ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/251546940

CIRCUIT MODELING IN DYMOLA

Article

CITATIONS
2

READS
5,620

1 author:

Daryl R. Hild
MITRE
10 PUBLICATIONS **73** CITATIONS

SEE PROFILE

All content following this page was uploaded by Daryl R. Hild on 22 August 2016.

The user has requested enhancement of the downloaded file.

CIRCUIT MODELING IN DYMOLA

by

Daryl Ralph Hild

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE
WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1993

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Daryl A Hild

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Dr. François E. Cellier

November 8, 1993

François E. Cellier
Associate Professor of
Electrical and Computer Engineering

Date

ACKNOWLEDGMENTS

I would like to thank Dr. Francois E. Cellier for his guidance and direction as I worked through the development, verification, and write-up of the bipolar junction transistor models discussed in this thesis. I wish to thank the MITRE Corporation, especially the management team at Fort Huachuca, Arizona – Jeff Hustad, Milt Hutchinson, Jack Moody, Dick Nixon, and Robert Ringdahl – for their support of my master's degree studies. Most of all, I thank my wife, Janis, for her support and encouragement throughout this effort.

DEDICATION

To Janis,

for her love,

support,

and friendship...

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

In this thesis, an alternative to SPICE as an electrical circuit modeling and simulation tool is explored. SPICE has been very popular automated circuit analysis tool of industry and universities, alike. A new modeling tool, Dymola, supports object-oriented modeling and can generate state-space models for simulation in several different simulation languages: ACSL, DESIRE, Simnon, SimuLink, and DSblock. In Dymola, objects are mathematically modeled as implicitly described sets of ordinary differential equations. Dymola objects may then be interconnected to form more complex systems.

This thesis develops a library of PNP and NPN bipolar junction transistor models, and investigates use of the transistor models as subcomponents to more complex circuits such as operational amplifiers. The simulation results are compared to those obtained from two Spice dialects, and are discussed in terms of accuracy, efficiency, flexibility, and robustness.

1. INTRODUCTION

Today's circuit designers use automated tools to model and simulate electronic circuit designs prior to committing a design to a fabrication process. One of the more popular modeling and simulation tools for electronic circuits is SPICE.

A variety of SPICE dialects exist to run on various computers from desktops to mainframes. SPICE uses a topological modeling approach for circuit description. In this approach, all circuit nodes are numbered and the circuit designer assigns circuit element terminals to nodes. Consider the simple logic circuit of Figure 1.

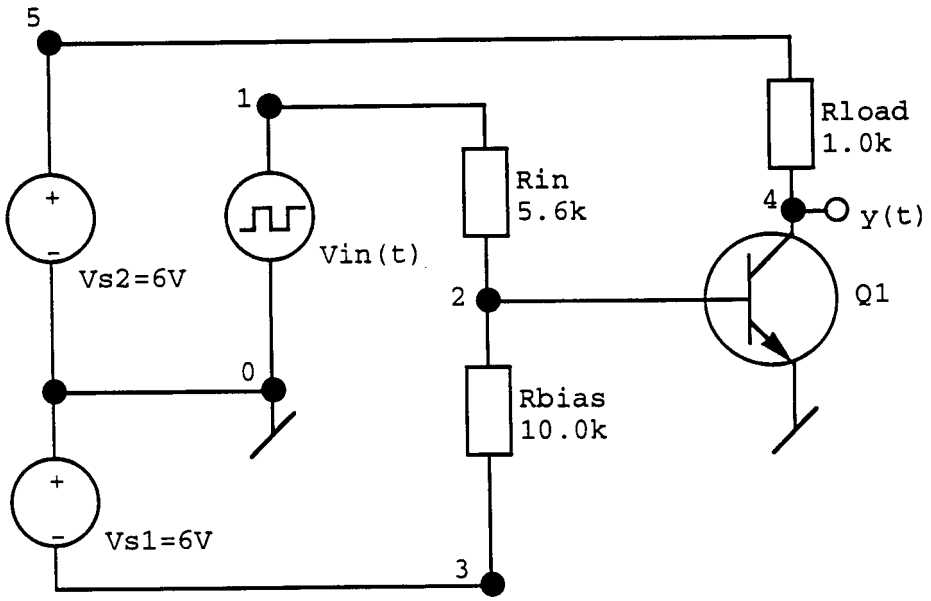


Figure 1. Simple Logic Inverter

In SPICE, the following description models this circuit.

```
Simple Logic Inverter
  Vin   1  0        0
  Vs1   0  3        6
  Vs2   5  0        6

  Q1    4  2  0  5  PROC35.N
  Rin   1  2        5.6k
  Rbias 2  3        10.0k
  Rload 4  5        1.0k

.MODEL PROC35.N NPN
+ IS=1.1fA BF=100 BR=1
+ RC=500 RE=15 RB=1000
+ CJC=.3pF CJE=.5pF CJS=.1pF
+ VJC=.65 VJE=.77 VJS=.75
```

Note that the first character of each element name determines type of circuit element being used: V - voltage source, R - resistor, Q - transistor. The element name is followed by node assignments for its terminals. The node assignments are followed by element parameter assignments. In the case of the bipolar junction transistor (BJT) element Q1, PROC35.N refers to a .MODEL statement which allows us to group sets of device parameters for the BJT element type.

Not all SPICE dialects use the same equations or device models. In fact, the equations and device models used in SPICE software packages are often proprietary and the circuit designer using a particular SPICE dialect will not normally have visibility of the equations coded into the software. This limitation constrains the designers

understanding of the model he is using and the relationship a particular parameter may play in modeling physical characteristics of a device. This is particularly true for the more complex active devices such as BJTs, junction field effect transistors (JFETs), and metal oxide semiconductor field effect transistors (MOSFETs). Furthermore, many versions of SPICE do not allow the user access to internal voltages, currents, and other circuit element parameters. These limitations constrain ones ability to analyze and interpret simulation results relative to a particular parameter or set of parameters.

Another limitation of SPICE as a modeling and simulation tool is its restriction to electrical and electronic components. SPICE has no facilities to model non-electrical systems or interfaces between electronic and non-electronic systems. SPICE is a well proven tool for electrical and electronic circuit engineering. It provides limited support, however, to the systems engineer designing electro-optic, electro-mechanical, electro-biological, or electro-chemical systems.

A modeling language is needed to (1) allow the designer to directly see how a parameter fits in the description of a device, (2) allow the designer to access internal operating parameters and values of a device during simulation, (3) support topological and/or power and energy flow descriptions of systems and subsystems, (4) support hierarchical modeling, and (5) support modeling of various types of physical devices - electronic, mechanical, biological, chemical - and coupling these devices together into systems.

One potential modeling language is Dymola [1] and [2]. Dymola is an object-oriented modeling language and preprocessor. As a modeling language, it allows the modeler to hierarchically decompose and describe a system and its subsystem

components in a very compact, yet easily readable, syntax. Furthermore, Dymola allows the modeler to separately describe the mode/system and the experiment/stimulus under which to simulate the model. As a model preprocessor, Dymola can determine the causalities, reduce the structural singularities, and solve the algebraic loops arising from the interconnection of subsystems (submodels) and generate a state-space description of the model in a variety of continuous system simulation language (CSSL) -type modeling and simulation languages [3].

Another possible modeling tool is the System Performance Simulator [4]. In an independent effort Martin Vlach researched and developed a modeling and simulation tool where the modeling capabilities are similar in approach to Dymola. Both tools model continuous systems using differential algebraic equations to describe the modeled elements. Models developed for one could easily be converted into a form readable by the other. Ironically, Vlach received a U.S. Patent on his work in 1992 despite the fact that such a modeling tool had already been developed by Elmqvist in 1978.

In [5], Cellier proposes using Dymola as an alternative to SPICE: "If we are able to make Dymola powerful enough that it can handle arbitrarily complex circuits containing arbitrary algebraic loops and structural singularities, we can automatically generate a state-space model that will execute much more efficiently at run-time than the currently used SPICE code." To support his assertion, Cellier began developing a Dymola model of a bipolar junction transistor (BJT). As described in this paper, this model has meanwhile been developed further, and its accuracy and validity were verified against BBSPICE [6] and PS pice [7]. This paper also discusses outstanding developments needed to achieve the full benefits of using Dymola over SPICE as envisioned by Cellier.

2. CIRCUIT MODELING USING DYMOLA

2.1. The Basic Electrical Components

Modeling the basic circuit elements in Dymola is a straightforward process. The following model, for example, describes the electrical characteristics of a resistor and its interfaces.

```
model type resistor
parameter R=1.0
cut WireA(Va / i) WireB(Vb / -i)
main path P <WireA - WireB>
local u
  u = Va - Vb
  R*i = u
end
```

For added clarity, Dymola keywords have been bolded. This description defines an object class **(model type)** of type resistor. The resistive value of the resistor is defined by the **parameter** R, which is set to a default value of 1.0. The **cut** statement defines the two wires of a resistor used to connect it into a circuit. The **path** statement further defines this connection as a directed path from the input cut to the output cut. The **local** variable u is used to compute the voltage drop across the resistor and Ohm's Law is used to describe the resistor itself.

Similar object class (model type) descriptions for the resistor and other basic circuit elements - inductor, capacitor, diode, voltage source, current source, and common

(ground node) - have been developed and consolidated into a Dymola library file of electrical components. This Dymola library file is listed in Appendix A. A modeler using Dymola can simply call this library file and then use these component descriptions in the modeling of more complex circuits and systems.

2.2. Modeling Circuits With The Basic Components

After defining and describing the model types to be used in a circuit, the next step is to declare (instantiate) each component in the circuit and describe the interconnections. For example, the simple passive electrical circuit of Figure 2 is described in the following Dymola model.

```
model circuit
  submodel (voltage) U0
  submodel (resistor) R1(R=100.0) R2(R=200.0)
  submodel (capacitor) C(C=1.0E-6)
  submodel (inductor) L(L=1.5E-3)
  submodel common
  input u
  output y1, y2

  connect common - U0 - R1 - C – common
  connect U0 - R2 - L - common

  U0.u = u
  y1 = C.u
  y2 = L.i
end
```

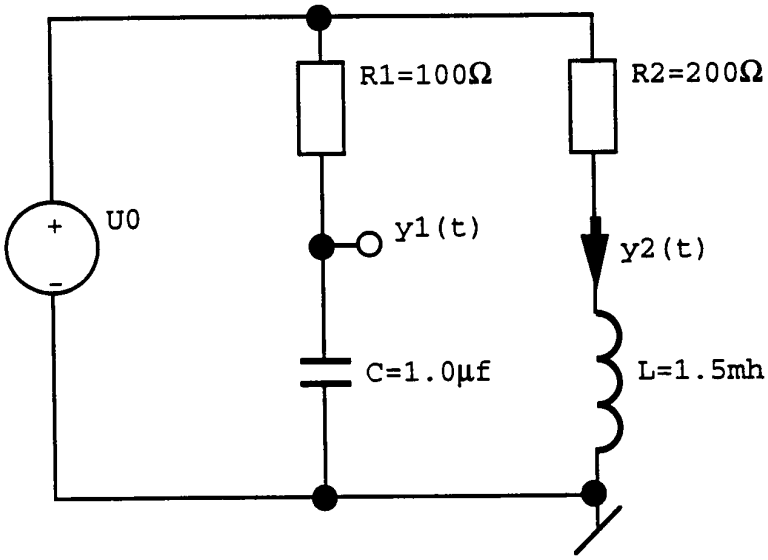


Figure 2. Simple Passive Element Electrical Circuit

The **submodel** declaration instantiates each model type (object). The parameter value settings for an object are annotated in parentheses and override the default settings found in the model type descriptions. The **connect** statements allow for a compact description of the component interconnections in terms of two directed paths. Finally, the three algebraic equations at the end of the model connect the input and output parameters to the appropriate model variables. In this model, the input and output variables are variables found within the voltage source, capacitor, and inductor submodels. A dot notation of the type 'submodel_name.variable_name' is used to identify the respective submodel variables.

From this description, Dymola can automatically generate the connecting equations based on Kirchhoffs voltage and current laws. From the connecting equations and the component description equations, Dymola can automatically determine the causality

assignments, reduce the structural singularities, and solve the algebraic loops that result from submodel coupling.

With this background in electrical circuit modeling in Dymola, we now turn to the development of a bipolar junction transistor model using Dymola.

3. THE BIPOLAR JUNCTION TRANSISTOR

The bipolar junction transistor is built around p-n junctions. These junctions are highly nonlinear in their electrical characteristics. A diode is a p-n junction with the anode equating to the p-side and the cathode equating to the n-side of the junction. Forward biasing the junction diode is achieved by placing the p-side at a higher potential than the n-side.

3.1. The P-N Junction

.Figure 3 provides three depictions of the p-n junction. The first depiction is of the physical device with a heavy p+ doping concentration for the anode and a lightly doped n-silicon base structure for the cathode. The p-n junction is depicted in the center with electrical schematic symbols. Note that the capacitance of the junction is a non-linear function of the charge stored in the junction. Also, the resistance of the junction accounts for the minimum admittance of the junction. An aggregate junction diode schematic symbol is depicted on the right.

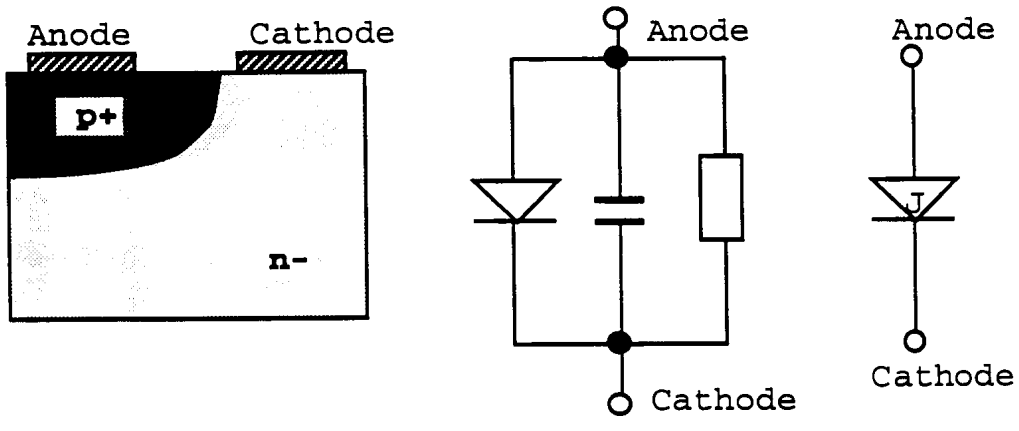


Figure 3. P-N Junction Models

From Figure 3 it is seen that the current through the p-n junction is simply the sum of currents resulting from the capacitance, admittance, and ideal diode effects of the junction. The relationship of the current through, and the voltage across, the junction is:

$$i = i_d + i_g + i_c$$
$$i_d = I_s * \exp(u_d/(V_t * N_d)) - I_s$$
$$i_g = G_{min} * u_d$$
$$i_c = C_d * \mathrm{der}(u_d) = \mathrm{der}(q_c)$$

where I_s is the transport saturation current of the diode; V_t is the thermal voltage; N_d is the current emission coefficient; G_{min} is the minimum admittance of the junction; and $q_c = C_d * u_d$ is the electrical charge stored in the junction capacitance. The thermal voltage is computed from $V_t = k*T/q$ where k is Boltzmann's constant; T is the temperature; and q is the electrical charge of an electron.

In [5], Cellier approximates the electrical charge of the junction as a function of current and voltage with the equation:

$$q_c = \tau_d * i_d + V_{bi} * C_d * (1 - (1 - u_d/V_{bi})^{(1-m_d)})/(1-m_d)$$

where τ_d is the transit time constant of the capacitance, i_d is the current through the junction diode which is also a function of u_d, V_{bi} is the built-in voltage potential of the junction, C_d is the zero-bias depletion capacitance, and m_d is the exponential grading factor of the junction.

By handling the first and second terms of q_c separately and taking their derivatives with respect to the junction voltage u_d, the diode diffusion capacitance and the junction depletion capacitance respectively are obtained as follows:

$$
\begin{aligned}
C_{dif} &= der(q_{c\text{-}dif}) \\
&= der(\tau_d * i_d) / der(u_d) \\
&= \tau_d * der(I_s * exp(u_d/(V_t * N_d)) - I_s) / der(u_d) \\
&= \tau_d * I_s * exp(u_d/(V_t * N_d)) / (V_t * N_d)) * der(u_d) / der(u_d) \\
&= \tau_d * I_s * exp(u_d/(V_t * N_d)) / (V_t * N_d))
\end{aligned}
$$

$$
\begin{aligned}
C_{dep} &= der(q_{c\text{-}dep}) / der(u_d) \\
&= der(V_{bi} * C_d * (1 - (1 - u_d/V_{bi}) ** (1 - m_d)) / (1 - m_d)) / der(u_d) \\
&= (V_{bi} * C_d * ((1 - u_d/V_{bi}) ** (-m_d)) * der(u_d) / V_{bi}) / der(u_d) \\
&= C_d / (1 - u_d/V_{bi}) ** m_d.
\end{aligned}
$$

This approximation of the depletion capacitance C_{dep}, however, has a singularity at $u_d = V_{bi}$. In [8], Van Halen proposed the following approximation equation for C_{dep} to eliminate this singularity:

$$C_{dep} = C_d / (1 - (u_d - 0.5 * V_t * exp((u_d - V_{bi})/V_t)/V_{bi}) ** m_d.$$

From the above equations, the p-n junction can easily be modeled in Dymola with the following description.

```
model type jdiode
  cut Anode(Va / I) Cathode(Vb / -I)
  main path P <Anode - Cathode>
  parameter ND=1 IS=1.0E-16 TD=0 CD=0 VD=0.75 MD=0.33 ->
            AREA=1 GMIN=1.0E-12
  external DTemp FTemp VT ISfact VDfact
  terminal Id u
  local ISv Vbi CDv ISe Ic Cdif Cdep denom
  { Electrical equations }
    u = Va - Vb
    I = Id + Ic
    Id = ISe - ISv + GMIN*u
    ISe = ISv*exp(u/(VT*ND))
    Ic = der(u) * (Cdif + Cdep)
  { Junction capacitance equations }
    Cdif = TD * (ISe/(VT*ND) + GMIN) {Diffusion cap.}
    Cdep = CDv / denom               {Depletion cap.}
    denom = (1 - (u - 0.5*VT*exp((u-Vbi)/VT))/Vbi)**MD
  { Temperature adjustment equations }
    ISv = IS*AREA*ISfact
    Vbi = FTemp*VD + VDfact
    CDv = CD*AREA*(1 + MD*(1 - Vbi/VD + 4.0E-4*DTemp))
end
```

This description introduces three more declaration statements available in Dymola: **external, terminal,** and **local. External** parameters allow for an implicit data exchange between the submodel and the next higher order system, i.e. externals are like global parameters. Calling models must acknowledge the existence of externals by declaring such variables as **internal. Terminals** and **locals** are variables in the model that may

change in value during a simulation run. Terminals support connections to other devices while locals do not. Terminals are accessed using the dot notation introduced earlier.

The above description also introduces the jdiode temperature adjustment equations. The performance of a p-n junction is temperature dependent. This model compensates for temperature effects on the transport saturation current, IS, the built-in junction potential, VD, and the zero-bias capacitance, CD. These temperature equations were found in both the HSPICE User's Manual [9] and the BBSPICE [6] source code.

3.2. Modeling The Bipolar Junction Transistor

The bipolar junction transistor consists of multiple p-n junctions. The two primary junctions are the base-collector and the base-emitter junctions. A third p-n junction occurs across the substrate. BJTs are classified as either NPN or PNP transistors based on the doping of the emitter, base, and collector. BJTs are further classified as being either vertically or laterally diffused transistors depending on the physical geometry of the device. Figure 4 shows a vertically diffused and laterally diffused NPN transistor. For the PNP transistor the doping concentration in each region is simply reversed.

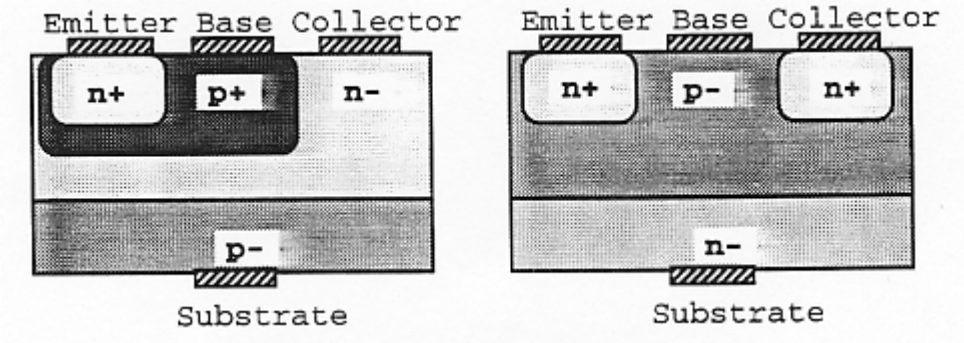


Figure 4. Vertical and Lateral NPN BJT

From Figure 4 it is seen that the substrate p-n junction is formed with the collector for the vertically diffused BJT, and it is formed with the base for the laterally diffused BJT.

For the BJT a standard convention for current flow into the transistor is specified to avoid memorization of different convention sets depending on the BJT type. This standard convention is shown in Figure 5.

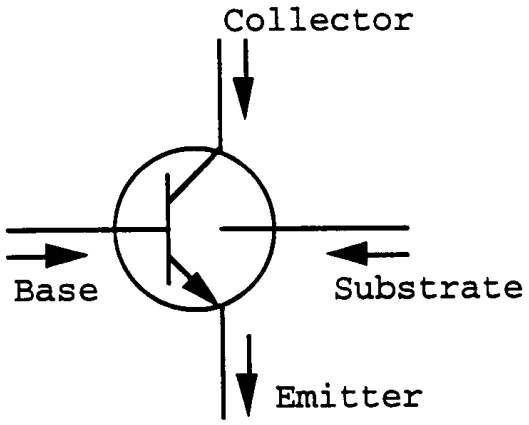


Figure 5. BJT Current Convention

The following Dymola statements model this BJT interface.

```
cut C(VC / IC) B(VB / IB) E(VE / - IE) S(VS / ISUB)
main cut CBES [C B E S]
path BE<B-E> BC<B-C> BS<B-S> CE<C-E> CS<C-S> ES<E-S>
path EB<E-B> CB<C-B> SB<S-B> EC<E-C> SC<S-C> SE<S-E>
```

The directed path flows given in this description allow for a compact specification for any combination of directed flows into and out of the transistor.

Figure 6 provides a graphic model of the vertically and laterally diffused NPN transistor. In the vertical NPN transistor, the substrate is connected to the collector. For the lateral NPN, the substrate is connected to the base. The PNP transistor model is the same except that the diode polarities are reversed. This is the same model as given in [5].

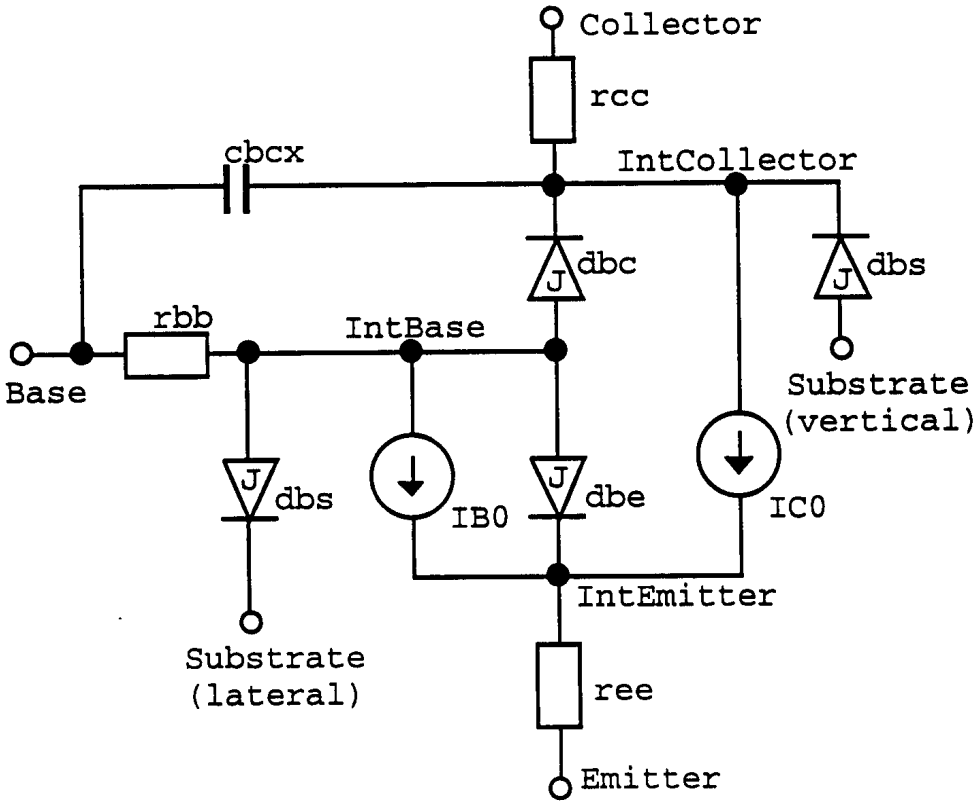


Figure 6. A Vertical And Lateral NPN Transistor Model

Describing these four types of BJTs is straightforward using the concept of inheritance as supported by Dymola. Using inheritance, the internal components of the BJT can be described in a generic BJT model which can then be called upon to describe the NPN and PNP, vertical and lateral, BJTs specifically. Using this method, the following model describes the laterally diffused NPN BJT.

```
model type (BJT) NPNlateral
  { Plug the internal circuits together. }
    connect  rbb from    B to IntB
    connect  rcc from    C to IntC
    connect  ree from IntE to E
    connect  dbc from IntB to IntC
    connect  dbe from IntB to IntE
    connect  dbs from IntB to IntS
    connect wire from    S to IntS
    connect cbcx from    B to IntC
    connect ibe0 from IntB to IntE
    connect ice0 from IntC to IntE
end
```

Models for the three other BJT types are contained in Appendix B. The model of the interface convention given previously is part of the generic BJT model as it is common across all BJT types. The rest of this generic BJT model is presented next.

3.3. The External Base-Collector Capacitance

The external base-collector capacitance, `Cbcx`, is used to model the physical distribution of the junction charge and current flow across the base. `Cbcx` is actually part of the base-collector junction depletion capacitance. In [5], Cellier modeled this capacitance as a separate capacitor. The parameter `XCJC` is used to set the fraction of the base-collector capacitance to be found within (internal to) the junction diode model, whereas `1-XCJC` is used to set the fraction of capacitance in `Cbcx`, the capacitance distributed across (external to) the junction diode. Rather than compute the base-collector capacitance twice and then multiply each result by `XCJC` and `(1-XCJC)` in

determining the associated capacitance currents, we can compute the total capacitance current once and multiply this by `XCJC` and `(1-XCJC)` to obtain the associated capacitance currents internal to and distributed across the junction. This approach to modeling the external base-collector capacitance also resolves the degenerate system problem with the BJT model as described in [5, p 228].

Using this approach and choosing to compute the base-collector capacitance in the junction diode model, the `Cbcx` capacitance can be modeled as a dependent current source:

```
model type Csource
  cut A(. / I) B(. / -I)
  main path P <A - B>
  terminal I0
    I = I0
end
```

where the terminal `I0` statement is used to connect the dependent current to its determinants, in this case, the base-collector junction diode where the total capacitance current and its distribution across the base are computed.

3.4. The Two Dependent Current Sources

The two dependent current sources `IC0` and `IB0` represent the DC component of the collector current and the base current. These current sources can be modeled with the following set of equations.

```
IC0 = (ibe-ibc)/qb - ibc/BR - icn
IB0 = ibe/BF + ibc/BR + ien + icn - ibe
icn = ISC*(exp(vbc/(VT*NC)) - 1)
ien = ISE*(exp(vbe/(VT*NE)) - 1)
```

In these equations, `ibc` and `ibe` are the base-collector and base-emitter diode currents; `icn` and `ien` are also diode currents, but they are based on the leakage saturation current parameters `ISC` and `ISE` and the leakage emission coefficients `NC` and `NE`. `VT` is the thermal voltage. `BF` and `BR` are the ideal maximum forward and reverse beta coefficients that represent the DC current gain factors `IE/IB` and `IC/IB`. The term `qb` is the base charge. SPICE models the base charge with the following set of equations.

```
q1 = 1/(1 - vbc/VAF - vbe/VAR)
q2 = ien/IKF + icn/IKR
qb = 0.5*q1*(1 + (1 + 4*q2)**0.5)
```

Here, `vbc` and `vbe` are the voltages across the base-collector and base-emitter diodes respectively; `VAF` and `VAR` are the forward and reverse early voltages; and `IKF` and `IKR` are the forward and reverse high current beta roll-off (degradation) parameters.

Also, as will be discussed later, the derivatives, with respect to `vbe`, of these base charge equations will be needed for the base-emitter diode model. These derivative equations are listed here.

```
dq1 = der(q1)

    = (der(vbe)/VAR)/(q1*q1)/der(vbe)

    = I/(VAR*q1*q1)
```

```
dq2 = der(q2)
    = der(ien)/IKF/der(vbe)
    = (ien+ISE)*der(vbe)/(VT*NC*IKF)/der(vbe)
    = (ien+ISE)/(VT*NC*IKF)

dqb = der(qb)
    = 0.5*der(q1)(1+sqrt(1+4*q2))
      + q1*der(q2)/sqrt(1+4*q2)
    = qb*der(q1)/q1 + q1*der(q2)/sqrt(1+4*q2)
```

The above equations are included as part of the top level BJT model. The computed currents `IC0` and `IB0` are connected to the dependent current source submodels `ice0` and `ibe0` respectively. The current sources `ice0` and `ibe0` use the same dependent current source model as given for the external base-collector model above.

3.5. The Collector And Emitter Resistances

The collector and emitter resistors of the BJT model are modeled as area dependent resistors. The area parameter is a scaling parameter. These resistors are modeled as:

```
model type varresistor
  cut A(Va / I) B(Vb / -I)
  main path P <A - B>
  cut Par(Rv)
  parameter AREA
  local u
    u = Va - Vb
    u = I*Rv/AREA
end.
```

Note that the same model type can be used to describe both the collector resistor, `rcc`, and the emitter resistor, `ree`. In this model, only the `AREA` is declared as a parameter; the resistive value `Rv` has been declared as a cut. Typically, transistor circuits are fabricated on a single chip. To minimize the number of fabrication processes, and thus the cost of manufacturing a chip, the transistors on a chip will be very similar and have the same parameters with the exception of the area occupied by each transistor. So, declaring all but the `AREA` parameter as cuts in these models will allow all the parameter declarations to be consolidated in a separate parameter specification model, `BJTpar`. The parameters for multiple transistors can be set once in `BJTpar` and then connected to the individual BJTs of the circuit being modeled.

