EXPERIENCE WITH THE VHDL ENVIRONMENT

M. Loughzail, M. Côté, M. Aboulhamid, E. Cerny

Dép. d'informatique et de recherche opérationnelle
Université de Montréal
P.O.Box 6128, Station A
Montréal, Qué., H3C 3J7 Canada

ABSTRACT

The purpose of this paper is to present our work in the use of the VHDL environment on three different models of DEC VAX (1, 4, and 5.8 Mips). The work included the development of a number of VHDL models and sequences of input stimuli and the gathering of performance data from their execution. Even though the set of benchmarks is not in any way exhaustive, they represent typical applications, allowing us to derive performance models for predicting both time and memory requirements for VHDL modelling and simulation. In addition, we show that the performances of the VHDL environment on the three computers correlate perfectly with their Mips figures; thus the performance models developed can be normalized to a 1 Mips VAX machine.

1 INTRODUCTION

The development of large integrated hardware/software systems usually requires that multiple independent suppliers provide the various components of the system. Therefore, non-ambiguous specifications must be created in a formal language that can be validated (by simulation) and which can serve for the suppliers to verify their designs (formally or by simulation). Furthermore, during the verification of the overall system design, all components must be described in a language that can express both structural and behavioral descriptions at various levels of abstraction. For this purpose, the DOD recently sponsored the development of the VHDL standard description language and programming environment [5,19,20,21].

The VHDL environment is currently available only on the DEC VAX machines operating under the VMS operating system, and it is built on top of the ADA software environment. It is a very complex system which is continuously being improved. The acceptance of the language and the environment by industry and universities depends on both the modelling power of the language and on the performance of the supporting environment. Unfortunately, not much real study of the use of the VHDL environment has been done and reported in the open literature. In view of the efforts to adopt this language as an industry-wide standard by the IEEE [5], we feel that performance models are needed that could help the potential users to estimate the requirements on computing resources and execution times for both model building and simulation. Furthermore, problems which are encountered when using the VHDL environment could help in guiding its further development.

The purpose of this paper is thus to present our work in the use of the VHDL environment on three different models of DEC VAX computers located at the Centre de Recherche Informatique de Montréal: VAX 8600, VAX 8650 and MicroVAX II. We are using the latest release of VHDL 7.2 (which was delivered to selected sites during summer 1987). We run the environment under VAX VMS 4.5 using ADA 1.4. The work includes the development of a number of VHDL models and sequences of input stimuli and the gathering of performance data from the execution of these models and other benchmarks obtained elsewhere. Even though the set of benchmarks is not in any way exhaustive, they represent typical applications, allowing us to derive performance models for predicting both time and memory requirements for VHDL modelling and simulation. In addition, we show that the performances of the VHDL environment on the three computers perfectly correlate with their Mips figures; thus the performance models developed can be normalized to a 1 Mips VAX machine.

Section 2 contains a brief description of the VHDL environment, the benchmark models and their validation, and the problems encountered. Section 3 first discusses the performance criteria selected and the techniques used for data normalization and performance model development. Then the performance measurements and models are shown, with a discussion about factors influencing the performance figures and comparison with other environments. Finally, in Section 4, we conclude the presentation with a number of recommendations for the improvement of the VHDL environment.

2 MODELLING WITH VHDL

2.1 Processing steps

Once a model has been developed it is processed by the following tools:

- The VHDL analyzer : The program is processed by the analyzer, and if fully completed, the components it describes are stored in the Design Library (DL) in Intermediate VHDL Attributed Notation (IVAN).
- The Model Generation (MG) : This produces an ADA source program from IVAN data stored in the DL and then compiles it. The object module is stored in the Simulation (host ADA) Library (SL).
- Build : This program builds an executable code, called Kernel, for simulation using the object modules from the SL and pre-compiled Run Time routines. The Kernel is stored in the DL.
- SIM : This program executes the Kernel. The result is a RUN node created in the DL which holds the events produced on signals during the simulation session.
- The Report Generator (RG): This uses the Run node and a format file (provided by the user) to produce a simulation report which indicates the values held by signals during simulation.
- The simplifier: This can be used optionally to flatten hierarchical descriptions.

