

CRITERIA FOR THE EVALUATION OF VHDL SIMULATORS¹

David A. Levine, Graduate Student and Ronald Waxman, Fellow IEEE
University of Virginia
Center for Semicustom Integrated Systems
Charlottesville, Virginia 22903

ABSTRACT

This paper presents approaches for the evaluation of VHDL simulators that are straightforward to apply. Performance indices are outlined by which a simulator - and other VHDL tools - may be evaluated. Two types of benchmarks, simple synthetic and composite synthetic, are defined. These are the cornerstones in the evaluation scheme that was developed. The structure, requirements, and considerations in the use of these benchmarks are also given. Means are proposed to reduce the effects of some of the factors that can cause variations in simulator evaluation results. A description of some of the benchmarks that were developed to validate the evaluation methodology is presented. Finally, sample results are given and discussed. For the evaluation scheme that was developed, the principal parameter of interest is execution time.

INTRODUCTION

VHDL, an IEEE/ANSI standard, is becoming the most popular language in its class among the design and academic communities. Numerous vendors are now offering a variety of products to support the language. Among these, the simulator - always present in every VHDL software system - is a tool of utmost importance to the designer. The latest version of the VHDL language is now almost five years old. To date, only isolated efforts have been made to evaluate the tools of a VHDL software system. A validation effort for the analyzer tool was pursued and completed in [ARM89]. A number of benchmarks designed to test mainly the analyzer and the simulator tool of a VHDL system can be found in [SER89]. The latter reference provides a good starting point to understand the state-of-the-art in VHDL simulator evaluation. We have developed simpler and more flexible techniques for the evaluation of a VHDL simulator. These techniques, in contrast to [SER89], execute totally in a given simulator's native VHDL environment.

The objective of the study leading to this paper was to compare the performance of commercially available simulators to the performance of a VHDL simulator that will run on a multiple instruction/multiple data type of computer. The parallel VHDL simulator is being developed at the University of Cincinnati. The overall project, Quest, is described in [CAR91]. This paper presents the methodologies and techniques that have been developed at the University of Virginia as a result of this study.

PERFORMANCE INDICES

The evaluation of a VHDL simulator is a highly complex task that involves the identification, selection, and measurement of its most important characteristics. The parameters of interest for an evaluation can be summarized in terms of a number of performance indices. Such indices describe a quantity or quality that is used to measure a capability, attribute, or characteristic by which a system carries out a task. According to this, the performance indices of a simulator can be separated into two broad categories: quantitative and qualitative. The result of a quantitative index is always a number. Often, this number is accompanied by units that reflect the characteristic(s) being measured. Qualitative indices are used to reveal conditions, capabilities, characteristics and virtues of the simulator being evaluated. Contrary to their quantitative

¹ This work was partially supported by the Defense Advanced Research Project Agency and the University of Cincinnati under contract number 7056 (DARPA), and monitored by the Federal Bureau of Investigation, contract J-FBI-89-094.

counterparts, the results of this type of index lack any units. Some qualitative indices may be objective in that they answer the question of whether or not a particular feature exists. Others are subjective. Results of qualitative indices usually take the form of a binary answer, a written description, or a quantity in some arbitrary scale. The classification of performance indices that is presented next is by no means complete. Many of the presented indices cannot be applied solely to the simulator component, but have to be considered for the overall VHDL software system (analyzer and simulator together).

A comprehensive evaluation of even a small number of VHDL simulators can become a formidable complex task due to the potentially large number of tests that can be drawn from all the performance indices that are presented. The evaluation methodology that was developed in this study is based primarily on the execution CPU (user) time performance index. This index was chosen because, contrary to the qualitative indices, execution time tests are objective, their results are unambiguous, and highly immune to any bias of the individuals conducting the evaluation. Furthermore, for many users, the speed of a simulator is considered to be one of the principal (if not the most important) characteristic of a simulator. Nevertheless, a significant subset of potential performance indices that could be considered, in addition to execution time, is presented in the following section for the sake of completeness. Only execution time is included in the evaluation methodology given in this paper.

