

# An Architecture for VHDL Integrated Test Development

Eddie Leung, Nafees Qureshy, Tim Rhodes, Tso-Sheng Tsai



**IKOS SYSTEMS, INC.**  
19050 Pruneridge Ave  
Cupertino, CA 95014  
Tel: (408) 366-8571  
Fax: (408) 366-8699  
Email: nafees@ikos.com

## Abstract

With the advent of HDL based VLSI design, synthesis and verification, there is an emerging need to move module and chip level testing earlier in the design cycle. The use of HDL based test benches to verify functionality could be applied to determine fault coverage for wafer sort testing. Test benches with good functional coverage should be able to provide close to, if not complete, targeted fault coverage. A VHDL simulation environment with this capability requires a VHDL fault simulation algorithm with certified gate level libraries support. The later requirement is being addressed with the VITAL effort, but efforts for the former are still in their infancy. Even if VHDL based fault simulators become available soon, it will be a few years before the simulators are able to deliver acceptable performance. To provide designers with language based testing tools today, this paper presents a VHDL based mixed level simulation environment with a fully integrated, high performance, full timing, deterministic fault simulator that supports over 120 certified ASIC libraries. This fault simulator, named Voyager FS, is integrated into the IKOS' mixed level simulation environment using a high speed simulation backplane which connects it to IKOS' VHDL simulator. A mixed level simulation architecture, with it's debugging and tracing capabilities enable designers to quickly and easily verify test patterns on module or chip level, early in the design phase. A set of benchmark results are also presented. Finally, future work to enhance performance and test capabilities is described.

## 1.0 Introduction

Nearly all complex integrated circuits today are designed using a top down methodology. This involves describing and simulating the design at higher level of abstraction before moving to gate level representation. Behavioral level and RTL (Register Transfer Level) VHDL descriptions of designs are simulated to verify functionality. Logic synthesis tools are then used to translate the RTL description to a gate level representation, targeting ASIC vendor's libraries which are certified on multiple gate simulation algorithms. Present synthesis tools are not perfect and may produce gate level designs with wrong functionality. Therefore, functional simulation of the design at gate level with its original test bench, mostly described behaviorally, is required to confirm correct functionality. An efficient mixed level simulator should suffice here.

One other important use of the gate level representation of a design is to develop test vectors, using fault simulation for wafer sort testing. IC vendors require faults to be placed and simulated at the gate level representation of a circuit. Most of the fault simulators offered by tool vendors today are stand-alone in the sense that they require the stimulus in a particular format and are not integrated in a VHDL based top down design environment. This requires extraction of stimulus from the behavioral test bench and also requires the fault simulation to be run outside the VHDL design environment. The balance of this paper first describes need for gate level fault simulation in a VHDL based design environment. It then presents an architecture of a mixed level simulation

environment that fully supports fault simulation at module and chip level, using VHDL test benches as test vectors, which can be merged with other vectors generated by ATPG tools. Finally, some benchmark data on this language based fault simulator is presented along with future enhancements.

## **2.0 VHDL based design environments and need for fault simulation**

VHDL based electronic design efforts start with a behavioral description of the circuit that is simulated using a VHDL (and sometimes C) based test benches. The design is then translated into RTL and simulated using the same test bench. The test bench itself may be iteratively enhanced to improve coverage on the RTL level description of the design. This improvement of the coverage helps increase designer's confidence in the design and the test bench. The same test bench should be used for gate level simulation of the circuit after the logic synthesis stage to verify synthesis output. It would be convenient if the same VHDL based stimulus, that has a high functional coverage, could be used for assessing the initial fault coverage of the circuit. Fault coverage analysis can only be done on a circuit when its representation is close to the physical representation. Even though gate level description is still a logical representation of the circuit, common physical defects in integrated circuits have been successfully modeled using this representation. Most of the IC vendors require their customers to fault simulate at gate level before accepting the design for fabrication. A test bench with a good functional coverage is most likely to provide a good, if not complete, fault coverage. What a designer need is a high performance fault simulator that is well integrated into a VHDL based design environment in which fault coverage may be measured at different stages of the design of modules and chips, so that design changes may be done iteratively to achieve close to desired fault coverage.

