The One Million Gate ASIC Challenge: Not with Behavioral Modeling and Synthesis

Steve Meyer
Pragmatic C Software Corp.
220 Montgomery Street, Suite 925
San Francisco, CA 94104
(415) 296-7017 - FAX (415) 781-1116

Abstract: The large ASIC IC design problem is discussed within the Verilog hardware description lanaguage research program. Large ASIC design using the behavioral design with synthesis method is shown to be infeasible by showing that evolutionary improvement of traditional gate level tools and increased logic designer skill is both necessary and sufficient. Suggestions for applying Verilog to large gate level ASIC design and for improving Verilog in the accurate timing gate level simulation area are made.

1. Introduction

Advances in semiconductor manufacturing technology will soon make ASICs containing one million usable gates (four million CMOS transistors) technologically feasible. The ten fold increase in functionality over current large gate arrays poses a challenge to digital ASIC logic design and verification. There are two competing design approaches applicable to such large ASICs. One approach advocates a fundamental change in method and design tool selection in which ASICs are designed at the abstract behavioral level and then mapped by means of synthesis computer programs to ASIC vendor nets lists for manufacturing. The term synthesis is applied both to mapping from behavioral to gate level and to the entire abstract behavioral method. This method uses abstraction and formal verification to solve the circuit complexity problem.

The other approach advocates evolutionary improvement to current digital logic design methods while continuing to use current tools such as schematic editors, simulators and timing verifiers at the gate level (circuits coded in terms of vendor defined macrocells). Improvement occurs through improved designer skill, increased designer experience, and faster or more accurate tools. This method advocates using the cunning of human reason to solve the circuit complexity problem.

The ASIC complexity problem is the central challenge now facing design using Verilog and in the authors opinion should provide the major impetus for Verilog language development. Verilog is now the sign off simulator of choice for most ASIC vendors. However, since the transfer of responsibility for the Verilog standard to Open Verilog International, the version 2.0 Verilog Language Reference Manual (LRM) and most commercial Verilog simulator improvements are aimed at improving Verilog's application to the behavioral modeling with synthesis method. The LRM now includes suggestions for using Verilog for synthesis intermixed with the language definition (OVI[1993], for example 4-4, 5-5, 8-11) instead of providing a separate manual that discusses using Verilog for synthesis so that only those interested in synthesis would need to read the material. When vendors discuss simulator improvements, improved behavioral and RTL (register transfer level) simulation are primary topics.

This paper presents the argument against abstract behavioral design. It shows that the behavioral method is at best an untried method in the ASIC cell based design area whose support has come solely from anecdotal evidence.[1] It argues Verilog should remain design methodology neutral. This paper challenges application of formalist behavioral design as characterized by the AI research program (Lighthill[1973]) to digital logic design.[2] It is important because most ASIC vendors have announced they will soon change to the behavioral method. The same ASIC vendors who spend years evaluating wafer production equipment, run extensive controlled experiments and even then slowly phase in new equipment are for some reason rushing to adopt the untried behavioral method.

---

1. See for example various user synthesis experience papers from any recent Design Automation or IC CAD conference proceedings.

2. See Meyer[1993a] for a falsification of the related silicon structures IC layout method. Silicon structures is related to behavioral synthesis since both methods apply formal mathematics (Lakatos[1976], Fetzer[1988]) to electronic circuit design.

Verilog design methods and language changes are presented for those who believe competitive advantage can be achieved by continuing to utilize traditional gate level tools.

After providing detailed definitions of the two methods, behavioral models are shown to lack timing and unknown state accuracy required for correct ASIC design. This falsifies[3] the strong form of abstract behavioral design that states that once abstract behavioral design is complete, remaining design steps are mere routine data processing. Section 3.2 falsifies the first weaker form of behavioral design through synthesis that starts with behavioral abstraction then uses synthesis to produce a gate level net list which then becomes the starting point for normal gate level ASIC design.

