A Graphical Data Management System for HDL-Based ASIC Design Projects

Claus Mayer, Hans Sahm, Jörg Pleickhardt

Lucent Technologies — Bell Labs Innovations
Thurn-und-Taxis-Str.10, D-90411 Nürnberg, Germany

Abstract

Efficient and secure project data management is a key requirement for todays HDL ASIC design projects. This paper introduces a RCS-based data management system, which focuses on the requirements of ASIC project teams. The project- and the user's working directories are strictly separated, data transfer is explicitly triggered by checking-in or checking-out files. There are cumulative check-out operations available that consider hierarchical and logical dependencies between the files and generate script files for further data processing.

The directory structures are configurable and transparent to the users, no proprietary or binary configuration files are involved.

The entire data management functionality is accessible on command level as well as from a graphical user interface, which also serves as a convenient interface to third party tools.

1 Introduction

With the growing complexity of todays ASICs and the trend towards a strictly HDL-based design methodology, the need for a project data management system that enables multiple designers to work on a common project database is apparent. The most important tasks of a ASIC project data management system are:

- store all project-related data in a common data repository with a configurable and transparent file structure
- restrict file access to authorized users
- enable multiple designers to work on the project files in a coordinated way (file locking)
- provide a history of all previous revisions of each file, including changelog information
- handle hierarchical and logical dependencies between the various files
- generate and handle script files for third party tools.

Using data management systems is common practice in the software development, and there are various toolsets available (such as RCS, SCCS, CVS [1,2]), which provide basic data management features for text files. However, if used without further measures, these tools only cover a subset of the above requirements and they are to be used on command level rather than providing a more convenient (graphical) user interface.

This led to the implementation of a RCS-based data management system that focuses on the demands of HDL ASIC design project members and project administrators.

The decision to use RCS [1] as the underlying toolset was derived due to its availability (RCS is part of the free GNU software distribution) and its compact and fast revision management, which is based on reverse delta changes.

2 Data Management Concepts

Choosing an appropriate structure of project- and user directories for storage and verification of project data is an essential prerequisite for successful project management. Furthermore, the transparency of all resulting data files and directory structures significantly contributes to the acceptance of a data management system. This also implies that no binary or undocumented configuration file formats must be introduced by the data management system. Bearing these points in mind, the following basic data management concepts and terms have been drawn:

- All project-related files are stored within a common *project directory*, which is owned by a dedicated *project user* account. The project directory is write protected on system level — only the project user may directly write into it. Read permissions are granted for all project members.
- The *project administrator* is the person responsible for the entire design project and the only one who has direct access (that is, the password) to the project user account. The project administrator also determines the project configuration by means of three special configuration files (which are explained below).
- each project members' working directory (also called *local directory*) is independent from the project directory — there are no symbolic links between both of them.

