Enhanced Visibility and Performance in Functional Verification by Reconstruction

Joshua Marantz

Ikos Systems, Inc.
josh@ikos.com
1-781-370-1714

230 Third Ave.
4th Floor North
Waltham, MA 02154

1. ABSTRACT

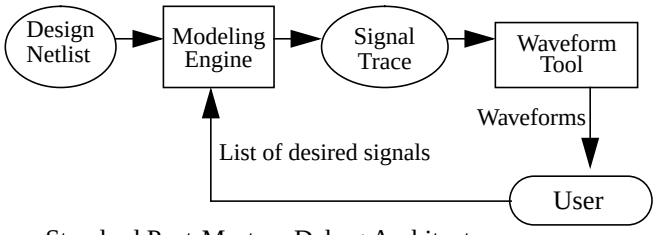
Cycle simulators, in-circuit emulators, and hardware accelerators have made it possible to rapidly model the functionality of large digital designs. But these techniques provide limited visibility of internal design nodes, making debugging hard. Simulators run slowly when all nodes are traced. Emulators provide full visibility only with limited depth, or with greatly reduced speed. This paper discusses software techniques for increasing design visibility while reducing tracing overhead in simulation, and achieving 100% visibility in emulation without reducing speed or compromising depth.

Keywords

Functional simulation, emulation, reconstruction, visibility.

2. INTRODUCTION

It is common to debug large designs with fast simulators "post-mortem." During simulation, a very large trace file is written, containing value-changes for every node in the design, or a subsystem within the design. Upon completion, the designer has rapid access to the entire simulation history, using a commercial waveform viewer.



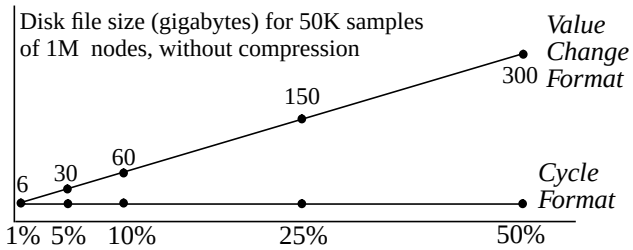
Standard Post-Mortem Debug Architecture

There are several reasons why this architecture is popular:

- Decoupling simulation from analysis speeds analysis.
- Long simulations can be run in the background.
- Expensive simulation licenses can be better managed by a group of designers, while relatively cheap waveform-viewer licenses can be purchased for everyone.
- Verilog's VCD (Value Change Dump) file format [2] has created a competitive market for waveform tools, greatly improving their quality. (Though each waveform vendor also has a proprietary optimized file format which is generated using PLI routines [3,10,12,13])
- The same waveform tool can be used with different simulators (event-driven, cycle-based, gate-optimized, RTL-optimized, analog, ...) and with emulation.

Performance Issues with Post-Mortem Debugging

The drawback of post-mortem debugging is the performance overhead of tracing a large number of nodes during simulation. The resultant VCD files (or vendor-specific optimized format) are very large; they take a long time to write during simulation, and a long time to read when debugging.



Depending on the value change density (defined here as the average number of events per signal per clock edge), a 50,000 cycle simulation of a million gate design can require tens, perhaps hundreds, of gigabytes. This assumes 64-bit (8 byte) time-stamps, 4-byte signal values, and no vendor-specific compression. Allowing for an order of magnitude or so reduction in file size for compression and storage of aggregate signals in one value-change, the file size is still going to be huge -- hundreds of megabytes to tens of gigabytes. This I/O bottleneck reduces the benefit of a fast simulator for the debug phase.

The storage requirement is reduced if two-state data is stored at all cycle boundaries (samples), rather than using timestamped value-changes. In this *Cycle Format*, about 6 gigabytes is required regardless of value-change density. This is still an awkward file to maintain, back up, write, and read. In addition, commercial waveform tools developed for timing simulations don't typically accept this format, and a cumbersome data-expansion and file-translation process would have to occur in order to debug.

