INTERCONNECTING SPICE-PAC WITH OSA90/hope™

OSA-92-OS-5-R

May 27, 1992

© Optimization Systems Associates Inc. 1992

I. INTRODUCTION

Complex engineering problems may require us to combine circuit simulation with other software tools supporting optimization, statistical analysis, higher or lower level simulations. This report describes Spicepipe™, a pipe-ready executable child for OSA90/hope™. Spicepipe allows the user to explore SPICE-PAC simulation capabilities from within the OSA90/hope circuit design environment.

We introduce OSA90/hope [1] and SPICE-PAC [2]. We describe how we have used the high speed Datapipe™ technology [1, 3] featured in OSA90/hope to interconnect SPICE-PAC to OSA90/hope. We provide illustrative examples of the utilization of Spicepipe. We assume that the reader is familiar with OSA90/hope and that he or she understands the SPICE-PAC input file language. Detailed understanding of the Datapipe technology is not necessary.

Spicepipe, connecting SPICE-PAC to the OSA90/hope design environment augments OSA90/hope's circuit design capabilities of simulation and modelling with those available in SPICE-PAC. OSA90/hope, as a circuit design environment, provides the user with harmonic-balance frequency-domain simulation, statistical analysis and most importantly, very flexible and versatile optimization. SPICE-PAC, being an optimization-structured version of the popular circuit simulator SPICE [4] contributes time-domain simulation, noise analysis and additional device models. As a result, Spicepipe creates a mixed frequency-time-domain design environment. This environment features versatile simulation and optimization, statistical analysis and yield-driven design in both domains.

To interconnect SPICE-PAC with OSA90/hope we have used the OSA90/hope Datapipe technology and the Datapipe Server implemented with the help of the IPPC library for inter-program pipe communication [3]. The IPPC library employs the concept of UNIX pipes [5] which are fast interprocess communication channels. The channels interconnect OSA90/hope, called the parent process, with processes born from OSA90/hope, called its children. Spicepipe is simply an OSA90/hope child designed to organize and control the exchange of information between

OSA90/hope and SPICE-PAC. The Datapipe Server and the IPPC library are integral parts of OSA90/hope.

In Sections II and III we provide a short introduction to OSA90/hope and SPICE-PAC, respectively. Section IV describes the general structure of the interconnection. In Section V we discuss our implementation of the SPICE-PAC driver. Such a driver is required by SPICE-PAC in any application. Section VI includes three circuit examples. We optimize an LC transformer circuit to illustrate the communication between OSA90/hope and SPICE-PAC. An NMOS inverter circuit problem explores production yield estimation with time-domain specifications. The last example uses a second-order RLC circuit to demonstrate mixed frequency-time-domain analysis. Here, we first perform optimization of the nominal circuit and then continue with yield optimization. We used a Sun SPARCstation 1 running UNIX as the platform to perform the experiments. Our conclusions and acknowledgments are in Sections VI and VII, respectively.

II. OSA90/hope [1]

OSA90/hope - *O*ptimization *S*hell *A*ssembly/*h*armonic *o*ptimization *p*ersonal *e*nvironment is a Computer-Aided Design (CAD) system developed by Optimization System Associates Inc. OSA90/hope contains general purpose simulators for DC, AC and large-signal harmonic-balance analyses and several state-of-the-art optimizers. It features statistical Monte Carlo analysis and yield optimization capabilities. OSA90/hope is also equipped with the Datapipe Server [1, 3, 5]. The Datapipe Server can connect several external programs with the OSA90/hope internal simulators and/or optimizers (see Fig. 1). The external programs are then called child programs. We have used the Datapipe technology and the Datapipe Server to interconnect SPICE-PAC with OSA90/hope.

The user communicates with OSA90/hope by means of an input file. The input file describes the circuit under consideration as well as the operations requested by the user. The Datapipe connections are also defined in the input file. OSA90/hope contains a set of predefined Datapipe protocols [1]. The protocols are ready-to-use and the user does not need to access the OSA90/hope source code. Furthermore, the internal organization of the Datapipe transfer mechanism is transparent to the user. The only thing the user has to do is to define the input and output for the Datapipe in the corresponding Datapipe statement.

III. SPICE-PAC [2]

SPICE-PAC was developed by W.M. Zuberek of the Department of Computer Science, Memorial University, Newfoundland. It is a simulation package that is upwardly compatible with the popular SPICE circuit simulator [4]. SPICE-PAC accepts the same circuit description language as SPICE (with only a few minor exceptions) and provides the same circuit analyses. It also supports a number of extensions and refinements which are not available in the original SPICE program.

The main difference between SPICE and SPICE-PAC lies in the internal organization of the programs. While SPICE is a program with a fixed flow of operation, SPICE-PAC is a collection of loosely coupled simulation primitives. The simulation primitives can be composed in many different ways, according to a particular application. The user has to build a driver, using the primitives, which organizes the flow of operation so that the behaviour of the system meets the specifications. A description of the primitives and their functions is given in [6].

The modular structure of SPICE-PAC makes the package very attractive, especially for specific applications. SPICE-PAC is particularly useful in optimization-oriented applications, where its modular structure may significantly increase the efficiency of the system. While creating a driver for SPICE-PAC, the user can include in it any additional tasks, e.g., statistical post-processing or graphical output facilities.

The availability of SPICE-PAC is described in Appendix A.

IV. GENERAL STRUCTURE OF THE INTERCONNECTION

While interconnecting SPICE-PAC with OSA90/hope we concentrated our effort on designing the system in such a way that the user would not be required to do any programming. Furthermore, we assumed that the user should be able to invoke SPICE-PAC from OSA90/hope in a completely "invisible" manner. To this end we created a version of the SPICE-PAC driver which is linked, together with SPICE-PAC, to a short interfacing driver forming altogether Spicepipe: a pipe-ready executable child for OSA90/hope. The structure of connection between OSA90/hope and SPICE-PAC is shown in Fig. 2. The interfacing driver, see Fig. 2, was created on the basis of the general child template described in [1]. We extended that template by adding some error checking and dynamical memory allocation. The program is able to detect and report most common errors, e.g., syntax errors in the SPICE-PAC input file, unknown analysis types passed to SPICE-PAC, etc. The interfacing driver also performs necessary data type conversions.

The second Datapipe channel in Fig.2, is employed to help the user in dealing with two input files, one for SPICE-PAC and another one for OSA90/hope. Each of the systems requires its own input file. Having two input files is then necessary, but more importantly, to synchronise the activities of the systems the files have to be consistent. To allow the user to work with one input file only we created a separate pipe-ready executable child program for OSA90/hope, which we named *create_file*. *create_file*, if called from OSA90/hope through the Datapipe mechanism, will create a disk file, e.g., an input file for SPICE-PAC. The name of the file as well as its contents are both character string type arguments for the second Datapipe in Fig. 2. In other words, the user can define his or her SPICE-PAC input file inside the OSA90/hope input file as a character string.

For a detailed description of the Spicepipe and *create_file* pipe-ready executable child programs the user is referred to Biernacki *et al.* [7].

V. SPICE-PAC DRIVER

Here we discuss the structure of our version of the SPICE-PAC driver, introduced in Section III, and the analysis types available through this driver.

The structure of the driver is shown in Fig. 3. The two paths of operation flow in Fig. 3. correspond to the first and subsequent calls to SPICE-PAC. If the user wants to simulate a circuit using SPICE-PAC, there will be only one call to SPICE-PAC and the operation will follow the path for the first entry to SPICE-PAC. If the user wants to perform optimization or multiple simulations of a circuit, the second and all subsequent calls will skip the initialization operations of SPICE-PAC. Such organization significantly saves CPU time.

During initialization, SPICE-PAC first tries to open its input and output files. If the files have been successfully opened, SPICE-PAC will proceed to read the input file. Next, the program checks if the input data from OSA90/hope is consistent with definitions in the SPICE-PAC input file. The input data, if it exists, consists mainly of sweep or optimization variables. If the information is consistent, corresponding SPICE-PAC variables are defined and internal identifiers are assigned to them. Defining the temperature for subsequent analysis completes the initialization process.

