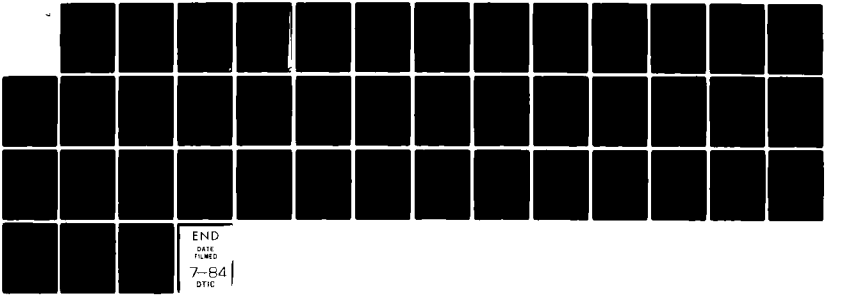
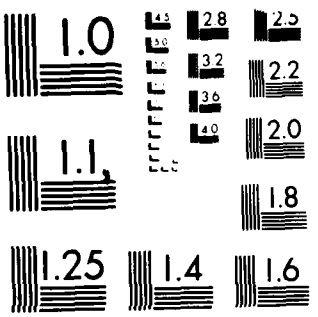
AD-A141 855 RESEARCH IN COMPUTER SIMULATION OF INTEGRATED CIRCUITS 1/1
(U) CALIFORNIA UNIV BERKELEY ELECTRONICS RESEARCH LAB
A R NEWTON ET AL. 31 JUL 83 AFOSR-TR-84-0447
UNCLASSIFIED AFOSR-82-0021 F/G 9/5 NL



END
DATE FILMED
7-84
DTIC



(11)

FINAL REPORT

Integrated Circuit Simulation for Very Large Circuits

Grant AFOSR-84-0021

Research in Computer Simulation of Integrated Circuits

Electronics Research Laboratory
University of California, Berkeley

by

A. Richard Newton and Donald O. Pederson

July 31, 1983

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) UNCLASSIFIED Unclassified

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFOSR-TR- 84-0447	2. GOVT ACCESSION NO. AD-A141855
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): Integrated Circuit Simulation for Very Large Circuits	5. TYPE OF REPORT & PERIOD COVERED: Final Report 1 Oct 81 - 31 Jul 83
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): A. Richard Newton, Donald O. Pederson	8. CONTRACT OR GRANT NUMBER(s): AFOSR-82-0021
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Electronics Research Laboratory, University of California, Berkeley, Berkeley, California 94720	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F, 2305/B1
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Office of Scientific Research/NE, Bolling AFB, Bldg. 410, Washington, D.C. 20332	12. REPORT DATE: 31 July 1983
	13. NUMBER OF PAGES: 23
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

"Performance Limits of the CLASSIE Circuit Simulation Program" The performance of the new LSI simulator CLASSIE is evaluated on several circuits with a few hundred to over one thousand semiconductor devices. A more accurate run time prediction formula has been found to be appropriate for circuit simulators. The design decisions for optimal performance under the constraints of the hardware (CRAY-1) are presented.

"Circuit Simulation on Vector Processors" Vector computers have an

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

increased potential for fast, accurate simulation at the transistor level of Large-Scale-Integrated Circuits. Design considerations for a new circuit simulator are developed based on the specifics of the vector computer architecture and of LSI circuits. The performance of the new LSI simulator, CLASSIE, is evaluated on a CRAY-1 vector-computer for several circuits with a few hundred to over a thousand semiconductor devices. Comments are given concerning the performance limits and relative hardware dependence.

"LSI Circuit Simulation on Attached Array Processors" The simulation of Large-Scale-Integrated (LSI) circuits requires very long run time on conventional circuit analysis programs such as SPICEZ and supermini computers. A new simulator for LSI circuits, CLASSIE, which takes advantage of circuit hierarchy and repetitiveness, and array processors capable of high-speed floating-point computation are a promising combination. The program development software environment of the Floating Point Systems 164 is evaluated based on the experience gained with the conversion of both SPICZ and CLASSIE to the machine. The FPS-164 has been used as an attached processor to a VAX 11/780 with the UNIX operating system. The performance of the two simulation programs on the host computer, the VAX, and the attached processor is compared. The FPS-164 architecture and Fortran compiler are evaluated by means of the speedup of CLASSIE compared to SPICEZ on the same processor.

"Data-Flow Based Behavioral-Level Simulation and Synthesis" While a large number of powerful design verification tools have been developed for IC design at the transistor and logic gate levels, there are very few silicon-oriented tools for architectural design and evaluation. As the number of gates which can be implemented on a single chip grows, these tools are becoming increasingly important. The FTL2 system described in this paper is an interactive system for specifying concurrent digital systems and analyzing their behavior. FTL2 differs from other behavior-level simulation systems in that the input specification for a circuit is a concurrent program. Specifications are incrementally compiled into augmented data-flow graphs which are then interpreted by a software data-flow machine. FTL2 includes special control structures for describing concurrent behavior in a structured fashion, a number of user-oriented input features, and an extensive macro facility. The concept of non-sharable resources is used to determine timing-dependent module access conflicts in a value independent manner. Incomplete specifications can be emulated and can be modified interactively. FTL2 has been implemented in LISP and is currently operational. A companion FLT2-based synthesis system is currently under development.

SUMMARY

Grant AFOSR-81-0021 became effective October 1, 1981 for research studies in [Project Title]. Although originally intended to support research activities over a 12 month period, because of the availability of complimentary support from industrial sources, it was possible to extend the research activities through July 31, 1983. In the following, the major activities associated with this research grant are summarized with particular attention to the research publications which have resulted.

For this research effort, it is a pleasure to acknowledge the cooperation received from Professor Donald Calahan of the University of Michigan, Ann Arbor. Professor Calahan was instrumental in establishing initial contacts with AFOSR for our grant. Over the years, Calahan's research group at Michigan and ours have worked closely together in the area of the effective use of parallelism and vector computation. Calahan's research, supported by AFOSR 80-0152, has been particularly helpful to us and is described in the two publications cited below.*

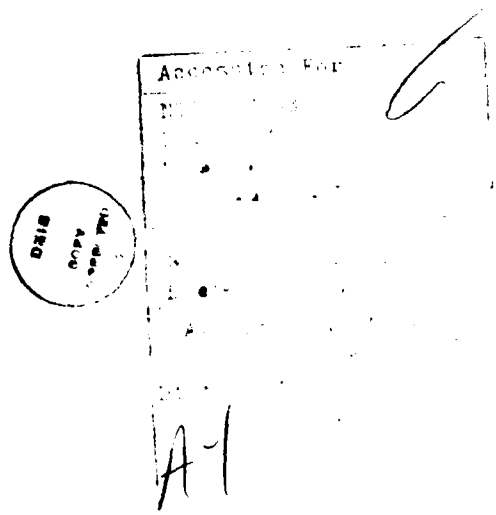
RESEARCH PERSONNEL

The research personnel associated with this grant include Professors Newton and Pederson together with Dr. Andrei Vladimirescu. During the initial period of the grant, Dr. Vladimirescu was completing his pre-doctoral studies, which were jointly supported by Bell Laboratories. In addition, he received extensive computer availability on a CRAY computer from United Information Services. His pre-doctoral work included the development of the circuit simula-

* D.A. Calahan, "Multilevel Vectorized Sparse Solution of LSI Circuits", Proc. ICCC '80.
* D.A. Calahan, "Decoupled Solution of Circuit Matrices on Pipelined Processes", Proc. ICCC '82, pp. 357 - ___

tor program CLASSIE.**

During the second year of the grant period, Dr. Vladimirescu continued as a post-doctoral scholar concentrating on the extension of his earlier research work to the use of array processors. Also in the second year of the grant, Professor Newton gave attention to the additional topic of behavioral simulation and synthesis based on data-flow machines.



** A. Vladimirescu and D.O. Pederson, "Circuit Simulation on Vector Processors", Proc. ICCC '82.

RESEARCH PUBLICATIONS

Four publications have resulted from this research grant. Three of these publications concern circuit simulation using vector computers or array processors while the fourth concerns behavioral simulation on advanced computers. In the following, abstracts of these papers are included.

"Performance Limits of the CLASSIE Circuit Simulation Program"

A. Vladimirescu and D.O. Pederson

Proc. ISCAS '82

Abstract:

The performance of the new LSI simulator CLASSIE is evaluated on several circuits with a few hundred to over one thousand semiconductor devices. A more accurate run time prediction formula has been found to be appropriate for circuit simulators. The design decisions for optimal performance under the constraints of the hardware (CRAY-1) are presented.

"Circuit Simulation on Vector Processors"

A. Vladimirescu and D.O. Pederson

Proc. ICCC '82

Abstract:

Vector computers have an increased potential for fast, accurate simulation at the transistor level of Large-Scale-Integrated Circuits. Design considerations for a new circuit simulator are developed based on the specifics of the vector computer architecture and of LSI circuits. The performance of the new LSI simulator, CLASSIE, is evaluated on a CRAY-1 vector-computer for several circuits with a few hundred to over a thousand semiconductor devices. Comments

are given concerning the performance limits and relative hardware dependence.

"LSI Circuit Simulation on Attached Array Processors"

Andrei Vladimirescu

Abstract:

The simulation of Large-Scale-Integrated (LSI) circuits requires very long run time on conventional circuit analysis programs such as SPICE2 and super-mini computers. A new simulator for LSI circuits, CLASSIE, which takes advantage of circuit hierarchy and repetitiveness, and array processors capable of high-speed floating-point computation are a promising combination.

The program development software environment of the Floating Point Systems 164 is evaluated based on the experience gained with the conversion of both SPICE2 and CLASSIE to the machine. The FPS-164 has been used as an attached processor to a VAX 11/780 with the UNIX operating system.

The performance of the two simulation programs on the host computer, the VAX, and the attached processor is compared. The FPS-164 architecture and Fortran compiler are evaluated by means of the speedup of CLASSIE compared to SPICE2 on the same processor.

"Data-Flow Based Behavioral-Level Simulation and Synthesis"

J.T. Deutsch and A.R. Newton

Proc. ICCAD '83

Abstract:

While a large number of powerful design verification tools have been developed for IC design at the transistor and logic gate levels, there are very few silicon-oriented tools for architectural design and evaluation. As the number of gates which can be implemented on a single chip grows, these tools are becom-

ing increasingly important.

The FTL2 system described in this paper is an interactive system for specifying concurrent digital systems and analyzing their behavior. FTL2 differs from other behavior-level simulation systems in that the input specification for a circuit is a concurrent program. Specifications are incrementally compiled into augmented data-flow graphs which are then interpreted by a software data-flow machine.