Quantitative Indices

- (1) Execution Time The evaluation scheme defined in this paper uses execution time as its principal parameter of interest. The execution time performance index requires the measurement of the time needed to simulate a particular hardware model. The time of simulation can be measured in terms of the elapsed (wall-clock) time, but is usually evaluated in terms of execution CPU (user) time. System time is not considered because it is usually very small in our experiments. There are two types of tests that can be used to measure the execution time index. These are: a) capacity tests, and b) atomic tests. Capacity tests are used to measure the quantity of information processing that a simulator can complete in a unit of time. Here, information can mean, to mention only a few examples, lines of source code [LOU88], number of gates [GRE87], events, and signals. If the objective of the test is only to measure the time required to simulate a model, then the test is simply called an execution (CPU) time test. Atomic tests measure the time required to simulate individual VHDL constructs. Although the term VHDL construct usually implies a single statement or expression, it can also be applicable to a set of coherent, closely-related statements that accomplish a meaningful function.
- (2) Memory Usage The memory usage index is used to establish the amount of memory that the system requires during and after the simulation of a hardware model. Both RAM and disk memory are candidates for evaluation. Two types of measurements can be contemplated: 1) peak requirements (RAM), 2) size of resulting files (disk).
- (3) Turnaround Time The turnaround time index requires the measurement of the interval of time that elapses from the moment a hardware model is submitted for processing (model is available) up to the moment of completion, when simulation results are obtained. The total turnaround time is not only the simulation time, but also intrinsic overhead times plus times required by other pre- and post-processing tools (e.g., initialization and load times, analyzer time, compiler time, and report generation time).
- (4) Throughput Rate The throughput rate index is employed to determine the amount of useful work performed by a computer system in a given time interval (wall-clock time). To determine the throughput rate of a system under a given load, a time window is established, and the number of jobs completed during this interval are counted; the throughput rate is the ratio between the job count and the elapsed time of the observation.
- (5) Cost By itself, this index cannot be considered to be a performance index, unless it appears as a component in a ratio with another performance index. Nevertheless, the cost of a VHDL system often is one of the primary factors that influences purchasing decisions.

Qualitative Indices

- (1) User Satisfaction Often, a system may be advertised as one having magnificent power and capabilities, but if the user finds it annoying, no useful work will be accomplished. An approach to the measurement of this index consists of keeping a log where the subjective frustrations of the user are tabulated, so that attitudinal trends may be determined [BOR79].

- (2) Ease of Use Human throughput can be greatly affected by the ease of use of the system. This index can be evaluated by measuring: a) effort required to prepare a problem for processing, b) effort required to operate the system [BOR79], and c) user's learning curve to operate the system.
- (3) Support from the manufacturer/extra features The support from the manufacturer and the number of extra features that are incorporated into the software system reveals the overall quality and maturity of a given VHDL software system. Some of the primary indicators of this index include: a) completeness and quality of documentation, b) library support (models of semi- custom and catalog components), c) compatibility with software from other standards, d) number of platforms on which it runs, e) compatibility with utility programs, pre- and post-processors, f) windows and graphics capabilities, g) operating modes: incremental compilation, save/restart, h) other capabilities, such as image patching, interactive simulation/debugging.
- (4) Response to Functionality Tests A well-designed simulator must satisfactorily confront tests that are specifically designed to uncover flaws and bugs in its implementation. The tests, known as functionality tests, can be extremely difficult to develop and to apply; they are used to determine if a simulator has a capability, or if it yields correct results.
- (5) Response to Simulator Implementation Tests Simulator implementation tests are designed to check and validate the simulator's conformance to the VHDL language. In reality, these tests belong to the category of analyzer tests, because they check that the system will report syntactic or semantic errors in a VHDL model. The tests must also verify that the VHDL system under test only simulates source code that conforms to the syntactic and semantic rules of the language.

The evaluation methodology that was developed in this study is based primarily on the execution CPU (user) time performance index (Quantitative index number 1).

