Department of Applied Analysis
and Computer Science
Research Report CSTR 1001
March, 1970
THE DIALATOR SYSTEM
FAUST USER'S GUIDE
by
Doron J. Cohen, Paul M. Fawcett
Eric G. Manning & Larry Smith

Faculty of Mathematics

University of Waterloo
Waterloo, Ontario
Canada
Department of Applied Analysis
and Computer Science
Research Report CSTR 1001
March, 1970
THE DIALATOR SYSTEM
FAUST USER'S GUIDE
by
Doron J. Cohen, Paul M. Fawcett
Eric G. Manning & Larry Smith
We wish to thank the Defence Research Board of Canada, the Northern Electric Research & Development Laboratories, Ottawa, and the Faculty of Mathematics of the University of Waterloo for financial, technical and moral support which made this system of programs possible.
This is a user's guide to program FAUST of the DIALATOR system. (A system programmer's guide explaining internal workings of the program appears later.) This guide is laid out as follows:

SECTION 1) explains how faults of a circuit are defined.
SECTION 2) explains how failed circuit behaviour is displayed.
SECTION 3) explains how faults of a circuit are diagnosed.
SECTION 4) explains the FAUST control card.
SECTION 5) explains the FAUST instruction keywords and their usage.
SECTION 6) explains how selected faults are defined.
SECTION 7) explains the control cards needed to make a FAUST run on a system 360 computer under OS/360.
SECTION 8) contains a few examples of FAUST runs.
SECTION 9) lists the FAUST error messages.

SECTION 1) Definition of circuit faults

Let us take a second look at CIRCUIT1, used in the TRAIIZE user's manual.
Suppose Al is faulty. The reason could be any of
(a) the output of Al is stuck at one,
(b) the output of Al is stuck at zero,
(c) the wire from X1 to Al has opened up,
(d) the wire from X2 to Al has opened up.

Just as in TRAIZE, where we described complete circuits by

giving lead names, their functions, and their input references, we can

now describe all single faults of a circuit by giving all single faults

possible at the particular lead, plus all single faults possible between

this lead and its input references. The former type of fault is called

a basic fault; the latter is called an input fault.

For CIRCUIT1 a list of all single faults would be:

1  fault free circuit.
2   A1  output stuck at zero.
3   A1  output stuck at one.
4   A1  open input diode from X1.
5   A1  open input diode from X2.
6   N1  output stuck at zero.
7   N1  output stuck at one.
8   N1  open input diode from X1.
9   N1  open input diode from X2.
10  A2  output stuck at zero.
11  A2  output stuck at one.
12  A2  open input diode from X1.
13  A2  open input diode from Al.
14  $\phi$1  output stuck at zero.
15  $\phi$1  output stuck at one.
16  $\phi$1  open input diode from Al.
17  $\phi$1  open input diode from N1.
One other basic fault, not applicable here, can occur in circuits with feedback loops. (i.e. the circuit may have an open feedback diode).

Notice that X1 and X2 have no associated faults. The reason for this is that the values at X1 and X2 are independent of CIRCUIT1.

For simplicity, we will discuss at this time circuits with single faults only. The handling of multiple faults is discussed in Section 6 of this manual.

Section 2) **Failed circuit displays**

In distinguishing failed circuits from each other and from the good circuit, it is important to see the outputs of the circuit under each fault condition.

For CIRCUIT1 we could accomplish this as follows, (suppose the inputs are X1 = 0 and X2 = 0):

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Øl</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**DIAGRAM 2**

Along the top of the table are listed the numbers of the particular faults. The first column is reserved for no faults. For each particular fault we write down the output, below the fault number, at A2 and Øl. You should check some of these values to verify that they are valid.
A more complete picture, of the action in CIRCUIT1 under the 17 different conditions, can be obtained by listing all the leads rather than just the outputs. Let the inputs be X1 = 1 and X2 = 0, then the table display would be as follows:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>X2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>N1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Ø1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**DIAGRAM 3**

Diagram 3 gives us the value of every circuit signal under every possible single fault. Thus it tells us everything that we could possibly ask about the behaviour of CIRCUIT1 (under single faults).