2.2 Description of models

The chip Modeling Group of Virginia Tech. provided us with some benchmarks, and we developed others ourselves. Among the designs we have simulated, five were selected for the performance measurements : a 16-bit up counter, an ALU, a Traffic light controller, a microprocessor (INTEL 8085), and a small floating point coprocessor (AM9512). For each of these circuits we had different descriptions as explained below.

The counter (four 4-bit up counters 74F163): we have four descriptions of this 16-bit up counter.

a. A full gate level description: this description uses a hierarchical approach where the primitives are AND, OR, and NOT gates. The

different levels of the hierarchy contain the primitives, the D Flip-Flops, 1-bit slice, a 4-bit counter and then the 16-bit counter.

b. A higher structural description where a D Flip-Flop is considered as a primitive and described at a behavioral level.

c. The same structural description as in b, but we disable the CLOCK signal while there is no event on the D input of the Flip-Flop.

d. A full functional description.

The ALU : we modelled a 4-bit ALU (SN74181) using a functional description and a structural description with three levels of hierarchy.

The Traffic light controller: we have three descriptions for this circuit: a behavioral description in which the traffic light controller is seen as a finite state machine, a structural description containing D Flip-Flops and logical gates, and a PLA description in which the traffic light controller is seen as a PLA where the AND and OR planes are described at the functional level.

The floating point coprocessor (AM9512) and the microprocessor 8085: These are both described at a register transfer level. The control of the 8085 is described as a microprogram.

2.3 Model validation

An important issue when a design is described in a Hardware Description Language is how to verify that the program is carrying out the desired behavior. The VHDL environment offers different ways to do this; the user provides inputs to the simulator and analyzes the output produced by the simulator. The inputs may be *exhaustive* so that the design is simulated for all possible input values and all possible state transitions. The inputs also may be *limited* to a subset of all possible input combinations. This is of course an incomplete test. It was the case for the ALU (tested with a few arithmetic and logical functions) and for the 8085 (tested with a 2 byte product). It was also the case for the 16-bit counter which was simulated to count from 0 to $2^{16} - 1$.

The ASSERT statement in VHDL offers an interesting way of validating a model without having recourse to a time consuming analysis of the simulation output. These statements, when inserted in a program, will be triggered by a given condition (usually a signal or a variable value). A message may be displayed while the design is being simulated. Depending on the severity level of the ASSERT, the simulation may or may not be halted as soon as its condition is verified. The ASSERT statement will be used to verify complex timing requirements or expected behavior (e.g., for an incremental resetable counter the present value is either zero or greater than the previous value). In the three descriptions of the Traffic Light Controller, a wide use of this statement is made to check if the lights are following the flow of cars. ASSERT statements do not significantly slow down the simulation time.

2.4 Learning VHDL

An important characteristic of any programming language is the ease with which it can be learned and the time required to become proficient in it.

The closeness of VHDL to the ADA language allows people already familiar with ADA to learn VHDL faster than others. It took 2 weeks for a member of our group who was familiar with ADA environment to feel at ease with VHDL. More than 3 weeks were necessary for another member to understand the VHDL philosophy.

As the concept of parallel statements execution does not exist in most ALGOL-like languages, the most obscure issue with VHDL was related to concurrent assignment statements, and the distinction between Block and Process structures. Delay issues and bus resolution functions are also new concepts not existing in the most widely used programming languages. But the language is intended, first of all, for hardware designers, who should be already familiar with all the concepts cited above.

A certain amount of time must also be spent to become familiar with the VHDL environment (Design Library, ADA Library, the simulator,...etc.).

2.5 Problems encountered

User Errors

Errors are very easy to correct when detected at the analysis step. These are syntactic and semantic errors related to the language itself. We estimate that more than 90 % of errors are of this kind.

