CircuitSim93:
A circuit simulator benchmarking methodology case study

J.A. Barby and R. Guindi
Electrical and Computer Engineering
200 University Avenue West
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

phone (519)885-1211x3995, FAX (519)746-5195, jabarby@vlsi.UWaterloo.ca

Abstract — **A circuit simulator benchmarking methodology is developed that follows the philosophy that one wants to exercise each of the simulators on each of the benchmark circuits and make a fair comparison of their performance. This methodology was tested out in a benchmarking of 6 commercial circuit simulators from 3 CAE companies using a new circuit simulator benchmark suite called CircuitSim93.**

I. Introduction

The CircuitSim90 circuit simulator benchmark suite is one of the very few defacto standard circuit simulator benchmark suites that is in the public domain. It is also well know that there are serious problems with some of the netlists. Therefore, what to do if you have 6 circuit simulators from 3 different simulator companies and need to use CircuitSim90 to benchmark them?

This paper outlines the methodology used to benchmark Hspice H9007D and H92A (from MetaSoftware), Saber 3.1a and 3.1d (from Analogy) and Spectre 4.2 and 4.2.1a (from Cadence) using CircuitSim90 as the starting point. The result is a greatly improved benchmark suite referred to as CircuitSim93 and a set of tools to simplify the benchmarking operation.

II. Benchmarking

We start by stating our view of circuit simulator benchmarking.

A. The Aims/Goals Of A Benchmarking Effort

The aim or goal of a benchmarking effort is to make a fair comparison of the performance of each simulator. The result should be a side-by-side comparison of each simulator's performance in the key areas of interest. A customer wants the results to show (from a technical view point) which is the better simulator for the target application given the specifications on the hardware/software platform to be run on. A vendor wants the results to show where they have room for improvement.

B. A Benchmarking Philosophy And Comparison Criteria

The benchmarking philosophy used in the CircuitSim93 project was that one wants to exercise each of the simulators on each of the benchmark circuits and make a fair comparison of their performance. This is much easier said than done, as making a fair comparison of simulators that have very different modelling and simulation philosophies (and market positioning) can be difficult.

- Hspice is a general circuit simulation solution that works on a flattened circuit description. Its default calibration is for digital circuits.
- Spectre is a simulation engine for a CAE analog and mixed-signal design framework. Its default calibration is for analog circuits. It has very good netlist and model parameter debugging features.
- Saber is good for top-down design methodology. It is targetted to handle all the simulation needs of a full system design from architecture down to detailed circuit simulation results in an efficient timely way. Its measure is how fast can a designer go from concept to working system that meets specifications.

In fact, a benchmarking of such a range of simulators using a set of predesigned netlists, can never be a "fair" measure of each of the simulator's performance. Yes the benchmarking can measure how good a particular simulator is at a given task (which it may or may not have been optimized for), but it is not a measure of the simulator's true potential. We are looking at other benchmarking approaches for this case and the mixed-signal case.

One has to carefully think out the performance criteria to be measured in advance as a foundation for making their decisions on the actual benchmarking process. We now look at some details behind each of eight criteria used and explain why they are important measures given the high performance workstation commonly in use by todays designers.

The work described in this paper was supported by NSERC Research Grant OGP0003984 and Canadian Microelectronics Corporation research contract CON93/02127.

0-7803-1375-5/93$03.00 © 1993 IEEE

1) Times: user cpu time, system overhead cpu time, and elapse time. The user cpu time is a measure of simulation computing cost. However, the system overhead cpu time is useful to diagnosis inefficiencies in the program. The elapse time is becoming one of the most significant measures of a simulator, as designer time is the major cost in most design projects.

2) Required space: swap space, resident memory required for efficient cpu utilization, and temporary disk space (data and executable files).

Simulators differ greatly in their efficient use of virtual memory (swap space) and physical memory (resident memory). Good memory management will result in the two memory sizes being close and small. When a simulator requires more virtual memory than swap space available, a designer is stop dead in his tracks. However, if the resident memory required for efficient cpu utilization exceeds the actual physical memory available for user processes, the workstation goes into severe paging mode increasing the elapse time 2 or 3 orders of magnitude. Once a workstation gets into severe paging mode, the cpu sits idle most of the time and the simulation proceeds at the speed of the disk and memory cache rather than at the speed of the cpu. Given the large difference between cpu and disk access speeds in modern workstations, this is significant. Basically, the workstation physical memory needs to be as large as the CAE software requires or you are wasting designer time. Therefore, all other things being equal, a simulator that requires less resident memory for efficient operation is the preferred simulator.

