Toward the Development of a VHDL Simulator Performance Metric

C Heine, D Nasoff, J Russo, J Stavash
Martin Marietta Laboratories - Moorestown
Building 145-2, Route 38
Moorestown, NJ 08057
(609)866-6603
{cheine,dnasoff,jrusso,jstavash}@atl.ge.com

Abstract

A procedure is described for parametrically characterizing the run-time performance of VHDL simulation tools and accelerators. An initial suite of three parameterizable models is proposed as a strawman toward the development of a standard benchmarking suite. Similar efforts have been proposed, and are in-progress, for developing a validation suite for VHDL simulators. This effort attempts to measure performance only; not compliance with the 1076 LRM.

The relative performance of the various VHDL simulators is not well understood. Vendors and users currently compare simulators based on individual benchmarks. However, since simulator performance depends greatly upon the nature of the models that are simulated, potential users cannot reliably predict the turn-around times they will experience in exercising their models.

The successful scheduling of large design projects, such as for RASSP [1], requires accurate estimates of simulation turn-around time to determine the level of modeling that can reasonably be accomplished. Since a simulation run could consume from minutes up to weeks, accurate prediction of the completion time prior to execution becomes advantageous.

A suite of three parameterizable models is used to explore the performance space. Measurements of analysis and simulation memory-consumption and run-time are collected from the test case models over various conditions. The test cases contain eleven attributes which provide for linear and independent variation of the modeling conditions.

Parametric coefficients are then estimated through the use of a curve-fitting technique that fits a curve to the measured observations. Inspection of a simulator's resultant performance coefficients provides a concise understanding of a simulator's comparative strengths and weaknesses.

An example of the application of the parametric expression is demonstrated for estimating the run-time of an arbitrary design. The accuracy of the resultant prediction should be tested by applying it to cases that were not used in collecting the curve-fitting data.

The technique can be applied to characterize and compare VHDL simulators. It should serve as an aid to improving the performance of existing tools over time and to understanding the impact of modeling abstraction levels, and as a guide to developing efficient modeling styles.
Need for Simulation Performance Characterization

- Increasing Use of Simulation-Based Design Methodologies.
- Advanced Parallel Processing System Design Depends on:
  - HW/SW Co-Design,
  - Virtual System Prototyping.
  - Top-Down Design Methodologies ==> Multiple Abstraction Levels.
- VHDL Simulation Becoming Critical Path for Advanced System Designs.
- Concurrent Engineering Requires Predictable Schedules.
- Relative Performance of VHDL Simulators Not Well Understood.

Simulator performance can become the critical path to design cycle time for future large complex projects, especially those requiring parallel processing systems.

Though several VHDL simulators have reputations within our industry of being slow or fast, there is little understanding as to the how their performance compares quantitatively over various conditions.

For instance, some simulators are interpreted while others are compiled. While compilation is advantageous for long simulation runs, it could result in a longer modify-run-cycle time for short simulations.

Likewise, some simulators are optimized for gate-level simulations, while others claim to provide performance gains at more abstract levels.

A definitive, quantifiable, repeatable, accurate, reliable measurement is needed that supports comparison.
Challenges for Current Methods

- Simulator Evaluation \(\Rightarrow\) Complicated, Time-consuming, & Costly.
- Evaluator's Criteria & Application \(\neq\) User's.
- Benchmark Reports Useful Only as Guide.
- User's Perform Their Own Comparisons.
- Vendors Often Avoid Benchmark Characterizations.

Performing product evaluations on VHDL simulators and accelerators is complicated, time-consuming, and costly.

Each evaluator’s requirements and usage inevitably differ from a given user's applications.

Users can use published benchmark results merely as a guide, and then embark upon performing their own comparisons.

CAD companies sometimes complain about how difficult it is for a customer to benchmark their tool and see benchmarking as an evil to be avoided. Vendors claim the user’s results depend on the user’s expertise with their tool or system. If the user performs a poor assessment, then the vendor suffers from a bad benchmark. Vendor provided benchmarks are often optimized for their software.

An independent source of benchmarks is needed.
Previous Efforts

- Many Previous Published Benchmark Reports [3,4,5,6].
- Disjoint, Good, But Never Repeated Methods.
- Broad in Scope, Contained Subjective Aspects.
- Limited Number of Simulators Compared.
- Examples Specific to Small Range of Design Abstractions.

There have been many benchmarks published in the literature [2,3,4,5,6].

Many good test sets have been developed, but they tend to seldom be re-used by other investigators. As a whole, the individual efforts appear disjoint.

Most evaluations attempt to offer broad, well rounded, product characterizations inevitably containing subjective or qualitative aspects, such as, ease of use, support, or productivity.

