This is a list of available technical reports, along with their abstracts.

Click on the title of a paper to download the text.

Abstracts are provided in Adobe Acrobat 3.0 format.

THE DESIGN AND IMPLEMENTATION OF A GRAMMAR-BASED DATA GENERATOR

DGL is a context-free grammar based language for generating test data. Although many of the features of DGL are implemented in a straightforward way, the implementation of several of the most important features is neither trivial nor obvious. Before one can understand the implementation of these features, however, it is necessary to understand the overall structure of the compiler and its output, which was designed to be flexible enough to incorporate new features easily. Variables and chains are two of the most important features of DGL, and also two of the trickiest features to implement. The run-time dictionaries, which are built into the C code generated by the compiler, are implemented as pure code rather than as table-lookup routines. The compiler itself is reasonably straightforward, except for the expansion of character sets and compile-time macros. These two features can cause the "multi-dimensional" expansion of a string, the implementation of which must be carefully designed.

GATEWAYS: A TECHNIQUE FOR ADDING EVENT-DRIVEN BEHAVIOR TO COMPILED UNIT-DELAY SIMULATIONS

The gateway technique is a method for switching segments of code into and out of the instruction stream. When added to the straight-line code generated by a compiled simulator, gateways can be used to enhance the performance of the generated code by switching only those segments of code that actually need to be executed into the instruction stream. The convergence algorithm is an oblivious compiled code algorithm that can be used with many different types of circuits, including cyclic asynchronous circuits. In its oblivious form, the convergence algorithm provides only modest gains in performance over interpreted event-driven simulation, but with the addition of gateways, the performance of the algorithm increases significantly. Experimental data shows that with gateways, the convergence algorithm runs in about 1/5th the time required for an interpreted event-driven simulation. Additional work has been done to reduce the amount of code generated by the convergence algorithm, and to enhance the locality of the code to improve its performance on machines with caches.

DYNAMIC FUNCTIONAL TESTING FOR VLSI CIRCUITS

Dynamic testing is the process of creating test-vectors during simulation and using the output of the simulator to guide the vector generation process. The two main problems of dynamic testing are the design of a high-level vector-generation language, and the design of the interface between the vector-generator and the simulator. Solutions to these two problems are presented. The paper discusses guidelines for designing a high-level vector generation language, and presents several examples written in the FHDL driver language which was designed according to these guidelines. The examples illustrate how dynamic testing can be used to simplify the verification of circuits at the functional level. The paper presents a solution to the interface problem which is designed around a special interface data structure. This data structure supports several different styles of vector generators and also supports the interactive debugging of circuits. The interface data structure also supports the independent simulation of subcircuit instances and the dynamic creation and simulation of new circuit instances.

THE FLORIDA HARDWARE DESIGN LANGUAGE

The Florida Hardware Design Language is an expandable language that was originally developed to support a Wafer Scale Integration project under way an the University of South Florida. FHDL currently supports the uniform specification of gates, high-level functional blocks, state machines, ROM and PLA contents, and parameterized functional blocks. The language is easily extendable and a number of enhancements are planned or currently under way.

Generating Test Data with Enhanced Context Free Grammars

Enhanced context free grammars are an effective means of generating test data, even for simple programs. Grammars can be used to create intelligent random samples of tests as well as to create exhaustive tests. Action routines and variables may be used to compute expected results and perform other actions that are difficult or impossible to perform with ordinary context free grammars. Grammars may be used to create subroutine test data that contains pointers and other binary data. Test grammars not only provide a method for improving software quality, but should prove to be the foundation of much future research.

USING GATEWAYS WITH LEVELIZED COMPILED SIMULATION

Although Levelized Compiled Code simulation performs well under moderate to high activity conditions, there are many circuits that exhibit extremely low activity rates, either overall, or in certain subcircuits. The techniques presented here allow event-driven behavior to be added to Levelized Compiled Code simulations, with the aim of improving the performance of circuits with very low activity. Two techniques are presented, one which can be applied selectively on a block-by-block basis, and one that can be applied selectively on a gate-by-gate basis. Both concepts are based on the concept of a gateway, or retargetable branch. Performance results are presented for both techniques.