BENCHMARKS AS VEHICLES FOR THE EVALUATION OF SIMULATORS

Within the scope of VHDL simulators, a benchmark is a program (or hardware description) that is designed or selected for the purpose of performance comparison or measurement of VHDL simulators. The benchmarks that are selected for the evaluation of VHDL simulators must exercise all the VHDL language's functions, statements, instructions, and language features in a manner in which these are used or expected to be used in real applications. There are two categories of benchmarks: natural (or real), and synthetic [BOR79][CLA86][CUR76][FER78]. A natural benchmark consists of a program or set of programs that have been used in real applications; they are selected for the testing of a simulator because they are representative of typical applications and workload. A synthetic benchmark, on the other hand, consists of models that have been coded specifically for testing purposes. They need not achieve any useful function other than the evaluation of a simulator. In fact, many of them do not represent hardware at all. Synthetic benchmarks are usually designed having as a principal objective the testing of a particular instruction or set of closely related instructions. In this way, the whole evaluation process is partitioned into different classes.

There are two varieties of synthetic benchmarks: simple [CLA86] (or atomic), and composite. The first type is designed to test the execution of a simple language feature. Simple benchmarks may be used for performance projection studies or to uncover constructs that result in poor performance. A composite synthetic benchmark is a test model that usually includes only a selected set of language constructs. This type of benchmark may be comparable to a natural benchmark, but because it was coded for testing purposes, it is considered to be a synthetic benchmark instead. Composite synthetic benchmarks may be further subdivided into two groups: specific and general. The first type is usually smaller (less lines of code) than the second one. Models categorized under the label of specific benchmarks are assembled around a specific language construct, capability, parameter, or feature of interest. General synthetic benchmarks, on the other hand, are descriptions of hardware that include a number of different statements and constructs; they are designed to evaluate overall simulator performance during the execution of statements and constructs that they contain.

Benchmark Requirements

Before implementing benchmarks, it is important to understand some of their requirements and important characteristics in order to facilitate their efficient design. Brief descriptions of their requirements are the following. Not all of these requirements may be met in a single benchmark. Those requirements which must be met in all benchmarks are indicated by an asterisk (*).

- (1) Representativeness Benchmarks should represent accurately the characteristics of real applications.
- (2) Design Domains Benchmarks should cover a wide range of design domains so that the evaluator can choose the benchmarks that are more representative of his own applications.
- (3) Flexibility and Stress Testing Benchmarks should be constructed in such a way that they can be easily modified to stress the system along a number of dimensions, such as the array size, number of: nested elements, gates, process statements, etc.
- (4) Verifiability (*) It should be easy to verify that the computations and results of the simulation are correct.
- (5) Portability (*) The generated models should be completely portable/compatible with the different simulators under test and for different platforms.
- (6) Ease of Use (*) The application of the benchmarks should be accomplished with minimum effort and in minimum time.
- (7) Consistency (*) The benchmarks should be error free; in addition, the benchmark and the evaluation scheme should be designed to provide consistent and repeatable results.
- (8) Compactness Benchmarks should be representative of large applications, but at the same time they should be compact.

Considerations in the Use of Benchmarks

As with any testing tool, benchmarks have limitations that should be considered if misleading results are to be avoided. Some of the most important limitations are the following.

- (1) Test results are only representative of the types of models that are included in the benchmark set. Generalizations should be avoided.
- (2) The development and application of benchmarks is a trade off; as the number of models included in the benchmark set increases, the accuracy of the results is likely to increase as well, but the time, effort, and costs of development and application also increase.
- (3) It is important to understand the objective of each benchmark. Otherwise, the benchmark will be misapplied, and the wrong quantity will be measured.
- (4) An effort must be made to understand the many external factors to the simulator that may affect the benchmark's execution results.

SIMULATOR COMPARISON

The evaluation of two VHDL simulators under a set of conditions that could yield results considered to be completely impartial is a rare event. Rather, most of the time a number of factors interfere with the impartiality and accuracy of the evaluation process. These factors can be classified as either internal or external to the simulator. Internal factors depend on conditions that can be set within the simulator. External factors, on the other hand, depend on characteristics and circumstances that affect the computer systems where the simulators execute. They may affect the performance of the system in a rather random fashion. The external factors to the simulators can be subdivided into two categories: inter-machine differences in hardware and/or operating system (when the simulators run on different machines), and variables intrinsic to the computer system where the simulators execute.

It is beyond the scope of this paper to discuss in detail all of the factors that can be considered (observed and/or controlled) in the evaluation of simulators. A good discussion of many of these factors can be found in [ALT87] and [LEV91]. Here, we only mention some of the strategies that can be used to minimize their effects. The general philosophy behind many of these strategies is to provide each simulator with the opportunity to show its best performance. This philosophy was adopted mainly for two reasons. First, this is perhaps the only absolute point of comparison that can be established. And second, there are no "typical conditions" for an evaluation. That is, the conditions that a user sets for his applications may benefit some simulators more than others. The strategies that we propose are the following.