FTL2 includes special control structures for describing concurrent behavior in a structured fashion, a number of user-oriented input features, and an extensive macro facility. The concept of non-sharable resources is used to determine timing-dependent module access conflicts in a value independent manner. Incomplete specifications can be emulated and can be modified interactively.

FTL2 has been implemented in LISP and is currently operational. A companion FLT2-based synthesis system is currently under development.

PERFORMANCE LIMITS OF THE CLASSIE CIRCUIT SIMULATION PROGRAM

Andrei Vladimirescu and Donald O. Pederson

Department of Electrical Engineering and Computer Sciences,
Electronics Research Laboratory,
University of California, Berkeley, California 94720.

ABSTRACT:

The performance of the new LSI simulator CLASSIE is evaluated on several circuits with a few hundred to over one thousand semiconductor devices. A more accurate run time prediction formula has been found to be appropriate for circuit simulators. The design decisions for optimal performance under the constraints of the hardware (CRAY-1) are presented.

1. Introduction

CLASSIE [1], a simulation program for large-scale-integrated circuits, has been developed to narrow down the speed performance gap between a circuit simulator and a timing simulator given an SIMD (single instruction, multiple data) architecture of the host computer, e.g., CRAY-1. In spite an inherent increase in the execution speed of SPICE2 when run on the CRAY-1, an additional order of magnitude increase is needed for the efficient analysis of LSI circuits (greater than one thousand devices).

By using vector capability the speed improvement in CLASSIE is due to ordering operations which can be performed in parallel [2]. Thus, subcircuits with identical topology and semiconductor devices described by the same model are evaluated in the vector mode. In addition the use of generated machine code for the matrix solution practically reduces the contribution of this part of the simulation to less than one fifth of the total analysis time, even for very large circuits. Finally the data structure has been specifically tailored to accomodate vector operations with minimal gather/scatter overhead.

The development of CLASSIE has gone through several stages. A most important program is a vectorized version of SPICE (SPICEV) which contains vectorized device model routines and a scalar machine code solver for the sparse linear equations. This intermediate version does not have the data structure and border-block diagonal matrix solver of CLASSIE. Its speed performance is approximately half of that of CLASSIE and a few times faster than SPICE2 depending on circuit size.

2. Speed-Performance Evaluation

It is generally accepted that the number of semiconductor devices and nodes (equations) of a circuit are approximately equal or at least have a constant ratio and that the analysis time can be characterized by a generic number 'N' of devices or nodes. It is also generally accepted that analysis time increases with the power 1.2 - 1.4 of the circuit complexity 'N'. Circuits analyzed with CLASSIE on a CRAY-1 contradict both of the above generalizations.

A more accurate time model is necessary in order to predict the performance of a circuit simulator. Semiconductor devices are represented in most circuit simulators, such as SPICE2 [3], [4], [5], ADVICE [6], ASPEC [7], SLIC [8] and CLASSIE, by a variable number of equations depending on whether parasitic terminal resistances are specified or not. For this reason there is an arbitrary relation between the number of devices and that of nodes as brought out in the large circuit examples listed below.

Two parameters have been found to provide a rather accurate characterization of the simulation time. The parameters refer to the two major parts of the analysis: semiconductor-device-model evaluation (Jacobian terms) and linear-equation solution. Parameter t_d is the time for one model evaluation in one iteration. The second parameter is t_e the time for solving one equation in one iteration. The analysis time can then be estimated as:

$$T = a_{iter}(a_d \times t_d + a_e \times t_e) + \text{overhead} \qquad (1)$$

where a represents the number of entities designated by the subscript, i.e., a_{iter} is the iteration number, a_d the number of devices and a_e the number of equations. From above it is obvious that the speedup one can get in circuit simulation depends on how much the two characteristic times can be reduced. The particular circuit determines the relative weight of the two terms in parantheses. The analysis time for a subcircuit oriented program like CLASSIE can be expressed as:

$$T = a_{iter}[a_{si} \times t_{si} + a_{ss} \times t_{ss} + a_{ss} \times (a_{sd} \times t_{sd} + a_{se} \times t_{se})] \qquad (2)$$

$$+ \text{overhead}$$

where a_{si} is the number of subcircuits and the second subscript 'i' stands for interconnection while 's' stands for subcircuit. As shown previously t_e can be reduced greatly by code generation [1]. In order to reduce t_{si} t_{ss} and t_{sd} in the vector processor environment it is obvious that the devices have to be grouped either by subcircuit and/or by model and evaluated in parallel. The model evaluation can be broken down into actual computation, e.g., equivalent conductances and charges, and parameter gathering and individual admittance scattering into the circuit matrix. If the devices are grouped by models, only the computation part can be vectorized whereas if grouped by subcircuits, both computational and

CH1681-6/82/0000-1229 $00.75 © 1982 IEEE

gather/scatter can be vectorized. Based on this observation the device evaluation times t_e, t_{ge}, and t_{se} are different for the same LSI circuit.

The time savings in model evaluation are, however, not impressive because of several reasons. First, model routine vectorization in the CRAY-1 context increases the overhead in addition to the speedup. Device evaluation relies on different analytical formulations depending upon the operating region. The CRAY-1 Fortran compiler prohibits branching when vectorizing code and thus all the different alternative results must be computed for all devices and only in the end the appropriate data are assigned in the solution vector depending on each device.

Second, memory traffic can account for a major part of t_e. The indefinite admittance matrix of each device must be scattered into the circuit representation; in SPICEV this representation is the overall circuit matrix while in CLASSIE it is the interconnection matrix and the diagonal submatrices. The scatter into the overall and interconnection matrices cannot be vectorized and it has been measured with a CRAY-1 simulator that the scatter can take 75% of the total time of the bipolar transistor evaluation when full 64 element vectors are used [9]. The scatter into the diagonal submatrices corresponding to different instances of the same cell definition is the only one that can be vectorized. In this case the computation dominates over memory access.

Third, the speed in the computational part depends upon the vector length. Grouping devices by subcircuits, e.g., all transistors MX of the different instances of subcircuit SUB1, cuts down the scatter time since the results are stored in parallel in the diagonal submatrices. However, the vector length is shorter because the number of occurances of a subcircuit is less than the number of devices described by the same model. Experiments show that in MOSFET evaluation only two thirds of the computational speed with full vectors (64 elements for the CRAY-1) can be achieved for vectors with only 10 elements.

Fourth, no a priori knowledge can be used to eliminate certain computations, e.g., no advantage can be taken of the fact that the drain and source junctions of a MOSFET are reverse biased.

For close monitoring of speed the device routines in CLASSIE are segmented according to a specific task. The bipolar routines are separated to perform the gather/scatter, computation and limiting. The MOSFET routines perform gather/scatter, drain-source junction equation evaluation, static MOSFET equation evaluation, capacitance computation, bipolar and FET limiting.

3. Results

A number of large circuits as shown in Table 1 containing from a few hundred to over one thousand devices or equations have been analyzed.

TABLE 1

	Circuit	Type	Devices	Eqs	%Sparse
1	Adder4	Bipolar	288	450	99.3
2	Adder16	Bipolar	1152	1747	99.65
3	Filter	MOS	756	410	97.6
4	Special	MOS	268	683	99

The two adders are all-NAND circuits containing approximately 60% bipolar transistors and 40% diodes. The filter is an NMOS switched-capacitor lowpass filter containing 12 lowpass sections with two operational amplifiers per section. The special-purpose circuit accomplishes a desired function using digital (NAND, NOR) and analog (op-amp) blocks.

Table 2 shows some statistical data on the matrices in CLASSIE for the two bipolar adders having 'Inst' instances of the same NAND gate each. Four equations out of sixteen representing the NAND subcircuit correspond to external nodes. The external node contributions is gathered in the interconnection matrix. An interesting observation from the above is the high percentage of sparsity in the interconnection matrix.

TABLE 2

Circuit	Subckt	Inst	Eqs	%Sparse
Adder4	NAND	36	55	87.14
Adder16	NAND	144	163	94.45
-	NAND	-	16	62

From the data in Table 3 and Table 4 one can see the increase in total number of iterations and speed for the above circuits on SPICEV and CLASSIE as compared to a standard Fortran version of SPICE2 on the CRAY-1. Also displayed are the two characteristic times t_e and t_v.

TABLE 3

	Ckt Param	Add4	Add16	Filt	Sp Ckt
SPICE2	t_e(μs)	46	46	99	99
	t_v(μs)	47	47	47	47
	iter	8648	1619	2816	740
	speed	1	1	1	1
SPICEV	t_e(μs)	27	27	52	52
	t_v(μs)	4	4	6	4.5
	iter	27601	5307	6145	2549
	speed	3.2	3.5	2.2	3.4

TABLE 4
Adder4 Statistics on CLASSIE

	t_{ss}(μs)	t_{ss}(μs)	MFLOPS	iter	speed
Interconn	*	5.4	5.3	36219	4.2
Subckts	20	1.2	17.6		

The second subscript, s, in Table 4 becomes '1' in the first and 's' in the second line. The '*' in the t_{ss} column corresponding to the interconnection matrix indicates that all devices are part of the subcircuits. The analyses for the first three circuits of Table 3 and the analysis in Table 4 have been stopped after five minutes of CRAY-1 cpu time. The iteration count in the above two tables gives a measure of the speedup. In general, it is obvious that the larger the number of equations the more important the speedup.

The above table also shows the run-time statistics for the 4-bit adder on CLASSIE. A characteristic number for supercomputers is the megaflop rate (million floating point operations per second). The rates for the sparse equation solvers in CLASSIE are 17.6 MFLOPS at the subcircuit level (generated code uses vector registers) and 5.3 MFLOPS at the interconnection matrix (code uses scalar registers only). The second number is quite general for the scalar sparse-matrix-code solver. The high megaflop rate at the subcircuit level diminishes even more the relative importance of linear equation solution in the overall simulation.

4. Comments and Conclusions

As predicted by Eqs. 1 and 2 the run time dependence with circuit complexity is a linear function. The individual times, t_{ld}, t_{ss} and t_{ss} depend on the vector length used for different circuits. For an all-Fortran circuit simulator implementing sparse matrix techniques the time per equation solution t_s increases with a power greater than 1. This fact explains the increase in speedup with increasing complexity of the circuit.

Table 5 provides a measure of the relative importance of different simulator functions in CLASSIE versus SPICEV for BJT and MOSFET circuits based on the analysis time breakdown of two of the above circuits. The columns entitled (%)Adder4 or (%)Filter list the percentage of the total transient analysis time spent in a few key sections of the program.

