VHDL Development System and Coding Standard

Hans Sahm *, Claus Mayer *, Jörg Pleickhardt *, Johannes Schuck *, Stefan Späth **

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

Abstract

With the growing complexity of todays ASICs and the number of designers involved in one VHDL ASIC project, the need for a VHDL development system together with coding rules for simulation and synthesis has emerged.
This paper describes the VHDL Coding Standard which has been established and the VHDL development system including code entry, code formatting, code compliance checkers, data management and multi-user project set-up.

1. Introduction

Designing ASICs for telecommunication applications like SDH (“Synchronous Digital Hierarchy”), ATM (“Asynchronous Transfer Mode”), Subscriber access and GSM (“Global System for Mobile communication”) results in design complexities of half a million logic gates and code development of more than a hundred thousand lines of code for VHDL testbenches, behavioral level modeling for simulation and RT-level for synthesis.
The project teams, creating those complex designs, need definitely assistance in a common set of rules to apply and in tool support apart from a VHDL simulator and synthesis tool.
While the expressive power, flexibility, technology and process independence and the ability of coding on different abstraction levels [1],[4] significantly contributes to the success of VHDL, these qualities may cause on the other hand severe problems, when experience and project coordination is insufficient. Potential problems are exchangeability, portability, reusability, understanding of all available language constructs and unexpected behavior in simulation and in synthesis results.
Given this situation, we decided to establish a VHDL Coding Standard, which is the reference for VHDL modeling in our company and is intended to be as independent of any particular EDA tool as possible. Basically the Coding Standard is organized as a collection of rules and recommendations, which are used as reference for the VHDL development system on source code formatting and compliance checking for simulation and synthesis models.
In order to cope with the huge amount of data created in the course of a VHDL project, methods and techniques known from software development [2],[3] have been adapted and enhanced with the special features of VHDL on dependencies and primary design units for overall data management and multi-user project set-up.

2. Overview

Figure 1 shows an overview of the structure of the VHDL development system:
VGuide and the embedded *XEmacs* language sensitive editor represent the graphical user interface.
Pci, pco, pdiff, plog and *pstatus* are revision control tools for administration and project data management for multi-user designs.
Vfmt, vlint, vpr, and *vlog* are VHDL source code tools for code formatting, code compliance checks, pretty printing and statistics.

The underlying reference for all tools is the VHDL coding standard.

The concepts, major tools and VHDL coding standard will be presented in the following sections.