Section 3.3 falsifies the second weaker form of synthesis in which a designer creates a net list indirectly by hand coding behavior descriptions, functional constraints, and synthesizer control commands that are used to control execution of a synthesis program. In the second weaker form, a designer studies output net lists from synthesis, modifies constraints and control commands and then reruns the synthesizer program. Synthesis here is, in the weakest form of abstract behavioral design, merely net list translation to reduce ASIC net list coding effort and in the view of the author is less efficient than simply drawing one's design with a schematic editor. The falsifications taken together show that one million gate ASICs will never be achieved by abstract behavioral design with synthesis.

The next section discusses problems for which Verilog style behavioral and RTL modeling can be advantageously used. It is important for successful one million gate ASIC design to understand that the primary circuit model is the gate level macro cell net list and that it must be understood in detail by experienced logic designers. But to debug and verify such large designs, behavioral features of Verilog become valuable. Also, correct ASIC function within a digital electronic system can be advantageously verified using Verilog. Finally, changes to Verilog to assist in large ASIC design are presented.

2. Design Method Comparison

2.1 Behavioral Abstraction through Synthesis

The widely publicized behavioral abstraction method advocates fundamental change in design tools selection in which designers work only at the behavioral level and computer programs map from behavioral specifications to semiconductor vendor ASIC macro cell net lists. This mapping is called synthesis.[4] According to advocates of abstract behavioral design through synthesis, by moving circuit design to a higher level of abstraction, interconnection complexity of large circuits can be reduced (deGeus[1993]). Large circuit blocks can be modeled behaviorally and verified using behavioral features of a simulator. Synthesis can map a circuit to gates for fabrication.[5] According to synthesis, the resulting gate level circuit need not undergo further verification since it's correctness is guaranteed (possibly by means of formal mathematical proof).[6] In practice ASIC vendors still require gate level simulation[7] using designer supplied test patterns.

The high level behavioral method claims to reduce labor involved in circuit design by moving design to the abstract behavioral level and by allowing large behaviorally coded circuit blocks to be reused in different designs. ASIC design becomes electronic system architecture and the one million gate ASIC design problem becomes object oriented behavioral programming.[8]

2.2 Evolutionary Improvement to Traditional Gate Level Design

The traditional tool gate level design approach advocates evolutionary improvement to current methods. Current tools such as schematic editors, simulators with accurate gate level capability and timing verifiers are utilized by experienced designers. According to this method, accurate gate level timing simulation is the correct level for ASIC design because gates physically exist on the ASIC substrate and because vendor macro cells are defined by intrinsic gate output loading and interconnect metalization delay

---

3. Falsification is used here in the Popperian (Popper[1959], Lakatos[1970] or Lakatos[1976]) sense of refutation of the correctness of a scientific theory by means of experiment or argument.

4. A better name for synthesis in evaluative neutral language might be: behavioral to gate model net list mapping. A possible name in critical language might be: designer blind replacement of inaccurate behavioral models by slow canned gate level circuit sections.

5. Fabrication here is used for any method that customizes ASIC interconnects such as field programming. However, currently large ASICs require at least fabrication (etching) of wafer metalization layer interconnects.

6. See Fetzer[1988] for a falsification of formal proof of computer programs.

7. Simulation in this paper means mimicry of the binary logic switching behavior of an ASIC design. High level behavioral design sometimes confuses digital simulation with system simulation that is used to verify correct function of a large system when statistically distributed stimuli are applied.

8. See McGregor[1990] for an CACM issue devoted to object oriented programming. See Brooks[1987] or Guthery[1989] for criticism of object oriented software design.

(Dhimant[1990]). Higher level behavioral simulation is inaccurate because differences between transistor pair rise and fall switching times and gate loading cannot be modeled behaviorally.[9]

Lower level analog simulation using probably Spice (Nagel[1975]) is unnecessary because all analog effects must be removed from macro cells or volume IC manufacturing would be impossible. Even if analog simulation were faster than gate level simulation, ASIC modeling and simulation must be performed at the gate level because gate level simulation can accurately model unknown state behavior of ASICs (during reset operations unknown logic levels and timing violations probably can be safely ignored), and because gate level models allow ASIC vendors to improve process parameters and macro cell design without effecting ASIC function and therefore gate level timing.