This scheme is very similar to the .MODEL statement of SPICE. This scheme will also allow for consolidation of BJT related constants and temperature compensation factors and equations. One such equation is for the temperature sensitive resistor presented below. This scheme can save CPU cycles at runtime by evaluating a common set of equations once for several transistors. The scheme also saves on the variable name space by minimizing the number of unique variables instantiated for multiple transistors.

The elements of an integrated circuit are temperature sensitive. From the HSPICE manual and the BBSPICE source code, the resistance found in the collector and emitter is a quadratic function of temperature, which is modeled with the equation:

```
Rv = R*(1 + TR1*DTemp + TR2*DTempSq)
```

where TR1 and TR2 are the first and second order temperature coefficients. DTemp and DTempSq are the difference and the difference squared of the device and room temperature. As stated above, these temperature adjustment equations are consolidated in the BJTpar model.

3.6. The Base Resistance

The base resistance is the most important resistance in the BJT model and correspondingly has a more complex model than the collector and emitter resistances. It is a variable resistor where the resistance is dependent on the current through the base as well as the area associated with the BJT. The SPICE model for the base resistance is:

```
rbb = RBM + 3*(RB - RBM)*(tan(z) - z)/(z*tan(z))**2
```

with

```
z = (-1 + sqrt((1+144*ib)/(IRB*pi**2)))
    /(24*sqrt(ib/IRB)/pi**2).
```

RBM is the minimum resistance given a high base current; RB is the maximum resistance given a low base current; ib is the base current; pi is the constant 3.14159; and IRB is the base current where the base resistance falls halfway between RB and RBM.

The above equation for z fails, however, if the parameter IRB is set to zero. In that case, SPICE automatically switches to the simpler equation:

```
rbb = RBM + (RB - RBM)/qb
```

where qb is the base charge discussed earlier. The following Dymola model for rbb implements this SPICE model. This implementation has commented out the base current model and uses only the simpler base charge equations.

```
model type rbb
  cut A(Va / I) B(Vb / -I)
  main path P <A - B>
  cut Par(RBv RBMv IRBv PiSq)
  parameter AREA
  external qb
  local u R {z tz)
    u = Va - Vb
  { R = if IRBv > 0.0 ->
      then (RBMv + 3.0*(RBv-RBMv)*(tz-z)/(z*tz*tz))/AREA ->
      else (RBMv + (RBv - RBMv)/qb)/AREA
    z = if IRBv > 0.0 ->
      then (-1 + sqrt(1 + 144*I/(PiSq*IRBv*AREA))) ->
        /(24*sqrt(I/(IRBv*AREA))/(PiSq)) ->
      else 0.0
      tz = if IRBv > 0.0 then tan(2) else 0.0
  }
  R = (RBMv + (RBv - RBMv)/qb)/AREA
  R*I = u
end
```

For this model, PiSq is a global constant that is included in the Par cut along with the other model parameters. Again, this mechanism allows consolidation of these factors in BJTpar. The base resistor is temperature sensitive and uses the same compensation equation as ree and rcc for the high and low current resistance temperature adjustment.

3.7. The Base-Collector Diode

A model for the p-n junction diode was presented earlier and will form the basis of our base-collector diode (dbc) model. Our deviation from the p-n junction diode model is based on the development of the external base-collector capacitance model. The goal was to calculate the base-collector depletion capacitance once and use the internal base-collector fraction parameter `XCJC` to distribute the current between the internal and external models. In the following `dbc` model, `Ix` is the external capacitance current and the terminal declaration of `Ix` allows it to be connected to the `Cbcx` model.

```
model type dbc
cut Anode(Va / I)  Cathode(Vb / -I)
main path P <Anode - Cathode>
cut Par(VTNR ISv TRv CJCv XCJCv VJCv MJCv GMINDCv VT)
parameter AREA
terminal Id u Ix
local ISe Ic Cdif Cdep denom
{ Electrical equations }
  u = Va - Vb
  I = Id + Ic
  Id = (ISe - ISv)* AREA + GMINDCv*u
  ISe = ISv*exp(u/(VTNR))
  Ic = der(u)*(Cdif+XCJCv*Cdep) {Internal cap current}
  Ix = der(u)*(1-XCJCv)*Cdep    {External cap current}
{ Junction capacitance equations }
  Cdif = TRv * (ISe/(VTNR) + GMINDCv) {Diffusion cap}
  Cdep = AREA*CJCv/denom              {Depletion cap}
  denom = (1 - (u - 0.5*VT*exp((u-VJCv)/VT))/VJCv)**MJCv
end
```

3.8. The Base-Emitter Diode

Again, the p-n junction diode model forms the basis of our base-emitter diode (`dbe`) model. The base-emitter diffusion capacitance term, however, is a bit more complex as it is dependent on the base charge as well as the base-emitter current. The HSPICE user's manual provides the following equation for this capacitance:

```
Cdif = der(TF*Id/qb)/der(u)
```

and by expansion, we can get

```
Cdif = TF*(der(Id)/qb + Id*der(1/qb))/der(u)
     = TF*((IS*exp(u/(VT*ND))/(VT*ND) + GMIN)/qb
       - Id*der(qb)/(qb*qb)).
```

Applying this modification to the p-n junction diode model results in the following `dbe` model.

```
model type dbe
  cut Anode(Va / I)  Cathode(Vb / -I)
  main path P <Anode - Cathode>
  cut Par(VTNF ISv TFv CJEv VJEv MJEv GMINDCv VT)
  parameter AREA
  external qb dqb
  terminal Id u
  local ISe Ic Cdif Cdep denom
  { Electrical equations }
    u = Va - Vb
    I = Id + Ic
    Id = (ISe - ISv)*AREA + GMINDCv*u
    ISe = ISv*exp(u/(VTNF))
    Ic = der(u) * (Cdif + Cdep)
  { Junction capacitance equations }
    Cdif = TFv*((ISe/(VTNF)+GMINDCV)/qb - Id*dqb)/(qb*qb)
                                          {Diffusion cap}
    Cdep = AREA*CJEv/denom                {Depletion cap}
    denom = (1 - (u - 0.5*VT*exp((u-VJEv)/VT))/VJEv)**MJEv
end
```

This implementation of the `dbe` model declares `qb` and its derivative `dqb` as external parameters. Equations for these two globals are defined in the higher level BJT model as presented earlier.

3.9. The Substrate Diode

For the substrate diode, the generic p-n junction diode model can actually be simplified. In the case of the substrate diode, the capacitance due to diffusion becomes negligible and this term can be eliminated from the model. Also, if the substrate diode is always assumed to be reverse biased, the singularity problem at V_{bi} is avoided and the simpler depletion capacitance equation can also be used. With these simplifications, the substrate diode model becomes:

```
model type dbs
  cut Anode(Va / I)  Cathode(Vb / -I)
  main path P <Anode - Cathode>
  cut Par(VTNS ISSv CJSv VJSv MJSv GMINDCv VT)
  parameter AREA
  local ISe Ic u Id Cdep
  { Electrical equations }
    u = Va - Vb
    I = Id + Ic
    Id = (ISe - ISSv)*AREA + GMINDCv*u
    ISe = ISSv*exp(u/(VTNS))
    Ic = der(u)*Cdep
  { Junction capacitance equations }
    Cdep = AREA*CJSv/(1-u/VJSv)**MJSv   {Depletion cap}
 end.
```

3.10. BJT Parameters, Constants, And Temperature Compensation Factors

Many of the parameters in the BJT model are dependent on the device temperature and on the relative difference between device and room temperature. These temperature compensation factors and other global constants are included in the BJTpar model. A single instantiation of this model can then be connected to multiple BJTs in a circuit where the parameters and temperature factors are the same across the BJTs. The complete Dymola BJT model library is listed in Appendix B.

4. VERIFYING THE BJT MODEL

Now that the BJT has been modeled, the validity and usefulness of this model as a tool to describe electrical circuits for the purpose of simulation is investigated. SPICE previously served as the basis for developing many of the equations to build this BJT model. Now SPICE will be used as the baseline to test the validity of this model. All that needs to be done is to model simple transistor circuits in SPICE and in Dymola – using the same parameters in both – and compare the results from simulating each. For this exercise, the two SPICE dialects BBSPICE [6] and PSpice [7] were used.

In BBSPICE, BJTs contain 54 different model parameters. In PSpice, BJTs contain 40 different model parameters. The Dymola model has implemented 48 of the BBSPICE parameters and 34 of the PSpice parameters. The Dymola model has omitted the same six parameters from both BBSPICE and PSpice; of the six remaining, one is a frequency multiplier to determine excess phase, to account for flicker noise, and three provide additional detail for high base-emitter current modeling. For verifying the 48 parameters, we can simplify the process by setting several of them to zero, one, or infinity. These settings effectively "turn-off these parameters and simplify the overall BJT model from a Gummel-Poon type transistor model to an Ebers-Moll type model. With this approach, the number of active parameters can be reduced to sixteen for a very simple BJT model. Assuming that the simulation results of the simple BJT in both Dymola and SPICE are a reasonably close match, we can then turn on the other parameters one by one and further verify the completeness and accuracy of the Dymola BJT model as an equivalent SPICE model.

4.1. Processing Dymola Models For Simulation

Dymola is not a simulation program; it provides no simulation support. Dymola is a modeling language. It facilitates object-oriented modeling, allowing the modeler to formulate very complex continuous system models. After composing the model, the modeler can use Dymola as a model preprocessor to (1) determine the computational causalities, (2) reduce the structural singularities, and (3) solve the algebraic loops arising from the interconnection of the subsystems (submodels). Finally, the modeler can use Dymola as a model generator to produce a state-space description of the continuous system in a variety of simulation languages for model simulation and analysis. While each of the above concepts are summarized in the following paragraphs, they are more fully addressed by Cellier and Elmqvist in [3].

Most of the continuous system simulation languages (CSSLs) in use today employ numerical integration algorithms that are designed to solve state-space models of the type

```
x_dot = f(x,u,t).
```

Because the same expression may appear several times in the various state equations forming a model, it is often more convenient and efficient to assign these expressions to auxiliary variables. Thus, the CSSLs typically support extended state-space model descriptions of the form

```
x_dot = f(x,z,u,t)
z = g(x,z,u,t)
```

where the auxiliary algebraic expressions are assigned to the auxiliary variables, z, and these auxiliary variables must be mutually independent. This mutual independence restriction is to ensure that no algebraic loops are formed in the various equations of the

model. This is important, as the simulation languages employ an equation sorter to establish an executable sequence for the state-space equations, but an equation sorter is unable to resolve the execution sequence of mutually dependent equations. Dymola, as a model preprocessor, facilitates the generation of state-space models that can meet these requirements.

The computational causality of a model determines how the physical laws, as encoded in the model equations, must be interpreted in order to obtain a program that can be executed on a sequential machine using existing numerical algorithms. For example, Dymola allows the resistor model to be described by Ohm's Law: U = R*I. With this description, the current through the resistor seems to "cause" a potential drop across the resistor. However, depending on how the resistor is interconnected with the surrounding components of a more complex circuit model, the Ohm's Law resistor description may require algebraic manipulation to describe the resistor as I = R/U in order to obtain an appropriate state-space model description of the circuit for simulation in a CSSL-type language.

The *partition* command in Dymola uses a set of algorithms to symbolically manipulate the model equations and solve the causality assignment problem. Dymola assumes by default that the state variables of the model are all variables that appear differentiated. Since the target simulation language is expected to use an explicit integration algorithm, Dymola automatically declares all these state variables as *known variables* in accordance with the state-space description format: `x_dot = f(x,u,t)`. Based on this set of *known variables*, Dymola determines the required causality

assignments on the state-space and auxiliary equations in order to solve for the remaining *unknown variables.*

Algebraic loops commonly occur when submodels are interconnected. A simple example is a set of two resistors in series that form a voltage divider. The current through each of the resistors is the same based on the series connection. The current through either of the resistors, however, is dependent on the voltage drop across that resistor. But, the voltage drop across either of the resistors can only be determined if the current through the resistor were already known. CSSL-type languages cannot solve such algebraic loops as the equation sorter cannot determine an executable sequence for mutually dependent equations.

In Dymola, the *partition* command detects algebraic loops when it can no longer uniquely solve the causality assignment problem. At this point, Dymola isolates the involved equations, determines the involved variables, and checks whether the algebraic loop is a linear or nonlinear problem. Given a linear algebraic loop, Dymola solves the loop through symbolic formula manipulation. Dymola can also identify common sub-expressions and will define auxiliary variables and equations for them to further support computational efficiencies in the simulation language model. Depending on the setting of Dymola's compiler options, the partition command may further simplify the problem by eliminating unnecessary equations and expressions such as "`a=0`" and any terms multiplied by "`a`" in other equations.

Structurally singular problems occur in systems that contain more energy storing elements than eigen modi. Such systems are also known as degenerate systems. A

structurally singular linear electrical circuit, for example, contains more capacitors and/or inductors than is indicated by the order of its transfer function. In structurally singular problems, the additional differentiators are true differentiators – they cannot be eliminated from the system or solved as integrators. As with algebraic loops, structurally singular systems often result from the interconnection of subsystems. Dymola can detect structurally singular problems during solution of the causality assignment problem by noting any integrators – energy storage elements – that assume differential rather than integral causality.

Dymola supports solving structurally singular problems with the *differentiate* command. The command initiates an algorithm that assumes all state variables – variables that appear differentiated – are *known.* It then looks for constraints between these variables. For each constraint, it generates new equations that are symbolically differentiated versions of the constraint equations. In the case of a chain of equations resulting from auxiliary variables, all equations in the chain are differentiated. The process is repeated to account for second or higher order derivatives that would also be considered *known.* After executing the *differentiate* command, Dymola no longer assigns any variables to the set of *known variables* automatically. It is up to the user to explicitly declare which variables are to be used as state variables, i.e. *known variables.* In this manner, all constraints are retained and the dimension of the state vector is reduced. The state variables that are removed from the state vector by the differentiation process are computed from the constraints.

4.2. Processing The BJT Circuit Models For Simulation

In verifying the BJT model, three different circuit models were used to test the model: an NPN inverter circuit, a PNP circuit, and a twelve transistor Operational Amplifier (OPAMP) circuit. As can be seen in appendices D, G, and J, the Dymola script files to process each of these circuits follows the same basic approach. This approach can be described as follows.

First, the *differentiate* command is executed to ensure structurally singular problems are avoided by reducing the dimension of the state vector. At the end of the differentiation process, Dymola does not automatically assume any state variables as *known variables*; the state variables must be explicitly chosen. Within the BJT model, the energy storage elements are the three junction diodes. In [5], the capacitive junction current is defined as `Ic = der(Qc)`. Using this equation, defining `Qc` as the state variable for each junction diode would seem to be a natural choice. However, using `Qc` as a state variable leads to a nonlinear algebraic loop, which can only be solved iteratively at run time. Instead, choosing the junction voltage `Ud` as the state variable solves the problem. After differentiation, a linear set of equations for `Ud`, `Id`, and `Ic` results, which can be solved through formula manipulation by the *partition* command. The *partition* command is discussed below. The BJT model described in this paper further avoids this problem by eliminating the junction capacitive charge equation for `Qc`. Instead, the junction diffusion and depletion capacitance equations are defined and used to define the capacitive junction current `Ic`. Thus, the natural choice for the BJT state variables now becomes `Ud`.

After differentiating the model and declaring its state variables, the initial conditions for these state variables are set. In SPICE, the initial conditions may be automatically found and set by iterating on a DC operating point. This solution corresponds to algorithms already employed in modem differential algebraic equation (DAE) solvers, which can find a consistent set of initial conditions. While Dymola embraces the DAE notation, ACSL does not yet support this feature. To compensate, the initial conditions automatically computed in BBSPICE were copied into the Dymola script file to set the initial conditions for ACSL. In PSpice, the internal voltages of a BJT are unavailable to the modeler. This inhibited further checks for consistencies between the three models on initial conditions.

Next, the *partition* command is given to determine causality assignments and solve algebraic loops in the model equations. This processing step results in a set of solved equations. This solved set of equations is then used to generate an ACSL program ready for simulation.

4.3. Numerical Integration Of Electrical Circuits

During the verification of the BJT models, three different numerical integration algorithms – as implemented in ACSL – were tried: 2nd Order Runge-Kutta(ialg=4), 4th Order Runge-Kutta (ialg=5), and Gear(ialg=2). Due to the limitations of these numerical integration algorithms in solving stiff problems, careful selection of the integration step size and certain model parameters – the diode saturation currents IS and ISS; the diode depletion capacitances CJC, CJE, and CJS; and the transit times TF and TR – was required in order to have a model that is stable and solvable without an

excessive number of integration steps. The most sensitive trade-off is between the saturation currents, the depletion capacitances, and the integration step size. Holding the saturation current parameters constant and increasing the capacitance values allows one to also increase the integration step size. However, the increased capacitance also causes the response time of the transistor to increase such that the total simulation time must also increase to capture a complete simulation of the transient analysis. Thus, increasing the capacitance values relative to the saturation currents can cause the number of integration steps required to go up. Decreasing the capacitance allows one to reduce the total simulation time required, but the decrease in capacitance also causes faster transient response times in the transistor. Now the integration step size must be reduced to keep numerical integration of the problem stable. Reducing the integration step size now requires additional integration steps to be computed to integrate over the total simulated time.

In both cases, the required number of integration steps goes up. And as the number of steps goes up, the CPU clock cycles and the required memory for the execution of the simulation goes up. In the case of the NPN and PNP circuit simulations discussed next, the parameter sets used still required 8000 integration steps in ACSL to maintain numerical integration stability while simulating over the full transient analysis. While the saturation current and capacitance parameters used may not reflect any real BJT, these parameters do allow the NPN and PNP circuits to be simulated and the accuracy of the Dymola models verified against the equivalent SPICE models.

Running the simulation on a VAXcluster, a typical simulation time for the NPN model was approximately 5 seconds for BBSPICE and 31 seconds for the Dymola

generated ACSL model. For the PNP model, simulation times were approximately 6 seconds for BBSPICE and 45 seconds for ACSL. For the OPAMP model, simulation times were approximately 15 seconds for BBSPICE and 254 seconds for ACSL. The PSpice models were run on an 80286 based personal computer (no math co-processor) making simulation time comparisons less meaningful. For the record, PSpice simulation times were approximately 4, 5, and 15 minutes for the NPN, PNP, and OPAMP models respectively.

4.4. The NPN BJT

The simple inverter circuit of Figure 1 is used to verify the NPN BJT model. The simulation is a transient analysis of the circuit with a stepped (pulsed) input signal. Appendix D contains the Dymola files used to describe the inverter circuit, the experiment file describing the transient analysis, and the command file to process the first two files in Dymola and generate a file for an ACSL [10] simulation. Appendix C contains the BBSPICE and PSpice files describing this same inverter circuit and transient analysis. The Dymola, BBSPICE, and PSpice models are all configured with the same parameter sets. For conciseness of these appendices, only these simple (most parameters turned off) inverter circuit model descriptions are provided. The simulation results are given in Appendix E and for those simulations with parameters turned on, the parameter name and its value are annotated on the simulation plots.

Figure 7 depicts the simulation plots for the simple NPN inverter circuit - most parameters turned off. For the voltage plots, the input voltage at the base and the output voltage at the collector are plotted. The emitter and substrate voltages are not of much

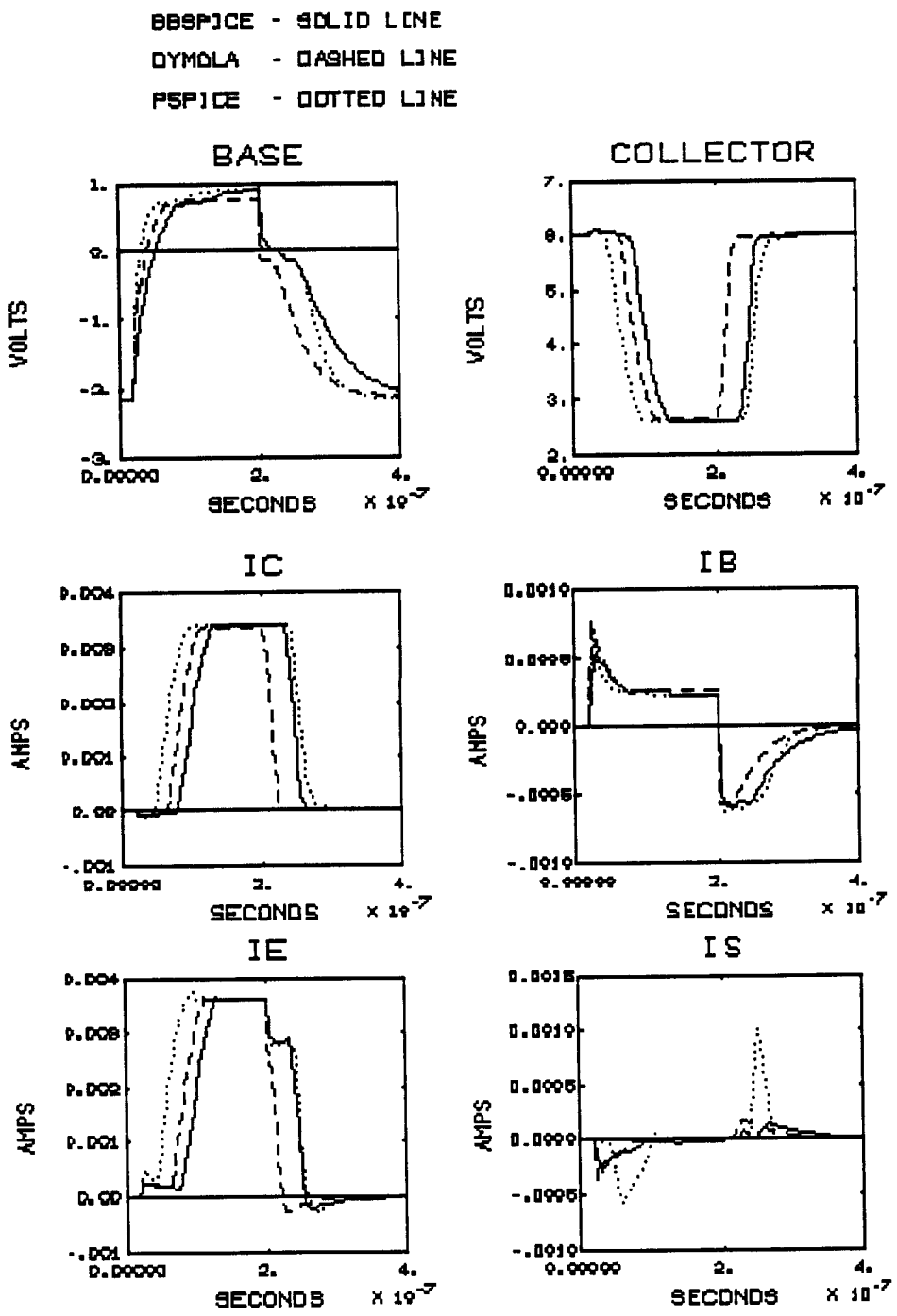


Figure 7. Simple NPN Inverter Simulation Plot

interest as they are held constant by the ground (common) and voltage supply terminals respectively, see Figure 1. The current plots IC, IB, IE, and IS correspond to the collector, base, emitter, and substrate currents respectively. As can readily be seen, simulation of the Dymola NPN BJT model provides a fairly accurate representation of its SPICE counterparts. In fact, a quick glance through Appendix E reveals that as many of the other parameters are turned on the correspondence of this Dymola model to the SPICE models improves.

4.5. The PNP BJT

To model and test the Dymola PNP BJT, a circuit very similar to the NPN inverter circuit was used. The PNP test circuit is depicted in Figure 8. Again, the simulation is a transient analysis of the circuit with a stepped (pulsed) input signal. Appendix F contains the BBSPICE and PSpice files describing this circuit and transient analysis. Appendix G contains the Dymola files modeling the circuit, the transient analysis experiment file, and the command file to process the other two files to generate the ACSL simulation file. The Dymola, BBSPICE, and PSpice models use the same set of parameters. The simulation results are plotted in Appendix H with parameter values annotated in the title when that parameter deviates from the simple sets listed in Appendices F and G. Some of the plots in Appendix H only depict the Dymola and BBSPICE trajectories. In these plots, PSpice was unable to process and simulate the model for that specific parameter set. In particular, PSpice was unable to simulate the model when the two dependent current sources IC0 and IB0 were effectively turned off.

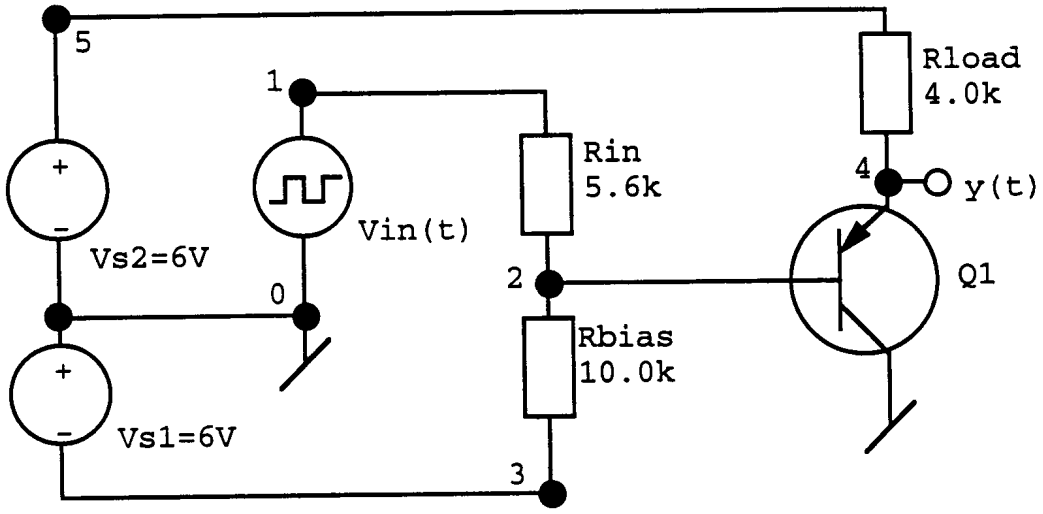


Figure 8. PNP Test Circuit

Figure 9 depicts the simulation plots of the PNP circuit with the ideal maximum forward BETA coefficient, BF, turned on and most other parameters turned off. For the voltage plots, the input voltage at the base and the output voltage at the emitter are plotted. The collector and substrate voltages are not of much interest as they are held constant. The current plots IC, IB, IE, and IS correspond to the collector, base, emitter, and substrate currents respectively. As is readily seen in Figure 9, simulating the Dymola PNP model with BF turned on provides a very accurate representation of its PSpice counterpart and a fairly accurate representation of its BBSPICE counterpart. Looking through the additional results plotted in Appendix H it is seen that only when the parameters BF, BR, ISE, ISC or IKR are turned on does the Dymola PNP model begin to match the BBSPICE and PSpice models.

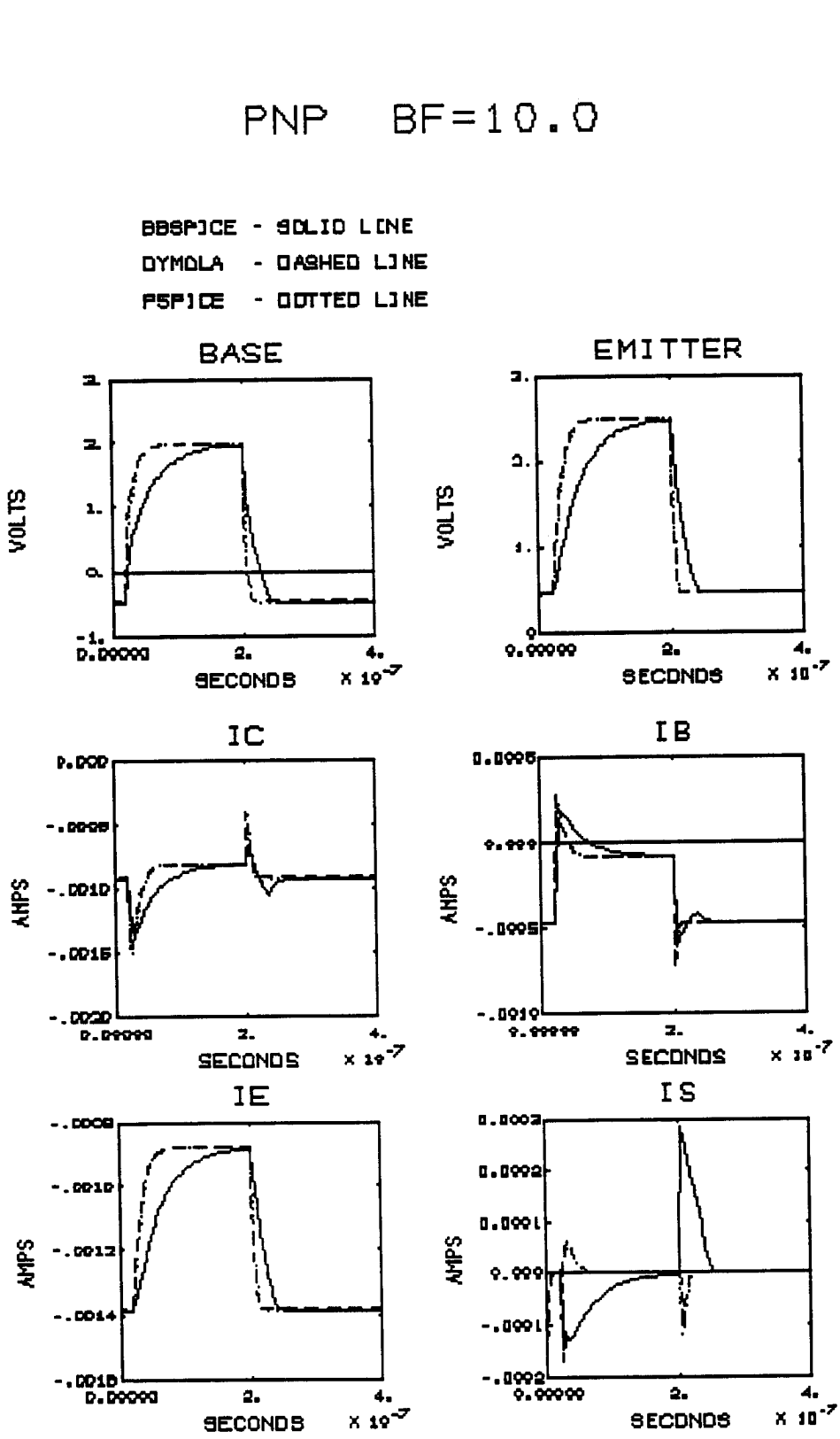


Figure 9. PNP BF=10.0 Circuit Simulation Plot

4.6. NPN And PNP Model Discrepancies

The discrepancies in the plotted trajectories could be the result of numerical integration errors, differences in the models, or both. To check for integration errors, the simulation programs were executed using different integration algorithms. In BBSPICE, the same plots are obtained when using either GEAR or TRAP as the integration algorithm. In Dymola, the same plots are obtained when using 2nd or 4th Order Runge-Kutta or Gear. The student version of PSpice had no options for integration algorithm selection. Thus, numerical integration error can be discounted as the source of discrepancies.