TABLE 5

Task	CLASSIE	SPICEV	
	BJT (%)Adder4	MOSFET (%)Filter	BJT (%)Adder4
Gather/Scatter	16	32.5	37.5
Device Eqs.	38	9	13
Drain-Source Eq.	-	6.5	-
MOS Capacitance	-	2.5	-
Bipolar Limiting	10	2	7
FET Limiting	-	2.5	-
Eq. Solution	15	5	23

From the above data it can be seen that the routines which perform gathering of parameters and scattering of matrix terms from and to memory take more than one third of the total time in SPICEV because of the sequential mode in which they are executed. In CLASSIE however the analytical model evaluation has become dominant as desired. If analytical models are used the maximum speedup is already in the program for this case. For SPICEV the percentage of the equation evaluation is approximately 20% for bipolar and 23% for MOS circuits. This represents in itself an impressive performance of the computation speed of the mathematical functions (exponentiation, logarithm, square root) and floating point units of the CRAY-1.

More speed can be obtained by incorporating some intelligence into the program; this would allow for definition of long vectors in the computation part and regrouping according to subcircuits in the gather/scatter part. In CLASSIE the percentage time spent to link the subcircuits with the interconnection matrix is of the order of 5-10% percent.

Another important observation is the overhead time. From simulation runs it can be noticed that as the main parts of the analysis (device evaluation and equation solution) are made faster the importance of the overhead grows from approximately 5% for SPICE2 to over 20% for CLASSIE. The major part of the overhead is contributed by memory manager operations, convergence checking and truncation-error-timestep-control. In the Adder4 example iteration-count-timestep-control has been used thus its overhead is different from the one of the filter.

A major problem for CLASSIE can be a reduced number of occurances of identically-structured subcircuits which will increase the time for computation. There are two ways of avoiding this problem.

First, the vector registers of the CRAY-1 can be used differently in the computation part in comparison with the memory transfer part. As mentioned above devices can be grouped by model in the computation phase and then be loaded by subcircuit in the scattering phase. This approach can save a few percent by preventing the deterioration of the computation performance caused by shorter vectors.

Another source of speed improvement is the reduction of the overhead by recoding of the critical routines in Cray Assembly Language, e.g., some of the utility memory manager routines. This will save another 10% depending upon circuit and time step control method.

The use of table models for devices will reduce by up to 5-10% the overall 30-40% which the evaluation time of analytic equations contributes to the analysis time. The major advantage of table models in this context is that the same sequence of operations is performed for all devices regardless of the operating region. Tables will probably be used for the static part only as the charges will still have to be computed analytically. For a megaflop machine the use of table models for speed is of secondary importance.

A total speedup close to an order of magnitude can be predicted for a circuit with over one thousand devices/nodes simulated on CLASSIE relative to SPICE2 running on the CRAY-1.

Acknowledgement

This work has been sponsored by a grant from the Bell Laboratories Inc. Also acknowledged are the research grants ARO DAAG29-81-K-0021 and AFOSR 82-0021. The CRAY computer time made available by United Information Services is greatly appreciated.

8. References

[1] A.Vladimirescu and D.O.Pederson, "A Computer Program for the Analysis of LSI Circuits", *Proceedings* IEEE International Symposium on Circuits and Systems, Chicago, Illinois, April 1981.

[2] D.A.Calahan, "Multi-Level Vectorized Sparse Solution of LSI Circuits", *Proceedings* Int. Conference on Circuits and Computers, New York, Oct. 1980.

[3] L.W.Nagel, "*SPICE2 - A Computer Program to Simulate Semiconductor Circuits*", ERL Memo No. ERL-M520, University of California, Berkeley, May 1975.

[4] E.Cohen, "*Program Reference for SPICE2*", ERL Memo No. ERL-M592, University of California, Berkeley, June 1976.

[5] A.Vladimirescu, K. Zheng, A.R.Newton, D.O.Pederson, and A. Sangiovanni-Vincentelli, "*SPICE Version 2G User's Guide*", University of California, Berkeley, 10 Aug. 1981.

[6] L.W.Nagel, "ADVICE for Circuit Simulation", *Proceedings* IEEE International Symposium on Circuits and Systems, Houston, Texas, April 1980.

[7] P.Jenkins, ASPEC is available through ISD, Sunnyvale, California.

[8] H.Kop, P.Chuang, A.Lachner, and W.J.McCalla, "SLIC - a Comprehensive Nonlinear Circuit Simulation Program", *Conference Record*, Ninth Annual Asilomar Conference on Circuits, Systems, and Computers, Pacific Grove, California, Nov. 1975.

[9] D.A.Calahan, Private Communication.

CIRCUIT SIMULATION ON VECTOR PROCESSORS

Andrei Vladimirescu and Donald O. Pederson

Department of Electrical Engineering and Computer Sciences,
Electronics Research Laboratory,
University of California, Berkeley, California 94720

ABSTRACT:
Vector computers have an increased potential for fast, accurate simulation at the transistor level of Large-Scale-Integrated Circuits. Design considerations for a new circuit simulator are developed based on the specifics of the vector computer architecture and of LSI circuits. The performance of the new LSI simulator, CLASSIE, is evaluated on a CRAY-1 vector-computer for several circuits with from a few hundred to over one thousand semiconductor devices. Comments are given concerning the performance limits and relative hardware dependence.

1. Introduction

In recent years the need for detailed, electrical simulation for large circuits has initiated the research for algorithms which are faster for LSI circuits and preserve the same accuracy as in standard circuit simulation. This research has resulted in a number of prototype circuit and circuit/timing simulators such as MACRO [1], SLATE [2], and RELAX [3]. From an algorithmic point of view these programs can be classified as third-generation simulators. All these new programs use computers with conventional architectures.

The advent of vector computers, e.g. the CRAY-1 and the CYBER 205, which are capable of performing hundreds of millions of floating-point operations per second (Mflops), represents a second major factor that can be considered in the development of a computationally involved simulator. High execution rates for floating-point arithmetic are achieved with new architectures which perform each single instruction on a multiple data stream (SIMD) and through pipelining of instructions. The specifics of the SIMD architecture and of LSI circuits have been the two major design considerations in the development of another third-generation prototype circuit simulator, CLASSIE [4], [5].

CLASSIE is also intended to reduce the speed performance gap between a circuit simulator and a timing simulator. It is the only third generation simulator to this point which takes advantage of a parallel architecture of the host computer.

In the development of CLASSIE, an additional program has been used for a first characterization of the architecture. SPICEV is a version of SPICE 2G which contains vectorized device-model routines and a scalar machine-code solver for the sparse linear equations. Its speed performance is approximately half of that of CLASSIE and a few times faster than SPICE2, depending on circuit size.

2. Vector Processors

The CRAY-1, a 160 Mflops machine, and the CYBER 205, an 800 Mflops computer, have a number of common architectural characteristics. Both processors have an instruction buffer and decode unit, a scalar and a vector processing unit. Both have a large number of arithmetic functional units, 13 for the CRAY-1 and 11 (or 17) depending on configuration for the CYBER 205. In most of these functional units concurrent processes can take place. In scalar or vector computation a floating-point operation is partitioned into a number of segments and when an intermediate result is ready, it can be chained directly to other functional units. A resulting vector element is available at each clock cycle on the CRAY-1 and two each cycle on the CYBER 205.

Beyond these overall similarities there are specifics for each processor. The most important differences between the CRAY-1 and the CYBER 205 are the clock cycle, 12.5 ns versus 20 ns, the instruction set, hardware level versus microprogrammed instructions, and the memory, 4 Mword, 64 bits per word, versus virtual memory. Other differences include the number of processor registers, 72 address, 72 scalar and 8 vector registers of 64 elements for the CRAY-1 compared to 256 registers overall for the CYBER 205. The concurrency of operations can be higher on the CYBER because of the maximum of four floating-point pipes which are part of the vector unit and which can process an addition and a multiplication each at the same time.

For application programs it is important to observe the simultaneous use of add and multiply units, chaining of operations *from* a functional unit to another and the average vector length. The last parameter is an important measure of the efficiency of a vector operation which is associated with an important overhead called start-up time. The average vector length for which the vector processor reaches half the advertised speed is approximately 15 for the CRAY-1 and 50 for the CYBER 205. The former is faster for short vectors whereas the latter for long vectors (over one hundred elements).

For a scientific application program the most interesting performance measure is an equivalent Mflops rate which incorporates the memory traffic present in any algorithm implementation. An estimate of this number is derived below for a circuit simulator and cannot be expected to be larger than a fraction of the maximum processor speed.

3. Simulation of LSI Circuits

A basic consideration in the design of the new simulator is the object of the analysis. Only simple circuits had to be analyzed when SPICE was designed over ten years ago. These same circuits constitute today mere cells of a LSI system. For the purpose of the simulation a LSI circuit can be described usually as a collection of a limited number of structurally different functional blocks such as logic gates, operational amplifiers, etc., each block occuring more than once at the system level. The decomposition of the large circuit is the major source for speed improvement in third-generation circuit simulators.

The partitioning of an LSI/VLSI circuit into a cell / building block / system structure is useful information at any level of simulation. The analysis in CLASSIE is done at two levels, the system (building block) and the cell (subcircuit) level. The new program groups the cells described by the same definition together based on hierarchical tearing

CH1813-5/82/0000 0172$00.75 © 1982 IEEE

(derived from the input description) and solves each group in one pass through the same code.

An important guideline in the design for speed-up is that the number of items which define a vector be maximum. For this purpose a feature in the input language is provided to pass parameters to the cells with identical topology (described by the same definition). Another necessary feature for the convergence of large circuits is the ability to define initial conditions local to each cell instance.

Two major parts of a circuit simulator can be singled out as requiring most of the floating-point computation and therefore of the run time. These are the evaluation of the nonlinear characteristics of the semiconductor devices and the solution of the resulting linear equations. The impact of vetorization on these two major components is presented in this section as well as in the section commenting on results.

The speed-up of the semiconductor model evaluation is essential since it usually accounts for more than half of the total cpu time of MOSFET circuits even when these are large. The use of generated machine code for the matrix solution [5] practically reduces the contribution of this part of the simulation to less than one fifth of the total analysis time, even for very large circuits. The data structure has been specifically tailored to accommodate vector operations with minimal gather/scatter overhead.

Model evaluation can be partitioned in a number of tasks: A model parameter gather is followed by device initialization, terminal voltages initialization, equivalent conductance computation and in the end by an indefinite matrix terms scatter.

In CLASSIE as in SPICE2, semiconductor devices are described by geometrical features which are individual for each device, and general parameters, e.g., saturation current of a pn junction or threshold voltage for a MOS structure. The first task accomplished by the model routine and called model parameter *gather*, is to obtain the model parameters from memory. After each device is linearized a *scatter* operation takes place during which the device indefinite-admittance matrix is stored in the circuit matrix. These two operations account for more than half of the time spent in the model routines of SPICE2.

