A Computer program for the extraction of bipolar transistor SPICE models

Larry Duane Delaney
A COMPUTER PROGRAM FOR THE EXTRACTION OF
BIPOLAR TRANSISTOR SPICE MODELS

by
Larry Duane Delaney

A Thesis Submitted
in
Partial Fulfillment
of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electrical Engineering

Approved by: Prof. Lynn Fuller, Thesis Advisor
Prof. Robert E. Pearson
Prof. R. Turkman
Prof. R. Unnikrishnan, Dept. Head

DEPARTMENT OF ELECTRICAL ENGINEERING
COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK
OCTOBER, 1992
A Computer Program for the Extraction of Bipolar Transistor Spice Models

I, Duane Delaney hereby grant permission to the Wallace Memorial Library of The Rochester Institute of Technology to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date Jan. 21, 1993
ABSTRACT

Each year semiconductor manufacturers spend millions of dollars in the development of new products. It can be very costly to create on silicon a newly developed circuit, especially if it is not very manufacturable or poorly designed. To eliminate risk it is of utmost importance that circuit simulation be done correctly in the early stages of development. Simulation can only be as good as the model being used. Almost all designers whether digital, analogue, or mixed mode use SPICE to simulate their circuits. SPICE accuracy is inherently dependent on the discrete element models being used, i.e. transistors, diodes, mosfets. Development of models usually includes physical measurements, SPICE parameter extraction from the measurement data, and then SPICE simulation to verify the extracted model parameters. This can be a tedious and time consuming process. To speed up this process as well as make it much easier to accomplish, a computer program has been written to aid in the extraction of SPICE parameters for bipolar transistors.

To use the program all that is needed is Gummel-Poon data, collector current vs collector-emitter bias voltage data, junction capacitance vs voltage data, and $F_t$ vs collector current data. The user can then methodically choose a parameter of interest and vary the value and immediately see the effect on simulation on screen in a graphical II
presentation. The user has the ability to simulate each of the above mentioned measurements and view the simulation simultaneously with the data. Using this technique the user can develop the entire SPICE model, including temperature effects and gain a very good working knowledge of parameter effects on the SPICE simulation.
# TABLE OF CONTENTS

INTRODUCTION ......................................................... 8

THEORY ................................................................. 9

EQUATIONS ............................................................... 9
Newton-Raphson Technique ........................................... 15
Parameter Description ............................................... 20
  IS, NF, BF ......................................................... 20
  ISE, NE ........................................................... 22
  RB, IRB, RBM, RE ............................................... 24
  IKF ................................................................. 26
  RC, VAF ........................................................... 28
REVERSE BIAS PARAMETERS .......................................... 30
  CJC, VJC, MJC, XCJC, CJE, VJE, MJE, CJS, VJS,
    MJS, FC ....................................................... 31
  TF, XTF, VTF, ITF ............................................... 35
  XTI, EG, XTB ..................................................... 38

How to Use the Program ............................................. 40
  Ic, Ib vs Vbe .................................................... 42
  Ic-Vce .......................................................... 47
  Capacitor ......................................................... 49
  Ft-Ic ............................................................. 50

IV
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NPN Symbol and Conventions</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>NPN Transistor with Integral limits</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>NPN Small Signal Model</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>IS, BF</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>Beta-IC</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>ISE</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>Beta, NE</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>Transistor Crosssection</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>IR Drop</td>
<td>24</td>
</tr>
<tr>
<td>10</td>
<td>IC-IKF</td>
<td>26</td>
</tr>
<tr>
<td>11</td>
<td>Beta-IKF</td>
<td>26</td>
</tr>
<tr>
<td>12</td>
<td>RC, VAF</td>
<td>28</td>
</tr>
<tr>
<td>13</td>
<td>Junction Capacitance</td>
<td>34</td>
</tr>
<tr>
<td>14</td>
<td>Typical $F_t$ vs $I_c$ Curve</td>
<td>35</td>
</tr>
<tr>
<td>15</td>
<td>Main Menu</td>
<td>40</td>
</tr>
<tr>
<td>16</td>
<td>$I_c$, $I_b$ vs $V_{be}$ (Gummel-Poon)</td>
<td>43</td>
</tr>
<tr>
<td>17</td>
<td>$I_c$, $I_b$ vs $V_{be}$ Bar Menu</td>
<td>44</td>
</tr>
<tr>
<td>18</td>
<td>$I_c$, $I_b$ vs $V_{be}$ Bias Menu</td>
<td>45</td>
</tr>
<tr>
<td>19</td>
<td>$I_c$, $I_b$ vs $V_{be}$ Parameter Menu</td>
<td>46</td>
</tr>
<tr>
<td>20</td>
<td>$I_c$ vs $V_{ce}$</td>
<td>47</td>
</tr>
<tr>
<td>21</td>
<td>$I_c$ vs $V_{ce}$ Bias Menu</td>
<td>48</td>
</tr>
<tr>
<td>22</td>
<td>Capacitor Screen</td>
<td>49</td>
</tr>
<tr>
<td>23</td>
<td>$F_t$-$I_c$ Screen</td>
<td>50</td>
</tr>
<tr>
<td>24</td>
<td>$F_t$-$I_c$ Parameter Menu</td>
<td>51</td>
</tr>
</tbody>
</table>
INTRODUCTION

The bipolar transistor model extraction program was written using the "C" language and is designed to be used on IBM or IBM compatible computers. The program uses the same "Gummel-Poon" model of the bipolar transistor that is used in the SPICE. The model and how it is implemented in the program are described in detail in the theory portion of this thesis. Also, a methodical approach to using the program is described and a description of each parameter is included.
THEORY

The circuit symbol and current conventions for an NPN transistor are shown in figure 1. This paper will discuss NPN transistors. PNP transistors are handled similarly with appropriate changes to the signs of the junction voltages.

EQUATIONS

The Gummel Poon model used by the SPICE is described by the following equations:
Collector Current:

\[ I_c = \frac{I_s}{q_b} \left( e^{\frac{V_{be}}{N_t \times V_t}} - e^{\frac{V_{bc}}{N_t \times V_t}} I_s \right) \left( e^{\frac{V_{bc}}{N_e \times V_t}} - 1 \right) - I_{sc} \left( e^{\frac{V_{bc}}{N_e \times V_t}} - 1 \right) \]  

(1)

Base Current:

\[ I_b = \frac{I_s}{B_f} \left( e^{\frac{V_{be}}{N_t \times V_t}} - 1 \right) - I_{se} \left( e^{\frac{V_{be}}{N_e \times V_t}} - 1 \right) - I_{sc} \left( e^{\frac{V_{bc}}{N_e \times V_t}} - 1 \right) + I_{sc} \left( e^{\frac{V_{bc}}{N_e \times V_t}} - 1 \right) \]  

(2)

Where:  
- BF = Forward maximum ideal current gain  
- BR = Inverse maximum ideal current gain  
- IS = Saturation Current  
- ISC = Base-collector leakage saturation current  
- ISE = Base-emitter leakage saturation current  
- NF = Forward current emission coefficient  
- NR = Reverse current emission coefficient  
- NE = Base-emitter leakage emission coefficient  
- NC = Base-collector leakage emission coefficient  
- VBE = applied base-emitter junction voltage  
- VBC = applied base-collector junction voltage  
- \( q_b \) = normalized majority base charge

All of these are SPICE parameters except \( q_b \) and are discussed in detail later in this paper. At this point it is
worthwhile to define the normalized majority base charge, $q_b$. This parameter provides necessary biasing dependency to the collector current. In the following analysis figure 2 defines all the limits of integration.

![NPN Transistor with Integral limits](image)

Figure 2 NPN Transistor with Integral limits

First, the total majority charge in the neutral base region is given by equation 3.

$$Q_b = \int_{x'_c}^{x'_e} q A_j p(x) \, dx$$

Where $A_j$ is the cross sectional area of the base and $q$ is the electronic charge. The zero bias majority base charge is given by equation 4. At zero bias, $p(x)$ can be approximated by $N_A(x)$. 

11
\[ Q_{bo} = \int_{x'_{e0}}^{x'_{c0}} qA_j x N_A(x) \, dx \] (4)

\( x'_{e0} \) and \( x'_{c0} \) are the zero bias values of \( x'_e \) and \( x'_c \) of figure 2. Now the normalized majority base charge is defined by equation 5.

\[ q_b = \frac{Q_b}{Q_{bo}} \] (5)

At low level injection the collector current is given by equation 6.

\[ I_c = \frac{qD_2 x N_A}{x'} \int_{x'_{e}}^{x'_{c}} \left[ e \left( \frac{v_{be}}{N_r x V_t} - 1 \right) - \left( e \frac{v_{bc}}{N_r x V_t} - 1 \right) \right] p(x) \, dx \] (6)

Multiplying by \( Q_{bo}/Q_{bo} \) yields:

\[ I_c = \frac{I_s}{q_b} x \left[ e \left( \frac{v_{be}}{N_r x V_t} - 1 \right) - \left( e \frac{v_{bc}}{N_r x V_t} - 1 \right) \right] \] (7)
The total majority base charge including the space charge layers is given by equation 8.

\[ Q_b = Q_{b0} + C_{je}V_{be} + C_{jc}V_{bc} + \tau_F I_{cf} + \tau_R I_{cr} \]  

(8)

where:
- \( C_{je} = \) Base-emitter junction capacitance
- \( C_{jc} = \) Base-collector junction capacitance
- \( \tau_F = \) Forward base transit time
- \( \tau_R = \) Reverse base transit time
- \( I_F \) and \( I_R \) are defined in the next equation

\( q_b \) is then defined as:

\[ q_b = 1 + \frac{V_{be}}{V_{AR}} + \frac{V_{bc}}{V_{AF}} \left( \frac{1}{Q_{bo} q_b} \right) \left( e^\frac{V_{be}}{N_tV_t} - 1 \right) + \frac{\tau_R I_s}{Q_{bo} q_b} e^\frac{V_{bc}}{N_cV_t} - 1 \]  

(9)

where:
- \( VAF = \) Forward early voltage \( Q_{b0}/C_{jc} \)
- \( VAR = \) Reverse early voltage \( Q_{b0}/C_{je} \)

The second and third terms model base width modulation and the last two terms model high level injection. In the forward mode at high level injection the slope of the natural log of the collector current vs \( V_{be} \) is \( 1/(2V_t) \). At low level injection the slope is approximately \( 1/V_t \). This is modeled by \( q_b \) as follows. First let \( q_1 \) equal the first three terms in the equation above. Let \( q_2 \) equal the last two terms multiplied by \( q_b \). Then:
\[ q_b = q_1 + \frac{q_2}{q_b} \]  

or

\[ q_b^2 - q_b \times q_1 - q_2 = 0 \]

Solving for \( q_b \) yields:

\[ q_b = \frac{q_1}{2} + \sqrt{\frac{q_1^2}{4} + q_2} \]

At high level injection:

\[ q_b = \sqrt{q_2} \sqrt{\frac{\tau_f \times I_s}{Q_{bo}}} \times e^{\frac{v_{be}}{N_f \times V_c}} \]

If \( NF \) equals 1 which is usually the case then:

\[ I_c = \frac{I_s (e^{\frac{v_{be}}{V_t}} - 1)}{\sqrt{\frac{\tau_f \times I_s}{Q_{bo}}} \times e^{\frac{v_{be}}{2V_c}}} \times \sqrt{\frac{Q_{bo} \times I_s}{\tau_f} \times e^{\frac{v_{be}}{2V_c}}} \]
and hence the explanation for the $1/(2V_t)$ slope mentioned earlier at high level injection.

**Newton-Raphson Technique**

All of these parameters will be discussed in detail later in this paper. The equivalent small signal circuit used by the SPICE to model the bipolar transistor is pictured in figure 3. On examination of figure 3 it is readily seen that $v_{be}$ and $v_{bc}$ are dependent on voltage drops created by $I_c'$ and $I_b$ across $rb$, $re$, and $rc$. Therefore $I_c'$ and $I_b$ are transcendental in nature and require numerical techniques to obtain a solution at a given biasing configuration. $I_c$ and $I_b$ are both functions of $V_{be}$ and $V_{bc}$. Therefore the Newton-Raphson technique was implemented in the "C" program due to it's rapid convergence characteristics.

![Figure 3 NPN Small Signal Model](image)

The Newton-Raphson technique is implemented in the program as follows:
The program runs iteratively until some small allowable difference delta is obtained between \( x^i \) and \( x^{(i+1)} \). \( x_1, x_2, f_1, \) and \( f_2 \) take on different values as the user interacts with the program. There are four different modes to use the program in; Gummel-Poon, \( I_c \) vs \( V_{ce} \), \( F_t \) vs \( I_c \), and junction capacitance vs voltage.

Usually forward Gummel-Poon measurements mean simply shorting the base and collector terminals and sweeping \( V_{be} \) over some range of positive values. For the inverse mode base and emitter are shorted together and \( V_{bc} \) is swept over some range of positive values.

When in the Gummel-Poon analysis mode, the Newton-Raphson variables are defined as follows (forward mode of operation):

\[
\begin{align*}
X_1 &= V_{be} \\
X_2 &= V_{bc} \\
f_1 &= V_{b'e'} - I_b x (I_b + I_e) - I_c x I_e - V_{be} \\
f_2 &= V_{c'e'} - I_c x I_c - (I_c + I_b) x I_e + V_{bc} - V_{be}
\end{align*}
\]
The function $f_1$ is acquired by applying Kirchoff's voltage law to the base emitter loop including a variable voltage source $v_{b'e'}$ connected between $B'$ and $E'$ in figure 2. Similarly $f_2$ is acquired by applying Kirchoff's voltage law to the collector-emitter loop including a variable voltage source $v_{c'e'}$ connected between $C'$ and $E'$ in figure 2.

The $I_c$ vs $V_{ce}$ measurements require forcing a family of base currents and sweeping $V_{ce}$ over some range at each base current. In the $I_c$ vs $V_{ce}$ mode, the variables are defined as (forward mode of operation):

$$
X_1 = v_{be} \\
X_2 = v_{bc} \\
f_1 = I_{bforce} - I_{bcalculated} \\
f_2 = v_{c'e'} - I_c \times I_c - (I_c + I_b) \times I_e + v_{bc} - v_{be}
$$

Here $I_{bsource}$ represents the constant current source value used for a $V_{ce}$ sweep. The Newton-Raphson technique minimizes the error between calculated and actual values of base current. Here $f_1$ is acquired in the same way as in the Gummel-Poon mode.

In the $F_t$ vs $I_c$ mode, the variables are defined as (forward mode of operation):

17
\[ x_1 = v_{be} \]
\[ x_2 = v_{bc} \]
\[ f_1 = I_{c\text{f source}} - I_{calc} \]
\[ f_2 = v_{c'e'} - I_c X_c - (I_c + I_b) \times I_e + v_{bc} - v_{be} \]  

(18)

Again, \( I_{c\text{source}} \) represents the constant current source value used for the \( F_t \) measurements. Usually several different collector currents are chosen and \( F_t \) is evaluated at each current value.

Junction capacitances are usually measured over a sweep of reverse bias voltages, say -10 to 0 volts for example. At forward bias voltages the equation below used by SPICE is not valid. SPICE uses other parameters to be discussed later to model the forward bias region.

\[ C_j = \frac{C_{j0}}{\left(1 - \frac{V_j}{\phi}\right)^{m_j}} \]  

(19)

For capacitor modelling, the first three capacitances measured are used to extract \( C_{j0} \), \( \phi \), and \( m_j \). Then the next three and so on until all values are used. The \( C_{j0} \), \( \phi \), and \( m_j \) values extracted at each iteration are averaged to obtain the final parameters. This tends to filter out
noisy data points. It should be noted that for this routine to be reliable the data should not be too noisy. Each step of the capacitor routine is implemented as follows:

\[
\begin{bmatrix}
    C_j^{(i+1)} \\
    \phi^{(i+1)} \\
    M_j^{(i+1)}
\end{bmatrix} =
\begin{bmatrix}
    C_j^{i} \\
    \phi^{i} \\
    M_j^{i}
\end{bmatrix} -
\begin{bmatrix}
    \frac{\partial C_1}{\partial C_j} & \frac{\partial C_1}{\partial \phi} & \frac{\partial C_1}{\partial M_j} \\
    \frac{\partial C_2}{\partial C_j} & \frac{\partial C_2}{\partial \phi} & \frac{\partial C_2}{\partial M_j} \\
    \frac{\partial C_3}{\partial C_j} & \frac{\partial C_3}{\partial \phi} & \frac{\partial C_3}{\partial M_j}
\end{bmatrix}^{-1}
\begin{bmatrix}
    C_1 \\
    C_2 \\
    C_3
\end{bmatrix}
\]

(20)

where:  \( C_1 = C_{1\text{measured}} - C_{1\text{calculated}} \)

\( C_2 = C_{2\text{measured}} - C_{2\text{calculated}} \)

\( C_3 = C_{3\text{measured}} - C_{3\text{calculated}} \)

In each case the Newton-Raphson algorithm is used to minimize the error between a known and calculated value. The program limits the number of iterations to fifty for each bias value. It will almost always converge before reaching the iteration limit.
As seen in figure 4, IS is the collector current 0-vbe intercept on the gummel-poon plot. Physically IS is defined by equation 21.

$$I_s = \frac{qD_n n_i^2 A_j}{x_{ce}} \int_{x_{ce}}^{x} p_0(x) \, dx$$

(21)

where $p_0(x)$ is the equilibrium hole concentration in the neutral base and space charge layers. Increasing IS will cause $I_c$ and $I_b$ to move up on the gummel-poon plot. Decreasing the value has the reverse effect.
NF dictates the slope (on the gummel-poon plot) of $I_C$ upto high level injection, and $I_B$ in the midcurrent range. The slope is given by equation 22. Increasing NF decreases the slope while decreasing NF obviously increases the slope.

$$\text{Slope} = \frac{1}{NF \times V_t}$$

(22)

BF is the ideal forward maximum current gain ($I_C/I_B$). Increasing BF raises the entire curve on the Beta vs $I_C$ plot (figure 5). It also causes $I_B$ to lower on the gummel-poon plot while $I_C$ remains fixed. This is explained by reexamination of equations 1 and 2 and noting that $I_B$ is inversely proportional to beta while $I_C$ is independent. Decreasing BF has the reverse effect.
Figure 7 shows the phenomena referred to as low current beta degradation. It is caused by components of base current that in other regions of operation are negligible. These components are recombination of carriers at the surface and base-emitter space charge layer and formation of emitter-base surface channels. The major component is base-emitter space charge layer recombination. It is modeled by ISE and NE. Figure 6 shows ISE is the base current 0-vbe intercept on the gummel-poon plot. NE determines the slope in this region as:

\[
Slope = \frac{1}{NE \times V_t}
\]  

(23)

The slope of beta vs \( \ln(I_c) \) in the low current region is defined as:
The high current beta degradation seen in figure 7 is due to high level injection which was discussed earlier. Here again the slope of $I_c$ on the gummel-poont plot is proportional to $1/2$ instead of 1.

\[ \text{Slope} = 1 - \frac{1}{NE} \]
The base resistance of a bipolar transistor is normally the largest parasitic resistance present. Figure 8 shows how the base resistance is composed.

