22-nd Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, Oct. 31 - Nov. 2, 1988, pp.412–416.
 (DOI 10.1109/ACSSC.1988.754028).

Dynamic Control of Time–Domain Analysis in the SPICE-PAC Simulation Package

W.M. Zuberek

Computer Science Department, Memorial University of Newfoundland
St. John's, Nfld, Canada A1C-5S7

Abstract

SPICE-PAC is a circuit simulation package which is upward compatible with the popular SPICE-2G6 simulation program. Although SPICE-PAC constitutes the "core" of several interactive simulators, the best use of its flexibility is in "integrated" software tools in which circuit simulation is combined with other software tools for circuit optimization, device parameter fitting, multilevel simulations, etc. Recently the package has been subjected to a series of modifications required by "enhanced circuit simulation", i.e., computer-aided circuit analysis that allows users to extend, modify and replace some of the "standard" simulation capabilities. This paper describes modifications which provide "external" run-time control of time-domain (transient) analysis, and shows simple applications of this mechanism.

1. INTRODUCTION

Computer-aided design (CAD) usually refers to applications of computers in the design process. This can include the design of physical systems, architectural environments, manufacture processes and many other areas. Circuit simulation, or computer-aided circuit analysis, is one of those CAD applications that have become well established and widely accepted tools in the design of electronic circuits. Using circuit simulators, the designers can easily determine the functionality and performance of designs before the expensive and time-consuming fabrication takes place.

SPICE-PAC [13] is a package of simulation routines that is upward compatible with the popular SPICE simulation program, developed in 1970's at the University of California, Berkeley [3,11]. That means that SPICE-PAC accepts the same circuit description and performs all the analyses which are available in the SPICE programs, but also provides mechanisms that do not exist in SPICE, for example accessing internal values of circuit elements, dynamic definitions of parameters and outputs, hierarchical naming scheme for subcircuits, parameterized subcircuit expansion, and an interface to libraries of standard modules. Typical applications of the package include interactive circuit simulation, circuit optimization, device parameter fitting, and other applications in which circuit simulation is combined with other CAD tools and techniques [4,5,7,12,13]. Recently the package has been extended by a number of interfaces required by "enhanced circuit simulation" i.e., such an organization of computer-aided circuit analysis in which users can extend, modify and replace some of the "standard" simulation capabilities by their own algorithms in order to increase efficiency, accuracy or applicability of existing simulators. Table-driven nonlinear elements and semiconductor devices or device characteristics specified by symbolic formulas are just a few examples of such enhancements.

One of the of the most useful, but computationally most complex, tasks of circuit simulation is the time domain transient analysis of dynamical circuits, i.e., circuits containing capacitors and inductors [2,6,8,10]. It requires a (numerical) solution of a set of (nonlinear) ordinary differential equations describing the circuit. The detailed algorithm depends upon the integration method used, but generally the simulation interval is divided into (usually variable) timesteps (sometimes called internal timesteps), and at each timepoint, the information from previous timepoints is used to derive the solution at the new timepoint. The stability and accuracy of the integration method has a significant effect on the stability, accuracy, and efficiency of the resulting simulation.

There are two basic classes of integration techniques, explicit and implicit methods [6,10]. Explicit methods find the values of the functions at new timepoints using only the information from the "past" They are quite simple to implement, however, they usually are rather slow because timestep values are seriously restricted by rather small stability regions (which - in general case - are very difficult to determine accurately). Implicit methods use the values of the functions at new timepoints as part of the system of equations to be solved; consequently, the solution must be iterative, but stability regions are much "better" than for explicit methods, and this allows the timestep to be chosen as a function of the accuracy desired at the solution. The two most popular integration methods used in circuit simulation are the trapezoidal rule and variable order backward differentiation (or Gear's) method, the second designed specifically to deal with stiff differential equations [10].

This paper briefly outlines the implementation of time-domain analysis in SPICE-like simulators, and

then describes the extensions that has been added to the SPICE-PAC package in order to provide an "external" "run-time" control of the time-domain (transient) analysis. This allows users (or user routines) not only to monitor, supervise and modify the "standard" execution of this analysis, but also implement elements of mixed-mode simulation [4,5]. A number of simple examples illustrates this new capability.

