Static Analysis for VHDL model Evaluation

Alessandro Balboni, Mirella Mastretti, Mario Stefanoni

ITALTEL-SIT, Castelletto di Settimo Milanese,
20019 Settimo Milanese (MI), ITALY

Abstract

Automated VHDL source analysis may be a valuable approach to develop, measure and compare models managed by high-level design tools. Moreover, VHDL code should be developed according to some well-founded guidelines and in a measurable way to improve the quality of the overall design process. This paper aims to describe in a detailed way a part of SAVE prototype exploring the areas of complexity analysis and simulation efficiency analysis starting from the VHDL system description.

1. Introduction

The VHDL 1076 IEEE standard is one of the most popular languages for building software models of hardware system. However, the increase in complexity of the system to be modelled brings a large addition of troubles in writing quality code. While emerging high-level design tools, such as System Design Station (Mentor Graphics), Express VHDL (i-Logix) and SpeedChart (SPEED), are able to generate VHDL source code in automated way, the typical VHDL-based design environment only provides tools for simulation and logic synthesis: no support is given at present in order to evaluate and improve the quality of VHDL code. An helpful aid to cope with large and complex VHDL descriptions and to develop, measure or compare models managed by all the above methodologies is needed.

Automated VHDL source analysis enhanced with advising capabilities, may assist the user in designing and improving source code. It should be pointed out that the goal of defining suitable quality measures for hardware description languages such as VHDL introduces specific aspects that may have no direct couterpart in the more assessed field of software engineering. From a general point of view, opportunities for quality analysis of VHDL descriptions should be investigated, at least, with respect to the following domains:

- code complexity and consistency
- efficiency of execution (simulation)
- feasibility of the synthesis process
- testability of the final implementation

Within each domain, general rules as well as rules tighly dependent on the specific design environment in use have to be found out. Possible conflicts between different analysis goals may also arise. Improving simulation efficiency, for instance, may reduce readability; observability or controllability, in the testing domain, may be in contrast with optimization carried out by automated synthesis. The present paper aims at providing an overview of the work in progress within a research project called SAVE (Static Analysis for VHDL Evaluation), aiming at developing methodologies and tools for automated VHDL quality analysis. The SAVE project includes theoretical analysis tasks as well as the implementation of prototype tools. From the theoretical point of view, suitable *metrics* have to be selected. Qualitative analysis of VHDL source code, in fact, is a completely unexplored research field and results from software engineering can provide only some limited suggestions. Because of the intrinsic complexity of the problem, heuristic techniques (giving approximate results) look more promising. A static analysis of VHDL models is performed in order to obtain measures of complexity, simulation efficiency, sinthesisability and testability. Moreover, an integrated expert system provides the user with advices to improve the code. The next sections focus on two particular domains of the quality analysis of VHDL descriptions: simulation efficiency and code complexity / consistency.

2. Efficiency of execution

Developing high performance simulation environments regards EDA vendors, but hardware designers exploiting

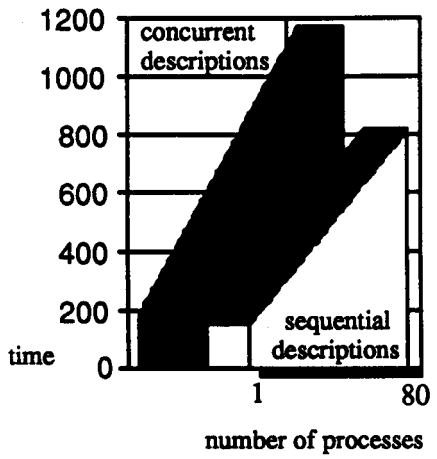


Fig 1: time spent in simulation for sequential descriptions versus concurrent ones.

