A High Performance VHDL Simulator for Large Systems Design

Steve Hodgson, Zak Shaar and Andy Smith

Design Automation Centre, High Performance Systems,
ICL, Wenlock Way, Manchester, M12 5DR, U.K.

Abstract

The requirements of large system design place great demands upon the performance and diagnostic capabilities of simulation. This paper explains how these requirements have been satisfied by an internally-developed simulator using a proprietary language and a proprietary platform. More significantly, this paper describes how the strengths of that system have been developed to create a VHDL simulator under UNIX while maintaining a performance level that significantly exceeds the leading commercial VHDL simulators.

1. Introduction

The High Performance Systems division (HPS) of ICL in the UK is involved in the design and development of large systems such as mainframe computers, large UNIX servers and parallel systems. The design methodology adopted for these complex designs relies heavily on design tools which support the High Level Design stage of the development route [1] . The majority of the development time is spent at this design level. As more automatic tools, such as synthesis, speed up the latter development stages, it is at the High Level Design stage that design tools can have the biggest impact. This has been recognised within the commercial CAD industry by the introduction of the term “Electronic Systems Design Automation” (ESDA), but has been relevant in the large systems arena for many years.

The most important of all high level design verification techniques is simulation. Within HPS, simulation has been the key to many successful hardware developments resulting in a succession of “right-first-time” chips and computer systems. Simulation, using a schematic design tool (SDS) for structural description and a proprietary language (S3) as the behavioural description, was introduced in HPS over 12 years ago [2] . This used the ICL mainframe, with VME as the operating system, as the simulation environment. Over the following decade the simulator (called MSIM) was enhanced and optimised to maintain a competitive advantage to HPS over those companies using commercial simulators. The most important of all simulation features to the systems designer has been, and still is today, *simulation performance.*

2. Requirements for Large System Simulation

Modelling and simulating a large system can involve hundreds of thousands of lines of behavioural description, many millions of tests, numerous model configurations and very long runtimes. These considerations place particular demands on the core simulator:

a Simulation Performance

Because of the need to simulate as many test cases as possible, simulation performance takes priority over every other aspect of the simulator design. The quality of the final hardware depends greatly on the amount of simulation that can be performed during the design phases and so the efficiency of the simulator at each stage of the development route is paramount. Because of the size of the validation task, the simulation performance also takes priority over the performance of model compilation - in fact the compilation phase is extended to ensure that the simulation model is fully optimised prior to simulation.

b Diagnostic Capability

Finding the source of a design error within a large model can be very time consuming. It is essential that errors can be located easily and quickly. An interactive simulation environment, with the ability to apply “what-if” tests at any time and in any area of the model, provides the most efficient solution.

c Memory Utilisation

Memory use is also very important with large system models. To avoid swapping during simulation the memory allocation must be efficient, though this always has to be balanced against the (sometimes conflicting) strive for performance.

d Model Re-use

Checkpointing is essential and the user should be able to quickly reload past simulations to assist error detection. Also, a single configuration of the system may be the basis of many simulation tests and be used by different designers exercising different aspects of the system design.

e Design Iteration

Even at the higher levels of description, system models are inherently very large and compilation times lengthy. It is very important, therefore, that, having found a design error, there is a mechanism for fast re-compilation which does not require a full re-compilation of the whole system.

3. Benefits of System Simulation

Figure 1 illustrates the impact of simulation on the number of errors remaining in the design at the manufacturing and commissioning phase - these figures are based on actual experience in the development of large systems in HPS.

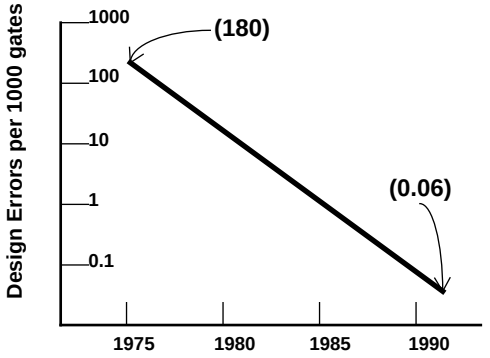

Figure 1: Improvement in Error Detection

In the early 1970s there was little system simulation and most design errors had to be found after the hardware had been built. Simulation became a key part of the development route in the late 1970s, but most simulation at this stage was at the gate level and, because of the poor simulation performance, relatively few tests could be completed prior to manufacture. In the 1980s, as the complexity of system design increased, higher levels of modelling were introduced, more powerful hardware was available to run the simulation and more effort was applied to the creation of simulation-specific tests.