About 7 % of errors occur when the model is being generated, more specifically when ADA code is being compiled. These errors are related to ADA source code generated. Assignment of arrays are frequent causes of this kind of errors. The large size of the ADA code and the names assigned to the various elements of the original VHDL code makes it very hard, if not impossible, to debug. Hence, we were obliged to proceed by trial and error on the VHDL source program to check out the errors. Even after discovering the "guilty code portion", it was often not obvious how to find out what was wrong with it.

The third kind of error (about 3 %) occurs during simulation. These are the most frustrating ones. An index out of bound could be an example of these errors. The only indication given by the simulator was the occurrence time of the error, and sometimes the code portion where the error occurred (Process or function or procedure name), but not the kind of error itself. Here again, the only way to check out the errors was to proceed by trial and error.

When the simplifier was used (on the counter gate level description) we encountered two kinds of errors. The first ones were caused by incorrect signal names (for example, a signal declared by the simplifier as "CLOCK_1" was invoked in the program body under the name "CLOCK_1_2"). The second kind of error occurred during ADA code generation (the message was "Internal Generator Error--please report"). Note that the source VHDL code generated by the simplifier reached 1450 lines at this moment .

Initialization problem in the simulator

In spite of our self confidence in the accuracy of the ALU gate model, we obtained incorrect results during simulation. We discovered that, at the start of the simulation step, there was in fact a '0' assignment to every signal in the circuit. Consequently, the signals to which we were assigning Zero values in our test_bench were not responding to this stimulation, as there was no event generated (a change in the value). As a result, they were not carrying the correct values to the output of the circuit.

To remedy this deficiency, we thought of two solutions. The first one was to initialize "by hand" each signal to the value that it should have at the start of the simulation. For example, if we set the input of an invertor to '0', we should set its output to '1' (while the simulator would set them both to '0'). Obviously, this solution is very hard to apply, especially for large circuits.

The second solution was to set the signals to a value other than '0' and '1', in order that a '0' assignment to those signals would be considered as an event and carried through the circuit. This is the solution for which we opted. We defined the type TERNARY ('0', '1', 'U'), and we set all signals to 'U' at the start of the simulation . This solution is very practical for structural descriptions where the number of signals is high, on the other hand as this number is smaller in behavioral descriptions the signals can be easily set by hand. A third solution using processes was proposed at [22].

Environment problem

An important problem not related to program bugs, but to the system as a whole, concerns the Design Library. When a user is removed from the Design Library, all the nodes in its context are of course deleted, however all the files in the user's directory and sub-directories are also deleted. We estimate that it is a system failure because there is no logical relationship between the Design Library and the user directory (which may contain any kind of files not related to VHDL).

3 PERFORMANCE MEASUREMENTS AND MODELLING

3.1 Performance criteria

The criteria considered in our measurements are the following :

- The size of : the VHDL code , the IVAN structures, the ADA source code, object code, the VHDL source code generated by the simplifier and the RUN file generated by the simulator.

- The CPU time for the main steps necessary to simulate a VHDL description: VHDL analysis, ADA code generation and compilation (MG), Kernel generation (Build and Simulation time.

We did not consider the Working Set size because none of the operations above made the Peak Working Set reach its maximum size.

All the measurements described above were taken twice : using the MG/CHECK and the MG/NOCHECK options. The use the MG/CHECK option allows the detection during simulation of some VHDL errors (an index expression out of bounds, multiple drivers for non-bus signal,...etc) but it increases the time and space requirements.

We were obliged to record the simulation trace for only a small part of the simulation session, otherwise the RUN file size would have exceeded many times our disk space limit. The inhibition of this trace improves the simulation time and space requirements [21]. The drawback is, of course, the fact of not having the results of the entire session.