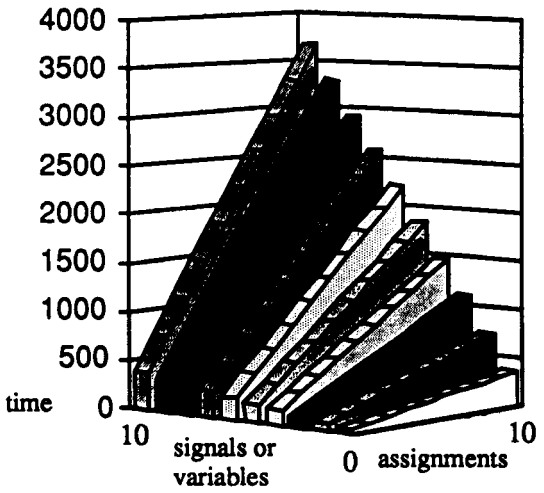


Fig 2: Simulation time differences using signals vs variables increasing assignments

VHDL can contribute to optimize simulation efficiency by writing source code according to specific guidelines in order to reduce the time spent in design verification. In [Lev91], [Hue91], [BGG92], a coding style is proposed and within the SAVE project it has been confirmed and enriched by experiments. In fact some guidelines have to be discovered on an experimental basis and some performance aspects may depend on the specific simulator.

2.1 Benchmarks and performance analysis

A lot of testbenches have been carried out to evaluate the efficiency of the different description styles and VHDL statements on QuicksimII (Mentor Graphics) and Vantage (View Logic) simulators. The same tests will be executed on other simulators (QuickVHDL, VSS) to confirm the general VHDL efficiency rules. These experiments have shown the higher efficiency of sequential VHDL descriptions versus concurrent ones.

Moreover, increasing the number of processes in the program, brings a large addition in simulation time as shown in figure 1: by replacing a concurrent description with an equivalent sequential one, a 20% reduction of simulation time has been observed. Moreover, a behaviour modelled using guarded blocks and guarded signal assignment statements, may often be managed by means of sequential processes: such a replacement causes simulation time to break down about 40%. In addition, reducing concurrency allows to get rid of a lot of signals that are less efficient than variables as shown in fig.2. Another critical aspect is the form used to specify the sensitivity list in processes: a static sensitivity list is more efficient than the equivalent dynamic one specified by only one WAIT statements. Also in this case the gained time increases with the number of processes in the program as shown in figure 4. Different performances for VHDL data types have been also observed: in general, integer types allow faster operations than logic types; moreover, for the latter ones, it may be computed a diagram in which speed decreases when the number of logic values grows; multiple value logic should be considered only for applications in which such level of detail is mandatory; for instance, in Mentor Graphics QuicksimII environment, there are special logic types QSIM_STATE and QSIM_12STATE, three times more efficient than STD_ULOGIC. The access to arrays and records is also slower with respect to the access to scalar variables. Finally, input stimuli are also performance-critical, so all the experiments have to be repeated in the same condition and with the same test patterns.

2.2 Strategies to increase simulation speed

Assuring that VHDL code is developed according to the following guidelines, may have a relevant impact on the time spent during the simulation phase:

a) follow classic rules of smart programming; this aspect includes ordering clauses in conditional structures according to execution probabilities, avoiding loop invariant statements, assigning intermediate results to temporary variables and minimizing the use of procedure calls; sequential and concurrent "case" constructs are typically more efficient than "if" statements, while less flexible (there are more opportunities for automated optimization);

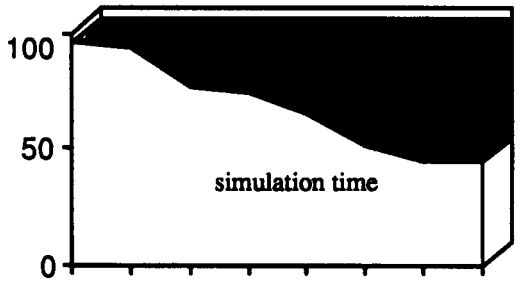


Fig 3: benefits achieved improving a VHDL description following the proposed rules.