2. IMPLEMENTATION OF TRANSIENT ANALYSIS

Time-domain transient analysis is performed in two consecutive steps, the so called "Initial Transient" analysis and the proper "Transient" analysis. Initial transient analysis finds the "initial" solution (i.e., the solution of circuit equations for the zero time instant) with or without initial conditions, which can be specified as device initial conditions ("IC=..." options in capacitors, inductors, controlled sources, etc.) or node initial conditions (the ".IC" line in circuit description). The proper transient analysis integrates the differential equations describing the circuit from one timepoint to another, using either the trapezoidal rule (default) or backward differentiation (or Gear) method [3,11].

The proper transient analysis is controlled by two mechanisms, (1) the variable timestep which is determined by user specified parameters as well as truncation errors estimated during integration steps, and (2) the so called "breakpoint table" which contains all characteristic time instances of time-dependent source functions (transmission lines are also taken into account in creating the breakpoint table).

The variable timestep is controlled by iteration count and an estimeted truncation error [3]. The iteration count uses the number of Newton-Raphson iterations steps required to converge at a given timepoint (subroutine SPPITR); if this number is less than the parameter (or "OPTION" ITL3, the timestep is doubled provided it does not exceed the value of DELMAX (subroutine SPPTCT). DELMAX (TMAX in [11]) is either specified by users, or (by default) is equal to the (internal) Time_step:

Time_step :=
min((Time_stop - Time_start)/50,Time_incr)

where "Time_incr", "Time_stop" and "Time_start" are parameters of transient analysis defined by users (TSTEP, TSTOP and TSTART in [11]). Moreover, if the number of iterations is greater than the limit ITL4, the Newton-Raphson iteration is terminated as nonconvergent, the timestep is divided by 8, and the iterative solution begins for a new timepoint, provided the reduced timestep is greater than the "minimum" timestep equal to 10^{-9}*DELMAX (otherwise the notorious message "internal timestep too small" is reported and the analysis terminates [1]).

The estimation of (local) truncation errors (performed by the subroutine SPPTER) uses the derivatives approximated by finite differences. The estimation procedure determines the minimum timestep for which the truncation errors are "acceptable". When the truncation errors (for the assumed timestep) are "unacceptable", the integration "backtracks" in time, and the new, "acceptable" timestep is used provided it is greater than the minimum timestep.

The breakpoint table is created (subroutines SPPBPT which performs a number of additional checks and adjustments and SPPBRT which actually creates the table) by analyzing independent voltage and current sources (and also transmission lines which are neglected here):

```
initialize;
append(zero);
append(Time_stop);
for each independent voltage & current source do {
   case (time_dependent function) of
   "PULSE" : { time:=0;
               while (time < Time_stop) do {
                  append(start_of_rise timepoint);
                  append(end_of_rise timepoint);
                  append(start_of_fall timepoint);
                  append(end_of_fall timepoint);
                  time:=time+pulse_period } };
   "SINE" : append(delay_time);
   "EXP" : { append(start_of_rise timepoint);
             append(start_of_fall timepoint) };
   "PWL" : for each point do
              if (time_value < Time_stop) then
                 append(time_value)
   endcase };
call SPPSHS to sort the breakpoint table;
TOL:=0.01*DELMAX;
compress all values which differ less than TOL;
delete all values greater than Time_stop;
```

The initial transient analysis (subroutine SPPDCA with parameters MODE=1, MODEDC=2) finds the solution of the set of (nonlinear) circuit equations with given (or default) initial conditions:

```
initialize;
TIME:=0;
call SPPSOR to set sources to time_zero values;
INITF:=2;
call SPPITR to solve the system of circuit eqns;
```

and the subroutine SPPITR performs the Newton-Raphson iteration:

```
ITERNO:=NONCON:=0;
DONE:=false;
while (not DONE) do {
   call SPPLDM to load the circuit matrix;
   if ((initial conditions are specified) and
      (analysis=Initial_Transient)) then exit;
   ITERNO:=ITERNO+1;
   case (INITF) of
   "1" : if (NONCON=0) then exit;
   "2" : INITF:=3;
   "3" : if (NONCON=0) then INITF:=1;
   "4","5","6" : INITF:=1
   endcase;
   if (ITERNO > iteration_limit) then
      exit("nonconvergent iteration");
   call SPPDCD to perform LU decomposition;
   call SPPDCS for forward/backward substitutions;
   NONCON:=0;
   for I:=1 to number_of_equations do {
      TOL:=RELTOL*abs(solution[I])+VNTOL;
      if (abs(correction[I]) > TOL) then
         NONCON:=NONCON+1 };
   if (NONCON=0) then DONE:=true };
```

The proper transient analysis (with extensions) is presented in the next section.