Evolutionary skill improvement means utilization of better trained and more experienced logic designers. One million gate ASIC design may not be so difficult for design groups with senior designers who have experience with hundred thousand gate macrocell coded circuits. Experienced ASIC designers can develop improved architecture and improved circuit gate level specifications during circuit development. Functional and logic improvement go hand in hand with improvements in one area facilitating improvements in another. Each solved concrete problem can improve conceptual understanding of a design.

Evolutionary tool improvement means incremental increase in functionality, speed of execution or ease of use. Faster programs on faster computers allows more testing and debugging within given economic design constraints. In addition, improvement can mean slower execution in order to more accurately model gate switching function. Such improvements are better and more rapid unknown (x) injection for problematic switching, spike analysis (Szygenda[1972]) and other kinds of timing problem detection (Bose[1977]). Finally, tool improvements can mean simplified circuit debugging and analysis. For example, improvements to specificity of breakpoint setting and signal viewing can increase the ability of skilled designers to deal with complexity of large ASIC designs.

The current gate level method unifies ASIC vendor substrate and macro cell library design and customer ASIC design by providing one unified view (accurate timing gate models with layout determined wire delays) referenced by all designers. The gate level method converts large ASIC design to application of circuit function specific knowledge by skilled designers. Such designers must be free to choose tools according to individual and design group tastes.

3. Falsification of Synthesis

3.1 The Strong Behavioral Method

In the strongest form of synthesis, a circuit is designed only at the behavior level. After behavioral design the final ASIC is synthesized by in effect pushing a button. The strong method cannot work because accurate timing analysis is impossible. It is impossible to predict complex circuit (containing sequential logic) behavior without simulation. Otherwise, at least some ASIC vendors would allow circuit sign off without simulation. Behavioral design lacks any conceptual path for converting timing problems detected during simulation to circuit design improvement. This falsification does not depend on arguments from practice. Efficiency considerations are irrelevant. If a high level behavioral design fails in simulation, a circuit must be thrown away. Only complete high level behavioral system redesign is possible.

This impossibility of high level behavioral synthesis should not be surprising for various reasons. Development of the synthesis tool itself is equivalent to the general artificial intelligence problem because circuit design includes much intuition and experience (Lighthill[1972]). Published technical papers supporting synthesis use examples that are redesigns of previously intuitively designed circuits and even then the class of synthesizable circuits, according to published research papers, is severely restricted (most notable is sequential feed back). This pattern of promising initial resulted followed by lack of confirmatory evidence is known as post facto scientific results and has historically indicated that a research program was in its degenerative stage (Lakatos[1970]).

The behavioral synthesis research program has followed the same pattern that failed for artificial intelligence (Lighthill[1972]), automatic programming (Earley[1973], Earley[1975]), and natural language translation.[10] Initial positive results from toy problems become negative results when more realistic problems are considered. One cause is the combinatorial explosion in the size of the circuit logic state space that

9. This lack of internal circuit detail representation explains why the various Verilog path delay timing constructs (OVI[1993], pp. 14,2-38) are only useful where path polarity changes do not effect off block delay and after gate level design so that path delays can be determined by measurement.

10. See Winograd[1973]. Dreyfus[1986], pp. 67-90 discusses this pattern in detail.

must be searched. The number of circuit states grows exponentially with circuit size since a computer program it at best able to perform combinatorial search of the state space. Notice human intuitive gestalt driven problem solving is not limited by this combinatorial explosion.

Behavioral synthesis forces exclusive top down design eliminating the common methods that combine various styles of design such as design toward a goal, work from bottom up by designing a project specific higher level macrocell library, using top down for the initial design conception and partitioning, mixed top down and bottom up design, etc.