In the setup phase a device reordering takes place by hierarchical level (cell or system) and by model. The model parameters need thus be gathered only as many times as there are model definitions. The importance of the model parameter gather is made negligible in CLASSIE based on the fact that more than one device uses the same model parameters.

The initialization and scatter operations can be performed using vectors only for subcircuits of the same topology. For the interconnection circuitry these tasks are performed sequentially, device by device. The voltage-limiting and equivalent conductance computation use vector operations on devices grouped mainly by models.

An important issue is the definition of vectors throughout the model evaluation. For the transistors at the system level it is quite straightforward to define a vector across all devices which reference the same model. As has already been mentioned only the computation part is vectorized for these elements.

The different possibilities to define vectors can be presented best by the following example. Assume that a circuit contains 12 instances of a subcircuit OPAMP which in turn has 20 MOS transistors. The semiconductor devices at the subcircuit level can define a vector across all instances of that cell. Thus, the transistors named M09 in all 12 instances of the subcircuit OPAMP are linearized in one pass through the code. This results in 20 passes through the model evaluation code with a vector length of 12 each time. Although all tasks can be vectorized in this approach a longer vector can be used in the computation phase where all transistors of the same model and for all instances of the subcircuit can be grouped together. In the initialization and scatter tasks the longer vector used in computation is divided into a number of short vectors which contain as many elements as cell instances. The gain in this approach comes from a reduction in start-up times for more vector operation with shorter vector lengths. In the above example assume that 5 of the 20 transistors are depletion loads and are characterized by the same model parameters and that the remaining 15 are enhancement devices and are also described by a unique model. For the 12 instances the execution of the computation loop is reduced from 20 times to 4 times (once for the depletion devices and three times for the 180 enhancement devices) for a maximum vector length of 64.

Another trade-off in the design can be between a longer vector loop which performs also more computation than necessary or a number of shorter loops to which the execution is directed depending by analysis status flags. Both approaches lead to almost similar speeds.

As a final comment, the convergence check of the semiconductor devices is performed in the same manner as for the node voltages. The terminal voltages and device currents are compared in a vector loop and a vector with ones for the diverging elements and zeroes for the converging ones is set up. A fast vector accumulation library routine is then used for a fast result.

4. Results on the CRAY-1

A number of large circuits containing from a few hundred to over one thousand devices or equations have been analyzed. Two typical circuits are built of two cells, a bipolar NAND gate and an MOS operational amplifier, together with interconnection circuitry. The size of the circuit is easily varied changing the number of instances of the different cells. In the case of the MOS filter of Table 1 the circuitry at the system level (MOS switches and capacitors) also increases with complexity. A statistical description of the benchmarks is given in Table 1 from both the point of view of a flat representation as in SPICE2 and a two-level analysis as used by CLASSIE.

SPICE2					
Circuit	Dev	Eqs	%Sprs	%Mod Ev	%Eq Sol
Adder1	72	116	95.14	44.3 (64.1)	52.2 (8)
Adder4	288	450	99.3	40.7 (69.6)	57.2 (11)
Adder16	1152	1747	99.65	37.3 (83)	61.0 (9.7)
Lowpass Filter	70	42	83.89	73.4 (79.5)	11.9 (1.7)
	756	410	97.6	66.7 (75.6)	23.6 (2.5)
CLASSIE					
Adder1	0	16	64.06	69.7	14.2
Adder4	0	56	87.14	74.5	16.7
Adder16	0	163	94.45	75.7	18.6
Lowpass Filter	20	26	74.7	8.6	6.3
	181	157	93.3	86.9	7.1
Subckt	Dev	Eqs	%Sprs	Instances	
NAND	8	16	62	9, 36, 144	
OPAMP	25	16	45	2, 25	

Table 1

The three adders are all-NAND circuits containing approximately 60% bipolar transistors and 40% diodes. The filter is an NMOS switched-capacitor lowpass filter containing 10 lowpass sections with two operational amplifiers per section and two antialiasing and reconstruction circuits.

Four equations out of sixteen representing the NAND subcircuit correspond to external nodes. The OPAMP circuit has five external nodes. The external node contributions are gathered in the interconnection matrix. It is interesting to note that two cells of totally different function and complexity, 8 devices for the NAND gate versus 25 for the OPAMP, have the same size matrix representation.

In Table 1 can be found also the percentage contributions of model evaluation and linear equation solution to the simulation of the respective circuits. The data written in parantheses for SPICE2 are obtained from runs using scalar code generation. The increase in relative importance for the Fortran equation solution can be explained by both the increase of search with increasing complexity as well as by the reduction of the model evaluation as more devices are bypassed.

The analysis time per iteration for a subcircuit-oriented program such as CLASSIE can be expressed for only one subcircuit type as [5]:

$$T = T_i + n_{ss} \times T_s + \text{overhead} \qquad (1)$$

where T_i and T_s are the times for one iteration at the interconnection and the subcircuit level, respectively, n_{ss} the total number of subcircuit instances. T_i and T_s are a function of two characteristic times for a circuit simulator, t_d, the time for one model evaluation, and t_e, the time for solving one equation.

$$T = n_d \times t_d + n_e \times t_e \qquad (2)$$

where n_d and n_e are the number of devices and of equations, respectively, and a second subscript 'i' will stand for interconnection while 's' will stand for subcircuit.

Param	Add1	Add4	Add16	Lowpas	Filter
SPICE2					
$t_d(\mu s)$	51	45	48	121	99
$t_e(\mu s)$	38	41	47	30	47
iter	352	8648	1619	478	2816
speed	1	1	1	1	1
SPICEV					
$t_d(\mu s)$	27	27	27	58	52
$t_e(\mu s)$	4	4	4	4	6
iter	352	27601	5307	508	6145
speed	2.3	3.2	3.5	2.6	2.4

Table 2

Table 2 summarizes the two characteristic times t_d and t_e introduced earlier in this section, as well as the time per iteration and and the speedup factor between the two programs. A first observation from the above data is the very large effect of the scalar machine code generation which cuts the equation solution time by a factor of 8 to 12. Considering still the numbers referring to the equation solution it should be noticed that t_e increases with the number of equations for the Fortran solver whereas it differs very little in the machine-code solver. The difference in the machine-code solver can be explained on the basis that another factor becomes dominant, i.e., number of floating-point operations. This means that there are more floating-point operations per equation in average for the Filter. The effect of this number seems to be absorbed in other search and memory operations when the Fortran solver is used.

t_d is also a very important number. The model vectorization is seen to bring about a speedup around 1.5 for the bipolar mix (diode and BJT) and around a factor of 2 for the MOS circuit. The speedup for this part is not larger because more computation is performed to evaluate all possible formulations of equivalent conductances which depend on region of operation. This approach replaces branching in the vectorized computation by vector merge operations. Another factor is that the parameter gather and conductance scatter is not vectorized. The larger value of t_d for smaller circuits run on SPICE2 (see Table 2) is the result of less bypass than for a large circuit.

The speedup factor is influenced by several elements such as the ratio of t_d vs t_e and of n_d vs n_e. The speedup is larger for the bipolar circuits because the contribution of equation solution in all-Fortran SPICE2 (see Table 1) is much larger than for the MOS circuits. This part is reduced more effectively by the code solver than the model evaluation by vector operations.

CLASSIE	$t_{dx}(\mu s)$	$t_{ex}(\mu s)$	MFLOPS	speed-up
Adder1	*	44	5.1	4 / 4.3 / 4.2
Adder4	*	5.4	5.3	5 / 10 / 5.5
Adder16	*	6.7	5.4	5.4 / * / 6.8
NAND	20	1.2	17.6	*
Lowpass	58	5	5.3	2.6 / 2.9
Filter	52	6.6	5.2	3.7 / 5
OPAMP	29	3	14.5	*

Table 3

The evaluation of CLASSIE is presented in Table 3. The characteristic times of Eq. 2 are derived. As already mentioned the characteristic times differ between the interconnection circuitry and the subcircuits because of the gather/scatter which is part of t_{dx} where x stands for ther i for the interconnection and s for the subcircuits,respectively. The equation-solver characteristics are also different based on the use of vector code for the subcircuits and scalar for the interconnection.

The data in Table 3 should be viewed in connection with the circuit statistics presented in Table 1. An important specification is that the runs for this table have had just one parasitic resistance in the BJT model and none in the MOSFET model. The reduction in characteristic times for the subcircuit can be seen to be larger with increasing number of instances. From the analysis of the results from SPICEV it is expected that the speedup is larger for the adder circuits compared to the filters. A first observation relates to t_{di} for MOSFETs which is reduced by another factor of almost 2 compared to the time in SPICEV. The devices at the interconnection circuitry, 181 out of 758 MOSFETs, are still characterized by a t_{di} of 52 μs. The reduction in the t_{de} parameter for the bipolar mix (diodes and transistors) is closer to 25 to 30%.

Two numbers characterize the sparse solver; one is the parameter t_{ex} while the second is the Mflops rate. These numbers are computed from the run statistics which provide information such as the time for the subcircuit and interconnection solver, the total number of iterations, the number of operations for each subcircuit matrix, etc. The number of operations includes the add, subtract, multiply and divide because on the CRAY-1 these times are very close to each other; an addition/subtraction takes 6, a multiplication 7 and a reciprocal approximation 14 cp cycles. The Mflops rate is a better characteristic of the solver than the t_e parameter. The reason for it is that the operation count provides the best measure of the computational effort. This number proves to be stable for *different* sparsity patterns and is therefore a good characteristic of the sparse solver on the CRAY-1. Both for SPICEV and CLASSIE interconnection equations the scalar solver performs at 5.3 Mflops.

The vector solver is more dependent on the matrix structure and vector length (instances). The speed is between 14.5 - 17.6 Mflops which is impressive but is below

that predicted by Calahan [6]. As in the case of SPICEV the speedup for the MOSFET circuit is lower, 3.7, compared to 5 in the case of the Adder4 and 6 for the Adder16. Changing the mix between equations and devices by introducing parasitic series resistances in the models brings about higher speedups as predicted by Eq. 2. The three numbers given in the speed-up column in Table 3 for bipolar circuits correspond to one series resistance in the base, two in the base and collector, and three in the base, collector and emitter, respectively. The two data for the MOS circuits are with and without parasitic drain and source resistances. The speed-up of 10 for the Adder4 with two parasitic resistances is due to an additional reordering of the circuit equations performed by SPICE2 which increases considerably the number of fill-ins compared to CLASSIE.