The first half of the 1990s has seen that trend continue with more and more dependence on simulation with the resulting improvement in “right-first-time” chips, boards and systems.

Over the period from 1970 to today, simulation throughput, with the benefit of better hardware, algorithms and modelling techniques, has improved 10,000 times. The increased simulation capacity has helped error detection to improve 3,000 times. However, the size and complexity of the design task continues to grow - the requirement for faster simulation and better design methodologies is still paramount.

4. Conversion to VHDL

The emergence of VHDL and the development of new EDA tools have given rise to a new set of requirements in addition to those already mentioned:

- The ability to access external VHDL models;
- The ability to take advantage of alternative platforms for simulation (i.e. Workstations);
- The increased use of synthesis tools using VHDL as the source language;
- The increasing acceptance of VHDL as the standard Hardware Description Language.

These new requirements caused the Design Automation group in HPS to take on the task of moving the simulation environment from S3 and VME to VHDL and UNIX, while maintaining the very high simulation performance level of the previous system. Also, all of the above-mentioned requirements (interactive simulation environment, low memory usage, fast re-compilation, etc.) had to be satisfied.

There were numerous issues to be tackled:

a Language Incompatibility

The S3 language used for behavioural descriptions on VME was designed as a standard programming language. Hence it has no concept of “signals”, “delayed assignments”, “delta delays”, concurrency or sensitivity lists. Although some of these features had been emulated in the MSIM/S3 system they had to be re-designed to conform accurately to the VHDL standard - and this had to be achieved without impacting performance.

b VHDL Performance

We examined the VHDL language as a whole in great detail, to determine which constructs enabled efficient simulation and which impacted it. As a result we have produced a subset which is tailored for performance by elimination of inefficient VHDL, while maintaining support for most of the language. (see 5.3)

c Model Incompatibility

The processes of compilation and simulation model building had to be completely rewritten for UNIX; again, of course, without affecting the performance of the gener-

ated model. The lessons of the last decade have been carried over to create a VHDL-based model optimised for both simulation performance and memory utilisation.

d User Interface Standard

Coming from a mainframe environment to a workstation we needed to introduce a Motif GUI for the system.

The interactive simulation environment has been ported almost transparently, allowing designers to use the new system without any need to learn a new simulation control language. In addition, we also now have the capability to run VHDL test benches, allowing us, for the first time, to measure the MSIM performance against the best commercial products.

We have now successfully achieved the conversion to a high performance, UNIX-based VHDL simulator; detailed in the next section.

5. The MSIM Simulation System using VHDL

5.1 The Overall System

Figure 2 illustrates the overall structure of the MSIM simulation system:

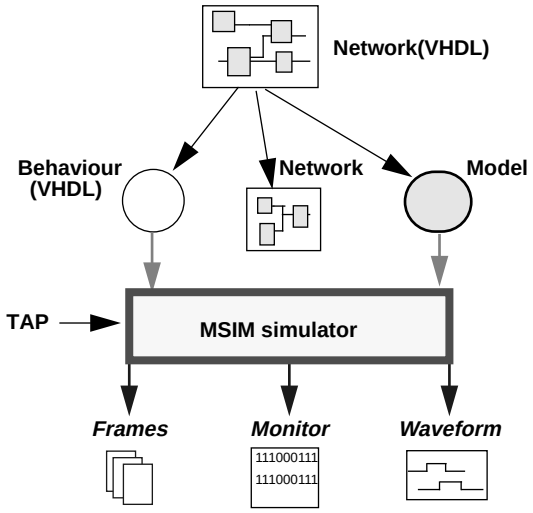

Figure 2: The MSIM Simulation System

The VHDL source is analysed using the VTIP Analyser [3] and is partitioned into Networks and Behaviours. The Networks describe the structure and hierarchy of the model in terms of instances and the connectivity between instances. The Behaviours describe the functional behaviour of each instance type. Behaviours are compiled into object code and the network description, or descriptions, are compiled into an internal structure ("the model").