### **2.1 VITAL and fault simulation**

With VITAL being close to IEEE standardization, one would think that a VHDL simulator that supports certified VITAL libraries can be used to simulate a full electronic system, which includes a test bench and the chips that are being designed. VHDL simulators optimized for VITAL have started showing up, but a purely VHDL based test development tool will also require a VITAL based VHDL fault simulator. Efforts in the direction of VITAL based fault modeling have started, but a real VITAL based fault simulator is still a few years away.

Movement towards VITAL certified libraries has started, but looking at the amount of effort and resources that go into certifying libraries, it will be a while before VITAL based VHDL simulators fully replace the present proprietary optimized gate level simulators. Another factor is simulation speed. Will the simulators optimized for VITAL be able to compete with high performance proprietary software simulators and will they deliver the capacity required for gate level simulation of an advanced microprocessor? Considering the above factors, it is apparent that at least for next three to four years another approach will have to be adopted to integrate test in VHDL based design environments.

### **2.2 Need for a high performance mixed level simulation environment**

The only practical way today to achieve high performance, post synthesis system simulation is to build a tool that combines a high performance VHDL simulator with a high performance gate level simulator or a hardware accelerator, using a highly optimized simulation backplane. Some EDA vendors have successfully built this type of tools and these tools have been successfully used in real projects with aggressive schedules.

A good mixed level VHDL simulation environment should provide all the debugging features of a front end design system and the performance required for a backend post synthesis type simulations. It is very important that the designer should be able to move modules that are being designed from one level of abstraction to another with ease. This is required so that modules can be iteratively synthesized and simulated at gate level without going out of the tool. A VHDL based mixed level environment should also allow easy access to non-VHDL models which include both software and hardware models.

### **2.3 Extending mixed level simulation environment to support fault simulation**

In order to provide advantages of a mixed level VHDL simulation environment and satisfy the need for incorporating test development early in the design cycle, a mixed level simulation environment could be extended to support fault simulation. During the design phase, a designer should be able to fault grade modules or parts of an IC using a VHDL test bench as the test stimulus, without going out of the simulation environment. Another advantage of extending a mixed level environment to provide fault simulation is that stimuli from various sources can be merged and applied under a programmed control, and once the IC is fully fault graded, the merged test vectors can be extracted from this environment to be used on a tester to test the real chip.

In a good mixed level, language based design and test environment, a designer may be able to play around with the RTL description, synthesis tool and the test bench to measure and improve gate level fault coverage of the IC at a module or chip level. The test bench designed for functional verification may not be able to provide targeted fault coverage. The remaining vectors to achieve near targeted coverage can be designed by hand or can be obtained using ATPG tools. These foreign vectors can be integrated into the test bench using VHDL's file I/O facility or using the programming language (C or C++) interface to the VHDL simulator. The advantage of this approach is that the foreign vector sets can be applied under programmed control at the desired time during the simulation. Once the IC is fully fault simulated, the test vectors applied to the chip during the complete fault simulation can be captured in a desired format using a VHDL or C/C++ model in the test bench.