HDL DRIVEN CHIP LAYOUT WITHIN THE FHDL DESIGN FRAMEWORK

Work is currently underway at the University of South Florida to integrate automated placement and routing with the language FHDL, for the purpose of general integrated circuit synthesis. Presented is the division of the system into two programs, which separates the tasks of layout synthesis to take advantage of CPU time and memory space. One of the over-riding principles of this work, has been to prevent any modifications of the FHDL format which would restrict the use of the language for multi-level compiled simulation. Some of the modifications which were required are presented. Most important of which is a scheme for the generation of zero/unit delay latch and flip-flop simulation models to match the I/O of actual cells.

MDCSIM: A COMPILED EVENT-DRIVEN MULTI-DELAY SIMULATOR

This paper describes a complied event driven logic simulator which allows gates to have delays that are integral multiples of some basic time unit. The nets and gates of a circuit are compiled into a routines that perform the evaluation of gates and process events. These routines also manage the current timing wheel slot and insert events into the appropriate future time slots. A threaded code implementation is used to reduce execution time and space. Experimental results have shown a 26% improvement in execution time for compiled simulation over a standard event driven simulator.

TWO NEW TECHNIQUES FOR COMPILED MULTI-DELAY SIMULATION

Two new techniques for compiled multi-delay simulation are presented, one which is event-driven and another which is based on the concept of levelized compiled simulation. Experimental results are presented which show a significant performance improvement for compiled event-driven simulation over interpreted event-driven simulation, although this improvement is somewhat less than would normally be expected. An analysis of both the compiled and interpretive simulators is presented that supports the experimental data. The effects of caching and locality of reference are presented for the compiled event-driven simulator. The performance enhancements for the non-event-driven technique are substantial, but this technique has the disadvantage of generating an enormous amount of code for some circuits. Suggestions for future research are also presented.

TECHNIQUES FOR MULTI-LEVEL COMPILED SIMULATION

This paper begins attacking the problem of multi-level compiled simulation by presenting a solution to the problem of independent compilation of subcircuits. The solution to this problem is an interface data-structure that contains a public section for I/O ports and a private section that contains internal signals and maintains internal states. One data structure is created for each instance of a subcircuit. The data structures are retained by the circuit-simulator that contains the instance, but are created by the simulator for the instance. This technique prevents changes in the subcircuit from forcing the recompilation of all circuits containing instances of the subcircuit. Since subcircuits can be compiled independently of one another, they can be compiled using different compilers. Furthermore any subroutine that complies with the interface specifications can be incorporated into the simulation. Translation tables are used to move one logic-system to another. Finally, the paper discusses compilation techniques for cyclic circuits.

OPTIMIZATION OF THE PARALLEL TECHNIQUE FOR COMPILED UNIT-DELAY SIMULATION

The parallel technique of compiled simulation is a purely compiled method for unit-delay simulation that is based on the concepts of levelized compiled simulation and parallel fault simulation. Although the parallel technique provides very rapid simulations with a reasonable amount of generated code, there are several opportunities for optimizing the generated code. This paper presents two optimization schemes which are called bit-field trimming and shift-elimination. Two different methods of shift elimination are presented. Performance results are presented for all optimization techniques. These results show that using both optimizations schemes together provides for an average performance improvement of 47% (over the unoptimized simulations) for the circuits tested.

AUTOMATIC ROUTING OF INTEGRATED CIRCUIT CONNECTIONS: A TUTORIAL

A tutorial on routing for non-CAD types.

THE SHADOW ALGORITHM: A SCHEDULING TECHNIQUE FOR BOTH COMPILED AND INTERPRETED SIMULATION

The shadow algorithm is designed to perform compiled event-driven unit-delay simulations of asynchronous sequential circuits. It has been specifically designed to take advantage of the instruction caches that are present in many of the latest workstations. The underlying mechanism is a dynamically created linked-list of environments called shadows. Each environment invokes a short code-segment to perform a portion of the simulation. Shadow-Algorithm simulations run in about 1/5th the time required for a conventional interpreted event-driven simulation. The shadow algorithm may also be run interpretively without generating a separate simulation model. This version of the algorithm runs in about 1/4th the time of a conventional interpretive simulation.