The base resistance is a base current dependent variable resistor. At low currents the base current follows the path along the resistor RB in figure 8. At high base currents it follows the path along the resistor RBM. This occurs due to the voltage drop induced by I_b across RB. There exists a voltage gradient along the resistor that more heavily forward biases the base-emitter junction nearer the surface rather than under the emitter. When the base current increases this junction becomes so heavily forward biased that almost all of the current will follow this least resistive path, and hence
RBM becomes a necessary parameter. IRB is the value of base current that produces a base resistance halfway between RB and RBM. Spice models this phenomena using the following analytic equation:

\[
R_{\text{base}} = RBM + 3(RB - RBM) \left( \frac{\tan z - z}{z \tan^2 z} \right)
\]

(25)

where:

\[
z = \frac{-1 + \sqrt{1 + \frac{144I_b}{\pi^2 IRB}}}{\frac{24}{\pi^2} \sqrt{\frac{I_b}{IRB}}}
\]

(26)

RE is shown in figure 8 and is usually small in value. It's effect is seen in the high base current deviation from a straight line on the gummel poon plot. \( R_{\text{base}} \) causes most of the deviation though (figure 9). Increasing any of the resistor values increases the IR drop of the base current. IRB controls the base current level at which a significant change in \( R_{\text{base}} \) occurs.
IKF is a parameter used to calculate $q_b$. Referring to the gummel-poorn plot IKF is the intersection of the low level injection region of the collector current with the high level injection region. Or mathematically:

$$I_{KF} = \sqrt{\frac{Q_{bo}I_s}{\tau_F}} e^{\frac{V_{KF}}{2V_c}}$$  \hspace{1cm} (27)$$

Where $V_{KF}$ is the value of $V_{be}$ to get $I_c = I_{KF}$ (see equation 14). This is the high level injection term for $I_{KF}$. The low level injection expression for $I_{KF}$ is given by:

$$I_{KF} = I_s e^{\frac{V_{KF}}{V_c}}$$  \hspace{1cm} (28)$$
Solving these two equations yields

\[ I_{kip} = \frac{Q_{b0}}{t_p} \]  \hspace{1cm} (29)

Using IKF therefore eliminates the need to perform the integration described in equation 4 to get \( Q_{b0} \). Figures 11 and 12 show the effect of varying IKF. Increasing IKF moves the high level injection portion of the collector current curve up on the gummel-poon plot. It also moves the high current degradation region of the beta curve to the right. Decreasing IKF has the opposite effect in both cases.
Figure 8 shows the components of RC (RC1, RC2, RC3). Surprisingly enough the effect of RC is negligible on $I_c$ of the gummel-poon plot. It's effect is seen mainly in the saturation area of the $I_c$ vs $V_{ce}$ plot. RC directly controls the slope of collector current vs $V_{ce}$. Increasing RC decreases the slope in this region of operation. The slope is actually proportional to $RC + RE$, but generally $RE$ is much smaller than $RC$. Here the voltage drop is approximately:

$$I_c(RC+RE) + V_{cesat}$$  \hspace{1cm} (30)

$V_{cesat}$ and $I_cRE$ are usually small compared to $I_cRC$, so quite often people use the slope in this region to approximate RC which can lead to error since the slope changes with different values of constant base current for the same $V_{ce}$.  

sweep.

The effect of early voltage VAF is readily seen in the slope of the I vs V curves in the forward active region of operation (figure 10). Here a small VAF yields a steeper slope than a large VAF. As mentioned earlier, SPICE uses VAF to model base width modulation indirectly through q_b. Base width modulation is primarily the result of the varying base collector space charge layer width by changing external V_{ce}.  

29
### REVERSE BIAS PARAMETERS

All of the parameters mentioned up to this point are forward bias parameters. Most of these parameters have a "dual" in the reverse bias region of operation. Instead of repeating descriptions already given, the table below gives the new parameters with their dual in the forward bias mode and a brief description.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Forward</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BR</td>
<td>BF</td>
<td>11</td>
<td>Reverse Ideal Maximum current gain</td>
</tr>
<tr>
<td>VAR</td>
<td>VAF</td>
<td>16</td>
<td>Reverse Early Voltage</td>
</tr>
<tr>
<td>IKR</td>
<td>IKF</td>
<td>17</td>
<td>Reverse knee current</td>
</tr>
<tr>
<td>NR</td>
<td>NF</td>
<td>11</td>
<td>Reverse current emission coefficient</td>
</tr>
<tr>
<td>TR</td>
<td>TF</td>
<td>18</td>
<td>Ideal reverse transit time</td>
</tr>
<tr>
<td>ISC</td>
<td>ISE</td>
<td>13</td>
<td>Base-collector leakage saturation current</td>
</tr>
<tr>
<td>NC</td>
<td>NE</td>
<td>13</td>
<td>Base-collector leakage saturation coefficient</td>
</tr>
</tbody>
</table>
CJC, VJC, MJC, XCJC, CJE, VJE, MJE, CJS, VJS, MJS, FC

These are the parameters that are used to model all of the voltage dependent junction capacitances. A short description of each follows (refer to figure 3).

CJC = base collector junction capacitance
VJC = base collector junction built in voltage
MJC = base collector junction grading coefficient
CJE = base emitter junction capacitance
VJE = base emitter junction built in voltage
MJE = base emitter junction grading coefficient
CJS = collector substrate junction capacitance
VJS = collector substrate junction built in voltage
MJS = collector substrate grading coefficient
FC = coefficient for forward bias depletion capacitance formula

XCJC = Fraction of base collector capacitance attached to external node B' in figure 2.
(determines $C_{box}$ in figure 3 and equation 34)

The general form for junction capacitance is as follows:

$$C_j = \frac{C_{j0}}{\left(1 - \frac{V_j}{\phi}\right)^{M_j}}$$ (31)
All of the junction parameters are solved automatically once a valid file containing data is read into the program.

It is important to understand the parameters XCJC and FC. In earlier versions of SPICE the base collector capacitance was modelled by one lump capacitor CJC. Due to the distributed nature of this capacitance and for added accuracy this capacitance can now be broken into two capacitors. One is internal and one external to $R_b$ (see figure 3). The external capacitor becomes

$$C_{cbx} = (1 - XCJC) C_{jc}$$

(32)

while the internal capacitor becomes

$$C_{jc_{new}} = XCJC C_{jc_{old}}$$

(33)

where XCJC takes values between 1 and 0. This leads to a more accurate model.

FC is used to model junction capacitance at forward bias. For slight forward bias values equation 34 is still valid but beyond a very slight forward bias equation 34 predicts infinite capacitance. SPICE avoids this by using a straight line approximation. Referring to figure 14, FC becomes the breakpoint by the value $FC \times \Phi_i$. A straight line approximates the capacitance having the same slope as the curve at this point. This relationship is shown in figure 14.
Spice then uses the following equation in place of equation 34.

\[ C_j = \frac{C_{j0}}{F_2} \left( F_3 + \frac{M_j V_j}{\phi} \right) \]  

(34)

where:

\[ F_2 = (1 - FC)^{1 + M_j} \]  

(35)

\[ F_3 = 1 - FC(1 + M_j) \]  

(36)

These equations apply to equations 32, 33, and 34 above for the junction capacitance terms. The collector substrate capacitance is modelled somewhat differently when forward biased. This is shown in equation 43.

\[ C_{cs} = C_{JS}(0) \left( 1 + \frac{MJS V_{cs}}{VJS} \right) \]  

(37)

The straight line approximation is adequate since once the junction becomes forward biased the diffusion capacitance becomes more dominant than the junction capacitance.
Figure 13 Junction Capacitance
These parameters are used by the SPICE to model forward transit charge. The transition frequency $F_t$ or unity gain frequency is determined by these parameters. $F_t$ is calculated in SPICE by the following equation:

$$F_t = \frac{g_m}{2\pi(C_s + C_r + C_{bcx})}$$  \(38\)

Refer to figure 3 for these parameters.

At low currents $F_t$ is proportional to $g_m$. As $g_m$ increases with collector current so does $F_t$. At still higher collector currents the base emitter diffusion capacitance increases canceling the increases in $g_m$. $F_t$ is constant in this region. Further increases in collector current result in saturation, base push out, and space charge limited current.
flow reducing $F_t$. To see how these parameters affect $F_t$ the capacitances must be examined. The capacitances are calculated as follows:

$$C_\pi = \tau_F \frac{I_g}{V_t} + C_{j\epsilon}(0) \left(1 - \frac{V_{be}}{\phi_c}\right)^{-M_{j\epsilon}} \tag{39}$$

$$C_\mu = \tau_F \frac{I_g}{V_t} + C_{j\epsilon}(0) \left(1 - \frac{V_{bc}}{\phi_c}\right)^{M_{j\epsilon}} \tag{40}$$

$$C_{bcx} = C_{j\epsilon}(0) \left(1 - X_{cjc}\right) \left(1 - \frac{V_{b'c}}{\phi_c}\right)^{-M_{j}\epsilon} \tag{41}$$

where:

$$\tau_F = TF \left[1 + XTF \left(\frac{I_{cc}}{I_{cc} + I_{TF}}\right)^2 e^\frac{V_{bc}}{1.44V_{TF}}\right] \tag{42}$$

where:

$$I_{cc} = \frac{I_g}{g_b} \left(e^{\frac{V_{bs}}{NFV_t}} - 1\right) \tag{43}$$
The equation for $\text{Tau}_f$ is a purely empirical formula derived to fit the $F_t$ characteristics as a function of current. The first term in equations 32 and 33 is referred to diffusion capacitance and models the mobile carrier stored charge. The second term is referred to as the junction capacitance. It is set up by the immobile ions present in the space charge layer.

In conclusion it is seen that $F_t$ is a function of transconductance, diffusion capacitance (and thus base transit time), and junction capacitance (and thus $V_{be}$ and $V_{bc}$).
XTI, EG, XTB

XTI is the saturation current temperature exponent while EG is the energy gap. For a typical bipolar transistor XTI is 3 and EG is 1.12 eV. The temperature dependence of IS is modelled as follows:

\[ I_s(T_2) = I_s(T_1) \left( \frac{T_2}{T_1} \right)^{XTI} e^{\left[ \frac{qEG(0)}{kT} \right] \left( 1 - \frac{T_2}{T_1} \right)} \]  \hspace{1cm} (44)

where:

\[ EG(T) = EG(0) - \frac{\alpha T^2}{\beta - T} \]  \hspace{1cm} (45)

\[ \alpha = 7.02 \times 10^{-4} \]
\[ \beta = 1108 \]
\[ EG(0) = 1.16 \text{ eV} \]

XTB is the forward and reverse beta temperature coefficient. The temperature dependence of beta is as follows:

\[ BF(T_2) = BF(T_1) \left( \frac{T_2}{T_1} \right)^{XTB} \]  \hspace{1cm} (46)

XTB is also used to model the temperature dependence of ISE and ISC as follows:
\[
\begin{align*}
ISE(T_2) &= ISE(T_1) \left( \frac{T_2}{T_1} \right)^{\frac{I_{\text{TB}}}{I_{\text{NE}}}} \left[ \frac{IS(T_2)}{IS(T_1)} \right]^{\frac{1}{N_{\text{E}}}} \\
ISC(T_2) &= ISC(T_1) \left( \frac{T_2}{T_1} \right)^{\frac{I_{\text{TB}}}{I_{\text{NC}}}} \left[ \frac{IS(T_2)}{IS(T_1)} \right]^{\frac{1}{N_{\text{C}}}}
\end{align*}
\]
How to Use the Program

The program can be run from the floppy disk or copied to a hard drive and run from the hard drive. At the prompt simply type "BIP". The program will begin execution and the following choices will be presented in a menu box.

<table>
<thead>
<tr>
<th>Ic, Ib vs Vbe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ic-Vce</td>
</tr>
<tr>
<td>Capacitor</td>
</tr>
<tr>
<td>Ft-Ic</td>
</tr>
<tr>
<td>Include Model</td>
</tr>
<tr>
<td>Save model</td>
</tr>
<tr>
<td>Temperature-C</td>
</tr>
<tr>
<td>Quit</td>
</tr>
</tbody>
</table>

Figure 15 Main Menu

At this point the user should set the temperature if the data was measured at other than 27 degrees C. Also, if the user is verifying a model, he or she should use the "Include Model" choice. The user will be prompted for further information (i.e. celcius temperature and/or filename of file containing the SPICE model for verification). The program will read in any valid SPICE parameters from an ASCII text file.

At least six files are needed for complete modelling of the bipolar transistor: Ic, Ib vs Vbe, Ic vs Vce, Capacitor vs voltage (base-emitter, base-collector, and collector-substrate), and Ft vs Ic. If temperature modelling is to be performed, an additional Ic, Ib vs Vbe file will be necessary for each additional temperature. The
room temperature model should be completed first.

Following is a short description of the various simulation screens and how to use them.
**Ic, Ib vs Vbe**

This choice should be the starting point for developing the model. The necessary data file should be formatted as follows:

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Vbe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column 2</td>
<td>Base Current</td>
</tr>
<tr>
<td>Column 3</td>
<td>Collector Current</td>
</tr>
<tr>
<td>Column 4 (optional)</td>
<td>Beta (Ic/Ib)</td>
</tr>
</tbody>
</table>

If beta is not included in the file, the program will compute it for each $V_{be}$. After initially choosing Ic, Ib vs Vbe the screen will appear as in figure 16.

Striking the enter key at this point will cause the bar menu to appear as in figure 17. To start the user should use the arrow keys or mouse to choose the "Data" menu choice. If using the arrow keys, striking enter will choose the highlighted menu choice. If a mouse is used striking the left key will choose the menu choice under the cursor. The user will be prompted for a file. The data will appear as discrete points on the plot. If the file is not found the user will not be notified, there will simply be no data present on the plot.

42
Next, the user should choose the "Bias" menu choice. The screen should appear as in figure 18. The first two choices, "NPN" and "Forward" will toggle to "PNP" and "Reverse" respectively. Choosing "Forward" will cause forward mode simulation while "Reverse" will cause reverse mode simulation. The rest of the choices on the menu are self explanatory. When the user is satisfied with the bias setup he or she should click on "Done".

Finally, the user should select the "Parameter" menu choice. The screen should appear as in figure 19. The parameters should be chosen in the order that they appeared
earlier in this document during the simulation to minimize the chance of non-convergence. The parameter chosen will be incremented or decremented whenever the up or down arrow keys are respectively pressed. The program will run a new simulation each time a parameter value is changed. To increase or decrease the amount a parameter is changed with each keypress, the user can choose "Increment" from the bar menu. A value between zero and one should be entered. The parameter will change as a fraction of its present value with each keypress.

The goal is to systematically adjust the parameters so
that the simulation and data match. This will be obvious when the simulation curve directly overlays the data.

It should be obvious from figures 16, 17, 18, and 19 that Beta vs Ic is plotted with the so-called gummel-poon plot. This will aid in better modelling the low and high current beta degradation mentioned earlier. This curve is quite sensitive to the parameters BF, ISE, NE, and IKF.

Figure 18 Ic, Ib vs Vbe Bias Menu
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Replot</th>
<th>Data</th>
<th>Increment</th>
<th>Bias</th>
<th>Quit</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS=1.0e-016</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF=1.0e+002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NF=1.0e+000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAF=1.0e+002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IKF=1.0e-002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISE=1.0e-016</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NE=1.5e+000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BR=1.0e+000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NR=1.0e+000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAR=1.0e+002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IKR=1.0e-002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISC=1.0e-016</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NC=2.0e+000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RB=1.0e+002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IRB=1.0e-006</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMB=1.0e+001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BE=1.0e+000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC=1.0e+000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XTB=1.0e+000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EG=1.1e+000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XTI=3.0e+000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Up-Down Arrow or Enter  Vce=Vbe  Temp= 27.0 C

Figure 19 Ic, Ib vs Vbe Parameter Menu
Choosing Ic-Vce from the main menu will result in the screen appearing as in figure 20. Again, striking enter will yield the bar menu. The only difference in these bar menu choices and those of the gummel-poont simulation is the "Bias" option. Figure 21 shows the options available under "Bias". Again "NPN" and "Forward" toggle to "PNP" and "Reverse" respectively. The rest of the options are self explanatory.