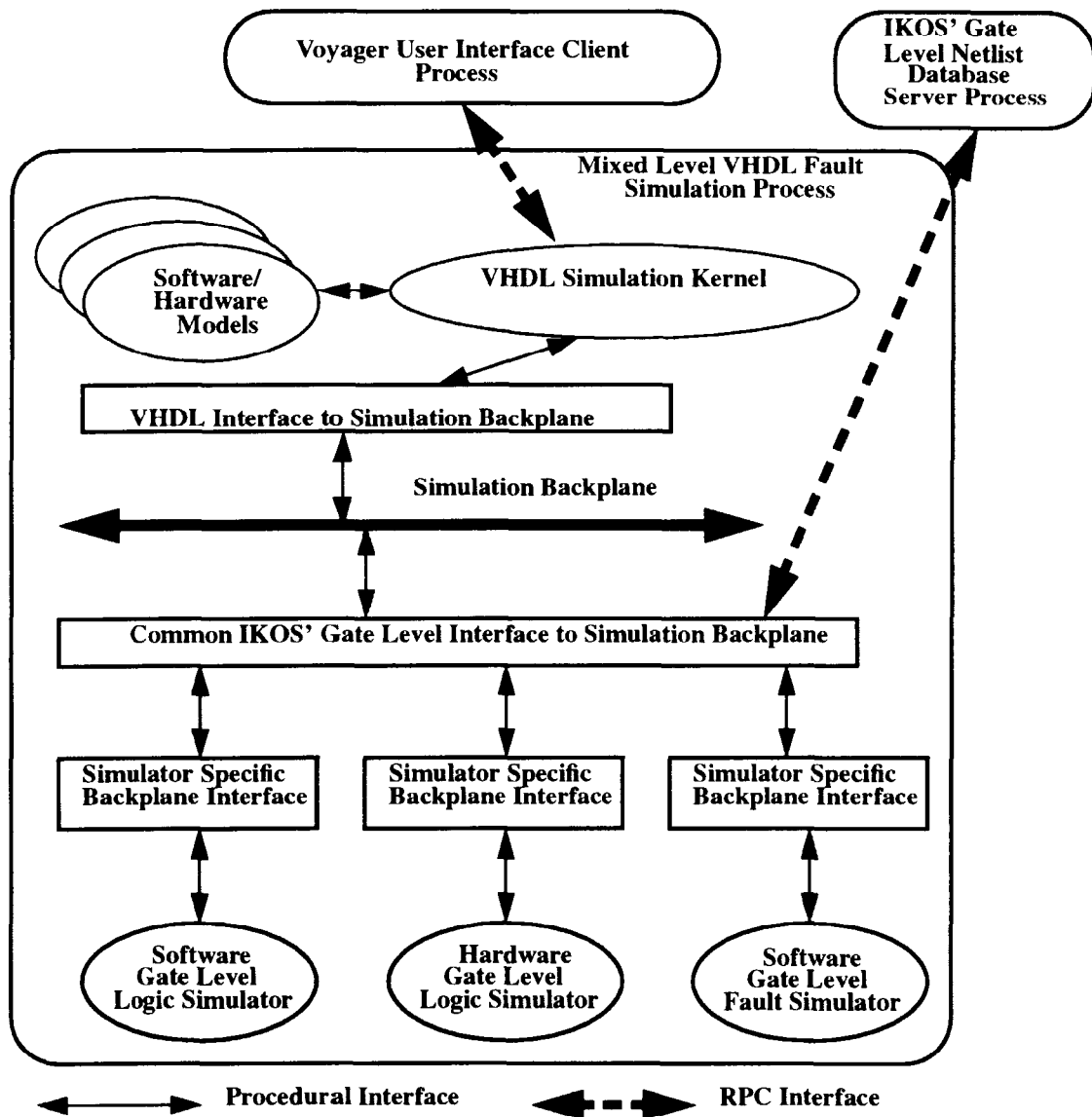
As mentioned earlier, certified VHDL libraries, VITAL optimized simulators and fault simulators are still a few years away. There is a need to provide an integrated VHDL based environment for fault simulating gate level representation of the circuit today. This can be achieved by providing a co-simulation environment that couples a high performance VHDL simulator to a proprietary high performance gate level fault simulator. Outstanding results have been achieved by coupling VHDL simulators with high performance gate level logic simulators and hardware accelerators, using customized simulation backplanes. Since the stimulus activity is very small compared to the event activity in a gate level fault simulator, the simulation backplane which may become a bottleneck in a mixed level logic simulation is not of a great concern here. The simulation backplane does have to be extended to handle fault simulation setup, control and fault event propagation.