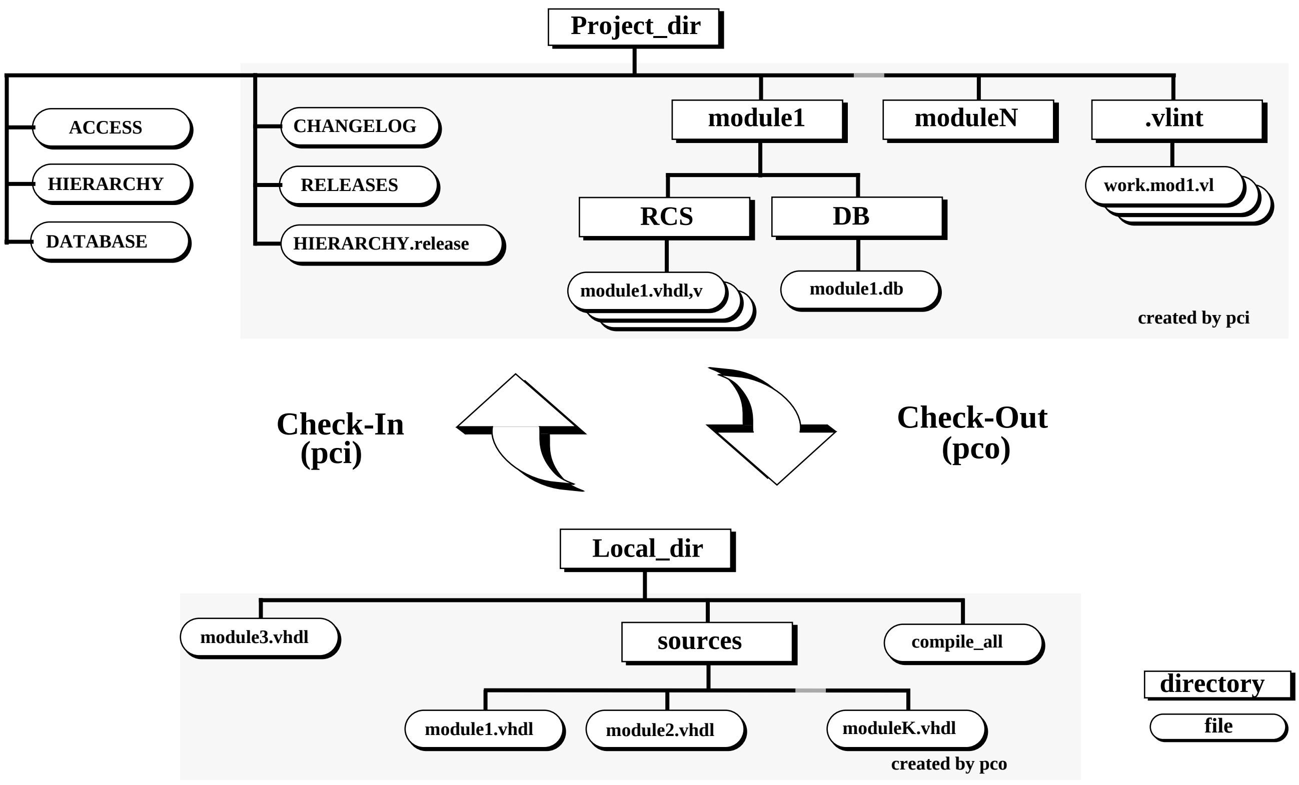


Figure 1: project- and local directory example

- The data management functions treat a design as a collection of *modules*, each of them representing a single design entity (or hierarchical block). For each module, the database can hold a set of *file types* (e.g., a VHDL source file, a synthesized schematic, etc. [3,4]).
- a project directory may refer to other project directories in order to import specific modules or module hierarchies from there. The reference may be bound to a specific release of the reference project — if it is not, any changes to the reference projects' data- and administrative files immediately propagate to the main project (including changes to the design hierarchy). All modules are imported 'read-only': the main project's members cannot directly alter the files in a reference project. However, imported files may be overridden by corresponding files in the main project.
  Typically, reference projects are used to provide libraries of versalite design units that can be referenced from within other ASIC projects later on.
- there are two classes of file types: the *text files* are kept under RCS control, which implies that they support all RCS-related features like file locking, revision/release control and changelog information. In contrast to that, the *binary files* are just copied into the project directory.
- data transfer between project- and local directories is handled by means of explicit *check in* and *check out* commands. The data management functions directly operate on files, which are identified by their name and file extension: the leading name refers to the module whereas the extension determines the corresponding file type. Note that this implies a mapping between design modules and related data file names [4].
- Only authorized users may check in or otherwise modify a file in the project directory. The check-in permissions are granted on a per-module basis. A global CHANGELOG file keeps track of all modifications that have been made to the project directory so far.
- A notification mail system ensures that the project members are kept informed about changes to all modules they are interested in.