First, the use of any forms of interactive simulation modes or graphical front-ends should be discarded. The machine where a test is being run should only have its console active (e.g., no windows system should be running); all instructions to the simulator should be entered from the command line. Second, all simulator options should be set in such a way as to allow each simulator to show its best performance. The number of

signals to be traced should be kept to a minimum (usually, only primary inputs and primary outputs). The time limit should be set to a large number to allow the simulation to complete; also, if applicable, the minimum time increment option should be set to a value that optimizes the simulation. Third, if possible, only the compiler type and version recommended by each simulator manufacturer should be used (if this is not feasible, then all simulators should use the same operating system). Except for the execution of simple benchmarks (discussed in the next section), all compiler options should be set to optimize the simulation. Fourth, the machine that hosts a simulator that is being tested should be in a steady state (e.g., running a minimum number of system's processes, without any additional user processes). Also, all non-dispensable daemons should be deactivated. Fifth, if the simulators being evaluated cannot run in the same machine, then the differences between these machines should be carefully considered. The results of each simulator test may be normalized by factors that describe the power of the machine where the simulator is executing. Sixth, whenever possible, all benchmark experiments should be run more than once. Only the execution CPU time is considered. Usually, the mean time of a set of trials is recorded; however, if very few values are available for each benchmark, the mean can be replaced by the best time (optimistic approach). Finally, all simulators should be evaluated using a minimum number of computer systems.

It is acknowledged that some of the strategies described above may not produce by themselves significant improvements in the accuracy of the evaluation of simulators. Nevertheless, we have found that the observance of all the above strategies greatly improves the accuracy of the overall evaluation process.

THE STRUCTURE OF BENCHMARKS

The evaluation scheme that was developed uses many small, medium, and large size benchmarks. The approach is very flexible, and allows the evaluation of simulators from either a general perspective or from the point of view of particular applications. The scheme is accomplished by the application of simple and composite synthetic benchmarks. These are described in the following paragraphs.

Simple Synthetic Benchmarks

A simple synthetic benchmark is a model that allows the precise measurement of the execution times of individual language constructs. Simple synthetic benchmarks are useful in identifying the language features that are likely to cause bottlenecks during the simulation of a model. Also, the detailed knowledge of the performance of individual constructs is valuable during the planning phase of a model, because constructs with relatively long execution times can be avoided. The application of simple synthetic benchmarks is also valuable in verifying the functionality of the simulator, because the models developed for this purpose do not represent hardware and usually present uncommon paradigms and structures that stress the simulators to a certain degree.

The structure of the simple synthetic benchmarks (atomic benchmark) that were developed is a modification of the dual loop technique [BAS85][CLA86] that has successfully been used to measure the execution time of Ada language features. The construction of a simple benchmark requires the development of two nearly identical programs: a test, and a control program. The test program contains two processes. The language feature of interest is included in one of the processes, which is exercised a number of times (n) during a given simulation experiment. The other process is used to control the number of times (n) that the process containing the language feature of interest is to be executed. The only difference between the test and control programs is that in the latter program, blank lines are substituted for the language feature of interest, and the NULL function (control program) is executed n times. Therefore, the control program is used to evaluate the overhead involved in executing all the statements that have to be added around the feature of interest in the test program. The execution time of a language construct is then found (for a given experiment) by first subtracting the time of n executions of the control program from the corresponding time of n executions of the test program, and then dividing this result by the number n . Figure 1 presents the VHDL framework for a test or control program. In this scheme, the language feature that is to be measured and the values of I and N are contained inside a separate package `test_pkg.vhdl`. This framework is required to reduce the possibility of compiler optimizations that may invalidate the intent of the benchmark and the accuracy of the result.