The integration of a fault simulator in a VHDL based design and verification tool also provides designer with tools that can trace a fault plane, and breakpoint on some system conditions. Tracing a fault plane may be required for checking out some unexpected behavior during fault simulation.

### **3.0 Voyager - a mixed level simulation and test development environment**

IKOS' Voyager simulation system is a very high performance mixed level simulation environment which allows components of an electronic system to co-simulate at different levels of abstraction, using different simulation engines, each optimized for a particular level of abstraction. This co-

simulation environment also provides access to foreign C models and Synopsys Modeling Systems' hardware and Swift models. Figure 1. depicts Voyager's mixed level system architecture.



**FIGURE 1. Voyager mixed level simulation architecture**

The major co-simulation components of this mixed level simulation system are:

1. A high performance VHDL 1076 compliant simulator (Voyager VS).
2. A high performance gate level simulator that supports over 120 certified gate level libraries (Voyager CS).
3. A high performance gate level hardware based simulation accelerator (Voyager CSX).
4. A gate level concurrent fault simulator that supports over 120 certified gate level libraries (Voyager FS).

5. A high performance object oriented simulation backplane that ties the above mentioned simulators together.

### 3.1 Voyager's mixed level architecture

Each simulator has an interface layer to the simulation backplane. There is a common interface layer for all the gate level simulators. This layer is responsible for handling all the common simulation control and debug facilities. Each gate level simulator itself has a simulator specific interface layer to the backplane. This layer is mainly used at simulation run time for exchanging event and time data. Also, the backplane interface layer for the gate level fault simulator is used for performing fault simulation specific tasks like wiring detectors and setting up strobes. The common gate level backplane interface layer also provides services to the gate simulators by providing a common service interface layer to the netlist database. This service avoids duplication of code in the gate level simulators. The gate level netlist database is a separate process and is connected to the backplane interface layer through a RPC interface.

#### 3.1.1 The simulation backplane

The simulation backplane used in Voyager is an object oriented interface that provides facilities for design elaboration using a specified simulator, simulation control and time and event exchange at run time. A simulator is designated as the master simulator. The VHDL simulator is the master simulator in the case of Voyager. A VHDL simulator may instantiate a component which is to be elaborated using a gate level simulator. This component may in turn instantiate another component which is to be elaborated using a VHDL simulator. The master simulator is used to synchronize all the simulators at run time and to advance the mixed level simulation time. The simulation backplane software itself does not have any implemented mechanism to synchronize time or to transfer event values on nets. This avoids an extra level of software and thus improves performance.

The simulation backplane allows elaborated sub-modules to be connected to other sub-modules, elaborated by another simulator, using signal pads. The types allowed for the signal values on these pads are the event types defined in some of the standard VHDL packages. The most commonly used event type is `std_ulogic` defined in the IEEE 1164 standard package `std_logic_1164`. A simulator that does not support these standard event types should provide mapping tables between its own event types and the event types supported by the backplane. If two different implementations of VHDL simulator, one optimized for VITAL and other optimized for behavioral level code are connected using the backplane, then no event value translation is required.

This simulation backplane has been benchmarked on a Sun Sparcstation 10 using Voyager-CSX hardware accelerator to deliver 22K synchronizations/second and a throughput of 700K events/second. With software gate level simulators the backplane performance is limited by the event processing of the simulators themselves.