Because each independent benchmark report compares only a subset of simulators, direct comparison of simulators not benchmarked in a common report is not possible. Also, the measurements are often taken on disparate platforms, resulting in apples and oranges comparisons.

The reported evaluations developed unique criteria, procedures, and benchmark models, so there was very little automation of the measurement process. The model source is an unknown variable, because it is sometimes optimized for specific simulators.

The example application models for a given benchmark report tend to lie within a narrow descriptive paradigm, such as the logic level only.
Scope of this Investigation

- There are Many Aspects to Comprehensive VHDL Product Evaluation.
  - 1076 Compliance
  - Technical Support
  - Cost
  - Design and Debugging Environment
  - Ease of Use
  - GUI Quality
  - Performance

- Present Focus is Performance Aspects Only.

- Only Minimal Simulator Compliance (ANSI/IEEE Std. 1076) Issues will be Tested.

- A Complete Benchmark is Beyond the Scope of this Paper, But a Strawman Approach is Proposed for Further Development.

VHDL products should be rated over a balanced and broad set of criteria, such as adherence to the LRM, technical support, cost, ease of use, GUI quality, the design and debugging environment, and performance. This paper focuses on the performance aspect only. Well balanced multi-criteria comparative evaluation of several VHDL products have been published, such as [4].

Discrepancies with implementations of the 1076 standard will be uncovered, though this is not the primary purpose of our investigation. The discrepancies will be detected both at compile time and at run time. Compile time discrepancies will be detected by observing errant compiler error messages. Run time discrepancies will be detected in the form of results check, to ensure the measured VHDL code execution produces the same results on each simulator.

Development and demonstration of the complete benchmark suite was beyond the scope of this preliminary investigation. However, the proposed strawman approach sets the groundwork for how further development will proceed.
Goals for Performance Measurement

- Propose Canonical Test and Metrics.
- Independently Characterize Simulators and Application Models.
- Reliably Predict Run-Time Performance.
- Similar to MIPS or MFLOPS Rating Concept, where Users Predict Run-Time Performance Based on Processor Rating and Application Operation Count.
- Benchmark Should Be Stable and Scale with Technology For Meaningful Year-To-Year Comparisons.

Develop and establish a canonical set of tests and measures for independently characterizing both application models and candidate simulators that can accurately and reliably predict the performance you will observe/experience with your application.

Simulation and accelerators vendors or independent evaluators could publish tool performance specifications. Users will characterize their applications according to this set of parameters. Each user could then use their parameters together with the tool performance specifications to predict simulator or accelerator performance on their application.

The parameters measure performance aspects similar to rating computer power in terms of MIPS, MFLOPS, Whetstones, or Dhrystones measurements. Each measure is intended to measure a specific aspect of performance. Users use them to estimate processor performance on their application by characterizing their applications.

The benchmark should be stable and able to grow with technology to support performance comparisons across years.
Measurement Methods

- "Black-Box" Testing Versus Open Analysis of Internal Contents.

- Basic Approaches for black-box testing:
  1. Canonical test, atomic, or "synthetic",
  2. General model type using typical model code.

- Testing Hypothesis Based on Prior Observations.

- Simulators Often Optimized for Specific Model Types:
  (e.g. Data-Flow, RTL, Procedural).

- Measure time and Memory Utilization for Compile/Analysis, Elaboration, and Simulation time.

We selected a form of "black-box" testing, as opposed to open analysis of a simulator’s internal contents. In black-box testing, a simulator’s behavior is observed under various stimuli to deduce features of its internal implementation.

There are two basic methods for black-box testing. The first method uses a set of canonical tests to independently measure each of a set of distinct performance attributes. The attributes are considered to be “atomic” features, which may be combined to elicit the full behavior of a simulator. Such tests are also known as “synthetic” benchmarks, since each are especially devised to measure a specific aspect of performance. The second method uses a set of general VHDL models which are typical of various types of model code. The simulator characteristics are deduced by observing the relative performance on the various model types.

Admittedly, it is not possible to develop benchmark tests in a vacuum. We know that simulators are often specialized for specific modeling classes. For instance, simulators based on different event queueing mechanisms such as discrete event based versus cycle based simulation kernels, will exhibit unique performance signatures between the various logic level, data-flow, RTL, or behavioral model types. Additionally, simulators that compile VHDL directly to the native simulation host instruction set will exhibit different run-times than simulators that run in interpreter mode, or those that compile to an intermediate form such as C-code.

Therefore, we developed tests that are posed to differentiate suspected simulator implementation features that determine performance.