Task	BJT Adder4		MOSFET Filter	
	CLASSIE (%)	SPICEV (%)	CLASSIE (%)	SPICEV (%)
Gather/Scatter	16	37.5	29	32.5
Device Eqs	46	13	17.3	9
D-S Junction Eq	-	-	11.3	8.5
MOS Capacitance	-	-	5	2.5
Bipolar Limiting	3	7	1	2
FET Limiting	-	-	4	2.5
Eq. Solution	16.7	16	6	5
Convergence Test	2	7	1	1

Table 4

In Table 4 the results of the choice of data structures, two-level analysis and other features of CLASSIE are compared with SPICEV task by task. From the above data it can be seen that the routines which perform gathering of parameters, initialization, and scattering of matrix terms from and to memory take more than one third of the total time in SPICEV because of the sequential mode in which they are executed. In the percentage for the above tasks is included also the contribution of the device linearization control. In CLASSIE however the analytical model evaluation has become dominant as desired. If analytical models are used the maximum speedup is already in the program for this part.

In CLASSIE the percentage time spent to link the subcircuits with the interconnection matrix is of the order of 5-10% percent.

Another important observation is the overhead time. From simulation runs it can be noticed that as the main parts of the analysis (device evaluation and equation solution) are made faster the importance of the overhead grows. The major part of the overhead is contributed by memory manager operations (moving blocks around) in SPICEV, are reduced to less than 5% in CLASSIE. Another source of concern are the time-step control computations. The adders use iteration-count while the filters use truncation error-time-step-control. There is not much time spent in the truncation error evaluation, since it has been vectorized. Its contribution is down from 15% in early versions of SPICEV to approximately less than 2% in CLASSIE.

A major problem for CLASSIE can be a reduced number of occurances of identically structured subcircuits which will increase the time for computation. The Lowpass section which achieves a vector length of only two in most vectorized code is simulated at half speed of SPICEV. Defining long vectors in model evaluation however reduces the run time on CLASSIE to that on SPICEV. This result in itself is very important because in the worst case of only 2 instances of a cell the two-level analysis of CLASSIE is equally fast to a vectorized SPICE. The performance of CLASSIE is expected to be superior for more than two instances of each cell type.

The use of table models for devices will reduce by up to 5-10% the overall 30-40% which the evaluation time of analytic equations contributes to the analysis time. The major advantage of table models in this context is that the same sequence of operations is performed for all devices regardless of the operating region. For a Mflops machine the use of table models for speed is of secondary importance. This is proven by a run of the Filter benchmark using the simple Shichman-Hodges model for MOSFETs. Because this model is so simple, its use provides a good estimate of table-lookup for dc characteristics. The percentage time spent in the 'Device Equation' part which computes the conductances associated only with the modeling of the transport in the inversion channel of Table 5 is reduced to 3% from 17.3%. This simple routine achieves approximately 60 Mflops.

5. Conclusion

The results presented in the last section suggest that the speed-up which CLASSIE offers compared to Fortran SPICE2 on the same computer is a function of the circuit. The two bounds on performance improvement are the model evaluation speed-up for a device linearization dominated simulation and the equation solution speed-up when this part is percentage-wise the most important.

CLASSIE can run on other vector and scalar computers but will not achieve the same speed-up as on the CRAY-1 comparative to SPICE2. It is estimated that changes in the data storage are necessary for optimal performance of CLASSIE on the CYBER 205.

In conclusion a total speedup of up to an order of magnitude can be predicted for a circuit with over one thousand devices (nodes) simulated on CLASSIE relative to SPICE2 running on the CRAY-1.

Acknowledgement

This work has been sponsored by a grant from the Bell Laboratories Inc. Also acknowledged are the research grants ARO DAAG29-81-K-0021 and AFOSR 82-0021. The CRAY computer time made available by United Information Services is greatly appreciated.

6. References

[1] N.G.B. Rabbat, A.L. Sangiovanni-Vincentelli, and H.Y. Hsieh, "A Multilevel Newton Algorithm with Macromodeling and Latency for the Analysis of Large-Scale Nonlinear Circuits in the Time Domain", *Trans. IEEE*, Vol. CAS-26, Sept. 1979.

[2] P. Yang, I.N. Hajj, and T.N. Trick, "SLATE: A Circuit Simulation Program with Latency Exploitation and Node Tearing", *Proc.*, Int. Conference on Circuits and Computers, New York, Oct. 1980.

[3] E. Lelarasmee, A.E. Ruehli, and A.L. Sangiovanni-Vincentelli, "The Waveform Relaxation Method for Time Domain Analysis of Large Scale Integrated Circuits", ERL Memo No. M81/75, June 1981.

[4] A. Vladimirescu and D.O. Pederson, "A Computer Program for the Analysis of LSI Circuits", *Proceedings*, IEEE International Symposium on Circuits and Systems, Chicago, Illinois, April 1981.

[5] A. Vladimirescu and D.O. Pederson, "Performance Limits of the CLASSIE Circuit Simulation Program", *Proceedings*, IEEE International Symposium on Circuits and Systems, Rome, Italy, May 1982.

[6] D.A. Calahan, "Multi-Level Vectorized Sparse Solution of LSI Circuits", *Proceedings*, Int. Conference on Circuits and Computers, New York, Oct. 1980.

ACKNOWLEDGEMENTS

I would like to thank Professor D. O. Pederson who has given me the opportunity to work on this topic as a post-doctorate research associate. The interaction with the members of the Computer-Aided-Design group, in particular discussions with Tom Quarles and Clem Cole, are acknowledged.

The helpful suggestions and support with up-to-date documentation from Ed Kushner and Steven Nakamoto from Floating Point Systems is greatly appreciated.

The support of the research grants AFOSR-81-0021 and SRC-82-11-008, and the interest of Analog Devices Semiconductor have been essential to the work presented in this report.

LSI Circuit Simulation on Attached Array Processors

Andrei Vladimirescu

ABSTRACT

The simulation of Large-Scale-Integrated (LSI) circuits requires very long run times on conventional circuit analysis programs such as SPICE2 and super-mini computers. A new simulator for LSI circuits, CLASSIE, which takes advantage of circuit hierarchy and repetitiveness, and array processors capable of high-speed floating-point computation are a promising combination.

The program development software environment of the Floating Point Systems 164 is evaluated based on the experience gained with the conversion of both SPICE2 and CLASSIE to this machine. The FPS-164 has been used as an attached processor to a VAX 11/780 with the UNIX operating system.

The performance of the two simulation programs on the host computer, the VAX, and the attached processor is compared. The FPS-164 architecture and Fortran compiler are evaluated by means of the speedup of CLASSIE compared to SPICE2 on the same processor.

CHAPTER 1

INTRODUCTION

The simulation of Large-Scale-Integrated (LSI) circuits requires very long run times on standard circuit analysis programs such as SPICE2 and standard hardware of the super-mini or main-frame computer class (0.5 to 2 Mips). A new simulator for LSI circuits, CLASSIE, has been developed recently [Vlad82] which is more efficient and preserves the same accuracy. This report describes the experience and results obtained when adapting SPICE2 and CLASSIE to a commercially available array processor, the Floating Point System 164, attached to a super-mini host computer, the VAX 11/780. As brought out later Cole [Cole83] has implemented a first version of SPICE2 on the FPS-164 attached to a VAX 11/780 with the UNIX operating system.

SPICE was developed over a decade ago for typical SSI circuits and scalar computers of the time. The program operates on an entire circuit which is processed at the individual electrical element level. Two basic factors of present technology have been considered in the design of the new LSI circuit simulator, CLASSIE. The first one is that LSI circuits are usually a collection of a limited number of structurally identical functional blocks such as logic gates, operational amplifiers, etc. The second factor is the availability of parallel computer architectures which provide an ideal environment for fast computations on repetitive structures. The analysis in the new program takes into consideration the hierarchy of the LSI circuit. The identical func-

tional blocks are grouped together and the simulation is performed at two-levels.

The above design considerations speed up the simulation of an LSI circuit performed by CLASSIE up to an order of magnitude compared to SPICE2 on a CRAY-1 super (vector) computer. From the point of view of the simulation speed for a large circuit on a vector computer CLASSIE rates between SPICE2 and a timing simulator.

The parallel architecture of the FPS-164 attached array processor is conceptually different from the CRAY-1; computationally intensive codes can be sped up however following the same basic concepts as in the case of the CRAY-1. The floating-point computation rates of the CRAY-1, the FPS-164 and the VAX 11/780 with a floating-point accelerator are 160, 12, and 1 Mflops, respectively. The speeds specified for the vector and array processor are estimates based on the assumption that more than one operation is processed at the same time. Thus, as a rule of thumb, a computationally intensive program such as a circuit simulator should run as many times faster on the parallel processors as specified by the raw speedup if the implementation takes full advantage of the architecture.

A general overview of the FPS-164 array processor (AP) is presented in Chapter 2. After a brief description of the architecture a critical view of the system and program development software available on the AP is presented.

Chapter 3 provides a closer look at the details of porting two circuit simulators, SPICE2 and CLASSIE, to the FPS-164. SPICE has been developed over the past 14 years with no specific computer architecture in mind while CLASSIE provides the same algorithms as the former program tailored for parallel processing.

A performance evaluation of the two programs follows. The execution speed of SPICE2 is compared to a general super-mini computer such as the VAX-11/780 while the speedup due to parallelism is emphasized for CLASSIE.

Conclusions on the implementation and performance of circuit simulation programs on the FPS-164 are the subject of Chapter 5. .

The work described in this report has been performed on an FPS-164 AP running the 'D' software release attached to a VAX 11/780 running release 4.1c BSD of the UNIX operating system.

CHAPTER 2

The FPS-164 Attached Processor

2.1. Introduction

This chapter provides a brief description of the FPS-164 processor. The architecture is outlined first with emphasis on the parallel processing features.

From a programmer's point of view the most important means to benefit from the architectural capabilities of a computer is its software environment. The second section takes a critical look at the two operation modes of the AP and the system and program development software. The main components of the program development software, e.g., the fortran compiler, debugger, mathematics library, etc., are evaluated. The experience gained from porting SPICE2 and CLASSIE to the FPS-164 is commented on wherever appropriate.

2.2. Hardware

The term array processor identifies a single peripheral processor with high-speed floating-point computation capability which can be attached to a general-purpose computer system. The tandem combination usually provides a much higher computation power than the host alone. Although the architectural synopsis and name can cause confusion with the vector computers the term array processor refers to a distinct category of pipelined Single-Instruction-Multiple-Data (SIMD) processors.