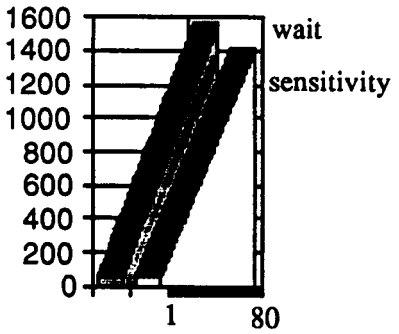


Fig 4: dynamic sensitivity list vs static one increasing processes.

b) give preference to a sequential style :

- joining together processes with the same sensitivity list, because they would be always executed during the same simulation delta;
- joining together processes sharing one signal as communication channel, to get rid of it.
- replacing signals with variables whenever possible;
- avoiding nesting of guarded blocks using

equivalent sequential processes in which the guarded condition is tested in a WAIT ON - UNTIL statement;

c) give preference to static sensitivity lists: replace dynamic ones whenever they are at the bottom of the process and they have no conditions;

d) choose the most suitable data types for the application at hand and avoid resolved signals whenever possible;

e) limit the use of attributes returning signals, such as STABLE .

It should be pointed out that the above guidelines provide faster code but in some cases may be in contrast with general readability, maintenability and other factors. Furthermore, speed bottlenecks are usually restricted to small specific sections of the whole program. An approach to investigate this performance-critical code, may be to weight the analysis results of that process with large sensitivity list. In fact, when the number of signals in sensitivity list is large, the process may be resumed by a large number of events and used in many different processing paths. For this reason, processes with large sensitivity list must be efficient: improving such processes may contribute in a significant way to increase simulation speed.

At present, commercial support tools for automated analysis of such kind of rules are not available. Therefore, a prototype system that statically analyzes VHDL source code for compliance to a specified performance-oriented coding style, and possibly suggests code modifications and/or improvements, has been developed. Figures 5, 6 below show some examples of execution of the system.

In the first example, a VHDL description developed ignoring the above guidelines is submitted to the efficiency analyser. Results are shown graphically to the user by means of an evaluation number generated computing the number and the weight of the violated metrics.

The user can also obtain a set of suggestions to improve the analysed code. The simulation of this model has taken 65 minutes. In the second example the same VHDL description, improved following the suggestion in figure 5, is submitted to the efficiency analyser. The simulation of the optimized model has taken 20 minutes.

3. Complexity and consistency

A significant effort has been devoted in the past to provide some kind of estimation for software complexity. The relevance of this issue is particularly related to the problem of maintenance of source code which, as well-known, involves different aspects such as modifiability, reusability, readability, and so on. A survey of the main approaches adopted up to now is not the goal of this section. However, in general, it should be noted that the results produced in the software engineering area are still controversial and are not widely applied in actual software development and management. In particular, such kind of metrics provide very approximate estimations, acceptable only for restricted class of applications. Moreover, programming languages proposed for hardware description, such as VHDL, introduce several specific issues which have no counterpart in more traditional software design (mixed behavioral/structural paradigms, event-driven behavior, deep concurrency, explicit timing, ...). While static analysis techniques for concurrent programs are emerging [GKB86], [RTY85], [Sha88], [CCK93],

several theoretical and practical issues still make VHDL analysis a very complex task. In fact, the concurrency model of VHDL is different at a large degree from the ADA model (from which many VHDL language constructs have been derived). Furthermore, most VHDL descriptions refer to explicit time concepts. The approach followed in the SAVE project, consists of investigating existing complexity metrics in order to discover if they may be adapted in order to satisfy VHDL requirements. The basic idea is to apply the most suitable sequential metrics to single modules (processes, procedures, functions) and develop higher-level metrics based on some cost function to estimate the complexity of the whole VHDL description. In the first prototype, complexity of single modules is evaluated computing Mc Cabe cyclomatic number: the number of computing path, decisional node complexity (number of operators), nesting level between control structures are considered complexity-critical issues. As general guidelines, there are two main issues that are critical in determining the complexity of VHDL descriptions. The first is timing: developers of VHDL models have to keep in mind that some statements produce effects (events) that are not immediate but are scheduled at some point in the future. The second is the influence among multiple processes, in particular in large descriptions with many shared signals and wait statements. These two guidelines should lead the research work towards more VHDL-oriented metrics. Anyway, we feel that qualitative metrics will be probably more successful than quantitative ones, because of the intrinsic complexity of the VHDL language.