1-58113-049-x-98/0006/$3.50

Post Mortem Debugging in Emulation

An in-circuit emulator is a programmable hardware device that models a user design in the context of a *target system*. The target system may be the exact board for which the chip is intended, or it may contain modifications to facilitate emulation. The emulator is usually cabled to the target system as a surrogate for a chip that may not yet be fabricated.

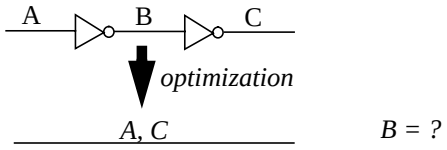
The target system supplies clock inputs and other stimuli to the emulator. The emulator must respond to this stimuli in *real time.* If a configured emulator is determined to run at 1Mhz, then it can't be allowed to lag behind, or it will miss its inputs and fail.

Commercial emulators can also be used in a mode where the clock is controlled by a host computer. For the purposes of this discussion, we shall call this *hardware-accelerated simulation*. For the moment, we are concerned with real-time, target-system driven emulation.

Streaming samples of a million signals at 1Mhz into a disk drive is not practical, so RAM must be used, either in a commercial logic analyzer or on-board. In either case, 6 gigabytes of RAM is expensive, and difficult to route to all design nodes in real time. Consequently, emulation vendors are forced to compromise on either sample depth, visibility, or speed. If speed is compromised, then in-circuit emulation may be impossible due to target-system slowdown limitations, arising from dynamic logic, timed bus protocols, etc. If visibility or depth is compromised, then design debugging is hard.

Visibility Obscured by Optimization

Simulators and emulators may transform the design to one that is functionally equivalent, but whose intermediate nodes may be altered or optimized away. The user can either disable these optimizations or lose visibility. Again, performance is pitted against debugging.



In this paper, we discuss the selection of a subset of nodes for tracing that allows all design nodes to be transparently *reconstructed on demand* in a commercial waveform tool. Reconstruction allows modeling engines to optimize designs without compromising visibility. It reduces real-time storage requirements enough to enable 100% visibility in a real-time emulation of a one million gate design for a time-window of 50,000 samples.

In section 3 we describe the software reconstruction techniques. In section 4 we discuss reconstruction in the context of functional simulation. In section 5 we discuss its application to an in-circuit emulator. In section 6 we discuss further work: event-based storage, reconstruction techniques with regard to timing simulation, and RTL-based modeling.

3. On-Demand Reconstruction

The premise of reconstruction is that when debugging, designers may need access to *any* signal in the design, but will not need access to *all* of them. Storing all the signals on disk or in memory is a waste of time and space. It is better to save a minimal subset of signals and interpolate the stored results only when the designer indicates interest in them. If this can be done quickly, the tools provide an illusion of having traced all nodes without incurring the overhead.

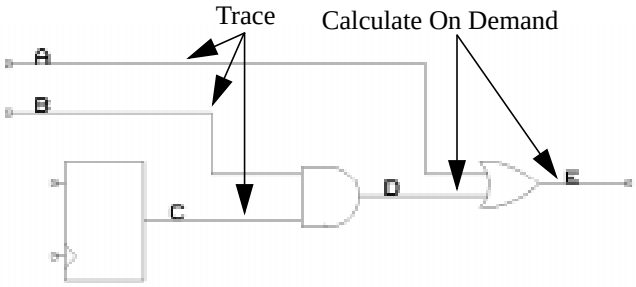
Using the waveform tool, the designer will find that any signal he chooses will be available for post-mortem analysis of its history. However, no designer will want to look at more than a small fraction of the signals in a design. No human can make sense of a million waveforms, and no waveform tool can display that quantity of data. So in all practicality, the enormous storage cost of full visibility will never be paid in full.

Choosing the Subset

In a digital design, it is simple to choose the subset of signals which must be traced during simulation or emulation:

- State Outputs (flip-flops, latches, memory)
- Primary Inputs
- Asynchronous Feedback Loops -- one signal in the feedback path must be traced.