The Floating Point Systems AP-120B and FPS-164 are examples of commercially available array processors. The former is limited by a 38-bit word while the latter is better suited for scientific applications where a 64-bit data word is necessary. The architectural features [Char81] include multiple (eight) functional units, multiple (seven) high-speed data paths, two data register units of 32 registers each, up to 7.25 Mword main memory where data and instructions are stored separately, and a 167 ns cycle time. The functional units allow a maximum of two data computations, two memory accesses, an address computation, four data registers accesses, and a conditional branch to be initiated in a given CPU cycle.

The processor achieves performance through parallelism and/or pipelining. *A short pipe, 2 stages for the add and 3 for the multiply unit, characterize* the FPS-164. This design matches the clock cycle time and explains the difference in performance compared to the faster vector computers, CRAY-1 and CYBER 205. The short pipe has an advantage of providing most of the computation speed for a relatively short vector length, [Vlad82].

2.3. Software Environment

The two major components of the processor software are the system software used at run time and program development software which assists the conversion of a high-level language code into an executable module. The specifics of both components of the AP software are outlined in the following two sections.

2.3.1. System Software

There are two major operating systems available for the FPS-164, the Attached Processor EXecutive APEX and the Single Job Executive SJE. The two operating systems correspond to the two basic approaches of using the AP. Programs executing under APEX perform certain tasks on the host computer and other tasks on the AP. Input and output routines which interact with the user and perform more character-string operations rather than floating-point operations can be effectively run on the host. The computation intensive parts of the program will however run fastest on the AP. APEX controls the timely transmission of data between host and AP during the execution of the program.

Programs executing under SJE run on the AP only. The executable module together with the relevant data files are transferred to the AP before a run is initiated. Upon completion of the job the files of interest are transferred back to the host computer.

The conversion of SPICE2 and CLASSIE to FPS-164 run under SJE only. The AP works together with a VAX running the UNIX operating system.

2.3.2. Program Development Software

The software available for program development includes a fortran compiler, APFTN64, a linker, APLINK64, object module librarian, APLIBR64, symbolic debugger, APDEBUG64, assembler, APAL64, and mathematics library, APMATH64.

2.3.2.1. APFTN64

APFTN64 is a cross compiler which runs on the host computer and produces instructions which are executed on the attached processor. This is basically an F77 compiler with a number of extensions intended to utilize the parallel/pipelined architecture of the processor. There are several ways a programmer can take advantage of the architecture. One approach is through 5 different levels of optimization provided by the compiler.

OPT=0 implies the simplest compiler action where each fortran statement is treated individually; experience has been that at this level a program always works once it is operational.

OPT=1 signals the compiler that it can consider blocks of statements at one time for generating machine code; a block consists of consecutive statements which finish in a 'jump' or I/O instruction.

OPT=2 enables the compiler to try a global optimization across statement blocks as defined above.

OPT=3 adds pipelining to the above optimizations which exploit only parallelism; multiple elements of an array are processed by setting up one or two pipes through the functional unit(s).

OPT=4 is defined as 'unsafe code motion' and consists in moving invariant expressions outside the body of DO loop. As long as no 'zero-trip' loops occur in the program this level of optimization may provide an additional few percent of speed improvement.

The approach for writing fortran code which takes advantage of the architecture is similar to the guidelines followed for other parallel machines, e.g., the CRAY-1, [Vlad82]. A 'well-behaved' DO loop in which operations with array elements are performed is translated on all machines into a 'vector

operation'. The difference is that on the CRAY-1 the elements of an array are loaded into hardware vector registers and a vector operation is performed whereas on the FPS-164 a 2-3 stage pipeline is set up through the functional units.

Release D of the fortran compiler which has been used in this project has been found to generate incorrect code for OPT≥2. A typical symptom is that the attached processor hangs without being able to be initialized unless the host computer is rebooted. The compiler seems to fail to interpret correctly loops based on test and jump. Working code has been however generated for 'well-behaved' DO loops.

In some cases even OPT=1 can produce wrong code. The approach of tracing back the latter case is to locate the routine which does not execute properly and recompile it with a lower level of optimization. This failure mode does not hang the machine; it results just in an erroneous behaviour of some routines, e.g., SPICE2 prints an error message for a perfectly valid statement.

An useful option of the APFTN64 fortran compiler is which turns off the overflow/underflow interrupts generated during the execution of a user program. Unless this option is used for some of the device routines SPICE2 aborts when an underflow occurs.

Another criticism of APFTN64 when compared to another parallel processor fortran compiler, viz., the CRAY CFT [Cray80] fortran compiler, is its noncommunicative nature. No reports are provided to the programmer on the action taken on different loops or program blocks which can be converted into parallel code.

2.3.2.2. APLIBR64, APLINK64, APDEBUG64

APLIBR64 is an useful utility for creating an object program library. For large programs consisting of tens of modules it is a convenient way to store the valid object modules and to replace only the ones which have been changed.

APLINK64 is used to produce the executable module called the '*.img' file by convention. The linker accepts both individual object files and object libraries. A problem encountered with APLINK64 is the erratic terminator message of a bad block encountered in an object module which was successfully compiled and added to the library. This problem has been cured every time it has occurred by recompiling the flagged module and recreating the library.

A relevant option for the linker is -SYM which generates a symbol table needed by the symbolic debugger.

The symbolic debugger, APDEBUG64, is a very useful tool for program development. It is a quite powerful debugger similar in its description to the fortran debugger running under the VMS operating system. An accurate trace back including line numbers in the pertinent fortran files can be obtained. Some of the other features, e.g., examining values of local and global variables, setting breakpoints, etc., could not be tested due to difficulties encountered with opening the symbol table file. The documentation is very vague on this subject and various sensible approaches have lead to the same debugger message of not finding the symbol table file. In these situation it has been found to be faster to use just the trace back.

A conceptual drawback of the debugger is that it can be used only for modules compiled entirely with OPT=0. This restriction deprives the user of

any possibility of debugging parallel code which is the primary objective for this processor.

2.3.2.3. APMATH64

APMATH64 is a collection of mathematical functions which operate on arrays and scalars. In a number of situations it is advantageous to use these efficiently coded vector routines. These functions prove effective only when the vector length is sufficient to offset the start-up time of the routine. The programmer must judge this on a routine-by-routine case based on the time spent per array element. Thus, for VADD which adds the elements of two arrays and stores the result in a third array, it takes 15-30 elements in an array for achieving a 50% efficiency in the vector computation. In other words it takes that many elements such that the computation time equals the setup time for the function.

CHAPTER 3

SPICE2 and CLASSIE on the FPS-164

3.1. Introduction

A major application of the array processor is in the area of circuit simulation.

The problems encountered during the implementation of SPICE2 on the FPS-164 are outlined. Although SPICE2 could not be compiled at a higher optimization level than 1 its performance is very close to a commercially available program which is another version of the same code tuned for the FPS-164.

The results obtained in porting CLASSIE to the AP are very encouraging. The programming style used in CLASSIE is geared towards parallel architectures and thus the critical parts could be compiled successfully at the highest optimization level on a Fortran compiler still under development. A factor of two speedup has been achieved over SPICE2 running on the FPS-164 for a representative medium-size circuit, a four-bit adder.

In this chapter a number of data on CLASSIE and SPICE2 are presented. These numbers are obtained from runs on both scalar and vector computers. SPICE2 performs sequential operations on both types of computers and the speedup stems from the differences in computer architectures. All data which refer to CLASSIE reflect a sequential execution of statements on a scalar computer and parallel execution on a vector computer or array processor. For a small circuit of the basic cell type, e.g., a logic gate or an

operational amplifier, the only difference between CLASSIE and SPICE2 is a different data organization which becomes a source of speed difference.

3.2. SPICE2

3.2.1. Implementation Notes

The first program to be implemented on the FPS-164 attached to a VAX 11/780 running UNIX has been SPICE2 [Nage75], [Cohe76], [Vlad81], [Cole83]. In the following paragraphs a UNIX operating system is assumed for the VAX unless specified otherwise. This provision is important because SPICE2 compiled with the VMS Fortran compiler runs roughly twice as fast as when compiled with the UNIX f77 compiler. Cole in his work with the FPS-164 has not been concerned primarily with the simulator performance; the reported speedup of 3 for a typical circuit such as the UA741 has been obtained by compiling the program with APFTN64 using OPT=0. This version of SPICE2 runs on the AP under SJE, Single Job Executive.

The next step in porting SPICE2 to the AP has been to recompile the entire program using OPT=1. The executable generated in this way did not run properly causing messages such as *'LESS THAN TWO CONNECTIONS AT NODE X'* to be printed for a perfectly correct input. It has been found that by selectively recompiling the subroutines which perform the I/O in SPICE2, viz., READIN, RUNCON, DCOP, OVTPVT, PLOT, with OPT=0 while preserving the code of all other routines at OPT=1 a working executable can be obtained. Typically this code which is referred to as an 'OPT=1' version in spite of the above idiosyncrasies runs twice as fast as the 'OPT=0' SPICE2. The size of the 'image file' is reduced by one third from roughly 1.8 Mbytes to 1.2 Mbytes.

The attempt to use OPT=2 for just the computation-intensive routines such as the device model routines failed. The code generated in this way would typically hang the attached processor with no possibility of recovery short of rebooting the VAX. The only routines which have been successfully compiled at an optimization level higher than 1 are the equation solution routines, DCDCMP and DCSOL. Both have been compiled with OPT=3 and a working SPICE2 version has been generated. The speed improvement over the above 'OPT=1' version has been less than 10%. This latter SPICE2 version is referred to as 'OPT=1' in Table 3.1 and 3.2.

The best performance ever reported for SPICE2 on the FPS-164 is the commercially available program QSPICE [Shan83] which is typically 1.3 times faster than the best code obtained in this work. It is believed that for obtaining the above performance a number of the SPICE2 routines had to be rewritten to overcome the deficiencies in the APFTN64 compiler and to obtain correct code for OPT=3. Another difference in QSPICE is that the linear equation solving routines have been coded in APAL64, the FPS-164 assembly language. The small advantage in speed for QSPICE over SPICE2 proves that no compute-intensive part of the program can be pipelined. This difference stems mainly from a better control of the operand flow in the sparse equation solution coded in APAL64.

3.2.2. Performance

Table 3.1 summarizes the execution times of SPICE2 for four examples. The numbers given represent the time in cpu seconds needed for the transient analysis. The UA741 and Adder4 are bipolar circuits while MOSAMP2 and DECODER are an NMOS operational amplifier and a binary-to-octal decoder. A LEVEL=2 device model has been used in the analysis of the latter

Run Statistics					
Circuit	#Iter	VAX	FPS		Speedup
			OPT=0	OPT=1	
UA741	178	32.75	10.7	4.9	6.7
Adder4	2828	3614.2	1136.9	604	6
MOSAMP2	279	134.7	24.3	11.05	12.2
DECODER	978	1009.8	139.6	65	15.5

Circuit Statistics				
Circuit	#Eqs.	#Xtor	#Diode	#Device/Model
UA741	52	22	0	16 NPN, 6 PNP
Adder4	451	180	108	180 NPN, 108 DIOD
MOSAMP2	25	27	0	27 NMOS
DECODER	36	48	0	31 EMOS, 17 DMOS

Table 3.1. SPICE2 Run Time on FPS-164 and VAX 11/780

two circuits.