Concerning correctness and consistency, a possible way of exploiting static analysis could consist of taking into account formal verification approaches recently proposed in many research works. However, in order to obtain short-term results (formal analysis, in our opinion, has more ambitious but long-term goals) it is important to focus on specific subclasses of VHDL models, such as for instance communicating finite state machines, for which some kinds of easier but still very useful static analysis may be carried out. As shown in figure 6, results of this analysis are the diagram of the fsm modelled and a set of controls on the conditions that control state transitions: if conditions that can not come true are found, they are displayed in dashed lines and possible trap-states or unreachable-states are highlighted. Also, in this case, timing is one of the most critical aspects to be captured by static reasoning. Anyway, many practical VHDL models do not explicitely address time (for instance, RTL synthesizable descriptions).

Some features improving readability of source code already implemented are shown in the examples in figure6 : complexity analysis results are presented by means of a mean complexity evaluation over all processes and a graphical specification of single process complexity. In this graph the sequence of processes in the description is represented with the abscissa enumeration.

Moreover a graphical browser exploits source-level analysis in order to display objects such as structures, processes, blocks and subprograms in a user-friendly way. Additional algorithms under development include the extraction of particular code sections translatable into procedures, by means of pattern recognition techniques.

4. Complexity - Efficiency Analyser Architecture

An architectural scheme of the prototype environment addressing efficiency and complexity analysis is depicted in figure 7. The LVS (LEDA VHDL System) commercial environment has been adopted to support the parsing step of VHDL source files, including semantic analysis concerning the standard language definition. LVS is also able to build an intermediate representation within an object-oriented database according to VIF (VHDL Intermediate Format) specifications.

Starting from the results of the parsing step, a custom tool (**Preprocessor**) builds a new representation more suitable for further processing by exploiting the LVS support for user-defined extensions to the basic VHDL schema.

Such an enriched representation collects all data needed for the computation of efficiency-oriented metrics such as the following: list of signals accessed by the process and access mode (read or write), kind of sensitivity list, attributes, number of processes accessing each signal, nesting depth for blocks and block guards.

The designer chooses to evaluate project in term of complexity or simulation efficiency and two different libraries are activated to compute the metrics embedded in the rule base.

A graphical interface module (**Presentation Manager**) enables the display of the above characteristics by using graphs, tables and diagrams and an example of its form has been shown in the previous sections.