The author is unaware of any publication of controlled experiments showing the advantages of behavioral synthesis. Such experiments would explain where the method works, where it does not, and would offer possible explanations of the results. No new method should be adopted without such experimental verification. Related to the lack of proof by controlled experiment is the difficulty characterizing human design methods. Because of the attention received by behavioral synthesis it is now advantageous for a designer's career to claim that synthesis was used even if an ASIC were designed using traditional gate level tools. Claims of successful application of synthesis must be carefully verified.[11]

3.2 The Initial Net List Creation Synthesis Method

In the first weaker form of synthesis, high level abstraction is used to design circuit architecture, synthesis is used to create a net list that then becomes the starting point for normal gate level ASIC design. Here synthesis is first cut net list generation. As shown above, strong behavioral synthesis cannot work because timing information has been removed from design. The first weaker form of synthesis is useless since timing information cannot be represented. If circuit performance and cost were irrelevant to ASIC design, this weak form of synthesis could conceivably work. An ASIC net list could be synthesized, simulated to debug logic function and results from simulation could then be used to set system clock period to the maximum delay path through the resulting ASIC. However, such a circuit would be considerably larger, slower and therefore more expensive than hand crafted circuits. In a competitive world, such inefficient methods are in effect unworkable. In reality, the improvement of 10 to 25 percent, assuming the same technology, in clock rate and 30 to 50 percent in circuit area achieved among the very best designers compared to only good designers can make the difference between a successful and unsuccessful system.

This weak form of synthesis as net list template generation is based on the assumption that once a high level architecture is designed, the ASIC implementing the architecture is just a combination of circuit submodules. Design is portrayed as reuse of expert knowledge. Here, gate level timing information added after synthesis is trivial and small in volume. This view misses both the difficulty of ASIC logic design and potential system improvement achievable through improved gate level design.[12]

The crucial part of ASIC design occurs at gate level during accurate timing net list and test pattern preparation. In fact, the following gate level design improvements offer the most potential for overall improvement to an electronic system.[13] 1) Improvement of latch or flip flop selection used to implement time critical paths. 2) Careful analysis of logic polarity versus required inverter insertion, 3) Elimination of as much clear, set and complicated flip flop clocking logic as possible. 4) Basic block specialization in which a circuit consisting of one type of basic register or block is improved by changing to specialized blocks for each required function. The cost is in syntactic complexity but experienced designers have little trouble with numerous but conceptually similar functional blocks. 5) Utilization of discovered timing problems to motivate redesign improvements in system partitioning and interface specifications. Of course, some interfaces such as connections to standard busses must not change. 6) Use of ASIC sign off and manufacturing test pattern development problems to locate problematic circuit areas.

This first weak form of synthesis is useless because it applies only to initial net list preparation that is a small part of ASIC design yet eliminates availability of

---

11. A similar lack of method workability proof exists in the related object oriented programming area. Even though C++ is replacing C in terms of number of units sold, the percent of C++ compiler software programmers who utilize the unique C++ features is low (Floyd[1989]).

12. An example of this oversimplification occurred in EDIF 2.0 (EIA[1988]). A form for representing gate capacitance was included but it only allowed representation of constant capacitance. No mechanism was included to represent the various mostly linear factors upon which capacitance depends let alone the various complicated formulas required to compute capacitance and inductance in net list interconnection trees. Also forms for distributing capacitance were not included.

13. Most of these areas of improvement were observed by the author in designs completed at the LSI Logic Milpitas design center during the middle 1980s.

circuit schematics.[14] It can not be iterated. Once an initial net list is synthesized, the method reverts to traditional gate level design. At best this method may contribute to designer psychology.

3.3 Synthesis by Means of Command and Constraint Coding

In the second weaker form, synthesis is used to indirectly create a gate level net list under control of synthesis computer program constraints, behavior descriptions, and commands. This could be called net list creation by template selection or possibly indirect design concept injection by means of computer program command language coding. The second weaker form of synthesis is also useless since the designer is still coding the ASIC net list but the coding is indirect through computer program input.