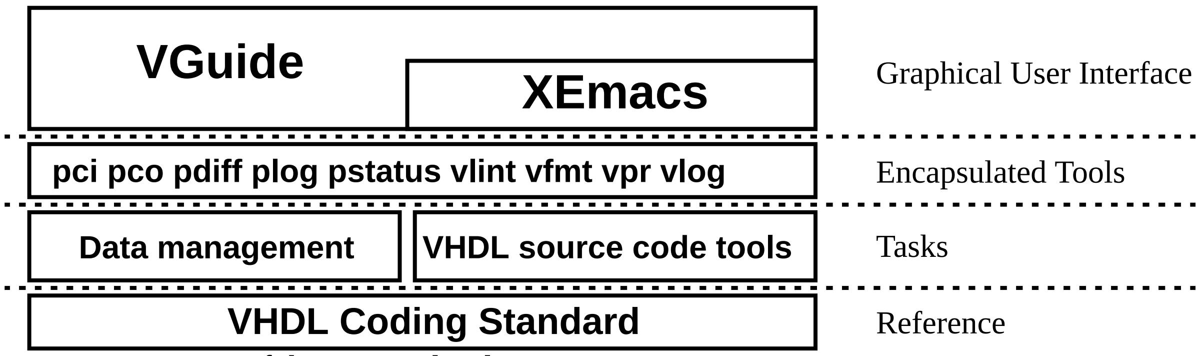


Figure 1: Overview of the VHDL development system

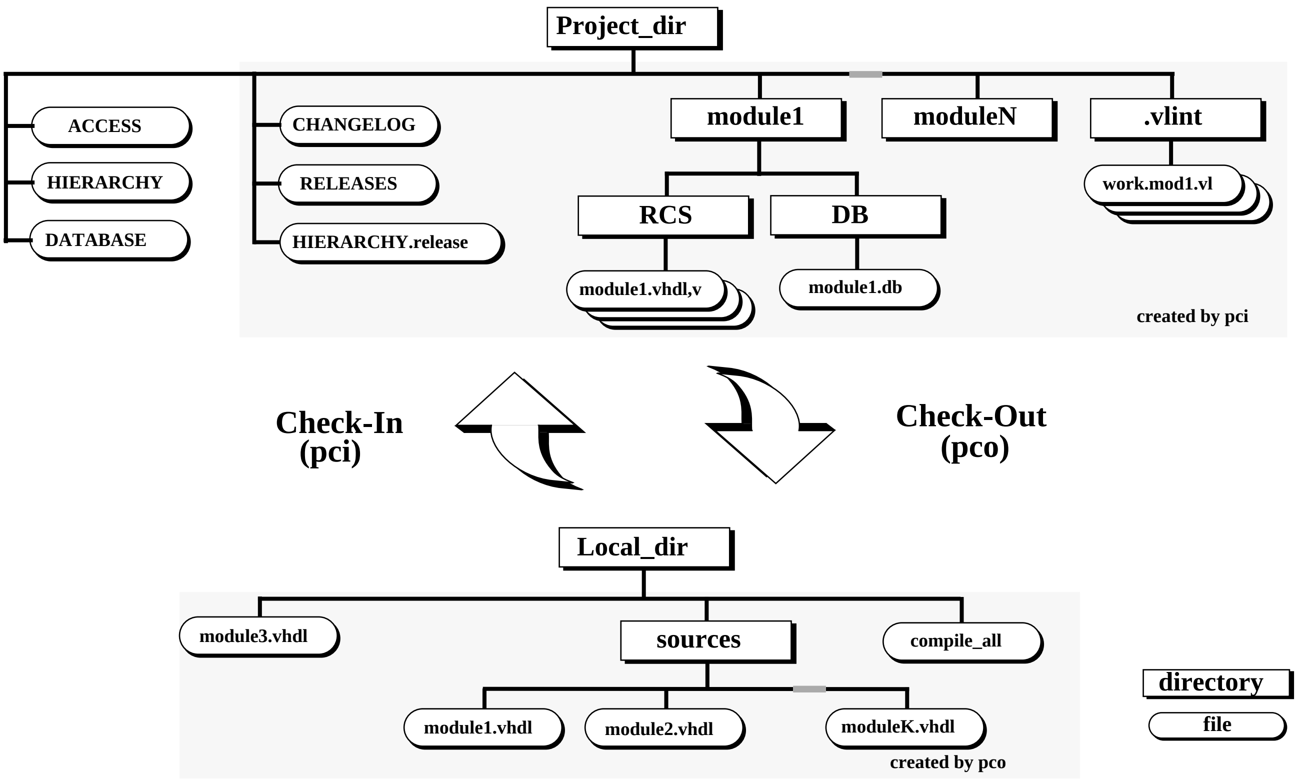


Figure 2: Set-up of project and user directories and data

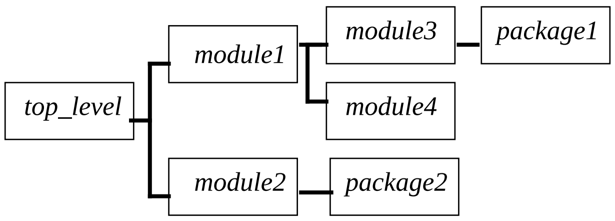
3. Project set-up

Choosing an appropriate structure of project and user directories for storage and verification of project data is an essential prerequisite for successful project management.
The chosen approach is file based data management with strict decoupling of project and working directories, the project directory structure being transparent to users and freely configurable.
Revision control is implemented based on RCS("Revision Control System") [3], since RCS offers several advantages like:

- lock mechanism to prevent interfering edits
- file storage with reverse deltas (fast access)
- availability of symbolic release pointers
- complete history of revisions including log messages.