3. MODIFIED TRANSIENT ANALYSIS

In order to allow a more flexible control of time-domain (transient) analysis, an in particular, to allow "dynamic" control of this analysis , i.e., control at run-time, the original implementation has been extended by three invocations of an auxiliary routine SPPRTR (SPPRTR is an interfacing subroutine which reorganizes some internal data and then invokes a user-supplied subroutine SPURTR that actually "controls" the analysis; SPURTR is described in the next section). The first invocation of SPPRTR is within the initialization part of analysis, the second is within the iterative integration scheme (DELOLD used in the "reject" section is stored within the "acceptance" sequence), and the third invocation corresponds to "nonconvergent" cases.

```
initialize;
TIME:=0;
DELTA:=Time_step;
if (run_time_control) then
   call SPPRTR with initial parameters;
DELBKP:=DELTA;
DELMIN:=10E-9*DELMAX
INITF:=5;
IBRTAB:=1;
NTIMEP:=0;
ITERTR:=0;
BREAKP:=true;
saveoutputs:
store outputs in an internal table;
if (run_time_control) then {
   call SPPRTR with actual parameters;
   if (terminate) then {
      interpolate output results;
      return from analysis } else
   if (reject) then {
      delete last results;
      DELTA:=DELOLD;
      TIME:=TIME-DELTA;
      DELTA:=DELTA/4;
      BREAKP:=false } else
   if (reset) then run_time_control:=false };
if (TIME > Time_stop) then {
   interpolate output results;
   return from analysis };
if (BREAKP) then {
   IBRTAB:=IBRTAB+1;
   DELBPT:=breakpoint_table[IBRTAB]-TIME;
   DELTA:=min(DELTA,0.1*min(DELBKP,DELBPT));
   if (NTIMEP=0) then DELTA:=0.1*DELTA;
   BREAKP:=false }
else
   if (TIME+DELTA > breakpoint_table[IBRTAB])
   then {
      DELBKP:=DELTA;
      DELTA:=breakpoint_table[IBRTAB]-TIME;
      BREAKP:=true };
NCHECK:=3;
newtime:
TIME:=TIME+DELTA;
if (execution_time > time_limit) then
   stop_analysis("execution time limit reached");
if (ITL5 > 0 and ITERTR > ITL5) then
   stop_analysis("iteration steps limit reached");
call SPPSOR to update time_dependent sources;
if (INITF <> 5) then INITF:=6;
call SPPITR to solve the system of circuit eqns;
ITERTR:=ITERTR+ITERNO
NTIMEP:=NTIMEP+1;
if (converged) then {
   DELNEW:=DELTA;
   call SPPTCT to adjust the timestep DELNEW;
   if (integration error is acceptable) then {
      DELOLD:=DELTA;
      DELTA:=DELNEW;
      go to saveoutputs };
   TIME:=TIME-DELTA;
   DELTA:=DELNEW }
else {
   TIME:=TIME-DELTA;
   DELTA:=DELTA/8 };
check:
if (DELTA >= DELMIN) then go to newtime;
```

```
if (run_time_control) then {
   call SPPRTR with termination parameters;
   if (continue) then {
      TIME:=TIME-DELTA;
      DELTA:=DELTA*8;
      NCHECK:=NCHECK-1;
      if (NCHECK > 0) then go to check } };
stop_analysis ("timestep too small");
```

4. USER INTERFACE (SPURTR)

In SPICE-PAC, the communication between subroutines and functions follows that of the original SPICE program, i.e., it uses mainly COMMON areas to pass numerical parameters, indicators and conditions, as well as to return results. Such a solution is quite efficient for internal communication, but is extremely unreliable and almost unacceptable for user interfaces. Therefore routines that can be modified and enhanced by users, use an additional level of interfacing procedures in which user-accessible parameters are reorganized into vectors, and passed as vector arguments. This also provides for the possibility of an elementary consistency check after user-defined modifications.