A little known fact (even unknown to many users) is that certain circuit simulators require extra disk space at run time for either executables and/or temporary disk space. For large circuits, this required space is up in the 100s of Mbytes.

3) Page faults and swapping counts: In most workstations, paging is more of a problem than swapping. By monitoring these during a simulation, one can quickly tell if the resident memory is exceeding the available physical memory. It is only when both, the page fault count is high and the resident memory is high, one can determine more physical memory is required for a given simulator:netlist combination to run efficiently.

4) IO operations (reads and writes): Given the differential between cpu speed and disk speed, a good simulator will minimize its disk activity. If files are NFS mounted, the disk write cost is significantly greater than a disk read. These two give one a clear picture on how well thought out the simulator architecture is. Significant disk reads and writes will result in increased elapse time and reduced cpu utilization.

5) Accuracy/correctness of response: Most users assume that if a simulator completes without issuing an error or warning message, that the waveforms are correct. This is an incorrect assumption. Models, numerical analysis, algorithms, well thought out coding, and calibration settings all factor into this question. A measure of this is needed in any benchmarking or one ends up making an invalid comparison of simulator performance. The unfortunate side effect of this is that one has to run all the simulators on all the circuits before they can determine if it is an issue.

6) Calibration effort: Associated with the above is the need to calibrate a simulator. There are three major factors effecting a simulator's accuracy: calibration of the model parameters, calibration of the nonlinear solution algorithm and calibration of the integration step size control. One has to understand that all these simulators never drive for zero error, but rather go for acceptable error. The user has to adjust one or more parameters, which in some cases are interacting, to get the desired accuracy. From a user's point of view, it is desirable to have one parameter controlling the accuracy of a nonlinear solution and a separated (noninteracting) parameter controlling the integration step size control.

The calibration effort is a measure of skill and training level needed for an arbitrary designer to be able to effectly/efficiently use a given simulator.

7) Porting effort: Given most simulators have slightly different syntax and handle parameter ranges in different ways, one is always porting netlists as part of a benchmarking. The ease with which one can port a netlist to a simulator is a measure on how well thought out the simulator was and the quality of the tools that come with the simulator. This is all important as it translates into designer time.

8) Parameter checking: The simulator's parameter checking is a solid measure of how well thought out the simulator was from a designer's point of view. Many designers waste significant valuable time simulating circuits with model parameters that are inconsistent or seem unreasonable for a given technology. A well thought out simulator will warn a user if a model parameter seems strange and possibly stop the simulator if the model parameter exceed the range of possible values for a given technology.

III. CircuitSim90 CIRCUIT SIMULATOR BENCHMARK SUITE

The CircuitSim90 circuit simulator benchmark suite from MCNC is one of the very few defacto standard circuit simulator benchmark suites that is in the public domain. These "MCNC benchmarks" have gone far beyond their original intent. The CircuitSim90 set is basically a small subset of a test suite MCNC used internally for CAzM regression testing. CircuitSim90 is made-up of public domain circuits from MCNC and Sandia Labs [1].

It is well known by MCNC and anyone who has tried to use the CircuitSim90 benchmark suite (including Analogy, Cadence, Dazix and MetaSoftware), that there are multiple problems with the netlists and probably circuit responses. Shortly after the initial CircuitSim90 effort was completed, MCNC's CAzM development was suspended (along with CircuitSim90). The benchmark suite has been in a state of disrepair for many years. All those who have tried to use it have either given up or kept quiet on what they did. The case study below walks through the problems and solutions to CircuitSim90's disrepair resulting in a greatly improved suite referred to as CircuitSim93. The CircuitSim93 is available from the authors at no charge.

IV. A BENCHMARKING METHODOLOGY

The main difficulty with doing a fair benchmarking, is that the various simulators use different netlist syntax as well as have different command sequences and options. This always raises the question "are the netlists and command sequences equivalent"? This is actually a real problem with the CircuitSim90 benchmark suite, in that a number of the netlists are very large (see Tab. 1) and have different node and instance names for Hspice (mos2_large directory) from those for Spice2g6 (mos2_large_sp directory). This has two significant benchmarking problems associated with it:

(1) Given the size of the netlists, it is far too time consuming to prove the netlists are equivalent.