Figure 2 outlines an example of directory structures for VHDL projects chosen at our site.
The reference versions of all source files of a project are stored in the project directory, which is maintained by a project administrator, who has write permissions to the directory. All other project members have only read permissions to files within the project tree.
Below the project directory a subdirectory exists for each module. All files related to the respective modules are stored there. Files like VHDL sources and scripts, which are under revision control are kept in a directory RCS according to RCS conventions[3]. Presynthesized modules, if existing, are stored within a directory DB. Separate project trees are created for the behavioral level and RT-level part of a project.
Three administration files have been defined:

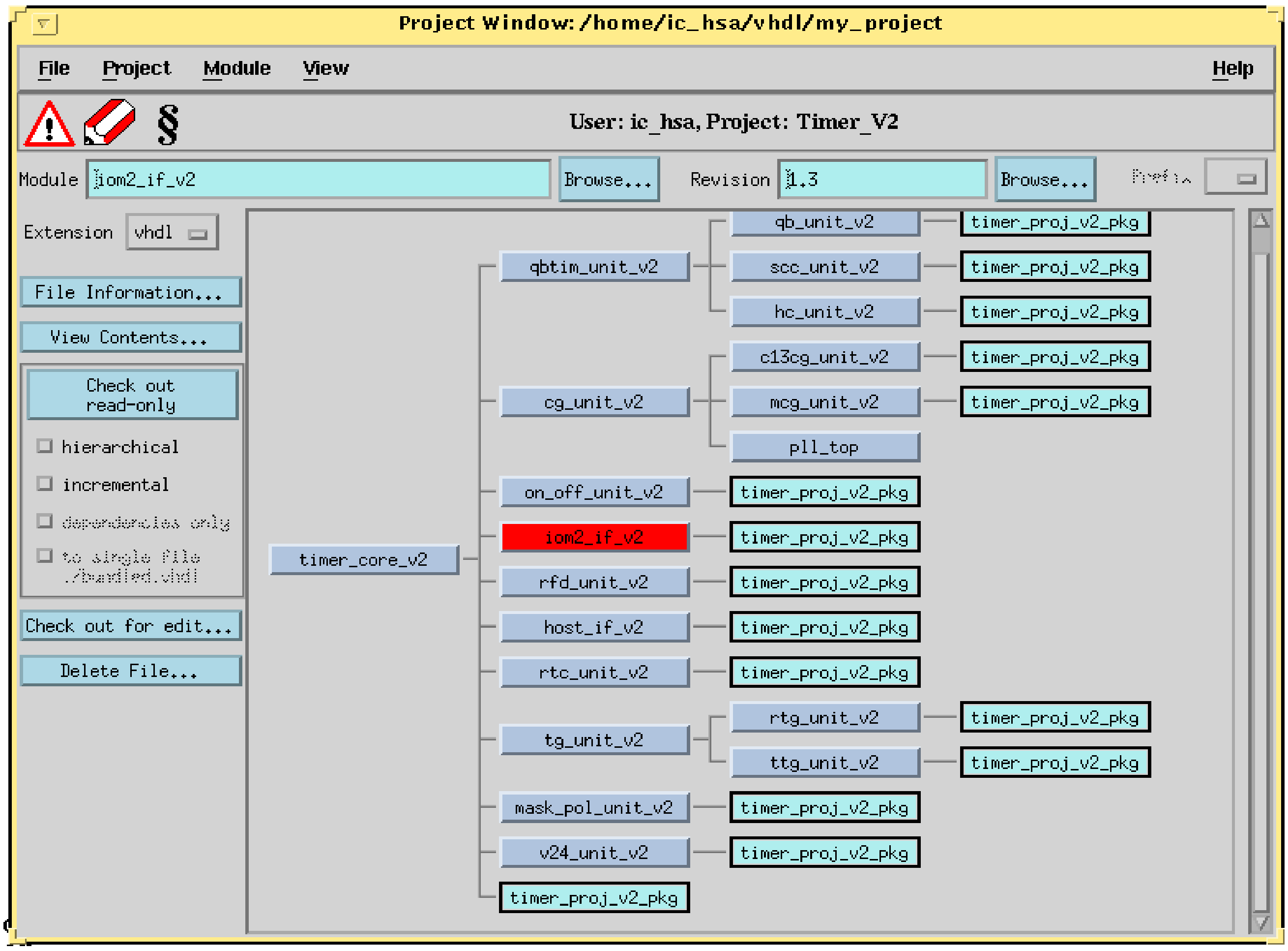
The *HIERARCHY* file represents the VHDL dependencies and levels of hierarchy. A project tree like:



would result in a *HIERARCHY* file:

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

This *HIERARCHY* file is either automatically generated by the VHDL development system [7] or could be edited by any text editor. Based on this *HIERARCHY* file, compile scripts are generated, the design hierarchy is displayed graphically within the VHDL development system and check-out operations are managed for any level of hierarchy.
The second administration file is called *ACCESS* and defines


Project Window: /home/ic_hsa/vhdl/my_project

User: ic_hsa, Project: Timer_V2

Module iom2_if_v2

Revision 1.3

Extension vhdl

Figure 3: Project window of VHDL development system

the list of authorized check-in users or groups per module and the list of users or groups, which are notified when modified modules have been checked-in.

The third one is the *DATABASE* file, which may be used to define any user defined directory structure, any file type to be handled, any particular dependency to be managed and name and contents of script files to be generated together with check-outs.

4. Data management tools

The basic task of the data management tools is to allow the project members to work on a common project directory in a coordinated way, without the need for anybody involved in the project to manually copy, move or delete any file on operation system level. Within the VHDL development system, commands based on check-in and check-out from RCS are used and a graphical user interface is provided with a project window representing the data management for project data(see figure 3) and a local window managing data within users local directories(see figure 4). The general concept is to have all functionality available from the graphical front-end (project and local window) as well as on command level of the operating system. Both ways of launching a command are functionally equivalent and may be combined or altered at any time.

The project window assists the designer in getting an overview of the project hierarchy, of the status of all files, releases and RCS changelog information. It allows the creation and update of all project administration tasks like defining and changing access rights per module for users and groups and notification lists per module for users. In addition it is the frontend for check-out operations of all modules either for read-only or editing purposes including file locking at ‘check-out for edit’ and unlocking at check-in time.

The local window of the VHDL development system is intended for all tasks of creating, verifying, beautifying, and finally checking-in the local modules of all designers involved in the project. It is the frontend for a number of VHDL source code tools:

- *XEmacs* - language sensitive editor with VHDL mode
- *vfmt* - source code beautifier
- *vlint* - VHDL code compliance checker
- *vpr* - VHDL pretty printer

In addition the local window delivers all status information and any available script file can be executed. The commands of the local window are user configurable and expandable.

5. VHDL Coding Standard

VHDL as a hardware description language offers almost complete freedom to the designer in how he may model his design. Experience shows different designers have preferences for different modeling styles, data types, packages, commenting, naming styles and design partitioning.

The intention of the VHDL coding standard [5],[6] is to provide guidelines how VHDL models should be coded to ensure portability of designs between different groups of designers and/or different EDA platforms. Care has been taken to keep this document as independent as possible of any particular VHDL implementation or EDA vendor. However, as indicated in the RTL language section, because of the different capabilities of the different synthesis tools, this section can not be strictly tool independent.

The VHDL coding standard is written in the form of rules and recommendations, each one identified by a symbolic name, which allows for automatic rule compliance checks of VHDL sources and trace back of the violations to the corresponding rule[8]. See figure 5 for an automatic compliance check example.

The VHDL coding standard is divided into the following main parts:

- General principles

This section covers the basic ideas behind the coding standard including file management, revision control and mapping between design units and files.