The subroutine SPPRTR is an intermediate routine that provides an interface to a user-defined "external" run-time control that (if used) must be supplied as a subroutine SPURTR with the following (Fortran) header:

```
      SUBROUTINE SPURTR (VOUT,NOUT,TDEL,IRET)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION VOUT(NOUT)
```

where the parameters are as follows:

VOUT - a DOUBLE PRECISION array of length NOUT which contains the values of outputs obtained for the present timepoint (NOUT is the number of output variables increased by 1); it should be noted that VOUT(1) is the actual value of TIME, VOUT(2) is the value of the first output variable, etc.;

NOUT - an INTEGER argument that is set to the length of VOUT;

TDEL - a DOUBLE PRECISION argument which is set to the value of timestep;

IRET - an INTEGER variable that is used as an entry/return flag; on entry, IRET=-1 indicates the initial call, IRET=0 indicates an accepted timepoint solution, and IRET=+1 indicates nonconvergence; on exit, IRET=0 indicates continuation of analysis, IRET=+1 terminates transient analysis at the current timepoint, IRET=-2 turns off external control (i.e., SPPRTR and SPURTR subroutines will not be called but the analysis continues), and IRET=-3 rejects the timepoint, after which the stored results are deleted from an iternal table, the TIME "returns" to the previous timepoint, timestep is divided by 4, and the analysis resumes.

5. EXAMPLES

Example 1: The following simple example shows a "dynamic" termination of transient analysis when the second output (i.e., VOUT(3) in SPURTR) "saturates", i.e., when the value of the second output variable insignificantly changes at a number of consecutive timepoints (5 in this example, determined by the LIMSIM parameter); it should be observed that no initial delay is allowed for this solution, however, only a minor modification of the termination conditions is needed to deal with such a situation:

```
      SUBROUTINE SPURTR (VOUT,NOUT,TDEL,IRET)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION VOUT(NOUT)
      DATA LIMSIM / 5 /
      IF (IRET.LT.0) THEN
        MARK=0
        NUMT=0
      ELSE IF (IRET.EQ.0) THEN
        WRITE(*,1) NUMT,MARK,(VOUT(I),I=1,NOUT),TDEL
1     FORMAT(2X,I3,' : ',I1,1P7D9.2)
        NUMT=NUMT+1
        IF (MARK.GT.0) THEN
          TOL=1D-3*(DABS(VOUT(3))+1D-3)
          IF (DABS(AOUT-VOUT(3)).LT.TOL) THEN
            IF (MARK.GE.LIMSIM) IRET=1
          ELSE
            MARK=0
          ENDIF
        ENDIF
        AOUT=VOUT(3)
        MARK=MARK+1
      ENDIF
      RETURN
      END
```

The final part of a trace of consecutive timepoints printed by this routine and the results of transient analysis for a very simple circuit are as follows:

```
** simple external control
VV 1 0 DC(1) PULSE(0 1 0 2NS 2NS 25NS)
R2 1 2 100
C2 2 0 10P
.TR 1NS 20NS
```

```
.PRINT TR V(1) V(2)
.END

  ....................................
  22 : 1 5.00d-09 1.00d+00 9.79d-01 4.00d-10
  23 : 1 5.40d-09 1.00d+00 9.86d-01 4.00d-10
  24 : 1 5.80d-09 1.00d+00 9.91d-01 4.00d-10
  25 : 1 6.20d-09 1.00d+00 9.94d-01 4.00d-10
  26 : 1 6.60d-09 1.00d+00 9.96d-01 4.00d-10
  27 : 1 7.00d-09 1.00d+00 9.97d-01 4.00d-10
  28 : 1 7.40d-09 1.00d+00 9.98d-01 4.00d-10
  29 : 2 7.80d-09 1.00d+00 9.99d-01 4.00d-10
  30 : 3 8.20d-09 1.00d+00 9.99d-01 4.00d-10
  31 : 4 8.60d-09 1.00d+00 9.99d-01 4.00d-10
  32 : 5 9.00d-09 1.00d+00 1.00d+00 4.00d-10

***** TRANSIENT ANALYSIS : TEMP = 27.0 DEG C

      TIME        V(1)      V(2)

    0.00d+00    0.00d+00  0.00d+00
    1.00d-09    5.00d-01  1.85d-01
    2.00d-09    1.00d+00  5.66d-01
    3.00d-09    1.00d+00  8.42d-01
    4.00d-09    1.00d+00  9.41d-01
    5.00d-09    1.00d+00  9.79d-01
    6.00d-09    1.00d+00  9.92d-01
    7.00d-09    1.00d+00  9.97d-01
    8.00d-09    1.00d+00  9.99d-01
    9.00d-09    1.00d+00  1.00d+00
```