The above diagrams (2 & 3) are known as $LVBs$ tables, meaning lead value bit string tables. Now that we have seen a way of displaying single faults of a circuit, we explain how you get FAUST to create such displays.

We need instructions to manipulate the primary inputs of the circuit. In diagram 3 we wanted X1 to be one. In FAUST, SET X1 does this. To set X2 to zero FAUST uses RESET X2. Another valuable instruction is FLIP. This command changes the values on all specified leads. For example,
if X1 and X2 are as in diagram 3 then FLIP X1, X2 changes X1 to zero and X2 to one.

We need an instruction to activate the simulation routine; RUN does this.

We also need instructions to show the $LVBS. Since it is important that we see the $LVBS after each run perhaps the instruction should be automatic. Even though an automatic showing after each run is good, we may wish to vary the amount of the $LVBS to be shown. To do this FAUST provides the instructions TRACE and UNTRACE. Suppose we are presently following the action of leads A2 and $\phi 1$. If we say TRACE X1, X2, A1, N1, then the next display would be similar to diagram 3. The TRACE instruction, then, adds the specified leads to the list to be shown. If we followed the above statement with UNTRACE X1, X2, A1, N1 we would be back to a display similar to diagram 2. The UNTRACE instruction removes the specified leads from the list to be shown. There is need also, for an instruction to display the $LVBS at times other than after a run. SHOW accomplishes this.

Section 3) Diagnosis of faults

We can now use the $LVBS table displays of SECTION 2, to help distinguish the faults of CIRCUIT1. When the inputs X1 and X2 of CIRCUIT1 are both zero, the faults can be divided into 3 groups. The first group of faults 2, 3, 4, 5, 7, 8, 9, 10, 12, 13, 15 and 16 are associated with outputs A2 = 0 and $\phi 1 = 1$; the second group of faults 6, 14 and 17 with outputs A2 = 0 and $\phi 1 = 0$, the third group of faults 10 with outputs A2 = 1 and $\phi 1 = 1$. There is no group associated with outputs A2 = 1, $\phi 1 = 0$. The

* except for large circuits, where the amount of output would be huge.
third group is interesting since it contains only one fault. In effect, we have isolated fault 11; that is if \( X_1 = 0 \) and \( X_2 = 0 \) causes \( A_2 = 1 \) and \( \phi_1 = 1 \) we know the circuit has fault 10.

The application of input strings, rather than single inputs, would increase diagnosing ability. Suppose we give commands:

RESET \( X_1, X_2 \);

TRACE \( A_2, \phi_1 \);

RUN ;

We will produce diagram 2 of Section 2). In the paragraph above we divided the faults, for this case, into the following groups:

```

GROUP 1
OUTPUT
A2=0
\phi1=0

GROUP 2
OUTPUT
A2=0
\phi1=1

GROUP 3
OUTPUT
A2=1
\phi1=1

Associated Faults
6
14
17
2, 3
4, 5
7, 8
9, 10
12, 13
15, 16
```

DIAGRAM 4
Suppose now we dictate:

```
SET X1;
TRACE X1,X2,A1,N1;
RUN;
```

We will get diagram 3 of SECTION 2.

Look at DIAGRAM 3, keeping in mind the above groups. The outputs of the elements of group 1 are all \((0,0)\), therefore no further distinction is accomplished. In group 2, fault 3 has output \((1,1)\) which is different from the remainder of outputs for that group. A picture of our efforts is given by the following tree diagram.

```
LEVEL 0
***
GROUP 1
***
no inputs
***
1 group
***

LEVEL 1
***
GROUP 1
***
first input
***
(0,0)
***
A2=0
***
\(\phi_1=0\)
***
3 groups
***

LEVEL 2
***
GROUP 1
***
second input
***
(1,0)
***
A2=0
***
\(\phi_1=0\)
***
3 groups
***

SUBSETS
6
2,4
6
14
5,7
8,9
3
17
10,12
13,15
16
11
```
Using the above diagram we can now isolate fault 3. That is, if we apply inputs

\[(0,0)\]

followed by \((1,0)\);

and if we observe the output patterns

\[(A_2,0_1) = (0,1)\]

followed by \((1,1)\)

then fault 3 and only fault 3 must be present.