As a general remark on the performance improvement on the attached processor it can be stated that SPICE2 runs up to an order of magnitude faster than on a VAX 11/780 with floating-point accelerator and UNIX. For the two bipolar circuits the run times are typically 6 times faster and for the MOS circuit 12-15 times faster. The difference between bipolar circuits and MOS can be explained by the much larger percent time spent in the model evaluation for the latter compared to the former. The model evaluation seems to benefit more on the AP than the equation solution.

3.3. CLASSIE

3.3.1. Implementation

The implementation of CLASSIE has been helped by the experience gained from the SPICE2 conversion.

As a first step the VAX/UNIX version of CLASSIE has been implemented; this version differs from the high performance CRAY-1 version only in the model evaluation routines which do not take advantage of vectorization. This version had the same limitation on the optimization level used for the semiconductor device routines as SPICE2.

The next step included the conversion of the diode and bipolar vectorised model routines used on the CRAY-1 for the FPS-164. Conceptually the 'well-behaved' DO loops of the CRAY-1 CLASSIE code should produce an equally efficient code on the AP.

A first factor affecting the performance has been the multiple branching used for the multiple expressions of the semiconductor-device behaviour.

The usage of the vector merge function 'CVMGx' on the CRAY-1 has been replaced by IF statements inside the DO loop. An equivalent CVMGx statement function [Mart83] could have been used which would have contributed an 10-15% speed improvement in the device-evaluation speed. This improvement is estimated based on a typical vector length of 30.

A second factor has affected the performance of CLASSIE on the FPS-164 more significantly. It is known as the 'potential data dependency' problem which prohibits vectorization (pipelining) of a DO loop. Both in SPICE2 and CLASSIE all circuit data are managed in a large block of memory defined as an array VALUE (maximum_available_data_memory). Different data can be distinguished by table pointers. The compiler however does not know that there is no interaction between the data in two different tables within the same array. On the CRAY-1 there is a 'force vectorization' statement which can be placed in front of a loop. Release 'D' of APFTN64 does not have this feature. This problem could be noticed as soon as the most time-consuming modules have been compiled with OPT=3; there was no spectacular jump in performance which is expected when pipelining takes place. The speed improvement is between 2-4 per DO loop at OPT=3 compared to OPT=2. In the simpler forward and back substitution routines for the subcircuit matrices the above problem has been overcome by using the APMATH64 vector functions. This has resulted in a 23% speed improvement for this portion of the code only. The vector length for the above number is 36.

It is believed that all semiconductor-modelling routines could be compiled at OPT=4 in CLASSIE because of the programming style, 'well behaved' DO loops, and regular data structures. The equivalent routines in SPICE2 could not be compiled correctly for OPT>1. The equation-solving routines in

PROGRAM	OPT	DEV EVAL	EQN SOL	TRAN	DCOP
SPICE	1	301	329	625	21.4
CLASSIE	1	326	172	504	9.7
	2	273	167	451	9.1
	3	244	148	399	8.1
	4	209	112	324	7.8

Table 3.2. CLASSIE / SPICE2 Run Time Comparison

CLASSIE could be compiled at OPT=3 maximum.

The most aggravating experience during the implementation of CLASSIE has been the fact that user data can overwrite the AP's system software components or buffers thereof if there is not sufficient memory for loading the user program. In such cases a message from the linker or SJE would be helpful instead of getting a trace back leading into the system routines.

3.3.2. Performance

A running version of CLASSIE compiled with OPT=1 has been obtained in a similar way as SPICE2. On any computer, in scalar mode, CLASSIE gains 15-25% in speed over SPICE2 for medium circuits in transient analysis. Even for a small circuit, such as the UA741, CLASSIE is 20% faster than SPICE2 on the attached processor due to more regular data structures and the possible optimization associated with it.

In the DC operating point analysis CLASSIE is typically twice as fast as SPICE2 on medium circuits. The additional reordering process in DC analysis is performed on the interconnection and one subcircuit matrix for each subcircuit type in CLASSIE rather than a large overall matrix for the entire circuit in SPICE2. In transient analysis there is no reordering and this explains the smaller speed difference. These same speed ratios as above between CLASSIE and SPICE2 is found also on the FPS-164. The ratio between CLASSIE compiled with OPT=1 and OPT=0 is also about 2 on the array processor as in the case of SPICE2.

Table 3.2 lists the effects of the different optimization levels used in the compilation of SPICE2 and CLASSIE [Vlad83]. The times in seconds are for a transient analysis of the bipolar 4-bit adder circuit of 288 semiconductor

COMPUTER	DEV EVAL/LOAD	EQN SOL	OVERALL
CRAY-1	2.5	18	6
FPS-164	1.3	2.6	2

Table 3.3. CLASSIE/CRAY-1 vs. CLASSIE/FPS-164 Speedup

devices, 451 equations or 36 NAND subcircuits. The transient analysis has been performed from 0 to 350ns using the same input waveforms as described in [Vlad82]. It should be noticed that SPICE2 could not be compiled successfully at a higher optimization level than 1.

Table 3.3 shows the speedup which is obtained by running CLASSIE on the CRAY-1 and on the FPS-164. The speedup numbers are relative to the performance of SPICE 2G5 on the same hardware. The overall speedup on the FPS-164 could conceivably be improved to 3 if machine code generation would be implemented for the linear equation solution. The speedup in the device-evaluation part is estimated to be better if the Fortran compiler of the systems software release 'E' is used. This latest version of the compiler is advertised to have better pipelining capabilities than the earlier versions. All the above factors can narrow the gap of the speedup ratio between CRAY-1 and FPS-164 to roughly 1.5 in favor of the former.

CHAPTER 4

CONCLUSION

The evaluation of circuit simulation on a commercially available array processor has been the purpose of the work presented in this report. Both the better known SPICE2 simulator and the prototype simulator CLASSIE for LSI circuits have been ported to the FPS-164 array processor.

The FPS-164 is a promising processor for 64-bit floating-point scientific computations from a hardware architecture point of view. The experience gained porting the above mentioned programs shows that the available system software and program development software is relatively unfriendly and not sufficiently debugged. The reported work has been carried out using the Single Job Executive (SJE); under SJE the application program runs solely on the AP. Large scientific programs intended to run on the AP are written in Fortran; only a solid and well-debugged Fortran compiler will enable the user to take advantage of the speed offered by the underlying architecture.

The performance of the two programs on the AP is noteworthy. SPICE2 has been found to run from 6-14 times faster on the AP than on a UNIX VAX 11/780 with floating-point accelerator. This ratio figure is between 3-7 relative to the same VAX running VMS. CLASSIE runs roughly twice as fast as SPICE2 on the AP which brings the ratio between CLASSIE on the AP and CLASSIE on VAX/VMS close to 12; this is also the ratio between the Mflop rate of the two computers.

REFERENCES

[Char81] A.E.Charlesworth, "An Approach to Scientific Array Processing: The Architectural Design of the AP-120B/FPS-164 Family", *Computer*, Vol. 14, Sept. 1981.

[Cohe76] E.Cohen, "*Program Reference for SPICE2*", ERL Memo No. ERL-M592, University of California, Berkeley, June 1976.

[Cole83] C.T.Cole, "*Attaching an Array Processor in the UNIX Environment*", MS Report, University of California, Berkeley, April 1983.

[CRAY80] CRAY-1 Fortran (CFT) Reference Manual, Publication Number 2240009, CRAY Research, Incorporated, Mendota Heights, Minnesota, 1980.

[Mart83] C.M.Martell, "Eliminating 'IF' Statements to Allow Software Pipelining", *FPS Newsletter*, Portland, Oregon, 1983.

[Nage75] L.W. Nagel, "*SPICE2 - A Computer Program to Simulate Semiconductor Circuits*", ERL Memo No. ERL-M520, University of California, Berkeley, May 1975.

[Shan83] L.J.Shanbeck and R.S.Norin, "QSPICE: An Application of Array Processors to CAD Simulation" *Proc.*, IEEE International Conference on CAD, Santa Clara, California, Sep. 1983.

[Vlad81] A.Vladimirescu, K. Zhang, A.R.Newton, D.O.Pederson, and A.L. Sangiovanni-Vincentelli, "*SPICE Version 2G Users' Guide*", University of California, Berkeley, 10 Aug. 1981.

[Vlad82] A.Vladimirescu, "*LSI Circuit Simulation on Vector Computers*", ERL Memo No. UCB/ERL-M82/75, University of California, Berkeley, Oct 1982.

[Vlad83] A.Vladimirescu, "CLASSIE on Vector and Array Processors", *Late News Presentation*, IEEE International Conference on CAD, Santa Clara, Sep 1983.

DATA-FLOW BASED BEHAVIORAL-LEVEL SIMULATION AND SYNTHESIS

J. T. Deutsch and A. R. Newton

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, Ca., 94720

ABSTRACT

While a large number of powerful design verification tools have been developed for IC design at the transistor and logic gate levels, there are very few silicon-oriented tools for architectural design and evaluation. As the number of gates which can be implemented on a single chip grows, these tools are becoming increasingly important.

The FTL2 system described in this paper is an interactive system for specifying concurrent digital systems and analyzing their behavior. FTL2 differs from other behavioral-level simulation systems in that the input specification for a circuit is a concurrent program. Specifications are incrementally compiled into augmented data-flow graphs which are then interpreted by a software data-flow machine.

FTL2 includes special control structures for describing concurrent behavior in a structured fashion, a number of user-oriented input features, and an extensive macro facility. The concept of non-sharable resources is used to determine timing-dependent module access conflicts in a value independent manner. Incomplete specifications can be emulated and can be modified interactively.

FTL2 has been implemented in LISP, and is currently operational. A companion FTL2-based synthesis system is currently under development.

1. Behavioral-Level Simulation and Synthesis

Computer Aids have been used with great success in several stages of the integrated circuit design process. However, although many tools have been designed for electrical and logic level simulation and for automatic layout of semi-custom chips, little support is available for the beginning stages of a digital system design. Computer aids for behavioral-level specification and synthesis have been developed[Bar77a]. One reason is that many existing behavioral simulators do a poor job of abstracting concurrent behavior[Kan81a] but they are not widely used in the IC industry[Bre82a]. One reason is that many existing behavioral simulators do a poor job of abstracting concurrent behavior. Therefore, they provide little more information than can be obtained from a sequential description in a conventional programming language. In many cases this difference is not enough and every time a new system or chip is designed a new simulator is designed with it.