5.2 Interactive Simulation Environment

TAP (Tests And Patterns) is the MSIM interactive simulation control language. It has been developed over more than 10 years to provide the full range of commands for interrogating, exercising and controlling the simulation model. There are 80 commands covering a whole range of facilities for the hardware designer.

The TAP environment provides an interactive editor which allows the user to build up a TAP programme within the simulation environment - executing individual commands or groups of commands dynamically, analysing the results of each execution, editing the TAP as necessary and finally saving the complete TAP programme for future re-use. This environment provides the capability of an "Interactive Test Harness" and can be used on any simulation model - large or small.

The key TAP facilities include:

- The ability to *Set* any signal value in the model and to *Compare* a signal value with a defined value or another signal.
- A set of user-defined *Pulse* shapes, which can be allocated to any single-bit signal. The *Clock* command then activates all or selected clock signals.
- The ability to quickly *Save* a model or *Restore* a previously saved model, which is vital when simulating large models.
- Any signal within the system can be interactively *Forced* to a value and that value then *Released*. This allows the user to partition the design and investigate *what/if* experiments on isolated areas.
- The *Declare* command provides a common aliasing feature - very important in large hierarchical models where signal names can be very long - for use in all TAP commands.

In addition, the model can be interrogated for details of model and event activity. This information can be used to highlight performance-critical areas of the model.

The TAP language has proved very popular with the hardware designers and is generally the method used for the initial testing of a new model. For more extensive testing a VHDL Test Harness is often used.

Frames, Monitor and Waveform are optional viewing windows which display signal and variable values during simulation. The Frame facility provides the user with a set of graphical objects which can be used to create a customised presentation of simulation results.

The simulation is controlled by a Simulation Control Panel which provides interrupt facilities, switching of viewing windows and model save/restore facilities.

5.3 The VHDL subset

The simulator does not currently support all of the VHDL language. The target use of the simulator is at the "pre-synthesis" stage and it does support 100% of the subsets supported by the leading synthesis products. However, the other target of the simulator is efficiency and so "simulator-inefficient" aspects of VHDL have been avoided - these include:

- some pre-defined attributes such as DELAYED, LAST_EVENT, LAST_ACTIVE, LAST_VALUE, QUIET.
- GUARDED signal assignment statements and blocks.
- User-defined attributes.
- User-defined resolution functions.

To further improve efficiency the following functionality is built-in to the simulator software:

- Arithmetic and Boolean operations on BIT and BIT_VECTOR types.
- Arithmetic and Boolean operations on the intrinsic types MBIT (a 4-value type (0,1,X,Z)) and MBITS (array of MBIT).
- The commonly used resolutions functions (wired-OR, wired-AND and Bus) for BIT, BIT_VECTOR, MBIT and MBITS.
- Conversion functions, including those between INTEGER, MBITS and BIT_VECTOR types.

Only a few, non-synthesisable, aspects of VHDL are unsupported. Overall the simulator supports >95% of the 1987 language syntax and there are plans to upgrade to the 1993 standard [4] .

5.4 The Simulator Kernel

5.4.1 Event Paths

The kernel structure is shown in Figure 3. The efficiency of the kernel is gained through the use of a number of parallel "event paths". Each event is allocated an "event type", according to the format of the event data, and takes a specific path through the kernel, customised and optimised for that data type. There is a single Event Manager which treats each event as a data/time item, but there are separate Event Processors for each of the Event Types. The role of an Event Processor is to update state values, propagate the values and form a list of Behaviour Instances to be called for this set of events.