The successive SPICE-PAC processing is performed in every call. First, the values of sweep or optimization variables are updated and then the requested analysis is performed. Before returning control to the connecting driver and then to OSA90/hope, SPICE-PAC saves the results of analysis in its output file. This takes place, however, only if the user requested SPICE-PAC to do so by setting an additional flag in the OSA90/hope input file.

Each action undertaken by the SPICE-PAC driver is performed by a call to an appropriate SPICE-PAC simulation primitive constituting a subroutine. If an error is detected upon a call to any of the invoked subroutines, SPICE-PAC sets the error flag and returns control to the interfacing driver immediately. The SPICE-PAC driver is intended to serve as general use of SPICE-PAC. The user, however, can extend this driver or adjust it to a particular application.

Our implementation of the SPICE-PAC driver allows the user to request one type of analysis at a time, out of the following: **DC** transfer curve analysis, **TRANSIENT** analysis, **AC** analysis, **NOISE** analysis, **DISTORTION** analysis or **FOURIER** analysis.

If two or more analyses are required the user has to create a separate Datapipe communication channel for each of the analysis types.

For a more detailed description of the SPICE-PAC driver used here refer to Biernacki *et al.* [7].

VI. APPLICATIONS

A. Optimization of an LC transformer

An LC transformer optimization is chosen to demonstrate the communication between OSA90/hope and SPICE-PAC. We want to optimize the modulus of the input reflection coefficient $|S_{11}|$ for the transformer in Fig. 4. We use 21 equally spaced points in the frequency range from 0.079578Hz to 0.187644Hz. All L and C elements in the circuit are optimizable. Input resistance R_{in}=3Ω and output resistance R_{out}=1Ω.

We use the *create_file* child to create the input file for SPICE-PAC. We use the expression processing capability of OSA90/hope to calculate reflection coefficient, providing relevant formulas in the input file. The maximum value of $|S_{11}|$ before optimization was 0.66. After 62 iterations, using the minimax optimizer it decreased to 0.076. The values of the L and C elements before and after optimization are listed in Table I. The diagrams of $|S_{11}|$ as a function of frequency before and after optimization are shown in Fig. 5.

We also solved the problem entirely by OSA90/hope. The results are practically the same. Small differences are most likely due to different numerical algorithms used in both simulators. The CPU times used running OSA90/hope with the Spicepipe connection to SPICE-PAC and OSA90/hope stand-alone were approximately the same.

B. Time-domain response and Monte Carlo analysis of an NMOS inverter

An NMOS inverter with depletion load [8], shown in Fig. 6, is used to illustrate the utilisation of Spicepipe to perform a Monte Carlo analysis with time-domain specifications. The Monte Carlo analysis is organized within the OSA90/hope design environment but the actual circuit simulations are performed by the SPICE-PAC time-domain simulator. We used the level 1 option of the SPICE-PAC MOS transistor model [4, 8] to model the transistors. We selected: channel length, channel width, threshold voltage and transconductance of both load and inverter transistors as the statistical parameters. In reality, production variations of the threshold voltage and transconductance are the most notable ones. We assumed normal distributions of the parameters and identity correlation matrix for simplicity. See Table II for transistor model parameters and statistical distributions assumed for statistical variables. We selected the inverter's propagation time $t_P < 2.5$ns as the acceptability criterion. The propagation time t_P was computed by an additional child program. The inverter was excited by a trapezoidal signal and its output was connected to another inverter of the same type to simulate a more realistic load. We did not include statistical variations in the load inverter. We also did not include the interconnection capacitances. The production yield, estimated using 200 outcomes, was 79.5%. The time-domain response of the nominal circuit as well as the Monte Carlo sweep results are presented in Fig. 7.

C. Mixed frequency-time-domain optimization of an RLC circuit

This example demonstrates the mixed domain optimization capability available through Spicepipe. We want to find a second-order model, with the schematic of Fig. 8, of a fourth-order system when the input to the system is an impulse. We consider the time interval from 0 to 10 seconds. The fourth-order system time-domain response is given analytically by

$$V_{out}(t) = \frac{3}{20}e^{-t} + \frac{1}{52}e^{-5t} - \frac{1}{65}e^{-2t}(3\sin 2t + 11\cos 2t). \tag{1}$$

The diagram of this response is shown in Fig. 9. In addition, we impose a frequency-domain specification on the insertion loss INSL of the modelling circuit. We want INSL to be less than 20dB in the frequency range from 0.1Hz to 0.4Hz. C_1, R_1 and R_2 are optimizable variables; R_{in}, R_{out} and L_1 are fixed.

We used the OSA90/hope ℓ_1 optimizer to perform optimization of the nominal circuit. The time-domain response of the second-order circuit was matched to (1). The maximum difference between the desired response (1) and the model response was reduced from 0.15 to 0.01, which satisfied our specifications. INSL satisfied the 20dB specification in the whole frequency range of interest.

Having found the optimum ℓ_1 solution to the problem we performed statistical analysis of the circuit. The Monte Carlo estimate of the production yield at the solution of the nominal problem was 50%. After 30 yield optimization iterations the yield was increased to 90.5%. We used the OSA90/hope yield optimizer with 50 outcomes to optimize yield. To estimate yield, before and after optimization, we used 200 outcomes. Table III lists the values of the optimization variables and assumed standard deviations for statistical variables. Fig. 10 shows the results of Monte Carlo analysis performed before yield optimization. The error between the time-domain response of the second-order circuit and the desired response (1) as well as INSL are plotted. The corresponding curves generated after yield optimization are plotted in Fig. 11.

The listings of the OSA90/hope input files for all three examples are provided in Appendices B through D.

VIII. CONCLUSIONS

We have described Spicepipe, a new child for OSA90/hope. Spicepipe, integrating OSA90/hope with SPICE-PAC, provides the user with all the features of OSA90/hope extended by the time-domain and noise analyses contributed by SPICE-PAC. Spicepipe augments the OSA90/hope modelling capabilities by the device models featured in SPICE-PAC. On the other hand, exploiting SPICE-PAC simulators through the OSA90/hope design environment is more flexible and efficient. The expression processing capability of OSA90/hope can be utilized to perform necessary calculations, just as in the LC transformer example in Section VI. Expression processing makes it also possible to impose algebraic relations among SPICE-PAC circuit parameters. Probably the most important feature of Spicepipe, however, is that by combining tools working in different domains it provides the user with the unique capability of simulating and optimizing a circuit in the frequency and time domains simultaneously.

We have used OSA90/hope's Datapipe technology as the means of the communication between OSA90/hope and SPICE-PAC. We have employed Datapipes again to execute the *create_file*, OSA90/hope's executable child, described in Section IV. We used *create_file* to create SPICE-PAC input files but it can be used to create any other file or files. The execution times obtained using the Spicepipe connection to SPICE-PAC and stand-alone OSA90/hope were similar, with differences smaller than 5%. This kind of comparison is possible only for problems that can be solved by OSA90/hope and Spicepipe independently. Spicepipe proves that the Datapipe technology can be used efficiently to functionally interconnect otherwise independent functional blocks. The OSA90/hope input file is then used to organize the flow of operation among the blocks. This can be done without having to link the separate blocks together. More importantly, the source code information of the functional blocks is not necessary.

To combine OSA90/hope with SPICE-PAC no additional reprogramming of OSA90/hope was required. For SPICE-PAC we had to create the driver organizing SPICE-PAC simulation primitives, but such a driver has to be written for SPICE-PAC anyway. Having to write such a driver requires that the user possess some programming knowledge as well as a good understanding of SPICE-PAC simulation primitives.

REFERENCES

[1] *OSA90/hope™ User's Manual*, Optimization Systems Associates Inc., P.O. Box 8083, Dundas, Ontario, Canada L9H 5E7, 1991.

[2] W.M. Zuberek, "SPICE-PAC version 2G6c an overview," Department of Computer Science, Memorial University of Newfoundland, St. John's, Newfoundland, Canada A1C 5S7, Technical Report 8903, 1989.

[3] J.W Bandler, Q.J. Zhang, G. Simpson and S.H. Chen, "IPPC: a library for inter-program pipe communication," Department of Electrical and Computer Engineering, McMaster University, Hamilton, Canada, Report SOS-90-10-U, 1990.