3.2 Performance modelling technique

Normalization

The modeling and simulation were performed on three different machines: MicroVAX(1 Mips), VAX8600(4 Mips), and VAX8650(5.8 Mips), with the aim to derive normalized performance figures that are independent of the machine. The characteristics of the 3 computers are summarized in Table 1. Disk space allocation was about 30 Mbytes per user. The Analysis, MG, Build and simulation times obtained on the VAX8600 and the VAX8650 were compared with those obtained on the µVAX. It was discovered that the performances on the three computers correlate perfectly with their Mips rates. For example, a description which required 1 sec. to be analyzed on the VAX8600 would require 4 sec. on the µVAX, while one requiring 1 sec. on the VAX8650 would take 5.8 sec. on the µVAX (fig. 1). The normalized time thus refers to the performance on a one-Mips machine.

Table 1: VAX computers characteristics

Computer	Memory	Mips	WSDEF	WSQUO	WSEXT
VAX 8650	44 MB	5.8	1,024	2,048	8,192
VAX 8600	"	4	"	"	"
µVAX	16 MB	1	"	"	"

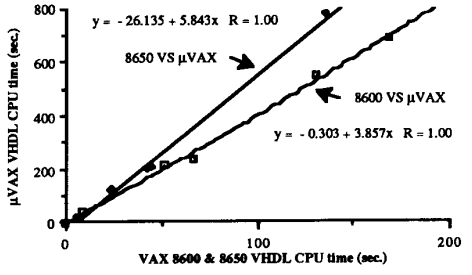

Fig. 1 :correlation between time and Mips(VHDL analysis).

Regression analysis

We aimed to derive some global formulas on the resources (time and space) necessary to process a description, given some of its features (e.g. size or number of events occurred during simulation). To do so, after running a number of different designs on the 3 computers and gathering results, we used the multiple regression capabilities of the SPSS^x package to find possible correlations between various criteria, such as VHDL code size VS ADA code size, VHDL code size VS analysis time, etc. Regression through origin was requested (i.e., an empty ADA file will generate an empty object code file). We started with a linear regression. After observing that the coefficient R^2 of explained variation was very good (> 98%) in all cases, and that choosing a higher degree polynomial did not improve significantly this coefficient, we opted for the simple solution y = ax.

Even if only 5 different circuits were modeled, we ended up with more than one hundred nodes in the design library (architectures, packages, interfaces). This allowed us to have a fairly wide range of observations, for example, the Model Generator was executed on most of the library nodes.

3.3 Normalized benchmark measurements and performance models

All the performance models suppose that we use a 1 Mips VAX machine and MG/Nocheck option only. Counter models were simulated for a count from 0 to 2^{16} - 1.The ALU was simulated using the 32 arithmetic and logical operations, each of them provided with 8 distinct pairs of 4 bits. The Traffic light Controller was run for 200 min (simulated time). The 8085 simulated a 25 line program performing a multiplication of two bytes. Only designs producing less than 12,000,000 events were simulated on the MicroVAX, otherwise, CPU time would be unacceptable. The results of the regression analysis are summarized in tables 2&3.

Table 2: size prediction

Estimated parameter		Regression equation	Coeff of explained variation
Kernel generation			
IVAN structure (bytes)	=	12 * VHDL code (bytes)	.985
source ADA code (lines)	=	10.4 * VHDL code (lines)	.976
object code (bytes)	=	18.4 * source ADA code (lines)	.994
Simulation			
RUN size (K bytes)	=	62.9 * K events	.998

Table 3: CPU time prediction

Estimated parameter		Regression equation	Coeff of explained variation
Kernel generation			
Analyzer time (sec)	=	0.47 * VHDL code (lines)	.985
Model Generator time (sec)	=	0.179 * ADA code (lines)	.982
Build time	=	90	
Simulation			
Simulation time (sec)	=	2.76 * thousand events	.995

Note that the Model generator time includes ADA code generation and compilation. As the variation of the Build time was small (75 to 105 sec) we took an average value of 90 sec.

Figures 2 to 5 plot the size values observed and the corresponding regression lines; while figures 6 to 8 show the CPU time requirements.