#### 3.1.2 Design elaboration

The root of a design, which is always a VHDL model is elaborated first. Components that have been synthesized are recognized using the 'FOREIGN' attribute string. This string has all the necessary information for the location of the gate level netlist. Configuration declarations could be used to select between purely VHDL simulation or mixed level simulation. Other methods like user provided instance or component list may be used to selectively elaborate modules for gate level simulation. During elaboration, the backplane can determine which gate level simulator libraries need to be loaded. If fault simulation is desired, then the fault simulation library is loaded

in by the simulation backplane software, and appropriate netlist modules are loaded into the simulator. The port list in the entity declaration is used by the backplane interface to setup event propagation paths for simulation runtime. Each simulator is given handles to virtual memory locations corresponding to signals where event values have to be written at run time.

### **3.1.3 Simulation control and setup**

After elaboration, simulation control setup may be performed. If a fault simulation is desired, then selection of fault list, detector list and strobe settings may be performed. Presently, only standard fault types (stuck at '0' and stuck at '1') are supported. The default detector list is the list of OUT and INOUT signals of modules that are being simulated using the gate level fault simulation kernel. A user may choose to provide a detector list. Only signals that are at the boundary of or inside the gate level portion of the design are allowed to be chosen as detectors. Portion of the system design that is being used as a test bench should not have fault effect propagation.

In most of the cases all the user provided faults may not be simulated in a fault simulation run. A fault simulation run here is referred to a single run of a test vector sequence using a set of user provided faults. Usually the physical memory size of a workstation is the reason for this. A run time working set of a fault simulation process cannot exceed the size of the workstation physical memory, as it inevitably results in excessive paging and eventually thrashing of the system, resulting in extremely poor simulation performance. To avoid this scenario, the fault list is divided into a set of smaller fault lists. A fault simulation run that simulates all the user provided faults now really consists of several simulation runs each for a set of selected faults.

The master simulator can make a query on the simbus to find out if more faults need to be simulated after each run. If a fault simulator has not simulated all its faults, then the simulation is reset to time zero, without elaborating the design, and restarted using the next fault list. All the fault list selection control may be incorporated in the simulation backplane interface for the fault simulators. The coverage data for each simulation run is collected and merged to generate a combined fault coverage report when all the faults have been simulated.

## **3.2 The concurrent fault simulation engine**

Voyager FS uses a highly optimized concurrent gate level fault simulator which supports over 120 certified ASIC libraries. This fault simulator uses the same simulation backplane interface as used by Voyager CS and Voyager-CSX gate level logic simulators. It does have part of the interface which is fault simulation specific. This part of the interface provide control facilities like detector installation, setting detection criteria and others that are more fault simulation specific.

The fault simulation kernel is optimized for RISC processors and supports both timing and functional simulation. This kernel also fully support simulation of multi port memories which have become quite prevalent in modern ASIC designs. The kernel keeps track of the workstation's physical memory and selects a set of faults from the user fault list depending on this physical memory size. During fault simulation, the kernel keeps track of memory usage and if it sees the usage starting to approach physical memory size, it starts dropping faults till the memory usage is under control. These dropped faults are selected for the next simulation run. Each fault simulation of a fault list may require several fault simulations of selected faults as mentioned in Section 3.1.3.

## **3.3 Creation of a single coherent vector set from multiple test vector sequences**

Voyager mixed level simulation environment has a very efficient and powerful C interface. This allows access to user defined C/C++ models and also allows access to industry standard soft-

ware/hardware models. Using this mixed level environment, a chip can be fault simulated by applying the exact stimulus that is expected when that chip is plugged in the system. This raises designers confidence as the real chip will be fault tested using this stimulus. If this system stimulus is not enough for a targeted fault coverage, then hand written test vectors and vectors from ATPG tools may be integrated into Voyager's mixed level fault simulation environment using the C interface. Several stimuli may even be concatenated in the same fault simulation run by programming each of them to be applied at a particular simulation time. By concatenating the test bench stimulus with foreign test vectors, a complete fault coverage number for the simulation may be obtained.