[4] A. Vladimirescu, K. Zhang, A.R. Newton, D.O. Pederson and A.L. Sangiovanni-Vincentelli, "SPICE Version 2G - User's guide," Department of Electrical Engineering and Computer Sciences, University of California, Berkeley CA 94720, 1981.

[5] *Programming Utilities and Libraries*, SPARCstation 1 Users Manual, Sun Microsystems Inc., 2550 Garcia Ave., Mountain View, CA 94043, pp. 21-26, 1988.

[6] W.M. Zuberek, "SPICE-PAC version 2G6c user's guide", Department of Computer Science, Memorial University of Newfoundland, St. John's, Newfoundland, Canada A1C 5S7, Technical Report 8902, 1989.

[7] R.M. Biernacki, J.W. Bandler, S.H. Chen and P.A. Grobelny, "Integrating the SPICE-PAC simulator with the OSA90/hope design environment", Department of Electrical and Computer Engineering, McMaster University, Hamilton, Canada, SOS Report, 1992, in preparation.

[8] D.A. Hodges and H.G. Jackson, *Analysis and Design of Digital Integrated Circuits.* New York: McGraw-Hill, Inc., 1988.

TABLE I

LC TRANSFORMER CIRCUIT:
L AND C ELEMENT VALUES BEFORE AND AFTER OPTIMIZATION

Element	L_1 (H)	C_2 (F)	L_3 (H)	C_4 (F)	L_5 (H)	C_6 (F)
Value Before Optimization	1	1	1	1	1	1
Value After Optimization	1.041	0.979	2.340	0.780	2.937	0.347

TABLE II

NMOS INVERTER:
MODEL PARAMETERS AND DISTRIBUTIONS ASSUMED

Name	SPICE-PAC Variable	Mean Value	Standard Deviation (%)
Both transistors:			
Transconductance*	KP (A/V^2)	20.0	6.0
Body factor*	GAMMA ($V^{1/2}$)	0.37	-
Body doping*	NSUB (cm^{-3})	5.0×10^{14}	-
Gate oxide thickness*	TOX (m)	0.1×10^{-6}	-
Junction depth*	XJ (m)	1.0×10^{-6}	-
Lateral diffusion*	LD (m)	1.0×10^{-6}	-
Zero-bias bulk capacitance*	CJ (F/m^2)	70.0×10^{-6}	-
Zero-bias perimeter capacitance*	CJSW (F/m)	220.0×10^{-12}	-
Gate-drain overlap capacitance*	CGDO (F/m)	345.0×10^{-12}	-
Gate-source overlap capacitance*	CGSO (F/m)	345.0×10^{-12}	-
The load transistor:			
Threshold voltage	VTO (V)	-3.0	12.0
Gate width	W (m)	5.0×10^{-6}	2.0
Gate length	L (m)	12.0×10^{-6}	2.0
Drain diffusion area	AD (m^2)	100.0×10^{-12}	-
Source diffusion area	AS (m^2)	25.0×10^{-12}	-
Drain area perimeter	PD (m)	40.0×10^{-6}	-
Source area perimeter	PS (m)	15.0×10^{-6}	-
The inverter transistor:			
Threshold voltage	VTO (V)	1.0	12.0
Gate width	W (m)	10.0×10^{-6}	2.0
Gate length	L (m)	7.0×10^{-6}	2.0
Drain diffusion area	AD (m^2)	100.0×10^{-12}	-
Source diffusion area	AS (m^2)	100.0×10^{-12}	-
Drain area perimeter	PD (m)	35.0×10^{-6}	-
Source area perimeter	PS (m)	40.0×10^{-6}	-

* The values for these parameters are the same for the load and inverter transistors.

TABLE III

YIELD OPTIMIZATION OF THE RLC CIRCUIT

Element	Before Optimization	After ℓ_1 Optimization	After Yield Optimization	Standard Deviation (%)
R_{in} (Ω)	1.00	1.00	1.00	*
R_1 (Ω)	0.50	0.92	1.00	5
C_1 (F)	0.50	0.43	0.44	5
L_1 (H)	1.00	1.00	1.00	5
R_2 (Ω)	2.00	0.28	0.26	5
R_{out} (Ω)	1.00	1.00	1.00	*

* Elements assumed fixed (non-statistical).

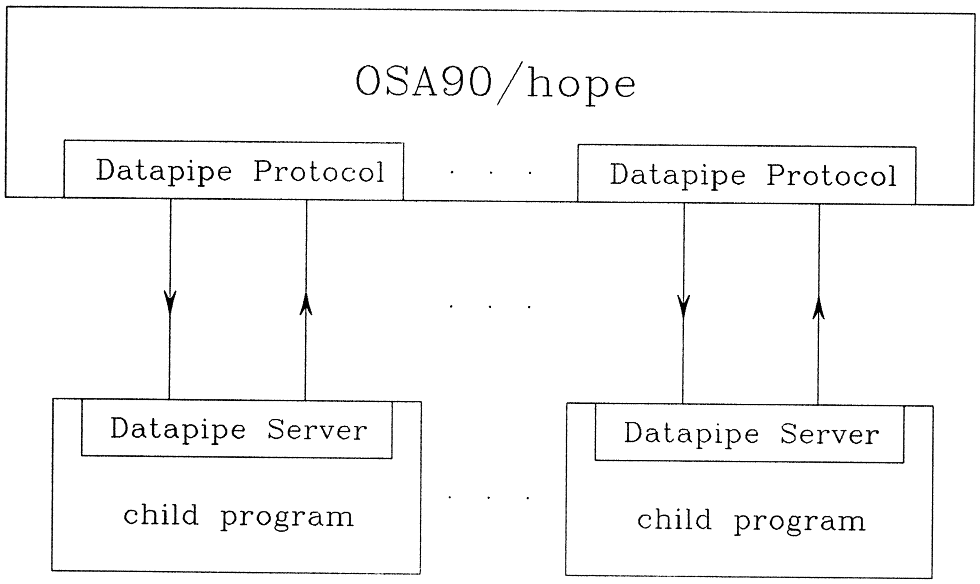


Fig. 1. OSA90/hope Datapipe schematic. Several child programs can be connected to OSA90/hope using the Datapipe technology.