**Figure 20 Ic vs Vce**

Usually, CPU use and simulation time are of no concern, but it should be pointed out that the user should be wise in using the "Bias" option. The number of simulation points
should be minimized to reduce CPU time. When modelling Rc, the simulation bias should be such that only the saturated region of operation should be used. No information is gained from the active region and using this area of operation will only increase simulation time. Also only two or at most three base currents are needed to model Rc and VAF, and quite often only a few points are needed for an accurate fit. When modelling VAF, use only the active region of operation since no information regarding VAF is obtained from the saturation area of operation. The Newton-Raphson technique is powerful but in some instances it may have a difficult time converging.

Figure 21 Ic vs Vce Bias Menu
Capacitor

When using the Capacitor simulation, a data file is needed that contains the bias voltage in column 1 and capacitance values in column 2. The user will be prompted for the capacitor type (base-emitter, base-collector, or collector-substrate), and a data file. If the data file is not found the simulation and fit will not be performed.

The resultant screen should appear as in figure 22. All related parameters are automatically extracted by the program.

![Base-Emitter Capacitance vs Voltage](image)

**Figure 22 Capacitor Screen**
**Ft-Ic**

The Ft vs Ic simulation appears as in figure 23.

![Ft vs Collector Current](image)

**Figure 23 Ft-Ic Screen**

Pressing the enter key will cause the bar menu to appear. The only difference between the choices available here and those of the previously described screens is the "Parameter" and "Bias" options. The parameter choices are shown in figure 24. Even though the capacitance values are automatically extracted in the capacitor simulation, they are available here for fine tuning the $F_t$ simulation.

The bias options are shown in figure 25. Again, "NPN" and "Forward" toggle to "PNP" and "Reverse" as in the previous
cases. Here each curve is driven by $V_{ce}$ much like the $Ic$-$Vce$ curves are driven by base current. $Ic$ becomes the sweep parameter for this simulation. Since this is a very complex simulation, care should be taken to minimize the number of simulation points. It is also important to choose data points and simulation points that will cover the entire typical $F_t$ curve as seen in figure 13. This will require careful and well thought out choices of biasing during measurement of $F_t$.

![Parameter Menu](image)

*Figure 24 $F_t$-$Ic$ Parameter Menu*
Figure 25 Ft-Ic Bias Options
Additional Notes

It is important to save the model that has been developed regularly throughout the modelling process. This is done by selecting the "Save Model" option of the main menu (see figure 15). The user is prompted for a filename for storing the model to. This model is a valid "MODEL" card that can be used in a SPICE input deck for simulation.

When using the "Include Model" option of the main menu, be careful to include the drive if the file is on other than the "C" drive. This also applies when choosing the "Data" option available on the bar menu that appears on each simulation screen.
/*
* SPFT.C - Finds Spice model for bipolar transistors.
* Written by Duane Delaney 1991
*/

#include <graph.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>
#include "menu.h"
#include "mouse.h"

/* Function prototypes */
void Gummel( void );
void Icvce( void );
void Capacitor( void );
void Ft( void );
void Save( void );
void Incl( char filename[20]);
void plot(int numcur,int icount, double y[5][100],double x[100],double xmin,
    double xmax,double ymin,double ymax,int new,short col,short writemode,
    short lox, short loy);
void newminmax(double *max, double *min);
void getgo(int count,int *chose,int *icur);
void pnjlims(double *vnew, double *vold);
void plotpix(int numcur, double y[5][100],double x[100],
    double xrange,double yrange, short col, int startcur);
void write_text(short r, short c, char *txt);
void write_ftext(short x, short y, short i, short j, unsigned char *txt);
int choice( int x, int y, int numstr, unsigned char *mnu[20],int place);
void setvid();
void Spice();
void Temper();

#define MAXLINELEN 81
#define pi 3.141592654
/* default SPICE parameters (mine) */ double f[] = 
{
0,1.0E-16,100.0,1.0,1.0E2,1.0E-2,1.0E-16,1.5,1.0,1.0,1.0E2,1.0E-2,
1.0E-16,2.0,100.0,1.0E-6,10.0,1.0,1.0,1.0,1.11,3.0,100.0E-12,0.9,2.0,
1.0E-3,1.0E-9,2.0E-12,0.75,0.33,2.0E-12,0.75,0.33,0.5,1.0,2.0E-13,0.75,0.5,1.0
};
char *sp[] = 
{
"IS","BF","NF","VAF","IKF","ISE","NE","BR","NR","VAR","IKR","ISC",
"NC","RB","IRB","RBM","RE","RC","XTB","EG","XTI","TF","XTF","VTF","ITF",
"TR","CJE","VJE","MJE","CJC","VJC","MJC","FC","XCJC","CJS","VJS","MJS","PTF"
};
enum mods
{
BL,IS,BF,NF,VAF,IKF,ISE,NE,BR,NR,VAR,IKR,ISC,NC,RE,RC,XTB,
EG,XTI,TF,XTF,VTF,ITF,TR,CJE,VJE,MJE,CJC,VJC,MJC,FC,XCJ,CJS,
VJS,MJS,PTF
};
int iMode,kind=0,time=0;
do u b l e bi,ci,vce,vbe,vt,vcrit,incr=.2,vbcoo,vbeoo,fti,te,je,jc,js;
double se,sc,si,bfo,bro,vej,vcj,vsj;
unsigned char *mnustuff[6] = {"Parameter","Replot","Data","Increment","Bias","Quit"};
unsigned char *mnupar[20];
unsigned char *mnuchoice[6];
/* type=1 npn -1 pnp */
int type=1;
/* iwhich=1 forward -1 inverse */
int iwhich=1;
double idata[5][100],vdata[100],icdata[100];
int gumdata=0,vcdata=0,gumcount,vcecount,vcecurve,numv;
int fart=0,ftdata=0,ftcount,ftcurve;
struct videoconfig vc;
char olttype[4]="NPN";

/* Array and enum for main menu */
ITEM mnuMain[ ] =
{
    /* Highlight Char Pos */
    { 9, "Ic,Ib vs Vbe" }, /* V 9 */
    { 0, "Ic-Vce" }, /* I 0 */
    { 0, "Capacitor" }, /* C 0 */
    { 0, "Ft-Ic" }, /* F 0 */
    { 8, "Include Model" }, /* M 8 */
    { 0, "Save Model" }, /* S 0 */
    { 0, "Temperature-C" }, /* T 0 */
    { 0, "Quit" }, /* Q 0 */
    { 0, "" }
};

/* Define constants (0, 1, 2,...) for menu choices */
eenum CHOICES
{
    GUM, IC, CAP, FT, INCM, SAVE, TEMPER, QUIT
};

/* Arrays of video mode menu items and of corresponding mode numbers.
 * Each has a temporary array containing all items, and a pointer version
 * including all except Olivetti.
 */
ITEM mnuModesT[ ] =
{
    /* Highlight Char Pos */
    { 0, "ORESCOLOR " }, /* O 0 */
    { 4, "MRES4COLOR " }, /* 4 4 */
    { 4, "MRESNOCOLOR " }, /* N 4 */
    { 4, "HRESBW" }, /* B 4 */
    { 0, "MRES16COLOR " }, /* M 0 */
    { 0, "HRES16COLOR " }, /* H 0 */
    { 0, "ERESCOLOR " }, /* E 0 */
    { 4, "VRES2COLOR " }, /* 2 4 */
    { 0, "VRES16COLOR " }, /* V 0 */
    { 1, "MRES256COLOR " }, /* R 4 */
    { 0, "" }
};

ITEM *mnuModes = &mnuModesT[1]; /* Default is no Olivetti mode */
int aModesT[ ] =
{
    _ORESCOLOR,
    _MRES4COLOR,
    _MRESNOCOLOR,
    _HRESBW,
    _MRES16COLOR,
    _HRES16COLOR,
int *aModes = &aModesT[1]; /* Default is no Olivetti mode */

int main()
{
    int rowMid, colMid, i;
    int fFirstTime = TRUE;
    int iMainCur = 0, iModesCur = 0;
    struct _fontinfo fi;

    if (_registerfonts("TMSRB.FON") <= 0)
        _outtext("you forgot the fonts");
    getchar();
    exit(1);

    if (_setfont("t'tms rmn'h15w10b") < 0)
    {
        _outtext("can't set font");
        getchar();
    }
    if (_getfontinfo(&fi))
    {
        _outtext("Can't get font info");
        exit(1);
    }              
    _displaycursor(_GCURSOROFF);
    _getvideoconfig(&vc);
    rowMid = vc.numtextrows / 2;
    colMid = vc.numtextcols / 2;

    /*
    * If no color stop !
    */
    switch (vc.adapter)
    {
        case _OCGA:
            mnuModes = &mnuModesT[0]; /* Turn on Olivetti mode */
            aModes = &aModesT[0];
        case _CGA:
            mnuModesT[4].achItem[0] = '\0'; /* Turn off EGA modes */
            iMode = _MRES4COLOR;
            break;
        case _HGC:
            mnuModesT[7].achItem[0] = '\0';
            break;
        }
iMode = _HERCMONO;
break;
case _OEGA:
    mnuModes = &mnuModesT[0];          /* Turn on Olivetti mode */
    aModes = &aModesT[0];
    case _EGA:
        mnuModesT[7].achItem[0] = '\0';    /* Turn off VGA modes */
        if( vc.memory > 64 )
            iMode = _ERESCOLOR;
        else
            iMode = _HRES16COLOR;
        break;
    case _OVGA:
        mnuModes = &mnuModesT[0];          /* Turn on Olivetti mode */
        aModes = &aModesT[0];
        case _VGA:
            iMode = _VRES16COLOR;
            break;
        case _MCGA:
            iMode = _MRES256COLOR;
            break;
        case _MDPA:
            default:
                puts("No graphics mode available.\n");
                return TRUE;
            }
switch( vc.mode )
{
    case _TEXTBW80:
    case _TEXTBW40:
    case _TEXTMONO:
    case _ERESNOCOLOR:
    case _HERCMONO:
        printf("Color Graphics Required to run Program\n");
        exit(0);
    default:
        break;
    }
}
vcj=f[VJC];
vsj=f[VJS];
je=f[CJE];
jc=f[CJC];
js=f[CJS];

/* Find current mode in mode array. */
for( iModesCur = 0; aModes[iModesCur] != iMode;
    iModesCur++)
{

while( TRUE )
{
    /* Set text mode and optionally clear the screen to blue. */
    if(fFirstTime)
        { setvideomode(_DEFAULTMODE);
        setbkcolor((long)JTRED);
        _clearscreen(_GCLEARSCREEN);
        /* Select from menu. */
        MouseInit();
        SetPtrVis(HIDE);}  
iMainCur=0;
    iMainCur = Menu( rowMid, colMid, mnuMain, iMainCur );

    /* Branch to menu choice. */
    switch( iMainCur )
    {
    case GUM:
        setvid();
        MouseInit();
        SetPtrVis(HIDE);
        Gummel();
        fFirstTime=TRUE;
        break;
    case IC:
        setvid();
        MouseInit();
        SetPtrVis(HIDE);
        Icvce();
        fFirstTime=TRUE;
        break;
    case CAP:
        Capacitor();
        fFirstTime=TRUE;
        break;
    case FT:
        setvid();
        break;
    
} 
}
MouseInit();
SetPtrVis(HIDE);
Ft();
fFirstTime=TRUE;
break;
case INCM:
kind=3;
Incl("nothing");
/*setvid();*/
fFirstTime=FALSE;
break;
case SAVE:
Save();/*setvid();*/
fFirstTime=FALSE;
break;
case TEMPER:
Temper();/*setvid();*/
fFirstTime=FALSE;
break;
case QUIT:
_unregisterfonts();
_setvideomode(_DEFAULTMODE);
return FALSE;
}
}
}

void setvid()
{
_setvideomode(iMode);
_displaycursor(_GCURSOROFF);
_getvideoconfig(&vc);
}

void write_text(short r, short c, char *txt)
{
_settextposition(r,c);
_outtext(txt);
}

void write_ftext(short x, short y, short i, short j, unsigned char *txt)
{
_moveto(x,y);
_setgtextvector(i,j);
_outgtext(txt);
}

/* Gummel Poon modelling segment */
void Gummel()

{
int xwidth,yheight,cols,rows,halfx,halfy,jtencount,icount;
double xmin1,xmax1,ymin1,ymax1,vstart,vstep,vstop;
double xmin2,xmax2,ymin2,ymax2,beta[2][100],ic[100];
double ii[2][100],v[100],val;
char lab[4],temp[20],temp2[20];
int i,j,k,rowMid,colMid,tip;
static int vidflag;
int chose=0;
int new=0,vbc=0,olwhich=1;
short linecol=10,pixcol=12;
struct _fontinfo fi;

kind=1; /* tells various routines call comes from gummel */

_getvideoconfig( &vc );
rowMid = vc.numtextrows / 2;
colMid = vc.numtextcols / 2;
xwidth=vc.numxpixels;
yheight=vc.numypixels;
halfx=xwidth/2;
halfy=7*yheight/8;
cols=vc.numtextcols;
rows=vc.numtextrows;
vstart=0.3;
vstop=1.0;
vstep=0.05;
start:icount=-1;
vidflag=FALSE;
for(vbe=vstart;vbe<vstop+vstep;vbe+=vstep)
{
if(vbc==0)
    vce=vbe;
    Spice();
    ii[0][++icount]=log10(bi);
    ii[1][icount]=log10(ci);
    beta[0][icount]=ci/bi;
    ic[icount]=ii[1][icount];
    v[icount]=type*vbe;
}
time=0;
if(vbc==0)
    vce=0.0;
start2:
    if(new==0)
    {xmin1=1E21;
     xmax1=-1E21;
     ymin1=xmin1;
     ymax1=xmax1;
     for(j=0;j<=1;j++)
     for(i=0;i<=icount;i++)
    }
{if(ii[j][i]<ymin1) ymin1=ii[j][i];
if(ii[j][i]>ymax1) ymax1=ii[j][i];
if(j==1)
   {if(v[i]<xmin1) xmin1=v[i];
    if(v[i]>xmax1) xmax1=v[i];
   }
}
newminmax(&xmaxl,&xminl);
newminmax(&ymaxl,&yminl);  
_clearscreen(_GCLEARSCREEN);