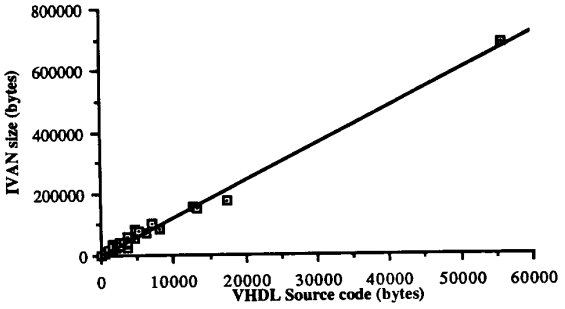

Fig.2: Size of IVAN structures

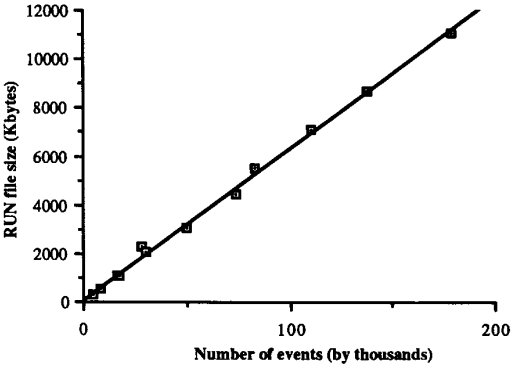

Fig. 5: RUN size

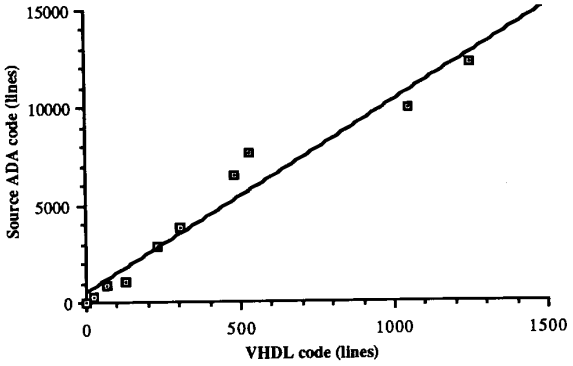

Fig. 3: ADA code size

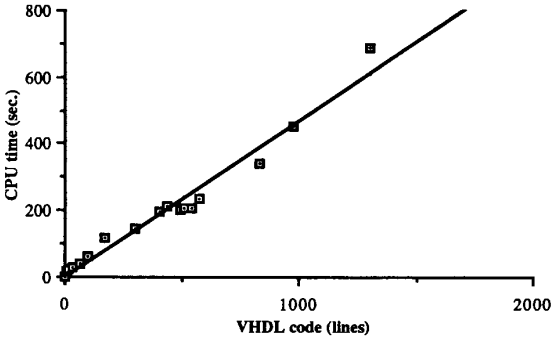

Fig. 6: Analysis time

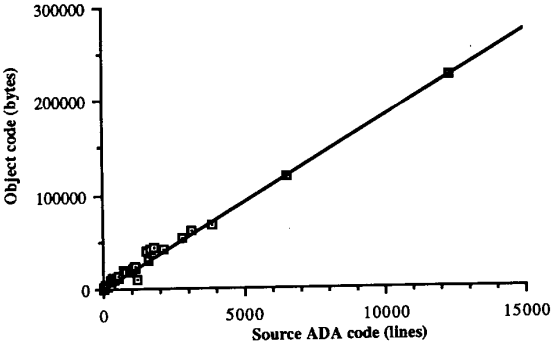

Fig. 4 : Object code generated.