SCHEDULING HIGH-LEVEL BLOCKS FOR FUNCTIONAL SIMULATION*

This paper presents a method for scheduling high-level blocks for functional simulation under the assumptions that circuits may be cyclic (due to element grouping), and that blocks cannot be broken down into simpler elements. The solution presented here may simulate one block many times per clock period. Obtaining a minimal schedule for a cyclic circuit is shown to be NP-complete, and two approximation algorithms are presented, along with empirical data to evaluate their effectiveness.

THE COMPLEXITY OF DETECTING SYMMETRIC FUNCTIONS

The characterization of the symmetries of boolean functions is important both in automatic layout synthesis, and in automatic verification of manually created layouts. It is possible to characterize the symmetries of an n-input boolean function as an arbitrary subgroup, G, of Sn, the symmetric group of order n. Given an expression e, which represents an n-input boolean function F, and a subgroup G of Sn, the problem of whether F possesses symmetry G is an NP-complete problem. The concept of an orbit can be used to characterize the various types of symmetry for a specified number of inputs. This classification can then be used, along with a few partitioning rules to completely determine the symmetries of a boolean function. This technique requires that the truth-table of a function be completely enumerated, and thus has a running time proportional to 2n, where n is the number of inputs of the function. Some of the mathematical concepts presented to support the NP-completeness result have intriguing possibilities for circuit minimization.

COMPILED UNIT-DELAY SIMULATION FOR CYCLIC CIRCUITS

Three techniques are presented for handling cyclic circuits in a compiled unit-delay simulation. These techniques are based on the PC-set Method and the Parallel Technique of compiled unit-delay simulation. The first technique, called the Synchronous Parallel Technique, is applicable only to synchronous circuits, but provides significant performance improvements over interpreted unit-delay simulation. The second and third techniques, called the Convergence Algorithm and the Asynchronous Parallel Technique, are applicable to all circuits, both synchronous and asynchronous. The Convergence Algorithm, which is based on the PC-set Method, provides significant performance increases for some circuits, but performs poorly on others. The Asynchronous Parallel Technique performs rather poorly and is covered only briefly.

TECHNIQUES FOR UNIT-DELAY COMPILED SIMULATION

The PC-set method and the parallel technique are two methods for performing compiled unit-delay simulation. The PC-set method analyzes the network, determines the set of potential change times for each net, and generates gate simulations for each potential change. The parallel technique, which is based on the concept of parallel fault simulation, is faster and generates less code than the PC-method, but is is less flexible. Benchmark comparisons with interpreted event-driven simulation show a factor of 4 improvement for the PC-set method and a factor of ten improvement for the parallel technique.

TWO NEW TECHNIQUES FOR UNIT-DELAY COMPILED SIMULATION

The PC-set method and the parallel technique are two methods for generating compiled unit-delay simulations of acyclic circuits. The PC-set method analyzes the network, determines the set of potential change times for each net, and generates gate simulations for each potential change. The parallel technique, which is based on the concept of bit-parallel simulation, is faster and generates less code than the PC-set method, but is not amenable to data-parallel simulation of multiple input vectors. Both techniques are based on the well known levelization algorithm used to generate zero-delay Levelized Compiled Code simulations. Although the parallel technique provides for efficient simulations with a reasonable amount of generated code, there are several opportunities for optimization. The two optimization schemes presented here are called bit-field trimming and shift-elimination. Two different methods of shift elimination are presented, one which is based on breaking cycles in an undirected graph, and one which is based on tracing paths through a network. Performance results are presented for both simulation techniques, and for all optimizations. Comparisons with interpreted event-driven simulation using the ISCAS 85 benchmarks show a factor of 4 improvement for the PC-set method and a factor of ten improvement for the parallel technique. Similar studies for the optimization schemes show an average performance improvement of 47% over the unoptimized simulations.

COMPILED UNIT-DELAY SIMULATION FOR CYCLIC CIRCUITS

The convergence algorithm is capable of generating a compiled unit-delay simulation for any cyclic circuit. The algorithm analyzes the circuit structure to created a sequence of subsets T0, T1, ... , Tn. The subset Tk contains the gates that will be executed at time k. Because the number of gates in the circuit is finite, this sequence will eventually converge. When the convergence point is reached, the sequence is used to generate code for the circuit. Code is generated in two segments called the prefix section and the iterative section. Bounds on the lengths of the two segments are given in terms of the structure of the circuit.