- Code layout

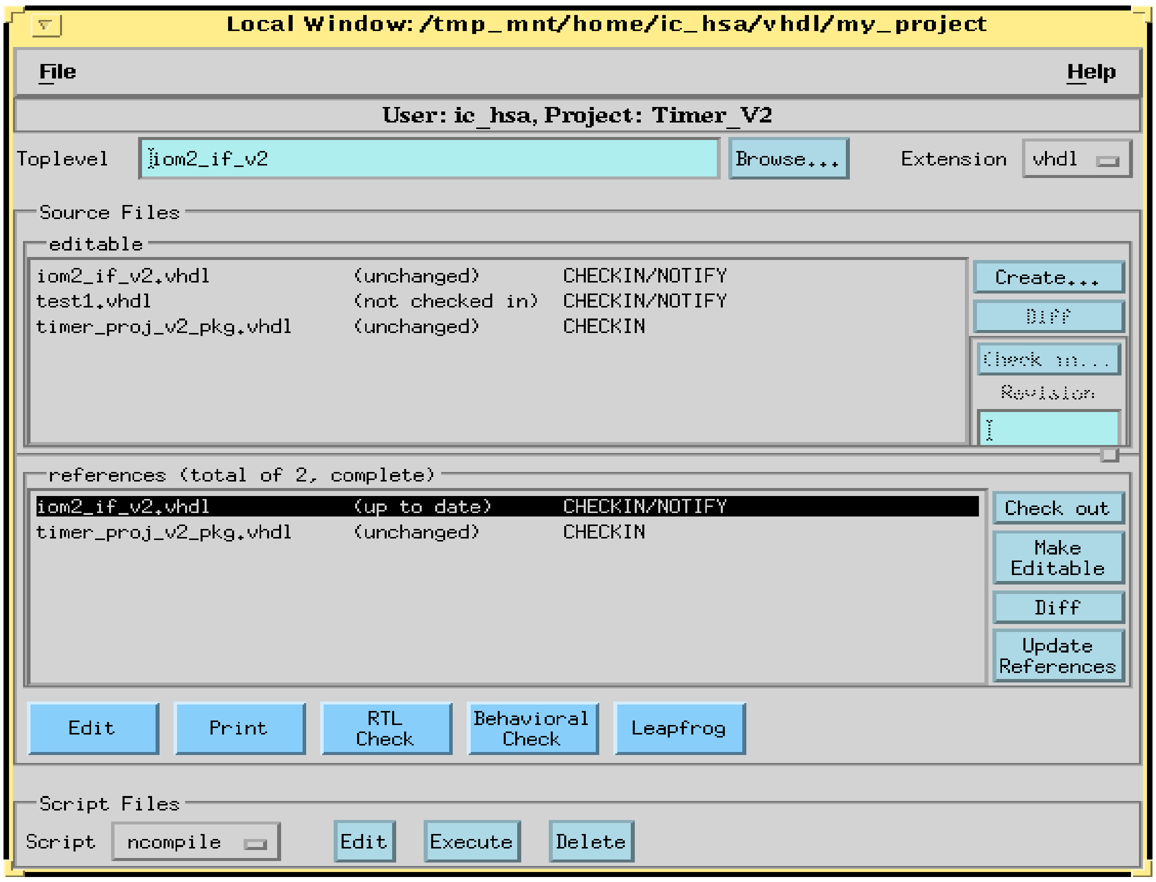
All formal aspects of VHDL coding are treated in this section. Rules and recommendations from traditional programming languages [2] have been adapted. Among the topics covered therein are: naming conventions, naming strategies, readability aspects, commenting requirements, commenting styles, indentation, alignment and spacing.
A large part of this section consists of predefined templates for all major VHDL constructs, which will be included by the language sensitive editor on keyword expansion [7].

- Language usage

This is the most comprehensive section and deals with all items concerning VHDL language usage. Subsections exist for abstract behavioral modeling, for register transfer level modeling and a general usage subsection, which is applicable for both kind of models. The behavioral modeling part deals with language rules for behavioral simulation, testbench models and board simulation models. Main topics are the definition of data types, process partitioning, performance modelling and model portability. The RTL modeling part deals with design decomposition, data types, process partitioning and non synthesizable constructs.
It was written with Synopsys synthesis in use, but having other sites of our company using synthesis tools of two different EDA companies.