The time and memory consumed during compilation/analysis, elaboration, and simulation execution will be recorded. Where necessary, the elaboration time will be deduced from the start-up overhead of differing length simulation runs.
Independent Parameters

- Partition Performance Space Into a Multi-Axis Space.
- Performance as a function of 11 Independent Variables:
     Model Complexity in number of:
     2. entities,
     3. processes,
     4. states,
     5. ports,
     6. signals / bit-width, and
     7. lines of code.
  8. Signal Activity.
 10. Tracing Activity.
 11. The Simulation run length.

- Need Extensible Parameterized Models.
- The test model must have parametric control over the independent variables.

Attributes were selected that affect simulation performance, that therefore, taken collectively, are indicative of the expected simulation performance.

Eleven attributes were chosen for independent variation to measure simulation performance sensitivities.

The abstraction level, or modeling style, is the first parameter, and is explained more thoroughly below, in terms of the Ecker cube [7]. Parameters two through seven are measures of model complexity. Parameters eight through ten indicate signal-, file I/O-, and tracing-activity. While the remaining parameter controls the simulation run length.

To efficiently explore these parameters, extensible and parameterized models are needed as test cases. The test models must be easily modified to control the independent variables separately.
Simulation speed depends on the modeling level or the type of models used.

The Ecker cube [7] provides a convenient context for describing and referring to the various design description resolution levels as a prototype proceeds through a top-down design process from: system/algorithmic, through architecture, and down to the RTL and gate levels.

The cube contains three orthogonal aspects of a design: the modeling paradigm, the timing resolution, and the value resolution. A point in the Ecker space represents a model's description level. For instance, a system or algorithmic model would be represented by a point (abstract, behavior, causality), while a gate level model would correspond to a point (bit, structure, propagation).

It is important that simulator performance be characterized at each of the design process levels and throughout the design space by independently measuring the dependence on each axis.
Benchmark Models

- Parameter Variation Requirements:
  - Independent Parameter Modification
  - Automatic Modification
  - Span Design-Level Space

- Selected Model Suite:
  - LFSR - Linear Feedback Shift Register [2]
  - DP32 Processor [8]
  - Parallel RISC uProcessor

Ideally for simplicity, a benchmark model should be obtained with which any of the 11 parameters can be varied independently. In this way, the effect that each parameter is responsible for can be isolated.

The parameters of the benchmark model should also be fairly easy to change, so that many different cases can be run from an automatic process.

The parameterization should be able to cover the range of design levels that will occur in designs of interest.

An existing model could not be found that was easily parameterized to cover the range of interest for all of the parameters. Additionally, it was felt that the benchmark should contain multiple model types to obtain a more significant sample base across an ensemble. Therefore, three models have been selected for the test suite.

To build upon earlier efforts, the Linear Feedback Shift Register (LFSR) and DP32 models were obtained from the public domain in previously published literature [2,8]. These models have consequently been previously tested on a large variety of platforms. They were selected because they inherently covered some of the parameters of interest.

The Parallel RISC uProcessor model was developed to investigate regions of the parameter space that are not covered by the public domain models.

The models and their usage and parameterization are described in the subsequent pages.
The Linear Feedback Shift Register (LFSR) model [2] was intended for benchmarking simulators at the logic, switch, and Fault simulation levels. It is composed entirely of simple logic gates, and it can be instantiated with as many as 100,000 gates in a four level hierarchy.

The major advantage of this model is that it's complexity can be scaled linearly in terms of the number of gates. This can be used to control the number of entities and processes.

The architecture could be re-organized away from the serial shift register toward a parallel shift register of arbitrary width, so that the signal bit-width and number of ports per flip-flop could be varied.

The descriptive level can be changed to span from the gate up to the RTL and data flow levels.

However, the ability to describe this circuit at higher abstraction levels is very limited. Therefore, additional models which offer non-trivial abstract descriptions are needed.
The DP32 Model

- Hypothetical Processor
- Typical of Current 32-Processors
- Described at Behavioral and RTL Levels
- Could Synthesize Down to Gates
- Can Describe System Behavior Algorithmically
- Can Investigate File I/O Overhead at Memory Initialization
- Number of: States (ie. variables), Ports, and Signals Not Easily Varied.
- No Network Architecture Level Description

The DP32 processor model [8] is a model of a hypothetical processor which is typical of current 32-bit processors with regard to its instruction set and bus interface. The instruction set architecture is based on approximately 20 instructions and 256 general purpose registers. All instructions are multiples of 32 bits.

Descriptions of the processor are available at two levels: behavioral (ISA) and RTL. With some modification, the RTL model could be synthesized down to a gate level description. Since it is programmable, the behavior of the DP32 system can also be described algorithmically. As an example, one description of a finite impulse response (FIR) filter is a software program that controls the ISA or RTL level DP32 model to perform the FIR filter transform. Alternatively, the same system behavior is described by an algorithmic behavioral description that does not include processor implementation details.