The selection of the language constructs to be evaluated with the use of simple synthetic benchmarks is an extremely difficult matter. Obviously, the main problem is that there exists an infinite number of combinations of language features. Ideally, only those constructs that are used the most should be evaluated; this sounds simple, but identifying the most used constructs is a problem by itself, since to this date no comprehensive statistical data exists regarding the frequency of use of VHDL constructs.

```

use work.test_pkg.all;           -- package containing variables and language
                                -- construct to be measured

(test_pkg.vhdl)
entity test is
end test;

architecture ar_test of test is

signal I: integer := index_var;  -- signal that triggers process
                                -- w/language feature

begin

Process_loop:                   -- process that generates signal
process(I)

variable N: integer:=iteration_var; -- number of iterations
                                -- called with a function

begin

if I /= N then                  -- if haven't reached top index, increment
    increment(I);               -- signal by one (procedure defined in test_pkg.vhdl)
end if;

end process;

Test_Process1:                  -- process that calls a procedure where language
                                -- feature is located
process(I)                       -- process is triggered when there's an event on I
                                -- variable declarations here if required

begin

    procedure_with_language_feature; -- may contain zero or more arguments
                                    -- dummy procedures/functions here if required

end process;

end ar_test;

```

Figure 1. A Typical Implementation of a Test or Control Program

Based on the analysis of statistics of some high-level computer languages [ALE75][DAT90][DIT80][ROB76][WEI84] and our own experience, we developed a set of 162 simple synthetic benchmarks to test a variety of categories of language features (Table 1 presents these categories). Included in this set are benchmarks to measure the execution time of assignment operations, arithmetic and logic operators (8 and 6 operators respectively), conditional control constructs (i.e., CASE and IF statements), file operations (write/read different objects), argument passing in functions and procedures, and loop structures (i.e., FOR-LOOP-END, LOOP-EXIT-END, WHILE-LOOP-END). Most of these categories involve the construction of benchmarks in multiple versions, using objects of different classes and data types.

Composite Synthetic Benchmarks

Composite synthetic benchmarks are models that contain from a medium to a large number of language constructs. Their purpose is to yield information regarding the execution and interaction of a set of language constructs. It is advantageous (but not always feasible) to limit and to carefully control the scope of the interactions. A model with few interactions is more suitable for projecting the effect of a particular construct, but the probable simplicity of the model may yield results that are not representative of real applications. A large model with multiple interactions, although more likely to represent real applications, will probably disguise the effects of multiple language constructs with results difficult to analyze. Since it is impossible to reconcile the discrepancies between models with few and many interactions, two types of composite benchmarks, specific and general, were developed to cover both cases.

Specific benchmarks are usually simple in structure, and may or may not represent hardware. They are designed to test an explicit language construct, and because of this, each specific benchmark must be as

assignment of constant type (integer, real, bit, bit_vector) to signal	assignment of constant type (integer, real, bit, bit_vector) to variable
assignment of signal type (integer, real, bit, bit_vector) to signal	assignment of variable type (integer, real, bit, bit_vector) to variable
arithmetic oper (+, -, *, /, mod, rem, **, abs) on signals type (integer, real)	arithmetic oper (+, -, *, /, mod, rem, **, abs) on variables type (integer, real)
logical oper (NOT, AND, OR, NAND, NOR, XOR) on signals (integer, real)	logical oper (NOT, AND, OR, NAND, NOR, XOR) on variables (int., real)
case statements with (2, 6) branches	if statements with (1, 2, 6) branches
write objects type (integer, real, bit, bit_vector, character, string) to file	read objects type (integer, real, bit, bit_vector, character, string) to file
function passing (0-5) arguments class signal type (integer, real, bit, bit_v)	function passing (0-5) arguments class variable type (integer, real, bit, bit_v)
procedure passing (1-3) arguments class (signal, variable) integer inout	procedure passing (1-3) arguments class (constant, signal) integer (in, out)
procedure passing (1-3) arguments class (constant, variable) integer (in, out)	construct for-loop-end
construct loop-exit-end	construct while-loop-end

Table 1. Categories of Language Features for the 162 Simple Synthetic Benchmarks Developed

independent of the others as possible. In contrast, a general benchmark always represents hardware. Rather than dueling with details, general benchmarks test overall performance. Both specific and general benchmarks share similarities in their construction. For example, most composite benchmarks contain a cycling stimuli mechanism that is used to extend the length of the execution time of the benchmark and to add flexibility to the evaluation process. Also, most generic parameters susceptible to modifications for testing purposes are contained in the topmost entity, so that recompilation after a modification is kept to a minimum.