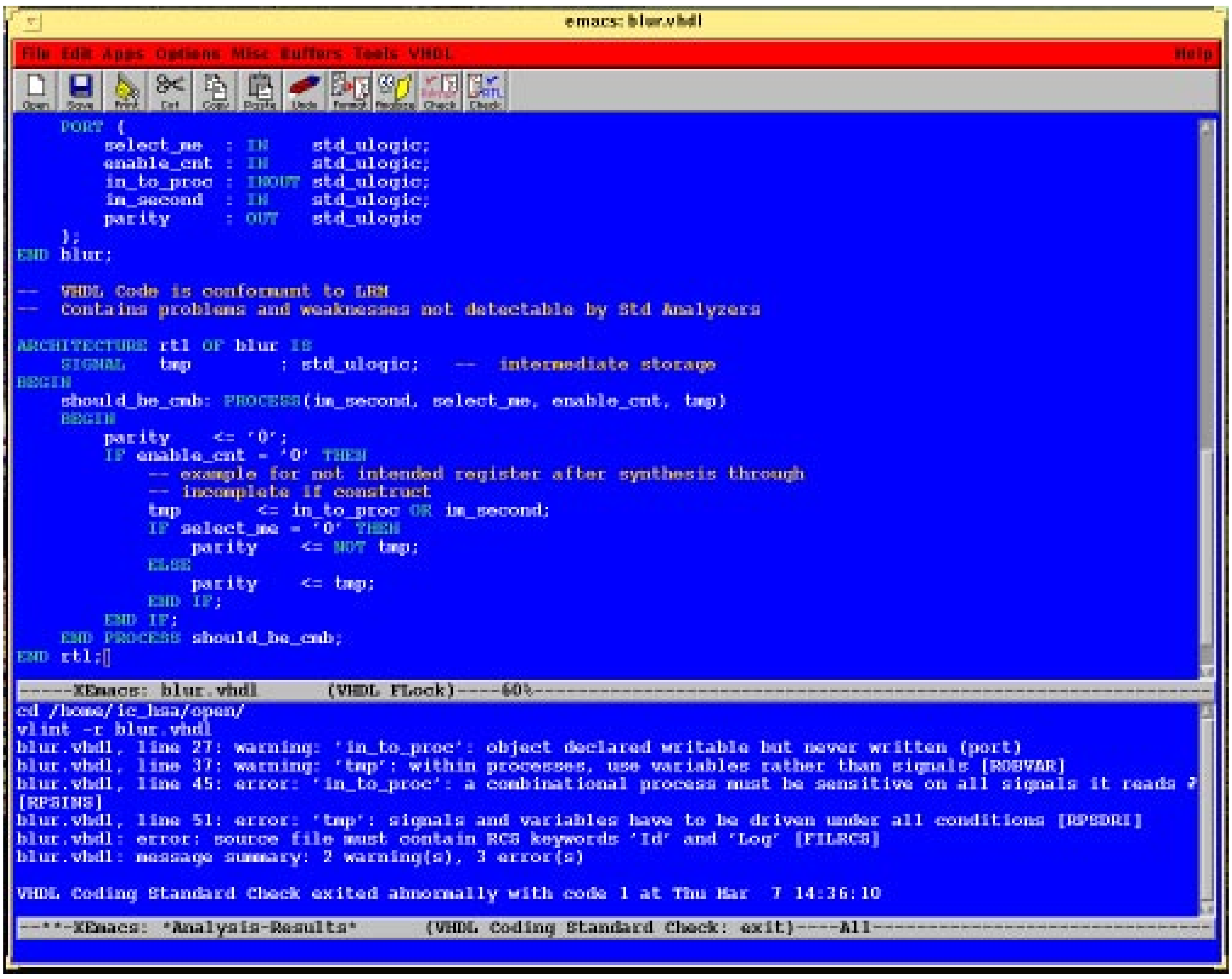
- Reference section

This final part contains definition and type declarations of standard and additional packages, which are part of the VHDL development system. For the definition of logic values the IEEE *std_logic_1164* package is used. String and conversion packages are proprieatary packages. For arithmetic functions an industrial package [9] has been adapted according to our standard. When balloting on IEEE numeric packages is concluded, they will replace the proprieatary arithmetic package.



```
Local Window: /tmp_mnt/home/ic_hsa/vhdl/my_project

User: ic_hsa, Project: Timer_V2

Toplevel  iom2_if_v2                              Extension  vhdl

Source Files
 editable
  iom2_if_v2.vhdl            (unchanged)       CHECKIN/NOTIFY
  test1.vhdl                 (not checked in)  CHECKIN/NOTIFY
  timer_proj_v2_pkg.vhdl     (unchanged)       CHECKIN

 references (total of 2, complete)
  iom2_if_v2.vhdl            (up to date)      CHECKIN/NOTIFY
  timer_proj_v2_pkg.vhdl     (unchanged)       CHECKIN

Script Files
 Script  ncompile
```