Until now, we focused on concepts that are related to the project directory. For the local directory (that is, the project member's working directory), the following points are of importance:

- single files may be checked out either for editing (which locks the file) or for read-only access.
- all files under edit are located directly within the local directory

- files that are checked out for read-only access are placed in their *checkout directory* rather than in the current directory, which helps to prevent the working directory from being trashed with lots of files. The checkout directories are defined by the project administrator - they determine file structure within every project member's working directory.
- read-only check-outs can be executed on single files as well as on complete hierarchical subtrees of the design, which causes all affected files to be checked out. The hierarchical dependencies are defined by the project administrator.
- the check-out operations also consider dependencies between different file types. For instance, a dependency 'file type .o depends on .c' would cause a check out on xxx.o to actually fetch xxx.c if it has changed more recently.
- incremental check-out operations cause only the outdated files to be checked out.
- As a side effect, the check-out procedure generates script files that are required to further process the files being checked out. The contents of the script files is freely configurable.

The check-out options mentioned above can be combined — thus, an incremental, hierarchical checkout would only affect the files below the specified top-level that have been modified since the last check-out operation.

3 Configuration Files

The directory- and file arrangements in the project- and local directories are freely configurable by the project administrator. Figure 1 gives an example of the directory structures for VHDL projects chosen at our site.

The various configuration items are specified by means of three ASCII files with predefined names that are located directly within the project directory. All configuration files have similar, format-free syntax conventions and allow C-style commenting. Each of the files deals with different configuration aspects, which are explained in the next sections.

3.1 The DATABASE File

The *DATABASE* file defines the set of binary and text file types (including their file extensions) that may be used within the project and the way to handle them. It determines the directory structure of the project- and local directories as well as all dependencies between various file types. In addition to that, it specifies the set of script files that are written upon check-out operations and describes their contents.

In order to provide more flexibility, the DATABASE file may contain references to *variables*, which expand to predefined textual values.

Besides a set of built-in variables, the project administrator may define an arbitrary set of global and module-specific variables in the ACCESS file.

3.2 The ACCESS File

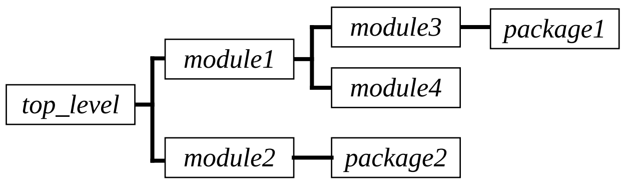
The *ACCESS* file defines all administrative parameters of a project (e.g., project name, project administrator name, global variables, etc.) and defines the set of modules the project consists of. For each module, a set of

- valid file extensions (which refer to the extensions defined in the DATABASE file)
- project members with write (check-in) permission
- project members to be notified about changes
- user defined variables to be referenced in the DATABASE file

can be specified.

3.3 The HIERARCHY File

The *HIERARCHY* file represents the inter-module dependencies and levels of hierarchy. A project tree like:



would result in the following HIERARCHY file:

```
top_level := module1, module2;
module2 := package2;
module1 := module3, module4;
module3 := package1;
```

Any hierarchical check-out operations of complete design subtrees are based on the information stored within the HIERARCHY file. As we will see later on, the graphical user interface displays the hierarchical dependencies as hierarchy tree.

The hierarchical dependencies may be grouped within *Hierarchy Sets*, which are selectively enabled or disabled depending on hierarchy configurations and the file types involved. This allows for multiple sets of hierarchy trees within the very same project.

Finally, the HIERARCHY file optionally specifies the referenced project directories and defines which module hierarchies are to be imported from which project directory (and release).

4 Implementation Concepts

A very important factor regarding the usability and thus the acceptance of a data management system is the way its functionality is presented to the project members. In order to satisfy the demands of both novice and experienced users, both basic user interface types are supported: the entire data management functionality is available on system (command-) level as well as on GUI level. Both ways of launching a command are functional equivalent and may be combined or alternated at any time — in fact, the graphical user interface employs the command level tools to perform its tasks, as indicated by the software hierarchy shown in Figure 2.