This method may work for standard circuit blocks such as adders. All the various parameters required to specify an adder are codeable in the synthesis tool constraint and command language. Adder factors are: behavioral clues such as carry handling requirements, constraints to determine timing, and commands learned by a designer from experience to cause the synthesis program to select the adder template that the designer knows is needed. The designer next studies and probably simulates the output circuit and if the net list is not the one the designer had in mind, parameters are changed and the process is iterated. It is possible for designers to become skilled at command language encoding to produce a desired output circuit. Here, if a gate in a synthesized block has the wrong drive (for example, causing it to have too much delay), a designer would modify specifications to change constraints to cause a gate type with correct drive strength to be selected. This indirect process is so cumbersome why bother with it. Why not just draw the design.[15]

Even in this simple case where synthesis by command selection is possible, the synthesis program will probably not be able to take advantage of circuit function specific knowledge. For example, unless the adder is to be used in a computer arithmetic unit, there are probably numerous problem specific constraints on the class of inputs that the adder will need to process, but it will be unlikely that the synthesizer template will be able to deal with those patterns.

For more complicated blocks such template selection will not work because there will probably be no matching template. Even if a matching template for a given function exists, large speed and area improvement will probably be achievable by combining different functions into the same logic block. A new combined block will probably not exist in the synthesizer template library because if a synthesizer attempts to incorporate all possible templates it will soon become useless. Circuit state space size will grow exponentially. Circuit arrangement options of a synthesis system will be limited in important intuitive design areas such as high assertion direction selection, latch or flip flop type and glue logic patterns.

Here, Circuit design becomes dependent on research in synthesis because synthesis of, for example, common sequential logic is still an open problem. There will be no designer skill improvement since low level interconnect experimentation and simulation is impossible. There will be no circuit design progress since synthesis from templates eliminates the possibly of discovery of better methods for circuit block net list construction. Undiscovered block designs cannot possibly already be in the template library. There is a good chance that experts who designed the synthesis template library will lack problem specific knowledge for non standard circuit functions. Finally, by generating net lists from templates, resulting circuits can be at best as good as competitor's circuits (also probably not much worse).

4. Applying Verilog to Large ASIC Design

The previous sections have shown that the object of ASIC design must be accurate timing model gate level net lists and that detailed intuitive understanding of the gate level design is crucial. This section suggests approaches for utilizing the extensive programmability including the programming language interface (PLI) and behavioral modeling capability of Verilog to assist in ASIC design. Although Verilog gate level simulation is not significantly better than that provided by previous gate level only simulators, it is much more useful for ASIC design. In earlier simulators, circuit error detection (timing errors), stop gap models for unfinished interfaces, and complex debugging conditions could not be expressed. Programming was only applicable through analysis of I/O port activity traces written in tabular form. Even though traces have utility and are available in Verilog using the $dumpvars system task, such traces are less valuable than online access to internal circuit state through monitoring and cross module hierarchical

---

14. Although a discussion of schematic editors is beyond the scope of this paper, the availability of schematics is considered important by many design groups especially for transfer of designs to manufacturing. There are programs that can map from net lists to schematics but they are not widely used because the important pictorial representation of design conception is missing. Schematics also may help traditional gate level design by providing gestalt images of circuit design conceptual frameworks in the same sense that X rays provide such images for medical diagnosis.

15. There must be people who would rather drive a car by sitting in the back seat and using remote controls, but that would not be considered a workable driving method.

variable examination.

4.1 Using Behavioral Features

Behavioral features can be applied during ASIC design and during verification of correct ASIC function within a larger system. There are a number of obvious uses of behavioral Verilog features such as monitoring and strobing signal changes, using event and delay controls to set change breakpoints or forcing a circuit to a required state during debugging. Here behavioral features implement the debugger.

Another use of behavioral features involves coding simplified models for uncompleted design sections and prototyping of experimental designs during ASIC development. In a sense behavioral features would not be required if every ASIC development project went according to plan. One important use of behavioral modeling allows debugging part of a circuit when blocks it interacts with are not yet finished. Here a simple behavioral model can be coded to mimic just enough of the missing subcircuit's behavior to allow debugging of the completed part. Subcircuit I/O behavior can be modeled by driving output ports with continuous assignments whose right hand side expression are registers controlled by procedural tasks and functions. Inputs can be monitored directly by procedural code. Whatever timing accuracy is needed can usually be modeled by specify section delay paths. The limitations of specify paths discussed in the first item in section 5 will normally not be a problem for this kind of modeling.