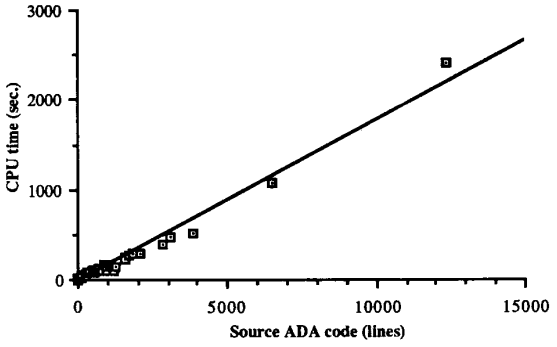

Fig. 7 : MG time.

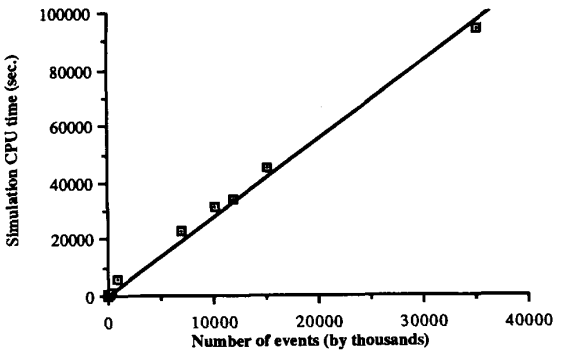

Fig. 8 : Simulation time.

To illustrate these rules, let us consider a 500 line VHDL program, generating 1M events, run on a 1 Mips computer: 5200 ADA code lines will be generated, which, once compiled will give 95680 object code bytes, with the run file needing about 63 MB. It will take 4 min for the analysis, 1 min 30 sec for the Build step, 15 min for model generator step, and 46 min for the simulation.

3.4 Factors influencing performance

Effects of /Nocheck option

When the /Nocheck option is in effect the Source ADA code generated is, on the average, 18.46 % smaller than when /Check is used and the average decrease of the Model Generator time 28%. An average of 10% reduction is observed on the Build time. The decrease of simulation time varies widely, from a negligible change up to 47.5 % of time reduction.

Effect of programming style :

Logical gates used in the 16-bit counter were initially described as generalized gates receiving an n-array of Ternary signals as input, and producing a single Ternary signal as output. The loop necessary to pass through the array was included in a process structure (language restriction). When ad hoc gates were used (each gate receives a fixed number of signals as inputs), the process and loop structures were no longer necessary, and they were replaced by simple signal assignment statements. The CPU time was reduced by more than 50 % in the case of the counter gate description (table 4). In the other descriptions the decrease was less important, due to the fact that they use the gates less often if at all.

Table 4 : Effect of programming style on simulation time (VAX 8650).

Design	Nb of events (Millions)		Sim. CPU time (min)	
	gen.	ad hoc	gen.	ad hoc
Gate desc.	35.2	10.7	270	107
FF behav.	15.2	12	121	97
Disable clk	10.2	6.9	93	68

Clock suppression

Minimizing clock pulse duration is an important factor to speed up simulation [14, 18]. It consists on evaluating the Clock signal at adequate moments only (and then to desensitize it during the remaining time). A simulation time decrease of 30 % was obtained (Table 4).

Description level

As expected a behavioral description is more compact and requires less resources (table 5).

Table 5: Abstraction level effect

Design	VHDL code (lines)		Simulation time (min on a µVAX)	
	Struct.	Behav.	Struct.	Behav.
16 bit counter	750	320	627	95
ALU	738	325	4	0.08
Traffic light controller	530	300	17	9

The Simplifier

We used the simplifier for Counter gate description. As mentioned in section 2.5, we could not totally simulate it. Nevertheless, we could simulate the 1-bit slice. Simulation time decreased from 32.01 min. (usual simulation) to 25.01 min. But the code increased from 79 VHDL lines to 726 lines.

3.5 Comparison with other tools

The comparisons we made are based first on the modelling and simulation of the 16-bit up counter and of the 8085, and on some typical simulation figures given in the literature. We compared our results against results obtained by Schlumberger BEHAVE system [14] and BNR FUNSIM system [6]. The VHDL, BEHAVE and FUNSIM environments were running respectively on VAX8650 (5.8 Mips), SUN workstation (1 Mips) and IBM3033 (4 Mips).