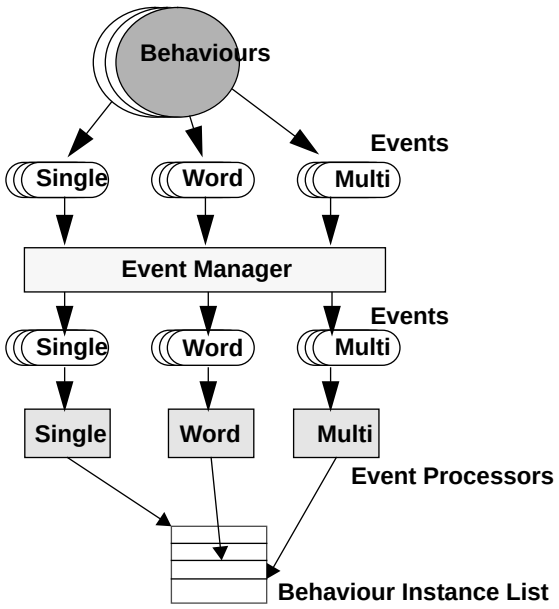

Figure 3. The MSIM Simulation Kernel

The different Event Types are:

Single2: This is used where the event data represents a single-bit, 2-value data item.

Single4: Single-bit, 4-value data item.

Single: Single-bit, multi-value data item.

Word2, Word4, Word: for multi-bit 2,4 and multi-value data items- but where the number of bits is 32 or less.

Multi2, Multi4, Multi: for Hmulti-bit 2,4 and multi-value events - but where the number of bits is greater than 32.

Composite2,Composite4, Composite: for all Record and multi-dimensional Array data items.

Time: Used for events which have no data - e.g. events created as a result of a WAIT FOR <time>.

Clock: A special event type used for signals which have been defined to the simulator (via TAP) as clock signals. These events are self-generating - as one clock is completed it generates another clock cycle on that signal.

5.4.2 The Event Manager

Since the introduction of event-driven simulation [5] a series of algorithms for event management have been implemented within the MSIM kernel. The types of mechanisms investigated include:

- a single time loop,
- a time loop with extension loops,
- multiple loops,

- a self-optimising loop system which altered according to the model.

Experiments have shown that, at least for the type of multi-level system simulation used in ICL, the single time loop has performance advantages over the others. It has also been found that keeping the size of the loop small gives optimum performance for models which are predominantly clocked designs.

The use of this type of event manager, together with the use of Event Paths, allows the simulator to be very efficient for clocked designs, but without being restricted, as with pure cycle-based simulators, to synchronous-only designs.

6. Simulation Performance

Two sets of benchmarks were used to demonstrate and compare the simulation performance of MSIM with commercial VHDL simulators. The performance of MSIM

Model	No. of Instances	Direct	MSIM
lfsr-1	1100	170	200
lfsr-2	100	62	40
lfsr-3	10	62	14
lfsr-4	1	60	10

Table 1: Performance for Different Levels of Description

was compared with a leading "direct-compiled" simulator. Other types of simulator were also benchmarked, but were generally slower.

Table 1 shows the results of four different architectures describing a 1000-bit LFSR. The first model (lfsr-1) is at gate level, the next describes 100 instances of a behavioural description of a 10-bit lfsr, the third 10 instances of a 100-bit lfsr behaviour and the last a single behaviour of a 1000-bit lfsr. Each run is for 10,000 clock cycles. Even this simple example demonstrates the effect of building high level features into the simulation kernel. The performance of MSIM continues to improve significantly as the level of description rises - taking advantage of the more abstract nature of the description.

The second set of VHDL data were collected from a variety of sources and each example attempts to exercise different aspects of VHDL:

Mixed-1 is a simple network of relatively small VHDL behaviours using multi-bit signals, arrays, functions, procedures and TextIO. The model is driven by two, independent, clocks.

Mixed-2 is part of a real system design again involving a mixture of structural VHDL and RTL descriptions. The whole design is synchronous with a single clock.

Behaviour represents a behavioural model of a microprocessor. This example is above RTL and makes good use of multi-bit values, arrays, functions, procedures etc.