This set of nodes comprises a *basis set* from which all others may be combinatorially derived. We call this set of nodes the *basis nets*. Armed with a trace of these nets, the *basis trace*, the reconstruction engine can deliver on the promise of full visibility without storing a full trace.

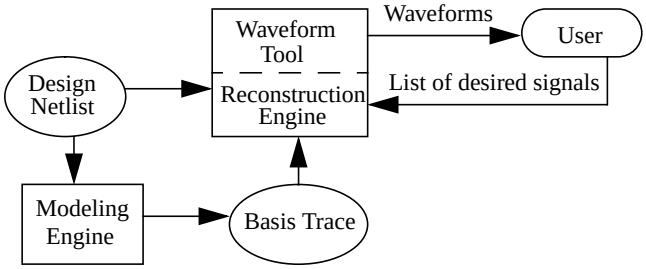


In large chip designs, we have observed that approximately 10% of all nodes are state outputs; 100,000 for a million net design. There are typically on the order of a thousand or fewer primary inputs, due to pin limitations on most chip packages. Most designs that are well-suited for fast functional simulation have a relatively small number of asynchronous feedback loops. To a first approximation, 10% of the design nodes need to be traced. The remaining 90% are combinational logic outputs that can be reconstructed on demand, and never stored to disk.

Reconstructing Combinational Logic

To provide an environment in which the combinational logic outputs can be reconstructed on demand from an interactive waveform

tool, the post-mortem debug architecture is modified:



Modified Post-Mortem Debug Architecture

In this architecture, the waveform tool is linked to a *reconstruction engine* with access to the design netlist. When the designer selects a set of signals for the waveform viewer, a cone of logic is traced back to the basis nets. A set of equations is constructed from this cone of logic, and is applied to the stored logic and input traces. For the schematic above, the equations are:

- D = B & C
- E = D | A

The equation output is sent to the waveform tool for graphical display. There is no need to save this data on disk. Depending on the complexity of the logic cone, it might be reconstructed from an in-memory basis trace faster than it could be retrieved from disk.

Several issues must be addressed to ensure sufficient performance to maintain the illusion of a full trace.

- The design netlist must be kept in memory to facilitate rapid construction of logic cones for desired combinational outputs. The in-memory design representation must conserve memory usage to avoid running out of virtual memory on 32-bit machines when a large design is loaded.
- The design netlist should be loaded quickly to encourage interactive use.
- The equation evaluation must be fast to minimize the delay between a user's request for a new set of signals and the appearance of their waveforms in the graphics display. The evaluation resembles half of a two-phase cycle simulation algorithm [10]. Hence many optimizations associated with cycle-based simulators can be applied to reconstruction, including the use of two-state logic and machine instructions for logic operations.

If performance is poor, users will avoid using reconstruction, and will suffer the debugging pain of huge trace files, incomplete visibility, or a reduced number of samples.

Partial Reconstruction

In some cases, the designer may find that recording the entire basis trace is burdensome, and may wish to focus on one or more design subsystems. In this case, reconstruction is still very interesting.

By tracing the inputs to a subsystem, as well as the state outputs and asynchronous feedback points within the subsystem, full visibility within the subsystem can be achieved at a cost that's much lower than that of tracing all nodes. In a chip design, subsystem inputs are not bounded in number like the primary inputs are. However, the total number of them, in most cases, should still be manageable.

Waveform Tool Integration

The reconstruction engine needs to be very tightly integrated with the waveform tool. This avoids a potential bottleneck in conveying the large amount of waveform data for combinational outputs to the graphics engine. It also avoids a user interface glitch where a user might have to manually transfer the set of desired signals from the waveform tool to the reconstruction engine, followed by a disk write and file reload. If this flow is not transparent to the user, it shatters the illusion of having traced all design nodes and makes debugging cumbersome.