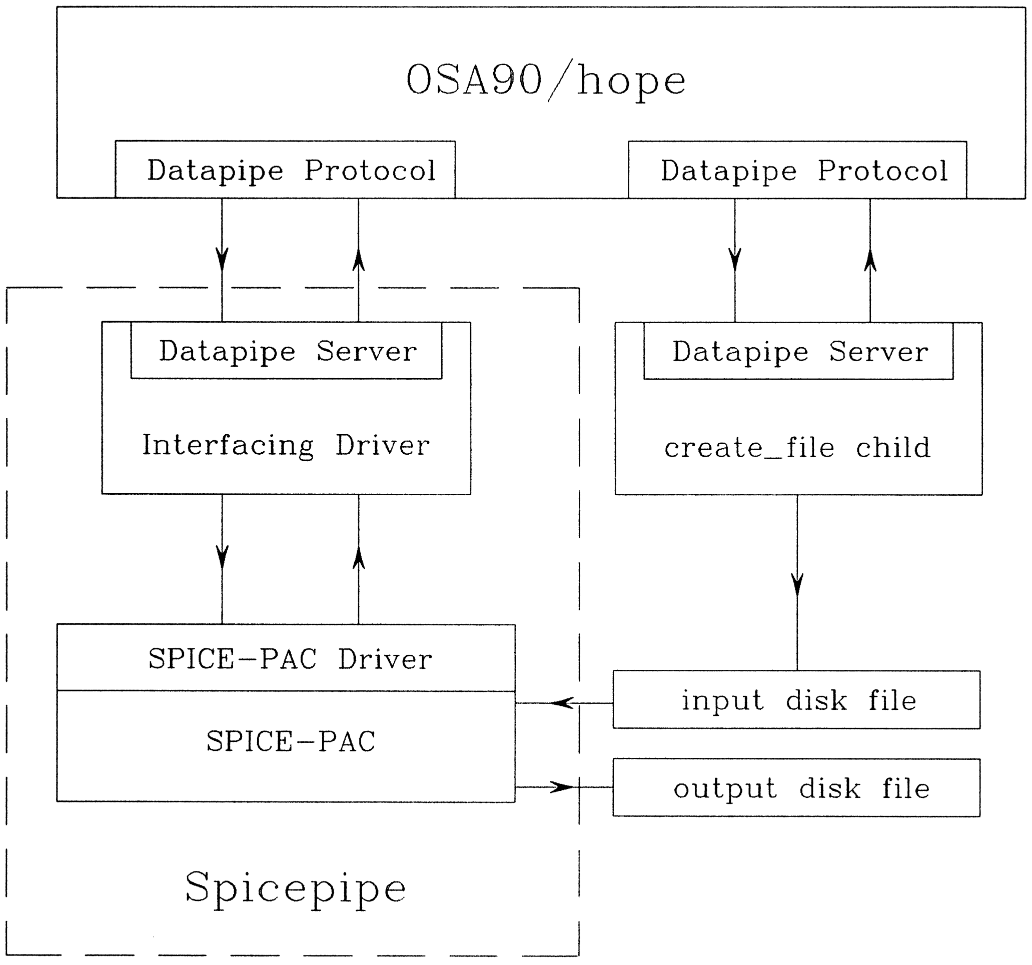


Fig. 2. Integrating SPICE-PAC to OSA90/hope via Spicepipe. Spicepipe consists of the interfacing and SPICE-PAC drivers. They use the IPPC and SPICE-PAC libraries, respectively. The create_file child creates the SPICE-PAC input file.

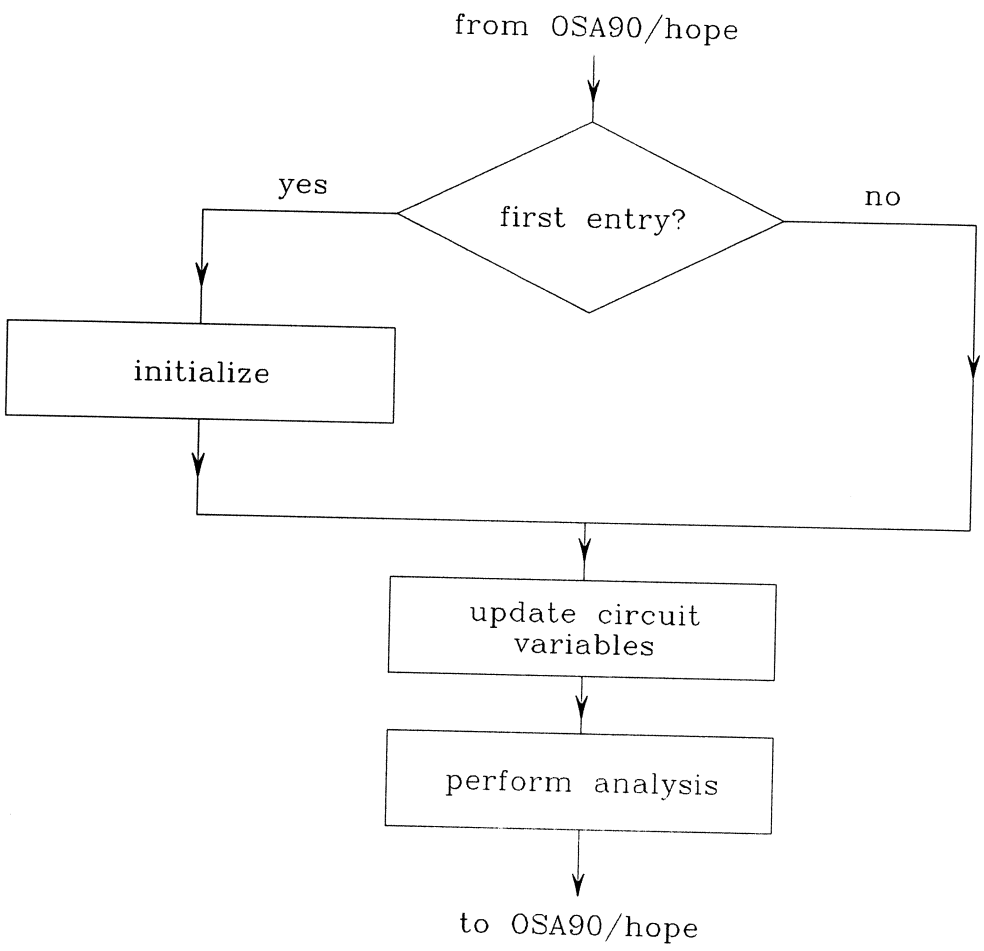


Fig. 3. Block diagram of SPICE-PAC's driver.

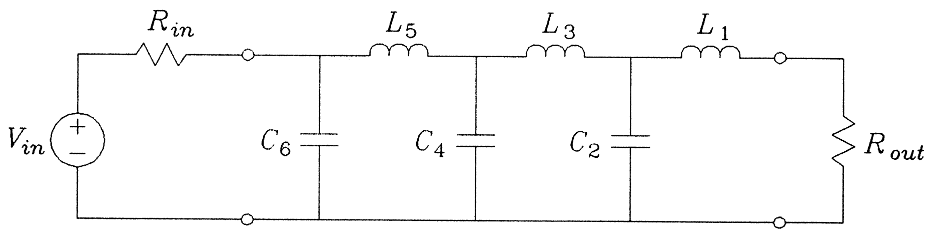
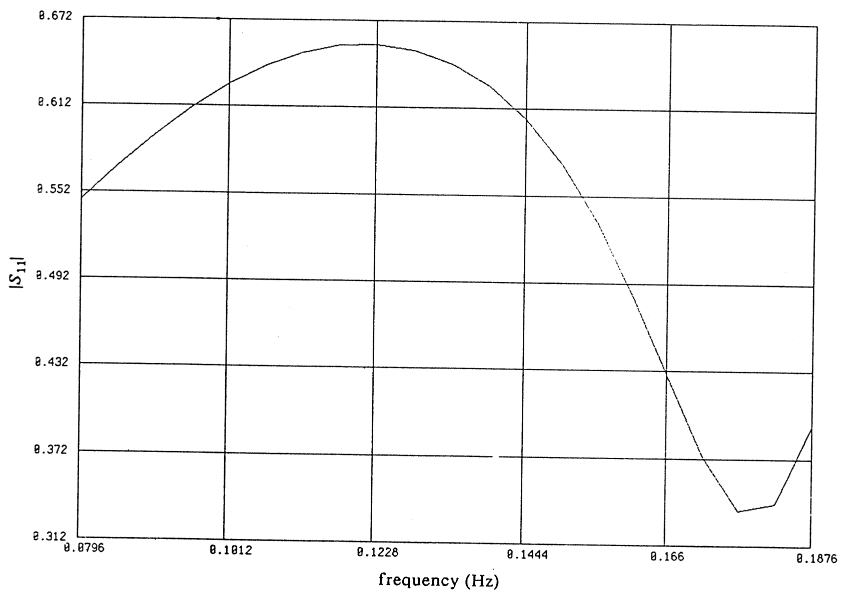
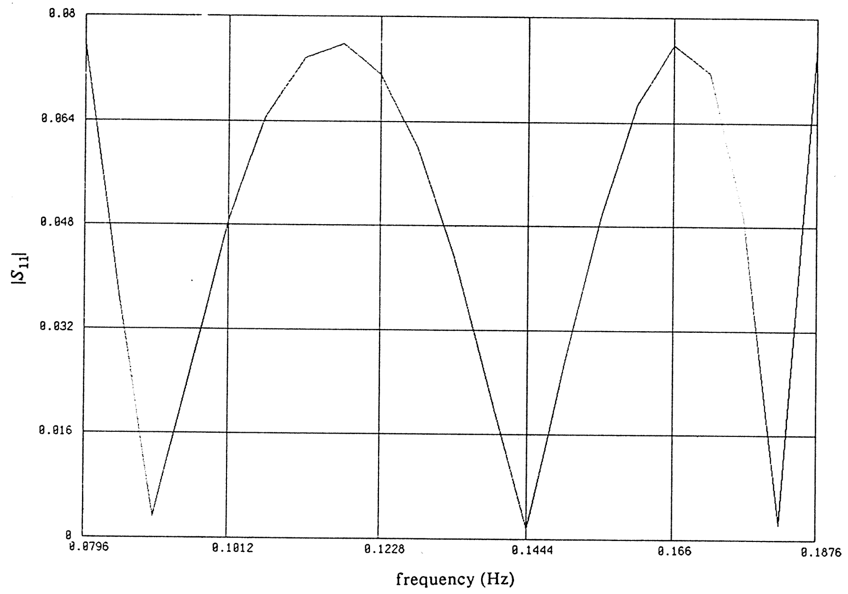


Fig. 4. LC transformer circuit.