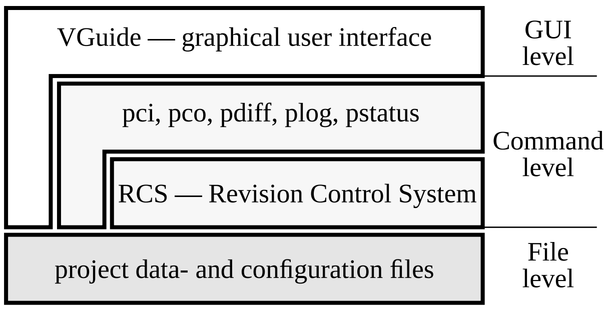

VGuide — graphical user interface
GUI level
pci, pco, pdiff, plog, pstatus
Command level
RCS — Revision Control System
project data- and configuration files
File level

Figure 2: the software hierarchy

5 The Command-Level Tools

On Command level, the following stand-alone-commands [5] are available:

- *pci*
  is the only tool that allows authorized users to modify the contents of the project directory. The project members use it to check in new files or to delete existing files. Furthermore, pci can alter the locking state and the description of a file.
  The project administrator uses pci to create or change symbolic releases on certain revisions of the files within the project directory.
  Any changes pci applies to the project directory are protocolled in the CHANGELOG and RELEASES files.

- *pco*
  retrieves data from the project directory by checking-out a single file or all files below a hierarchical branch of the design. Pco also considers dependencies between the various file types and generates script files for third party tools.
- *pdiff*
  compares one or more local files against their database pendants and lists the differences.
- *pstatus*
  displays the contents of the project directory and optionally provides detailed information on single modules.
- *plog*
  displays all RCS-related information that is available for a specific text file (changelog, revision numbers, symbolic release names).

All remaining administrative tasks can be handled by directly editing the respective configuration file in the project directory. However, the graphical user interface provides a much more convenient interface to these files.

6 The Graphical User Interface

The main objective of the graphical user interface *VGuide* [5] is to enhance the usability of the project data management tools by

- visualizing the current state and the dependencies between the project-related data files, including the option to focus on well-defined subsets thereof
- providing a convenient interface to the functionality of the stand-alone commands
- providing a convenient way to configure the project without the necessity to edit configuration files by hand.

The window structure of VGuide (see figure 3) closely corresponds to the project- and local directories explained in the previous sections.

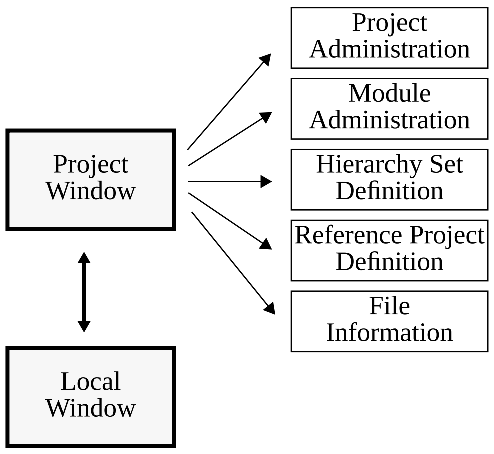


Figure 3: VGuide window structure

All windows are non-modal, which implies that they can be opened or closed at any time and in any combination if at least either the project- or the local window stays open. Selections (e.g., the current file and current top-level module) are propagated through the entire application, so all windows focus on the same component.

6.1 The Project Window

The project window represents the project directory and displays all related information. The display area within the project window offers two modes: the hierarchy tree visualizes the hierarchical dependencies (figure 4) whereas the file table focuses on the state and availability of the various files (figure 5).