Ideally, the waveform tool selected should be one the designer is already accustomed to using. Otherwise, established debugging habits must be broken, making the solution less effective. Unfortunately, this requirement is at odds with the tight integration. There is no standard API for waveform tools.

At the expense of speed, VCD files can transmit waveforms from the reconstruction engine to most waveform viewers. If the waveform tool supports a scripting language such as TCL, the interaction between the waveform tool and the reconstruction engine can potentially be hidden, avoiding the UI glitch.

Risks

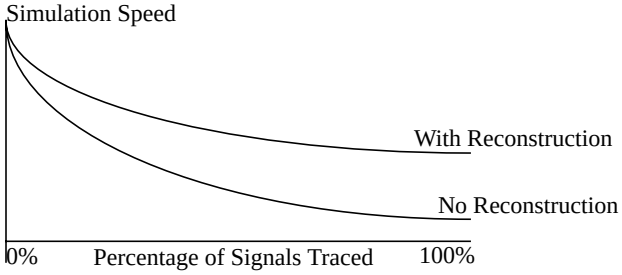
Aside from performance, the biggest risk in using a reconstruction engine is that an inconsistency with the modeling engine (simulator or emulator) could result in incorrect data being displayed. Potential pitfalls include:

- Behavior of undriven nets
- Behavior of multiply driven nets
- Behavior of temporally undriven tri-state busses
- Behavior of unstable asynchronous feedback loops

When two-state logic is used, the modeling and reconstruction engines must use the same policy for dealing with these conditions. For example, if a two-state modeling engine grounds undriven nets and lets tri-state busses float to 1 when all drivers are off, the reconstruction engine must do the same thing. Otherwise, the waveform display may show confusing results, where the Q output of a flop does not match its D input following a clock edge.

4. RECONSTRUCTION IN FUNCTIONAL SIMULATION

The main benefit of reconstruction for functional simulation is that it creates a more favorable speed-visibility curve.



Sim Speed vs. Visibility, with & without Reconstruction

In simulators where combinational outputs are optimized away, or converted to Binary Decision Diagrams [6,9], reconstruction is *required* to achieve full visibility, even for a small number of samples. Reconstruction is thus ideal for cycle simulators. It is particularly compelling when the simulator transforms the design into a functionally equivalent one, so that "lost" signals can be recovered during debug.

However, reconstruction can be easily used for event simulations without timing. Of course, if the simulator provides multi-valued logic (e.g. 0, 1, X, Z), then the reconstruction engine will have to perform equation evaluation using table lookups rather than machine instructions.

Beyond the reconstruction engine and simulator, a trace-selection mechanism is needed. It takes a list of interesting subsystems from the designer (possibly the entire design), finds the appropriate basis nets, and enables tracing for those nets in the simulator. In Verilog, this could be implemented via a PLI call and controlled by the user from the test bench.

Simulation Results

We integrated a reconstruction engine with Cadence Verilog-XL 2.3.3. We tested it using three gate level designs synthesized to a library of timing-free primitives. The designs have multiple asynchronous clock domains, asynchronous feedback loops, wide multi-ported RAMs, including asynchronous FIFOs, and include both latches and flops. We simulated each design four times to determine how tracing impacts performance:

- No Trace - no signals are traced, no file is saved
- VCD Trace - write standard Value Change Dump file [2]
- VPD Trace - write Virsim's optimized binary format [11]
- Basis Trace - write cycle-based, compressed binary file with values for basis nets only.

We used an unloaded Sun Ultra 2/200 with 1 gig of memory, spooling trace files to a file-server over NFS. The simulations fit comfortably in physical memory.