(a)



(b)

Fig. 5. $|S_{11}|$ of an LC transformer (a) before and (b) after optimization.

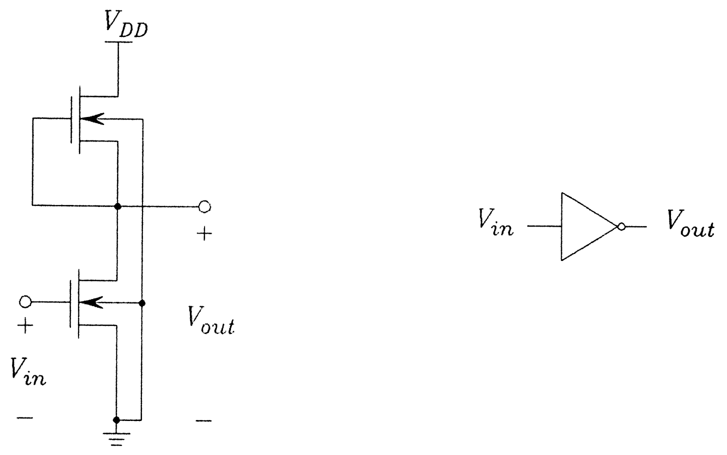
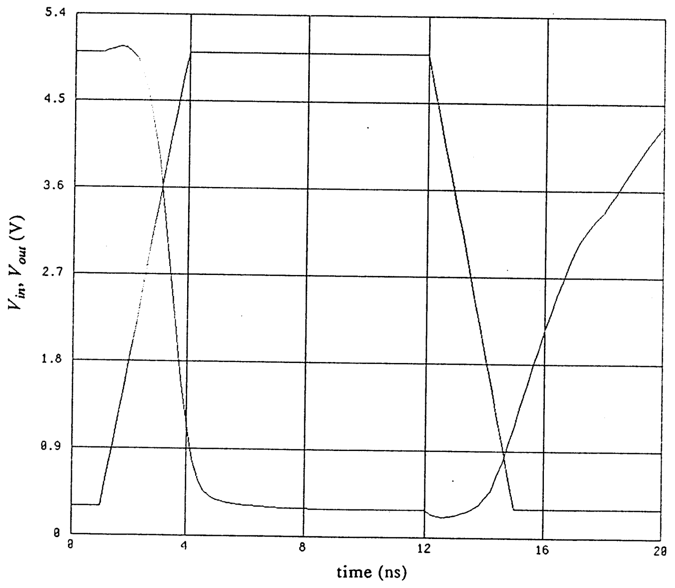
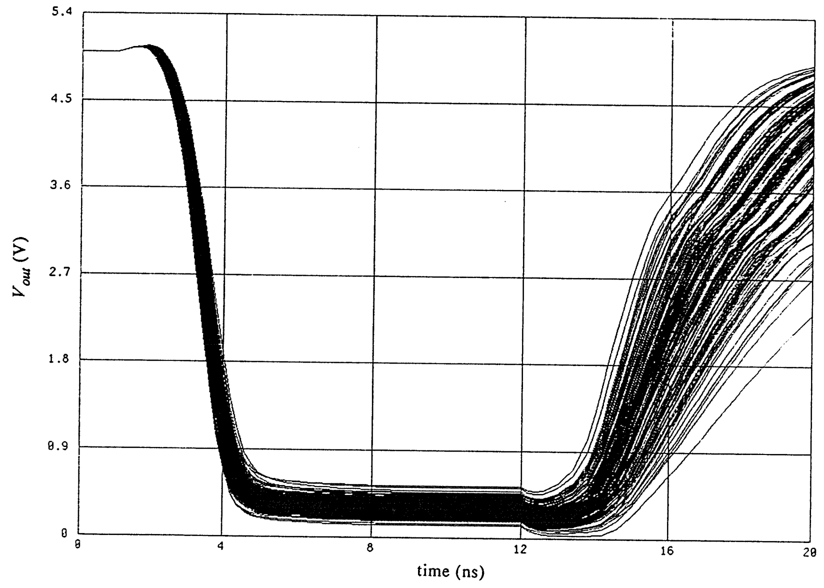


Fig. 6. NMOS inverter, depletion load [8].



(a)



(b)

Fig. 7. Time-domain responses of an NMOS inverter. (a) input and output waveforms and (b) Monte Carlo output waveform sweep.

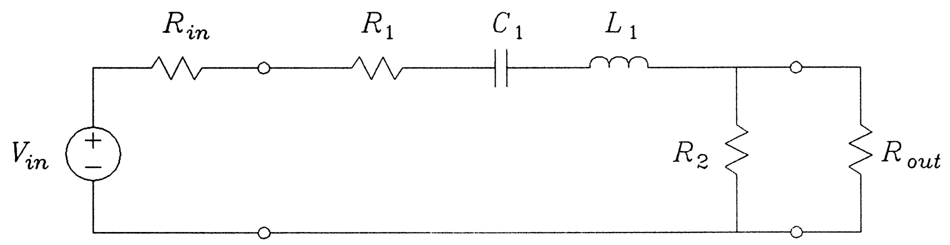


Fig. 8. RLC second order circuit.

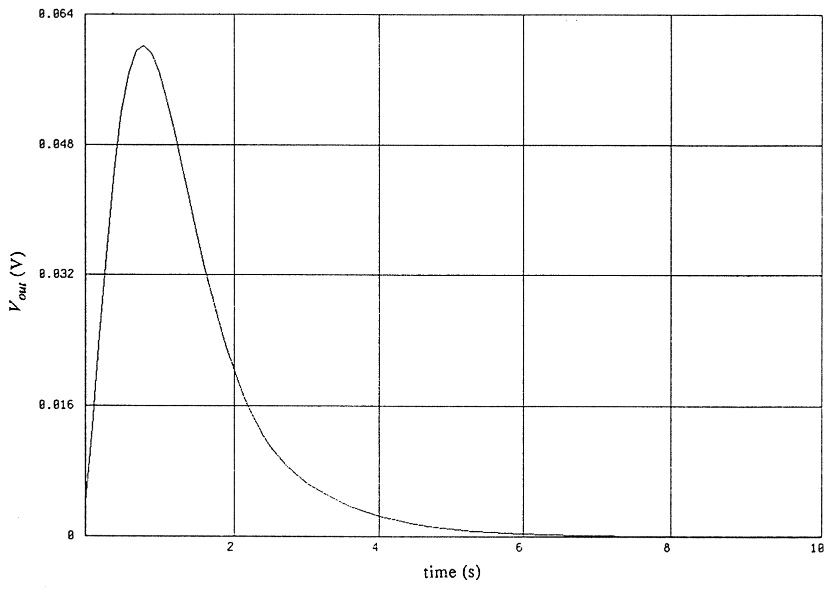
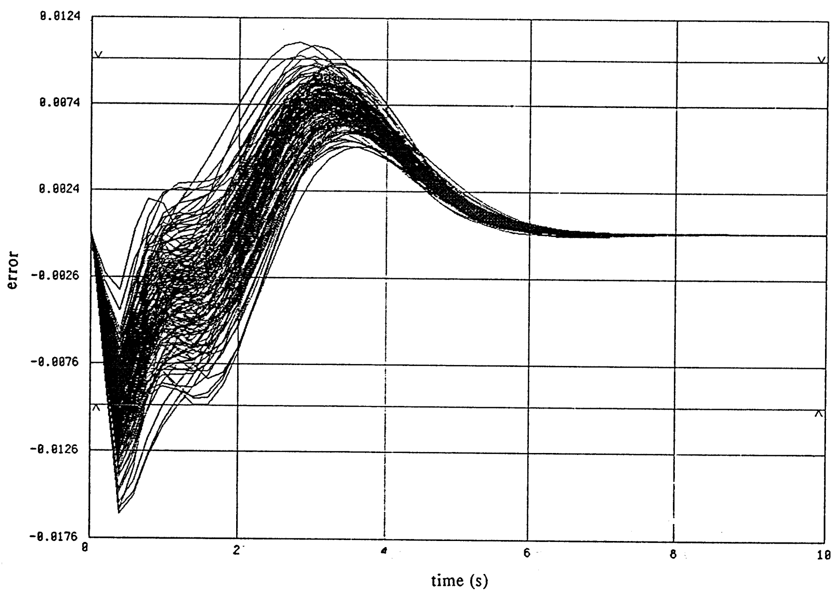
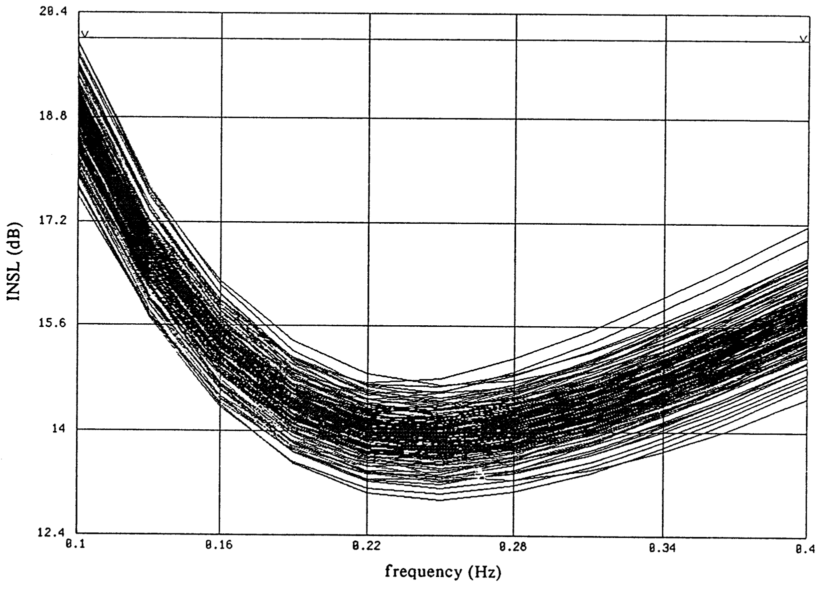