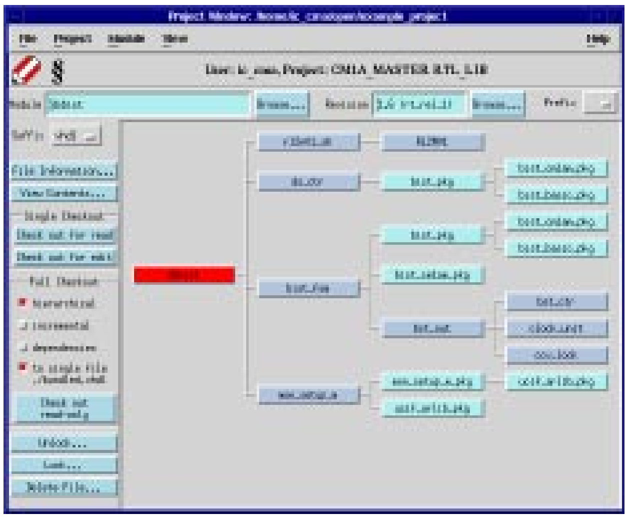
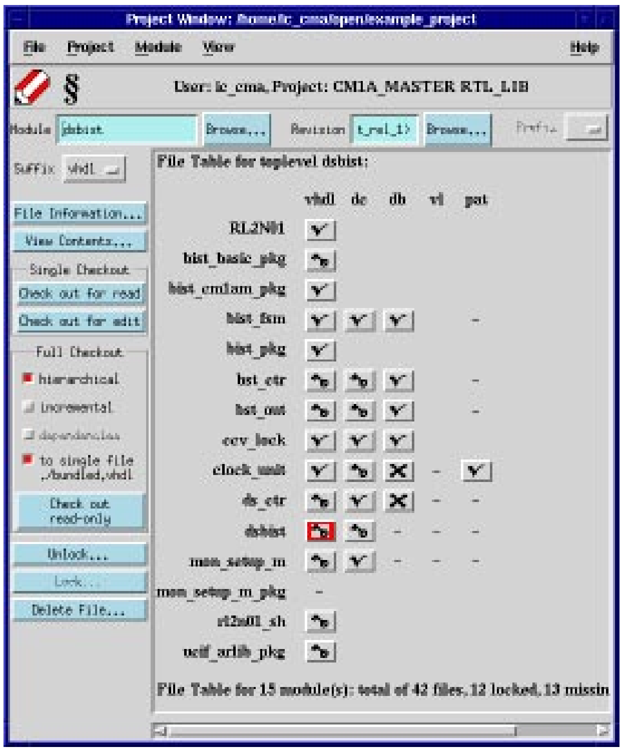


Figure 4: project window displaying a hierarchy tree



File Table for toplevel dsbist:

	vhdl	dc	db	vl	pat
RL2N01	✓				
bist_basic_pkg	↳				
bist_cm1am_pkg	✓				
bist_fsm	✓	✓	✓		-
bist_pkg	✓				
bst_ctr	↳	↳	✓		-
bst_out	↳	↳	✓		-
ccv_lock	✓	✓	✓		
clock_unit	✓	↳	✗	-	✓
ds_ctr	↳	✓	✗	-	-
dsbist	🟥	↳	-	-	-
mon_setup_m	↳	✓	-	-	-
mon_setup_m_pkg	-				
rl2n01_sh	↳				
ucif_arlib_pkg	↳				

File Table for 15 module(s): total of 42 files, 12 locked, 13 missin

Figure 5: project window displaying a file table

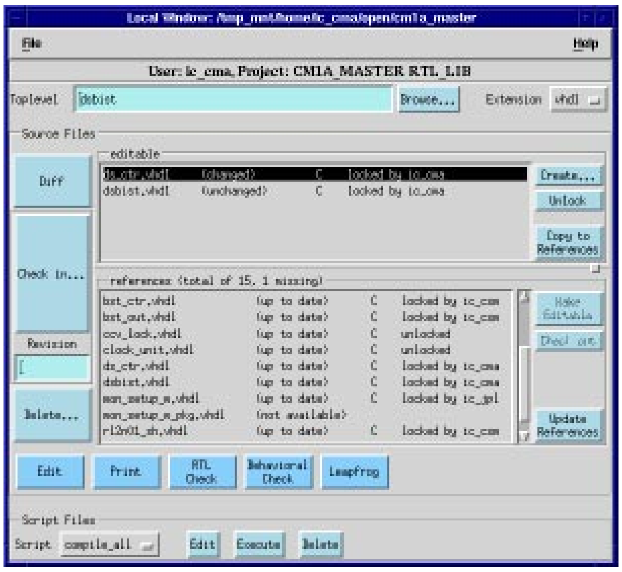
The project window provides access to sub-windows, which allow to

- view and — if authorized — edit the project- and module-specific configuration items. Modifying these items results in new versions of the HIERARCHY and/or ACCESS files, which are created by VGuide.
- view the detailed state and location of the current file, including the RCS-specific settings.