The differences in the resulting trajectories are characteristic of the differences in the equations used to form the BJT models. For the Dymola model, the equations were developed from Cellier's efforts to develop a BJT model in [5], Meta-Software's equations listed in [9], Van Halen's work on an improved junction capacitance equation in [8], and the scanning of BBSPICE source code. While the BBSPICE source code serves as the most comprehensive source of information, it is also the most difficult to read and interpret. For this reason, the BBSPICE source code was used primarily to verify the use of equations extracted from other sources. Thus, some of the equations in the Dymola BJT model are different from those in BBSPICE and PSpice. These equation differences cause the discrepancies in trajectories resulting from the simulations of all three models: Dymola, BBSPICE, and PSpice. It is these model equation differences that also result in PSpice being unable to simulate some of the parameter combinations used in the PNP circuit models.

5. MORE COMPLEX CIRCUIT MODELING

Now that fairly decent NPN and PNP BJT models have been designed, we can explore using these models in more complex circuits. For this purpose, consider the twelve transistor operational amplifier (OPAMP) of Figure 10. This OPAMP consists of both NPN and PNP transistors. Its Dymola description is in Appendix J.

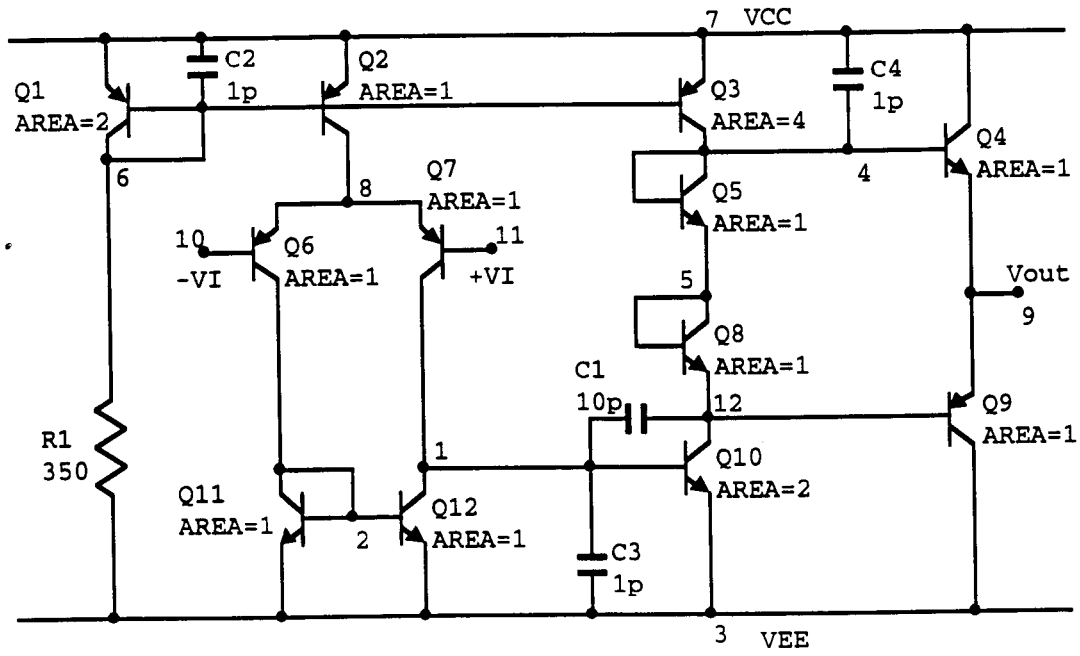


Figure 10. An Operational Amplifier Model

Now consider using the OPAMP of Figure 10 in a higher level circuit. Figure 11 depicts an inverter circuit based on the OPAMP as a sub circuit. The Dymola description of this inverter is also found in Appendix J. Appendix I contains the BBSPICE and PSpice equivalent descriptions.

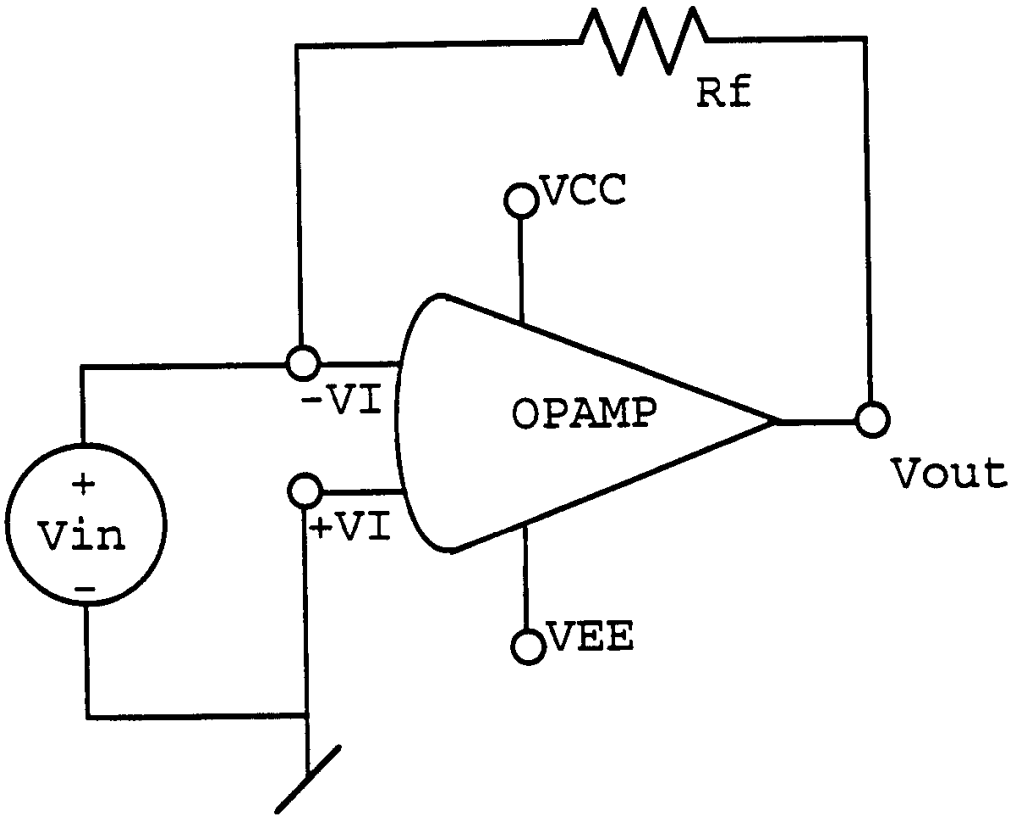


Figure 11. An OPAMP Inverter Circuit

The OPAMP inverter circuit is simulated over a stepped input signal for a transient signal analysis. The results of this simulation are shown in Figure 12 and in Appendix K. From these simulation results, it is seen that the Dymola model provides a fairly accurate description of the circuit when compared to the SPICE models.

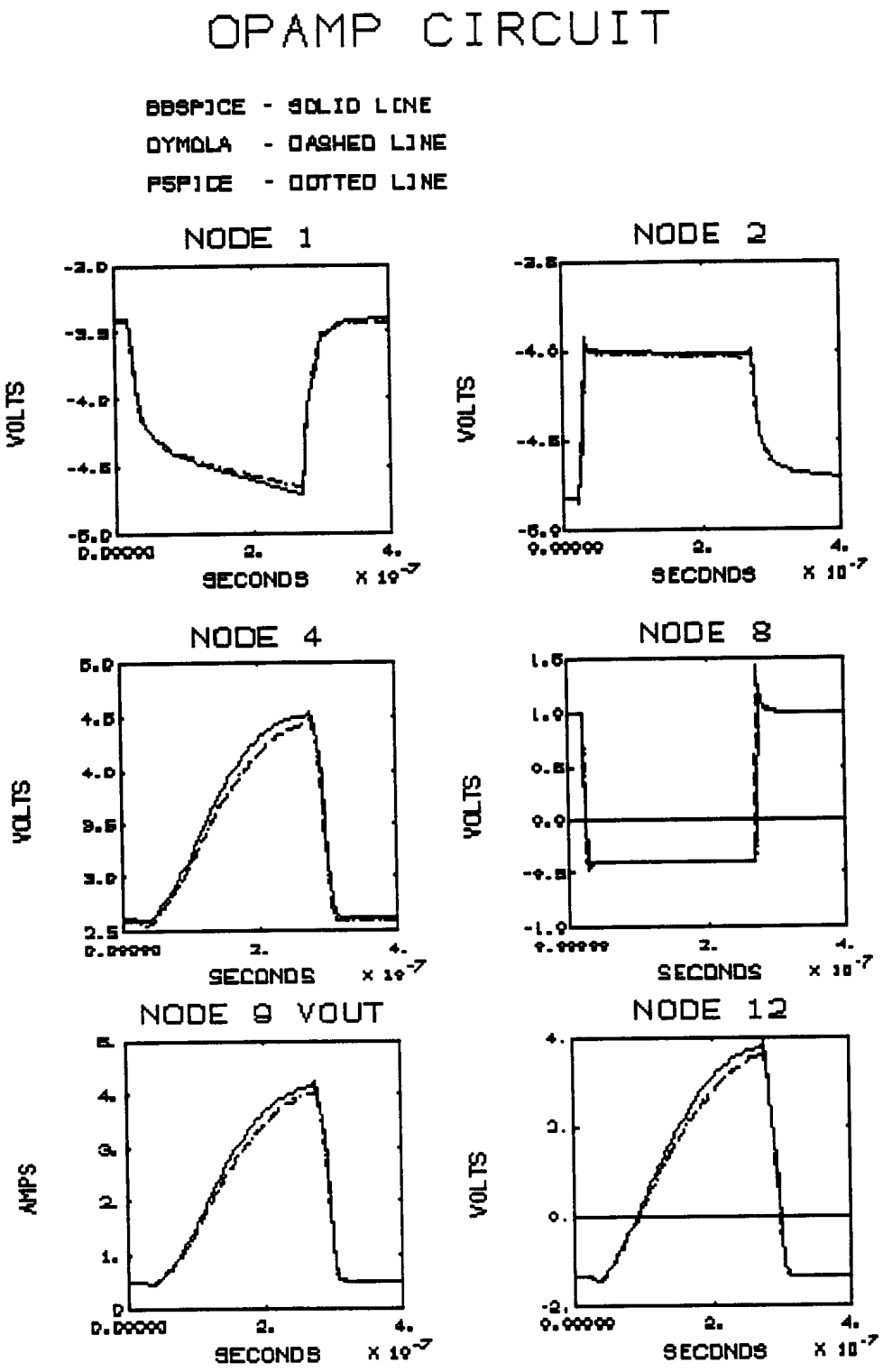


Figure 12. OPAMP Simulation Plot

6. CONCLUSIONS

Dymola is a very powerful, object-oriented, continuous-system modeling tool. Its automated formula manipulation capabilities allow it to determine causality assignments, reduce structural singularities, and solve algebraic loops formed from the coupling of subsystems. From these formula manipulations, Dymola can generate state-space models for system simulation and analysis in a variety of different simulation languages. Dymola currently supports ACSL, DESIRE, Simnon, SimuLink, and DSblock.

The Dymola BJT models developed in this study are very readable and quickly understandable descriptions of NPN and PNP transistors. The NPN model is a fairly accurate representation of its SPICE equivalents. The accuracy of the PNP model, however, is dependent upon the complexity of the model used. For the more complex PNP model, i.e. the parameters BF, BR, ISE, ISC, IKF, IKR turned on, the model provides a very accurate representation of its SPICE equivalents.

The Dymola BJT model can be easily adapted to an improved set of equations that better model real devices. One possible adaptation may be to better account for temperature dynamics in a system. Most SPICE dialects, if not all, assume a static temperature environment. The Dymola BJT could be adapted to model temperature dynamics by adding in a set of equations to account for the power dissipated by the components and to account for heat flow between components. Similarly, the BJT could be easily adapted for interconnection with other types of subsystems – chemical, mechanical, or biological – providing more diversity in the types of systems modeled and simulated.

Dymola has been proven to support the modeling of complex electrical circuits and is capable of preprocessing them into state-space models ready for simulation. Simulation of circuits involving nonlinear devices like BJTs, however, requires numerical integration algorithms capable of handling these numerically stiff – very negative eigenvalued – problems. The algorithms available in ACSL proved to be poorly suited to deal with the numerical stiffness of these BJT circuit models. Even the Gear algorithm performed poorly on these highly nonlinear models. All algorithms required very small step sizes, which then required additional CPU cycles and memory to numerically integrate and save the results of the problem. This was the sole reason for the careful BJT parameter selection described in subsection 4.3. In order to overcome these difficulties, a DAE formulation may be more suitable, a formulation that is already supported by Dymola, but not yet by ACSL, the simulation language that was available to me, since the process of converting the model to an explicit ODE form destroys some of the natural scarcity of the model equations, and since the DAE solver is believed to be better suited to suppress spurious solutions of the numerical integration.

APPENDIX A
DYMOLA MODEL OF
ELECTRICAL COMPONENTS

```
{ File: elcomp.dym
  Library of electrical components.
  Author: Hilding Elmqvist
  Date:    Jan, 1992
  Latest revision: May 20, 1992
  Version: 1.1
}
```

```
model type resistor
 parameter R
 cut A (Va / i) B (Vb / -i)
 main cut C [A B]
 main path AB <A - B>
 local u
  u = Va - Vb
  R*i = u
end
```

```
model type capacitor
  parameter C
  cut A (Va / i) B (Vb / -i)
  main cut C [A B]
  main path AB <A - B>
  local u
    u = Va - Vb
    C*der(u) = i
end



model type varcapacitor
  terminal C
  cut A (Va / i) B (Vb / -i)
  main cut C [A B]
  main path AB <A - B>
  local u udot Q
    u = Va - Vb
    Q = C*u
    der(Q) = i
    udot = der(u)
end
```

```
model type inductor
  parameter L
  cut A (Va / i) B (Vb / -i)
  main cut C [A B]
  main path AB <A - B>
  local u
    u = Va - Vb
    L*der(i) = u
end
```

```
model type diode
  parameter I0 K
  cut A (Va / i) B (Vb / -i)
  main cut C [A B]
  main path AB <A - B>
  local u
    u = Va - Vb
    i = I0*(exp(K*u) - 1)
end
```

```
model type voltage
  cut A (Va / i) B (Vb / -i)
  main cut C [A B]
  main path AB <A - B>
  terminal U0
    U0 = Vb - Va
end



model type current
  cut A (. / i) B (. / -i)
  main cut C [A B]
  main path AB <A - B>
  terminal I0
    i = I0
end



model type common
  cut A (V / . ) B (V / . )
  main cut C [A B]
  main path AB <A - B>
    V = 0
end
```

APPENDIX B
DYMOLA MODEL OF
BIPOLAR JUNCTION TRANSISTORS

```
{ File: BJT.dym
  Library of bipolar junction transistors.
  Author:    Francois Cellier, ECE, University of Arizona
  Date:      1991
  Version:   2.1
  Last revision by: Daryl Hild May 1993
  Reference: Francois Cellier: "Continuous System Modeling"
             Springer Verlag, 1991
  Status:    Model works, but poses a very stiff problem.
             Rewrote diode Ic equations in terms of der(u).
             Consolidated all but AREA parameter into BJTpar
             so that parameters for multiple BJTs can be set
             once with BJTpar.
}

model type varresistor
  cut A(Va / I) B(Vb / -I)
  main path P <A - B>
  cut Par(Rv)
  parameter AREA
  local u
    u = Va - Vb
    u = I*Rv/AREA
end
```

```
model type rbb
  cut A(Va / I) B(Vb / -I)
  main path P <A - B>
  cut Par(RBv RBMv IRBv PiSq)
  parameter AREA
  external qb
  local u R {z tz}
    u = Va - Vb
  { R = if IRBv > 0.0 ->
      then (RBMv + 3.0*(RBv-RBMv)*(tz-z)/(z*tz*tz))/AREA ->
      else (RBMv + (RBv - RBMv)/qb)/AREA
    z = if IRBv > 0.0 ->
      then (-1 + sqrt(1 + 144*I/(PiSq*IRBv*AREA))) ->
        /(24*sqrt(I/(IRBv*AREA))/(PiSq)) ->
      else 0.0
    tz = if IRBv > 0.0 then tan(z) else 0.0
  }
    R = (RBMv + (RBv - RBMv)/qb)/AREA
    R*I = u
End


model type Csource
  cut A(. / I) B(. / -I)
  main path P <A - B>
  terminal I0
  I = I0
end
```

```
model type wire {Added to set IS direction into BJT}
  cut A(Va / I)  B(Vb / -I)
  main path P <A - B>
    Va = Vb
end


model type dbc
  cut Anode(Va / I) Cathode(Vb / -I)
  main path P <Anode - Cathode>
  cut Par(VTNR ISv TRv CJCv XCJCv VJCv MJCv GMINDCv VT)
  parameter AREA
  terminal Id u Ix
  local ISe Ic Cdif Cdep denom
  { Electical equations }
    u = Va - Vb
    I = Id + Ic
    Id = (ISe - ISv)*AREA + GMINDCv*u
    ISe = ISv*exp(u/(VTNR))
    Ic = der(u)*(Cdif+XCJCv*Cdep) (Internal cap current}
    Ix = der(u)*(1-XCJCv)*Cdep    (External cap current)
  { Junction capacitance equations }
    Cdif = TRv * (ISe/(VTNR) + GMINDCv)  {Diffusion cap}
    Cdep = AREA*CJCv/denom               {Depletion cap}
    denom = (1 - (u - 0.5*VT*exp((u-VJCv)/VT))/VJCv)**MJCv
end
```

```
model type dbe
  cut Anode(Va / I)  Cathode(Vb / -I)
  main path P <Anode - Cathode>
  cut Par(VTNF ISv TFv CJEv VJEv MJEv GMINDCv VT)
  parameter AREA
  external qb dqb
  terminal Id u
  local ISe Ic Cdif Cdep denom
  { Electrical equations }
    u = Va - Vb
    I = Id + Ic
    Id = (ISe - ISv)*AREA + GMINDCv*u
    ISe = ISv*exp(u/(VTNF))
    Ic = der(u) * (Cdif + Cdep)
  { Junction capacitance equations }
    Cdif = TFv*((ISe/(VTNF)+GMINDCv)/qb - Id*dqb)/(qb*qb)
                                          {Diffusion cap}
    Cdep = AREA*CJEv/denom                (Depletion cap}
    denom = (1 - (u - 0.5*VT*exp((u-VJEv)/VT))/VJEv)**MJEv
end
```

```
model type dbs
  cut Anode(Va / I)  Cathode(Vb / -I)
  main path P <Anode - Cathode>
  cut Par(VTNS ISSv CJSv VJSv MJSv GMINDCv VT)
  parameter AREA
  local ISe Ic u Id Cdep
  { Electrical equations }
    u = Va - Vb
    I = Id + Ic
    Id = (ISe - ISSv)*AREA + GMINDCv*u
    ISe = ISSv*exp(u/(VTNS))
    Ic = der(u)*Cdep
  { Junction capacitance equations )
    Cdep = AREA*CJSv/(1-u/VJSv)**MJSv   {Depletion cap.)
end
```

```
model type BJT
  submodel (varresistor) rcc(AREA=AREA) ree(AREA=AREA)
  submodel rbb(AREA=AREA)
  submodel dbc(AREA=AREA)
  submodel dbe(AREA=AREA)
  submodel dbs(AREA=AREA)
  submodel (Csource) ice0 ibe0 cbcx
  submodel wire

  parameter AREA=1.0

  cut C(VC / 1C) B(VB / IB) E(VE / - IE) S (VS / ISUB)
  main cut CBES [C B E S]
  cut par (BFv BRv ISv ISSv VTNF VTNR VTNS ->
    CJCv CJEv CJSv XCJCv MJCv MJEv MJSv VJCv VJEv VJSv ->
    ISCv ISEv VTNC VTNE VAFv VARv IKFv IKRv ->
    IRBv RBv RBMv RCv REv TFv TRv GMINDCv VT PiSq)
  path BE<B-E> BC<B-C> BS<B-S> CE<C-E> CS<C-S> ES<E-S>
  path EB<E-B> CB<C-B> SB<S-B> EC<E-C> SC<S-C> SE<S-E>

  node IntC IntB IntE IntS
  node rccPar(RCv)
  node reePar(REv)
  node rbbPar(RBv RBMv IRBv PiSq)
  node dbcPar(VTNR ISv TRv CJCv XCJCv VJCv MJCv GMINDCv VT)
  node dbePar(VTNF ISv TFv CJEv VJEv MJEv GMINDCv VT)
  node dbsPar(VTNS ISSv CJSv VJSv MJSv GMINDCv VT)

  local vbc vbe ibc ibe icn ien IB0 IC0 q1 q2 dq1 dq2
  internal qb dqb
```

```
  { Define parameters and globals for components }
    connect rccPar at rcc:Par
    connect reePar at ree:Par
    connect rbbPar at rbb:Par
    connect dbcPar at dbc:Par
    connect dbePar at dbe:Par
    connect dbsPar at dbs:Par
  { Define frequently used internal voltages and currents }
    vbc = dbc.u
    vbe = dbe.u
    ibc = dbc.Id
    ibe = dbe.Id
    cbcx.I0 = dbc.Ix
  { Compute the base charge }
    q1 = 1/(1 - vbc/VAFv - vbe/VARv)
    q2 = (ibc/IKRv + ibe/IKFv) / AREA
    qb = q1*0.5*(1 + (1 + 4*q2)**0.5)
  { Compute derivatives of base charge }
    dq1 = I/(VARv*q1*q1)
    dq2 = (ien+ISEv)/(VTNC*IKFv)
    dqb = qb*dq1/q1 + q1*dq2/(1+4*q2)**0.5
  ( Compute the nonlinear current sources }
    icn = AREA*ISCv*(exp(vbc/(VTNC))-1)
    ien = AREA*ISEv*(exp(vbe/(VTNE))-1)
    IC0 = (ibe-ibc)/qb - ibc/BRv - icn
    IB0 = ibe/BFv + ibc/BRv + ien + icn - ibe
    Ice0.I0 = IC0
    Ibe0.I0 = IB0
end
```

```
model type (BJT) NPNIateral
    connect  rbb from    B to IntB
    connect  rcc from    C to IntC
    connect  ree from IntE to E
    connect  dbc from IntB to IntC
    connect  dbe from IntB to IntE
    connect  dbs from IntB to IntS
    connect wire from    S to IntS
    connect cbcx from    B to IntC
    connect ibe0 from IntB to IntE
    connect ice0 from IntC to IntE
end



model type (BJT) NPNvertical
    connect  rbb from    B to IntB
    connect  rcc from    C to IntC
    connect  ree from IntE to E
    connect  dbc from IntB to IntC
    connect  dbe from IntB to IntE
    connect  dbs from IntS to IntC
    connect wire from    S to IntS
    connect cbcx from    B to IntC
    connect ibe0 from IntB to IntE
    connect ice0 from IntC to IntE
end
```

```
model type (BJT) PNPlateral
    connect  rbb from    B to IntB
    connect  rcc from    C to IntC
    connect  ree from IntE to E
    connect  dbc from IntC to IntB
    connect  dbe from IntE to IntB
    connect  dbs from IntS to IntB
    connect wire from    S to IntS
    connect cbcx from    B to IntC
    connect ibe0 from IntE to IntB
    connect ice0 from IntE to IntC
end



model type (BJT) PNPvertical
    connect  rbb from    B to IntB
    connect  rcc from    C to IntC
    connect  ree from IntE to E
    connect  dbc from IntC to IntB
    connect  dbe from IntE to IntB
    connect  dbs from IntC to IntS
    connect wire from    S to IntS
    connect cbcx from    B to IntC
    connect ibe0 from IntE to IntB
    connect ice0 from IntE to IntC
end
```

```
model type BJTpar
  parameter ->
    {DC Model Parameters} ->
    BF=100.0   BR=1.0   IS=1.0E-16  ISS=0.0          ->
    NF=1.0     NR=1.0   N3=1.0      GMINDC=1.0E-12   ->
  {Low Current BETA Degradation Effect Parameters}   ->
    ISC=0.0    ISE=0.0    NC=2.0    NE=1.5           ->
  {Base Width Modulation Parameters}                 ->
    VAF=9E+30  VAR=9E+30                             ->
  {High Current BETA Degradation Effect Parameters}  ->
    IKF=9E+30  IKR=9E+30                             ->
  {Parasitic Resistor Parameters}                    ->
    IRB=0.0    RB=0.0     RBM=0.0   RC=0.0   RE=0.0  ->
  {Junction Capacitor Parameters}                    ->
    CJC=0.0    CJE=0.0    CJS=0.0   XCJC=1.0         ->
    MJC=0.33   MJE=0.33   MJS=0.33  {0.3<=m<=0.5}    ->
    VJC=0.75   VJE=0.75   VJS=0.75                   ->
  {Transit Time Parameters}                          ->
    TF=0.0      TR=0.0                               ->
  {Temperature Compensation and Area Parameters}     ->
    TRB1=0.0    TRM1=0.0    TRC1=0.0  TRE1=0.0       ->
    TRB2=0.0    TRM2=0.0    TRC2=0.0  TRE2=0.0       ->
    TNOM=25.0   TEMP=25.0   XTI=3.0   XTB=0.0   EG=1.16
  main cut Par (BFv BRv ISv ISSv VTNF VTNR VTNS ->
    CJCv CJEv CJSv XCJCv MJCv MJEv MJSv VJCv VJEv VJSv  ->
    ISCv ISEv VTNC VTNE VAFv VARv IKFv IKRv ->
    IRBv RBv RBMv RCv REv TFv TRv GMINDCv VT PiSq)
  local Tref Tdev DTemp DTempSq RTemp Rxtb EGref EGdev ->
    facin ISfact VDfact
```

```
constant Pi = 3.14159  ->
  Charge=1.6021918E-19  (Electron Charge} ->
  Boltz=1.3806226E-23   {Boltzmann's constant} ->
  CtoK=273.15           {Celius to Kelvin conversion} ->
  GapC1=7.02E-4  {1st bandgap correction factor Silicon} ->
  GapC2=1108.0   {2nd bandgap correction factor Silicon}
{ Define temperature globals and constants }
  PiSq = Pi*Pi
  Tref = TNOM + CtoK        {Reference (room) temperature}
  Tdev = TEMP + CtoK        (Device temperature}
  DTemp   = Tdev - Tref     (Temp Delta}
  DTempSq = DTemp*DTemp     (Temp Delta Squared}
  RTemp   = Tdev/Tref       (Temp Ratio}
  Rxtb    = RTemp**XTB      {Temp Ratio exponentiated}
  VT = Boltz*Tdev/Charge    {Thermal Voltage}
{ Electron Gap of Silicon at Tref and Tdev }
  EGref = EG - GapC1*Tref*Tref/(Tref + GapC2)
  EGdev = EG - GapC1*Tdev*Tdev/(Tdev + GapC2)
{ Temperature compensation factors }
  facln  = (RTemp-1)*EGdev/VT + XTI*ln(RTemp)
  ISfact = exp(facln)
  VDfact = EGdev - EGref*RTemp - 3*VT*ln(RTemp)
{ Define temperature adjusted resistors }
  RCv  =  RC*(1 + TRC1*DTemp + TRC2*DTempSq)
  REv  =  RE*(1 + TRE1*DTemp + TRE2*DTempSq)
  RBv  =  RB*(1 + TRB1*DTemp + TRB2*DTempSq)
  RBMv = RBM*(1 + TRM1*DTemp + TRM2*DTempSq)
  IRBv = IRB
```

```
{ Define temperature adjusted diode parameters }
  VTNF = VT*NF
  VTNR = VT*NR
  VTNS = VT*NS
  ISv  = IS*ISfact
  ISSv = ISS*ISfact
  VJCv = VJC*RTemp + VDfact
  VJEv = VJE*RTemp + VDfact
  VJSv = VJS*RTemp + VDfact
  CJCv = CJC*(1 + MJC*(1 - VJCv/VJC + 4.0E-4*DTemp))
  CJEv = CJE*(1 + MJE*(1 - VJEv/VJE + 4.0E-4*DTemp))
  CJSv = CJS*(1 + MJS*(1 - VJSv/VJS + 4.0E-4*DTemp))
  MJCv = MJC . MJEv = MJE
  MJSv = MJS
  XCJCv = XCJC
  GMINDCv = GMINDC
  TFv = TF
  TRv = TR
{ Define nonlinear current source parameters }
  VTNC = VT*NC
  VTNE = VT*NE
  BFv = BF*Rxtb
  BRv = BR*Rxtb
  ISCv = ISC*exp(facln/NC)/Rxtb
  ISEv = ISE*exp(facin/NE)/Rxtb
  VAFv = VAF
  VARv = VAR
  IKFv = IKF
  IKRv = IKR
end
```

APPENDIX C
SPICE MODELS OF THE
NPN INVERTER CIRCUIT

This appendix contains the BBSPICE and pSpice models of the NPN SIMPLE Inverter Circuit. The BBSPICE model is as follows.

```
NPN INVERTER

VI      1   0         PULSE(0V 6V 20nS 0.1nS 0.1nS 180nS 400nS)
VS1     0   3         6V
VS2     5   0         6V

Q1      4   2   0   5   PROC35.N   1
RIN     1   2         5.6k
RBIAS   2   3         10.0k
RLOAD   5   4         1.0k

.MODEL PROC35.N NPN
+ ISC=0.0fA   ISE=0.0fA    BF=1.0E30    BR=1.0E30
+ NC=1.0      NE=1.0       VAF=1.0E30   VAR=1.0E30
+ IKF=1.0E30 IKR=1.0E30
+ NR=1.0    IS=0.11nA   TR=1.0pS   VJC=0.64   CJC=3.6pF    MJC=0.5
+ NF=1.0                TF=1.0pS   VJE=0.77   CJE=5.7pF    MJE=0.5
+ NS=1.0    ISS=0.11nA             VJS=0.75   CJS=11.0pF   MJS=0.5
+ XCJC=1.0
+ RB=1000.0  TRB1=0.005   TRB2=0.0005
+ RBM=500.0  TRM1=0.005   TRM2=0.0005   IRB=0.0
+ RC=750.0   TRC1=0.005   TRC2=0.0005
+ RE=13.3    TRE1=0.005   TRE2=0.0005
+ EG=1.16    XTB=0.0     XTI=3.0      SUBS=-1
+ LEVEL=2    TLEV=0
```

```
.TRAN 5.0nS 400.0nS
.TEMP 25
.OPTIONS TNOM=25 BJTMAX=2 DCAP=2 METHOD=TRAP
.OPTIONS CO=80 NOPAGE CURNTS NUMDGT=4
.PARAM SUB=0
.PRINT TRAN V(2) V(4) I1(Q1) I2(Q1) I3(Q1) I4(Q1)
.PLOT TRAN V(2) V(4) I1(Q1) I2(Q1) I3(Q1) I4(Q1)
.RAMP 10uS 12uS NOOP
.END
```

