LSI CIRCUIT SIMULATION ON ATTACHED ARRARY PROCESSORS

by

A. Vladimirescu

Memorandum No. UCB/ERL M84/6

20 January 1984

[illegible]

LSI CIRCUIT SIMULATION OF ATTACHED ARRAY PROCESSORS

by

Andrei Vladimirescu

Memorandum No. UCB/ERL M84/6

20 January 1984

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

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.

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.

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 functional 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.

E5C

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 vectorized 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.