If(iwhich==1)
{write_ftext(10,2*halfy/3,0,1,"Ic and Ib (Amps)");
 write_ftext(xwidth/5,27*yheight/30,1,0,"Vbe (Volts)");
 write_ftext(xwidth/5,25,1,0,"Ic, Ib vs Vbe");
 if(vbc==0)
    write_ftext(xwidth/2-_getgtextextent("Vce=Vbe")/2,yheight-15,1,0,"Vce=Vbe");
 else
    { sprintf(temp,"Vce=%6.3f",vce);
      write_ftext(xwidth/2-_getgtextextent(temp)/2,yheight-15,1,0,temp);}
}
else
{write_ftext(10,2*halfy/3,0,1,"Ie and Ib (Amps)");
 write_ftext(xwidth/5,27*yheight/30,1,0,"Vbc (Volts)");
 write_ftext(xwidth/5,25,1,0,"Ie, Ib vs Vbc");
 if(vbc==0)
    write_ftext(xwidth/2-40,yheight-15,1,0,"Vec=Vbc");
 else
    { sprintf(temp,"Vce=%6.3f",vce);
      write_ftext(xwidth/2-40,yheight-15,1,0,temp);}
}
write_ftext(1,yheight-15,1,0,"Up-Down Arrow or Enter");
sprintff(temp,"Temp=%5.1f C",te-273.0);
write_ftext(xwidth-120,yheight-15,1,0,temp);
sprintff(lab,"%3.1f",xmin1);
write_ftext(40,27*yheight/30,1,0,lab);
sprintff(lab,"%3.1f",xmax1);
write_ftext(halfx-30,27*yheight/30,1,0,lab);
write_ftext(1,26*yheight/30,1,0,"10");
write_ftext(1,55,1,0,"10");
write_ftext(xwidth-25,halfy/2+40,0,1,"Beta");
write_ftext(2*xwidth/3,25,1,0,"Beta vs Ic");
if(iwhich==-1)
    write_ftext(2*xwidth/3,27*yheight/30,1,0,"Ie (Amps)");
 else
    write_ftext(2*xwidth/3,27*yheight/30,1,0,"Ic (Amps)"};
xmin2=1E21;
xmax2=-1E21;
ymin2=xmin2;
ymax2=xmax2;
for(i=0;i<=icount;i++)
{if(ic[i]<xmin2) xmin2=ic[i];
 if(ic[i]>xmax2) xmax2=ic[i];
 if(beta[0][i]<ymin2) ymin2=beta[0][i];
 if(beta[0][i]>ymax2) ymax2=beta[0][i];
}
newminmax(&xmax2,&xmin2);
newminmax(&ymax2,&ymin2);
if(ymax2-ymin2>1.0)
 sprintf(lab,"%4.0f",ymin2);
else
 sprintf(lab,"%5.1f",ymin2);
write_ftext(xwidth-36,25*yheight/29,1,0,lab);
if(ymax2-ymin2>1.0)
 sprintf(lab,"%4.0f",ymax2);
else
 sprintf(lab,"%5.1f",ymax2);
write_ftext(xwidth-36,40,1,0,lab);
write_ftext(halfx+10,27*yheight/30,1,0,"10");
write_ftext(xwidth-65,27*yheight/30,1,0,"10");
if(_setfont("t'tms rmn'hl0w5b")<0)
 {_outtext("can't set font");
     getchar();}
if(_getfontinfo(&fi))
 { _outtext("Can't get font info");
     exit(1);
 }
 sprintf(lab,"%3.2d",(int)ymin1);
write_ftext(14,27*yheight/32,1,0,lab);
 sprintf(lab,"%3.2d",(int)ymax1);
write_ftext(14,45,1,0,lab);
 sprintf(lab,"%3.2d",(int)xmin2);
write_ftext(halfx+25,halfy+5,1,0,lab);
 sprintf(lab,"%3.2d",(int)xmax2);
write_ftext(xwidth-50,halfy+5,1,0,lab);
if(_setfont("t'tms rmn'h15w10b")<0)
 {_outtext("can't set font");
     getchar();}
if(_getfontinfo(&fi))
 { _outtext("Can't get font info");
     exit(1);
 }
if(chose != 0)
{if(strlen(sp[chose-1])==3)
```c
    sprintf(temp,"%3s=%+10.2e",sp[chose-1],f[chose]);
    else
        sprintf(temp,"%2s=%+10.2e",sp[chose-1],f[chose]);
    _settextwindow(1,1,rows,cols);
    _setviewport(0,0,xwidth-1,yheight-1);
    _setwindow(TRUE,0.0,0.0,(double)(xwidth-1),(double)(yheight-1));
    write_text(rows-1,cols/2-4,"               ");

write_ftext(xwidth/2-_getgtextextent(temp)/2,yheight-32,1,0,temp);
    }
    _settextwindow(1,1,rows,cols/2);
    _setviewport(40,45,halfx-10,halfy);
    _setwindow(TRUE,xmin1,ymin1,xmax1,ymax1);
    if(new==0)
    { _setcolor(1);
        _rectangle_w(_GBORDER,xmin1,ymax1,xmax1,ymin1);
        _settextcolor(11);
        _setcolor(2);
        if(gumdata)
            plotpix(2,idata,vdata,xmax1-xmin1,ymax1-ymin1,pixcol,0);
    }

plot(2,icount,ii,v,xmin1,xmax1,ymin1,ymax1,new,linecol,_GXOR,0,1);
    _settextwindow(1,cols/2+1,rows,cols);
    /* 20-45 */
    _setviewport(halfx+10,45,xwidth-40,halfy);
    _setwindow(TRUE,xmin2,ymin2,xmax2,ymax2);
    if(new==0)
    { _setcolor(1);
        _rectangle_w(_GBORDER,xmin2,ymax2,xmax2,ymin2);
        if(gumdata)
            plotpix(1,idata,icdata,xmax2-xmin2,ymax2-ymin2,pixcol,2);
    }

plot(1,icount,beta,ic,xmin2,xmax2,ymin2,ymax2,new,linecol,_GXOR,1,0);
    while( TRUE )
    { switch( GetKey(CLEAR_WAIT))
        { case ENTER:
            if(type==l)
                mnuchoice[0]="NPN";
            else
                mnuchoice[0]="PNP";
            if(iwhich==1)
            
            if(iwhich==1)
            
            if(iwhich==1)
            
            if(iwhich==1) 64
```
mnuchoice[1]="Forward";
else
    mnuchoice[1]="Inverse";
if(vbc==0)
    strcpy(temp,"Vce=Vbe");
else
    sprintf(temp,"Vce=%6.3f",vce);
    mnuchoice[2]=strdup(temp);
    sprintf(temp,"Vbestart=%6.3f",vstart);
    mnuchoice[3]=strdup(temp);
    sprintf(temp,"Vbestop=%6.3f",vstop);
    mnuchoice[4]=strdup(temp);
    sprintf(temp,"Vbestep=%6.3f",vstep);
    mnuchoice[5]=strdup(temp);
    mnuchoice[6]="Done";
if(vbc==0)
    strcpy(temp,"Vec=Vbe");
else
    sprintf(temp,"Vec=%6.3f",vce);
    mnuchoice[7]=strdup(temp);
    sprintf(temp,"Vbcstart=%6.3f",vstart);
    mnuchoice[8]=strdup(temp);
    sprintf(temp,"Vbcstop=%6.3f",vstop);
    mnuchoice[9]=strdup(temp);
    sprintf(temp,"Vbcstep=%6.3f",vstep);
    mnuchoice[10]=strdup(temp);
    if(iwhich==-1)
        for(i=2;i<6;i++)
            {strcpy(temp,mnuchoice[i]);
             mnuchoice[i]=mnuchoice[i+5];
             mnuchoice[i+5]=strdup(temp);}
    getgo(7,&chose,&tip);
    switch(tip)
    { case 0:
        /* parameter chosen */
            new=1;
            break;
    case 1:
        /* replot chosen */
            new=0;
            goto start2;
            break;
    case 2:
        /* data chosen */
            _settextwindow(1,1,rows,cols/2);
            _setviewport(40,45,halfx-10,halfy);
            _setwindow(TRUE,xmin1,ymin1,xmax1,ymax1);
        plotpix(2,idata,vdata,xmax1-xmin1,ymax1-ymin1,pixcol,0);
plotpix(1, idata, icdata, xmax2-xmin2, ymax2-ymin2, pixcol, 2);
new=1;
brack;
case 3:
  /* increment chosen */
  if(incr<=0) incr=.1;
  if(incr>=1) incr=.9;
  new=1;
brack;
case 4:
  if(vbc==0)
    vce=0.0;
  if(strcmpi(oltype, mnuchoice[0])!=0)
  {strcpy(oltype, mnuchoice[0]);
    vidflag=TRUE;}
  if(iwhich!=olwhich)
  {olwhich=iwhich;
    vidflag=TRUE;}
  for(i=2; i<6; i++)
  {j=2;
    strcpy(temp, mnuchoice[i]);
    while((temp[j]!="\""&&(j<strlen(temp)))
      j++;
    strncpy(temp2, temp+j+1, 20);
    if(i==2 && strlen(temp2)!=0)
    {if(strlwr(temp2)=="vbe" || strlwr(temp2)=="vbc")
      val=0.0;
    else
      val=atof(temp2);
    if(val!=vce)
      {vce=val;
        vidflag=TRUE;
        if(vce==0.0)
          vbc=0;
        else
          vbc=1;
      }
    }
    if(i==3 && (val=atof(temp2))!=0 && vstart!=val)
    {vstart=val;
      vidflag=TRUE;}
    if(i==4 && (val=atof(temp2))!=0 && vstop!=val)
    {vstop=val;
      vidflag=TRUE;}
    if(i==5 && (val=atof(temp2))!=0 && vstep!=val)
if (vidflag)
  {new=0;
   setvid();
   goto start;}
break;
case 5:
   iwhich=1;
   olwhich=1;
   return;
   break;
}
break;
case U_DN:
   f[chose]=f[chose]-f[chose]*incr;
   new=1;
   /* 20-45 */
   _setviewport(halfx+10,45,xwidth-40,halfy);
   _setwindow(TRUE,xmin2,ymin2,xmax2,ymax2);
plot(1,icount,beta,ic,xmin2,xmax2,ymin2,ymax2,new,linecol,_G XOR,1,0);
   /* 20-45*/
   _setviewport(40,45,halfx-10,halfy);
   _setwindow(TRUE,xmin1,ymin1,xmax1,ymax1);
plot(2,icount,ii,v,xmin1,xmax1,ymin1,ymax1,new,linecol,_GXOR ,0,1);
   goto start;
break;
case U_UP:
   f[chose]=f[chose]+f[chose]*incr;
   new=1;
   /*20-45*/
   _setviewport(halfx+10,45,xwidth-40,halfy);
   _setwindow(TRUE,xmin2,ymin2,xmax2,ymax2);
plot(1,icount,beta,ic,xmin2,xmax2,ymin2,ymax2,new,linecol,_G XOR,1,0);
   /*20-45*/
   _setviewport(40,45,halfx-10,halfy);
   _setwindow(TRUE,xmin1,ymin1,xmax1,ymax1);
plot(2,icount,ii,v,xmin1,xmax1,ymin1,ymax1,new,linecol,_GXOR ,0,1);
   goto start;
break;
/* Ic-Vce modelling segment */
void Icvce()
{
    int xwidth, yheight, cols, rows, halfx, halfy, jtencount, icount;
    double xmin1, xmax1, ymin1, ymax1, vstart, vstep, vstop;
    double ii[3][100], v[100], istart=50E-6;
    double istep=50E-6, ib[5], val;
    double ymin2, ymax2;
    char lab[4], temp[20], temp2[20];
    int i, j, k, rowMid, colMid, tip;
    static int vidflag;
    int chose=0, olwhich=1;
    int new=0, numi, num;
    short linecol=10, pixcol=12;

    kind = 2; /* tells rest of program call comes from IC-Vce */
    _getvideoconfig( &vc );
    rowMid = vc.numtextrows / 2;
    colMid = vc.numtextcols / 2;
    xwidth=vc.numxpixels;
    yheight=vc.numypixels;
    halfx=xwidth;
    halfy=7*yheight/8;
    cols=vc.numtextcols;
    rows=vc.numtextrows;
    vstart=0.0;
    vstop=1.0;
    vstep=0.2;
    numi=2;
    start: num=numi;
    vidflag=FALSE;
    for(j=0; j<numi; j++)
    { icount=-1;
      bi=istart+(double)j*istep;
      ib[j]=bi;
      for(vce=vstart; vce<=vstop; vce+=vstep)
      { Spice();
        ii[j][++icount]=type*ci;
        v[icount]=type*vce;
      }
    }

    time=0;
    start2: if(new==0)
    { xmin1=1E21;
    }
xmax1=-1E21;
ymin1=xmin1;
ymax1=xmax1;
for(j=0;j<numi;j++)
  for(i=0;i<=icount;i++)
    { if(ii[j][i]<ymin1) ymin1=ii[j][i];
      if(ii[j][i]>ymax1) ymax1=ii[j][i];
      if(j==0)
        { if(v[i]<xmin1) xmin1=v[i];
          if(v[i]>xmax1) xmax1=v[i];
        }
    }
newminmax(&xmax1,&xmin1);
newminmax(&ymax1,&ymin1);
 clearance( _GCLEARSCREEN );
if(iwhich==1)
  { write_ftext(xwidth/2-70,25,1,0,"Collector Current vs Vce");
    write_ftext(xwidth/2-25,27*yheight/30,1,0,"Vce (Volts)"); }
else
  { write_ftext(xwidth/2-70,25,1,0," Emitter Current vs Vce");
    write_ftext(xwidth/2-25,27*yheight/30,1,0,"Vce (Volts)"); }
if(type==-1)
  { val=ymax1;
    ymax1=ymin1;
    ymin1=val; }
  if(fabs(ymax1)<1.0e-9)
    { if(iwhich==1)
      strcpy(temp,"Ic pAmps");
        else
      strcpy(temp,"Ie pAmps");
        ymax2=ymax1*1.0e12;
        ymin2=ymin1*1.0e12;
        goto next; }
  else if(fabs(ymax1)<1.0e-6)
    { if(iwhich==1)
      strcpy(temp,"Ic nAmps");
        else
      strcpy(temp,"Ie nAmps");
        ymax2=ymax1*1.0e9;
        ymin2=ymin1*1.0e9;
        goto next; }
  else if(fabs(ymax1)<1.0e-3)
    { if(iwhich==1)
      strcpy(temp,"Ic uAmps");
        else
      continue;
    }
strcpy(temp,"Ie uAmps");
    ymax2=ymax1*1.0e6;
    ymin2=ymin1*1.0e6;
    goto next;
} else if(fabs(ymax1)<=1.0) {
    if(iwhich==1)
        strcpy(temp,"Ic mAmps");
    else
        strcpy(temp,"Ie mAmps");
    ymax2=ymax1*1.0e3;
    ymin2=ymin1*1.0e3;
}
next:write_ftext(65,yheight/2,0,1,temp);
write_ftext(1,yheight-15,1,0,"Up-Down Arrow or Enter");
sprintf(temp,"Ib=%8.1e",ib[0]);
for(j=1;j<numi;j++) {
    sprintf(temp2,"%8.1e",ib[j]);
    strcat(temp,temp2);
}
write_ftext(xwidth/2-_getgtextextent(temp)/2+10,yheight-15,1,0,temp);
sprintf(temp,"Temp=%5.1f C",te-273.0);
write_ftext(xwidth-120,yheight-15,1,0,temp);
sprintf(lab,"%3.1f",xmin1);
write_ftext(85,27*yheight/30,1,0,lab);
sprintf(lab,"%3.1f",xmax1);
if(type==-1) {
    val=ymax2;
    ymax2=ymin2;
    ymin2=val;
    val=ymax1;
    ymax1=ymin1;
    ymin1=val;
}
write_ftext(9*xwidth/10+5,27*yheight/30,1,0,lab);
sprintf(lab,"%5.1f",ymin2);
write_ftext(40,27*yheight/30-15,1,0,lab);
sprintf(lab,"%5.1f",ymax2);
write_ftext(40,40,1,0,lab);
}
_settextwindow(1,1,rows,cols);
_setviewport(90,45,halfx-40,halfy);
_setwindow(TRUE,xmin1,ymin1,xmax1,ymax1);
if(new==0) {
    _setcolor(9);
    _settextcolor(11);
    rectangle_w(_GBORDER,xmin1,ymin1,xmax1,ymax1);
    if(vcdata)
plotpix(vcecurve, idata, vdata, xmax1-xmin1, ymax1-ymin1, pixcol, 0);
}
num=numi;

plot(num, icount, ii, v, xmin1, xmax1, ymin1, ymax1, new, linecol, _GX OR, 0, 0);
if(chose != 0)
{
if(strlen(sp[chose-l])==3)
   sprintf(temp,"%3s=%+10.2e",sp[chose-1],f[chose]);
else
   sprintf(temp,"%2s=%+10.2e",sp[chose-1],f[chose]);
_settextwindow(1,1,rows,cols);
_setviewport(0,0,xwidth-1,yheight-1);
_setwindow(TRUE,0.0,0.0,(double)(xwidth-1),(double)(yheight-1));
write_text(rows-1,cols/2-5," "
write_ftext(xwidth/2-_getgtextextent(temp)/2+10,yheight-32,1,0,temp);
}
while( TRUE )
{
   switch( GetKey(CLEAR_WAIT))
   {
   case ENTER:
      if(type==1)
         mnuchoice[0]="NPN";
      else
         mnuchoice[0]="PNP";
      if(iwhich==1)
         mnuchoice[1]="Forward";
      else
         mnuchoice[1]="Inverse";
      sprintf(temp,"Vcstart=%5.3f",vstart);
      mnuchoice[2]=strdup(temp);
      sprintf(temp,"Vcstop=%5.3f",vstop);
      mnuchoice[3]=strdup(temp);
      sprintf(temp,"Vcstep=%5.3f",vstep);
      mnuchoice[4]=strdup(temp);
      sprintf(temp,"Istart=%10.2e",istart);
      mnuchoice[5]=strdup(temp);
      sprintf(temp,"Istep=%10.2e",istep);
      mnuchoice[6]=strdup(temp);
      sprintf(temp,"Isteps=%d",numi);
      mnuchoice[7]=strdup(temp);
      mnuchoice[8]="Done";
      sprintf(temp,"Vestart=%5.3f",vstart);
      mnuchoice[9]=strdup(temp);
      sprintf(temp,"Vestop=%5.3f",vstop);
      mnuchoice[10]=strdup(temp);
   }
sprintf(temp,"Vestep=%5.3f",vstep);
mnuchoice[11]=strdup(temp);
if(iwhich==-1)
  for(i=2;i<5;i++)
    {strcpy(temp,mnuchoice[i]);
      mnuchoice[i]=mnuchoice[i+7];
      mnuchoice[i+7]=strdup(temp);} 
getgo(9,&chose,&tip);
switch(tip)
 { case 0:
    /* parameter chosen */
    new=1;
    break;
  case 1:
    /* replot chosen */
    new=0;
    goto start2;
    break;
  case 2:
    /* data chosen */
    _settextwindow(1,1,rows,cols);
    _setviewport(90,45,halfx-40,halfy);
    _setwindow(TRUE,xminl,yminl,xmaxl,ymaxl);

    plotpix(vcecurve,idata,vdata,xmax1-xmin1,ymax1-ymin1,pixcol,0);
    new=1;
    break;
  case 3:
    /* increment chosen */
    if(incr<=0)incr=.1;
    if(incr>=1)incr=.9;
    new=1;
    break;
  case 4:
    if(strcmpi(oltype,mnuchoice[0])!="0")
      {strcpy(oltype,mnuchoice[0]);
       vidflag=TRUE;}
    if(iwhich!=olwhich)
      {olwhich=iwhich;
       vidflag=TRUE;}
    for(i=2;i<8;i++)
      {j=2;
       strcpy(temp,mnuchoice[i]);
       while((temp[j]!=' ')&&(j<strlen(temp)))
         j++;
       strncpy(temp2,temp+j+1,20);
       if(i==2 && (val=atof(temp2))!=0 && vstart!=val)
         {vstart=val;
vidflag=TRUE;
if(i==3 && (val=atof(temp2))!=0 && vstop!=val)
    {vstop=val;
     vidflag=TRUE;
    }
if(i==4 && (val=atof(temp2))!=0 && vstep!=val)
    {vstep=val;
     vidflag=TRUE;
    }
if(i==5 && (val=atof(temp2))!=0 && istart!=val)
    {istart=val;
     vidflag=TRUE;
    }
if(i==6 && (val=atof(temp2))!=0 && istep!=val)
    {istep=val;
     vidflag=TRUE;
    }
if(i==7 && (val=atoi(temp2))!=0 && numi!=val)
    {
        numi=val;
        vidflag=TRUE;
    }
}
num=numi;
if(vidflag)
    {new=0;
     setvid();
     goto start;
    }
break;
case 5:
iwhich=1;
oWhich=1;
return;
break;
break;
case U_DN:
f[chose]=f[chose]-f[chose]*incr;
    new=1;
    _setviewport(90,45,halfx-40,halfy);
    _setwindow(TRUE,xmin1,ymin1,xmax1,ymax1);
    num=numi;
plot(num,icount,ii,v,xmin1,xmax1,ymin1,ymax1,new,linecol,_GX OR,0,0);
    goto start;
    break;
case U_UP:
f[chose]=f[chose]+f[chose]*incr;
    new=1;
    _setviewport(90,45,halfx-40,halfy);
    _setwindow(TRUE,xmin1,ymin1,xmax1,ymax1);
    num=numi;
plot(num,icount,ii,v,xmin1,xmax1,ymin1,ymax1,new,linecol,_GX
OR, 0, 0);
    goto start;
    break;
}
}
return;