The pSpice model is as follows.

```
NPN INVERTER

VI      1   0          PULSE(0V 6V 20nS 0.1nS 0.1nS 180nS 400nS)
VS1     0   3          6V
VS2     5   0          6V

Q1      4   2   0   3  PROC35.N  1
RIN     1   2          5.6k
RBIAS   2   3          10.0k
RLOAD   5   4          1.0k

.MODEL PROC35.N NPN
+ ISC=0.0fA  ISE=0.0fA   BF=1.0E30   BR=1.0E30
+ NC=1.0     NE=1.0      VAF=1.0E30  VAR=1.0E30
+ IKF=1.0E30 IKR=1.0E30
+ NR=1.0   IS=0.11nA  TR=1.0pS  VJC=0.64  CJC=3.6pF   MJC=0.5
+ NF=1.0              TF=1.0pS  VJE=0.77  CJE=5.7pF   MJE=0.5
+ NS=1.0   ISS=0.11nA           VJS=0.75  CJS=11.0pF  MJS=0.5
+ XCJC=1.0
```

```
+ RB=1000.0   TRB1=0.005   TRB2=0.0005
+ RBM=500.0   TRM1=0.005   TRM2=0.0005   IRB=0.0
+ RC=750.0    TRC1=0.005   TRC2=0.0005
+ RE=13.3     TRE1=0.005   TRE2=0.0005
+ EG=1.16     XTB=0.0      XTI=3.0

.TRAN 5.0nS 400.0nS
.TEMP 25
.OPTIONS TNOM=25
.PRINT TRAN V(2) V(4) IC(Q1) IB(Q1) IE(Q1) IS(Q1)
.PLOT TRAN V(2) V(4) IC(Q1) IB(Q1) IE(Q1) IS(Q1)
.probe
.END
```

APPENDIX D
DYMOLA FILES MODELING THE NPN INVERTER CIRCUIT

```
{ File: npn.txt)

NPN Inverter Circuit

This problem contains the following files:

  npn.txt    : This file of explanations
  eicomp.dym : Dymola library file of electrical components
  bjt.dym    : Dymola library file of BJT transistors
  hpn.dym    : The NPN inverter circuit model description
  npn.ctl    : Dymola experiment description (ACSL)
  npn.dcm    : Dymola command file for this problem

During execution of the dymola command file "npn.dcm" the
following files are generated:

  npn.sol : A history file of the dymola execution
            to include a listing of the solved equations
  npn.csl : The dymola generated ACSL program to simulate
            the inverter model
```

```
{ File:  npn.dym

  Logical inverter with NPNlateral transistor.
  Author:  Daryl Hild
  Date:    May 1993
  Version: 1.2
}
model inverter

  submodel (NPNlateral) Q1(AREA=1.0)
  submodel (BJTpar) Q1par  ->
 (ISC=0.0      ISE=0.0      ->
  NC=1.0       NE=1.0       ->
  BF=1.0E+30   BR=1.0E+30   ->
  VAF=1.0E+30  VAR=1.0E+30  ->
  IKR=1.0E+30  IKF=1.0E+30  ->
  NR=1.0 IS=0.11E-9 TR=1E-12 VJC=0.64 CJC=3.6E-12  MJC=0.5 ->
  NF=1.0            TF=1E-12 VJE=0.77 CJE=5.7E-12  MJE=0.5 ->
  NS=1.0 ISS=0.11E-9         VJS=0.75 CJS=11.0E-12 MJS=0.5 ->
  XCJC=1.0                                         ->
  RB=1000.0 TRB1=0.005  TRB2=0.0005                ->
  RBM=500.0 TRM1=0.005  TRM2=0.0005 IRB=0.0        ->
  RC=750.0  TRC1=0.005  TRC2=0.0005                ->
  RE=13.3   TRE1=0.005  TRE2=0.0005                ->
  XTI=3.0 XTB=0.0 GMINDC=1.0E-12  TNOM=25.0  TEMP=25.0)
  submodel (voltage) Ein Es1 Es2
  submodel (resistor) Rin(R=5.6E+3) Rbias(R=10E+3) ->
    Rload(R=1E+3)
  submodel common
```

```
  input A Us1 Us2
  output Y IC IB IE IS

  { Set Q1 parameters }
    connect Q1par at Q1:par

  { Connect up circuit }
    connect common - Ein - Rin -   ->
      ( (Rbias - Es1) //  Q1..BE ) - common
    connect common - Es2 - Rload - Q1..CE
    connect Q1..BS - Rload

  { Set inputs and outputs }
    Es1.U0 = Us1
    Es2.U0 = Us2
    Ein.U0 = A
    Y = Rload.Vb
    IC = Q1.IC
    IB = Q1.IB
    IE = Q1.IE
    IS = Q1.ISUB
end
```

```
{  File:  npn.ctl }

cmodel

  maxtime tmax=400.0E-9
  cinterval cint=0.5E-10
  input 3, A(depend,Ain), Us1(depend,Us1in), Us2(depend,Us2in)

  INITIAL
    algorithm ialg=5
    nsteps nstp=1000
    CONSTANT thigh=20.0E-9, tlow=200.0E-9
    Ain=0.0
    Us1in=6.0
    Us2in=6.0
    schedule high .at. thigh
    schedule low .at. tlow
  END

  DISCRETE high
    Ain=6.0
  END
  DISCRETE low
    Ain=0.0
  END

end
```

```
{ File: npn.dcm }
set
set LogCommands on
outfile npn.sol

{ DEFINE THE MODEL }
enter model
@elcomp.dym
@bjt.dym
@npn.dym

{ PROCESS THE MODEL }
differentiate
variable state Q1::dbc.u
variable state Q1::dbe.u
variable state Q1::dbs.u
variable value Q1::dbc.u=-8.1538
variable value Q1::dbe.u=-2.1538
variable value Q1::dbs.u=-8.1538
partition
set Statistics on
output solved equations

{ GENERATE AN ACSL MODEL FOR SIMULATION }
set LogCommands off
set ACSLold on
language acsl
enter experiment
@npn.ctl
outfile npn.csl
output program
outfile
```

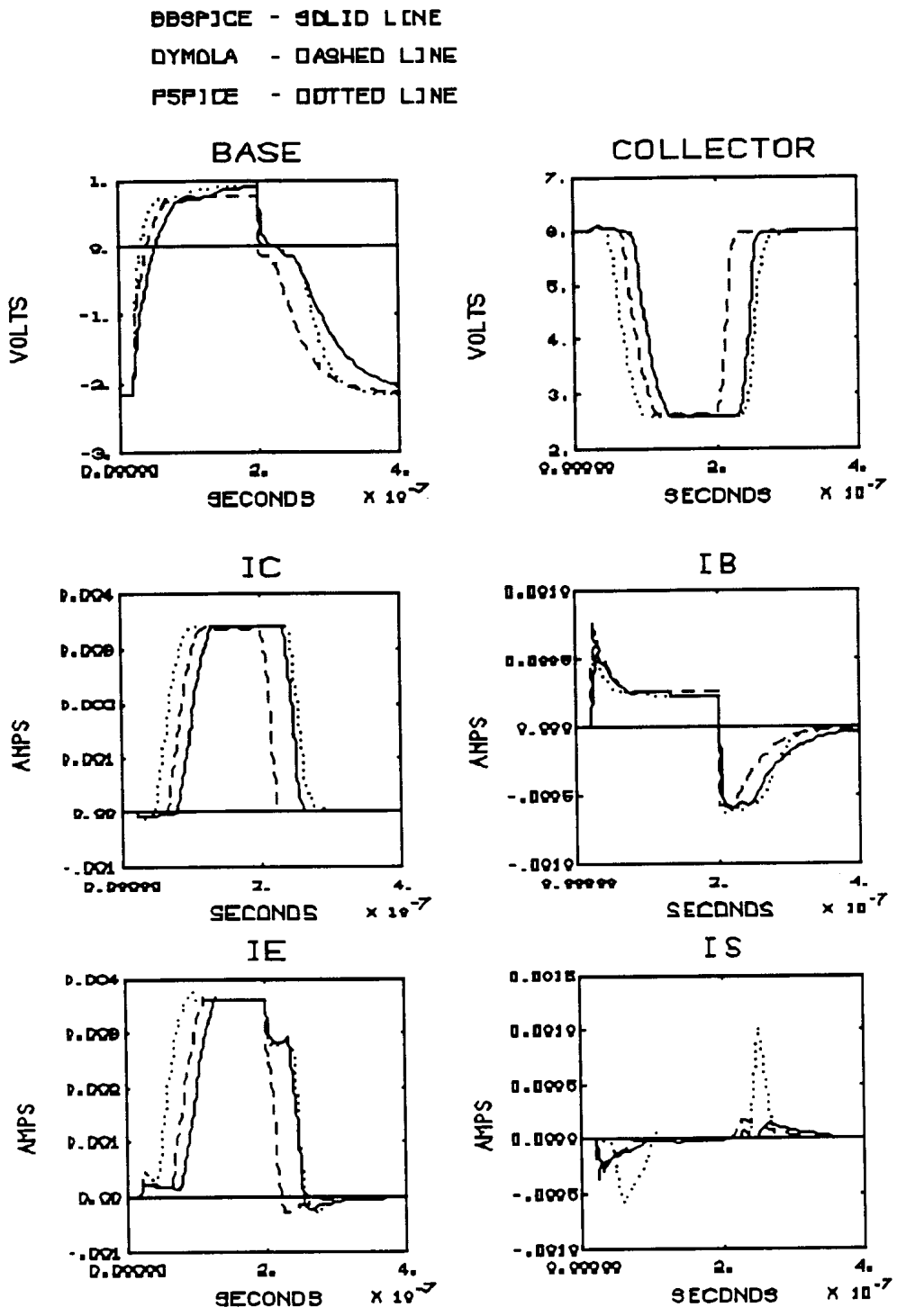
APPENDIX E
SIMULATION PLOTS OF THE NPN INVERTER CIRCUIT

This appendix contains the simulation plots of the NPN Inverter Circuit. The first set of plots, NPN SIMPLE, are based on the parameters given in the BBSpice and pSpice models found in Appendix C and the Dymola model of Appendix D. For convenience, these parameter settings – for the NPN SIMPLE plot – are repeated below. The remaining plots use the same parameters with deviations being noted in the plot title.

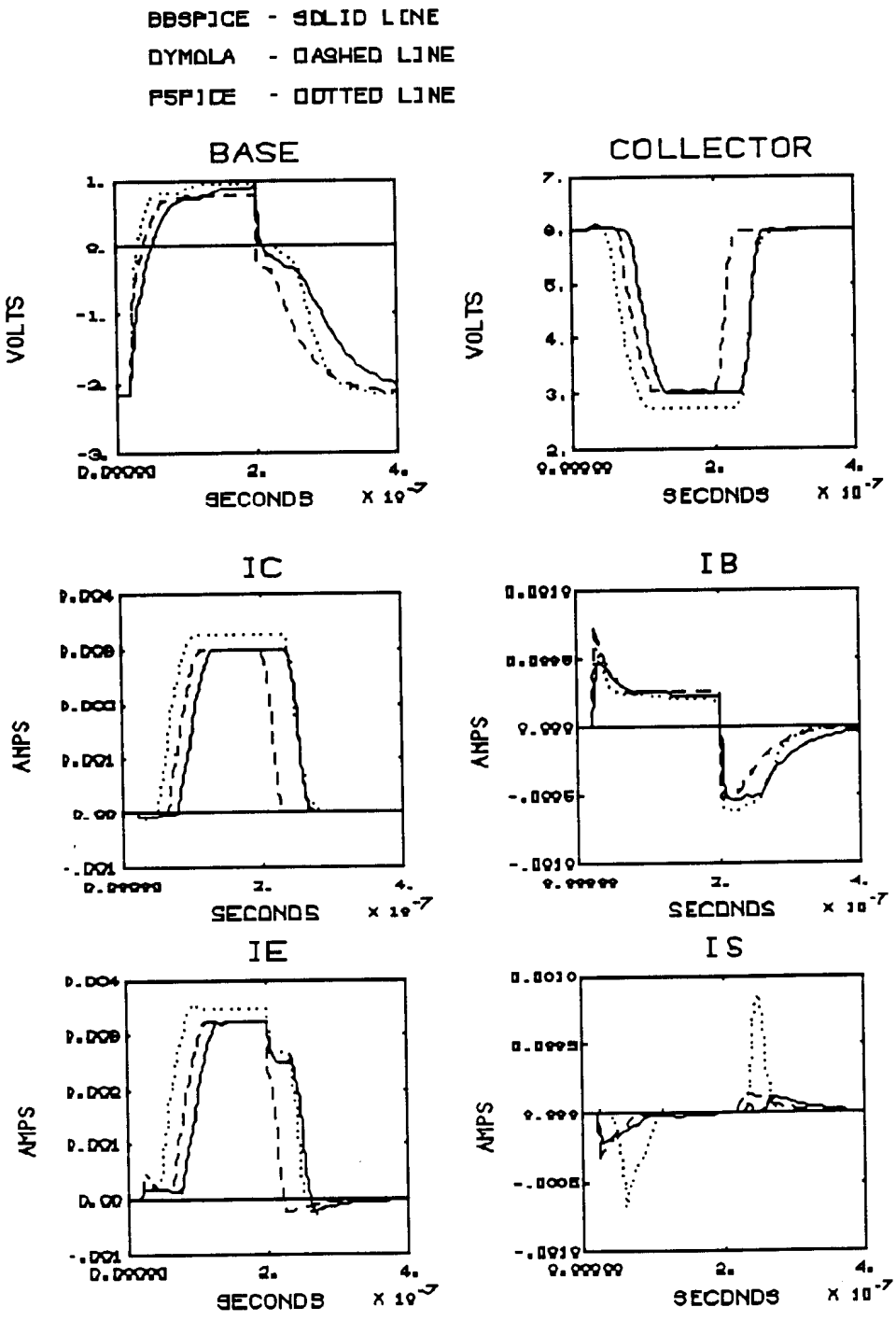
The basic parameter settings are as follows.

```
NPN
AREA=1.0
ISC=0.0         ISE=0.0
NC=1.0          NE=1.0
BF=1.0E+30      BR=1.0E+30
VAF=1.0E+30     VAR=1.0E+30
IKR=1.0E+30     IKF=1.0E+30
NR=1.0    IS=0.11E-9    TR=1E-12 VJC=0.64 CJC=3.6E-12    MJC=0.5
NF=1.0                  TF=1E-12 VJE=0.77 CJE=5.7E-12    MJE=0.5
NS=1.0    ISS=0.11E-9             VJS=0.75 CJS=11.0E-12  MJS=0.5
XCJC=1.0
RB=1000.0 TRB1=0.005    TRB2=0.0005
RBM=500.0 TRM1=0.005    TRM2=0.0005 IRB=0.0
RC=750.0  TRC1=0.005    TRC2=0.0005
RE=13.3   TRE1=0.005    TRE2=0.0005
XTI=3.0   XTB=0.0       GMINDC=1.0E-12  TNOM=25.0  TEMP=25.0
```

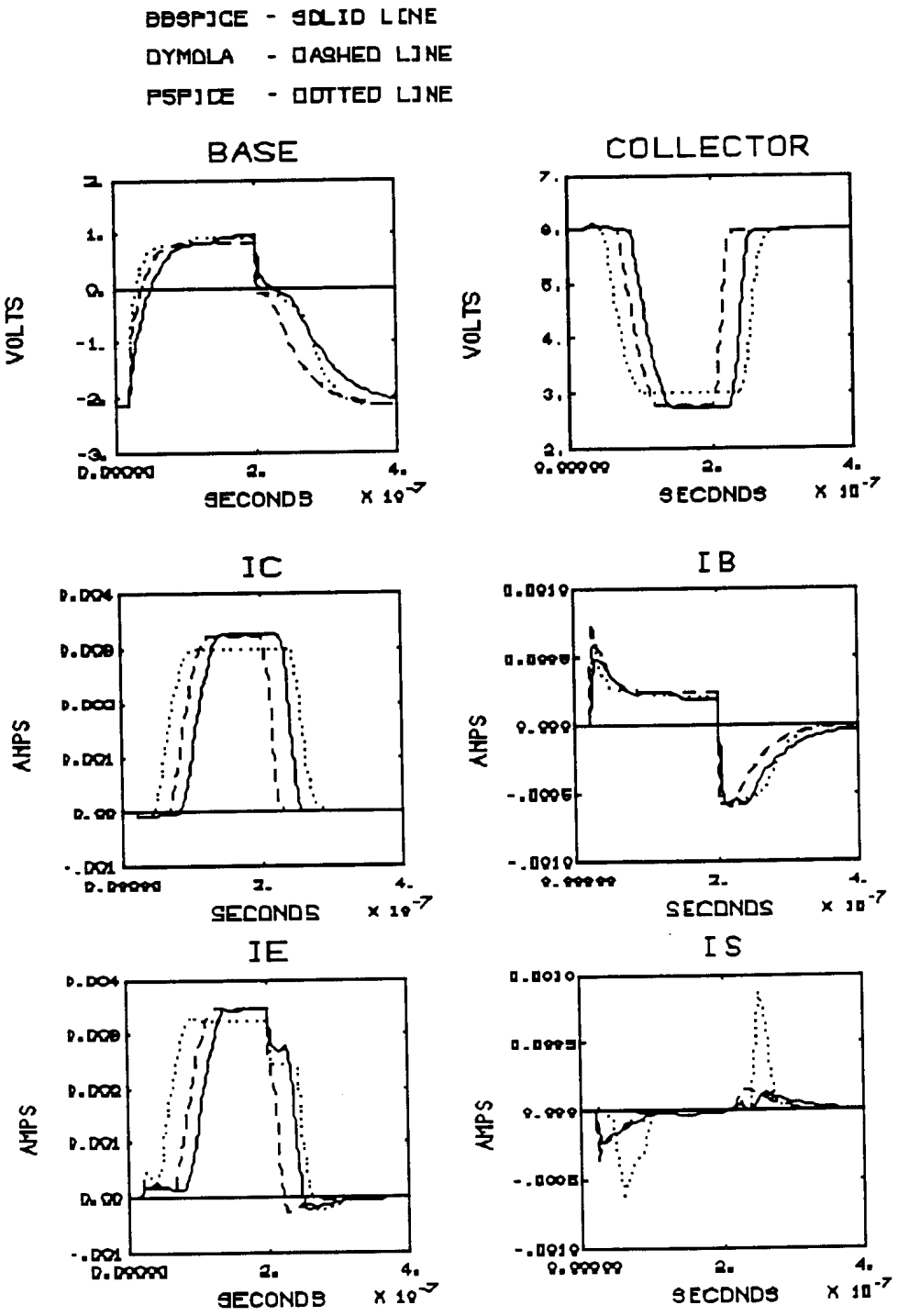
NPN SIMPLE


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
COLLECTOR
VOLTS
SECONDS
IC
AMPS
SECONDS
IB
AMPS
SECONDS
IE
AMPS
SECONDS
IS
AMPS
SECONDS

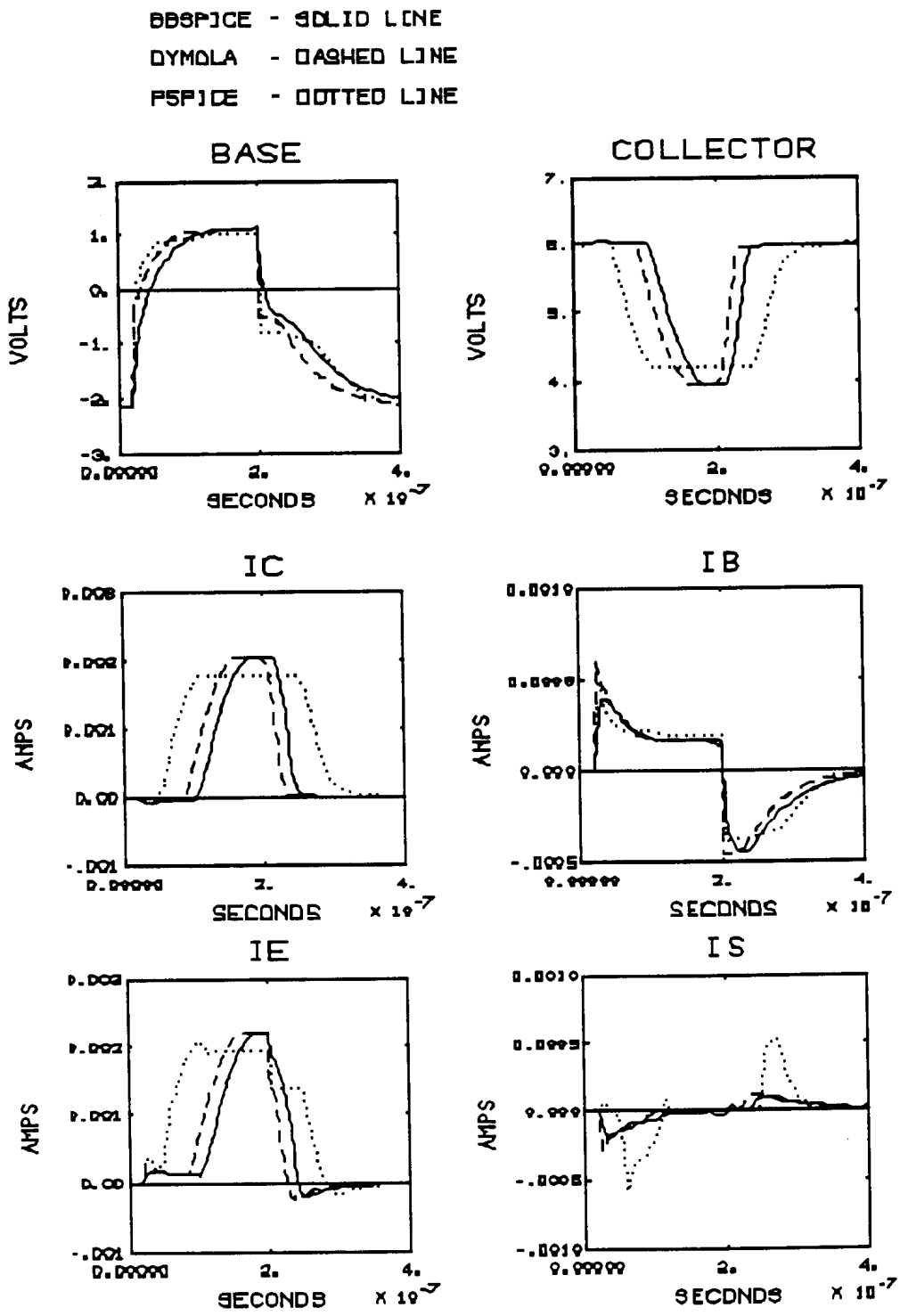
NPN TNOM=5.0


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
X 10^-7
COLLECTOR
VOLTS
SECONDS
X 10^-7
IC
AMPS
SECONDS
X 10^-7
IB
AMPS
SECONDS
X 10^-7
IE
AMPS
SECONDS
X 10^-7
IS
AMPS
SECONDS
X 10^-7

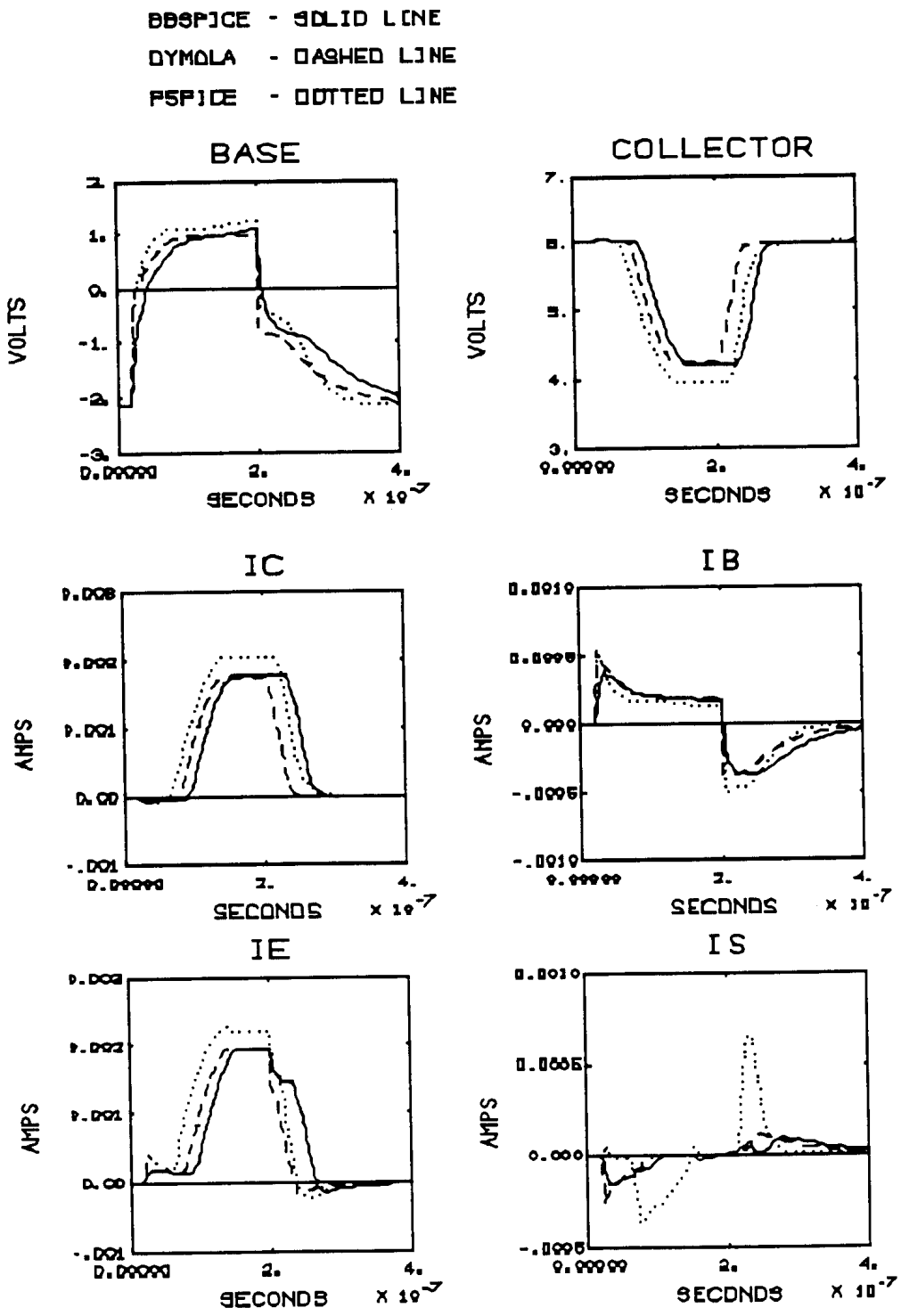
NPN TNOM=45.0


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
X 10^-7
COLLECTOR
VOLTS
SECONDS
X 10^-7
IC
AMPS
SECONDS
X 10^-7
IB
AMPS
SECONDS
X 10^-7
IE
AMPS
SECONDS
X 10^-7
IS
AMPS
SECONDS
X 10^-7

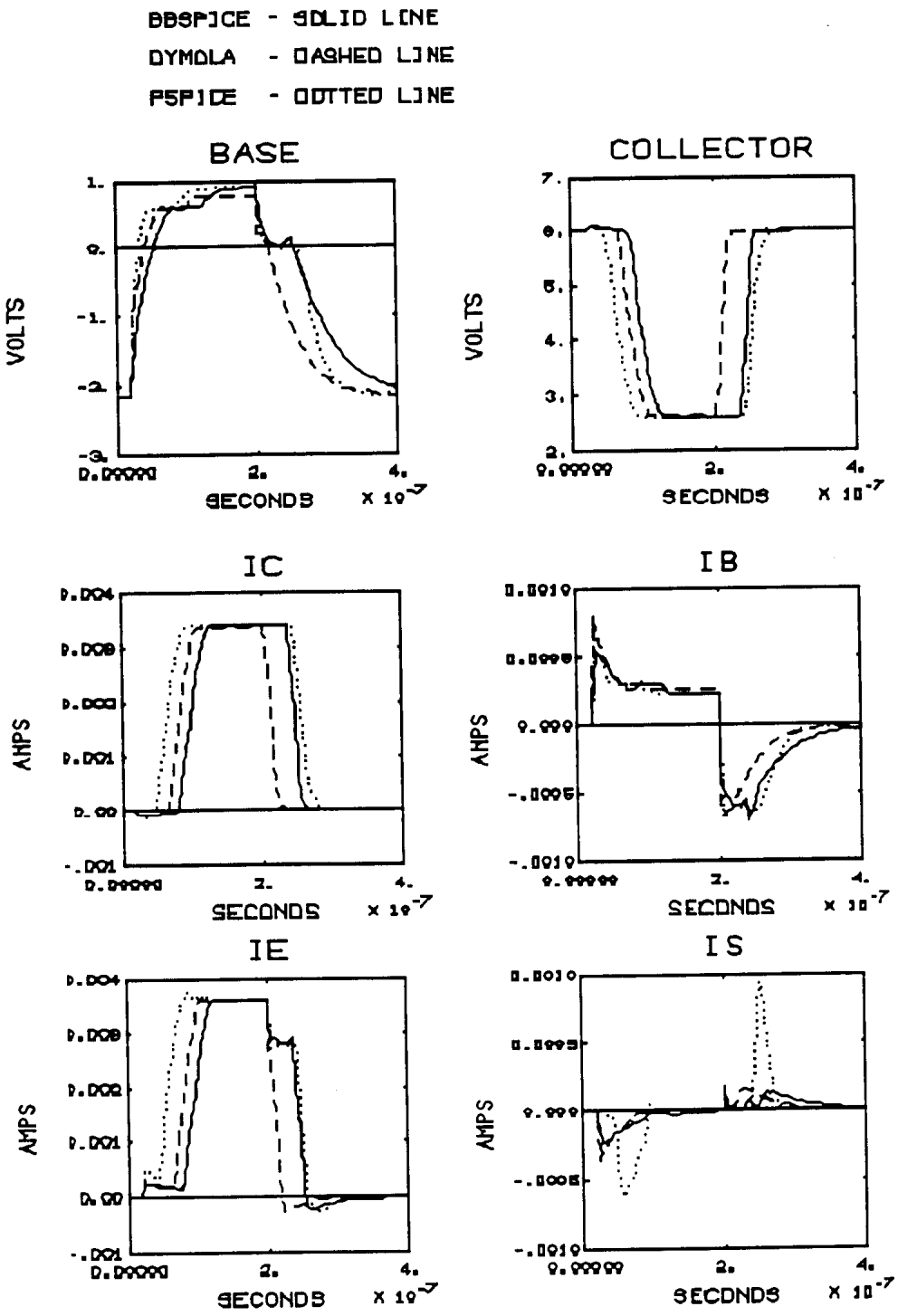
NPN TEMP=-35.0


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
COLLECTOR
VOLTS
SECONDS
IC
AMPS
SECONDS
IB
AMPS
SECONDS
IE
AMPS
SECONDS
IS
AMPS
SECONDS