while the "standard" results of transient analysis for the same circuit would continue until the "Time_stop" without any furhter significant change.

Example 2: The SPURTR routine is used to find the "exact" time instances at which the output signal of a simple MOSFET inverter reaches the 10% and 90% levels.

```
** double inverter - transient analysis
VDD 5 0 DC=5
VIN 1 0 PULSE(0 5 1NS 2NS 2NS 3NS 10NS)
M1 5 1 2 5 PMOD L=3U W=6U AS=36P AD=36P
M2 2 1 0 0 NMOD L=3U W=3U AS=18P AD=18P
M3 5 2 3 5 PMOD L=3U W=6U AS=36P AD=36P
M4 3 2 0 0 NMOD L=3U W=3U AS=18P AD=18P
*... NMOS transistor model
.MODEL NMOD NMOS LEVEL=2
*... PMOS transistor model
.TR 0 20NS
.MODEL PMOD PMOS LEVEL=2
.PRINT TRAN V(1) V(3) V(2)
.END

... time :  2.0866d-09 output rising at 10%
... time :  2.7116d-09 output rising at 90%
... time :  6.9168d-09 output falling at 90%
... time :  7.3557d-09 output falling at 10%
... time :  1.2087d-08 output rising at 10%
... time :  1.2712d-08 output rising at 90%
... time :  1.6917d-08 output falling at 90%
... time :  1.7356d-08 output falling at 10%
```

It can be observed that the time increment parameter in the ".TR" line is equal to zero; this is one of (minor) SPICE-PAC modifications which indicates that the results are required at the original internal timepoints rather than at the interpolated equidistant points at which the detailed information about changes and slopes of the waveforms may be significantly distorted or lost.

The SPURTR routine (which uses linear interpolation to determine the intercept points) can be as follows:

```
      SUBROUTINE SPURTR (VOUT,NOUT,TDEL,IRET)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION VOUT(NOUT)
      CHARACTER*21 TXT
      IF (IRET.LT.0) THEN
        Ta=VOUT(1)
        Va=VOUT(3)
      ELSE IF (IRET.EQ.0) THEN
        Tb=VOUT(1)
        Vb=VOUT(3)
        TXT=' '
        IF (Va.LT.0.5 .AND. Vb.GE.0.5) THEN
          TEMP=Ta+(0.5-Va)*(Tb-Ta)/(Vb-Va)
          TXT='output rising at 10%'
        ELSE IF (Va.LT.4.5 .AND. Vb.GE.4.5) THEN
          TEMP=Ta+(4.5-Va)*(Tb-Ta)/(Vb-Va)
          TXT='output rising at 90%'
        ELSE IF (Va.GE.4.5 .AND. Vb.LT.4.5) THEN
          TEMP=Ta+(4.5-Va)*(Tb-Ta)/(Vb-Va)
          TXT='output falling at 90%'
        ELSE IF (Va.GE.0.5 .AND. Vb.LT.0.5) THEN
          TEMP=Ta+(0.5-Va)*(Tb-Ta)/(Vb-Va)
          TXT='output falling at 10%'
        ENDIF
        IF (TXT.NE.' ') WRITE(*,100) TEMP,TXT
  100   FORMAT(' ... time :',1PD12.4,1X,A)
        Va=Vb
        Ta=Tb
      ENDIF
      RETURN
      END
```

6. CONCLUDING REMARKS

One of known "problems" of transient analysis in SPICE-like simulators is due to its variable timestep method, which often leads to an "internal timestep too

small" termination [1]. It should be observed, that the run-time control can "trace" the values of internal timesteps (TDEL parameter of SPURTR), and (automatically) adjust the tolerance parameters (using the SPICEX routine [13]) during transient analysis. Moreover, a more elaborate SPURTR routine could "switch" to an interactive mode in cases of "nonconvergent" analyses, and then users might interactively "help" the analysis to converge by appropriate changes of simulation parameters.