(2) The longer node and instance names require more swap and resident memory to store the names and associated lookup tables. In addition, the cpu run times will be effected as the name lookup times will vary some what with the length of the node and instance names.

To overcome these and other problems, a methodology is required to simplify things and allow a fair comparison of the simulators. This can be summarized in the following points used for the CircuitSim93 project:

(1) The netlists for the various simulators should be algorithmicly generated (from the CircuitSim90 netlists in the bjt, mos2, mos3, and mos2_large_sp directories) using Unix's "sed", "awk", and "csh" scripts along with Analogy's Spitos in the Saber case. These scripts give one a simple paper trail on each netlist and allow one to fix the CircuitSim90 netlists without actually changing the CircuitSim90 source (making it easy to prove using Unix's "diff" that the source used is the actual CircuitSim90 source). Thus, it is a simple process to show that the netlists used by each simulator are equivalent. In addition, the node, instance and subcircuit names in the netlist for each simulator are the same (within the syntax of the simulator), thus making sure the benchmark comparisons are fair.

(2) The (Unix) csh's "time" command measures most of the performance parameters of interest and reports them in a single output line which can be post processed with various scripts into tables for a benchmarking report. By using special scripts to run the various simulators and save the "time" information in files following a set naming convention, the benchmarking report tables can be automatically generated. This avoids the error prone and time consuming problems when they are manually generated.

(3) The Unix's Makefile facility with the necessary run time scripts makes gluing all this together a straight forward task and reduces the chance of human error.

V. A SUMMARY OF THE CASE STUDY

The CircuitSim93 project evaluated Hspice H9007D and H92A (from MetaSoftware), Saber 3.1a and 3.1d (from Analogy) and Cadence 4.2 and 4.2.1a (from Cadence) using MCNC's CircuitSim90 benchmark suite as a starting point.

The actual benchmarking runs were on a sparc10/41 with 264Mb swap and 96Mb memory in equivalent to single user mode. The binaries on 1.2 Gb disk were NFS mounted from a sparc10/41 and the user files on a 750 Mb disk were NFS mounted from a sparc2. The performance data was measured using csh's time command.

The application of the above benchmarking philosophy and methodology when applied to MCNC's CircuitSim90 benchmark suite resulted in the new CircuitSim93 circuit simulator benchmark suite with CircuitSim90 problems/bugs fixed. In this new suite, the Hspice, Spectre and Saber netlists are all generated from the original MCNC's CircuitSim90 spice2 netlists (bjt, mos2, mos3, and mos2_large_sp directories) using various scripts which supply the paper trail of the netlist conversion. The scripts (sed, awk, csh, and in the case of Saber, Analogy's Spitos product) are controlled via Makefiles.

One of the problems with the CircuitSim90 netlists is bad ".model" statements with obsolete or wrong model parameters. Since the various simulators do different things when they run across model parameters outside of normal bounds, the only fair thing to do, from a benchmarking point of view, is to adjust the erroneous model parameter to be within normal bounds. These adjustments are all cleanly documented in the various scripts. In addition, the scripts document the necessary mapping of obsolete model parameter names to correct model parameter names.

In general, Hspice and Saber issue a warning when they detected an unusual model parameter and use the default value. However, Spectre would normally issue a fatal error message and stop the simulation. Basically, the Spectre view is that if the user went to the trouble to specify a model parameter and the value is invalid, then the user should

correct the value or change the hard/soft limits associate with that parameter.

Since part of the benchmarking will include IO measurements and we are trying to make a fair comparison, the various scripts adjust the netlists commands to make this comparison fair.

As was mentioned, many of the netlists in CircuitSim90's mos2_large_sp directory are badly broken. These netlists are many Mbytes long with over 10,000 transistors (see Tab. 1). Manual corrections are not feasible. Following our benchmarking methodology, the corrections are algorithmic in nature. These errors can be grouped into:

(1) nodes with only one connection to them,
(2) nodes with no DC path to ground,
(3) nodes with nothing connected to them,
(4) multiple (thousands) of instances of various components with the same instance name.

Case (3) results in an error/warning message in the various simulators, however, cases (1), (2), and (4) result in error messages. The surprising case (3) happens in some of the netlists where a few subcircuits have external nodes to which nothing is connected internally or externally. Unfortunately, the most common occurrence of case (1) are gates of a MOS transistor, i.e. the MOS transistors have a floating gate connection.