Most of the time it is not useful enough to obtain a single result that represents the execution time of a given model. Rather, several results may be required to highlight the effects of certain language constructs. For this purpose, two techniques are used: Parameter Variation and Multiple Versions. Frequently, the characteristics of interest in a model can be varied in extension (e.g., timing, range of subtypes, size of array objects) or in the number of times they appear or are being called (i.e., recursion levels, instantiated subcomponents). In the parameter variation technique, the characteristics of interest that can be varied are assembled in a list of parameters that is placed in the topmost entity. By assigning different values to specific parameters for each experiment, the characteristics of interest can be evaluated by observing their effects on the execution time of the model. The multiple version technique, on the other hand, involves the construction of a number of versions of a given model (e.g., structural vs. behavioral versions). The only requisite among the versions is that, to the extent possible, they should have the same functionality (i.e., for the same inputs, they should produce the same outputs and with the same time stamps).

Twelve common characteristics of hardware descriptions guided the development of composite synthetic benchmarks. These characteristics are: number of inputs, number of sequential input signals, number of gates, number of nested elements, array size, generate statements, design domains, levels of recursion, delay type, generics, sensitivity list vs. explicit wait statements, type and subtype declarations. A total of 58 benchmarks were developed to test the effects of the above characteristics. Among these, various versions of models represent hardware (an arithmetic logic unit, a parity checker, and a parallel full adder); other models are specific synthetic benchmarks for the testing of some of the characteristics described above.

RESULTS

Simple and composite synthetic benchmarks were applied to three VHDL simulators. The following considerations are presented to explain how we interpreted the data. Note that the purpose of the study was to determine an approach to measuring the performance of VHDL simulators that is more precise than heretofore utilized. It was not our purpose to compare and report on specific simulators. Therefore, the simulators measured are not identified.

Platform Considerations

The simulators measured are designated as Sim A, Sim B, and Sim C. Sim A ran on a Sun 3/60, while Sim B and Sim C ran, respectively, on a Sun SparcStation 2 and a Sun SparcStation 1. Some of the characteristics and configurations of these machines are presented in Table 2.

At least two experiments were conducted for each benchmark in accordance to the requirements outlined (many thousands of iterations on the construct(s) of interest were performed in each experiment). In all cases, the execution CPU times of the benchmarks were obtained by using the UNIX command *time*.

None of the results were normalized to account for the differences in the simulators' platforms. This fact is particularly important when evaluating the results from simulator A. Based on only the non-normalized data, one may conclude that simulator A and C are no match for simulator B (See Table 3 and 4). This is not necessarily true, because system A ran on the slowest machine among the ones that were used while system B ran on the fastest one. What we really want to know is which simulator is intrinsically the fastest. A viable but approximate solution to this question involves the determination of a figure of merit for each of the machines that were used. It is beyond the scope of this paper to perform a detailed analysis to come to the figures of merit for the machines that were used. Here, we only give proportions of values based on the data presented in Table 2. For simulators B and C, the primary factors to consider are the MIPS, SPECmarks, and memory sizes. For these platforms, the major variance is probably due to the MIPS rate; in terms of this table, the platform for simulator B appears to be faster by a factor of more than 2. In terms of memory size, the platform for simulator B has a memory four times the size of the corresponding memory for simulator C, which could induce more page faults when benchmarks are executed on simulator C. The same reasoning applies for the platform of simulator A compared to the platforms of B and C. The MIPS ratio that exists between platforms of simulators A and B is close to 9:1, while the memory ratio is 8:1. In addition, the small cache size for the platform of simulator A may lead to more cache misses. Note that the study was performed during the early part of 1991. Since that time many VHDL simulators have been vastly improved.