/* Ft modelling segment */
void Ft()
{
    int xwidth, yheight, cols, rows, halfx, halfy, jtencount, icount;
    double xmin1, xmax1, ymin1, ymax1, vstart=2.0, vstep=1.0, div;
    double istart=10.0e-3;
    char lab[4], temp[20], temp2[20];
    int i, j, k, rowMid, colMid, tip, vidflag;
    int chose=0, ival;
    int new=0, numi, num;
    short linecol=10, pixcol=12;
    struct _fontinfo fi;

    kind = 5; /* tells rest of program call comes from Ft */
    _getvideoconfig( &vc );
    rowMid = vc.numtextrows / 2;
    colMid = vc.numtextcols / 2;
    xwidth=vc.numxpixels;
    yheight=vc.numypixels;
    halfx=xwidth;
    halfy=7*yheight/8;
    cols=vc.numtextcols;
    rows=vc.numtextrows;
    numi=2;
    start:num=numi;
    vidflag=FALSE;
    for(j=0; j<numi; j++)
    {
        icount=-1;
        vce=vstart+(double)j*vstep;
        vcc[j]=vce;
        for(ic=log(istart); ic<=log(istop); ic+=(log(istop)-log(istart))/istep)
        {
            ci=exp(ic);
            Spice();
            ftc[j][++icount]=fti;
            ii[icount]=log10(ci);
        }
    }
    start2:if(new==0)
    {
        xmin1=1E21;
xmax1=-1E21;
ymin1=xmin1;
ymax1=xmax1;
for(j=0;j<numi;j++)
    for(i=0;i<=icount;i++)
    {
        if(ftc[j][i]<ymin1) ymin1=ftc[j][i];
        if(ftc[j][i]>ymax1) ymax1=ftc[j][i];
    }
    if(j==0)
    {
        if(ii[i]<xmin1) xmin1=ii[i];
        if(ii[i]>xmax1) xmax1=ii[i];
    }
if(ymax1>=1.0e8)
{div=1.0e9;
 strcpy(temp,"ft GHz");}
else
{div=1.0e6;
 strcpy(temp,"ft MHz");}
newminmax(&xmax1,&xmin1);
newminmax(&ymax1,&ymin1);
_clearscreen(_GCLEARSCREEN);
write_ftext(xwidth/2-70,25,1,0,"Ft vs Collector Current");
next:write_ftext(xwidth/2-25,27*yheight/30,1,0,"Ic (Amps)");
write_ftext(65,yheight/2,0,1,temp);
write_ftext(1,yheight-15,1,0,"Up-Down Arrow or Enter");
sprintf(temp,"Vce=%5.2f",vcc[0]);
for(j=1;j<numi;j++)
    { sprintf(temp2,"%5.2f",vcc[j]);
      strcat(temp,temp2);
    }
strcat(temp," Volts");
write_ftext(xwidth/2-_getgtextextent(temp)/2+10,yheight-15,1,0,temp);
sprintf(temp,"Temp=%5.1f C",te-273.0);
write_ftext(xwidth-120,yheight-15,1,0,temp);
write_ftext(80,27*yheight/30,1,0,"10");
write_ftext(9*xwidth/10+5,27*yheight/30,1,0,"10");
sprintf(lab,"%5.2f",ymin1/div);
write_ftext(40,27*yheight/30-15,1,0,lab);
sprintf(lab,"%5.2f",ymax1/div);
write_ftext(40,40,1,0,lab);
if(_setfont("t'tms rmn'h10w5b")<0)
{ _outtext("can't set font");
  getchar();}
if(_getfontinfo(&fi))
{ _outtext("Can't get font info");


exit(1);
}
sprintf(lab,"%3.2d",(int)xmin1);
write_ftext(94,27*yheight/30-10,1,0,lab);
sprintf(lab,"%3.2d",(int)xmax1);
write_ftext(9*xwidth/10+19,27*yheight/30-10,1,0,lab);
if(_setfont("t'tm's rmn'\h15\w10b")<0)
{_outtext("can't set font");
  getchar();}
if(_getfontinfo(&fi))
{ _outtext("Can't get font info");
  exit(1);
}

_settextwindow(1,1,rows,cols);
_setviewport(90,45,halfx-40,halfy);
_setwindow(TRUE,xmin1,ymin1,xmax1,ymax1);
if(new==0)
{  _setcolor(9);
  _settextcolor(11);
  rectangle_w(_GBORDER,xmin1,ymax1,xmax1,ymin1);
  if(ftdata)
    plotpix(ftcurve,idata,vdata,xmax1-xmin1,ymax1-ymin1,pixcol,0);
  num=numi;
}

plot(num,icount,ftc,ii,xmin1,xmax1,ymin1,ymax1,new,linecol,_GXOR,1,0);
if(chose != 0)
{  if(strlen(sp[chose-1])==3)
      sprintf(temp,"%3s=%+10.2e",sp[chose-1],f[chose]);
  else
      sprintf(temp,"%2s=%+10.2e",sp[chose-1],f[chose]);
  _settextwindow(1,1,rows,cols);
  _setviewport(0,0,xwidth-1,yheight-1);
  _setwindow(TRUE,0.0,0.0,(double)(xwidth-1),(double)(yheight-1));
  write_text(rows-1,cols/2-6,"");
  write_ftext(xwidth/2-_getgtextextent(temp)/2+10,yheight-32,1,0,temp);}
sprintf(temp,"Icstop=%8.1e",istop);
mnuchoice[1]=strdup(temp);
sprintf(temp,"Icsteps=%d",(int)istep);
mnuchoice[2]=strdup(temp);
sprintf(temp,"Vcstart=%5.3f",vstart);
mnuchoice[3]=strdup(temp);
sprintf(temp,"Vcstep=%5.3f",vstep);
mnuchoice[4]=strdup(temp);
sprintf(temp,"Vcsteps=%d",numi);
mnuchoice[5]=strdup(temp);
mnuchoice[6]="Done";
getgo(7,&chose,&tip);
switch(tip)
{
case 0:
    /* parameter chosen */
    new=1;
    break;
case 1:
    /* replot chosen */
    new=0;
    goto start2;
    break;
case 2:
    /* data chosen */
    _settextwindow(1,1,rows,cols);
    _setviewport(90,45,halfx-40,halfy);
    _setwindow(TRUE,xmin1,ymin1,xmax1,ymax1);
    plotpix(ftcurve,idata,vdata,xmax1-xmin1,ymax1-ymin1,pixcol,0);
    new=1;
    break;
case 3:
    /* increment chosen */
    if(incr<=0)incr=.1;
    if(incr>=1)incr=.9;
    new=1;
    break;
case 4:
    for(i=0;i<6;i++)
    {
        j=2;
        strncpy(temp,mnuchoice[i]);
        while((temp[j]!='')&&(j<strlen(temp)))
            j++;
        strncpy(temp2,temp+j+1,20);
        if(i==0 && (val=atof(temp2))!=0 && istart!=val)
            {istart=val;
             vidflag=TRUE;}
        if(i==1 && (val=atof(temp2))!=0 && istop!=val)
            {istop=val;
    vidflag=TRUE;
if(i==2 && (ival=atoi(temp2))!=0 && (int)istep!=ival)
    {istep=(double)ival;
     vidflag=TRUE;}
if(i==3 && (val=atof(temp2))!=0 && vstart!=val)
    {vstart=val;
     vidflag=TRUE;}
if(i==4 && (val=atof(temp2))!=0 && vstep!=val)
    {vstep=val;
     vidflag=TRUE;}
if(i==5 && (val=atoi(temp2))!=0 && numi!=val)
    {numi=val;
     vidflag=TRUE;}
}
num=numi;
if(vidflag)
    {new=0;
     setvid();
     goto start;}
break;
case 5:
    return;
break;
}
break;
case U_DN:
    f[chose]=f[chose]-f[chose]*incr;
    new=1;
    _setviewport(90,45,halfx-40,halfy);
    _setwindow(TRUE,xmin1,ymin1,xmax1,ymax1);
    num=numi;
plot(num,icount,ftc,ii,xmin1,xmax1,ymin1,ymax1,new,linecol,_,
    GXOR,1,0);
    goto start;
break;
case U_UP:
    f[chose]=f[chose]+f[chose]*incr;
    new=1;
    _setviewport(90,45,halfx-40,halfy);
    _setwindow(TRUE,xmin1,ymin1,xmax1,ymax1);
    num=numi;
plot(num,icount,ftc,ii,xmin1,xmax1,ymin1,ymax1,new,linecol,_,
    GXOR,1,0);
    goto start;
break;
}
return;
}
void Spice()
{
    /* iwhich = 1 - forward mode; iwhich = -1 inverse mode */
    /* kind = 1 gummel; kind = 2 ic vs vce; kind = 5 Ft */

    double vcei,vbci,vbei,vbx,vtne,vte,vtc,evbe,cbe,evben;
    double cben,evbc,cbc,evbcn,cbcn,q1,q12,q13,q2,sqarg,vcs;
    double qb,cex,ar1,ar2,ar3,rb,gmin,ar,vtnc,temp,sarg,f3,cbef2;
    double gbe,gben,gbc,gbcn,dqbdve,dqbdvc,gex,gx,det,argtf;
    double gpi,gmu,gm,go,vbeo,vbco,fcpc,cbcf2,cbx,cbxf2,fcpe;
    double ib,ic,cb,f1,f2,al1,capbe,capbc,capbx,capcs;
    double a12,a21,a22,delf1,delf2;
    /*double gbe,qbc,qbx,qcs;*/
    int i,cnv=0;

gmin=0.0;
/*1.0E-14;*/
switch(kind)
{
case 1:
    /* gummel poon */
    if(time==1)
    {vbci=vbc0;
        vbei=vbe0;
    }else if(iwhich==-1)
    {vbci=vbe;
        vbei=vbci-vce;
    }else
    {vbei=vbe;
        vbci=vbei-vce;
    }
break;
case 2:
    /* Ic Vce */
    if(vce==0.0)
    {vbci=0.0;
        vbei=0.0;
    }else if(iwhich==1)
    {vbei=.7;
        vbci=vbei-vce;
    }else
    {vbci=.7;
        vbei=vbc0-vce;
    }
    ib=bi;
    break;
case 5:
    ic=ci;
    vbei=.7;
vbci=vbei-vce;
break;
}
start:vte=f[7]*vt;
vtc=f[13]*vt;
vtne=f[3]*vt;
vcrit=vt*log(vt/(sqrt(2.0)*f[1]));
if(vbei<= -5.0*vtne) goto nex320;
evbe=exp(vbei/vtne);
cbe=f[1]*(evbe-1.0)+gmin*vbei;
gbe=f[1]*evbe/vtne+gmin;
evben=exp(vbei/vte);
cben=f[6]*(evben-1.0);
gben=f[6]*evben/vte;
goto nex350;

nex320:gbe=-f[1]/vbei+gmin;
cbe=gbe*vbei;
gben=-f[6]/vbei;
cben=gben*vbei;

nex350:vtnc=vt*f[9];
if(vbci<-5.0*vtnc) goto nex370;
evbc=exp(vbci/vtnc);
cbc=f[1]*(evbc-1.0)+gmin*vbci;
gbc=f[1]*evbc/vtnc+gmin;
evbcn=exp(vbci/vtc);
cbcn=f[12]*(evbcn-1.0);
gbcn=f[12]*evbcn/vtc;
goto nex400;

nex370:q1=1.0/(1.0-vbci/f[4]-vbei/f[10]);
ar=(1.0+4.0*q2>0.0) ? 1.0+4.0*q2 : 0.0;
sqarg=1.0;
if(ar!=0) sqarg=sqrt(ar);
qb=q1*(1.0+sqarg)/2.0;
dqbdve=q1*(qb/f[10]+(gbe/sqarg)/f[5]);
dqbdvc=q1*(qb/f[4]+(gbc/sqarg)/f[11]);
cex=cbe;
gex=gbe;

cex=cbc;

cx=1.0/rb;
gpi=gbe/f[2]+gben;
gmu=gbc/f[8]+gbcn;
go=(gbc+(cex-cbc)*dqbdvc/qb)/qb;
gm=(gex-(cex-cbc)*dqbdve/qb)/qb-go;
vbeo=vbei;
vbcovbci;
cnv=cnv+1;
if(cnv>50)
goto next1;
switch(kind)
{case 1:
  delfl=.005;
  if(iwhich==-1)
  {
    f1=vbe-(cb*rb-ci*f[18]+vbco);
    f2=vce-(vbe-vbei-(cb+ci)*f[17]-cb*rb);
  }else
  {
    f1=vbe-(cb*(rb+f[17])+ci*f[17]+vbeo);
    f2=vce-(ci*f[18]+(ci+cb)*f[17]-vbco+vbeo);
  }
bREAK;
  case 2:
  delfl=.01*ib;
  f1=ib-cb;
  if(iwhich==-1)
  {
    f2=vce-(vbci-vbei-(cb+ci)*f[17]-ci*f[18]);
  }else
  {
    f2=vce-(ci*f[18]+(ci+cb)*f[17]-vbco+vbeo);
  }
bREAK;
case 5:
  f1=ic-ci;
  delfl=.01*ic;
  f2=vce-(ci*f[18]+(ci+cb)*f[17]-vbco+vbeo);
  break;
}
if(fabs(f1)<=delf1 && fabs(f2)<.005)
goto next1;
det=-(a11*a22-a12*a21);
vbei=vbeo+(a22*f1-a12*f2)/det;
pnjlims(&vbei,&vbeo);
vbcovbco=(a11*f2-a21*f1)/det;
pnjlims(&vbci,&vbco);
if(fabs(vbei-vbeo)+fabs(vbci-vbeo)<.01)
goto next1;
goto start;
nex1: if(kind==5)
    { /* begin ac */
        vbe=cb*rb+vbei+(cb+ci)*f[RE];
        vcs=vce-ci*f[RC];
        vbx=vbe-vcs;
        vcei=vbei-vbci;
        cbx=f[CJC]-f[CJC]*f[XCJ];
        if(f[TF]==0.0) goto nex505;
        if(vbei<0.0) goto nex505;
        argtf=0.0;
        ar2=0.0;
        ar3=0.0;
        if(f[XTF]==0.0) goto nex504;
        argtf=f[XTF];
        if(f[VTF]!=0.0) argtf=argtf*exp(vbci*f[VTF]);
        argtf=argtf*temp*temp;
        ar2=argtf*(3.0-temp-temp);
    }

nex503: ar3=cbe*argtf*f[VTF];
    goto nex520;

nex510: fl=f[VJE]*(1.0-pow(1.0-f[FC],1.0-f[MJE]))/(1.0-f[MJE]);
    f2=pow(1.0-f[FC],1.0+f[MJE]);
    f3=1.0-f[FC]*(1.0+f[MJE]);
    cbef2=f[CJE]/f2;
    /*qbe=f[TF]*cbe+f[VJE]*f[CJE]*(1.0-ar*sarg)/(1.0-f[MJE]);/*
    capbe=f[TF]*gbe+f[CJE]*sarg;
    goto nex520;

nex520: fpc=f[FC]*f[VJC];
    fl=f[VJC]*(1.0-pow(1.0-f[FC],1.0-f[MJC]))/(1.0-f[MJC]);
    f2=pow(1.0-f[FC],1.0+f[MJC]);
    f3=1.0-f[FC]*(1.0+f[MJC]);
    if(vbci>=fcpc) goto nex530;
    ar=1.0-vbci/f[VJC];
    sarg=(-f[MJC]*log(ar));

/*qbc=f[TR]*cbc+f[VJC]*f[CJS]*f[XCJ]*(1.0-ar*sarg)/(1.0-f[MJC]);/*/
void pnjlims(double *vnew, double *vold)
{
    double vlim, delv, a;
    delv = vnew - *vold;
    