A mixed level simulation and test development environment with a C programming interface also allows reading of various different test vector formats. Once all the vectors are concatenated, and a desired fault coverage achieved, then the final test vector file may be written out using VHDL's file I/O facility or using the C programming interface. This final test vector file in the desired format can be incorporated in a test program to be used on an IC tester. Several test vector writers, each for a different format, may be written in C or VHDL and may be selected to write out the vectors during simulation.

### 3.4 Benchmark results

Voyager mixed level fault simulator has been benchmarked against the most popular fault simulators in the industry today, including a hardware accelerated fault simulator. The benchmark results have proven that a mixed level fault simulator with highly optimized fault simulation kernel along with an optimized VHDL simulator can even out perform hardware accelerated fault simulators.

Two customer circuits each with on-chip memory were chosen for benchmarking. One of them is a graphics chip and the other is a telecom chip. The benchmarks were done against Zycad's Paradigm XP-2004 equipped with three Super fault boards. Voyager's fault simulation performance for these benchmarks were measured on a Sun Sparcstation 20 with 256MB physical memory. This benchmark data is presented in Table 1. Note that in both cases Voyager FS out-performed Zycad's hardware accelerated fault simulator. Also, this performance data was collected using a single workstation version of Voyager FS. A network distributed version of Voyager FS which is under development, along with a multithreaded kernel is expected to further improve performance by a significant amount.

**TABLE 1. Voyager FS performance on a single workstation**

Circuit Type	Size	Number of Vectors	On-Chip Memories	Voyager Time	Zycad Time	Performance Increase
Graphics	70k gates	135k	2	4.5 hours	24 hours	5.3x
Telecom	28k gates	2.8k	8	34.5 hours	168 hours	4.9x

### 4.0 Future enhancements

Even though benchmark results have proven that Voyager FS is one of the fastest software fault simulator in the industry today, designers who want to incorporate test development early in the design cycle require minimal simulation times, as long fault simulations during design may appreciably increase design cycle. The following sections propose a couple of performance enhancements and incorporation of VITAL based fault simulation engine to the current architecture. Also, a couple of test based enhancements are discussed briefly.

#### 4.1 Network distributed fault simulation

Fault simulations in general are very compute and memory intensive. But at the same time fault simulation algorithms like the concurrent algorithm are inherently parallel in nature. To take advantage of this parallelism, a big fault simulation task may be divided into several smaller fault simulation tasks. Each of these fault simulation tasks may then be assigned to powerful computing resources that are connected via a LAN (local area network). The fault list is partitioned into a set of smaller fault lists and each of the fault simulation tasks is given a fault list to simulate. Each task has a complete run time image of the circuit and also each simulation task applies the same complete test vector sequence.

Maximum speedup for a distributed fault simulation is achieved by minimizing the simulation time for each of the fault simulation tasks and by achieving load balancing among these tasks. Simulation time for each task can be minimized by simulating as many faults per pass as is possible without allowing the process working set size to exceed the physical memory size. Load balancing can be achieved by partitioning the faults in a way that each fault simulation finishes at the same time [1]. Static fault partitioning strategies may or may not yield desired load balancing as several factors affect the load balancing. Some of the factors are location of the faults with respect to primary inputs/outputs, detectability profile of each partition and ordering of test patterns. Some dynamic configuration and dropped fault reassignment techniques may be applied to achieve optimal load balancing. Another technique that could be used is where the fault list is divided into several small lists and these lists are supplied to each of the fault simulation tasks on demand. This technique has been used in some commercial fault simulators and the Chiefs fault simulator [2].

In Voyager, the netlist database server can be assigned the task to partition the fault list. Client software can gather the required information about servers and provide it to the fault partitioner. The client should monitor all the fault simulation tasks on the network and try to perform dynamic load balancing. This part of the client process is also responsible for collecting fault coverage data and invoking the selected report formatting programs. Figure 2. depicts a high level view of the distributed Voyager FS fault simulation architecture.