THE FHDL MACRO PROCESSOR

The FHDL (Florida Hardware Design Language) Macro processor provides a mechanism for extending the language features provided by the other components of the FHDL system (the ROM language, the PLA language, and the logic specification language). The primary use of the Macro processor is to provide flexible cells, such as ripple-carry adders, that can expand to match the size of the interface. The use of the Macro processor for this purpose is transparent with more standard hierarchical specification mechanisms. In addition, the Macro processor was designed to be an implementation vehicle for more sophisticated hardware specification and synthesis systems. The Macro processor provides most of the features found in other macro languages, and provides several new features that are found in few, if any, existing macro languages. The use of the Macro processor for high-level synthesis is the subject of much on-going research.

COSIM: AN EXPERIMENTAL CONCURRENT FAULT SIMULATOR

A concurrent fault simulator.

LECSIM: A LEVELIZED EVENT DRIVEN COMPILED LOGIC SIMULATOR

LECSIM is a highly efficient logic simulator which integrates the advantages of event driven interpretive simulation and levelized compiled simulation. Two techniques contribute to the high efficiency. First it employs the zero-delay simulation model with levelized event scheduling to eliminate most unnecessary evaluations. Second, it compiles the central event scheduler into simple local scheduling segments which reduces the overhead of event scheduling. Experimental results show that LECSIM runs about 8-77 time faster than traditional unit-delay event-driven interpretive simulator. LECSIM also provides the option of scheduling with respect to individual gates or with respect to fan-out free blocks. When the circuit is partitioned into fan-out free blocks, the speed increases by a factor of 2-3. With partitioning, the speed of LECSIM is only about 1.5-3.4 times slower than a levelized compiled simulation.

THE FHDL PLA TOOLS

The FHDL (Florida Hardware Design Language) PLA tools provide a means for specifying, simulating, and automatically laying out Programmed Logic Arrays (PLAs). These tools were created to facilitate VLSI design projects, to improve the quality of hardware design courses, and to serve as a basis for future research in VLSI design automation. At the specification level, the PLA tools allow the contents of a PLA to be specified as a set of logic equations. In addition, they provide features for facilitating the construction of PLA-based state machines. Once a PLA has been specified, it can be simulated at a high level in coordination with the simulation of the other portions of a VLSI design. After a PLA has been verified through simulation, it can be laid out automatically through an interface to the Berkeley PLA layout tools. The primary motivation for developing these tools was to provide a basis for future research in VLSI design automation.

THE FHDL ROM TOOLS

The FHDL (Florida Hardware Design Language) ROM tools provide a method for specifying, simulating, and automatically laying out ROMs. The primary focus of the ROM tools is on providing powerful methods for specifying microcode. Because the ROM tools were designed to support both VLSI design projects and other course work in hardware design, the ROM language contains many features that allow it to emulate other ROM programming languages. This allows students to complete laboratory exercises using a language that is similar to the one used in their textbook. Once the contents of a ROM have been specified, the ROM can be simulated concurrently with the simulation of the other hardware comprising the design. This allows designs to be debugged before they are fabricated. Once a design has been completely verified, the ROM can be laid out automatically and incorporated into a larger VLSI circuit. The automatic layout portion of the FHDL ROM tools is the subject of on-going research at the University of South Florida.

SCHEDULING BLOCKS FOR HIERARCHICAL COMPILED SIMULATION

Although preserving the hierarchy in compiled simulation can significantly reduce the compilation time for the code generated by the circuit compiler, the possibility of introducing pseudo-cycles due to element grouping can impair the performance of the generated code. A new approach to this problem is presented which uses dependency information to reduce the number of times a particular block must be simulated. The problem of determining the minimum schedule is shown to be NP-Complete, and a set of heuristics for the problem is presented. Experimental results for different combinations of these heuristics are presented. An algorithm for determining dependency information from the contents of a block is presented along with a new approach that can be used when the content of one or more blocks is unknown.