Fig. 9. Response of the fourth-order system, given analytically by (1).

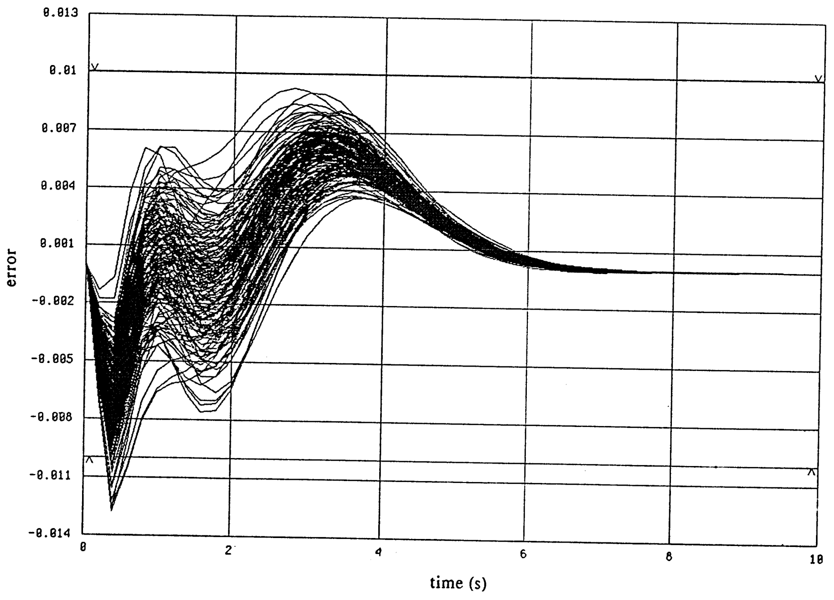


(a)

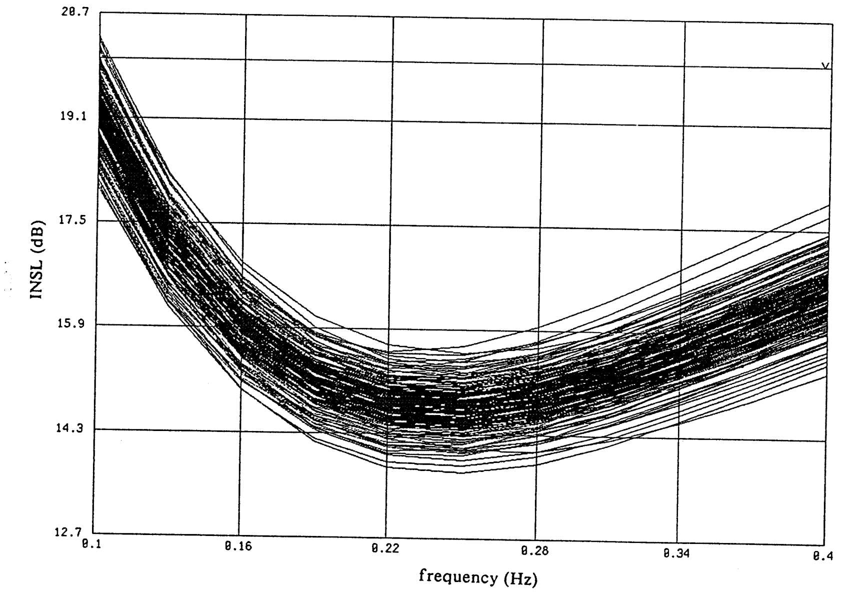


(b)

Fig. 10. Monte Carlo sweep results for the RLC circuit before yield optimization: (a) time-domain match error, and (b) insertion loss.



(a)



(b)

Fig. 11. Monte Carlo sweep results for the RLC circuit after yield optimization: (a) time-domain match error, and (b) insertion loss.

APPENDIX A

HOW TO OBTAIN SPICE-PAC

SPICE-PAC is a version of the public domain software circuit simulator SPICE. Anyone interested can obtain SPICE-PAC free of charge.

To obtain SPICE-PAC you can use the UNIX *ftp* utility and connect to the *garfield* computer at the Memorial University of Newfoundland, Canada. The command to invoke ftp is

```
ftp garfield.cs.mun.ca
```

or

```
ftp 134.153.1.1
```

When the system responds with the *login:* prompt, login to the anonymous account providing your electronic mail address as the password:

```
login: anonymous
password: ...
```

SPICE-PAC can be found in the *pub/sppac* subdirectory. The *pub/sppac* subdirectory contains the following files:

Makefile *Readme* *sppac.tar.Z* *tr-8902.tex.Z* *tr-8903.tex.Z*

Makefile installs SPICE-PAC, *Readme* provides brief information about SPICE-PAC and *sppac.tar.Z* is a compressed and "tared" version of SPICE-PAC. *tr-8902.tex.Z* and *tr-8903.tex.Z* are compressed versions of [6] and [2], respectively written in LaTEX.

To copy the files to your local machine set the mode to binary and perform the following commands:

```
get Readme
get Makefile
get sppac.tar.Z
get tr-8902.tex.Z
get tr-8903.tex.Z
```

or

```
mget *
```

This completes the process of obtaining SPICE-PAC.

Directions on how to install SPICE-PAC can be found in the *Readme* file.

To create Spicepipe change the very last line in *makefile* from

```
f77 -o sppac sppac-drv.f sppac.a
```

into the following sequence of commands:

```
cc  -c -o Sppipe_c.o Sppipe_c.c
f77 -c -o Sppipe_f.o Sppipe_f.f
cc  -o Spicepipe Sppipe_c.o Sppipe_f.o ippcv2.o sppac.a -lF77 -lc -lm
rm Sppipe_c.o Sppipe_f.o
```

Sppipe_c.c and *Sppipe_f.f* are the interfacing driver and SPICE-PAC driver, respectively. *ippcv2.o* contains Datapipe functions used by *Sppipe_c.c*.