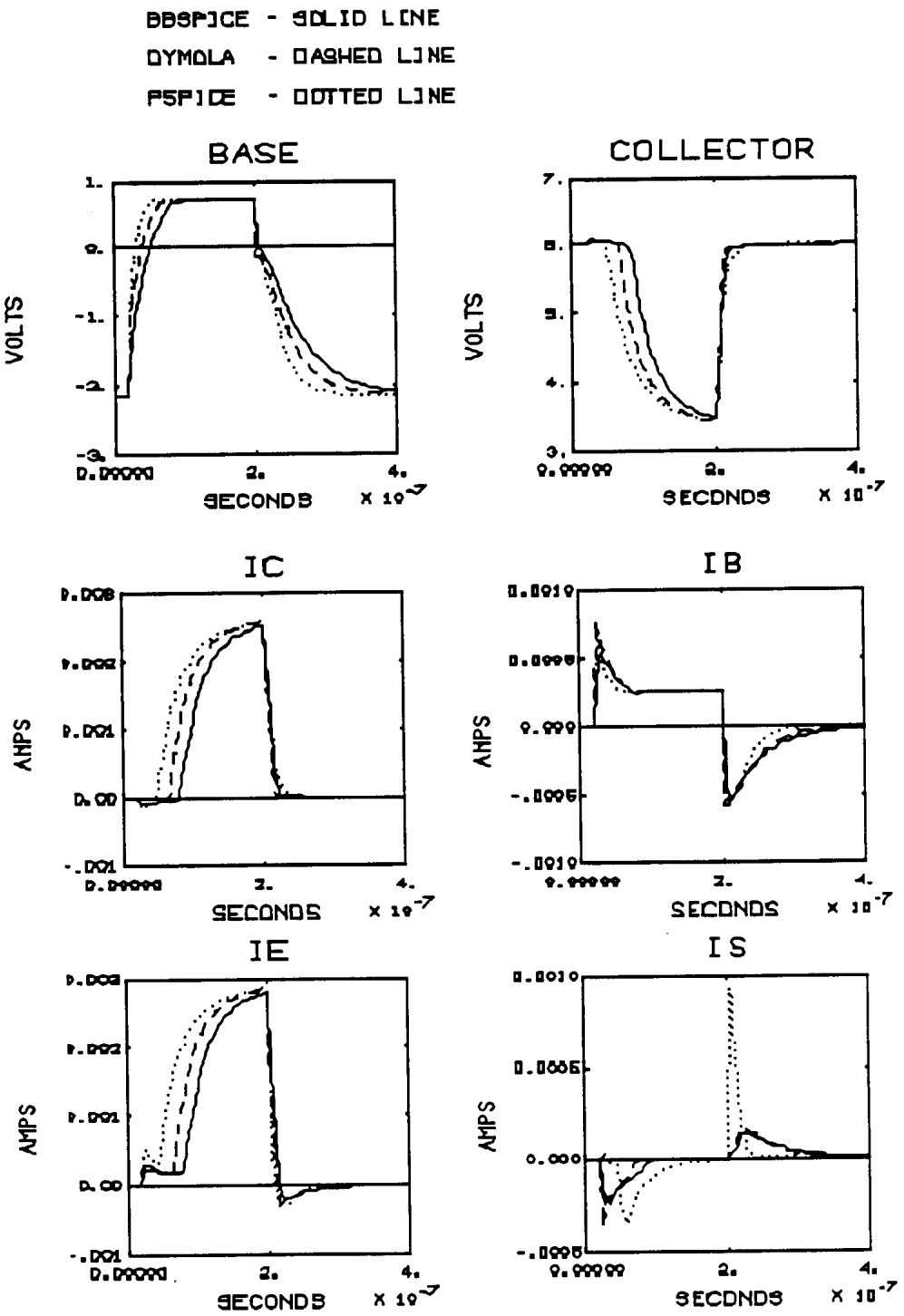
NPN TEMP=85.0


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
COLLECTOR
VOLTS
SECONDS
IC
AMPS
SECONDS
IB
AMPS
SECONDS
IE
AMPS
SECONDS
IS
AMPS
SECONDS

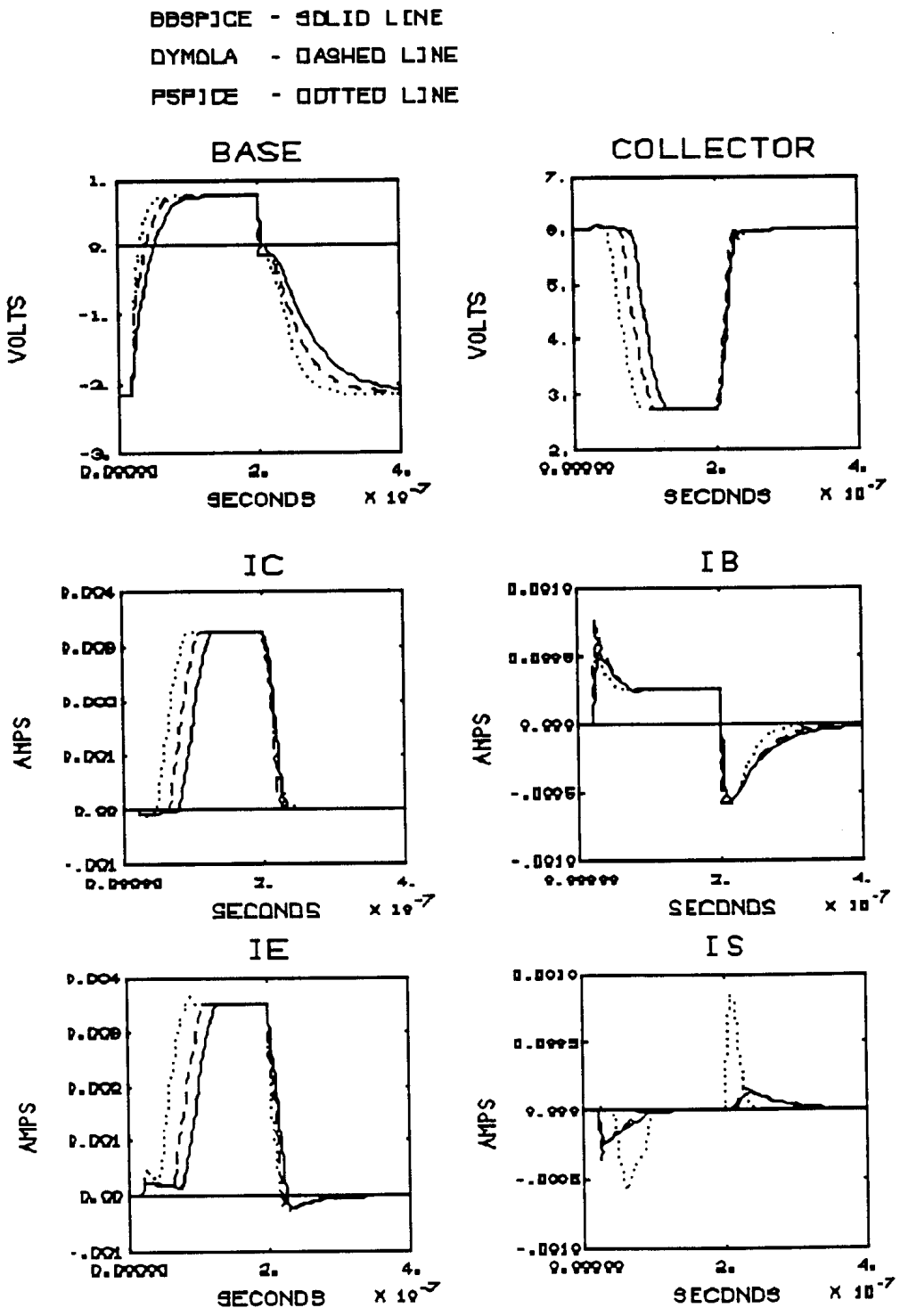
NPN XCJC=0.4


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
X 10^-7
COLLECTOR
VOLTS
SECONDS
X 10^-7
IC
AMPS
SECONDS
X 10^-7
IB
AMPS
SECONDS
X 10^-7
IE
AMPS
SECONDS
X 10^-7
IS
AMPS
SECONDS
X 10^-7

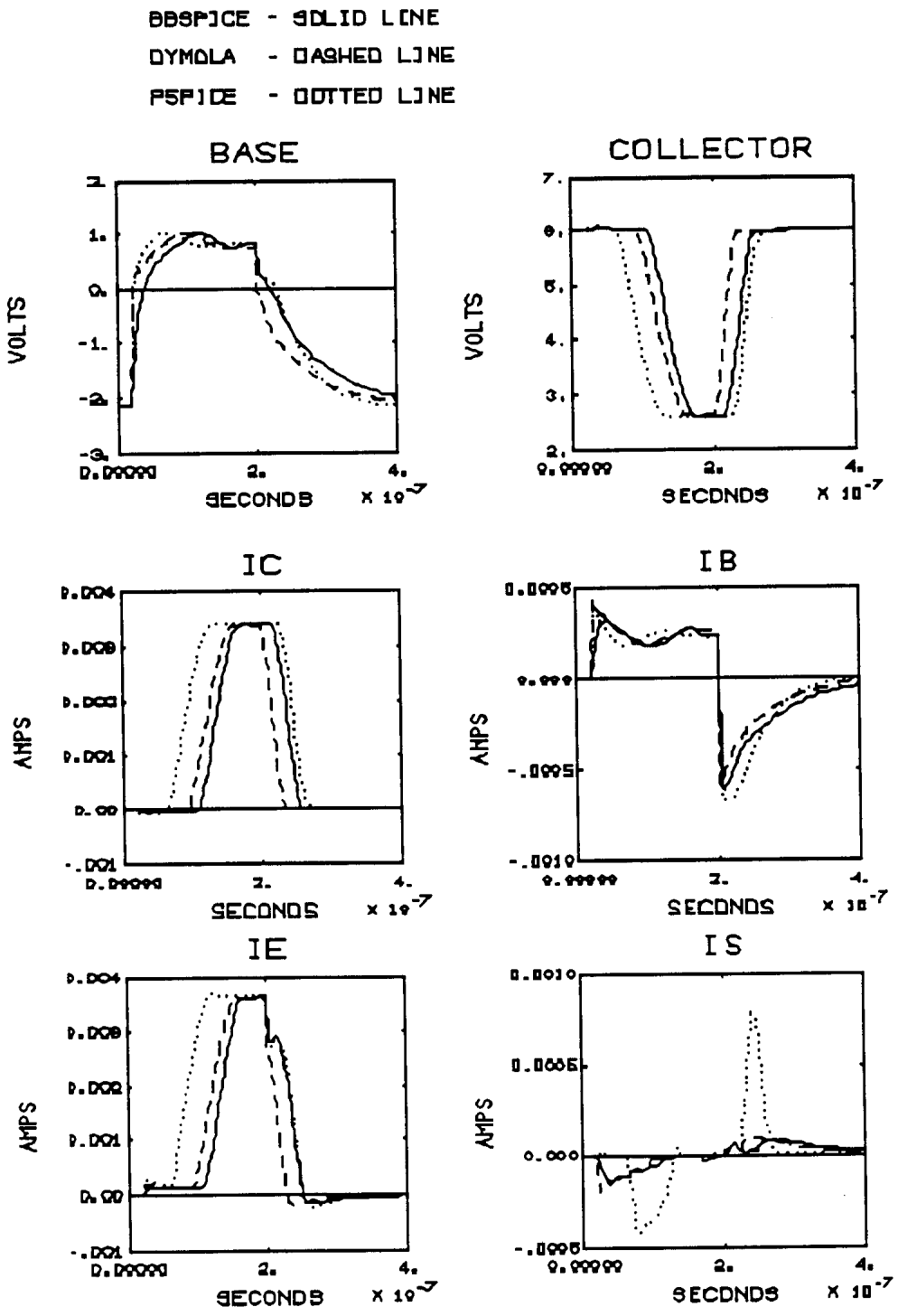
NPN BF=10.0


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
COLLECTOR
VOLTS
SECONDS
IC
AMPS
SECONDS
IB
AMPS
SECONDS
IE
AMPS
SECONDS
IS
AMPS
SECONDS

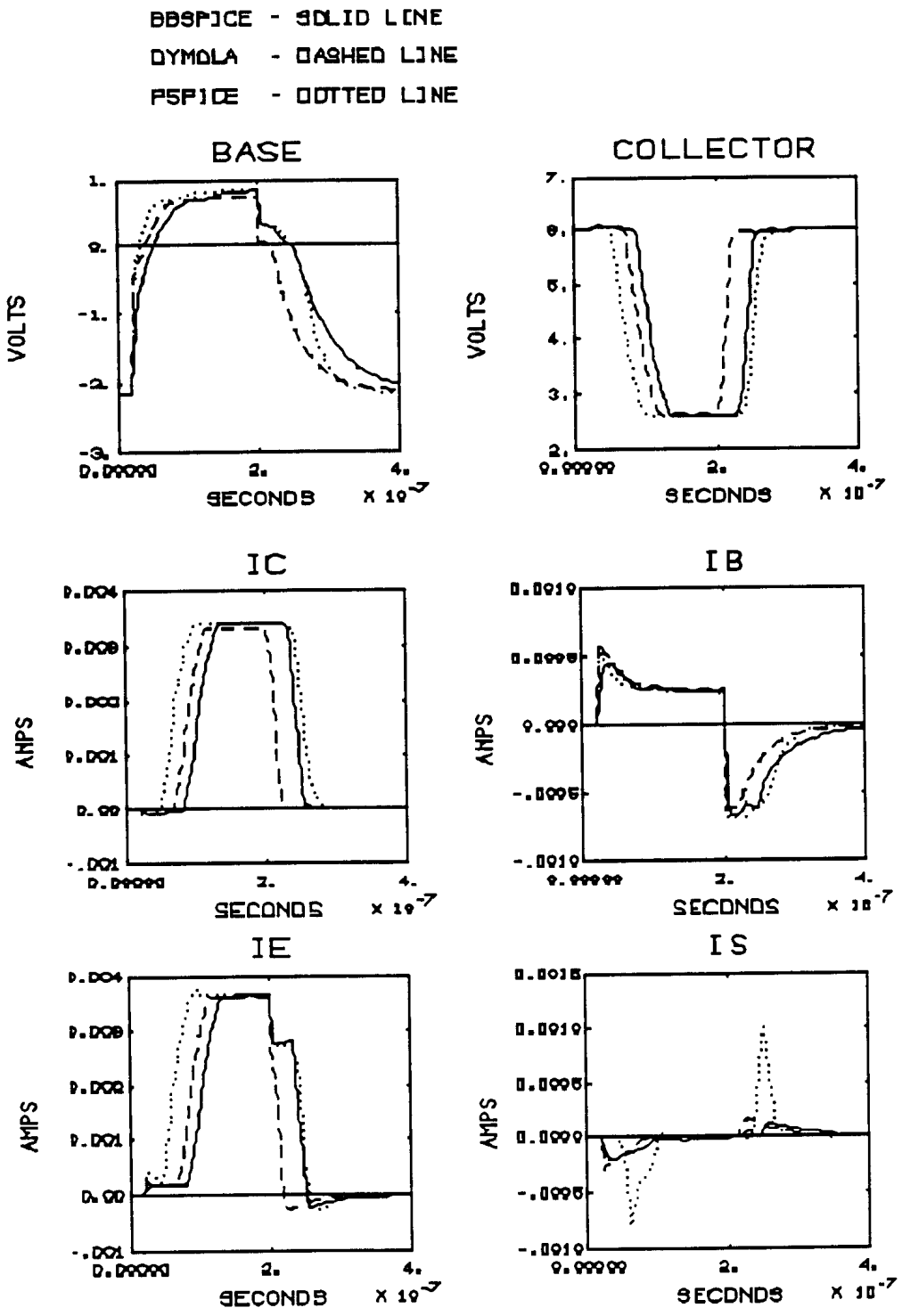
NPN BR=0.001


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
X 10^-7
COLLECTOR
VOLTS
SECONDS
X 10^-7
IC
AMPS
SECONDS
X 10^-7
IB
AMPS
SECONDS
X 10^-7
IE
AMPS
SECONDS
X 10^-7
IS
AMPS
SECONDS
X 10^-7

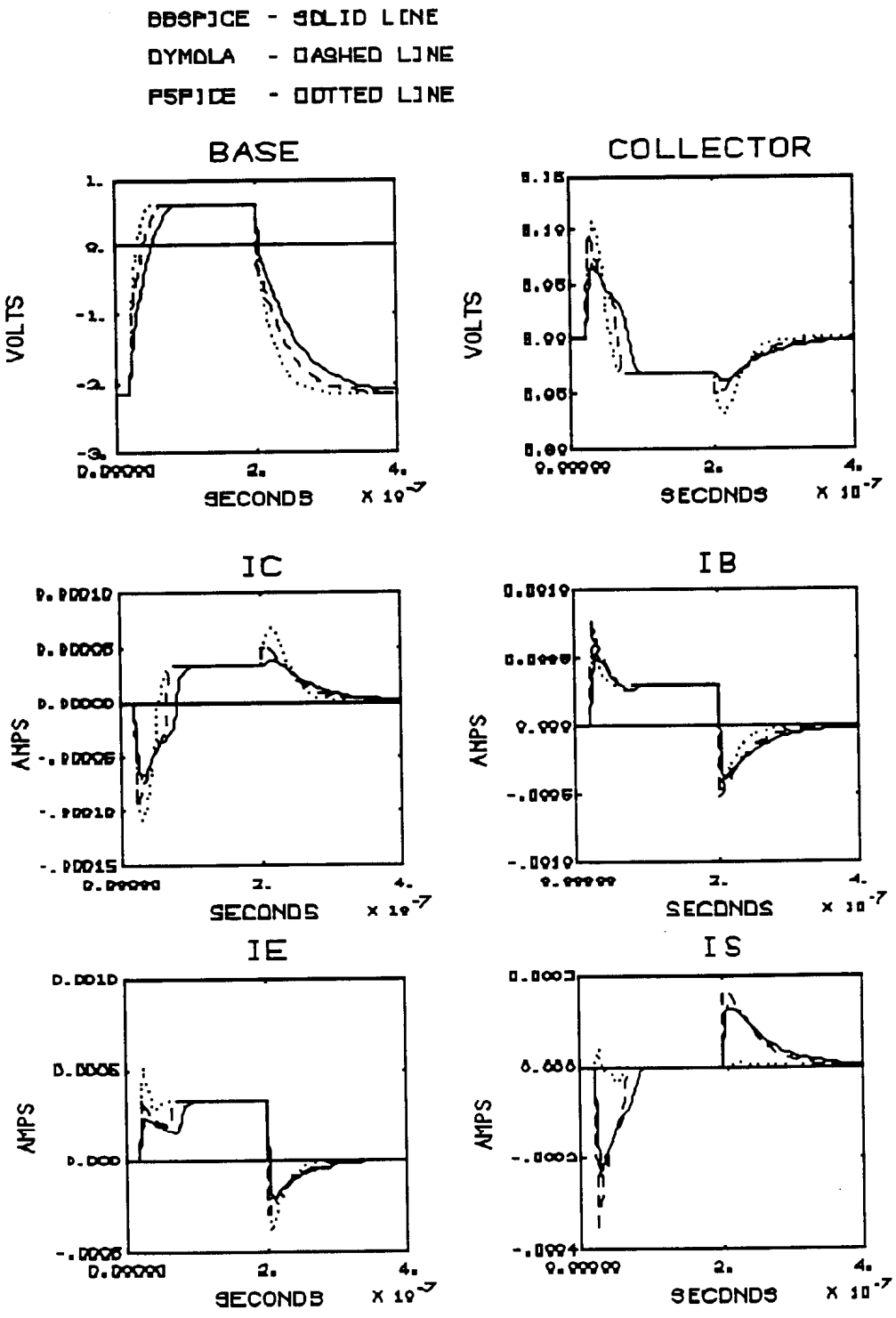
NPN VAF=1.0


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
COLLECTOR
VOLTS
SECONDS
IC
AMPS
SECONDS
IB
AMPS
SECONDS
IE
AMPS
SECONDS
IS
AMPS
SECONDS

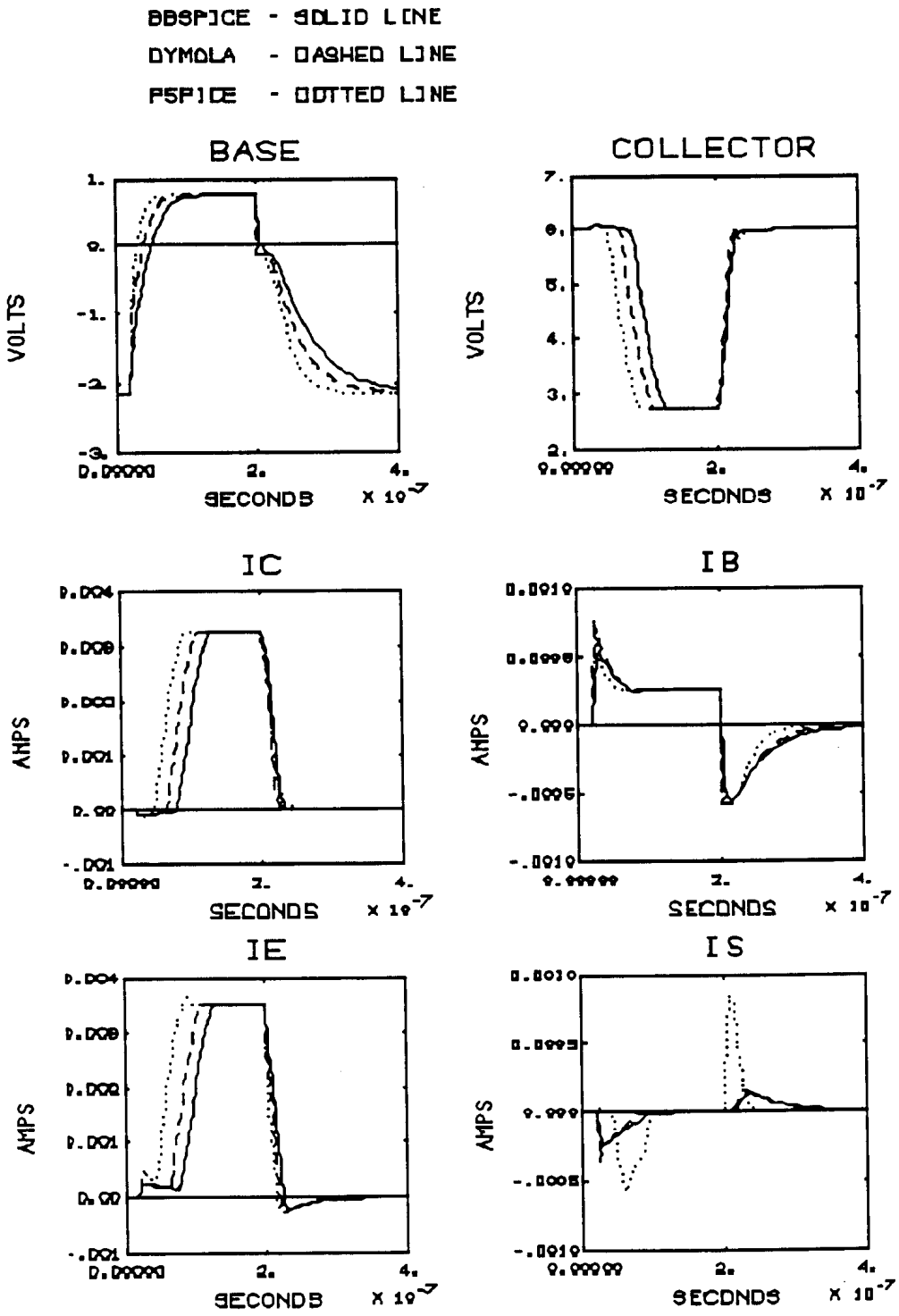
NPN VAR=0.7


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
X 10^-7
COLLECTOR
VOLTS
SECONDS
X 10^-7
IC
AMPS
SECONDS
X 10^-7
IB
AMPS
SECONDS
X 10^-7
IE
AMPS
SECONDS
X 10^-7
IS
AMPS
SECONDS
X 10^-7

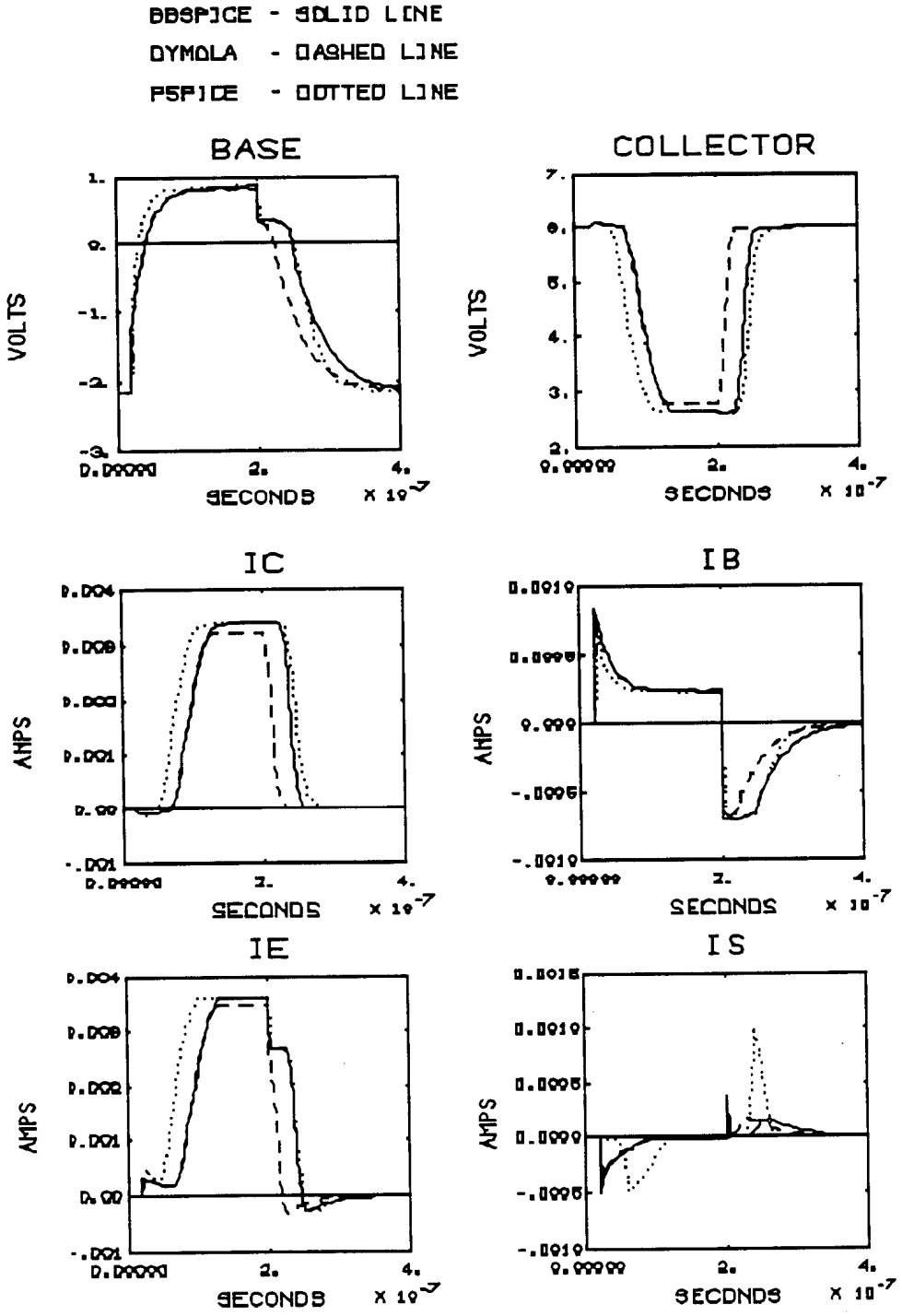
NPN ISE=1.0E-9


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
COLLECTOR
VOLTS
SECONDS
IC
AMPS
SECONDS
IB
AMPS
SECONDS
IE
AMPS
SECONDS
IS
AMPS
SECONDS