Table 1: Simulation Time (mm:ss) / File Size

Design	25k vectors 200k nets	650 vectors 320k nets	5k vectors 12k nets
File Size			
VCD Trace	2100 Meg	167 Meg	35 Meg
VPD Trace	579 Meg	65 Meg	19 Meg
Basis Trace	12 Meg	10 Meg	1 Meg
Simulation Time			
No Trace	16:00	4:30	0:27
VCD Trace	54:00	6:58	1:08
VPD Trace	61:00	14:17	1:40
Basis Trace	30:00	11:01	0:54
Number Basis Nets	22,000	39,500	950

The times listed are measured in wall-clock time via the Unix "time" command. The number of nets given includes the internals of library elements, which may not be needed for debugging. The simulation time includes netlist compilation, which is about a minute. The file sizes given each include space required for design hierarchy. A network disk was used for all software, design, and output files.

From this data, we conclude that using the basis trace approach has a profound impact on trace file size. But simulation speed is very design and data dependent.

File Size Conclusions

The basis trace files were consistently dramatically smaller than the VCD and VPD files. This arises from storing fewer signals, and the use of a two-state cycle-based format rather than value changes. X and Z values are stored as exceptions.

Simulation Speed Conclusions

In two of the three designs tested, using the reconstruction engine improved simulation performance. In the 200k net design, the basis trace simulation ran twice as fast as VCD or VPD. In the 12k net design, it simulated 30% faster than VCD and 60% faster than VPD. Some of the speed improvement may be due to the reduced file system overhead for the smaller files, particularly using network file servers. Another source of speed improvement is the need to examine the values of about 90% fewer nets. The ratio of 1 basis net for every 10 nets in the design appears to hold consistently.

In the 320k net design, the basis trace simulation was slower than the VCD trace. This is a large design with a fairly small number of vectors. Much of the design is not covered by these vectors, and remains dormant. This favors an event-based storage approach, where we are employing a cycle-based mechanism. Thus, while the basis trace file is smaller than the VPD or VCD files, the ratio is 6:1, rather than the 50:1 or 20:1 that we see in the other designs.

This data dependency is an artifact of the current implementation, rather than a fundamental limitation of reconstruction. An event-based basis-trace storage mechanism optimized for four-state logic would likely yield consistently superior results to VPD and VCD, both in file size and simulation time.

Waveform Results

We used Summit Design's Virsim waveform browser, with its optimized VPD trace file, to characterize debug performance. Virsim was also used with the reconstruction engine and the basis trace file to compare performance, using a tight API-based integration

When using the basis trace to analyze waveforms, a Netlist database file must be read. The size of the database for the three designs is given in Table 2. The time to read this file is included in the basis trace invoke time. The netlist connectivity information is required by the reconstruction engine to recover values for untraced nodes on demand.

Data is not included for Virsim displaying data from VCD files because the it does not read VCD files directly; a translation tool is used to convert to the vendor's format.

Several operations were timed, in order to convey a feel for system performance in typical debug scenarios. The clock nets, busses, and modules chosen mostly had a medium to high value change

density, although this was not measured.

Table 2: Waveform Analysis Speed

Design	25k vectors 200k nets	650k vectors 320k nets	5k vectors 12k nets
Invoke Time			
VPD Trace	13 sec	22 sec	10 sec
Basis Trace	30 sec	40 sec	7 sec
Netlist DB file size	25 Meg	30 Meg	2 Meg
Add 3-4 clock nets			
VPD Trace	12 sec	1 sec	1 sec
Basis Trace	3 sec	3 sec	1 sec
Add 32-bit bus			
VPD Trace	1 sec	1 sec	1 sec
Basis Trace	1 sec	1 sec	1 sec
Add 2-3k nets			
VPD Trace	27 sec	13 sec	
Basis Trace	22 sec	20 sec	
Add 5-10k nets			
VPD Trace	28 sec	65 sec	22 sec
Basis Trace	47 sec	11 sec	15 sec
Memory Footprint			
VPD Trace	118 Meg	120 Meg	47 Meg
Basis Trace	115 Meg	114 Meg	52 Meg

From this we conclude that there is comparable performance between a debug environment where all signals are traced in a large VPD file, compared to reading a smaller basis trace file and reconstructing the combinational outputs. Which method is faster is dependent on the number of equations that must be evaluated in order to reconstruct the desired nets, locality of reference within the large trace file, etc.