APPENDIX B

OSA90/hope INPUT FILE FOR THE LC TRANSFORMER CIRCUIT

```
#define DUMPON        1
#define DUMPOFF       0

Expression
  char cir_contents[]=
" ****************************
 * TRANSFORMER SIMULATION *
 ****************************
VG    1 0 AC 1
RIN   1 2 3
C6    2 0 1
C4    3 0 1
C2    4 0 1
L5    2 3 1
L3    3 4 1
L1    4 5 1
ROUT  5 0 1
.PRINT AC VR(2) VI(2)
.AC LIN 21 0.079578H 0.187644H
.END/EXT
.VAR L1
.VAR L3
.VAR L5
.VAR C2
.VAR C4
.VAR C6
.END
";
   char cir_name[]="lc6.cir";
   char out_name[]="lc6.out";
   char ac[]=".ac";
   Datapipe:     COM              FILE = "create_file"
                 N_INPUT = 2      INPUT = (cir_name, cir_contents)
                 N_OUTPUT = 1     OUTPUT = (char in_name[8]);
   input[1:6]=[?1?, ?1?, ?1?, ?1?, ?1?, ?1?];
   Datapipe:     COM              FILE = "Spicepipe"
                 N_INPUT = 10     INPUT = (in_name, DUMPOFF, out_name, ac, input)
                 N_OUTPUT = 42    OUTPUT = (Vinr[1:21], Vini[1:21]);

! Calculate the reflection coefficient |Ref| using  Ref=2*Vin/Vg-1 where Vg=1
   Refr[1:21]=Vinr+Vinr-1;
   Refi[1:21]=Vini+Vini;
   Refm[1:21]=sqrt(Refr*Refr+Refi*Refi);
   i=0;
End

Sweep
   Title = "Reflection Coefficient"  i: from 1 to 21 step 1 Refm[i];
End

Specification
   Refm=0;
End
```

APPENDIX C

OSA90/hope INPUT FILE FOR THE NMOS INVERTER CIRCUIT

```
#define DUMPON          1
#define DUMPOFF         0
#define UP              0
#define DOWN            1
#define Vmin            0.3
#define Vmax            5
#define TIMESTEP        0.2

Expression
  char cir_contents[] =
" *******************
 *  MOS inverter  *
 *******************
* SUBCIRCUIT DEFINITION, Nodes: Input, Output, VCC
 .SUBCKT NOTGATE 1 2 3
 M_INVER 2 1 0 0   NE W=10U L=7U  AD=100P PD=35U AS=100P PS=40U
 M_LOAD  3 2 2 0   ND W=5U  L=12U AD=100P PD=40U AS=25P  PS=15U
 .MODEL NE NMOS (VTO=1.0 KP=20U GAMMA=0.37 NSUB=5E14 TOX=0.1U XJ=1.0U
 +               LD=1.0U CJ=70U CJSW=220P CGSO=345P CGDO=345P)
 .MODEL ND NMOS (VTO=-3.0 KP=20U GAMMA=0.37 NSUB=5E14 TOX=0.1U XJ=1.0U
 +               LD=1.0U CJ=70U CJSW=220P CGSO=345P CGDO=345P)
 .ENDS NOTGATE
* NOMINAL CIRCUIT DEFINITION
 VDD 4 0 5
 VIN 1 0 PULSE(0.3 5 1N 3N 3N 8N 22N)
 XNOT1  1 2 4 NOTGATE
 XNOT2  2 3 4 NOTGATE
 .PRINT  TRAN V(2) V(1)
 .OPTIONS LIMPTS=5001
 .TRAN 0.2NS 20NS 0N
 .END/EXT
* INVER transistor variables
 .VAR XNOT1.M_INVER'W
 .VAR XNOT1.M_INVER'L
 .VAR XNOT1.M_INVER'AD
 .VAR XNOT1.M_INVER'PD
 .VAR XNOT1.M_INVER'AS
 .VAR XNOT1.M_INVER'PS
 .VAR XNOT1.NE'VTO
 .VAR XNOT1.NE'KP
 .VAR XNOT1.NE'GAMMA
 .VAR XNOT1.NE'NSUB
 .VAR XNOT1.NE'TOX
 .VAR XNOT1.NE'XJ
 .VAR XNOT1.NE'LD
 .VAR XNOT1.NE'CJ
 .VAR XNOT1.NE'CJSW
 .VAR XNOT1.NE'CGSO
 .VAR XNOT1.NE'CGDO
* LOAD transistor variables
 .VAR XNOT1.M_LOAD'W
 .VAR XNOT1.M_LOAD'L
 .VAR XNOT1.M_LOAD'AS
 .VAR XNOT1.M_LOAD'PS
 .VAR XNOT1.ND'VTO
 .VAR XNOT1.ND'KP
 .VAR XNOT1.ND'GAMMA
 .VAR XNOT1.ND'NSUB
 .VAR XNOT1.ND'TOX
 .VAR XNOT1.ND'XJ
 .VAR XNOT1.ND'LD
 .VAR XNOT1.ND'CJ
 .VAR XNOT1.ND'CJSW
 .VAR XNOT1.ND'CGSO
 .VAR XNOT1.ND'CGDO
 .END
";
   char cir_name[]="mos_inv.cir";
   Datapipe:      COM                    FILE = "create_file"
                  N_INPUT = 2            INPUT = (cir_name, cir_contents)
                  N_OUTPUT = 1           OUTPUT = (char in_name[12]);

   XNOT1_M_INVER_W = 10e-6                        {Normal Sigma=2%};
   XNOT1_M_INVER_L = 7e-6                         {Normal Sigma=2%};
   XNOT1_NE_VTO    = 1.0                          {Normal Sigma=12%};
   XNOT1_NE_KP     = 20e-6                        {Normal Sigma=6%};
   XNOT1_NE_NSUB   = 5e14;
   XNOT1_NE_TOX    = 0.1e-6;
   XNOT1_NE_XJ     = 1e-6;
   XNOT1_NE_LD     = 1e-6;

   XNOT1_M_LOAD_W  = 5e-6                         {Normal Sigma=2%};
   XNOT1_M_LOAD_L  = 12e-6                        {Normal Sigma=2%};
   XNOT1_ND_VTO    = -3.0                         {Normal Sigma=12%};
   XNOT1_ND_KP     = 20e-6                        {Normal Sigma=6%};
   XNOT1_ND_NSUB   = 5e14;
   XNOT1_ND_TOX    = 0.1e-6;
   XNOT1_ND_XJ     = 1e-6;
   XNOT1_ND_LD     = 1e-6;

   XNOT1_NE_PB   = 0.0259*log(XNOT1_NE_NSUB/2.1);
   XNOT1_ND_PB   = 0.0259*log(XNOT1_ND_NSUB/2.1);
   XNOT1_NE_CJ   = sqrt(1.6e-19*11.7*8.85e-6*XNOT1_NE_NSUB/2/XNOT1_NE_PB);
   XNOT1_ND_CJ   = sqrt(1.6e-19*11.7*8.85e-6*XNOT1_ND_NSUB/2/XNOT1_ND_PB);
   XNOT1_NE_CJSW = XNOT1_NE_XJ*sqrt(10)*XNOT1_NE_CJ;
   XNOT1_ND_CJSW = XNOT1_ND_XJ*sqrt(10)*XNOT1_ND_CJ;
   XNOT1_NE_Cox  = 3.97*8.85e-12/XNOT1_NE_TOX;
   XNOT1_ND_Cox  = 3.97*8.85e-12/XNOT1_ND_TOX;
   XNOT1_NE_CGSO = XNOT1_NE_Cox*XNOT1_NE_LD;
   XNOT1_NE_CGDO = XNOT1_NE_CGSO;
   XNOT1_ND_CGSO = XNOT1_ND_Cox*XNOT1_ND_LD;
   XNOT1_ND_CGDO = XNOT1_ND_CGSO;
   XNOT1_NE_GAMMA= sqrt(2*11.7*8.85e-6*1.6e-19*XNOT1_NE_NSUB)/XNOT1_NE_Cox;
   XNOT1_ND_GAMMA= sqrt(2*11.7*8.85e-6*1.6e-19*XNOT1_ND_NSUB)/XNOT1_ND_Cox;

   XNOT1_M_INVER_AD=10e-6*XNOT1_M_INVER_W;
   XNOT1_M_INVER_PD=20e-6+2*XNOT1_M_INVER_W-XNOT1_M_LOAD_W;
   XNOT1_M_INVER_AS=XNOT1_M_INVER_AD;
   XNOT1_M_INVER_PS=20e-6+2*XNOT1_M_INVER_W;
   XNOT1_M_LOAD_AS =5e-6*XNOT1_M_LOAD_W;
   XNOT1_M_LOAD_PS =10e-6+2*XNOT1_M_LOAD_W;

   char tr[]=".tr";
   char spp_out[]="mos_inv.out";
   input[1:32]=[XNOT1_M_INVER_W XNOT1_M_INVER_L XNOT1_M_INVER_AD XNOT1_M_INVER_PD
                XNOT1_M_INVER_AS XNOT1_M_INVER_PS
                XNOT1_NE_VTO XNOT1_NE_KP XNOT1_NE_GAMMA XNOT1_NE_NSUB XNOT1_NE_TOX
                XNOT1_NE_XJ XNOT1_NE_LD XNOT1_NE_CJ XNOT1_NE_CJSW XNOT1_NE_CGSO XNOT1_NE_CGDO

                XNOT1_M_LOAD_W XNOT1_M_LOAD_L XNOT1_M_LOAD_AS XNOT1_M_LOAD_PS
                XNOT1_ND_VTO XNOT1_ND_KP XNOT1_ND_GAMMA XNOT1_ND_NSUB XNOT1_ND_TOX
                XNOT1_ND_XJ XNOT1_ND_LD XNOT1_ND_CJ XNOT1_ND_CJSW XNOT1_ND_CGSO XNOT1_ND_CGDO];
  Datapipe:      COM                    FILE = "Spicepipe"
                 N_INPUT = 36           INPUT = (in_name, DUMPOFF, spp_out, tr, input)
                 N_OUTPUT = 202         OUTPUT = (VOUT[1:101], VIN[1:101]);
! calculate the propagation time tp
  Datapipe:      SIM                    FILE = "tp"
                 N_INPUT = 104          INPUT = (Vmin, Vmax, DOWN, VOUT)
                 N_OUTPUT = 2           OUTPUT = (t21, t22);
  Vmed=(Vmax-Vmin)/2+Vmin;!   2.65, for Vmax=5 and Vmin=0.3;
  t201=t21+1; t202=if(t22>0)(t22+1) else (101); t221=t202-1;
  tp21h=TIMESTEP*(Vmed-VOUT[t21])/(VOUT[t21]-VOUT[t201])+t21*TIMESTEP - 2.5;
  tp2hl=TIMESTEP*(Vmed-VOUT[t221])/(VOUT[t221]-VOUT[t202])+t221*TIMESTEP - 13.5;
```

```
   tp=(tp2hl+tp2lh)/2;
   i=0;
   time=0.2*(i-1);
End

Sweep
  Title="VOUT and VIN"   i:  from 1 to 101 step 1 time VOUT[i] VIN[i];
End

MonteCarlo
   N_outcomes=200 i: from 1 to 101 step 1 time tp<2.5 VOUT[i];
end
```