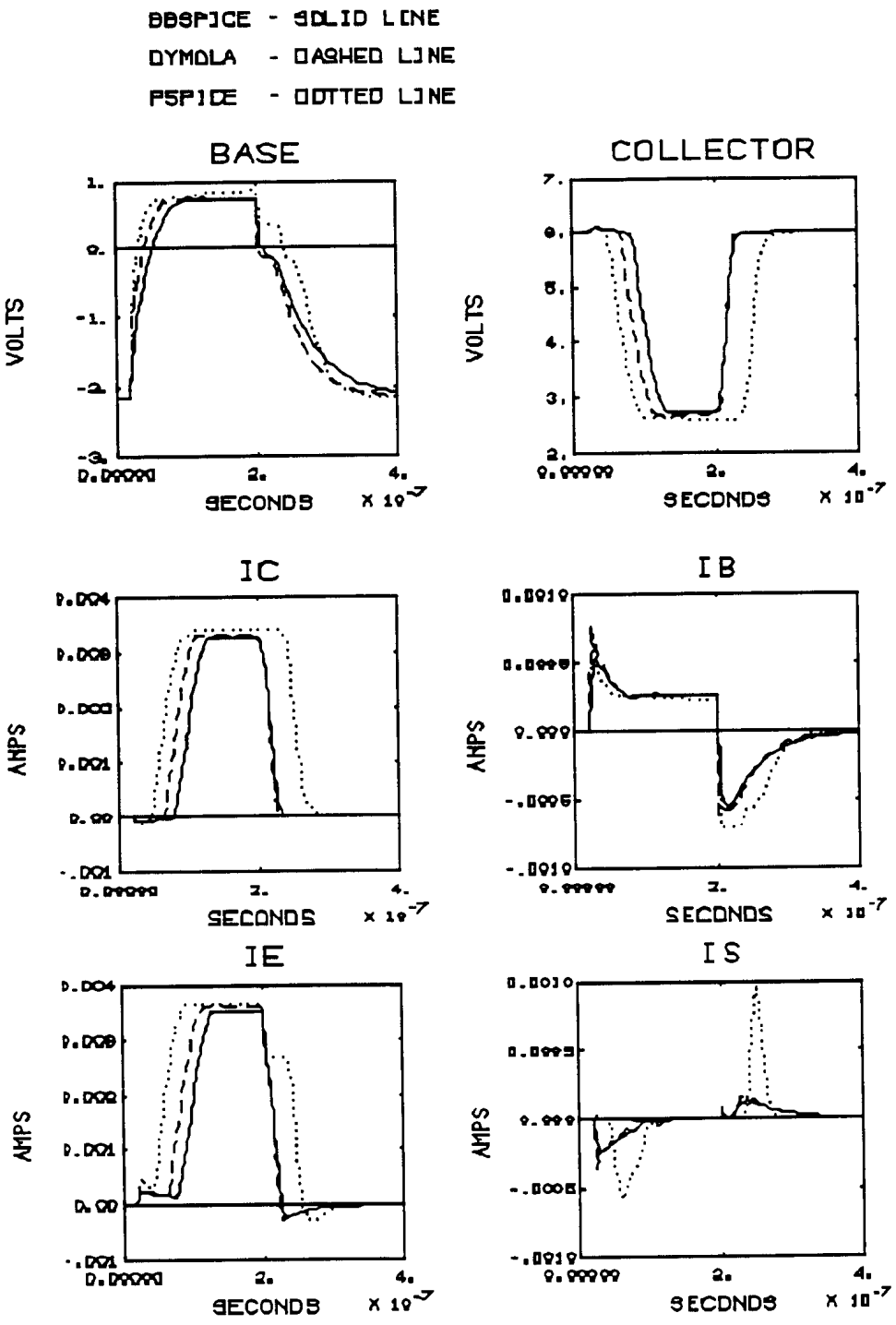
NPN ISC=0.1E-6


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
X 10^-7
COLLECTOR
VOLTS
SECONDS
X 10^-7
IC
AMPS
SECONDS
X 10^-7
IB
AMPS
SECONDS
X 10^-7
IE
AMPS
SECONDS
X 10^-7
IS
AMPS
SECONDS
X 10^-7

NPN IKF=1.0E-6


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
COLLECTOR
VOLTS
SECONDS
IC
AMPS
SECONDS
IB
AMPS
SECONDS
IE
AMPS
SECONDS
IS
AMPS
SECONDS

NPN IKR=1.0E-6


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
COLLECTOR
VOLTS
SECONDS
IC
AMPS
SECONDS
IB
AMPS
SECONDS
IE
AMPS
SECONDS
IS
AMPS
SECONDS

APPENDIX F
SPICE MODELS OF THE PNP CIRCUIT

This appendix contains the BBSPICE and pSpice models of the PNP SIMPLE Circuit. The BBSPICE model is as follows.

```
PNP INVERTER

VI      1   0          PULSE(0V 6V 20nS 0.1nS 0.1nS 180nS 400nS)
VS1     0   3          6V
VS2     5   0          6V

Q1      0   2   4   5  PROC35.P  1
RIN     1   2          5.6k
RBIAS   2   3          10.0k
RLOAD   5   4          4.0k

.MODEL PROC35.P PNP
+ ISC=0.0fA  ISE=0.0fA   BF=1.0E30   BR=1.0E30
+ NC=1.0     NE=1.0      VAF=1.0E30  VAR=1.0E30
+ IKF=1.0E30 IKR=1.0E30
+ NR=1.0   IS=0.11nA  TR=1.0pS  VJC=0.64  CJC=3.6pF   MJC=0.5
+ NF=1.0              TF=1.0pS  VJE=0.77  CJE=5.7pF   MJE=0.5
+ NS=1.0   ISS=0.11nA           VJS=0.75  CJS=11.0pF  MJS=0.5
+ XCJC=1.0
+ RB=1000.0  TRB1=0.005  TRB2=0.0005
+ RBM=60.0   TRM1=0.005  TRM2=0.0005  IRB=0.0
+ RC=450.0   TRC1=0.005  TRC2=0.0005
+ RE=23.3    TRE1=0.005  TRE2=0.0005
+ EG=1.16    XTB=0.0     XTI=3.0     SUBS=+1
+ LEVEL=2    TLEV=0      BULK=5
```

```
.TRAN 5.0nS 400.0nS
.TEMP 25
.OPTIONS TNOM=25 BJTMAX=2 DCAP=1 METHOD=TRAP
.OPTIONS CO=80 NOPAGE CURNTS NUMDGT=4
.PARAM SUB=0
.PRINT TRAN V(2) V(4) I1(Q1) I2(Q1) I3(Q1) I4(Q1)
.PLOT TRAN V(2) V(4) I1(Q1) I2(Q1) I3(Q1) I4(Q1)
.RAMP 10uS 12uS NOOP
.END
```

The pSpice model is as follows.

```
PNP INVERTER

VI      1   0          PULSE(0V 6V 20nS 0.1nS 0.1nS 180nS 400nS)
VS1     0   3          6V
VS2     5   0          6V

Q1      0   2   4   5  PROC35.P  1
RIN     1   2          5.6k
RBIAS   2   3          10.0k
RLOAD   5   4          4.0k

.MODEL PROC35.P PNP
+ ISC=0.0fA  ISE=0.0fA   BF=1.0E30   BR=1.0E30
+ NC=1.0     NE=1.0      VAF=1.0E30  VAR=1.0E30
+ IKF=1.0E30 IKR=1.0E30
+ NR=1.0   IS=0.11nA  TR=1.0pS  VJC=0.64  CJC=3.6pF   MJC=0.5
+ NF=1.0              TF=1.0pS  VJE=0.77  CJE=5.7pF   MJE=0.5
+ NS=1.0   ISS=0.11nA           VJS=0.75  CJS=11.0pF  MJS=0.5
```

```
+ XCJC=1.0
+ RB=1000.0  TRB1=0.005  TRB2=0.0005
+ RBM=60.0   TRM1=0.005  TRM2=0.0005  IRB=0.0
+ RC=450.0   TRC1=0.005  TRC2=0.0005
+ RE=23.3    TRE1=0.005  TRE2=0.0005
+ EG=1.16    XTB=0.0     XTI=3.0

.TRAN 5.0nS 400.0nS
.TEMP 25
.OPTIONS TNOM=25
.PRINT TRAN V(2) V(4) IC(Q1) IB(Q1) IE(Q1) IS(Q1)
.PLOT TRAN V(2) V(4) I1(Q1) I2(Q1) I3(Q1) I4(Q1)
.probe
.END
```

APPENDIX G
DYMOLA FILES MODELING THE PNP CIRCUIT

```
{ File: pnp.txt}
PNP Circuit
This problem contains the following files:

  pnp.txt    : This file of explanations
  elcomp.dym : Dymola library file of electrical components
  bjt.dym    : Dymola library file of BJT transistors
  pnp.dym    : The PNP circuit model description
  pnp.ctl    : Dymola experiment description (ACSL)
  pnp.dcm    : Dymola command file for this problem

During execution of the dymola command file "pnp.dcm" the
following files are generated:

  pnp.sol : A history file of the dymola execution
            to include a listing of the solved equations
  pnp.csl : The dymola generated ACSL program to simulate
            the circuit model
```

```
{ File: pnp.dym

  Circuit with PNPvertical transistor.
  Author:  Daryl Hild
  Date:    May 1993
  Version: 1.2
}

model circuit
  submodel (PNPvertical) Q1(AREA=1.0)
  submodel (BJTpar) Q1par ->
 (ISC=0.0      ISE=0.0      ->
  NC=1.0       NE=1.0       ->
  BF=1.0E+30   BR=1.0E+30   ->
  VAF=1.0E+30  VAR=1.0E+30  ->
  IKR=1.0E+30  IKF=1.0E+30  ->
  NR=1.0 IS=0.11E-9 TR=1E-12 VJC=0.64 CJC=3.6E-12  MJC=0.5 ->
  NF=1.0            TF=1E-12 VJE=0.77 CJE=5.7E-12  MJE=0.5 ->
  NS=1.0 ISS=0.11E-9         VJS=0.75 CJS=11.0E-12 MJS=0.5 ->
  XCJC=1.0                                         ->
  RB=1000.0  TRB1=0.005  TRB2=0.0005               ->
  RBM=60.0   TRM1=0.005  TRM2=0.0005 IRB=0.0       ->
  RC=450.0   TRC1=0.005  TRC2=0.0005               ->
  RE=23.3    TRE1=0.005  TRE2=0.0005               ->
  XTI=3.0 XTB=0.0 GMINDC=1.0E-12 TNOM=25.0 TEMP=25.0)

  submodel (voltage) Ein Es1 Es2
  submodel (resistor) Rin(R=5.6E+3)  Rbias(R=10E+3)  ->
    Rload(R=4E+3)
  submodel common
```

```
  input A Us1 Us2
  output Y IC IB IE IS

  { Set Q1 parameters }
    connect Q1par at Q1:par
  { Connect up the circuit }
    connect common - Ein - Rin -   ->
      ( (Rbias - Es1) // Q1..BC ) - common
    connect common - Es2 - Rload - Q1..EC
    connect Es2 - Q1..SB
  { Set up inputs and outputs }
    Es1.U0 = Us1
    Es2.U0 = Us2
    Ein.U0 = A
    Y = Rload.Vb
    IC = Q1.IC
    IB = Q1.IB
    IE = Q1.IE
    IS = Q1.ISUB
end
```

```
{  File:  pnp.ctl }

cmodel

  maxtime tmax=400.0E-9
  cinterval cint=5.0E-11
  input 3, A(depend,Ain), Us1(depend,Us1in), Us2(depend,Us2in)

  INITIAL
    algorithm ialg=5
    nsteps nstp=1000
    CONSTANT thigh=20.0E-9, tlow=200.0E-9
    Ain=0.0
    Us1in=6.0
    Us2in=6.0
    schedule high .at. thigh
    schedule low .at. tlow
  END

  DISCRETE high
    Ain=6.0
  END
  DISCRETE low
    Ain=0.0
  END
end
```

```
{ File: pnp.dcm }
set
set LogCommands on
outfile pnp.sol

{ DEFINE THE MODEL }
enter model
@elcomp.dym
@bjt.dym
@pnp.dym

{ PROCESS THE MODEL }
differentiate
variable state Q1::dbc.u
variable state Q1::dbe.u
variable state Q1::dbs.u
variable value Q1::dbc.u=0.39
variable value Q1::dbe.u=0.39
variable value Q1::dbs.u=-5.65
partition
set Statistics on
output solved equations

{ GENERATE AN ACSL MODEL FOR SIMULATION }
set LogCommands off
set ACSLold on
language acsl
enter experiment
@pnp.ctl
outfile pnp.csl
output program
outfile
```

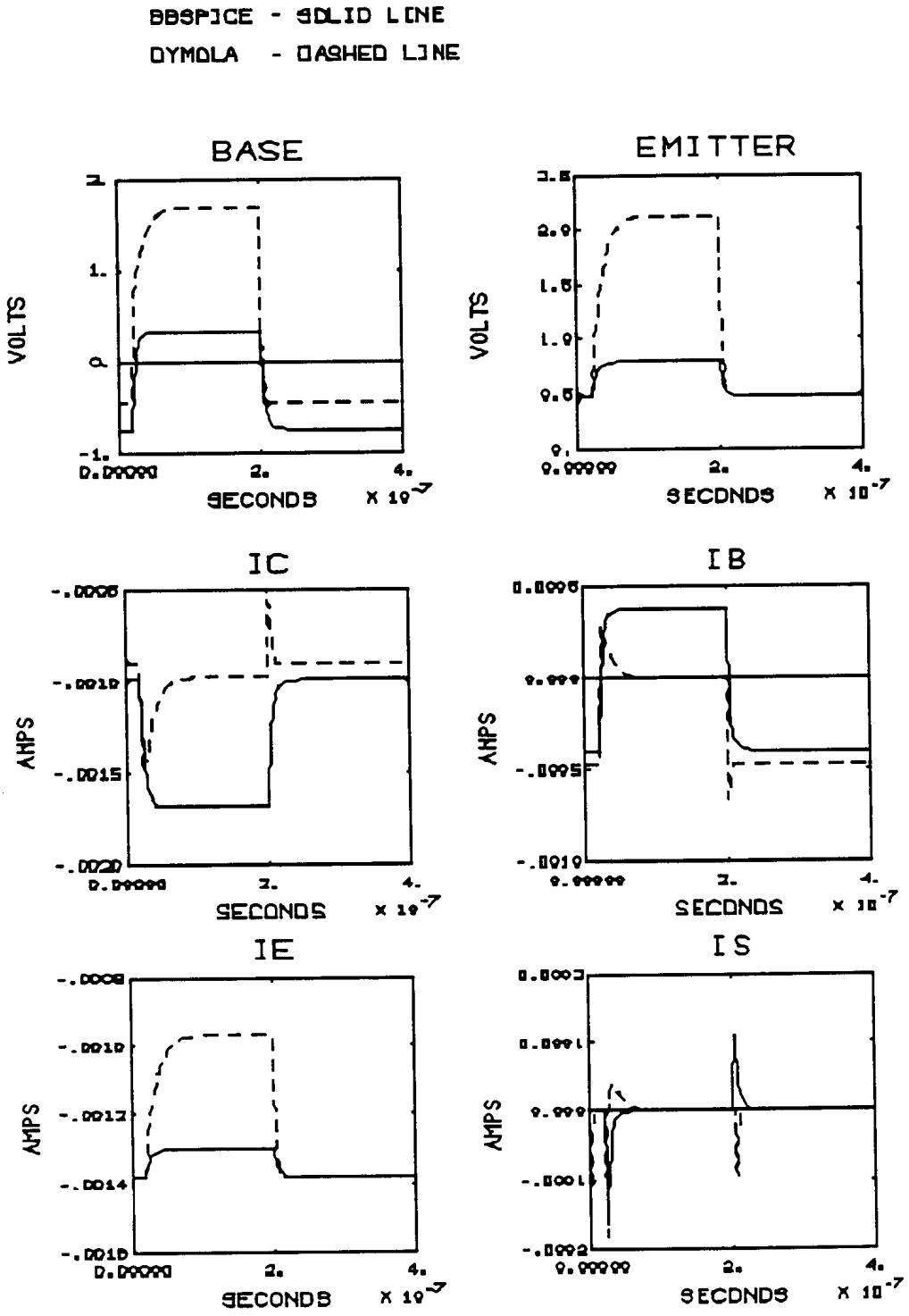
APPENDIX H
SIMULATION PLOTS OF THE PNP CIRCUIT

This appendix contains the simulation plots of the PNP Circuit. The first set of plots, PNP SIMPLE, are based on the parameters given in the BBSpice and pSpice models found in Appendix F and the Dymola model of Appendix G. For convenience, these parameter settings – for the PNP SIMPLE plot – are repeated below. The remaining plots use the same parameters with deviations being noted in the plot title. Note that in some of these plots only the Dymola and BBSPICE trajectories are depicted. In these cases, pSPice was unable to process and simulate the model for that specific parameter set. In particular, one can note that pSpice was unable to simulate the model when the two dependent current sources `IC0` and `IB0` were effectively turned off.

The basic parameter settings are as follows.

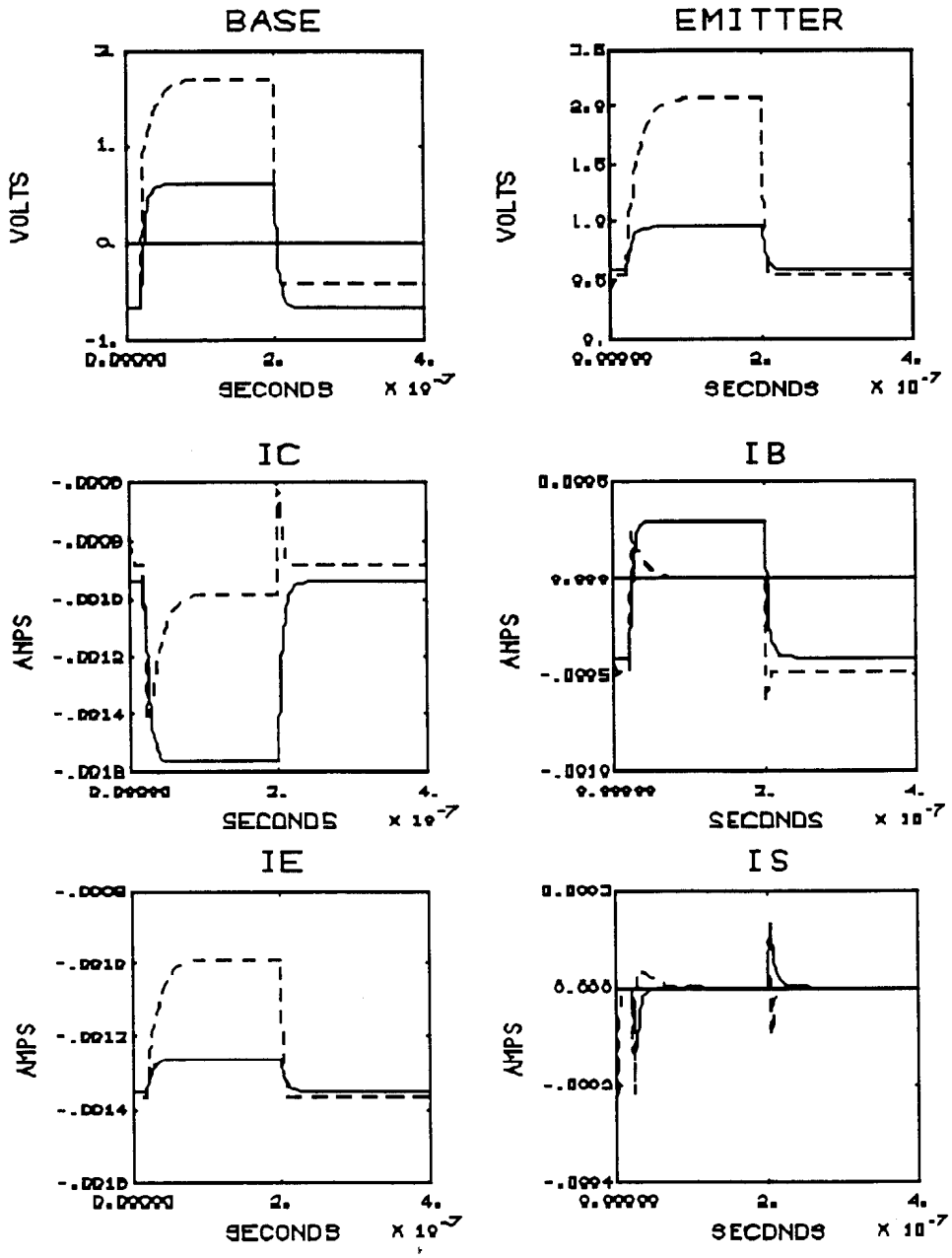
```
PNP
AREA=1.0
ISC=0.0       ISE=0.0
NC=1.0        NE=1.0
BF=1.0E+30    BR=1.0E+30
VAF=1.0E+30   VAR=1.0E+30
IKR=1.0E+30   IKF=1.0E+30
NR=1.0    IS=0.11E-9 TR=1E-12 VJC=0.64 CJC=3.6E-12  MJC=0.5
NF=1.0               TF=1E-12 VJE=0.77 CJE=5.7E-12  MJE=0.5
NS-1.0    ISS=0.11E-9         VJS=0.75 CJS=11.0E-12 MJS=0.5
XCJC=1.0
RB=1000.0  TRB1=0.005 TRB2=0.0005
RBM=60.0   TRM1=0.005 TRM2=0.0005 IRB=0.0
RC=450.0   TRC1=0.005 TRC2=0.0005
RE=23.3    TRE1=0.005 TRE2=0.0005
XTI=3.0 XTB=0.0 GMINDC=1.0E-12 TNOM=25.0 TEMP=25.0
```

PNP SIMPLE

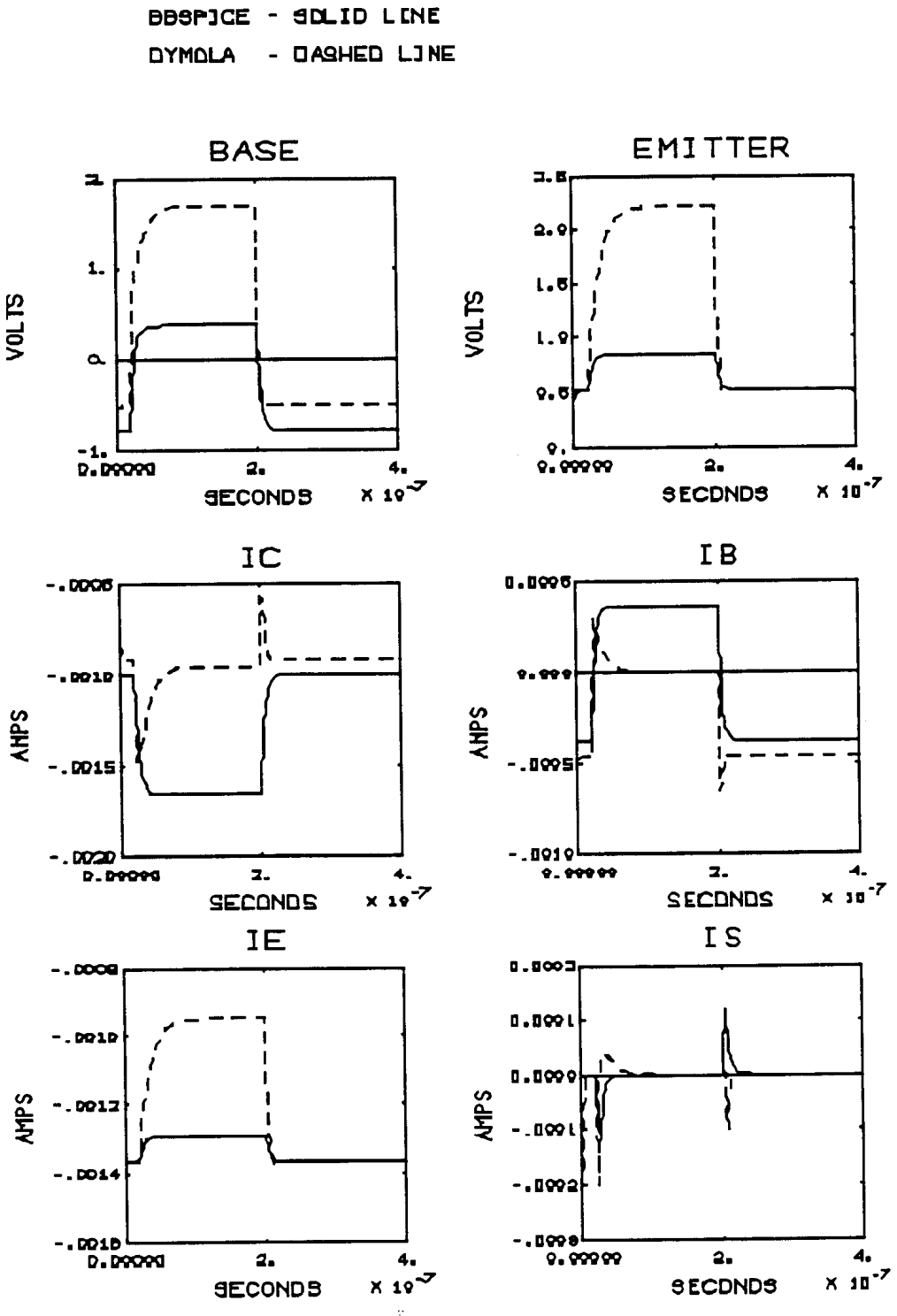

BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
BASE
VOLTS
SECONDS
X 10^-7
EMITTER
VOLTS
SECONDS
X 10^-7
IC
AMPS
SECONDS
X 10^-7
IB
AMPS
SECONDS
X 10^-7
IE
AMPS
SECONDS
X 10^-7
IS
AMPS
SECONDS
X 10^-7

PNP TNOM=10.0

BBSPICE - SOLID LINE
DYMOLA - DASHED LINE

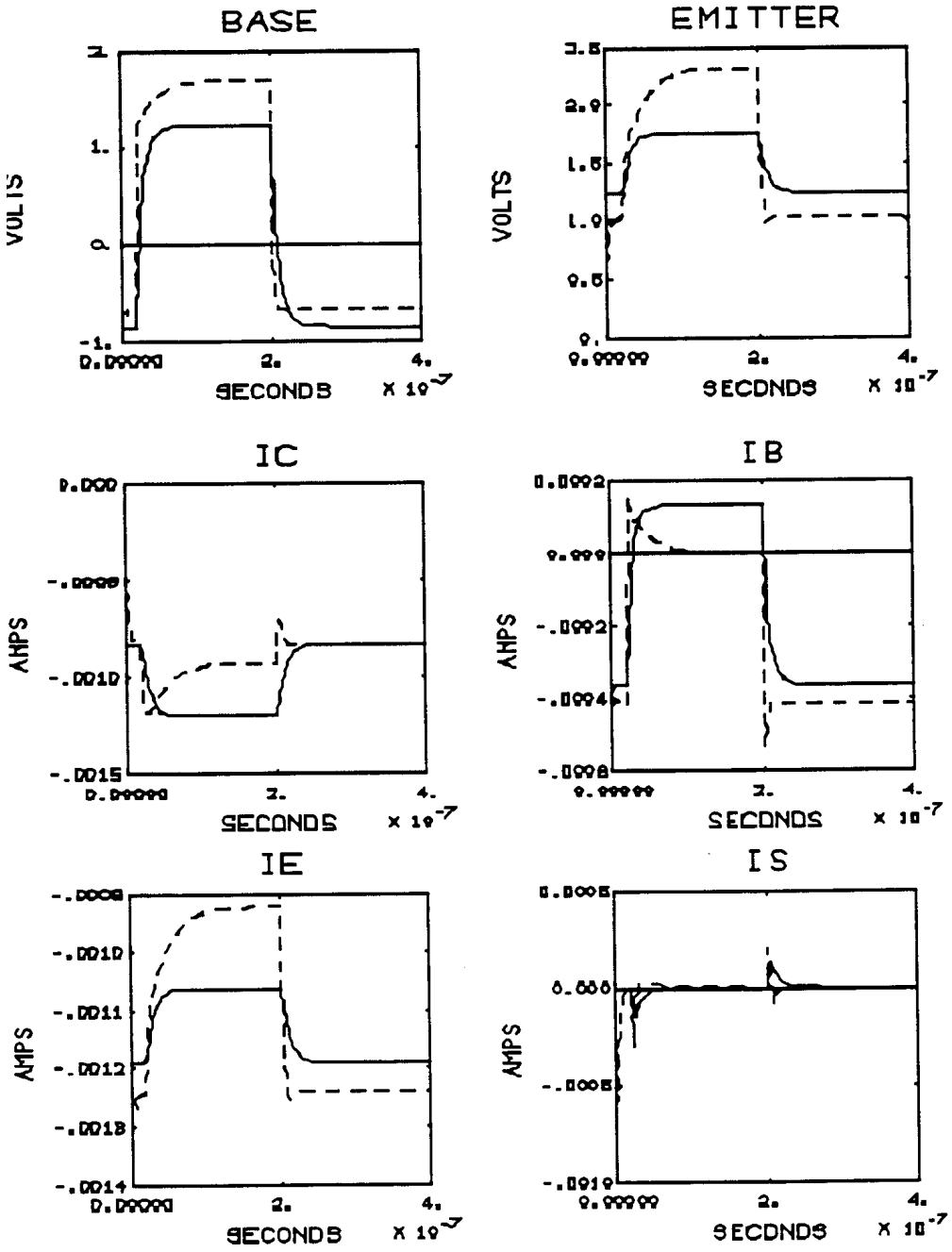


PNP TNOM=45.0


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
BASE
VOLTS
SECONDS
EMITTER
VOLTS
SECONDS
IC
AMPS
SECONDS
IB
AMPS
SECONDS
IE
AMPS
SECONDS
IS
AMPS
SECONDS

