *ADD* cell to the *techlib* library.

For the compiler, the library map file maps each Verilog source to a library.

For the elaborator, the library map file maps each instance to a symbolic library. The instance mapping can be by default, by cell type, by individual instance, and by user-defined configurations.

For the elaborator, you can alternatively provide on the command line as you invoke the tool an ordered list of libraries for it to search to bind instances to cells.

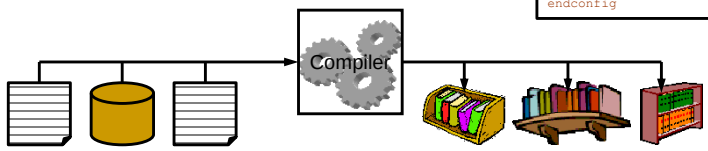
Declaring Libraries

Use the **library** keyword to declare a symbolic library and map source locations to the library:

```
library identifier pathname [ { , pathname } ]
[ -incdir pathname [ { , pathname } ] ;
```

- You can use **.**, **..**, and **...** to represent directories and directory trees.
- You can use **?** and ***** to represent any character, any string.
- File names matching multiple paths are resolved with this priority:
  1. Path names ending with an explicit filename
  2. Path names ending with a wildcarded filename
  3. Path names ending with a directory name

```
include ${HOME}/libmap.txt;
library mylib myfile.v;
library techlib cell_lib/;
library worklib .../;
config top_rtl;
  design worklib.top;
  default liblist worklib;
  instance cpu1 use mylib.cpu:config;
  cell ADD use techlib.ADD;
endconfig
```



509 © Cadence Design Systems, Inc. All rights reserved.

cadence

A library is a logical collection of cells, each associated with a specific source. Tools do not have to pre-compile these sources, but many do.

All compliant compilers provide a tool-specific way to specify at least one library map file for a compilation session. The compiler reads the library map file, and using the list of pathnames associated with each library, matches each source file or directory of source files to a library. Tools that pre-compile the sources store the libraries in the file system in a vendor-dependent manner.

The standard permits vendors to also provide a tool-specific means to map source locations to libraries and permits configurations to utilize libraries not declared in the library map file.

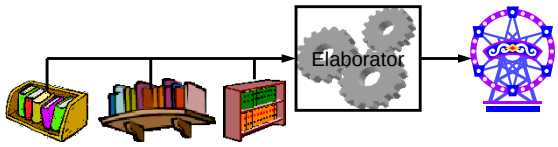
Stating Configurations

Use the **config** keyword to state a configuration:

```
config identifier ;
  design { [library.]cell } ;
  { rule }

endconfig
```

- A configuration has a name that can be the same as the design it configures.
- A configuration applies to one or more design units (usually just one).
  - You can omit the design library specification if it resides in same library as the config.
- Everything else is configuration rules (next page).




```
include ${HOME}/libmap.txt;
library mylib myfile.v;
library techlib cell_lib/;
library worklib .../;
config top_rtl;
  design worklib.top;
  default liblist worklib;
  instance cpu1 use mylib.cpu:config;
  cell ADD use techlib.ADD;
endconfig
```


cadence

The library map file can also state one or more configurations. This is not strictly necessary, as the standard permits vendors to also provide a tool-specific means to provided an ordered list of libraries for the elaborator to search to bind instances to cells.

A configuration has a name and applies to one or more designs. If you provide configurations, you must provide a configuration for each top-level design, and as a convenience you may also provide configurations for sub-designs. A typical configuration usually applies to just one design. The configuration name can be the same as the design it applies to, but this is not required. If multiple configurations have the same name then they must reside in different libraries. The remainder of the configuration states rules for binding instances to cells of the design.

You provide to the elaborator the top-level units to elaborate, either as design names or as configuration names. You also provide either one or more library map files of declared configurations, or the before-mentioned tool-specific ordered list of libraries for the elaborator to search to bind instances to cells.

Configuration Rules and Precedence

1. For one instance of a cell, a specific cell or configuration.
   ```
   instance top{.instance} use [library.]cell[:config]
   instance cpu1 use mylib.cpu:config;
   ```
2. For all instances of a cell type, a specific cell or configuration.
   ```
   cell [library.]cell use [library.]cell[:config]
   cell ADD use techlib.ADD;
   ```
3. For one instance of a cell, a library list for an ordered search.
   ```
   instance top{.instance} liblist [{library}]
   ```
4. For all instances of a cell type, a library list for an ordered search.
   ```
   cell [library.]cell liblist [{library}]
   ```
5. A default library list for an ordered search.
   ```
   default liblist [{library}]
   default liblist worklib;
   ```
6. The library declaration order.

cadence

Configuration rules are simple:

- You can specify a default ordered library list;
- For all instances of a cell type, you can specify an ordered library list or a specific cell or configuration.
- For a specific instance of a cell, you can specify an ordered library list or a specific cell or configuration.

For each instance the elaborator uses the most specific of the applicable rules to bind the appropriate representation.