A more sophisticated form of behavioral modeling involves coding circuit models for the sole purpose of monitoring ASIC state during debugging. Normally, tasks will be written that access wires through cross module references. The internal states can then be compared to expected values and messages can be written to indicate discrepancies. These tasks are run during simulation but will not exist in the fabricated circuit. Using the PLI for monitoring allows additional programmability. For example, a circuit that implements a communications protocol could be checked with a PLI task C program that computes and compares intermediate results.

The second area of behavioral modeling involves verification of a completed ASIC in a system. One approach involves first replacing the finished accurate timing model ASIC (accurate timing here means including post layout wire delays) with a unit delay model and verifying identical system function with the new delay model. Next replace the unit delay model with a behavioral model (possibly implementing only the interface) and then repeat the system tests. Any needed I/O port delays here are determined from measurements of the completed ASIC. These steps insure that the chain from gate level function to electronic system function matches reality.[16] The author believes that exactly this Verilog capability allowing complete connection of the chain from gate level to system function will lead to replacement of all other HDLs by Verilog.

5. Changes to Verilog to Improve ASIC Design

Even though Verilog is the reference simulator for most ASIC vendors, there is still room for improvements in Verilog gate level modeling accuracy and flexibility.[17] The following changes should be made:

1. **Per Instance Back Annotation of Gate Delays**
   Verilog needs a mechanism for back annotation of different gate delays for each instance of the module containing a given gate. Currently, specify paths are required solely to allow PLI back annotation. However specify paths have a number of problems. First, accurate modeling sometimes requires use of inconsistent and obscure module input port delays (MIPDs) for multiple paths with one destination but different delays. Second, for macro cells with many ports but little internal activity, gate models are more efficient since only the gates on active paths are evaluated and scheduled. Third, specify paths can be inefficient for modules with many input and output ports because the number of specify paths through a module is the product of the number of input bits times the number of output bits. All specify paths that have the same destination must be evaluated sequentially to determine the latest input change. Fourth, rise and fall delays can not be accurately modeled for paths with numerous inversions. Finally, the calculation for adding internal layout wire delays to macrocells modeled using specify paths is difficult. Which paths need to be changed after an internal wire length change?

2. **All Specify Section Capabilities Should be Available for Gates**
   There should be some concept of pseudo gate so

16. This is the lesson learned from almost a century of aircraft design first in wind tunnels and now by computer simulation. Simulation is useless and will not prevent aircraft failure unless constantly connected to and improved from test pilot experience and measurements of manufactured airplanes.

17. See Meyer[1993b] for a discussion of changing Verilog behavioral features.

that timing checks and possibly path pulse checking and other delay calculator capabilities would be available by coding some kind of special gate. A debugging library then would include models coded with checking pseudo gates while a verification library would not include any specify section function checking bodies. Also, PLI tf_ style calls and some construct for interactive back annotation should be added for specify section functionality pseudo gates.

3. **Add Spike and Hazard Analysis While Keeping Inertial Delays.**
   Spike is defined as a pulse that is so narrow that it may or may not cause a gate to switch (Szygenda[1972]). Since most processing required to perform spike analysis is needed for Verilog's inertial delays, adding this feature to produce timing check like warnings and at time of detection unknown (x) injection would improve gate modeling accuracy. Inertial only delay modeling should be kept since it offers a good compromise between simulation efficiency and accuracy. It allows rapid changes to unknowns when switching problems occur and has the advantage that no list searching is required in the gate evaluate-schedule-propagate loop. Other more accurate delay model either require more storage per wire bit, or require scheduling algorithms that take time "roughly" proportional to fan-in and/or fan-out.

4. **Change to C Language Style Preprocessor**
   It is currently difficult to quickly modify a gate level net list because it is not possible to simply define a textual macro substitution. For example, given the following PLI task call:

   ```
   $a_long_lost_user_pli_task(xx);
   ```

   it is not possible to replace it with a call to a dummy Verilog task because it is missing the leading back quote macro indicator. At best, one must plan for symbols that may need substitutions by using a macro from the start. Many of the net list problems and library problems caused by multiple models with the same name are caused by limitations in current macro preprocessor semantics. For efficiency reasons, the C style preprocessor should be built into the language so that one pass parsing remains possible.