Propagate is a purpose-made example for demonstrating simulation performance in the propagation of multi-bit, multivalue states. There are over 3000 instances of a very simple behaviour with complex multi-bit port-to-port connections.

Arithmetic is an example which performs many arithmetic operations and collects a "signature" of the results. The types used include INTEGER, BIT_VECTOR, SIGNED and UNSIGNED.

.

Model	Lines of VHDL	No. of Clocks	Direct	MSIM
Mixed-1	1000	30k	54	30
Mixed-2	3000	50k	365	130
Behaviour	1400	25k	115	10
Propagate	200	1k	1800	90
Arithmetic	150	100k	350	32

Table 2: Performance Comparison over a range of VHDL styles

This set of results highlights the importance of the MSIM optimisations for propagation of large state values and for arithmetic operations. Within a large system model, such operations will be common and without special handling can impact the overall simulation performance significantly.

NB: All simulation runs were performed on the same Sparc10 machine and all times are in seconds.

7. Conclusion

This paper has described the successful transition from a proprietary simulator to a standard VHDL simulator. The benefits of over 12 years experience in large systems simulation have been transferred to the new environment with excellent results in terms of performance and usability. The comparison with commercial simulators illustrates the need for simulation technology to consider

the requirements demanded when modelling large systems.

Figure 4 summarises the strengths of our VHDL simulator when compared with both commercial direct-compiled simulators and "other" types of simulator (where other covers interpreted and conventional compiled-code simulators). Assuming all simulators are, more or less, on a par at the gate level, the graph illustrates how MSIM, with the built-in features and algorithms for high level descriptions, provides significant benefits for "pre-synthesis" simulation. For some examples this benefit can amount to greater than 100 times simulation performance improvement compared to commercial simulators.

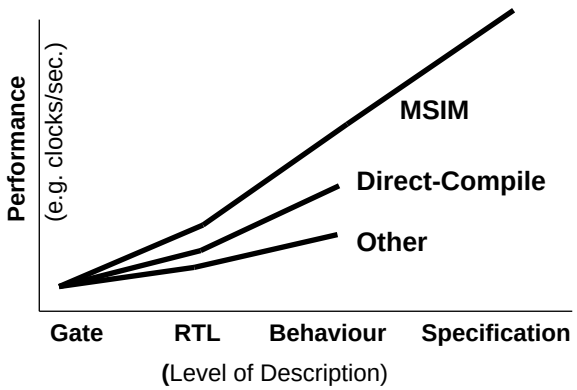

Figure 4. Simulation Performance at different levels of VHDL

The use of a simulator designed for performance has already provided a significant advantage to ICL in design productivity and design correctness and, as the higher levels of modelling become more prevalent, this attribute will maintain its importance for many years.

8. Future Enhancements

Figure 4 deliberately shows the performance of MSIM improving beyond the Behavioural level. This is to illustrate the expected performance at Specification level - a level of description not currently supported by VHDL. Use of System Specification descriptions has already proved successful at ICL [6] using the MSIM/S3 simulator and we confidently predict that these results will be transferred to MSIM/VHDL [7] .

9. References

[1] G.P.Abraham, D.C. Freeth and H.Vosper, "SX Design Processes", ICL Technical Journal Vol. 7, 1990.

[2] S.Hodgson, "A Multi-Level, Mixed State Simulator for Hierarchical Design Verification", *IEEE European Design Automation Conference 1984.*

[3] VHDL Tool Integration Platform, COMPASS Design Automation.

[4] IEEE Standard VHDL Language Reference Manual. IEEE Std 1076-1993. The Institute of Electrical and Electronic Engineers, New York, USA, 1994.

[5] E.G.Ulrich, "Event Manipulation for Discrete Simulation Requiring Large Number of Events", Communications of the ACM, September 1978.

[6] A.Jebson, C.Jones and H.Vosper, "CHISLE: An Engineer's tool for hardware system design", *ICL Technical Journal May 1993.*

[7] M.M.K.Hashmi and A.C.Bruce, "Design and Use of a System-Level Specification and Verification Methodology", *IEEE Euro-DAC 1995*