For large designs, basis-trace invocation time is penalized because of the netlist database read step. The cost of this is justified when the number of vectors is large, or when multiple simulation runs are debugged in the context of a single netlist revision.

The memory footprint data was noted after all the operations in Table 2. This verifies that holding the design netlist in memory is not problematic.

The main benefits of reconstruction in simulation are:

- It can increase simulation performance without reducing visibility (or increase visibility with minimal speed loss.)
- It creates dramatically smaller trace files.

5. RECONSTRUCTION IN EMULATION

Commercial emulators today offer the user several choices for design-node visibility:

- Most emulators allow a probe-set of 1k-16k nodes for 10k-100k samples using external or internal logic analyzers [4, 7,8]. The probe-set is hard to choose prior to going "in circuit". When a problem is detected, it may be hard to debug if you didn't choose wisely. Re-running the test may prove difficult if the erroneous behavior was due to a combination of internal state and real world stimulus that is not easily repeatable.
- Some emulators provide a mechanism for retrieving the value of all nodes for one sample by using shadow registers and a "readback" feature available on some FPGAs [8,14]. It is often difficult to debug problems from a single snapshot. You can answer "What?" but not "Why?"
- Record all nodes for all time by slowing the emulator to reduce bandwidth requirements to bulk storage. It is unlikely that such a configuration would run fast enough to respond to a target system, so we label this *simulation acceleration*.
- Some emulators allow rapid reconfiguration of the probe-set, without taking the emulator out of circuit. This is convenient, but you still may have the wrong set of probes selected when real-world stimulus causes erroneous behavior, and you may not get a second chance. You cannot rewind the real world.

Why is 100% Visibility in Emulation Hard?

The fundamental limitation of visibility in emulation is *bandwidth*. Consider an emulation of a million gate design, running at 1Mhz. Assuming that only 2-state values are recorded on clock edges, brute force 100% visibility of a 50ms (50,000 samples) snapshot would require substantial resources:

$$Storage = 10^6 nodes \times 50ms \times 1Mhz = 5 \times 10^{10} bits$$

$$Bandwidth = 10^6 nodes \times 1Mhz = 10^{12} bits/\text{sec}$$

The high storage requirement (6.25 gigabytes) makes it impractical to satisfy with fast RAM. The high bandwidth requirement makes it difficult to transmit to a disk in real time. Recall that the emulator cannot be allowed to pause at any time for saving data because it must respond in real time to an external clock.

Using the reconstruction engine cuts down the nets we need to trace by 90%, reducing the trace memory requirement to 625 megabytes. This is a more manageable quantity of memory to distribute in an emulator box.

The bandwidth requirement is also cut down by 90% to 100 gigabits/sec. This can be spread across the emulation FPGAs or processors,

There are ways to further reduce the fast RAM requirements without compromising visibility, depth, or emulation speed. They are beyond the scope of this paper.

Limitations

Thanks to reconstruction, visibility of all nodes in a real-time emulation is possible. But the sample depth available is still limited by the depth of the trace memory in the logic analyzer.

6. FUTURE WORK

Experiment with Event-based Storage Mechanism

The results obtained using the two-state optimized cycle-storage mechanism indicated uniformly superior file-size, but simulation

speed has proved more data dependent. An event-based storage mechanism should be incorporated into the basis trace and reconstruction engine to see the impact on simulation speed and file size over a wide range of value change densities and a predominance of 4-state logic. We believe that this will result in a consistent speed and file-size advantage over a full VCD or VPD trace.

Reconstruction with RTL Modeling Engines

There are several species of RTL modeling engines. The first is based on direct compilation or interpretation, where the RTL code is compiled to C, native code, p-code, or emulation processor instructions [7]. The second is compile-to-gates, where a fast synthesis engine is used to translate the RTL description to a gate-level description with a translation table between RTL names and gate names.