The FTL2 system addresses this problem by allowing specifications that combine the characteristics of *control flow* and *data flow* models to allow users to accurately evaluate the *concurrent algorithms* that a digital system represents before choosing a detailed implementation for it.

2. Describing Digital Systems

Digital systems can be described by net and component lists, by logic equations, by sequential or concurrent programs, by data-flow graphs or by systems of constraints. These methods differ in the degree to which they specify the structure of a specific implementation.

Standard electrical and logic simulators use the net and component list method. This type of description gives the structure of a particular implementation explicitly, but leaves the behavior of that system largely implicit. At the other extreme is the method of specifying digital circuits as as system of constraints. These constraints specify the what the system accepts as input and produces as output, but leave the procedures it goes through and the structure of any specific implementations implicit. The specification method provided by FTL2 lies in between these extremes. In FTL2 *systems are* described by programs for a software implementation of an augmented data-flow machine. In the data-flow model of computation, programs are described by directed graphs. The vertices of the graphs represent combinational functions, the edges of the graph represent communications paths between these functions. Augmented data-flow is an extension of the data-flow model which allows storage at the nodes and allows more general node firing rules[Dav82a]. Actually, this model is closer to the *dependence model* described in [Kuc81a] and some of the existing data-flow machines[Wat82a] than it is to the classical data-flow model[Den75a].

The text representation of FTL2 specifications is a sequence of functions and control structures called *forms*. Specifications are compiled into data-flow graphs by examining each form in sequence, determining the type of object the form represents, calling a function to compile the form, and repeating the process recursively for each form it contains. These data flow graphs are then passed to a software data-flow machine for evaluation.

The FTL2 software data-flow machine has nodes for primitive functions, for storage, and for sequential and parallel control. The concurrency provided is synchronous and deterministic. Therefore, explicit synchronization of parallel processes is not required and systems ranging from synchronous and deterministic to asynchronous and non-deterministic can be described and emulated.

In FTL2 it is possible to specify *explicitly* groups of operations to be performed either *sequentially* or in *parallel*. These specifications compile directly into special control nodes in the augmented data flow graph. Special functions are also provided for performing sequential and parallel array operations.

2.2 The Syntax of FTL2

A FTL2 description is made up of a collection of *forms*. A form consists of an open parenthesis, followed by zero or more elements (which themselves may be forms) followed by a close parenthesis. For example

```
(+ (element i a) (element j b))
```

is a form which contains two other forms, and whose value is a[i]+b[j].

2.2 Data-Types

The basic data-types that FTL2 provides are standard data types of most programming languages integer, floating point and strings, and arrays of these types. The basic form of a variable declaration is

```
(declare (<scope> (<type> |<name>|)))
```

Where <scope> is either local or global, type is integer, real or string, <name> is any alphanumeric name, and the braces mean that there can be zero or more occurrences of the syntactic unit inside of them.

2.3 Modules

Modules are the main partitioning facility provided in FTL2. The syntax for a module declaration is

```
(module <name>
 (declare (inputs |<inputs-names>|)
          (outputs |<outputs-names>|))
 |<forms>|
 )
```

Modules in FTL2 also provide a means for resource management. When declaration of a user-defined module is read, it defines a prototype of that module and creates a single instance with the same name. Whenever a module is invoked at runtime, FTL2 checks to determine if that module is in use. If an attempt is made to re-use a module which is already in use, an error is reported to the user. Thus, timing errors which result in resource conflict are detected and reported.

2.4 Macros

FTL2 provides a macro facility which can be used when copies of a module are needed, as well as cases where encapsulation without resource management is desired. Macros are declared as

```
(macro <name> (<parameter1>|<parameter>|)
       <forms>
)
```

3. Concurrency Modes

There are two concurrency modes in FTL2, lockstep and PC. These modes provide different ways of dealing with time and assignment. In Lockstep mode, every primitive operation, including assignment, is defined to take one unit of time. This mode has the advantage of always providing fixed times for operations, but has the disadvantage that balancing delays must be done by the user.

The other mode is called "PC" mode and combines the "Fetch-Store" style of assignment described in[Tow76a] with a hierarchal method of structuring time. In PC mode, each state is divided into two phases. Computation is performed during the first phase and registers are set to their new values during the second one. As an example, consider the following fragment.

```
(parallel
(set a (+ b (+ a c))
(set b a))
```

Under Lockstep mode, the value of b at the end of the block is the original value of a, call it old_a, and the value of a at the end of the block will be old_a + old_a + c. The reason for this is that b is set to the value of a by the second assignment statement before it is fetched by the first one. Under PC mode, the value of b at the end of the block is the original value of a, but the value of a is old_a + old_b + c.

PC mode also provides a different model of time. The statements in the main module of the user's description are defined execute one basic time unit apart. When necessary, time is broken up into finer units. The semantics of a parallel block are that all statements inside of it begin execution at that same time, and that the block is exited when all statements inside it are finished. The semantics of a serial block are that it executes the first statement in the block, waits for that statement to complete, and then executes the next statement in the block and continues in this way until all of the statements in the block have been executed. To preserve these semantics when a serial block is nested inside a parallel one, it is necessary that the inner serial block have a finer granularity of time than the outer one. Because the number of statements in the inner block are not known as the compiler is examining the input, time is treated as a mixed-radix floating-point number. Whenever a serial block occurs inside a parallel one, a new digit (of unknown radix) is added onto the least significant end of the state. Then, each element of the serial block is assigned a time (state) which is one greater in that digit position than the previous element. When hardware is actually generated the mixed radix floating-point numbers are converted to integer state numbers and the states are then re-coded using an algorithm such as[DeM83a] to generate an efficient state-assignment for a PLA-based controller.

4. Data-Flow Based Synthesis

The previous sections of this report have described the use of explicit concurrency in FTL2. It is also possible to use implicit concurrency and to let the FTL2 compiler determine where it is possible to have operations go on in parallel. The FTL2 compiler performs a data-flow analysis of the specification and determines the execution ordering of statements based on data-dependencies, as in[Pad80a].

Because FTL2 combines explicit concurrency and data-flow analysis it is able to detect errors which previous systems could not detect. These errors fall into two categories. First, the compiler may detect that two operations can go on in parallel in a case where the user has specified that they must proceed sequentially. Second, the compiler may detect that two operations are dependent and must proceed sequentially even though the user has specified that they are to go on in parallel. In the first case, there are two possibilities. One is that the designer has not recognized a possible optimization, and therefore may get a system with less than the highest possible performance. The other is that there is a constraint, perhaps critical, which the designer has left out of the specification. In the second case, the extra dependency and the serialization it requires may or may not be critical depending on whether or not it occurs on the critical path.

5. Simulator Implementation

Emulation of the augmented data-flow graphs in FTL2 is performed by passing messages between the nodes of the data-graphs. The data-flow graph of an FTL2 specification is unusual in that it is a tree, rather than a general cyclic graph. Sequential behavior is provided by sequential evaluation of sub-trees and looping is provided by repeated evaluation of sub-trees.

Concurrent message passing is simulated through the use of a central message queue and function dispatcher. The queue contains ordered (message, node) pairs. The main simulator loop removes pairs from the queue and calls the specified message handler with the given node as its argument. The side-effects of a message handler are highly restricted. A message handler for a node may only change the state of that node, the state of that node's parent, or send messages to other nodes.

When a module is defined, an augmented data-flow graph is created for it. When the module is invoked, the graph is checked to determine if it is already in use. If it is, an error message is printed which specifies where and when (in simulated time) the conflict occurred.

6. Results and Conclusions

The interactive top-level for FTL2 and the software augmented-data-flow machine have been implemented and the system has been used to describe and evaluate several designs, including the RISC microprocessor[Pat82a] and a perfect-shuffle network node chip. The FTL2 system is currently being used to describe a special-purpose augmented data-flow machine for performing iterated timing analysis[Sal83a, Kle82a].

7. Acknowledgements

Support from the Air Force Office of Scientific Research under grant 82-0021 and the US Army Research Office under grant DAAG29-81-K-0021 is gratefully acknowledged.

References

Bar77a.
M. Barbacci, *The ISPL Language*. Carnegie Mellon University Department of Computer Science (1977).

Kan81a.
S.K. Kang, "Automatic Generation of PLA Based Systems," *PhD Thesis*, Stanford University, (1981).

Bre82a.
M.A. Breuer, "A Survey of the State-of-the-Art of Design Automation," *Proceedings of the 19th Design Automation Conference*, p. 1 ACM, (1982).

Deu82a.
J.T. Deutsch, "Behavioral-Level Simulation and Synthesis of Digital Systems," *Master's Report*, University of California, (1982).

Kuc81a.
D.J. Kuck, R.H. Kuhn, D.A. Padua, B. Leasure, and M. Wolfe, "Dependence Graphs and Compiler Optimizations," *POPL Conference*, ACM, (1981).

Wat82a.
I. Watson and J. Gurd, "A Practical Data Flow Computer," *Computer Magazine*, pp. 51-57 IEEE, (February 1982).

Den75a.
J.B. Dennis and D.P. Misunas, "A Preliminary Architecture for a basic Dataflow Processor," *Proceedings Symposium on Computer Architecture*, pp. 126-132 IEEE, (1975).

Tow76a.
R. Towle, "Control and Data dependence for Program Transformation," *PhD Thesis*, University of Illinois Department of Computer Science, (March 1976).

DeM83a.
G. DeMicheli, A. Sangiovanni-Vincentelli, and T. Villa, "Computer-Aided Synthesis of PLA-Based Finite State Machines," *This proceedings*, (1983).

Pad80a.
D.A. Padua, D.J. Kuck, and D.H. Lawrie, "High-Speed Multiprocessors and [illegible]pilation Techniques," *Transactions on Computers*, pp. 763-776 IEEE, (September 1980).

Pat82a.
D.A. Patterson, "A RISCy Approach to Computer Design," *Compcon Digest of Papers*, pp. 8-14 IEEE, (1982).

Sal83a.
R.A. Saleh, J.E. Kleckner, and A.R. Newton, "Iterated Timing Analysis in SPLICE1," *This proceedings*, (1983).

Kle82a.
J.E. Kleckner, R.A. Saleh, and A.R. Newton, "Electrical Consistency in Schematic Simulation," *Proceedings of the ICCC*, pp. 30-33 IEEE, (1982).


DAT
ILM