There are many internal features that can be traced during simulation. Consequently, this model can be used to measure the effect of tracing activity. Similarly, the impact of file I/O can be measured at initialization time, by loading increasingly larger program memory segments.

The number of states (variables), ports, and signals is not easily varied with this model.

The DP32 model is not easily extendable to a network architecture level description without additional networking components.
Parallel-Risc uP

- Existing Models Do Not:
  Span All Levels or Cover All Parameters

- Proposed New Model:
  Parallel Processing System of RISC uProcessors

- RISC uProcessor:
  Minimum Complexity for Multiple Instantiation
  8 Instruction Set, Minimum of 16-Registers
  2 Inter-Processor Communication Ports
  Description Levels: Algorithmic, Network Architecture, ISA-Behavioral,
  Data-Flow, and Gate
  Scalable Data Path Width (16, 24, 32, 64, 128)
  Scalable Number of States (Register Set and/or Memory)
  Scalable Number of Entities (parallel processor (1-N))
  Scalable Number of Lines of Code

The existing models were not easily parameterizable for all the 11 parameters, nor were did they span all the design levels efficiently.

Therefore, a new model was proposed and developed. The new model is called the Parallel-RISC micro-processor. It is a model of a hypothetical parallel processing system composed of reduced instruction set computers (RISC) u-processors.

The RISC micro-processor has a very simple instruction set consisting of only 8-instructions. It has a minimum of 16-registers, though, the number of registers is indefinitely extendable as a function of a model parameter. Each RISC micro-processor contains two inter-processor communication ports to support network architecture level models.

A minimum complexity baseline processor element design was selected to facilitate practical instantiation of systems containing many multi-processors.

The modeled system can be described at the gate/structural, data-flow, RTL, ISA-behavioral, and network architecture levels. Since it is programmable, the behavior of the system can be described algorithmically for instance terms of the FIR filter algorithm.

The data path width can be selected as one of 16, 24, 32, 64, or 128 bits. This is used to control the number of bits per signal. The number of states or variables is controlled by the register set size and/or by the memory size. The number of significant entities is controlled by the number of processor elements instantiated from 1 up to N. The complexity of the behavioral model entity can be scaled upward by duplicating and unrolling the main instruction "case"-loop.
Parameterization Methodology:

- Many Parameters $\Rightarrow$ Need Efficient Method.
- For 11 Parameters, Need 12 Simulation Runs.
- Alter One (or more) Parameter(s) Per Run.
- Record Compile- and Run-Times and Memory Use, $y$.
- Solve $y = MX + b$ Simultaneous Equation.
- Search for Transition Points.
- Predict New Simulations by Applying Coefficients.

A simple method to characterize a simulator would be to run at least three two cases of each parameter value. Each case would vary the parameter independently. However, a more efficient method is needed because we have so many parameters. Also, some parameters cannot be changed completely independently, since they are not completely orthogonal.

Assuming linear dependency, the 11 parameters can be determined by a minimum of 12 simulation runs. On each run, a different parameter, $X_i$, is altered. Multiple parameters may be altered simultaneously, as long as the resulting parameters are not a linear combination of a previous run. In each case the dependent variable, $Y$, is recorded. The dependent variables are the compile, run-time, and peak memory consumption.

The dependency coefficient vector, $M$, is then computed by solving the $Y = MX + b$ equation, where $M$ is the transpose of the asymptotic dependence vector, $b$ is the fixed overhead, $Y$ is the time or memory quantity, and $X$ is the independent parameter vector.

Additional runs should be made to find any transition points. These are points where the dependence changes radically, due to cache or RAM limitations, or where the implementation breaks. The transition points provide an indication of storage efficiency, and/or internal limitations. The data from the extra runs can be factored into the solution above by a variety of curve fitting techniques.

Once a simulator is characterized, simulation compile and run-times ($T$) and memory utilizations for application models can be predicted prior to compilation by characterizing the application model's parameters, $Z$, and applying the coefficients solved above.

$$T = MZ + b$$
Conclusions

- Proposed Strawman Method for VHDL Performance Measurement.
- Will Allow Prediction of Model Execution Time.
- Needs Further Development.
- User and Vendor Group Develop Benchmark Suite.
- Lead to Standardization.

A strawman method for VHDL simulator performance measurement was proposed that provides the run-time predictions for arbitrary application models.

Further development is needed to test concept.

Then, a small diverse group of users and vendors should develop the draft benchmark suite for industry acceptance and standardization.
References