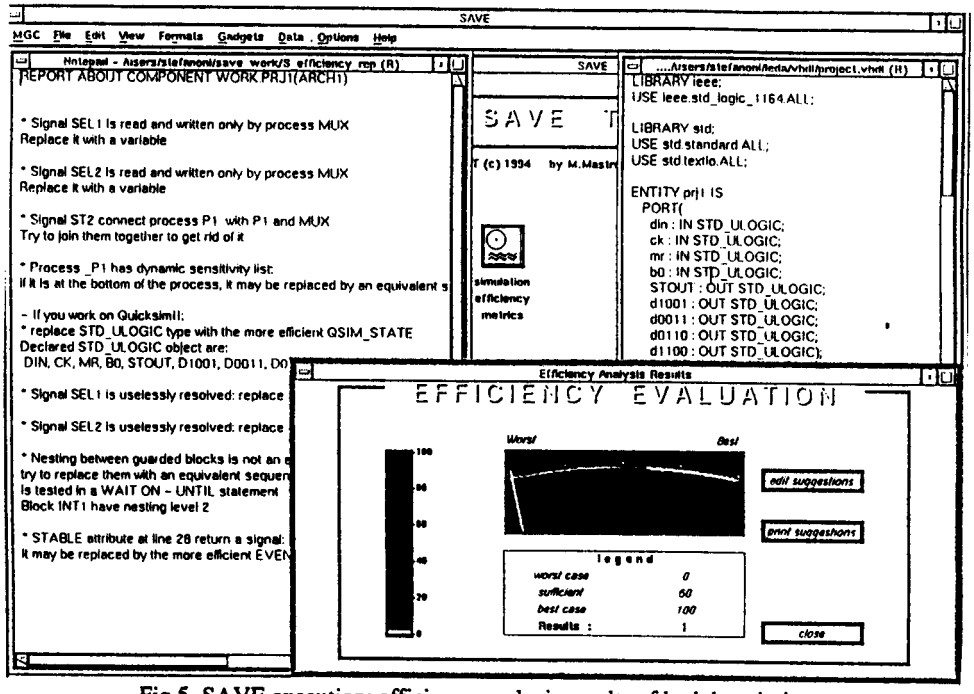


Fig 5 SAVE execution: efficiency analysis results of bad description

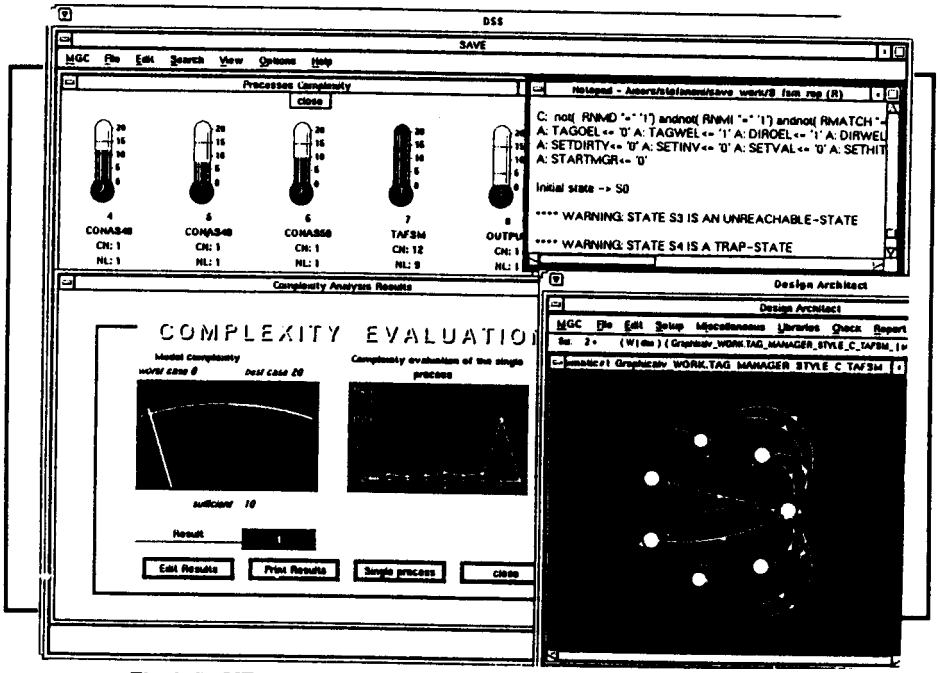
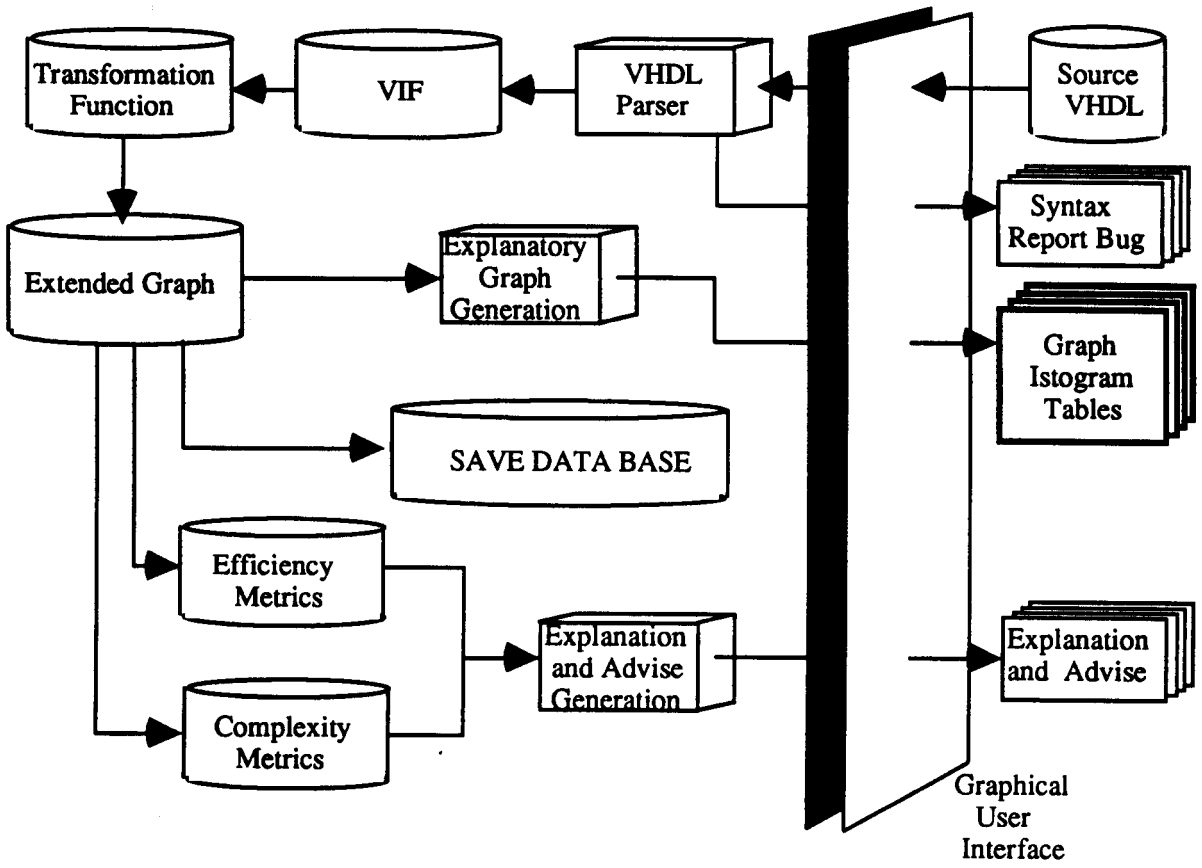