	Sun 3/60	SparcStation 1	SparcStation 2
Processor	MC68020	SPARC/RISC	SPARC/RISC
Clock Frequency	20 MHz	25 MHz	40 MHz
MIPS	3	12	28.5
SPECmark	-	11.8	20.6
Memory Size	8 MB	16 MB	64 MB
Cache Size	24 KB	64 KB	64 KB
Disk Capacity	1 GB	211 MB	2 disks: 669 MB, 207 MB
Operating System	SunOS 4.1.1	SunOS 4.1.1	SunOS 4.1.1
Platform for Simulator:	A	C	B

Table 2. Characteristics and Configurations of the Platforms Used

Rational for the Choice of Data Points

As described in the previous section, at least two experiments were performed for all benchmarks (some had three or more experiments run). The main reason for establishing such a number was a practical one. Although for each experiment (i.e., the execution of a benchmark) one would like to have a large number of results (so that a representative value of the sample can be determined after the statistical characteristics of the data set have been established), it may be impractical to repeat each experiment a large number of times, especially when such experiments have to be run on more than one simulator. If only two execution time values are available for each benchmark, and if these values differ, the evaluator has three choices regarding the value to report as the result of a benchmark's execution time: the minimum value (optimistic), the average of the two values, or the maximum value (pessimistic). In this study, each benchmark that was executed on a given simulator produced either the same result or a very close result. Since these values were often very close, only the minimum value was selected. Nevertheless, it is important to note that some undetermined error exists in the results that are presented.

In order to evaluate if the minimum value is representative of the typical execution time of a benchmark, the following experiment was conducted. First, one of the slowest and one of the fastest simple synthetic benchmarks were selected, and 20 experiments executed on each on a given simulator. Then, from the results obtained, the mean and standard deviation were determined. Next, the entire experiment was repeated, and the new mean and standard deviations were calculated. In addition, the overall mean and

standard deviation for each series of experiments (40 results) was determined. Finally, the results from the original experiments that were used to determine the values that are included in the tables were compared to the newly obtained results. From these data, it was observed that the minimum value was a good choice.

For each of the controlled experiments, an error may exist due to the resolution of the measuring device. Obviously, the difference in execution times of test and control programs can be increased (and the corresponding error can be reduced) by increasing the iteration index in both control and test programs. From the results of the experiments described above, it appears reasonable to consider that the results of these tests follow a normal distribution. Increasing the number of iterations also reduces the likelihood of overlapping in the results of test and control programs, which could result in a negative value upon subtraction, due to their relative positions within the curves of their distributions.

The Benchmark Data

The experiments, benchmark code and results corresponding to the application of simple synthetic benchmarks and composite synthetic benchmarks are presented in [LEV91], and will be supplied upon request. Sample data are given in Table 3 and 4 to illustrate the kind of results obtained.

CONCLUSIONS

Three simulators have been evaluated. Each of them runs in a different model of a Sun workstation. The differences in the machines used preclude a direct comparison of the simulators. The execution CPU (user) time was obtained in all cases with the *time* command, which for most machines have very limited accuracy. The iteration index of a simple synthetic benchmark should be selected in such a way that the difference in execution time of a given test and a control program is large, so as to reduce the error due to the limited resolution of the measuring method.

A number of interesting trends regarding the execution of certain language constructs in different simulators can be found by using the benchmarks we have developed. Some simulators appear to execute certain

Conditional Statements			
Simple Synthetic Benchmark	Execution Time (ms)		
	Sim A	Sim B	Sim C
Case test with two branches	0.00614	0.00083	0.00950
Case test with six branches	0.00735	0.00500	0.01000
If-then-end	0.00757	0.00716	0.00433
If-then-else-end	0.00671	0.00466	0.00433
If-then-else-if-else-if-else-if-else-if-else-end	0.01200	0.00983	0.00533
Minimum	0.00614	0.00083	0.00433
Mean	0.00795	0.00549	0.00669
Maximum	0.01200	0.00983	0.01000

Table 3. Simple Synthetic Benchmarks - Sample Results

Effects of Delay Type			
	Execution Time (sec)		
	Sim A	Sim B	Sim C
Benchmark 8.1 <specific> (delta delay)	525.1	27.9	98.7
Benchmark 8.2 <specific> (inertial delay)	562.4	28.7	96.5
Benchmark 8.3 <specific> (transport delay)	554.4	28.7	96.9
Benchmark 3.1 <general> (inertial delay)	457.4	28.8	78.4
Benchmark 3.4 <general> (delta delay)	454.0	29.1	77.2
Benchmark 3.5 <general> (transport delay)	504.3	32.0	84.1

B 8.1: Mechanism to exercise a 2-input AND gate that uses delta (default) delay.
 B 8.2: Mechanism to exercise a 2-input AND gate that uses inertial delay.
 B 8.3: Mechanism to exercise a 2-input AND gate that uses transport delay.
 B 3.1: ALU structural (SN74181). All gates use inertial delay.
 B 3.4: ALU structural (SN74181). All gates use delta delay.
 B 3.5: ALU structural (SN74181). All gates use transport delay.