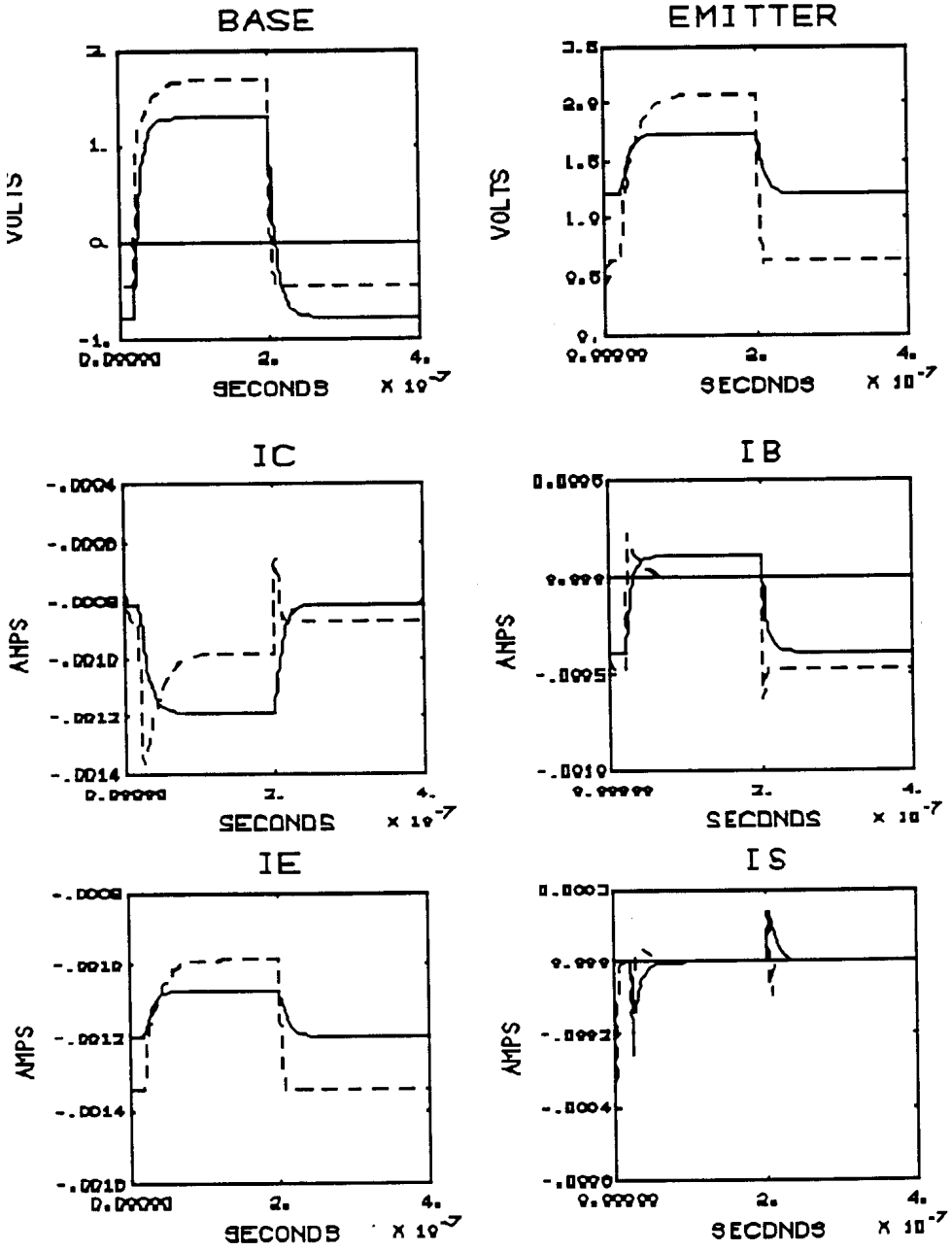
PNP TEMP=-35.0

BBSPICE - SOLID LINE
DYMOLA - DASHED LINE

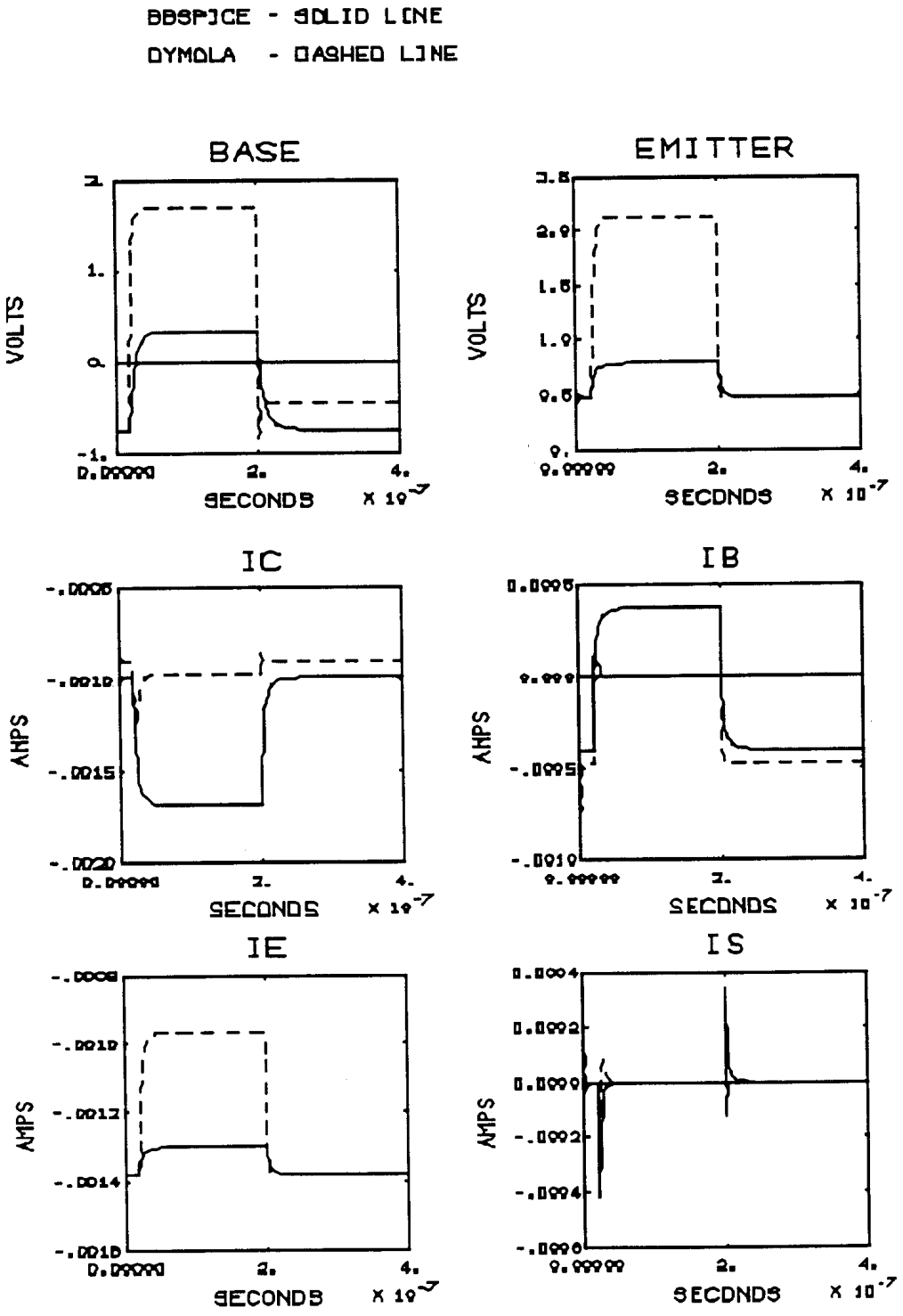


PNP TEMP=50.0

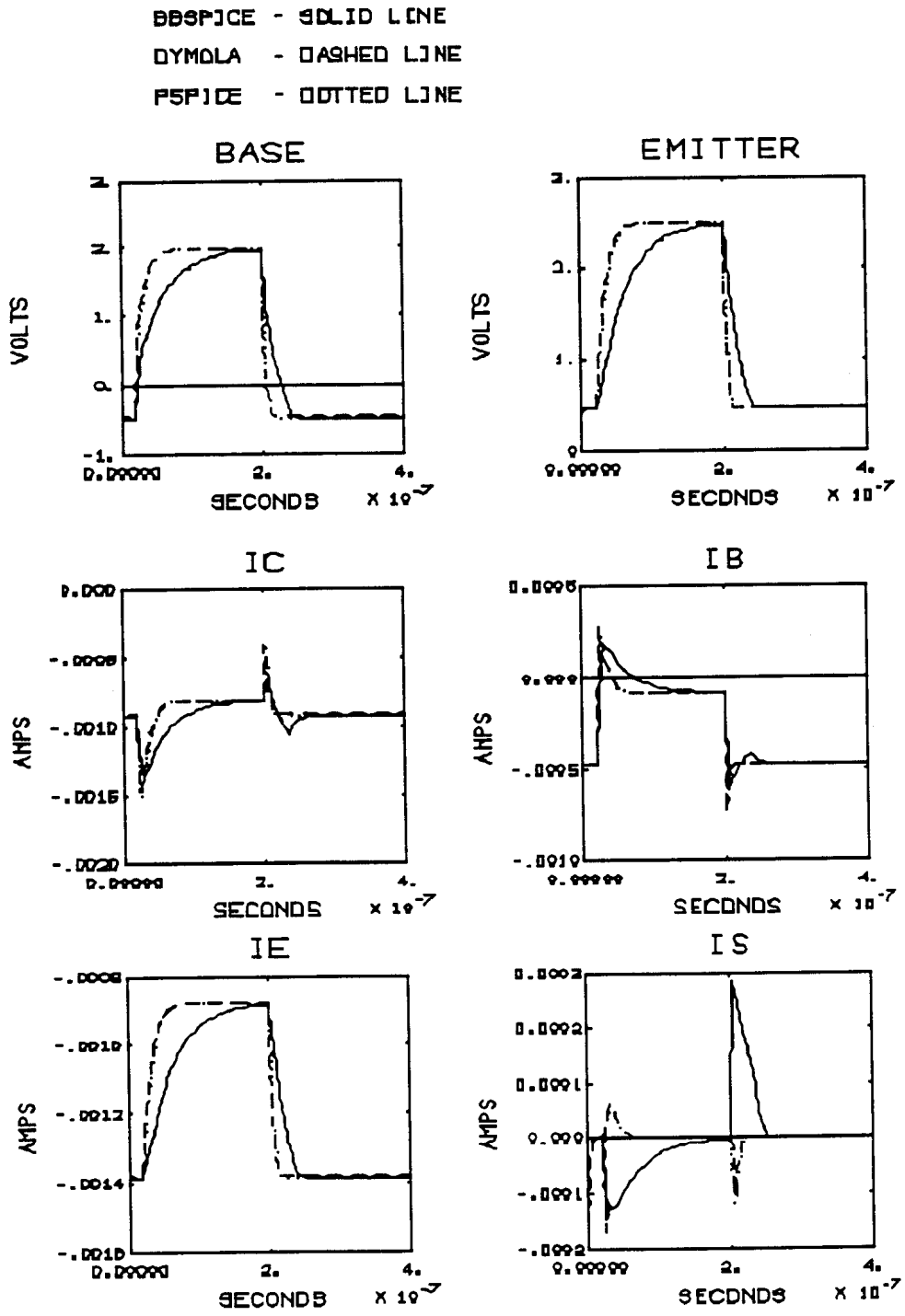
BBSPICE - SOLID LINE
DYMOLA - DASHED LINE



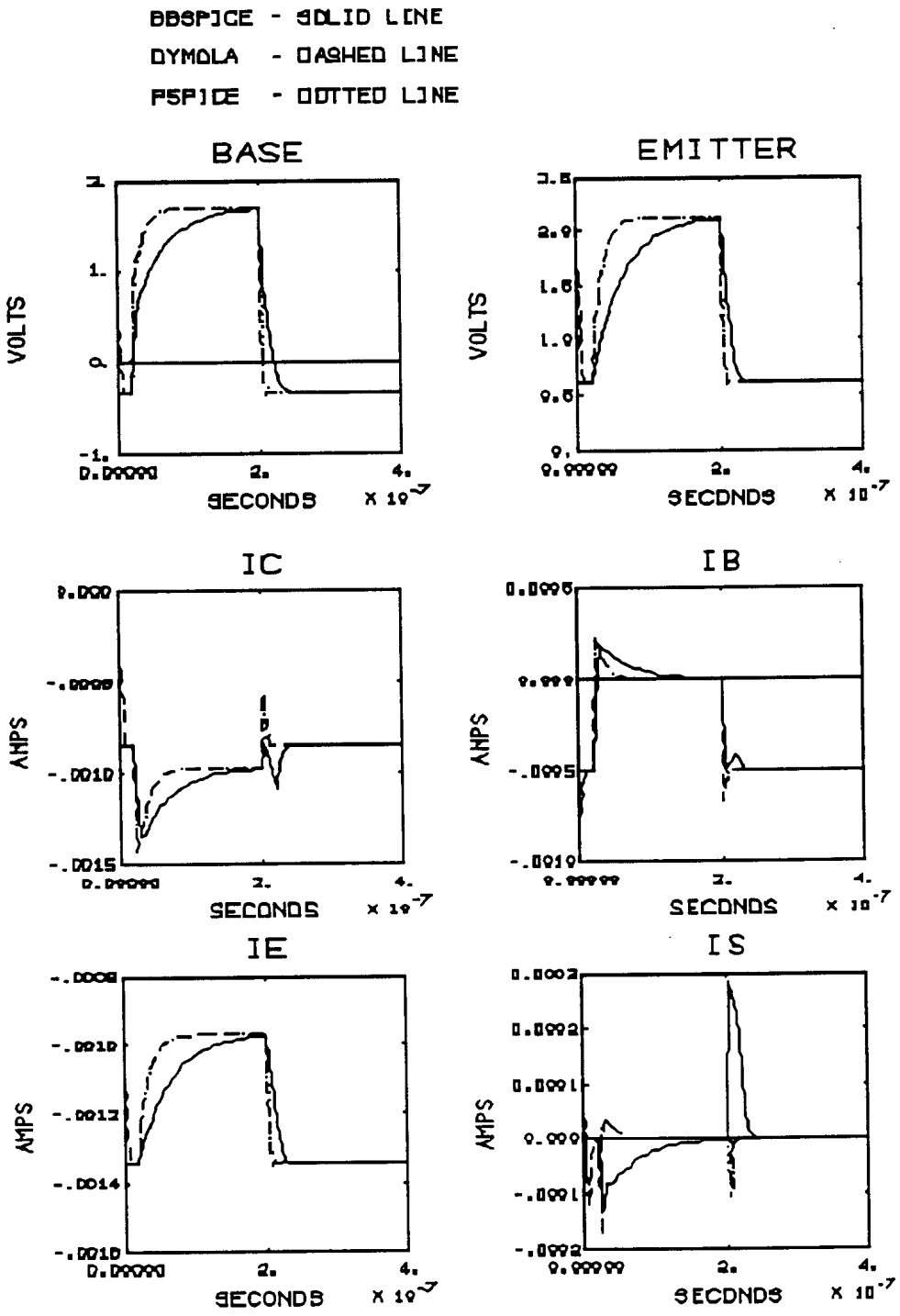
PNP XCJC=0.6


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
BASE
VOLTS
SECONDS
EMITTER
VOLTS
SECONDS
IC
AMPS
SECONDS
IB
AMPS
SECONDS
IE
AMPS
SECONDS
IS
AMPS
SECONDS

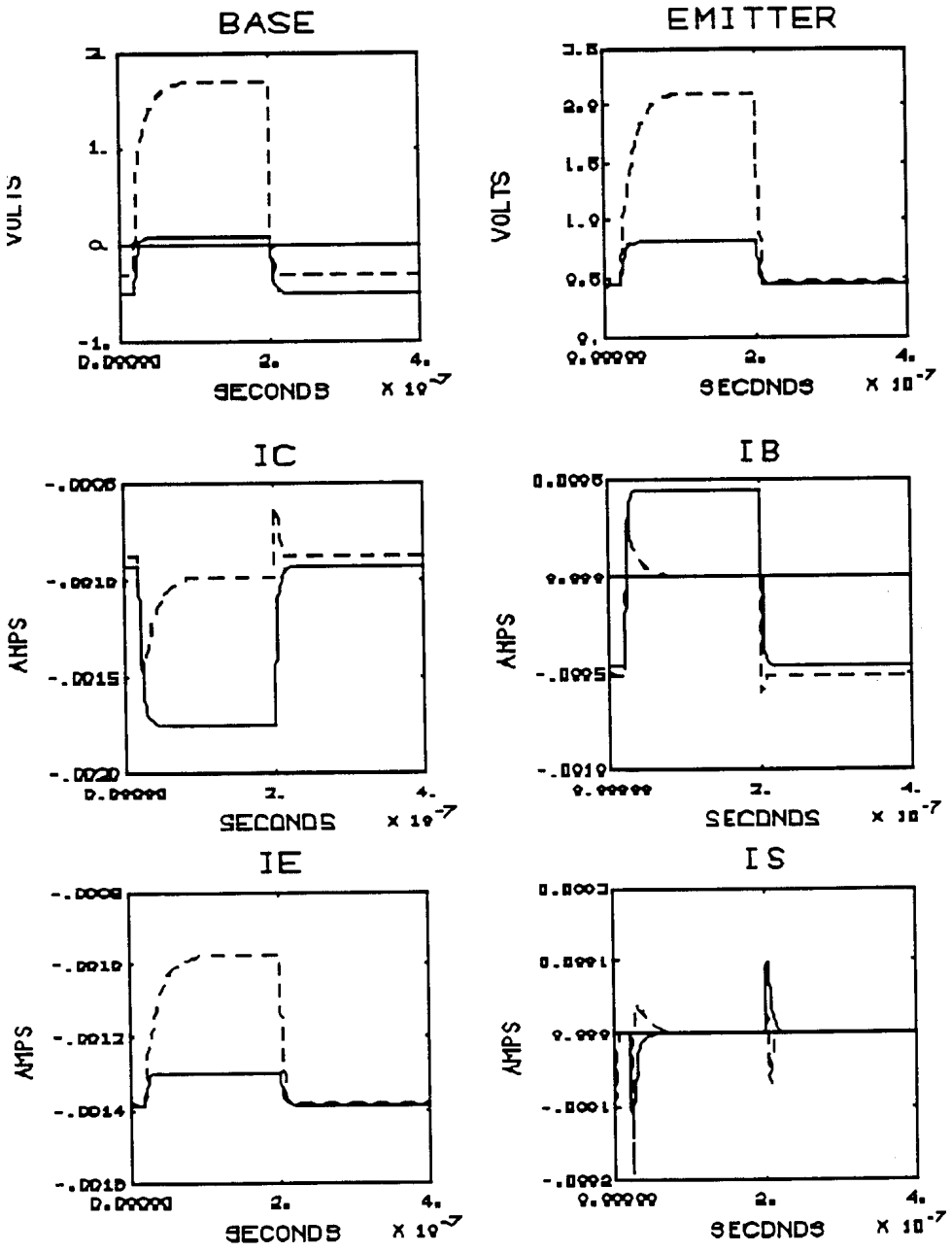
PNP BF=10.0


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
EMITTER
VOLTS
SECONDS
IC
AMPS
SECONDS
IB
AMPS
SECONDS
IE
AMPS
SECONDS
IS
AMPS
SECONDS

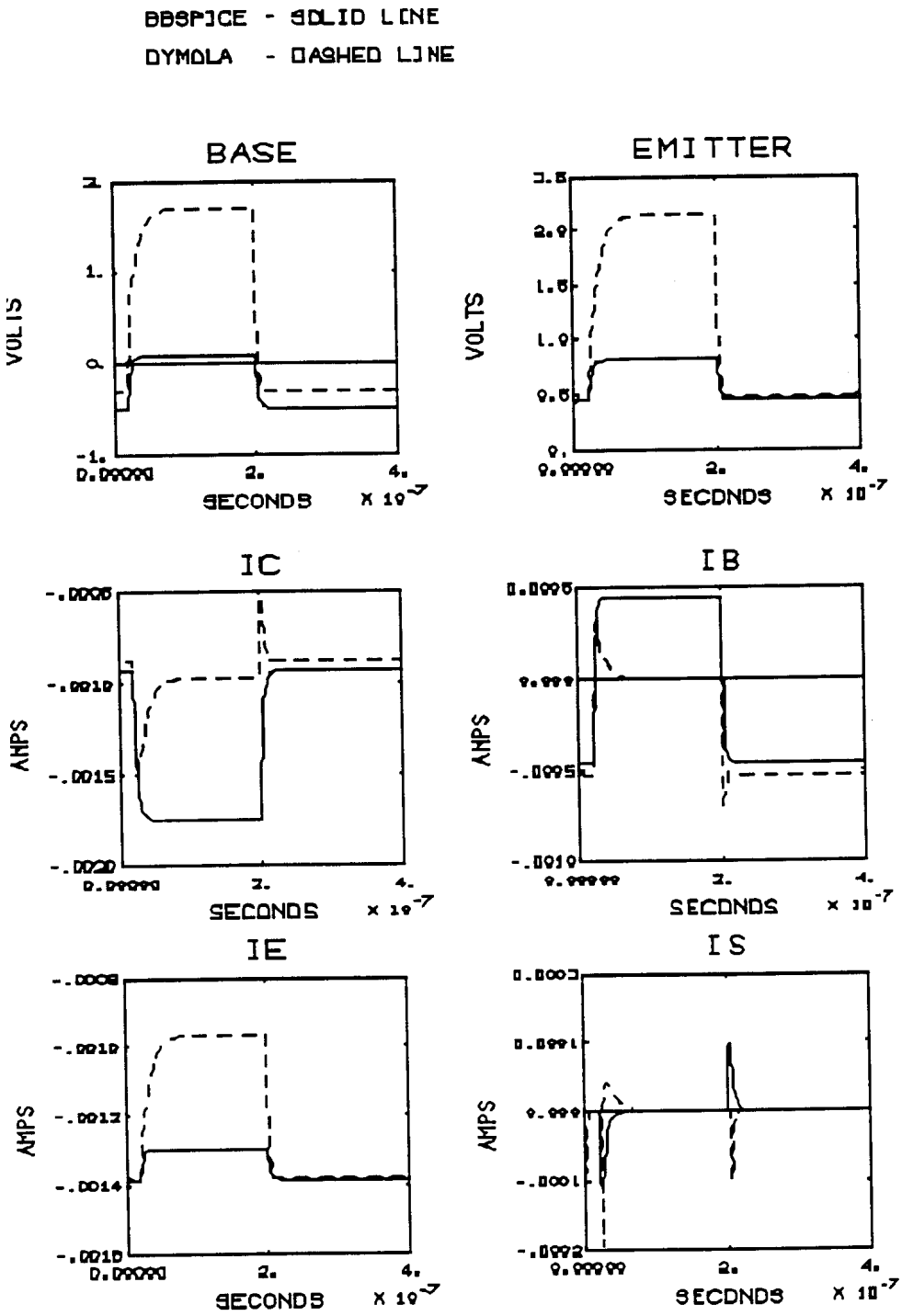
PNP BR=0.001


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
EMITTER
VOLTS
SECONDS
IC
AMPS
SECONDS
IB
AMPS
SECONDS
IE
AMPS
SECONDS
IS
AMPS
SECONDS

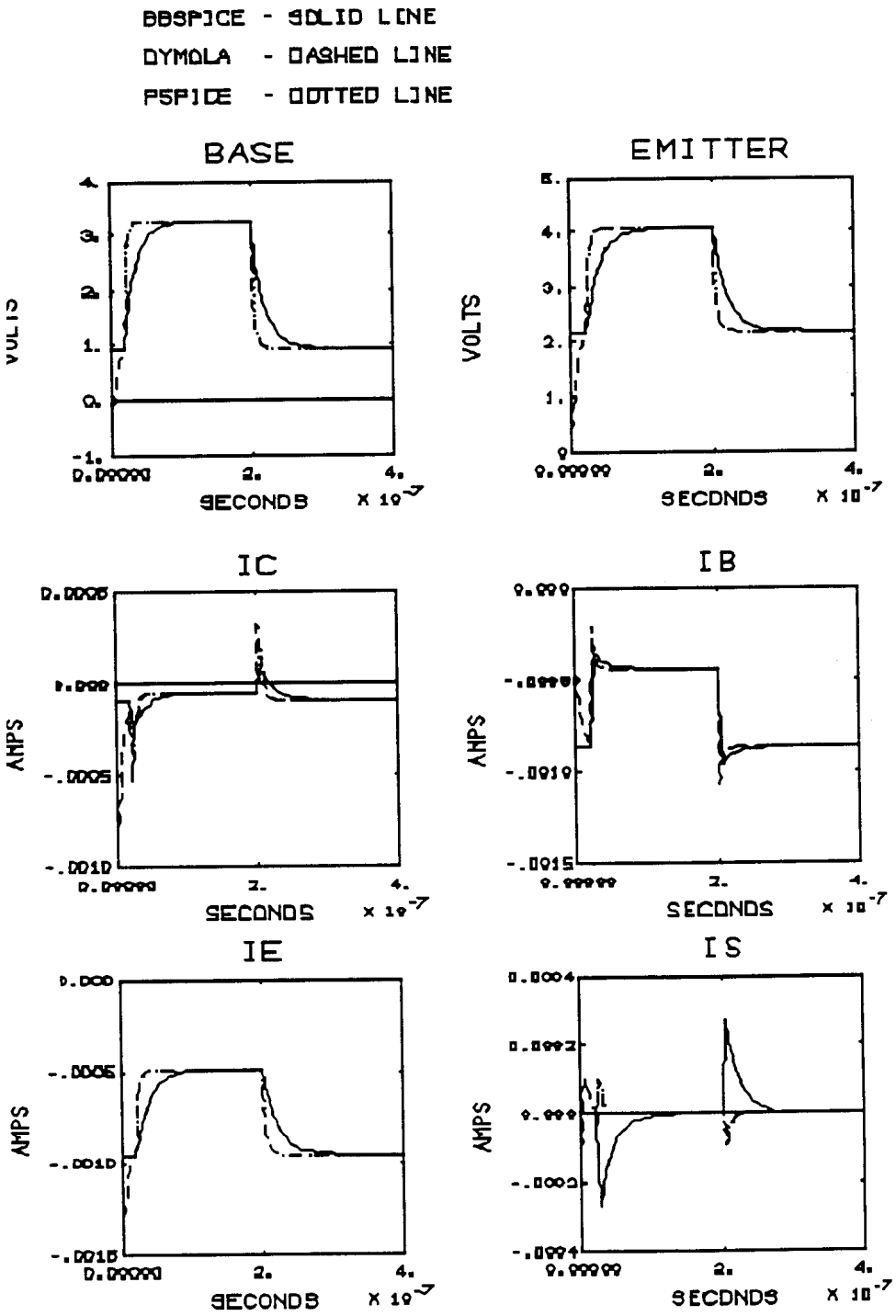
PNP VAF=1.0


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
BASE
VOLTS
SECONDS
X 10^-7
EMITTER
VOLTS
SECONDS
X 10^-7
IC
AMPS
SECONDS
X 10^-7
IB
AMPS
SECONDS
X 10^-7
IE
AMPS
SECONDS
X 10^-7
IS
AMPS
SECONDS
X 10^-7

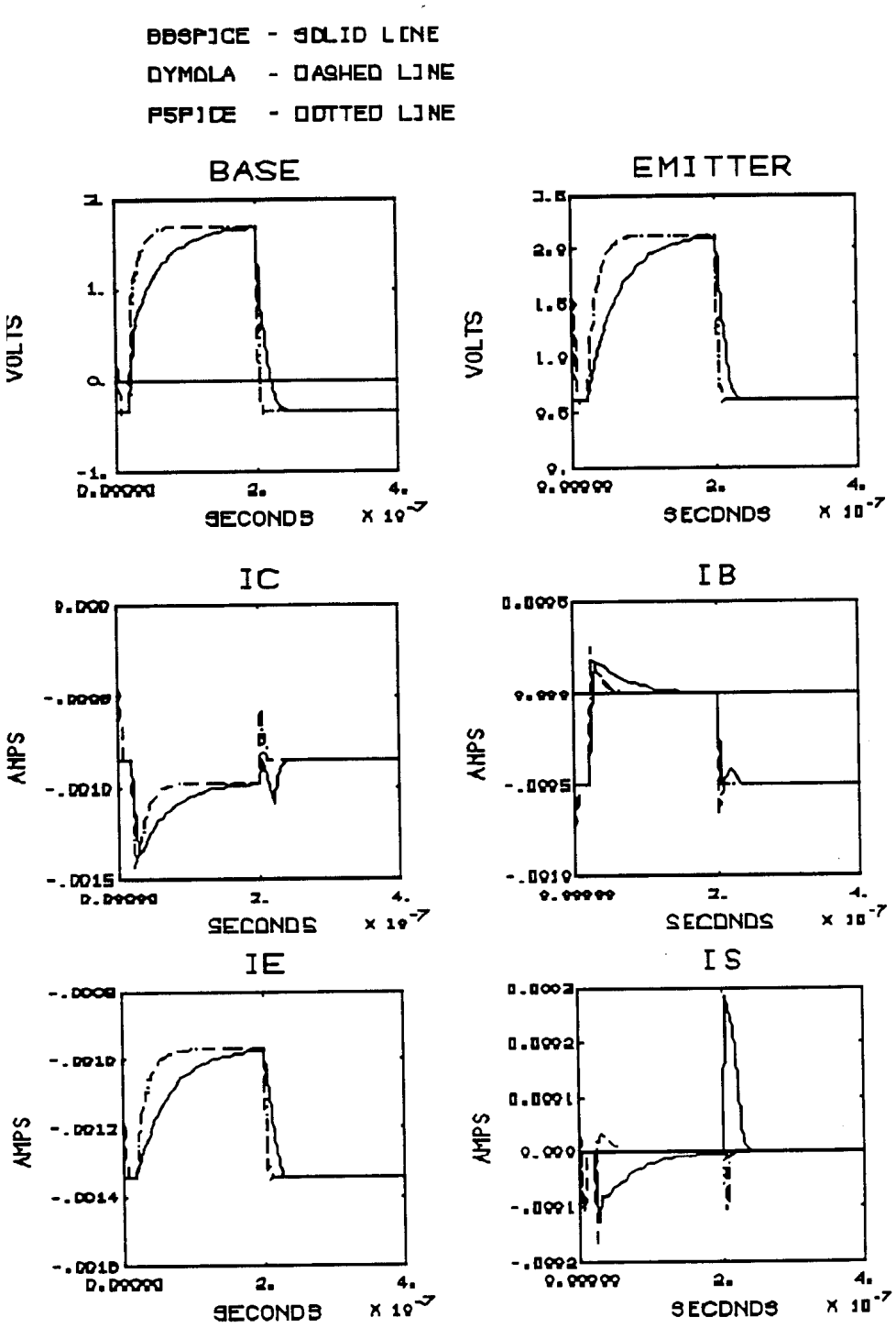
PNP VAR=1.0


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
BASE
VOLTS
SECONDS
EMITTER
VOLTS
SECONDS
IC
AMPS
SECONDS
IB
AMPS
SECONDS
IE
AMPS
SECONDS
IS
AMPS
SECONDS