Fig 6 SAVE execution: complexity analysis results and fsm analysis

5. Conclusion

Goal of the SAVE project is the definition and the implementation of a set of tools to support the circuit designer to improve the quality of VHDL descriptions. In particular, four main parameters have been identified to evaluate the quality of VHDL code: *code complexity*, *simulation efficiency*, *feasibility of the synthesis* and *circuit testability*. The work presented in this paper concerns the definition and the implementation of new rules allowing VHDL code analysis focusing on the first two points of view. As discussed in the previous sections, only a limited number of research contributes really oriented to VHDL metrics are available up to now. Our project aims at filling the gap by exploring the areas of complexity and simulation efficiency analysis.

Fig 7: Architecture of the prototype environment for efficiency / complexity analysis.



References

[AKS92] E.J. Aas, K. Klingsheim, T. Steen "Quantifying design quality: a model and design experiments", Proc. of the 6th European Conf. on State-of-the-art ASIC, Paris, 1992.

[BKM90] S. Burson, G.B. Kotik, L.Z. Markosian, "A Program Transformation Approach to Automating Software Reengineering", Proc. of the 14th Int. Computer Software and Application Conf., 314-322, Washington, D.C., IEEE Computer Society

[C&A91] C.H. Cho, J.A. Armstrong, "VHDL Semantics for Behavioral Test Generation", in *Computer Hardware Description Languages and their Applications*, D. Borrione, R. Waxman (Editors), 1991 Elsevier Science Publishers B.V., North Holland, pp. 427-444.

[C&G93] S. Carlson, E. Girczyc "Increasing Design Quality and Engineering Productivity through Design Reuse" Proc. of the 30th Design Automation Conf., Dallas, Texas, 1993

[CCK93] S. Cha, I.S. Chung, Y.R. Kwon, "Complexity measures for concurrent programs based on information-theoretic metrics", Proc. of the 30th Design Automation Conf., Dallas, Texas, 1993.

[GKB86] J.D. Gannon, E. Katz, V. Basili, "Metrics for ADA packages: an initial study" Communication of the ACM, July 1986.

[Hue91] M. Hueber, "VHDL experiments on Performance", EURO-VHDL '91.

[Lev91] O. Levia, "Writing High Performance VHDL Models" EURO-VHDL '91.

[RTY85] J. Ramamoorthy, W. Tsai, T. Yamaura, A. Bhide, "Metrics guided methodology" Proc. 9th Computer Software and Application Conf. Oct.1985.

[Sha88] S. Shatz, "Towards Complexity Metrics for ADA tasking", IEEE Transactions on Software Engineering, August 1988