    
}
if(*vnew<=vcrit) 
  return;
vlim=2.0*vt;
if(fabs(delv)<=vlim) 
  return;
if(*vold<=0.0) 
  {*vnew=vt*log(*vnew/vt);
  return;}
a=1.0+delv/vt;
if(a<=0.0) 
  {*vnew=vcrit;
  return;}
*vnew=*vold+vt*log(a);
}/* Save Model segment */
void Save()
{
  FILE *fptr;
  char ans, filename[20], model[20];
  int i;

  some:write_text(18,30,"Enter Output file name: ");
  scanf("%s",filename);
  if((fptr=fopen(filename, "w"))==NULL)
  {write_text(19,30,"Can't open file, try another ? (Y/N): ");
   ans=getch();
   if(ans=='y'||ans=='Y')
   { write_text(18,30," ");
     write_text(19,30," ");
   }
   goto some;
  }
  else
  { write_text(19,30," ");
    write_text(18,30," ");
    return;
  }
  write_text(19,30,"Enter Model Name:");
  scanf("%s",model);
  write_text(19,30," ");
  write_text(18,30," ");
  fprintf(fptr,".model %s %s(%s=%10.2e %s=%10.2e %s=%10.2e
",model,
    oltype,sp[0],f[1],sp[1],f[2],sp[2],f[3]);
for(i=0;i<8;i++)
    fprintf(fptr,"+%s=%10.2e %s=%10.2e %s=%10.2e
",sp[3+i*4],f[4+i*4],sp[4+i*4],f[5+i*4],sp[5+i*4],f[6+i*4],
    sp[6+i*4],f[7+i*4]);
    fprintf(fptr,"+%s=%10.2e %s=%10.2e %s=%10.2e\n",sp[35],f[36],
    sp[36],f[37],sp[37],f[38]);
fclose(fptr);
return;

#include "File segment"*/
void Incl(char filename[20])
{
    FILE *fptr;
    int colnum,i,j=0,k,len,cnt,c,word=0,wordflag=0,count=0;
    int n=0,1;
    char ans,dum1[20],dum2[20],temp[20];
    char dum3[81];

    some:
    if(kind==3||kind==4)
    {
        _settextposition(19,30);
        _outtext("Enter Input file name: ");
        scanf("%s",filename);
        if((fptr=fopen(filename,"r"))==NULL)
        {
            _settextposition(20,30);
            _outtext("Can't open file, try another ? (Y/N): ");
            ans=getch();
            write_text(20,30," ");
            if(ans=='y'||ans=='Y')
            {
                write_text(19,30," ");
                goto some;
            }
            else
            {
                fart=1;
                return;
            }
        }
    }
    if(kind==3)
    write_text(19,30," ");
    while(1)
    {if(fgets(dum3, 80, fptr )==NULL)
    break;
n++;}
rewind(fptr);
for(i=0;i<n;i++)
{
cnt=0;
strcpy(dum1,"");
/* skip over the first character if this is a model read
since it may be a continuation character */
if(kind==3)
{
if(i==0)
    while((c=fgetc(fptr))!='(');
else
    c=fgetc(fptr);
while((c=fgetc(fptr))!='\n')
{if(isspace(c)||c=='='||c==')'||!isgraph(c))
    word=FALSE;
else
    {word=TRUE;
     wordflag=TRUE;
     sprintf(dum2,"%c",c);
     strcat(dum1,dum2);
    }
if(wordflag==TRUE && word==FALSE)
{
    wordflag=FALSE;
    if(kind==3)
    {
      count++;
      if(count==2)
      {
        for(l=0;l<37;l++)
        {if(strcmpi(temp,sp[l])==0)
            {f[l+1]=atof(dum1);
             goto nex10;}
        }nex10:count=0;
      }
      else
      {
       if(cnt==0)
       {if(kind==1||kind==2||kind==4||kind==5)
        vdata[i]=atof(dum1);
        }
       else
       {if(kind==1)
        {if(cnt==3)
         idata[0][i]=atof(dum1);
else
    idata[cnt-1][i]=atof(dum1);
} else if(kind==2||kind==5)
    idata[cnt-1][i]=atof(dum1);
else if(kind==4)
    idata[0][i]=atof(dum1);
}
strcpy(dum1,"");
cnt++;}
}
if(word==TRUE)
{wordflag=FALSE;
i=kind==1||kind==2||kind==5)
    idata[cnt-1][i]=atof(dum1);
else if(kind==4)
    idata[0][i]=atof(dum1);
else if(kind==3)
{ count++;
    if(count==2)
    {
        for(l=0;l<37;l++)
            if(strcmpi(temp,sp[l])==0)
            {
                f[l+1]=atof(dum1);
                goto nexll;
            }
    }
    else
        strcpy(temp,dum1);
}
}
if(kind==2)
{ vcdatalTRUE;
    vcecount=n;
    vcecurve=cnt;
}
else if(kind==5)
{ ftdata=TRUE;
    ftcount=n;
    ftcurve=cnt;
    for(i=0;i<n;i++)
       vdata[i]=log10(vdata[i]);
}
else if(kind==1)
{ gumcount=n;
gumdata=TRUE;
if(cnt==2)
   for(i=0;i<n;i++)
      idata[2][i]=idata[1][i]/idata[0][i];
for(i=0;i<n;i++)
   { idata[0][i]=log10(idata[0][i]);
      idata[1][i]=log10(idata[1][i]);
      icdata[i]=idata[1][i];
   }
else if(kind==4)
   numv=n;
else if(kind==3)
   { se=f[ISE];
      sc=f[ISC];
      si=f[IS];
      bfo=f[BF];
      bro=f[BR];
      vej=f[VJE];
      vcj=f[VJC];
      vsj=f[VJS];
      je=f[CJE];
      jc=f[CJC];
      js=f[CJS];
      Temper();
      kind=0; }
fclose(fptr);
return;
}
void newminmax(double *max, double *min)
{
   double diff,ydiff;
   int jtencount=0;

   diff=fabs(*max-*min);
   ydiff=diff;
   if(diff<1.0)
      while(ydiff<1.0)
      { *max=*max*10.0;
        *min=*min*10.0;
        ydiff=ydiff*10.0;
        jtencount=jtencount-1;
      }
   if(diff>15)
      while(ydiff>15.0)
      { (*max)=(*max)/10.0;
        (*min)=(*min)/10.0;
        ydiff=ydiff/10.0;
        jtencount=jtencount+1;
      }
   *min=(double)floor(*min)*pow(10.0,(double)jtencount);
*max = (double) ceil(*max) * pow(10.0, (double) jtencount);
}

void getgo(int count, int *chose, int *icur)
{
    int i, j, k, xwidth, yheight, row, col, iprev;
    int inc;
    char temp[20];
    char huge *buffer, huge *box;
    long imsize, boxsize;
    unsigned key = 0;
    EVENT mevent;
    int fBtnDown;
    unsigned char *tempmnu[2];
    FILE *fp;

    _getvideoconfig(&vc);
    xwidth = vc.numx pixels;
    yheight = vc.numy pixels;
    row = vc.numtextrows;
    col = vc.numtextcols;
    _settextwindow(1, 1, row, col);
    _setviewport(0, 0, xwidth - 1, yheight - 1);

    _setwindow(TRUE, 0.0, 0.0, (double)(xwidth - 1), (double)(yheight - 1));
    imsize = _imagesize(0, 0, xwidth - 1, 20);
    boxsize = _imagesize(1, 1, 1 + (xwidth - 5)/6, 19);
    if((buffer = (char huge *) halloc(imsize, (size_t) 1)) == NULL)
        {_outtext("problems 1");
         exit(1);}
    if((box = (char huge *) halloc(boxsize, (size_t) 1)) == NULL)
        {_outtext("problems 3");
         exit(1);}
    _setcolor(3);
    _rectangle(_GFILLINTERIOR, 2, 2, 1 + (xwidth - 5)/6, 17);
    _getimage(1, 1, 2 + (xwidth - 5)/6, 18, box);
    _putimage(1, 1, box, _GXOR);
    _setcolor(15);
    _rectangle(_GFILLINTERIOR, 1, 1, xwidth - 2, 19);
    _setcolor(0);
    for(i = 0; i < 6; i++)
        {
            write_ftext(10 + i * xwidth / 6, 3, 1, 0, mnustuff[i]);
            _rectangle(_GBORDER, 2 + i * (xwidth - 5)/6, 2, 2 + (i + 1) * (xwidth - 5)/6, 18);
        }
    _getimage(0, 0, xwidth - 1, 20, buffer);
if (GetMouseEvent(&mevent))
    if (mevent.fsBtn & LEFT_DOWN);
SetPtrPos((xwidth-5)/12, 9);
SetPtrVis(SHOW);
fBtnDown=FALSE;
mevent.fsBtn=0;
while(TRUE)
{
    {switch(key)
        case U_LT:
            SetPtrVis(HIDE);
iprev=i;
i=(i>0)?iprev-1:iprev;
if(iprev!=i)
            {_putimage(1+iprev*(xwidth-4)/6, 2, box, _GXOR);
            write_ftext(10+iprev*xwidth/6, 3, 1, 0, mnustuff[iprev]);
            _putimage(1+i*(xwidth-4)/6, 2, box, _GXOR);
            write_ftext(10+i*xwidth/6, 3, 1, 0, mnustuff[i]);
            SetPtrPos((i+1)*(xwidth-5)/6-(xwidth-5)/12, 9);}
            SetPtrVis(SHOW);
            break;
        case U_RT:
            SetPtrVis(HIDE);
iprev=i;
i=(i<5)?iprev+1:iprev;
if(iprev!=i)
            {_putimage(1+iprev*(xwidth-4)/6, 2, box, _GXOR);
            write_ftext(10+iprev*xwidth/6, 3, 1, 0, mnustuff[iprev]);
            _putimage(1+i*(xwidth-4)/6, 2, box, _GXOR);
            write_ftext(10+i*xwidth/6, 3, 1, 0, mnustuff[i]);
            SetPtrPos((i+1)*(xwidth-5)/6-(xwidth-5)/12, 9);}
            SetPtrVis(SHOW);
            break;
        case ENTER:
            SetPtrVis(HIDE);
goto next;
            break;
    }
}
else if (GetMouseEvent(&mevent))
{
    if (mevent.y>18)
    {SetPtrPos(mevent.x, 18);
    mevent.y=18;}
    else if(mevent.y<2)
    {SetPtrPos(mevent.x, 2);
    mevent.y=2;}
if (mevent.x > xwidth-3) {
    SetPtrPos(xwidth-3, mevent.y);
    mevent.x = xwidth-3;
} else if (mevent.x < 2) {
    SetPtrPos(2, mevent.y);
    mevent.x = 2;
} if (mevent.fsBtn & LEFT_DOWN) {
    fBtnDown = TRUE;
    SetPtrVis(HIDE);
    iprev = i;
    i = 6 * mevent.x / (xwidth - 5);
    if (iprev != i) {
        _putimage(1 + iprev * (xwidth - 4) / 6, 2, box, _GXOR);
        write_ftext(10 + iprev * xwidth / 6, 3, 1, 0, mnustuff[iprev]);
        _putimage(1 + i * (xwidth - 4) / 6, 2, box, _GXOR);
        write_ftext(10 + i * xwidth / 6, 3, 1, 0, mnustuff[i]);
        SetPtrPos((i + 1) * (xwidth - 5) / 6 - (xwidth - 5) / 12, 9);
    } else
        continue;
} else
    continue;
} else
    continue;
} next: _setcolor(15);
switch(i) {
    case 0:
        *choose = 0;
        if (kind == 1 || kind == 2) {
            for (k = 0; k <= 20; k++)
                {sprintf(temp, "%3s=%8.1e", sp[k], f[k + 1]);
                 mnupar[k] = strdup(temp);
                } j = choice(0, 19, 21, mnupar, i);
        *choose = j + 1;
    } else if (kind == 5) {
        for (k = 21; k <= 36; k++)
            {sprintf(temp, "%3s=%8.1e", sp[k], f[k + 1]);
             mnupar[k - 21] = strdup(temp);
            } j = choice(0, 19, 16, mnupar, i);
        *choose = j + 22;
        _putimage(1 + i * (xwidth - 4) / 6, 2, box, _GXOR);
        _putimage(0, 0, buffer, _GXOR);
        hfree(buffer);
        hfree(box);
        *icur = 0;
        write_ftext(10 + i * xwidth / 6, 3, 1, 0, mnustuff[i]);
        return;
        break;
    case 1:
case 2:
    tempmnu[0]="File=Lotsofdata.dat"
    j = choice(165,19,1,tempmnu,i);
    strcpy(temp,tempmnu[0]+5);
    if((fptr=fopen(temp, "r"))==NULL) {
        *icur=3;
        _putimage(1+i*(xwidth-4)/6,2,box,_GXOR);
        write_ftext(10+i*xwidth/6,3,1,0,mnustuff[i]);
        _putimage(0,0,buffer,_GXOR);
        hfree(buffer);
        hfree(box);
        return;
    }
    _putimage(1+i*(xwidth-4)/6,2,box,_GXOR);
    write_ftext(10+i*xwidth/6,3,1,0,mnustuff[i]);
    _putimage(0,0,buffer,_GXOR);
    fclose(fptr);
    hfree(buffer);
    hfree(box);
    Incl(temp);
    *icur=2;
    return;
    break;
    case 3:
    sprintf(temp,"Incr=%4.2f",incr);
    tempmnu[0]=strdup(temp);
    j = choice(325,19,1,tempmnu,i);
    strcpy(temp,tempmnu[0]+5);
    incr=atof(temp);
    _putimage(1+i*(xwidth-4)/6,2,box,_GXOR);
    _putimage(0,0,buffer,_GXOR);
    hfree(buffer);
    hfree(box);
    *icur=3;
    write_ftext(10+i*xwidth/6,3,1,0,mnustuff[i]);
    return;
    break;
    case 4:
    j = choice( 400,19,count,mnuchoice,i);
    _putimage(0,0,buffer,_GXOR);
    _putimage(1+i*(xwidth-4)/6,2,box,_GXOR);
    hfree(buffer);
    hfree(box);
*icur=4;
write_ftext(10+i*xwidth/6,3,1,0,mnustuff[i]);
return;
break;
    case 5:
        _putimage(1+i*(xwidth-4)/6,2,box,_GXOR);
        _putimage(0,0,buffer,_GXOR);
        hfree(buffer);
        hfree(box);
        gumdata=FALSE;
        *icur=5;
        return;
        break;
    }

int choice( int x, int y, int numstr, unsigned char *mnu[20],int place)
{
    int i,j,len=0,iprev,icur,k,e,l,i2,add;
    int row,col,xwidth,yheight,neg,neg2,en;
    long imsize,boxsize;
    char _huge *plotch, _huge *box;
    char temp[20],temp2[20];
    unsigned ukey=0;
    EVENT mevent;
    int fBtnDown=FALSE;

    icur=0;
    _getvideoconfig( &vc );
    Row = vc.numtextrows;
    col = vc.numtextcols;
    xwidth=vc.numxpixels;
    yheight=vc.numypixels;
    for(i=0;i<numstr;i++)
        if(strlen(mnu[i])>len)
            len=strlen(mnu[i]);
    len=10*len;
    imsize= _imagesize(0,0,len+7,20*numstr+4);
    boxsize= _imagesize(0,0,len+5,20);
    if((plotch=(char _huge *)malloc (imsize,(size_t) 1))==NULL)
        {_outtext("problems 4");
            exit(1);}
    if((box=(char _huge *)malloc (boxsize,(size_t) 1))==NULL)
        {_outtext("problems 5");
            exit(1);}
    _getimage(x,y,x+len+7,y+20*numstr+4,plotch);
    _putimage(x,y,plotch,_GXOR);
    _setcolor(12);
    _rectangle(_FILLINTERIOR,x+2,y+2,x+len+3,y+20);
    _getimage(x+1,y+1,x+len+4,y+21,box);
again: _setcolor(15);
_rectangle(_GFILLINTERIOR,x+1,y+1,x+len+6,y+20*numstr+3);
_setcolor(0);
for(i=0;i<numstr;i++)
  {write_ftext(x+3,y+3+i*20,1,0,mnu[i]);
   _rectangle(_GBORDER,x+2,y+2+i*20,x+len+5,y+2+(i+1)*20);
  }
_putimage(x+2,y+2+icur*20,box,_GXOR);
write_ftext(x+3,y+3+icur*20,1,0,mnu[icur]);
SetPtrPos(x+(len+5)/2,y+11+icur*20);
SetPtrVis(SHOW);
if(GetMouseEvent(&mevent))
  if( mevent.fsBtn & LEFT_DOWN );
  fBtnDown=FALSE;
while(TRUE)
  {if(ukey=GetKey(NO_WAIT))
    {switch(ukey)
      { case U_UP:
          SetPtrVis(HIDE);
          iprev=icur;
          icur=(icur>0)?iprev-1:iprev;
          if(iprev!=icur)
            {_putimage(x+2,y+2+iprev*20,box,_GXOR);
             write_ftext(x+3,y+3+iprev*20,1,0,mnu[iprev]);
             _putimage(x+2,y+2+icur*20,box,_GXOR);
             write_ftext(x+3,y+3+icur*20,1,0,mnu[icur]);
             SetPtrPos(x+(len+5)/2,y+11+icur*20);}
          SetPtrVis(SHOW);
          break;
        case U_DN:
          SetPtrVis(HIDE);
          iprev=icur;
          icur=(icur<numstr-1)?iprev+1:iprev;
          if(iprev!=icur)
            {_putimage(x+2,y+2+iprev*20,box,_GXOR);
             write_ftext(x+3,y+3+iprev*20,1,0,mnu[iprev]);
             _putimage(x+2,y+2+icur*20,box,_GXOR);
             write_ftext(x+3,y+3+icur*20,1,0,mnu[icur]);
             SetPtrPos(x+(len+5)/2,y+11+icur*20);}
          SetPtrVis(SHOW);
          break;
        default:
          continue;
        case ENTER:
          SetPtrVis(HIDE);
          goto choose;
          break;
    }