PNP ISE=1.0E-9

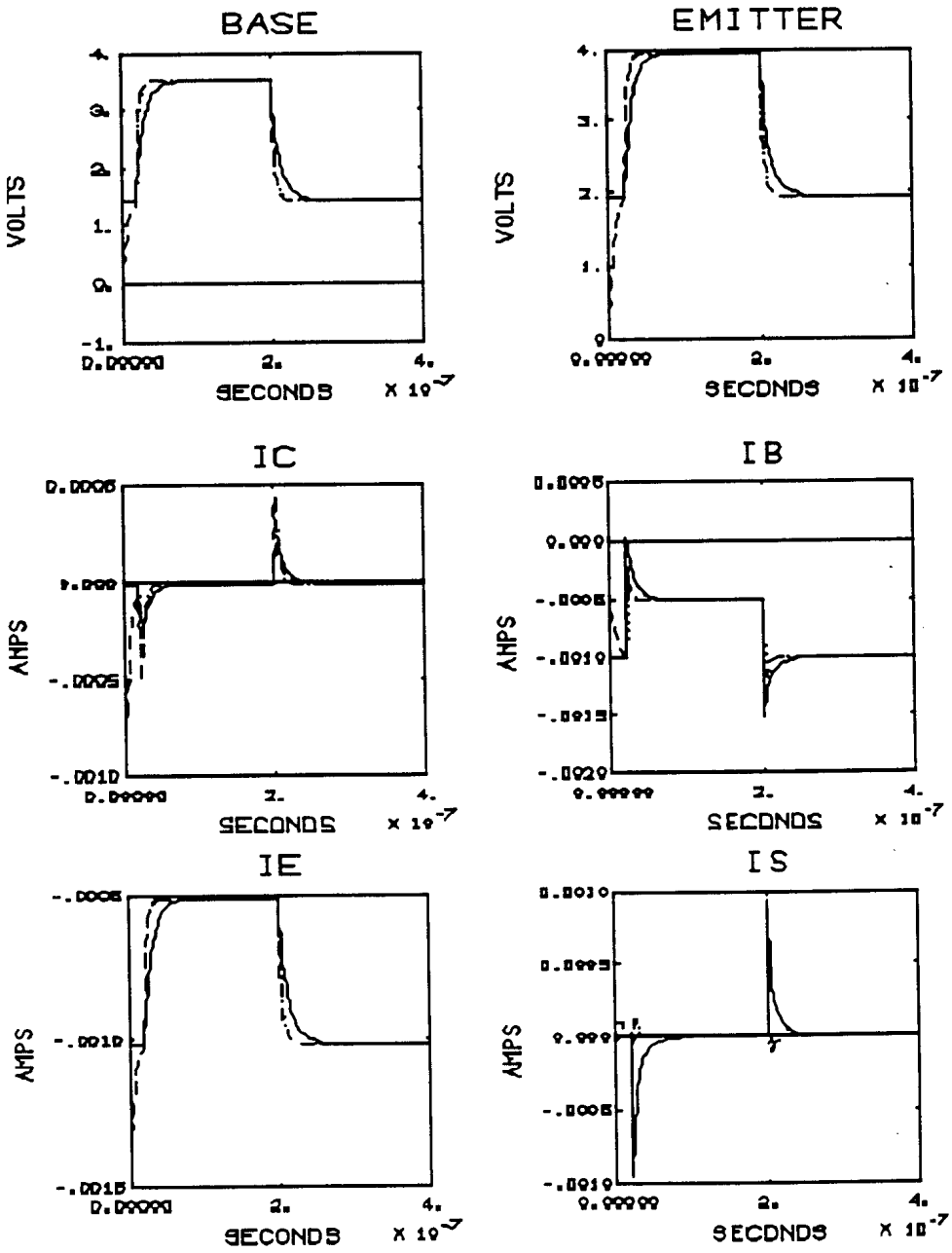

BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
X 10^-7
EMITTER
VOLTS
SECONDS
X 10^-7
IC
AMPS
SECONDS
X 10^-7
IB
AMPS
SECONDS
X 10^-7
IE
AMPS
SECONDS
X 10^-7
IS
AMPS
SECONDS
X 10^-7

PNP ISC=0.1E-6


BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE
BASE
VOLTS
SECONDS
X 10^-7
EMITTER
VOLTS
SECONDS
X 10^-7
IC
AMPS
SECONDS
X 10^-7
IB
AMPS
SECONDS
X 10^-7
IE
AMPS
SECONDS
X 10^-7
IS
AMPS
SECONDS
X 10^-7

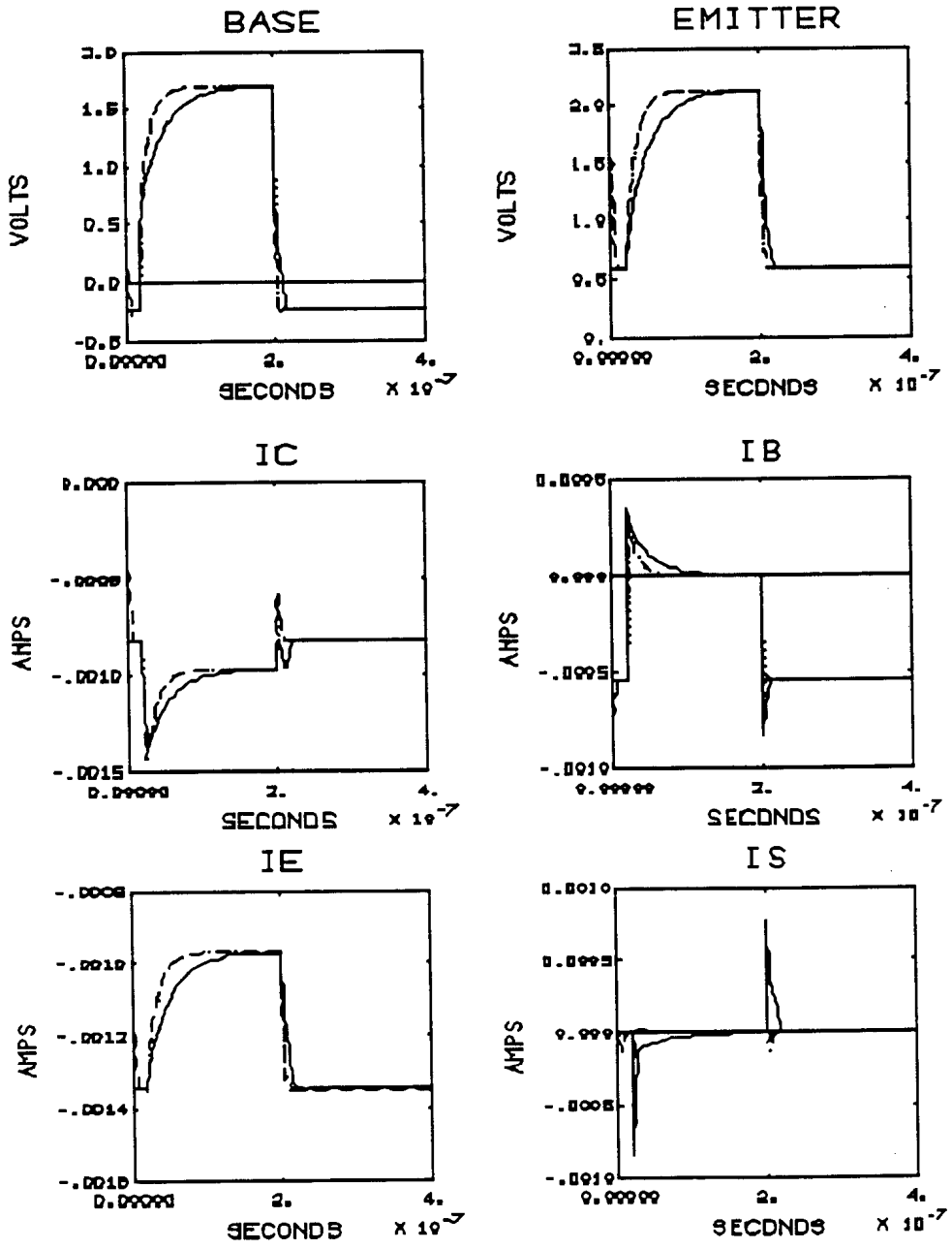
PNP IKF=1.0E-6 ISE=1.0E-9

BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE



PNP IKR=1.0E-6 ISC=0.1E-6

BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE



APPENDIX I
SPICE MODELS OF THE
OPAMP CIRCUIT

This appendix contains the BBSPICE and pSpice models of the OPAMP Circuit. The BBSPICE model is as follows.

```
OPAMP
VIP    11   0        0V
VIM    10   0        PULSE(+2V -2V 20nS 0.1nS 0.1nS 250nS 400nS)
VCC     7   0        5V
VEE     0   3        5V

Q12     1   2   3   3   PROC35.N   1
Q5      4   4   5   3   PROC35.N   1
Q3      4   6   7   7   PROC35.P   4
Q1      6   6   7   7   PROC35.P   2
Q2      8   6   7   7   PROC35.P   1
Q4      7   4   9   3   PROC35.N   1
Q6      2  10   8   7   PROC35.P   1
Q7      1  11   8   7   PROC35.P   1
Q8      5   5  12   3   PROC35.N   1
Q9      3  12   9   7   PROC35.P   1
Q10    12   1   3   3   PROC35.N   2
Q11     2   2   3   3   PROC35.N   1
C1      1  12            5p
R1      3   6            350
C2      6   7            1p
C3      3   1            1p
C4      4   7            1p
Rf     10   9            150000
```

```
.MODEL PROC35.N NPN
+ ISC=0.1uA  ISE=0.0fA   BF=100.0   BR=1.0
+ NC=2.0     NE=1.5      VAF=1.0E30  VAR=1.0E30
+ IKF=1.0E30 IKR=1.0E30
+ NR=1.0  IS=0.11nA  TR=1.0pS  VJC=0.75  CJC=3.6pF  MJC=0.33
+ NF=1.0             TF=1.0pS  VJE=0.75  CJE=5.7pF  MJE=0.33
+ NS=1.0  ISS=0.11nA           VJS=0.75  CJS=11.0pF MJS=0.33
+ XCJC=1.0
+ RB=200.0   TRB1=0.00  TRB2=0.000
+ RBM=100.0  TRM1=0.00  TRM2=0.000  IRB=0.0
+ RC=750.0   TRC1=0.00  TRC2=0.000
+ RE=123.3   TRE1=0.00  TRE2=0.000
+ EG=1.16    XTB=0.0    XTI=3.0
+ SUBS=+1    LEVEL=2    TLEV=0

.MODEL PROC35.P PNP
+ ISC=0.0fA  ISE=1.0nA   BF=100.0   BR=1.0
+ NC=2.0     NE=1.5      VAF=1.0E30  VAR=1.0E30
+ IKF=1.0E30 IKR=1.0E30
+ NR=1.0  IS=0.11nA  TR=1.0pS  VJC=0.75  CJC=3.6pF  MJC=0.33
+ NF=1.0             TF=1.0pS  VJE=0.75  CJE=5.7pF  MJE=0.33
+ NS=1.0  ISS=0.11nA           VJS=0.75  CJS=11.0pF MJS=0.33
+ XCJC=1.0
+ RB=200.0   TRB1=0.00  TRB2=0.000
+ RBM=100.0  TRM1=0.00  TRM2=0.000  IRB=0.0
+ RC=750.0   TRC1=0.00  TRC2=0.000
+ RE=123.3   TRE1=0.00  TRE2=0.000
+ EG=1.16    XTB=0.0    XTI=3.0
+ SUBS=+1    LEVEL=2    TLEV=0
```

```
.TRAN 5nS 400nS
.TEMP 25
.OPTIONS TNOM=25 BJTMAX=2 DCAP=1 METHOD=TRAP
.OPTIONS CO=80 NOPAGE CURNTS NUMDGT=4 ITL5=50000
.PARAM SUB=0
.PRINT TRAN V(1), V(2), V(4), V(5), V(6), V(8), V(9), V(12)
.PLOT  TRAN V(1), V(2), V(4), V(5), V(6), V(8), V(9), V(12)
.RAMP 20uS 5uS NOOP
.END
```

The pSpice model is as follows.

```
OPAMP
VIP   11   0          0V
VIM   10   0          PULSE(+2V -2V 20nS 0.1nS 0.1nS 250nS 400nS)
VCC    7   0          5V
VEE    0   3          5V

Q12    1   2   3   3  PROC35.N  1
Q5     4   4   5   3  PROC35.N  1
Q3     4   6   7   7  PROC35.P  4
Q1     6   6   7   7  PROC35.P  2
Q2     8   6   7   7  PROC35.P  1
Q4     7   4   9   3  PROC35.N  1
Q6     2  10   8   7  PROC35.P  1
Q7     1  11   8   7  PROC35.P  1
Q8     5   5  12   3  PROC35.N  1
Q9     3  12   9   7  PROC35.P  1
Q10   12   1   3   3  PROC35.N  2
Q11    2   2   3   3  PROC35.N  1
C1     1  12          5p
```

```
R1     3   6          350
C2     6   7          1p
C3     3   1          1p
C4     4   7          1p
Rf    10   9          150000

.MODEL PROC35.N NPN
+ ISC=0.1uA  ISE=0.0fA   BF=100.0   BR=0.1
+ NC=2.0     NE=1.5      VAF=1.0E30  VAR=1.0E30
+ IKF=1.0E30 IKR=1.0E30
+ NR=1.0   IS=0.11nA  TR=1.0pS  VJC=0.75  CJC=3.6pF  MJC=0.33
+ NF=1.0              TF=1.0pS  VJE=0.75  CJE=5.7pF  MJE=0.33
+ NS=1.0   ISS=0.11nA           VJS=0.75  CJS=11.0pF MJS=0.33
+ XCJC=1.0
+ RB=200.0   TRB1=0.00  TRB2=0.000
+ RBM=100.0  TRM1=0.00  TRM2=0.000  IRB=0.0
+ RC=750.0   TRC1=0.00  TRC2=0.000
+ RE=123.3   TRE1=0.00  TRE2=0.000
+ EG=1.16    XTB=0.0    XTI=3.0

.MODEL PROC35.P PNP
+ ISC=0.0fA  ISE=1.0nA   BF=100.0   BR=0.1
+ NC=2.0     NE=1.5      VAF=1.0E30  VAR=1.0E30
+ IKF=1.0E30 IKR=1.0E30
+ NR=1.0   IS=0.11nA  TR=1.0pS  VJC=0.75  CJC=3.6pF  MJC=0.33
+ NF=1.0              TF=1.0pS  VJE=0.75  CJE=5.7pF  MJE=0.33
+ NS=1.0   ISS=0.11nA           VJS=0.75  CJS=11.0pF MJS=0.33
+ XCJC=1.0
+ RB=200.0   TRB1=0.00  TRB2=0.000
+ RBM=100.0  TRM1=0.00  TRM2=0.000  IRB=0.0
+ RC=750.0   TRC1=0.00  TRC2=0.000
+ RE=123.3   TRE1=0.00  TRE2=0.000
+ EG=1.16    XTB=0.0    XTI=3.0
```

```
.TRAN 5nS 400nS
.TEMP 25
.OPTIONS TNOM=25
.PRINT TRAN V(1), V(2), V(4), V(5), V(6), V(8), V(9), V(12)
.PLOT  TRAN V(1), V(2), V(4), V(5), V(6), V(8), V(9), V(12)
.probe
.END
```

APPENDIX J
DYMOLA FILES MODELING THE OPAMP CIRCUIT

```
{ File: opamp.txt}

OpAmp Circuit

This problem contains the following files:

  opamp.txt  : This file of explanations
  elcomp.dym : Dymola library file of electrical components
  bjt.dym    : Dymola library file of BJT transistors
  opamp.dym  : The OpAmp circuit model description
  opamp.ctl  : Dymola experiment description (ACSL)
  opamp.dcm  : Dymola command file for this problem


During execution of the dymola command file "opamp.dcm" the
following files are generated:

  opamp.sol : A history file of the dymola execution
              to include a listing of the solved equations
  opamp.csl : The dymola generated ACSL program to simulate
              the OpAmp circuit model
```

```
{ File:  OpAmp.dym

  Operational Amplifier
  Author:    Daryl R. Hild
  Date:      May 1993 Version:   1.0
  Reference:  Francois Cellier:  "Continuous System Modeling"
              Springer Verlag, 1991  }

model type opamp
  submodel (PNPvertical) Q1(AREA=2.0) Q2 Q3(AREA=4.0) ->
    Q6 Q7 Q9
  submodel (NPNvertical) Q4 Q5 Q8 Q10(AREA=2.0) Q11 Q12
  submodel (BJTpar) ->
    PNPpar(ISE=1.0E-9 IS=0.11E-9 ISS=0.11E-9 TR=1E-12 ->
      TF=1E-12 CJC=3.6E-12 CJE=5.7E-12 CJS=11.0E-12 ->
      RC=750.0 RE=123.3 RB=200.0 RBM=100.0 BR=0.1) ->
    NPNpar(ISC=0.1E-6 IS=0.11E-9 ISS=0.11E-9 TR=1E-12 ->
      TF=1E-12 CJC=3.6E-12 CJE=5.7E-12 CJS=11.0E-12 ->
      RC=750.0 RE=123.3 RB=200.0 RBM=100.0 BR=0.1)
  submodel (resistor) R1(R=350.0)
  submodel (capacitor) C1(C=5.0E-12)  ->
    C2(C=1.0E-12) C3(C=1.0E-12) C4(C=1.0E-12)

  cut VIplus (Vviplus/Iviplus)
  cut VIminus (Vviminus/Iviminus)
  cut Vout (Vvout/-Ivout)
  cut VCC (Vvcc/Ivcc)
  cut VEE (Vvee/Ivee)
  path Power <VCC - VEE>
  path PlusInOut <VIplus - Vout>
  path MinusInOut <VIminus - Vout>
  node n1 n2 n3 n4 n5 n6 n7 n8 n9 n10 n11 n12
```

```
  { Set PNP and NPN parameters }
    connect PNPpar at Q1:par at Q2:par at Q3:par ->
      at Q6:par at Q7:par at Q9:par
    connect NPNpar at Q4:par at Q5:par at Q8:par ->
      at Q10:par at Q11:par at Q12:par
  { Connect OpAmp circuit together }
    connect ->
      Q12 at ( n1,   n2,   n3,   n3) ->
       Q5 at ( n4,   n4,   n5,   n3) ->
       Q3 at ( n4,   n6,   n7,   n7) ->
       Q1 at ( n6,   n6,   n7,   n7) ->
       Q2 at ( n8,   n6,   n7,   n7) ->
       Q4 at ( n7,   n4,   n9,   n3) ->
       Q6 at ( n2,  n10,   n8,   n7) ->
       Q7 at ( n1,  n11,   n8,   n7) ->
       Q8 at ( n5,   n5,  n12,   n3) ->
       Q9 at ( n3,  n12,   n9,   n7) ->
      Q10 at (n12,   n1,   n3,   n3) ->
      Q11 at ( n2,   n2,   n3,   n3) ->
       C1 at ( n1,  n12) ->
       R1 at ( n3,   n6) ->
       C2 at ( n7,   n6) ->
       C3 at ( n1,   n3) ->
       C4 at ( n7,   n4) ->
      VCC at n7  ->
      VEE at n3  ->
  VIminus at n10 ->
   VIplus at n11 ->
     Vout at n9
end
```

```
model OpampCircuit
  submodel opamp
  submodel (voltage) Uin Vcc Vee
  submodel (resistor) Rf(R=150.0E3)
  submodel common

  input A Us1 Us2
  output n1 n2 n4 n5 n6 n8 n9 n12

  connect common - Vcc - opamp..Power - Vee – common
  connect common - opamp..PlusInOut
  connect common - Uin - opamp..MinusInOut - Rf -   ->
    opamp..MinusInOut

  Vcc.U0 = Us1
  Vee.U0 = Us2
  Uin.U0 = A
  n1 = opamp::Q7.VC
  n2 = opamp::Q6.VC
  n4 = opamp::Q3.VC
  n5 = opamp::Q8.VC
  n6 = opamp::Q1.VC
  n8 = opamp::Q2.VC
  n9 = opamp.Vvout
  n12= opamp::Q9.VB
end
```

```
{  File:  opamp.ctl }

cmodel

  maxtime tmax=400.0E-9
  cinterval cint=5.0E-11
  input 3, A(depend,Ain),      ->
           Us1(depend,Us1in),  ->
           Us2(depend, Us2in)

  INITIAL
    algorithm ialg=5
    nsteps nstp=1000
    CONSTANT thigh=270.0E-9, tlow=20.0E-9
    Ain=2.0
    Us1in=5.0
    Us2in=5.0
    schedule high .at. thigh
    schedule low .at. tlow
  END

  DISCRETE high
    Ain=2.0
  END
  DISCRETE low
    Ain=-2.0
  END

end
```

```
{ File: opamp.dcm }
set
set LogCommands on
outfile opamp.sol
{ MODEL DEFINITION }
enter model
@elcomp.dym
@bjt.dym
@opamp.dym
{ PROCESS THE MODEL }
differentiate
variable state opamp::C1.u
variable state opamp::C2.u
variable state opamp::C3.u
variable state opamp::C4.u
variable state opamp::Q1::dbc.u
variable state opamp::Q1::dbe.u
variable state opamp::Q1::dbs.u
variable state opamp::Q2::dbc.u
variable state opamp::Q2::dbe.u
variable state opamp::Q2::dbs.u
variable state opamp::Q3::dbc.u
variable state opamp::Q3::dbe.u
variable state opamp::Q3::dbs.u
variable state opamp::Q4::dbc.u
variable state opamp::Q4::dbe.u
variable state opamp::Q4::dbs.u
variable state opamp::Q5::dbc.u
variable state opamp::Q5::dbe.u
variable state opamp::Q5::dbs.u
variable state opamp::Q6::dbc.u
variable state opamp::Q6::dbe.u
variable state opamp::Q6::dbs.u
variable state opamp::Q7::dbc.u
```

```
variable state opamp::Q7::dbe.u
variable state opamp::Q7::dbs.u
variable state opamp::Q8::dbc.u
variable state opamp::Q8::dbe.u
variable state opamp::Q8::dbs.u
variable state opamp::Q9::dbe.u
variable state opamp::Q9::dbe.u
variable state opamp::Q9::dbs.u
variable state opamp::Q10::dbe.u
variable state opamp::Q10::dbe.u
variable state opamp::Q10::dbs.u
variable state opamp::Q11::dbe.u
variable state opamp::Q11::dbe.u
variable state opamp::Q11::dbs.u
variable state opamp::Q12::dbe.u
variable state opamp::Q12::dbe.u
variable state opamp::Q12::dbs.u
variable value opamp::C1.u=-2.09338
variable value opamp::C2.u=1.70094
variable value opamp::C3.u=1.57716
variable value opamp::C4.u=2.40042
variable value opamp::Q1::dbc.u=0.379591
variable value opamp::Q1::dbe.u=0.452165
variable value opamp::Q1::dbs.u=-0.655184
variable value opamp::Q2::dbc.u=0.368327
variable value opamp::Q2::dbe.u=0.459308
variable value opamp::Q2::dbs.u=-0.882448
variable value opamp::Q3::dbc.u=0.376657
variable value opamp::Q3::dbe.u=0.454543
variable value opamp::Q3::dbs.u=-0.723755
variable value opamp::Q4::dbc.u=0.378742
variable value opamp::Q4::dbe.u=0.465322
variable value opamp::Q4::dbs.u=-6.56189
variable value opamp::Q5::dbc.u=0.384462
```

```
variable value opamp::Q5::dbe.u=0.457409
variable value opamp::Q5::dbs.u=-6.40434
variable value opamp::Q6::dbc.u=-6.82695
variable value opamp::Q6::dbe.u=-1.00862
variable value opamp::Q6::dbs.u=-9.82695
variable value opamp::Q7::dbc.u=-0.439423
variable value opamp::Q7::dbe.u=0.447386
variable value opamp::Q7::dbs.u=-5.40937
variable value opamp::Q8::dbc.u=0.384462
variable value opamp::Q8::dbe.u=0.457409
variable value opamp::Q8::dbs.u=-4.43982
variable value opamp::Q9::dbc.u=0.364157
variable value opamp::Q9::dbe.u=0.464419
variable value opamp::Q9::dbs.u=-5.56911
variable value opamp::Q10::dbc.u=0.36553
variable value opamp::Q10::dbe.u=0.457387
variable value opamp::Q10::dbs.u=-0.80985
variable value opamp::Q11::dbc.u=6.9984E-5
variable value opamp::Q11::dbe.u=0.173034
variable value opamp::Q11::dbs.u=-0.172976
variable value opamp::Q12::dbc.u=-1.40395
variable value opamp::Q12::dbe.u=0.173054
variable value opamp::Q12::dbs.u=-1.57701
partition
output solved equations
{ GENERATE AN ACSL MODEL FOR SIMULATION }
set LogCommands off
set ACSLold on
language acsl
enter experiment
@opamp.ctl
outfile opamp.csl
output program
outfile
```

APPENDIX K
SIMULATION PLOTS OF THE OPAMP CIRCUIT

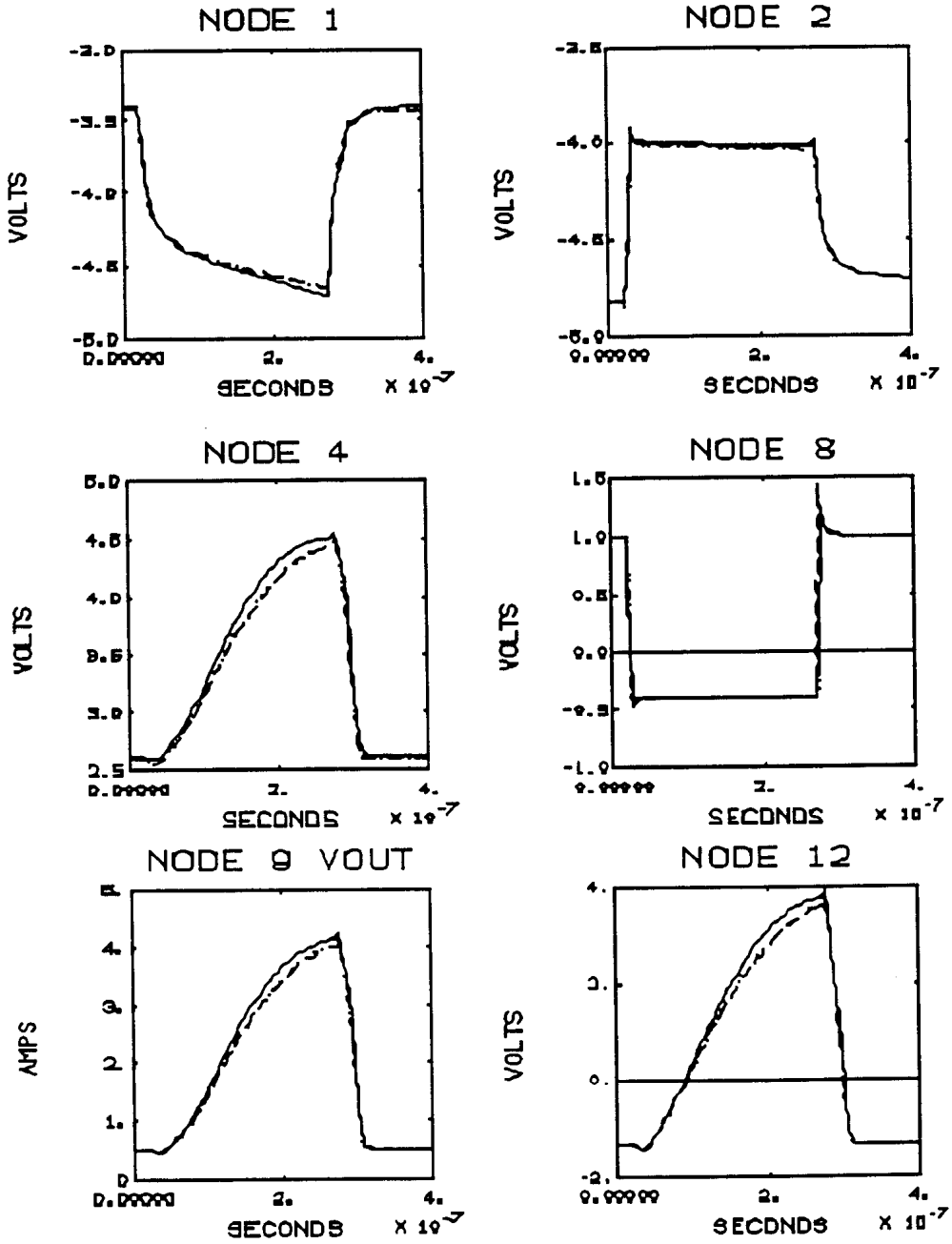
This appendix contains the simulation plots of the OPAMP Circuit. The parameter settings for the NPN and PNP BJTs arc as follows.

```
PNP   BF=100.0    BR=1.0      IS=0.11E-9   ISS=0.11E-9
      NF=1.0      NR=1.0      NS=1.0       GMINDC=1.0E-12
      ISC=0.0     ISE=1.0E-9  NC=2.0       NE=1.5
      VAF=9E+30   VAR=9E+30   IKF=9E+30    IKR=9E+30
      RB=200.0    RBM=100.0   RC=750.0     RE=123.3
      CJC=3.6E-12 CJE=5.7E-12 CJS=11.0E-12
      MJC=0.33    MJE=0.33    MJS=0.33
      VJC=0.75    VJE=0.75    VJS=0.75
      TF=1E-12    TR=1E-12    IRB=0.0      XCJC=1.0
      TRB1=0.0    TRM1=0.0    TRC1=0.0     TRE1=0.0
      TRB2=0.0    TRM2=0.0    TRC2=0.0     TRE2=0.0
      TNOM=25.0   TEMP=25.0   XTI=3.0      XTB=0.0   EG=1.16

NPN   BF=100.0    BR=1.0      IS=0.11E-9   ISS=0.11E-9
      NF=1.0      NR=1.0      NS=1.0       GMINDC=1.0E-12
      ISC=0.1E-6  ISE=0.0     NC=2.0       NE=1.5
      VAF=9E+30   VAR=9E+30   IKF=9E+30    IKR=9E+30
      RB=200.0    RBM=100.0   RC=750.0     RE=123.3
      CJC=3.6E-12 CJE=5.7E-12 CJS=11.0E-12
      MJC=0.33    MJE=0.33    MJS=0.33
      VJC=0.75    VJE=0.75    VJS=0.75
      TF=1E-12    TR=1E-12    IRB=0.0      XCJC=1.0
      TRB1=0.0    TRM1=0.0    TRC1=0.0     TRE1=0.0
      TRB2=0.0    TRM2=0.0    TRC2=0.0     TRE2=0.0
      TNOM=25.0   TEMP=25.0   XTI=3.0      XTB=0.0   EG=1
```

OPAMP CIRCUIT

BBSPICE - SOLID LINE
DYMOLA - DASHED LINE
PSPICE - DOTTED LINE



REFERENCES

[1] H. Elmqvist (1978), *A Structured Model Language/or Large Continuous Systems*, Ph.D. dissertation, Report CODEN: LUFTFD2/(TFRT-1015), Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden.

[2] H. Elmqvist (1993), *Dymola - Release Notes*, Pre-Release Version 1.8.1, Lund, Sweden.

[3] F. Cellier and H. Elmqvist (1993), "Automated Formula Manipulation Supports Object-Oriented Continuous-System Modeling", *IEEE Control Systems*, volume 13, no. 2, pp. 28-38, April 1993.

[4] M. Vlach (1992), "System Performance Simulator", U.S. Patent Number: 5,092,780; March 3,1992.

[5] F. Cellier (1991), *Continuous System Modeling*, Springer-Verlag New York Inc., New York, New York.

[6] Burr Brown Corp. (1987), *BBSPICE - User's Manual*, Tucson, Arizona.

[7] MicroSim Corporation (1988), *PSpice Manual*, Irvine, California.

[8] P. Van Halen (1988), "A New Semiconductor Junction Diode Space Charge Layer Capacitance Model," *Proceedings IEEE 1988 Bipolar Circuits & Technology Meeting*, (J. Jopke, ed.), Minneapolis, Minnesota, IEEE Publishing Services, New York, pp. 168-171.

[9] Meta-Software Inc. (1990), *HSPICE User's Manual*, Campbell, California.

[10] E. E. L. Mitchell and J. S. Gauthier (1986), *ACSL: Advanced Continuous Simulation Language – User Guide and Reference Manual*, Mitchell & Gauthier Associates, Concord, Massachusetts.

View publication stats