In compile-to-gates, the reconstruction engine can be used with little modification. A translation table should be used in the waveform viewer so that the user is presented with familiar design names.

To support direct compilation, the reconstruction engine could use a Control Data Flow Graph in place of the logic cone to generate the equation list for the desired nodes. This graph can be used prior to simulation to enumerate the basis nets. Alternatively, with the extra semantic knowledge available from an RTL flow analysis, a smaller basis set can potentially be found, offering further reductions in trace overhead [5].

Timing Simulation

Adding timing capabilities to the reconstruction engine is not prohibitively complex, although it's not clear whether the result would be fast enough to be usable interactively.

In a timing simulation that includes back-annotated net delays from layout, state elements will all change when their clock edge arrives, and each state element may see its clock edge at a slightly different time. A timing accurate waveform display will need to reflect this.

Consequently, the basis trace would have to be value-change based, rather than cycle based. As we have noted above, this approach may prove superior in any case.

The equation-list would need to be augmented with a delay map from the basis nets to the user's desired combinational outputs. This would allow reconstruction to report the correct transition times for each signal to the waveform viewer.

If the user requests waveforms for a large number of signals simultaneously, it might be prohibitively slow to evaluate all equations every time any state element transitions. We may then want to partition the logic cone in some fashion, triggering the evaluation of each a given partition when any of its state inputs change. This *selective trace* methodology [1] is another step along the spectrum from pure cycle-based simulation to pure event-based simulation.

The question of whether the end result of these changes will be fast enough may be superseded by the question of whether designers wish to debug functionality in a timing simulation environment.

7. CONCLUSION

The reconstruction engine presented here provides an opportunity to dramatically improve the debugging environment for both fast functional simulation and emulation. Simulation users can see more of their design with reduced tracing overhead. Emulation users can achieve 100% visibility with significant depth in a real-time emulation. This represents an opportunity for unprecedented debugging power of hard-to-reproduce system-level problems.

8. ACKNOWLEDGEMENTS

Thanks to Jason Campbell, Pat Julik, and crew at Summit Design for providing the tight integration path into Virsim and for extensive support and cooperation. Thanks to Charley Selvidge, Mark Seneski, and others at Ikos for ideas, infrastructure, test cases, and draft review.

9. REFERENCES

[1] M. A. d'Abreu, "Gate Level Simulation", IEEE Design and Test, pp 63-71, December, 1985A.L. Sangiovanni-Vincentelli et al., "Verification of Electronic Systems", 33rd DAC, pp 106-111, 1996

[2] Cadence Design Systems Inc., "Verilog-XL Reference Manual", Version 2.3, 1995

[3] Design Acceleration Inc., Signalscan Database, http://www.designacc.com/Products/sst2db/sst2db.html

[4] Ikos Systems Inc, VirtuaLogic Data Sheet, http://www.ikos.com/products/vsli/index.html

[5] D. Kirovski, M. Potkonjak, "A Quantitative Approach to Functional Debugging", ICCAD 1997, pp 170-173

[6] P.C. McGeer, et al, "Fast Discrete Function Evaluation using Decision Diagrams", ICCAD 1995, pp 402-407

[7] Quickturn Design Systems, Cobalt Data Sheet, http://www.quickturn.com/products/cobalt.htm

[8] Quickturn System Realizer Data Sheet, http://www.quickturn.com/products/systemrealizer.htm

[9] A.L. Sangiovanni-Vincentelli et al., "Verification of Electronic Systems", 33rd DAC, pp 106-111, 1996

[10] Speedsim Inc, Cycle simulation technology, http://www.speedsim.com/technology/cbs.htm

[11] Summit Design Inc, Virsim Data Sheet, http://www.sd.com/products/verification/virsim.html

[12] Systems Science Inc, Magellan Data Sheet, http://www.systems.com/

[13] Veritools, Undertow Data Sheet, http://www.veritools-web.com/towv.htm

[14] XILINX, Inc., "The XC4000 Data Book", Aug. 1992