Figure 4: Local window of VHDL development system



```
    PORT (
        select_me   : IN    std_ulogic;
        enable_cnt  : IN    std_ulogic;
        in_to_proc  : INOUT std_ulogic;
        im_second   : IN    std_ulogic;
        parity      : OUT   std_ulogic
    );
END blur;

--  VHDL Code is conformant to LRM
--  Contains problems and weaknesses not detectable by Std Analyzers

ARCHITECTURE rtl OF blur IS
    SIGNAL   tmp        : std_ulogic;   --  intermediate storage
BEGIN
    should_be_cmb: PROCESS(im_second, select_me, enable_cnt, tmp)
    BEGIN
        parity    <= '0';
        IF enable_cnt = '0' THEN
            -- example for not intended register after synthesis through
            -- incomplete if construct
            tmp       <= in_to_proc OR im_second;
            IF select_me = '0' THEN
                parity    <= NOT tmp;
            ELSE
                parity    <= tmp;
            END IF;
        END IF;
    END PROCESS should_be_cmb;
END rtl;
-----XEmacs: blur.vhdl      (VHDL FLock)----60%----------------------------------
cd /home/ic_hsa/open/
vlint -r blur.vhdl
blur.vhdl, line 27: warning: 'in_to_proc': object declared writable but never written (port)
blur.vhdl, line 37: warning: 'tmp': within processes, use variables rather than signals [ROBVAR]
blur.vhdl, line 45: error: 'in_to_proc': a combinational process must be sensitive on all signals it reads ?
[RPSINS]
blur.vhdl, line 51: error: 'tmp': signals and variables have to be driven under all conditions [RPSDRI]
blur.vhdl: error: source file must contain RCS keywords 'Id' and 'Log' [FILRCS]
blur.vhdl: message summary: 2 warning(s), 3 error(s)

VHDL Coding Standard Check exited abnormally with code 1 at Thu Mar  7 14:36:10
--**-XEmacs: *Analysis-Results*      (VHDL Coding Standard Check: exit)----All---------
```

Figure 5: *Vlint* checks within *XEmacs* editor

The VHDL coding standard has been in use within our company for 3 years. It has been continuously improved through valuable feedback from current VHDL projects and is controlled by a VHDL Steering Group updating and distributing improved versions on a regular basis.
In addition the standard is currently in open review within the european community ESPRIT project OMI (Open microprocessor systems initiative) [6].

6. Vlint compliance checker

However, a stand-alone VHDL coding standard with hundreds of rules and recommendations is of limited use, if no supporting tools are available to check automatically the VHDL sources of a project against those rules.This is the main task of *vlint*.

Vlint includes a full VHDL scanner/parser and therefore can be used to check VHDL sources for syntactical and semantic errors. The significant difference compared with VHDL compilers for simulation and/or synthesis is the required analyze time and the improved error reporting. The main subject of *vlint* is code compliance checking against the rules of the VHDL coding standard. Figure 5 shows an extract of a VHDL architecture and the resulting vlint messages.
One mode is rule checking for synthesis, which refers to RTL rules and general rules, the second mode is rule checking for simulation referring to behavioral modeling and general rules. Violated hard rules of the coding standard cause error messages issued by *vlint*, ignored recommendations lead to warnings from *vlint*.
In addition *vlint* checks a number of items, which are potential sources of errors, like:

- declared but unused objects or ports
- NULL ranges
- consistency of sensitivity lists and signals read in a process
- dead code
- whether declaration and usage of port modes are consistent (eg INOUT, only used as IN or OUT).