}
else if(GetMouseEvent(&mevent))
{
    if(mevent.y>y+20*numstr+2)
        {SetPtrPos(mevent.x,y+20*numstr+2);
           mevent.y=y+20*numstr+2; }
    else if(mevent.y<y+2)
        {SetPtrPos(mevent.x,y+2);
           mevent.y=y+2; }
    if(mevent.x>x+len+5)
        {SetPtrPos(x+len+5,mevent.y);
           mevent.x=x+len+5; }
    else if(mevent.x<x+2)
        {SetPtrPos(x+2,mevent.y);
           mevent.x=x+2; }
    if( mevent.fsBtn & LEFT_DOWN )
    {fBtnDown=TRUE;
        SetPtrVis(HIDE);
        iprev=icur;
        icur=(mevent.y-y-2)/20;
        if(iprev!=icur)
            {_putimage(x+2,y+2+iprev*20,box,_GXOR);
                write_ftext(x+3,y+3+iprev*20,1,0,mnu[iprev]);
            _putimage(x+2,y+2+icur*20,box,_GXOR);
                write_ftext(x+3,y+3+icur*20,1,0,mnu[icur]);
                SetPtrPos(x+(len+5)/2,y+1+icur*20);
            goto choose; }
        else
            continue;
    }
    else
        continue;
}
choose:if(if(place==4 && icur==0 && (kind==1 || kind==2))
{ _putimage(x+2,y+2,box,_GXOR);
    type=-1*type;
    if(type==-1)
        mnu[0]="PNP";
    else
        mnu[0]="NPN";
    goto again;
}
else if(if(place==4 && icur==1 && (kind==1 || kind==2))
{ _putimage(x+2,y+22,box,_GXOR);
    iwhich=-1*iwhich;
    if(iwhich==-1)
        mnu[1]="Inverse";
    else
        mnu[1]="Forward";
en=(kind==1)?6:5;
add=(kind==1)?5:7;
for(i=2;i<en;i++)
    {strcpy(temp,mnu[i]);
     mnu[i]=mnu[i+add];
     mnu[i+add]=strdup(temp);}
go to again;
}
else if((place==4 & icur!=numstr-1) || place==3 || place==2)
{strcpy(temp,mnu[icur]);
j=2;
while(temp[j]!='$')
j++;
for(i=j+1;i<strlen(temp);i++)
    temp[i]=' ';
l=j;
_putchar(x+2,y+2+icur*20,box,_GPSET);
_write_ftext(x+3,y+3+icur*20,1,0,temp);
i=0;
e=0;
neg=0;
neg2=0;
while((k=getch())!=13)
{if(place==2)
    {if(k!=8)
     {if(j+1<18)
      {j++;
else
    __putchar(x+2,y+2+icur*20,box,_GPSET);
    temp[j]=k;}
    else
    {if(j>l)
     {j--;
     temp[j+1]=' ';
    __putchar(x+2,y+2+icur*20,box,_GPSET);}}
    _write_ftext(x+3,y+3+icur*20,1,0,temp);
else  if(isdigit(k) || (k==46 & i==0) ||
((k=tolower(k))==101 & e==0) || (k==45 & neg==0) || (k==45 & neg2==0 & e==1))
    {if(place==3)
     {if(j+1<11)
      {j++;
else
    __putchar(x+2,y+2+icur*20,box,_GPSET);}
else
    {if(j+1<15)
      {j++;
else
/* Capacitor modelling segment */
void Capacitor()
{
    int xwidth,yheight,cols,rows,halfx,halfy,icase;
double xminl,xmaxl,yminl,ymaxl,a11,a12,a21,a22,f1,f2;
double f3,a31,a32,a33,det,cj0tot,f1o,f2o,f3o,a13,a23;
double m,phi,cj0,bot,cap[2][100],capl[100];
double mtot,phitot,ymax2,ymin2;
char lab[4],temp[50];
int i,j,k,rowMid,colMid;
int iMainCur=0,new=0;
short linecol=10,pixcol=12;

    _putimage(x+2,y+2+icur*20,box,_GPSET);
    temp[j]=k;
    if(k==46)i=1;
    if((k==45 && e==0) || isdigit(k) || i==1)neg=1;
    if(k==45 && e==1)neg2=1;
    if(k==101)
    {e=1;
     i=1;
     neg2=0;}}
else if(k==8 && j>l)
        {j--;
        if(temp[j+1]==46)i=0;
        if(temp[j+1]==45)
        if(e==1)
            neg2=0;
        else
            neg=0;
            if(temp[j+1]==101)e=0;
        temp[j+1]='';
        _putimage(x+2,y+2+icur*20,box,_GPSET);
        write_ftext(x+3,y+3+icur*20,1,0,temp);}
    mnu[icur]=strdup(temp);
    _putimage(x+2,y+2+icur*20,box,_GXOR);
    if(place==3||place==2)
    {_putimage(x,y,plotch,_GPSET);
    hfree(plotch);
    hfree(box);
    return icur;}
    else goto again;
    else
    {_putimage(x,y,plotch,_GPSET);
    hfree(plotch);
    hfree(box);
    return icur;}
}
ITEM mnup[4];

kind=4;
_getvideoconfig( &vc );
rowMid = vc.numtextrows / 2;
colMid = 3*vc.numtextcols / 4;
strcpy(mnup[0].achItem,"Base-Emitter");
mnup[0].iHilite=0;
strcpy(mnup[1].achItem,"Base-Collector");
mnup[1].iHilite=1;
strcpy(mnup[2].achItem,"Collector-Sub");
mnup[2].iHilite=3;
strcpy(mnup[3].achItem,""),
mnup[3].iHilite=0;
imainCur = Menu( rowMid, colMid, mnup, iMainCur);
switch(imainCur)
{ case 0:
icase=22;
break;
 case 1:
icase=30;
break;
 case 2:
icase=35;
break;
 }
strcpy(temp,mnup[iMainCur].achItem);
Incl("nothing");
if(fart==1)
{
fart=0;
return;
}

/* Initialize and save pen and fill flags. */
_clearscreen( _GCLEARSCREEN );
setvid();
rowMid = vc.numtextrows / 2;
colMid = vc.numtextcols / 2;
xwidth=vc.numxpixels;
yheight=vc.numypixels;
halfx=xwidth;
halfy=7*yheight/8;
cols=vc.numtextcols;
rows=vc.numtextrows;
xminl=1E21;
xmaxl=-1E21;
yminl=xminl;
ymaxl=xmaxl;
for(i=0;i<numv;i++)
{ if(idata[0][i]<ymin1) ymin1=idata[0][i];
  if(idata[0][i]>ymax1) ymax1=idata[0][i];
  if(vdata[i]<xmin1) xmin1=vdata[i];
  if(vdata[i]>xmax1) xmax1=vdata[i];
}

newminmax(&xmax1,&xmin1);
newminmax(&ymax1,&ymin1);
_settextcolor(15);
strcat(temp," Capacitance vs Voltage");
write_ftext((xwidth-8*strlen(temp))/2+20,25,1,0,temp);
write_ftext(xwidth/2,27*yheight/30,1,0,"Volts");
if(fabs(ymax1)<1.0e-12)
  {strcpy(temp,"Capacitance (femptoFarads)");
   ymax2=ymax1*1.0e15;
   ymin2=ymin1*1.0e15;
   goto next;}
else if(fabs(ymax1)<1.0e-9)
  {strcpy(temp,"Capacitance (picoFarads)");
   ymax2=ymax1*1.0e12;
   ymin2=ymin1*1.0e12;
   goto next;}
else if(fabs(ymax1)<1.0e-6)
  {strcpy(temp,"Capacitance (nanoFarads)");
   ymax2=ymax1*1.0e9;
   ymin2=ymin1*1.0e9;}
next:write_ftext(65,2*yheight/3,0,temp);
sprintf(lab,"%3.1f",xmin1);
write_ftext(85,27*yheight/30,1,0,lab);
sprintf(lab,"%3.1f",xmax1);
write_ftext(9*xwidth/10+5,27*yheight/30,1,0,lab);
sprintf(lab,"%5.1f",ymin2);
write_ftext(40,27*yheight/30-15,1,0,lab);
sprintf(lab,"%5.1f",ymax2);
write_ftext(40,40,1,0,lab);
_settextwindow(1,1,rows,cols);
_setviewport(90,45,halfx-40,halfy);
_setwindow(TRUE,xmin1,ymin1,xmax1,ymax1);
_setcolor(9);
_rectangle_w(_GBORDER,xmin1,ymax1,xmax1,ymin1);
plotpix(1,idata,vdata,xmax1-xmin1,ymax1-ymin1,pixcol,0);

mtot=0.0;
phitot=0.0;
cj0tot=0.0;
m=.5;
phi=.7;
cj0=1.0e-12;
j=0;
allstart:j=j+2;
start:flo=f1;
f2o=f2;
f3o=f3;
f1=log(cj0)-m*log(1.0+vdata[j]/phi)-log(idata[0][j]);

f2=log(cj0)-m*log(1.0+vdata[numv-j]/phi)-log(idata[0][numv-j]);

f3=log(cj0)-m*log(1.0+vdata[numv/2]/phi)-log(idata[0][numv/2]);
if(fabs(f1)<.001&&fabs(f2)<.001&&fabs(f3)<.001)
goto plotter;
if(flo==f1 && f2o==f2 && f3o==f3)
goto plotter;

det=all*(a22*a33-a23*a32)-a12*(a21*a33-a23*a31)+a13*(a21*a32-a22*a31);

m=m-(f1*(a22*a33-a23*a32)-f2*(a12*a33-a13*a32)+f3*(a12*a23-a13*a22))/det;

phi=phi-(-f1*(a21*a33-a23*a31)+f2*(a11*a33-a13*a31)-f3*(a11*a23-a13*a21))/det;

cj0=cj0-(f1*(a21*a32-a31*a22)-f2*(a11*a32-a12*a31)+f3*(a11*a22-a12*a21))/det;
goto start;

plotter:mtot=mtot+m;
phitot=phitot+phi;
cj0tot=cj0tot+cj0;
if(j<20&&j<floor((numv-2)/2))
goto allstart;
phi=phitot/((float)j*.5);
m=mtot/((float)j*.5);
cj0=cj0tot/((float)j*.5);
for(i=0;i<numv;i++)
{bot=pow(1.0+vdata[i]/phi,-m);
cap[0][i]=cj0*bot;}

_plot(1,numv-1,cap,vdata,xmin1,xmax1,ymin1,ymax1,new,linecol, _GXOR,0,0);
_setviewport(0,0,xwidth-1,yheight-1);

_setwindow(TRUE,0.0,0.0,(double)(xwidth-1),(double)(yheight-1));
_setcolor(15);
_rectangle(_GFILLINTERIOR,100,yheight-20,xwidth-50,yheight);
_setcolor(4);
void Temper()
{
  double egl, eg2, te2, te3, rat, phi;
  char temp[10];

  if(kind!=3)
  {
    write_text(19,30,"Enter Celsius Temperature: ");
    te2=atof(gets(temp))+273.0;
    if(te2<0.0)return;
    write_text(19,30," ");
    rat=te2/300.0;
    vt=vt*te2/te;
  }
  else
  {
    rat=te/300.0;
    te3=te;
    te2=te;
  }
  phi=si*pow(rat,f[XTI])*exp(-(1.0-rat)*f[EG]/vt);
  f[ISE]=se*pow(rat,-f[XTB])*pow(phi/si,1.0/f[NE]);
  f[ISC]=sc*pow(rat,-f[XTB])*pow(phi/si,1.0/f[NC]);
  f[IS]=phi;
  f[BF]=bfo*pow(rat,f[XTB]);
  f[BR]=bro*pow(rat,f[XTB]);
  egl=1.115127841;
  eg2=1.16-7.02e-4*pow(te2,2.0)/(1108.0+te2);
  phi=rat*vej-3.0*vt*log(rat)-(rat*egl-eg2);
  f[CJE]=je*(1.0+f[MJE]*)((400e-6*(te2-300.0)-(phi-vej)/vej));
  f[VJE]=phi;
  phi=rat*vcj-3.0*vt*log(rat)-(rat*egl-eg2);
  f[CJC]=jc*(1.0+f[MJC]*)((400e-6*(te2-300.0)-(phi-vcj)/vcj));
  f[VJC]=phi;
  phi=rat*vsj-3.0*vt*log(rat)-(rat*egl-eg2);
  f[CJS]=js*(1.0+f[MJS]*)((400e-6*(te2-300.0)-(phi-vsj)/vsj));
  f[VJS]=phi;
  if(kind!=3)
    te=te2;
}
else
  te=te3;
return;
}

/* MENU - Module of functions to put menus on the screen and handle keyboard input. To use it, include the MENU.H file in your program. The following functions are public:
  * Menu - Puts a menu on screen and reads input for it
  * Box - Puts a box on screen (fill it yourself)
  * GetKey - Gets ASCII or function key
  * putchar - Displays character using current text position and color
  * The following structures are defined:
  * MENU - Defines menu colors, box type, and centering
  * ITEM - Defines text of menu item and index of highlight character
  * The global variable "mnuAtrib" has type MENU. Change this variable to change menu appearance. */

#include <string.h>
#include <stddef.h>
#include <ctype.h>
#include <graph.h>
#include <bios.h>
#include "menu.h"
#include "mouse.h"

/* Prototype for internal function */
static void Itemize( int row, int col, int fCur, ITEM itm, int cBlank );

/* Default menu attribute. The default works for color or B&W. You can override the default value by defining your own MENU variable and assigning it to mnuAtrib. Or you can modify specific fields at...*/

102
* run time. For example, you could use a different attribute for color
 * than for black and white.
 */

MENU mnuAtrib =
{ _TBLUE, _TBLACK, _TWHITE, _TLIGHTGREEN, _TLIGHTGREEN,
  _TCYAN, _TCYAN, _TBLACK, _TCYAN, _TBLACK,
  TRUE,
  'r', 'γ', 'J', 'L', '|', '~' }

/* Menu - Puts menu on screen and reads menu input from keyboard. When a
 * highlighted hot key or ENTER is pressed, returns the index of the
 * selected menu item.
 * *
 * * Params: row and col - If "fCentered" attribute of
 "mnuAtrib" is true,
 * center row and column of menu; otherwise top left of menu
 * aItem - array of structure containing the text of
 each item
 * and the index of the highlighted hot key
 * iCur - index of the current selection--pass 0 for
 first item,
 * or maintain a static value
 * *
 * Return: The index of the selected item
 * *
 * * Uses: mnuAtrib
 */

int Menu( int row, int col, ITEM aItem[], int iCur )
{
  int i;
  int cItem, cchItem = 2; /* Counts of items and chars per item
   */
  int acchItem[MAXITEM]; /* Array of counts of character in items
    */
  char achHilite[36]; /* Array for highlight characters
    */
  long bgColor; /* Screen color, position, and cursor */
  short fgColor;
  struct rccoord rc;
  unsigned fCursor;

  /* Save screen information. */
fCursor = __displaycursor( _GCURSOROFF );
bgColor = __getbkcolor();
fgColor = __gettextcolor();
rc = __gettextposition();

/* Count items, find longest, and put count of each in array. Also,
 * put the highlighted character from each in a string. */
for( cItem = 0; aItem[cItem].achItem[0]; cItem++ )
{
    acchItem[cItem] = strlen( aItem[cItem].achItem );
    cchItem = (acchItem[cItem] > cchItem) ?
        acchItem[cItem] : cchItem;
    i = aItem[cItem].iHilite;
    achHilite[cItem] = aItem[cItem].achItem[i];
}

achHilite[cItem] = 0; /* Null-terminate and lowercase string */
strlwr( achHilite );

/* Adjust if centered, and draw menu box. */
if( mnuAtrib.fCentered )
{
    row -= cItem / 2;
    col -= cchItem / 2;
}
Box( row++, col++, cItem, cchItem );