Listing of the additional pipe-ready child *tp* used to compute the propagation time.

```
#define UP 0
#define DOWN 1
#include <stdio.h>
#include <math.h>
#include "ippcv2.h"

void main()
{
  int    input_no,        /* number of input variables */
         output_no,       /* number of output variables */
         error=0;
  float Vmin, Vmax, Dir, Vmed;
  float input_data[512];  /* contains the input data */
  float output_data[2];   /* will contain output data */
  float diff=10;
  int i;

  for(;;)
  {
    pipe_initialize2();
    pipe_read2(&input_no, sizeof(int), 1);
    pipe_read2(&output_no, sizeof(int), 1);
    pipe_read2(&Vmin, sizeof(float), 1);
    pipe_read2(&Vmax, sizeof(float), 1);
    pipe_read2(&Dir, sizeof(float), 1);

    input_no=input_no-3;
    pipe_read2(input_data, sizeof(float), input_no);
    Vmed=(Vmax-Vmin)/2.+Vmin;
    output_data[0]=-1;
    output_data[1]=-1;
    for(i=0;i<(int)input_no;i++)
    {
       switch((int)Dir)
       {
       case UP   :  if(input_data[i]>=Vmed)
                    {
                      output_data[0]=i;
                      break;
                    };
                    break;
       case DOWN :  if(input_data[i]<=Vmed)
                    {
                      output_data[0]=i;
                      break;
                    };
                    break;
       };
       if(output_data[0]>=0)break;
    }
    for(;i<(int)input_no;i++)
    {
       switch((int)Dir)
```

```
        {
        case UP   :  if(input_data[i]<=Vmed)
                     {
                       output_data[1]=i;
                       break;
                     };
                     break;
        case DOWN :  if(input_data[i]>=Vmed)
                     {
                       output_data[1]=i;
                       break;
                     };
                     break;
        };
        if(output_data[1]>=0)break;
    }
    pipe_write2(&error, sizeof(int), 1);
    if(!error)
      pipe_write2(output_data, sizeof(float), output_no);
  }
}
```

APPENDIX D

OSA90/hope INPUT FILE FOR THE RLC SECOND-ORDER CIRCUIT

```
#define DUMPON          1
#define DUMPOFF         0

Expression
  char cir_contents[]=
" ***********************
 * RLC serial circuit *
 ***********************
 VIN  1 0 PULSE(0 1 0 0 0 19S 20S)
 RIN  1 2 1
 R1   2 3 1
 C1   3 4 0.5
 L1   4 5 1
 R2   5 0 1
 ROUT 5 0 1
 .PRINT TRAN V(5)
 .TRAN 0.2S 10S 0S
 .OPTIONS CPTIME=6000
 .END/EXT
 .VAR L1
 .VAR C1
 .VAR R1
 .VAR R2
 .END
";
   L1=1         {Normal Sigma=5%};
   C1=?0.5?     {Normal Sigma=5%};
   R1=?0.5?     {Normal Sigma=5%};
   R2=?2?       {Normal Sigma=5%};
   char cir_name[]="rlc.cir";
   char spp_out[]="rlc.out";
   char tr[]=".tr";
   Datapipe:      COM                     FILE = "create_file"
                  N_INPUT = 2             INPUT = (cir_name, cir_contents)
                  N_OUTPUT = 1            OUTPUT = (char in_name[8]);
   input[1:4]=[L1 C1 R1 R2];
   DataPipe:      COM                     FILE = "Spicepipe"
                  N_INPUT = 8             INPUT = (in_name, DUMPOFF, spp_out, tr, input)
                  N_OUTPUT=51             OUTPUT = (F[1:51]);
   t=1;
   time[1:51]=[0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0
               3.2 3.4 3.6 3.8 4.0 4.2 4.4 4.6 4.8 5.0 5.2 5.4 5.6 5.8 6.0
               6.2 6.4 6.6 6.8 7.0 7.2 7.4 7.6 7.8 8.0 8.2 8.4 8.6 8.8 9.0
               9.2 9.4 9.6 9.8 10.0];
   S[1:51]=3/20*exp(-time)+1/52*exp(-5*time)-1/65*exp(-2*time)*(3*sin(2*time)+11*cos(2*time));
   E[1:51]=S-F;
   Error=E[t];
End
Model
   RES 1 2 R=R1;
   CAP 2 3 C=C1;
   IND 3 4 L=L1;
   RES 4 0 R=R2;
   PORT 1 0 NAME=Vinput V=1 R=1;
   PORT 4 0 NAME=Voutput R=1;
   CIRCUIT;
End
Sweep
   Title="Function, Specification, Error"       t: from 1 to 51 step 1 time[t] S[t] F[t] E[t];
   AC: Title="Gain"      freq: from 0.1 to 0.4 n=10 INSL;
End
```

```
Specification
   E< 0.01   E>-0.01;
   AC: freq: from 0.1 to 0.4 n=10 INSL<20 w=10;
End
MonteCarlo
   N_outcomes=200    t: from 1 to 51 step 1  Error<0.01 Error>-0.01 F[t];
                     AC: freq: from 0.1 to 0.4 n=10  INSL<20;
End
```