Also, the external control mechanism can be used for implementation of simple mixed-mode simulations, however, for more complicated circuits, such "manually" controlled interactions can easily become irregular and unreliable. Therefore, for realistic mixed-mode simulations, a more systematic approach is needed in which the "control" of analog simulation is derived "automatically" from (formalized) specifications of the digital part of the circuit. In fact, an independent module could be developed to generate software routines from logic descriptions of digital parts, and this software (compiled or interpreted) could be used for combined simulation with analog parts.

The extensions described in this paper are implemented in the SPICE-PAC versions 2G6c and beyond.

Acknowledgement

The Natural Sciences and Engineering Research Council of Canada partially supported this research through Operating Grant A8222, and Northern Telecom Canada through Memorial University Interaction Program.

References

[1] P. Antognetti, G. Massobrio, "Semiconductor device modeling with SPICE"; McGraw-Hill 1988.

[2] R.E. Bank, W.M. Coughran Jr, W. Fichtner, E.H. Grosse, D.J. Rose, R.K. Smith, "Transient simulation of silicon devices and circuits"; IEEE Trans. on Computer Aided Design, vol.4, no.4, pp.436-451, 1985, and also IEEE Trans. on Electron Devices, vol.32, no.10, pp.1992-2007, 1985.

[3] E. Cohen, "Program reference for SPICE 2"; Memorandum UCB/ERL M592, University of California, Berkeley CA 94720, 1976.

[4] J.T.J. van Eijndhoven, J.A.G. Jess, "Mixed-mode mixed-level analysis with PWL systems"; Proc. IEEE Int. Symp. on Circuits and Systems, Montreal, Canada, pp.1377-1380, 1984.

[5] H. de Man, G. Arnout, P. Reynaert, "Mixed-mode simulation techniques and their implementation in DIANA"; in: "Computer Design Aids for VLSI Circuits", (eds) P. Antognetti, D.O. Pederson, H. de Man, Sijthoff and Noordhoff 1981.

[6] D.O. Pederson, "Computer aids in integrated circuit design"; in: "Computer Design Aids for VLSI Circuits", (eds) P. Antognetti, D.O. Pederson, H. de Man, Sijthoff and Noordhoff 1981.

[7] Ch. Poivey, "Methodes d'optimisation globale pour la CAO de circuits integres; interface avec le simulateur SPICE-PAC" (Global optimization methods for CAD of integrated circuits; an interface to the SPICE-PAC simulation package); These de Docteur Ingenieur, l'Universite Blaise Pascal, serie D.I., no.203, Clermont-Ferrant, France, 1987.

[8] R.A. Rohrer, H. Nosrati, K.W. Heizer, "Quasi-static control of explicit algorithms for transient analysis"; IEEE Trans. on Computer-Aided Design, vol.3, no.3, pp.226-233, 1984.

[9] K.A. Sakallah, S.W. Director, "SAMSON : a mixed circuit-logic level simulator"; in: "Advances in Computer-Aided Engineering Design - vol.1", (ed.) A. Sangiovanni-Vincentelli, pp.149-223, JAI Press 1985.

[10] J. Vlach, K. Singhal, "Computer methods for circuit analysis and design"; Van Nostrand Reinhold 1983.

[11] A. Vladimirescu, K. Zhang, A.R. Newton, D.O. Pederson, A.L. Sangiovanni-Vincentelli, "SPICE Version 2G - User's Guide (10 Aug. 1981)"; Department of Electrical Engineering and Computer Sciences, University of California, Berkeley CA 94720, 1981.

[12] K. Walsh, B. Wolfe, "Mixed-domain analysis for circuit simulation"; VLSI Systems Design, vol.8, no.8, pp.44-49, 1987.

[13] W.M. Zuberek, "SPICE-PAC 2G6a.84.05 - User's Guide"; Technical Report 8404, Department of Computer Science, Memorial University of Newfoundland, St. John's, Newfoundland, Canada A1C 5S7, 1984.