/* Put items on menu. */
for( i = 0; i < cItem; i++ )
{
    if( i == iCur )
        Itemize( row + i, col, TRUE, aItem[i], cchItem -
            acchItem[i] );
    else
        Itemize( row + i, col, FALSE, aItem[i], cchItem -
            acchItem[i] );
}
SetPtrPos( col + (cchItem / 2), row + iCur );
iCur = EventLoop( row, col, aItem, iCur, cItem, cchItem,
    acchItem, achHilite );
_setbkcolor( bgColor );
_settextcolor( fgColor );
_settextposition( rc.row, rc.col );
__displaycursor( fCursor );
return iCur;
int EventLoop( int row, int col, ITEM aItem[], int iCur, int cItem,
    int cchItem, int acchItem[], char achHilite[])
{
    unsigned uKey; /* Unsigned key code */
    int iPrev; /* Previous index */
    EVENT meEvent;
    char *pchT; /* Temporary character pointer */
    static int fBtnDown = FALSE;

    while( TRUE )
    {
        if( uKey = GetKey( NO_WAIT ) )
        {
            switch( uKey )
            {
                case U_UP: /* Up key */
                    iPrev = iCur;
                    iCur = (iCur > 0) ? (--iCur % cItem) : cItem - 1;
                    break;
                case U_DN: /* Down key */
                    iPrev = iCur;
                    iCur = (iCur < cItem) ? (++iCur % cItem) : 0;
                    break;
                default:
                    if( uKey > 256 ) /* Ignore unknown function key */
                        continue;
                    pchT = strchr( achHilite, (char)tolower( uKey ) );
                    if( pchT != NULL ) /* If in highlight string, evaluate */
                        iCur = pchT - achHilite; /* and fall through */
                    else
                        continue; /* Ignore unknown ASCII key */
            }
            case ENTER:
                return iCur;
        }
    }
}
else if( GetMouseEvent( &meEvent ) )
{
    SetPtrVis( SHOW );
    /* If mouse is on the menu, respond to various events. */
    if( (meEvent.x >= col) && (meEvent.x < (col + cItem)) &&
        (meEvent.y >= row) && (meEvent.y < (row + cItem)) )
    {
        /* If button is down, drag selection. */
        if( meEvent.fsBtn & LEFT_DOWN )
        {
            fBtnDown = TRUE;
            iPrev = iCur;
            iCur = meEvent.y - row;
        }
        /* If button goes up from down, select current. */
        else if( fBtnDown && !(meEvent.fsBtn & LEFT_DOWN) )
        {
            fBtnDown = FALSE;
            iCur = meEvent.y - row;
            return iCur;
        }
        /* Ignore if no button has been pushed. */
        else
            continue;
    }
    /* Ignore if off menu. */
    else
        continue;
}
else
    continue;
/* Redisplay current and previous if we get here through arrow
* move or drag.
*/
Itemize( row + iCur, col, TRUE, aItem[iCur],
        cchItem - acchItem[iCur] );
Itemize( row + iPrev, col, FALSE, aItem[iPrev],
        cchItem - acchItem[iPrev] );
/* Box - Draw menu box, filling interior with blanks of the border color. *
* * Params: row and col - upper left of box
*  rowLast and colLast - height and width
* * Return: None
* * Uses: mnuAtrib */
void Box( int row, int col, int rowLast, int colLast )
{
    int i;
    char achT[MAXITEM + 2];          /* Temporary array of characters */

    /* Set color and position. */
    _settextposition( row, col );
    _settextcolor( mnuAtrib.fgBorder );
    _setbkcolor( mnuAtrib.bgBorder );

    /* Draw box top. */
    achT[0] = mnuAtrib.chNW;
    memset( achT + 1, mnuAtrib.chEW, colLast );
    achT[colLast + 1] = mnuAtrib.chNE;
    achT[colLast + 2] = 0;
    _outtext( achT );

    /* Draw box sides and center. */
    achT[0] = mnuAtrib.chNS;
    memset( achT + 1, ' ', colLast );
    achT[colLast + 1] = mnuAtrib.chNS;
    achT[colLast + 2] = 0;
    for( i = 1; i <= rowLast; ++i )
    {
        _settextposition( row + i, col );
        _outtext( achT );
    }

    /* Draw box bottom. */
    _settextposition( row + rowLast + 1, col );
    achT[0] = mnuAtrib.chSW;
    memset( achT + 1, mnuAtrib.chEW, colLast );
    achT[colLast + 1] = mnuAtrib.chSE;
    achT[colLast + 2] = 0;
    _outtext( achT );
}

/* Itemize - Display one selection (item) of a menu. This
function
* is normally only used internally by Menu.
* Params: row and col - top left of menu
* fCur - flag set if item is current selection
* itm - structure containing item text and index of highlight
* cBlank - count of blanks to fill
* Return: none
* Uses: mnuAtrib
*/
void Itemize( int row, int col, int fCur, ITEM itm, int cBlank )
{
    int i;
    char achT[MAXITEM];       /* Temporary array of characters */
    */ Set text position and color. */
    _settextposition( row, col );
    if( fCur )
    {
        _settextcolor( mnuAtrib.fgSelect );
        _setbkcolor( mnuAtrib.bgSelect );
    }
    else
    {
        _settextcolor( mnuAtrib.fgNormal );
        _setbkcolor( mnuAtrib.bgNormal );
    }
    */ Display item and fill blanks. */
    strcat( strcpy( achT, "" ), itm.achItem );
    _outtext( achT );
    memset( achT, ' ', cBlank-- );
    achT[cBlank] = 0;
    _outtext( achT );
    */ Set position and color of highlight character, then display it. */
    i = itm.iHilite;
    _settextposition( row, col + i + 1 );
    if( fCur )
    {
        _settextcolor( mnuAtrib.fgSelHilite );
        _setbkcolor( mnuAtrib.bgSelHilite );
    }
else
{
    _settextcolor( mnuAtrib.fgNormHilite );
    _setbkcolor( mnuAtrib.bgNormHilite );
}
_outchar( itm.achItem[i] );

/* GetKey - Gets a key from the keyboard. This routine distinguishes
* between ASCII keys and function or control keys with different shift
* states. It also accepts a flag to return immediately if no key is
* available.
* * Params: fWait - Code to indicate how to handle keyboard buffer:
*   * NO_WAIT Return 0 if no key in buffer, else return key
*   * WAIT Return first key if available, else wait for key
*   * CLEAR_WAIT Throw away any key in buffer and wait for new key
*   * * Return: One of the following:
*   * * Keytype High Byte Low Byte
*    * -------  -------
*    * -------
*    * No key available (only with NO_WAIT) 0    0
*    * ASCII value 0    ASCII
* code
*    * Unshifted function or keypad 1    scan
* code
*    * Shifted function or keypad 2    scan
* code
*    * CTRL function or keypad 3    scan
* code
*    * ALT function or keypad 4    scan
* code
*    * * Note: getkey cannot return codes for keys not recognized
*    * * by BIOS int 16, such as the CTRL-UP or the 5 key on the numeric keypad.
*/
unsigned GetKey( int fWait )
{
unsigned uKey, uShift;

/* If CLEAR_WAIT, drain the keyboard buffer. */
if( fWait == CLEAR_WAIT )
    while( _bios_keybrd( _KEYBRD_READY ) )
        _bios_keybrd( _KEYBRD_READ );

/* If NO_WAIT, return 0 if there is no key ready. */
if( !fWait && !_bios_keybrd( _KEYBRD_READY ) )
    return FALSE;

/* Get key code. */
uKey = _bios_keybrd( _KEYBRD_READ );

/* If low byte is not zero, it's an ASCII key. Check scan code to see
 * if it's on the numeric keypad. If not, clear high byte and return.
 */
if( uKey & 0x00ff )
    if( (uKey >> 8) < 69 )
        return( uKey & 0x00ff );

/* For function keys and numeric keypad, put scan code in low byte
 * and shift state codes in high byte. */
if( uKey >= 8; uShift = _bios_keybrd( _KEYBRD_SHIFTSTATUS ) & 0x000f;
switch( uShift )
{
    case 0:
        return( 0x0100 | uKey ); /* None (1) */
    case 1:
    case 2:
    case 3:
        return( 0x0200 | uKey ); /* Shift (2) */
    case 4:
        return( 0x0300 | uKey ); /* Control (3) */
    case 8:
        return( 0x0400 | uKey ); /* Alt (4) */
}

/* _outchar - Display a character. This is the character
   equivalent of _outtext. It is affected by _settextposition,
   _settextcolor, and _setbkcolor. It should not be used in loops. Build strings
and then
* _outtext to show multiple characters.
*
* Params: ch - character to be displayed
*
* Return: none
*/
void _outchar( char ch )
{
    static char ach[2] = " "; /* Temporary array of
characters */
    ach[0] = ch;
    _outtext( ach );
}

/* ClickOrPress - Checks to see if a key has been pressed or
a mouse
* button clicked. A click is defined as pressing and then
releasing.
*
* Params: none
*
* Return: TRUE or FALSE
*/
int ClickOrPress()
{
    EVENT ev;
    int i = 0;

    /* Check for press. */
    if( GetKey( NO_WAIT ) )
        return TRUE;

    /* Check for click. If button is down, wait until it is
released. */
    if( !GetMouseEvent( &ev ) )
        return 0;
    if( ev.fsBtn )
    {
        while( TRUE )
            if( GetMouseMoveEvent( &ev ) && !ev.fsBtn )
                return TRUE;
    }
    return FALSE;
}
/ * MOUSE - Module of mouse functions. To use it, include the MOUSE.H file
 * in your program. The following functions are public:
 * 
 * MouseInit - Initialize mouse
 * GetMouseEvent - Get information about most recent mouse event
 * SetPtrVis - Set visibility of pointer to HIDE or SHOW
 * SetPtrPos - Set position of pointer
 * SetPtrShape - Set shape of pointer in graphics modes, or
 * GetPtrPos - Get pointer position and button status
 * 
 * The following structure is defined:
 * 
 * EVENT - Defines x, y, and mouse status of a mouse event
 */

#include <graph.h>
#include "mouse.h"

/* Internal information used by various mouse functions. */
struct MOUINFO
{
    int fExist, fInit, fGraph;
    short xVirtual, yVirtual;
    short xActual, yActual;
    short xLast, yLast;
    unsigned fsBtnLast, cBtn;
} static mi =
{
    1, 0, 0,
    0, 0,
    0, 0,
    0, 0,
    0, 0
};

/* MouseInit - Initialize mouse and turns on mouse pointer. Initializes
 * all internal variables used by other mouse functions. This function
 * should be called whenever a new video mode is set, since internal
 * variables are mode-dependent.
 * */


* Params: none
* Return: 0 if no mouse available, otherwise number of
  buttons available
*/

int MouseInit()
{
    struct videoconfig vc;
    char _far *pMode = (char _far *)0x00000449; /* Address for
    mode */

    /* Get video configuration. */
    _getvideoconfig( &vc );

    /* Handle special case of Hercules graphics. To use mouse
    with video
    * page 0. assume mode 6. To use mouse with page 1, assume
    mode 5.
    * Since the mouse functions couldn't easily detect and
    adjust for
    * page changes anyway, this code assumes page 0. Note
    also that the
    * mouse for Hercules graphics must be set in text mono
    mode.
    */
    if( vc.mode == _HERCمونو )
    {
        _setvideomode( _TEXTمونو );
        *pMode = 6;
    }

    mi.fInit = 1;
    asm
    {
        sub ax, ax ; Mouse function 0, reset
        int 33h
        mov mi.fExist, ax ; Set existence flag for
        future calls
        or ax, ax ; If AX = 0, there is no
        mouse
        jnz exist
        ret ; so quit
        exist: mov mi.cBtn, bx ; Save number of mouse
        buttons for return
    }

    /* Set graphics flag. */
if( vc.numxpixels )
{
    mi.fGraph = 1;
    mi.yActual = vc.numypixels - 1;
    mi.xActual = vc.numxpixels - 1;
}
else
    mi.fGraph = 0;

/* The mouse works on a virtual screen of 640 x pixels by (8 * textrows) vertical pixels. By default, it assumes 640 x 200 for 25-line mode. You must call function 8 to adjust for other screen sizes. */

mi.xVirtual = 639;
if( mi.fGraph )
    mi.yVirtual = vc.numypixels - 1;
else
    mi.yVirtual = (vc.numtextrows << 3) - 1;

/* Reset Hercules graphics mode and reset the height. */
if( vc.mode == _HERCMONO )
{
    _setvideomode(_HERCMONO);
    mi.xVirtual = 719;
}

_asm
{
    mov ax, 8 ; Set minimum and maximum vertical
    sub cx, cx ; Minimum is 0
    mov dx, mi.yVirtual ; Maximum is 8 * rows (or rows SHL 3)
    int 33h ; Adjust for 25, 30, 43, 50, or 60 lines

    mov ax, 1 ; Turn on mouse pointer
    int 33h

    mov ax, 3 ; Get initial position and button status
    int 33h
    mov mi.xLast, cx ; Save internally
    mov mi.yLast, dx
    mov mi.fsBtnLast, bx
}
return mi.cBtn;                /* Return the number of
mouse buttons */
}

/* GetMouseEvent - Check to see if there has been a mouse
event. If event
* occurred, update event structure.
* Params: pEvent - Pointer to event structure
* Return: 1 if event, 0 if no event
*/
int GetMouseEvent( EVENT _far *pEvent )
{
    int rtn;

    /* Make sure that mouse is initialized and exists. */
    if( !mi.fInit )
        MouseInit();
    if( !mi.fExist )
        return 0;

    _asm
    {
        mov ax, 3            ; Get Mouse position and
        button status
        int 33h
        sub ax, ax            ; Assume no event
        cmp cx, mi.xLast      ; Has column changed?
        jne event
        cmp dx, mi.yLast      ; Has row changed?
        jne event
        cmp bx, mi.fsBtLast   ; Has button changed?
        je noevent
        event:
        mov ax, 1            ; If something changed,
        event occurred
        mov mi.xLast, cx     ; Update internal
        variables
        mov mi.yLast, dx
        mov mi.fsBtLast, bx
        noevent:
        mov rtn, ax          ; Set return value
    }

    /* If event, put adjust values in structure. */
    if( rtn )
    {


/* If graphics mode, adjust virtual mouse position to actual screen coordinates. */
if( mi.fGraph )
{
    pEvent->x = ((long)mi.xLast * mi.xActual) / mi.xVirtual;
    pEvent->y = ((long)mi.yLast * mi.yActual) / mi.yVirtual;
}
/* If text mode, adjust virtual mouse position to 1-based row/column. */
else
{
    pEvent->x = (mi.xLast >> 3) + 1;
    pEvent->y = (mi.yLast >> 3) + 1;
}
pEvent->fsBtn = mi.fsBtnLast;
return rtn;

/* GetPtrPos - Get mouse pointer position and button status regardless of whether there was an event. */

int GetPtrPos( EVENT _far *pEvent )
{
    /* Make sure that mouse is initialized and exists. */
    if( !mi.fInit )
        MouseInit();
    if( !mi.fExist )
        return 0;

    _asm
    {
        mov    ax, 3   ; Get Mouse position and button status
        int    33h
        les    di, pEvent
        mov    es:[di].x, cx
        mov    es:[di].y, dx

    return 0;
}
mov es:[di].fsBtn, bx

/* If graphics mode, adjust virtual mouse position to actual
 * screen coordinates. */
if( mi.fGraph )
{
pEvent->x = ((long)pEvent->x * mi.xActual) / mi.xVirtual;
pEvent->y = ((long)pEvent->y * mi.yActual) / mi.yVirtual;
}
/* If text mode, adjust virtual mouse position to 1-based
 * row/column. */
else
{
pEvent->x >>= 3;
pEvent->y >>= 3;
pEvent->x++;
pEvent->y++;
}
return 1;

/* SetPtrVis - Set pointer visibility. *
 * Params: state - SHOW or HIDE
 * Return: 0 if no mouse, otherwise 1 */
int SetPtrVis( enum PTRVIS pv )
{
    /* Make sure that mouse is initialized and exists. */
    if( !mi.fInit )
        MouseInit();
    if( !mi.fExist )
        return 0;

    _asm
    {
        mov ax, pv ; Show or hide mouse pointer
        int 33h
    }
}
/* SetPtrPos - Set mouse pointer position. */
/* Params: x - column position in text modes, actual x coordinate in graphics */
/* y - row position in text modes, actual y coordinate in graphics */
/* Return: 0 if no mouse, otherwise 1 */
int SetPtrPos( short x, short y )
{
    /* Make sure that mouse is initialized and exists. */
    if( !mi.fInit )
        MouseInit();
    if( !mi.fExist )
        return 0;

    /* If graphics, adjust actual coordinates to virtual coordinates. */
    if( mi.fGraph )
    {
        x = ((long)x * mi.xActual) / mi.xVirtual;
        y = ((long)y * mi.yActual) / mi.yVirtual;
    }
    /* If text, adjust row/column to 0-based virtual coordinates. */
    else
    {
        x--;
        y--;
        x <<= 3;
        y <<= 3;
    }

    asm
    {
        mov  ax, 4     ; Set mouse position
        mov  cx, x
        mov  dx, y
        int  33h
    }
    return 1;
}

/* SetPtrShape - Set mouse pointer shape. */
/* Params: x - column position in text modes, actual x coordinate in graphics */
/* y - row position in text modes, actual y coordinate */
in graphics
*
* Return: 0 if no mouse, otherwise 1
*/
int SetPtrShape( PTRSHAPE _far *ps )
{
    /* Make sure that mouse is initialized and exists. */
    if( !mi.fInit )
        MouseInit();
    if( !mi.fExist )
        return 0;

    /* If graphics, use pointer shape bitmask array. */
    if( mi.fGraph )
    {
        asm
            {
                les    di, ps
                mov    bx, es:[di].g.xHot    ; Load hot spot
                mov    cx, es:[di].g.yHot
                mov    dx, di
                add    dx, 4
                mov    ax, 9
                ; Set graphics pointer
                int    33h
            }
    }

    /* If text, use pointer color/character values. */
    else
    {
        asm
            {
                les    di, ps
                mov    bx, 0
                ; Use software cursor
                mov    cl, es:[di].t.chScreen
                mov    ch, es:[di].t.atScreen
                mov    dl, es:[di].t.chCursor
                mov    dh, es:[di].t.atCursor
                mov    ax, 10
                ; Set text pointer
                int    33h
            }
        return 1;
    }
}
Bibliography