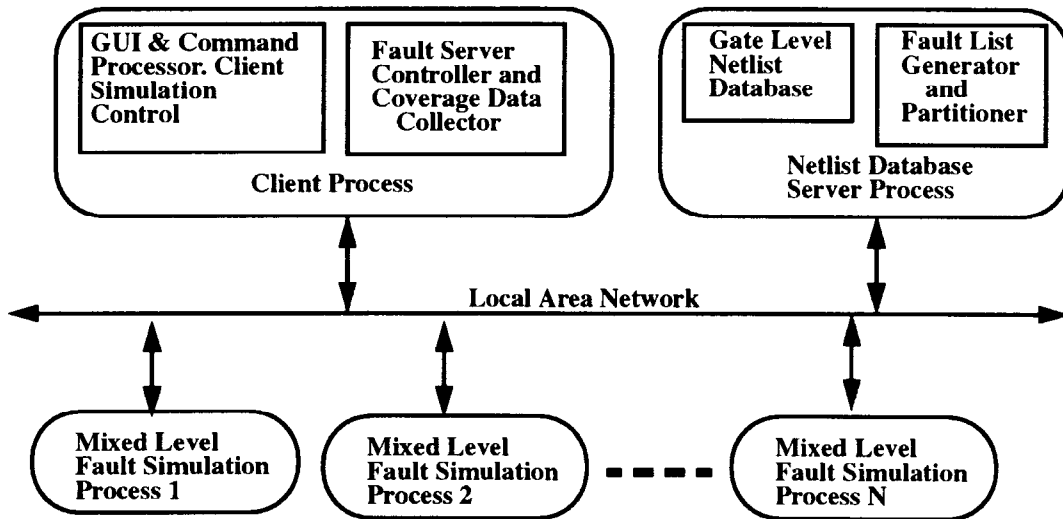


FIGURE 2. Architecture of network distributed Voyager Vhdl mixed level fault simulator

Previous work in distributed fault simulation [1] has observed near linear speedup of fault simulations. For a given circuit and a fault list, the performance drops from the ideal linear speedup as

the number of workstations increase. This drop in performance is because the fault list becomes smaller and other factors like good circuit simulation start dominating simulation time. But usually this drop is observed after significantly improving the fault simulation performance.

#### **4.2 Multithreading mixed level fault simulation**

Execution of a simulation cycle for a gate level fault simulation, concurrently simulating many faulty machines, should typically take much longer than execution of a gate level or a behavioral level logic simulation cycle. If a gate level design is partitioned into 2 to 4 sub-modules and simulated on a desktop SMP machine and if each cycle of simulation on a processor takes appreciable time compared to overhead required for time synchronization of the simulations on each processor, then considerable speed improvement can be achieved. The design should be partitioned at boundaries which are expected to have low event activity. One could regard the fault simulation kernel as an object with its own time queue and evaluator. For each partition of the design, a fault simulation object can be instantiated at elaboration and the netlist partition is loaded in. One could consider these simulation objects as independent simulators connected to the simulation backplane. Each processor in a SMP will be executing the same code with a different set of data [3]. This method can be used to further improve fault simulation speed in addition to the improvement achieved by distributing fault simulation tasks across a LAN as described in Section 4.1. Also, for simulations where timing is not of much concern, cycle based techniques can be used to further improve the simulation speed. Gate level cycle based simulation products on SMP machines have recently started appearing [4]. Cycle based simulations are much more suitable for the parallelism described above, as synchronization only takes place at user defined cycle advance.

#### **4.3 VITAL support**

As more and more ASIC vendors embrace VITAL, existing VHDL simulators will have to start optimizing for VITAL gate level simulations. Optimizations for gate level simulation is quite different than the optimizations for a behavioral logic simulation. One approach could be to simulate VITAL models using a totally different simulation kernel optimized for gate level simulations. This VHDL simulator can be used along with a VHDL simulation kernel optimized for behavioral simulation using the simulation backplane. The mixed level simulation architecture presented here can be easily enhanced to run mixed level simulations using VITAL gate level models.