6. References

McGregor[1990] McGregor, J., and Korson, T. (eds.) Entire issue on object oriented programming. *CACM.* 33, 9 (September 1990), 39-159.

Bose[1977] Bose, A., and Szygenda, S. Detection of static and dynamic hazards in logic nets. Proceedings 14th Design Automation Conference, June 1977, 220.

Brooks[1989] Brooks, F. No silver bullet: essense and accidents of software engineering. *IEEE Computer*, 20,4 (April 1987) 10-19.

deGeus[1993] de Geus, A." 1,000,000 gate asic? Not with present EDA tools. IEEE International ASIC conference, 1993.

Dell'oca[1988] Dell'oca, C. Gate array technology. Proceedings IEEE ICCD, 1988, 296-299.

Dhimant[1990] Dhimant, P. Charms: characterization and modeling system for accurate delay prediction of ASIC designs. Proceeding IEEE Custom Integrated Circuits Conference, 1990, 9.5.1-9.5.6.

Dreyfus[1986] Dreyfus, H., and Dreyfus, S., *Mind over Machine.* MacMillan Inc. Free Press, 1986.

Earley[1973] Earley, J. Relational level data structures in programming languages. *Acta Informatica*, vol. 2, (1973), 293.

Earley[1975] Earley, J. High level iterators and a method for automatically designing data structure representation. *J. Computer Languages*, vol. 1 (1975), 321-342.

EIA[1988] Electronic Industries Association, "EDIF design interchange format 2 0 0", ANSI/EIA-548-1989, march 1988.

Fetzer[1988] Fetzer, J. Program verification: the very idea. *CACM.* 31, 9 (September 1988) 1048-1063.

Floyd[1989] Floyd, M. Hindsight and the crystal ball (editorial), *Dr. Dobbs Journal*, 14, 12(December 1989) 6.

Guthery[1989] Guthery, S. Are the emperor's new clothes object oriented? *Dr. Dobbs Journal*, 14, 12(December 1989) 80-86.

Lakatos[1970] Lakatos, I. Falsification and methodology of scientific research programmes. In I. Lakatos and A. Musgrave (eds.), *Criticism and the Growth of Knowledge.* scientific research programmes. Cambridge, 1970, 91-196.

Lakatos[1976] Lakatos, I. *Proofs and Refutations.* Cambridge, 1976.

Lighthill[1972] Lighthill, J. Artificial intelligence - A general survey. Also known as the Lighthill Report. Cambridge University, July, 1972.

Meyer[1993a] Meyer, S. Against the Silicon Structures IC Design Methodology. In preparation, 1993.

Meyer[1993b] Meyer, S. The argument for leaving the Verilog language unchanged. Proceedings 2nd International Verilog HDL Conference, 1993, 154-159.

Nagel[1975] Nagel, L. SPICE2: a computer program to simulate semiconductor circuits. ERL Memo ERL-520, University of California, Berkeley, May 1975.

Newell[1983] Newell, S.B., de Geus, A.J., and Rohrer, R.A. Design automation for integrated circuits. *Science*, 220 (4956) (29 April 1983) 465-471.

OVI[1993] Open Verilog International, Verilog Hardware Description Language Reference Manual. Release 2.0, March, 1993.

Popper[1959] Popper, K. R. *The Logic of Scientific Discovery*. Hutchinson: London, 1959.

Szygenda[1972] Syzgenda, S, Tegas2-Anatomy of a general purpose test generation and simulation system for digital logic. Proceedings 9th Design Automation Workshop, June, 1972.

Wilkes[1990] Wilkes, M. It's all software, now. *CACM*. 33, 10 (October 1990) 19-21.

Winograd[1973] Winograd, T. A procedural model of language understanding. In R. Schank and K. Colby, (eds.) *Computer Models of Thought and Language.* W. H. Freeman Press, 1973.