Consistent with our benchmarking methodology, the above scripts were designed so that nodes with nothing connected to them will be deleted from the netlists (using awk scripts). Nodes without a DC path to ground or only one connection to them will have a 1e12 resistor (G_{min}) connected between them and ground (using sed). When multiple component instance names appear, they will be replaced using algorithmicly generated instance names (using awk).

Both Hspice and Saber tend to find nodes with no DC path to ground during run time, while Spectre's graph search picks them out when compiling the netlist. This resulted in Spectre finding all floating nodes in a couple of seconds (one run), while Hspice and Saber would take thousands of seconds per run to detect these nodes one at a time. In the project, we very quickly switched to using Spectre to debug the netlists and then find out how long it would take Hspice and Saber to find the first error.

As part of the CircuitSim93 project, the effort to port the spice2g6 netlists to the various simulators was documented. In addition, we kept track of which simulators could detect the above errors. These will appear in a future publication.

With oscillator circuits, not only is the operating point a problem, but they require noise to start the oscillation. This noise is naturally present in a real circuit but not necessarily in a simulated one. CircuitSim93 avoids both these problems by starting the oscillators with all nodes/current zero at time=0 and turns on the voltage supplies at the first integration time step (time=0^+). We refer to this as a trip=zero operation. This is accomplished by modifying all the sources to be " pwl(0 0 10e-12 ...)".

After making the above syntax and circuit corrections, most netlist:simulator combinations worked. A few cases initially failed during operating point analysis or DC transfer analysis. In keeping with our philosophy, we made minor run time option changes in these few cases to get the netlist:simulator combination to run. These changes were:

- trip=zero adjustment for H9007D:dac, H9007D:smult20, H92A:chip2, H92A:smult20, Saber:pc_frame, and Saber:sram.
- Spectre 4.2.1a required a nodeset to step around a numerical problem during the .op portion of the .tran in chip2.
- Saber required a reduced step size in the .dc of vreg.
- Saber required a different solution approach for the .dc in bias, schmitfast, and schmitslow.

The outputs/waveforms of each of the simulators were compared to each other and significant differences were detected. Most of the significant differences occurred in the analog circuits. We found that if a circuit had three possible operating points, the three simulators (Hspice, Saber, and Spectre) would all converge to different ones. This of course resulted in their transient responses being very different, invalidating the cpu performance measurements.

The above results were a clear indication that each simulator would have to be calibrated for each circuit and the performance measurements rerun. (This was partially unexpected.) It takes 1 sparc10/41 cpu month (in single user mode) just to rerun the performance measurements. In additional it would take one or two elapse months to adjust all the calibration parameters to force each of the simulators to generate the same response curve. The calibration and rerunning of the performance measurements is being left for phase two of the CircuitSim93 project.

ACKNOWLEDGEMENTS

The authors wish to thank Ken Kundert (Cadence), David Bedrosian (Analogy), and Ernst Christen (Analogy) for their insight into problems with the various circuits within the MCNC CircuitSim90 benchmark suite. Without their input, the CircuitSim93 project would not have been such a success. In addition, the authors wish to thank thank Bill Richards (MCNC) for his insight into the history of the CircuitSim90 project.

REFERENCES

[1] William R. Richards Jr., CircuitSim90, MCNC, Nov, 1992 [unpublished].