Table 4. Composite Synthetic Benchmarks - Sample Results

language constructs better than others. These discrepancies are likely to be the result of differences in the implementation of the simulators. Knowledge of these trends is beneficial for a number of reasons. When working with a single simulator, an evaluator can choose the design style or the constructs that are optimized in that simulator. Conversely, when the designer has available a number of simulators, he can choose the one most suitable for his applications. The scheme that we have developed may be applied on any hardware platform for any simulator that runs on that platform without any special hardware or software aids. This represents an improvement over techniques that have been used in the past, where the benchmarks are only useful for simulators that can only run on a given platform.

REFERENCES

- [ALE75] W. G. Alexander, D.B. Wortman, "Static and Dynamic Characteristics of XPL Programs", *Computer*, November 1975, pp. 41-46.
- [ALT87] N. Altman, "Timing Variation in Dual Loop Benchmarks", Technical Report CMU/SEI-87-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburg, PA, October, 1987.
- [ARM89] J. Armstrong, C. Cho, S. Shah, The VHDL Validation Suite Test Development Manual, Department of Electrical Engineering, Virginia Institute of Technology, March, 1989.
- [BAS85] M. J. Bassman, G. A. Fisher, A. Gargaro, "An approach for evaluating the performance of ADA compilers", ADA in use, Proc. of the ADA International Conference, May 1985.
- [BOR79] I. Borovits, S. Neumann, *Computer Systems Performance Evaluation*. Lexington Books, Lexington, MA, 1979.
- [CAR91] H. Carter, et al, "High Speed Acceleration of VHDL Simulation, Synthesis, and ATPG: Overview of the QUEST Project", Using VHDL for Electronic Product Design, VHDL Users' Group Spring 1991 Conference, Cincinnati, Ohio, April 1991, pp. 85-90.
- [CLA86] R. M. Clapp, L. Duchesneau, R. A. Voltz, T. N. Mudge, T. Schultze, "Toward Real-Time Performance Benchmarks for Ada", *Communications of the ACM*, Vol. 29, No. 8, 1986, pp. 760-778.
- [CUR76] H. J. Curnow, B. A. Wichmann, "A Synthetic Benchmark", *Computer Journal*, Vol. 19, No. 1, 1976, pp. 43-49.
- [DAT90] S. Datta, P. A. Wilsey, "Static and Dynamic Characteristics of VHDL Programs", First Annual QUEST Review, University of Cincinnati, June 1990.
- [DIT80] D.R. Ditzel, "Program Measurements on a High-Level Language Computer", *Computer*, August 1980, pp. 62-72.
- [FER78] D. Ferrari, *Computer Systems Performance Evaluation*. Prentice-Hall, Englewood Cliffs, NJ 1978, 554 pp.
- [GRE87] D.L. Greer, "The Quick Simulation Benchmark", *VLSI System Journal*, November 1987, pp. 40-57.
- [LEV91] D. Levine, "Criteria for the Evaluation of VHDL Simulators", Master's Thesis, University of Virginia, August 1991.
- [LOU88] M. Loughzail, M. Cote, M. Aboulhamid, E. Cerny, "Experience with the VHDL Environment", Proc. 25th ACM/IEEE Design Automation Conference, 1988, pp. 28-33.
- [ROB76] S. K. Robinson, I. S. Torsun, "An empirical analysis of FORTRAN programs", *The Computer Journal*, Vol. 19, No. 1, 1976, pp. 56-62.
- [SER89] K. Serafino, VHSIC Hardware Description Language (VHDL) Benchmark Suite, Technical Report WRDC-TR-89-5046, Design Branch, Microelectronic Division, Wright Research and Development Center Air Force Systems Command, Wright-Patterson Air Force Base, Ohio, October 1989.
- [WEI84] R. P. Weicker, "Dhrystone: A Synthetic Systems Programming Benchmark", *Communications of the ACM*, Vol. 27, No. 10, October 1984, pp. 1013-1030.