Code generation

Tables 6 and 7 give the size and time requirements for the code generation concerning the 8085. Note the gap between sizes of C, generated by BEHAVE, and ADA code generated by the VHDL simulator (a factor of 6). A large gap (a factor of 5) also exists between respective FUNSIM and VHDL object codes. VHDL CPU time requirements are also larger than for the other systems.

Table 6: Code size (8085)

Group	Description (lines)	host code (lines)	object code (bytes)
Schlum.(BEHAVE) SUN (68010)	1832 BML	3K C	-
BNR(FUNSIM) IBM 3033	- FML	- Fortran	60K
VHDL	1800 VHDL	17K ADA	310K

Table 7: Analysis & generation time (8085)

Group	Analysis (sec)	lang. gen. (sec)	compilation (sec)
Schlumberger	-	96	154
BNR	3	-	9.1
VHDL	107	45	642

Simulation

Table 8 gives simulation time required for the counter. Note that in the best case, VHDL is 4.8 times slower than BEHAVE.

Table 8: 16-bit counter simulation time (min)

	gate desc	behav. FF	disable clk	funct.
Schlumberger SUN(68010): 1 Mips	129.43	36.04	24.29	-
VHDL VAX 8650: 5.8 Mips	107.41	97.47	67.59	17.3

The performance of a logic simulator on a 1-Mips machine should be around 2000 events/sec [2]. The simulator currently included in the VHDL environment performs at about 360 events/sec, which is quite below the

desired rate. In comparison, the simulators CSIM, ESIM, DECSIM [7,10, 12, 18] approach this 2000 events/sec. rate, and much higher values are reported [1, 2] using hardware accelerators.

4 CONCLUSION

In this work, we derived a performance model for predicting both time and memory requirements for the present VHDL environment running on VAX machines. We are aware that we need more data to improve our performance model, by adding other criteria not included in the regression analysis at the moment. These criteria may be, for example, the influence of different kinds of description abstractions and the expertise of the programmer who designs the model. Nevertheless we can draw the following conclusions.

Presently, the Kernel generation (VHDL analysis, MG, Build) can be done on a 1-Mips machine, however the simulation has to be performed on at least a 6-Mips machine otherwise the CPU time will be unacceptably high. The debugging can be improved to allow faster model development. Improvement of user interfaces may help considerably for the acceptance of VHDL by a broad category of users. These kinds of tools (command menus, synthesis tools, graphical tools,...etc) as well as bridges to other standard languages (like EDIF and IGES) are starting to emerge [3, 9, 16, 17] and are well accepted by the designer community.

When comparing VHDL to other tools, we noticed that code generated by VHDL is too large. We estimate that reducing the size of ADA code generated may improve overall VHDL performances, especially simulation time. A C-implementation of the VHDL environment is under development [22] and is expected to be one order of magnitude faster and more space efficient than the ADA-version. In that case, we will have a very powerful modelling environment with performances comparable to other existing tools. This will make VHDL acceptable to a wide community of 1-2 Mips workstation users. Nevertheless, even with these improvements, VHDL still will be time-consuming for educational purposes, especially at the undergraduate level where there are relatively limited resources and large numbers of students.

It is important to note that, independently of the VHDL environment, the simulation is a very time-consuming process and that severe performance is the main limitation of software simulations running on general purpose computers . For this reason, hardware accelerators and simulation engines [e.g. 1, 2] are used for logic-level simulation. It would thus be desirable to develop such machines for accelerating high-level functional simulation [8]. Such simulation machines take advantage of the high speed allowed by concurrency, distributed processing [4,11,13,15] and pipeline architectures. Running VHDL environment on such machines could give very interesting performance figures, as its potential concurrency can be utilized in a more interesting way than on a general purpose machine.