As stated earlier, VITAL optimized VHDL simulators could have many copies simulating using the backplane, each simulating part of the design on a processor of a SMP (Symmetric Multi-Processing) machine. This type of mixed level simulation combined with cycle based techniques will achieve appreciable performance improvements. VITAL based fault simulators will also start appearing eventually. The mixed level architecture presented here will support a VITAL based fault simulator in the same way as it supports IKOS' proprietary gate level fault simulator in Voyager FS. Voyager should also be able to support a multithreaded implementation of VITAL based fault simulator.

At present propagation of fault through a VHDL behavioral model is not supported in Voyager. Many ASIC designs use standard embedded cores. A simulation of such an IC usually uses behavioral models of these cores, as there is no need to simulate them at a more detailed level of representation. Fault simulation of such a design will require propagation of faults through the standard behavioral models. Propagation of faults through models require fault simulation support in the VHDL kernel. This feature will be implemented in the VITAL based fault simulation engine that will be developed for the Voyager's mixed level test development environment.

#### 4.4 Other test development oriented enhancements

Voyager's mixed level architecture can also be extended to support test development beyond pure fault simulation. One of the proposed extensions is to enhance the fault simulation engine to run in a special mode which provides data on fault coverage increase if a user selected internal node could be brought out as a primary output. This analysis is required in the case where the behavioral test bench, hand generated vectors and ATPG vectors are not enough to achieve targeted fault coverage. A user may select several of these internal nodes and bring them out as a single primary test output using a parity tree. The best candidate nodes for this analysis are the boundary signals of sub-modules in a design as they are easily accessible. A designer may use tools like toggle test to select among these nodes. Nodes which do not change at all during the simulation should obviously be rejected for this analysis. This type of observability tool is most appropriately used during design phase as pin count decisions have to be made.

Another proposed test development extension to Voyager is IDDq vectors detection in a test vector sequence. A mixed level simulation may be used with either a gate level or a switch level simulation kernel to detect IDDq vectors in a test vector sequence which can be a combination of a behavioral test bench, hand generated vectors and ATPG vectors. Voyager-CS or a VITAL compliant simulation kernel could be enhanced to detect IDDq vectors. The final test vector sequence could be used for detecting IDDq vectors, either by running a mixed level simulation or by running a gate level logic simulation in IDDq mode without any behavioral test bench by loading the final test vectors through the C programmable interface.

#### 5.0 Conclusions

This paper has described an architecture that provides a VHDL based test development facility as an extension to a high performance mixed level simulation environment. This VHDL test development tool supports over 120 vendor certified ASIC libraries. Actual performance data has been presented which clearly demonstrates that VHDL based high performance test development systems are available today with complete support of IKOS' certified gate level libraries. With the architecture presented, one could easily incorporate VITAL optimized simulation and fault simulation engines using the high performance simulation backplane. Performance enhancement using network distributed processing and SMP based multi-threading implementations are presented. Again, these proposed enhancements can be easily integrated in the present mixed level simulation architecture. Other test development oriented enhancements to the mixed level architecture have also been proposed.

#### References

- [1] Tassos Markas, Mark Royals and Nick Kanopoulos, "On Distributed Fault Simulation," *Computer*, January 1990, pp. 40-52.
- [2] P. A. Duba, R. K. Roy, J. A. Abraham and W. A. Rogers, "Fault Simulation in a Distributed Environment," *Proc. 25th ACM/IEEE Design Automation Conf.*, 1988, pp. 686-691.
- [3] Steve Walter, "Don't get unraveled by multithreading," *Electronic Engineering TIMES*, July 3, 1995, pp. 34-40.
- [4] Don McInnis and Keith Westgate, "Speedy verification rides 'cycle'," *Electronic Engineering TIMES*, June 12, 1995, pp. 75.