All *vlint* output messages include the mnemonic rule identifier and a verbose problem description to reduce the need to read the corresponding coding standard section.
It is obvious that a VHDL checker tool cannot analyze a source file without the knowledge of its design context, because every non-trivial model contains references to objects declared externally in different source files or packages.
Since it would be very inefficient to locate and reparse every ref-

erenced VHDL file and package, *vlint* uses a mechanism to store all vlint related data permanently within binary vlint libraries[8].

7. VHDL beautifier

The need for a VHDL beautifier either in conjunction with a language sensitive editor or as a stand-alone formatter was identified by VHDL designers.
The result is *vfmt*, a source code beautifier.

Figure 5 shows an extract of *vfmt* formatted code within *Xemacs* editor window (upper window).

On one hand it is very easy to use since a default configuration file is used to generate code fully compliant with the code layout section of the coding standard. On the other hand the configuration file gives so much flexibility, that it may be adapted easily to any user requirements.
A few examples on the configuration abilities are:

- character case conversion

  KeywordCase = UpperCase;
  IdentifierCase = AsIs;
  AttributeCase = Capitalized;

would tell *vfmt* to convert all VHDL keywords to upper case and leave all user defined identifiers untouched and change the predefined attribute names to capitalized.

- code layout changes

  Linewidth = 80;
  Indentation = 4;

Vfmt handles proper indentation and wraps long lines exceeding a maximum line length. Wherever possible, structures are vertically aligned for better readability.

- code modifications

  Modifications are safe, since only additions for automatic repeating of entity names or process labels at the corresponding end statement are done, if they are missing.

- comment handling

  Block CommentCharacters = "-=";*
  CommentStartColumn = 41;
  ExtendComments = True;

Comments are indented to the same level as the corresponding code, trailing comments are vertically aligned.

- header file inclusion

  ProcessHeaderFilename = "/path_to_my_headers"

  Vfmt includes the predefined headers for documentation purposes and automatically expands variables to those headers like filename, date, time, name of unit or process, organization, author, simulator version, synthesis version etc.

- piece-wise formatting

  --vfmt off
  --vfmt on

  Current work in progress on *vfmt* includes the feature of formatting selected ranges of the VHDL source code using meta-comments.
Applications are sources which contain manual pre-formatted sections or tables which should remain as they are.

8. Conclusions

The previous sections have outlined the methods we established for VHDL designs in terms of coding rules and the supporting tool environment.
By analyzing the statistical data on ASIC designs over the last few years, the impact of VHDL top down design on design productivity, silicon first time right and design quality is significant. For SDH ASICs the rate of first time right designs has increased to 80%, which is significantly higher compared with designs where previous design methods have been applied and compared with industrial average.
The VHDL development system is an environment which gives useful tool support to designers in their daily tasks of VHDL source code entry, verification, data management and project setup.
The vision of the authors is to improve this VHDL development system by further requested functionality on design reuse, verification and system functionalities.

9. References

[1] ANSI/IEEE Standard 1076-1993 VHDL Language Reference Manual, New York, 1994, IEEE Press
[2] Straker,D.: C-Style: Standards & Guidelines, Englewood Cliffs, 1992, Prentice Hall
[3] Tichy,F.W. RCS - A system for version control, Software Practice & Experience, Vol 15, 1985, pp 637 - 654
[4] Lipsett,R., Schaefer,C. and Ussery,C.: VHDL: Hardware Description and Design, Boston, 1989, Kluwer Academic Press
[5] Sahm, H., Mayer,C., Pleickhardt,J. and Späth,S.: VHDL Coding Standard, Rev. A-0-9, 1996, Lucent Technologies document
[6] Sahm, H., Mayer,C., Pleickhardt,J. and Späth,S.: OMI-326 VHDL Draft Standard for open review, 1996, OMIMO, Brussels http://www.omimo.be/standard
[7] Mayer, C.: VHDL Development System Documentation, 1996, Lucent Technologies Online documentation
[8] Hack, W. and Mayer, C.: Supporting Tools for a VHDL Coding Standard, 1994, VHDL Forum for CAD in Europe,Tremezzo, Proceedings pp 117 - 121
[9] Synopsys, Inc.: VHDL System Simulator Packages Manual, Version 2.2, 1991