ACKNOWLEDGMENTS

This work was partially supported by NSERC Canada Grants No. G1582 and A0861. We also thank the Centre de Recherche Informatique de Montreal (CRIM) for making freely available the 3 VAX computers and the DoD for providing us with early versions of the VHDL environment.

REFERENCES

[1] M. Abramovici, V.H. Levendel, P.R. Menon, "A Logic Simulation Machine", *Proc. 19 th. ACM/IEEE Design Automation Conference*, 1982, pp. 65-73.

[2] N. P. Van Brunt, "High performance simulation engine applied to large system design", *Proc. IEEE ISCAS*, 1985, pp. 35-38.

[3] R. K. Chun, K. J. Chang, L. P. McNamee ,"VISION : VHDL Induced Schematic Imaging On Net-lists", *Proc. 24 th. ACM/IEEE Design Automation Conference*, 1987, pp. 436-442.

[4] J.Cloutier , Roy C., E. Cerny , "A Distributed Mixed-Mode Hierarchical Simulator", *Proc. 14th IASTED Conf. on Simulation and Modelling*, Vancouver, June 1986.

[5] IEEE Design & Test of Computers, *special issue on VHDL, April 1986.*

[6] P.J. DesMarais, E.S.Y. Shew, P.S. Wilcox, "A Functional Level Modeling Language for Digital Simulation", *Proc. 19 th. ACM/IEEE Design Automation Conference*, 1982, pp. 315-319.

[7] E. Frey, "A Functional Level Simulation Tool", *Proc. ICCAD-84*, 1984, pp. 48-50.

[8] D.J. Geiger, D.E. Thomas, "Special Purpose Hardware for Algorithmic Level Simulation", *Proc. ICCAD-87*, 1987, pp. 496-499.

[9] J. Hines, "Where VHDL fits within the CAD Environment", *Proc. 24th. ACM/IEEE Design Automation Conference*, 1987, pp.491-494.

[10] A.K. Insinga, "Behavioral modeling in a structural logic simulator", *Proc. ICCAD-84*, 1984, pp. 42-44.

[11] D. Jefferson, H. Sowisral, "Fast Concurrent Simulation Using the Time Warp Mechanism", *1985 Distributed Simulation Conference*, San Diego, January 1985.

[12] M. R. Lightner, P. H. Moceyunas, H. P. Mueller, B.L. Vellandi, H.P. Vellandi, "CSIM: The Evolution of a Behavioral Level Simulator: Implementation issues and Performance Measurements", *Proc. ICCAD-85*, 1985, pp. 350-352.

[13] M. R. Lightner, "Modeling and Simulation of VLSI Digital Systems", *Proceedings of The IEEE*, Vol. 75, NO. 6, June 1987, pp. 786-796.

[14] A. Miczo, D. Mohapatra, S. Perkins, K. Kaufman, K. Huang, "The effects of Modeling on simulation Performance", *IEEE Design & Test of Computers*, April 1986, pp .46-54.

[15] W. Najjar, J. L. Jezouin, J. L. Gaudiot, "Parallel Discrete-Event Simulation", *IEEE Design & Test of computers*, December 1987, pp. 41-44.

[16] L. F. Saunders, "The IBM VHDL design system", *Proceedings of 24 th. ACM/IEEE Design Automation Conference*, 1987, pp. 484-490.

[17] M. Shahdad, "An interface between VHDL and EDIF", *Proc. 24th. ACM/IEEE Design Automation Conference*, 1987, pp. 472-478.

[18] E. Ulrich, D. Hebert, "Speed and accuracy in digital network simulation based on structural modeling", *Proc. 19th ACM/IEEE Design Automation Conference*, 1982, pp. 587-593.

[19] VHDL Analyzer User's Manual, *Intermetrics, Inc.*, 1987.

[20] VHDL Language Reference Manual, *Intermetrics, Inc.*, 1987.

[21] VHDL Simulator User's Manual, *Intermetrics, Inc.*, 1987.

[22] 2nd VHDL User Conference, Blacksburg, Virginia, October 1987.