Tab. 1: CircuitSim93 circuit sizes

circuit	nodes	eqn	d	bjt	mos2	mos3	c	r	v	l	ml	tl	vct	vvt	a
bjt															
astabl	6	12	0	2	0	0	2	4	2	0	0	0	0	0	0
bias	12	55	0	13	0	0	0	5	4	0	0	0	0	0	0
bjtff	48	177	0	41	0	0	1	26	6	0	0	0	0	0	0
bjtinv	26	40	0	12	0	0	0	24	2	0	0	0	0	0	0
latch	19	65	0	14	0	0	0	10	4	0	0	0	0	0	0
loc	326	739	0	96	0	0	12	276	5	48	36	9	54	54	0
nagle	26	54	0	23	0	0	1	11	5	0	0	0	0	0	0
opampal	71	518	0	148	0	0	4	28	3	0	0	0	0	0	0
optrans	270	1860	0	528	0	0	19	148	6	0	0	0	0	0	0
rca	18	32	0	11	0	0	0	12	3	0	0	0	0	0	0
ring11	34	101	0	22	44	0	11	0	1	0	0	0	0	0	0
schmitecl	8	18	0	4	0	0	1	8	2	0	0	0	0	0	0
vreg	19	20	0	20	0	0	0	10	1	0	0	0	0	0	0
mos2															
ab_ac	25	28	0	0	31	0	22	1	3	0	0	0	0	0	0
ab_integ	28	32	0	0	31	0	24	3	4	0	0	0	0	0	0
ab_opamp	28	31	0	0	31	0	24	4	3	0	0	0	0	0	0
cram	32	44	0	0	60	0	42	0	12	0	0	0	0	0	0
e1480	145	204	49	0	28	0	17	130	3	0	0	0	0	0	0
g1310	66	97	28	0	14	0	21	56	3	0	0	0	0	0	0
gm6	7	20	0	0	5	0	0	0	3	0	0	0	0	0	0
hussamp	14	17	0	0	16	0	2	1	3	0	0	0	0	0	0
mosrect	6	10	0	0	4	0	0	2	2	2	1	0	0	0	0
mux8	30	42	0	0	64	0	29	0	12	0	0	0	0	0	0
nand	17	19	0	0	25	0	0	0	2	0	0	0	0	0	0
pump	3	4	0	0	1	0	2	1	1	0	0	0	0	0	0
reg0	15	16	0	0	0	0	13	30	1	0	0	0	6	0	0
ring	18	19	0	0	34	0	1	0	1	0	0	0	0	0	0
schmitfast	5	19	0	0	6	0	0	0	2	0	0	0	0	0	0
schmitslow	7	25	0	0	8	0	0	0	2	0	0	0	0	0	0
slowlatch	12	37	0	0	0	14	0	1	5	0	0	0	0	0	0
toronto	25	36	0	0	0	58	33	0	11	0	0	0	0	0	0
mos3															
arom	57	62	0	0	0	116	23	2	5	0	0	0	0	0	0
b330	163	856	0	0	0	330	0	0	33	0	0	0	0	0	0
counter	93	96	0	0	0	220	0	0	3	0	0	0	0	0	0
gm1	31	129	0	0	0	46	8	7	6	0	0	0	0	0	0
gm17	31	148	0	0	0	56	7	3	5	0	0	0	0	0	0
gm19	89	428	0	0	0	162	83	1	15	0	0	0	0	0	0
gm2	5	21	0	0	0	7	5	0	2	0	0	0	0	0	0
gm3	17	79	0	0	0	30	1	0	2	0	0	0	0	0	0
jge	180	243	0	0	0	348	157	1	63	0	0	0	0	0	0
mike2	11	38	0	0	0	12	1	0	5	0	0	0	0	0	1
rich3	51	56	0	0	0	106	12	2	5	0	0	0	0	0	0
todd3	13	43	0	0	0	13	0	1	6	0	0	0	0	0	2
mos2_large															
add20	521	2479	0	0	958	0	4091	0	42	0	0	0	0	0	0
add32	1058	1124	0	0	1984	0	800	0	66	0	0	0	0	0	0
chip2	9197	9218	0	0	18816	0	21548	0	21	0	0	0	0	0	0
dac	1094	6366	0	0	2635	0	11291	271	2	0	0	0	0	0	0
fadd32	161	178	0	0	288	0	25	0	17	0	0	0	0	0	0
mem_plus	2865	17788	0	0	7454	0	14274	0	15	0	0	0	0	0	0
pc_frame	6527	6568	0	0	14265	0	1796	21	41	0	0	0	0	0	0
pchip	407	2298	0	0	942	0	345	0	7	0	0	0	0	0	0
ram2k	4849	32632	0	0	13880	0	156	0	23	0	0	0	0	0	0
smult20	5591	28759	0	0	11559	0	34466	1	50	0	0	0	0	0	0
sqrt	515	2900	0	0	1188	0	1022	0	9	0	0	0	0	0	0
sram	343	2374	0	0	1008	0	24	0	15	0	0	0	0	0	0
voter	1708	1731	0	0	4243	0	460	1	23	0	0	0	0	0	0
voter25	43	51	0	0	74	0	0	0	8	0	0	0	0	0	0