Furthermore, the project window provides direct access to all administrative files in the project directory and thus fully covers all project-administrative tasks.

6.2 The Local Window

The Local Window (see figure 6) represents the local working directory of the project members and serves as a convenient frontend to all tools that are frequently executed during the design process. This applies to the VGuide-specific data management functions as well as to all other third party tools like editors, simulators, checkers, etc. that operate on the various source files.



```
ds_ctr.vhdl      (changed)      C    locked by ic_cma
dsbist.vhdl      (unchanged)    C    locked by ic_cma
```

references (total of 15, 1 missing)

```
bst_ctr.vhdl          (up to date)     C    locked by ic_cm
bst_out.vhdl          (up to date)     C    locked by ic_cm
ccv_lock.vhdl         (up to date)     C    unlocked
clock_unit.vhdl       (up to date)     C    unlocked
ds_ctr.vhdl           (up to date)     C    locked by ic_cma
dsbist.vhdl           (up to date)     C    locked by ic_cma
mon_setup_m.vhdl      (up to date)     C    locked by ic_jpl
mon_setup_m_pkg.vhdl  (not available)
rl2n01_sh.vhdl        (up to date)     C    locked by ic_cm
```

Figure 6: the local window

The two lists in the local window represent the set of working files that are instantiated below the current top-level module. They display the set of editable files and the pool of referenced files, respectively. Each entry in the file lists provides information on the state of the file (locking state, availability, etc.).

A programmable button panel serves as interface to external tools.

6.3 Update Strategy

As mentioned above, VGuide may be used in combination with the stand-alone p-tools — even more, the project configuration files may be modified on system level (e.g., the ACCESS file may be edited by the project administrator). This implies that VGuide cannot rely on any (redundant) configuration files of its own but has to scan all the project-related files in both the project- and the local directory in order to build up its display. Furthermore, VGuide should automatically recognize changes to the project files and refresh its display accordingly.

In order to accomplish this task without consuming too much CPU time by instantly re-reading all the project files (polling), a trade-off between CPU usage and the coverage of the detection of external changes had to be made.

In the project directory, only the administrative files are checked for modifications, which ensures that all pci-related modifications to the project directory are caught as well (since they affect the CHANGELOG file).

The very same applies to the configuration files of the reference projects (if any).

7 Conclusion and future work

In this paper, we have introduced a data management system that satisfies the typical requirements of HDL ASIC designers and project administrators. Its key features are data security, transparency and configurability. All functions are available on command level as well as on GUI level.

At our site, the data management system is combined with a tool environment that further supports VHDL designers in writing and debugging their code [6, 7]. The close relationship to the daily design work during the development phase has led to a very stable and reliable set of tools, which is in practical use for more than two years.

Depending on the demands of the project teams, the data management system is constantly extended and adapted to the practical requirements.

8 References

[1] Tichy, F.W.: RCS - A system for version control, Software Practice & Experience, Vol 15, 1985, pp 637 - 654

[2] The SCCS Manual Pages, available on all common UNIX platforms

[3] Sahm, H., Mayer, C., Pleickhardt, J. and Späth, S.: VHDL Coding Standard, Rev. A-0-9, 1996, Philips internal document

[4] Sahm, H., Mayer, C., Pleickhardt, J. and Späth, S.: OMI-326 Draft Standard for open review, 1995, OMIMO, Brussels
(online version: http://www.omimo.be)

[5] Mayer, C.: VHDL Development System Documentation, 1995, Philips internal document

[6] Hack, W. and Mayer, C.: Supporting Tools for a VHDL Coding Standard, 1994, VHDL Forum for CAD in Europe,Tremezzo, Proceedings pp 117 - 121

[7] Sahm, H., Mayer, C., Pleickhardt, J., Schuck, J. and Späth, S.: VHDL Development System and Coding Standard, 1996, 33rd Design Automation Conference (DAC) Proceedings, pp. 777 - 782