Now we will look at some simple instructions for manipulating the diagnostic tree. Each time we wish to observe the output of a test we will say SORT. This instruction will sort the members of the present groups according to output and add a new level to the diagnostic tree. Now that a sort is not always done after each run.

Thus we can simulate a test setup wherein we apply several test inputs between obervations of the circuit's outputs.

To print the tree we use the instruction TREE.

If we wish to save the tree and the $LVBS$ in the file, we say, SAVE followed by the name it is to be saved under, and, to retrieve it from the file we use, BACKUP followed by the proper name. We can use the back up feature to try out tests and forget them if they do not produce good results.

Here is a sample set of statements which will make a FAUST run for CIRCUIT1.
ZSAMP1:  SIMULATE CIRCUIT=CIRCUIT1,DISP=NO,FAULTS=ALL;
SORT;
ACTIVATE ALL;
TRACE ALL;
RUN FAULTS=NO;  SHOW ALL;
RESET X1,X2;  RUN;  SORT;
SET X1;  RUN;  SORT;
SAVE RECFNAME=TEST;
FLIP X1,X2;  RUN;  SORT;
TREE;
END;

DIAGRAM 6

The run would go as follows:

First, the FAUST control card is read. The following output results:

---
CPU TIME= 260,400.000, TOTAL CPU TIME= 0.000 MILLISECONDS.

ZSAMP1:  SIMULATE CIRCUIT=CIRCUIT1,DISP=NO,FAULTS=ALL;
CPU TIME= 139.991, TOTAL CPU TIME= 139.991 MILLISECONDS.
04540000 SOURCE.
LCID==CIRCUIT NAME: CIRCUIT1, OPTIONS: 0000
FAULTS OPTION: ALL
FDIR==OLD DIRECTORY FETCHED.

CIRCUIT NODE NO.79
$CIDS  NODE NO.71 86 BYTES,STARTING REGION NO.12 WORD NO.525
$LIDS  NODE NO.78 168 BYTES,STARTING REGION NO.12 WORD NO.547
$FMCS  NODE NO.84 132 BYTES,STARTING REGION NO.12 WORD NO.450
$SYMTA NODE NO.81 56 BYTES,STARTING REGION NO.12 WORD NO.492
$REFS  NODE NO.82 75 BYTES,STARTING REGION NO. 0 WORD NO. 20
TERM NOT FOUND.

STRUCTURES LOCATIONS

STRUCTURE:$CIDS  OF NODE NO.71,FETCHED 86 BYTES,STARTING REGION NO.12 WORD NO.525
STRUCTURE:$LIDS  OF NODE NO.78,FETCHED 168 BYTES,STARTING REGION NO.12 WORD NO.547
STRUCTURE:$FMCS  OF NODE NO.84,FETCHED 132 BYTES,STARTING REGION NO.12 WORD NO.450
STRUCTURE:$SYMTA OF NODE NO.81,FETCHED 124 BYTES,STARTING REGION NO.12 WORD NO.492
STRUCTURE:$REFS  OF NODE NO.82,FETCHED 56 BYTES,STARTING REGION NO.12 WORD NO.589
STRUCTURE:$SUCS  OF NODE NO.83,FETCHED 75 BYTES,STARTING REGION NO. 0 WORD NO. 20
EV='01111111'B  R= 126
FETCH==$COP'T
STRUCTURE:$COP'T  OF NODE NO.60,FETCHED 2610BYTES,STARTING REGION NO. 0 WORD NO.111
FETCH==FAUST.$OPTAB
STRUCTURE:$OPTAB OF NODE NO.35,FETCHED 324 BYTES,STARTING REGION NO.10 WORD NO.408
CPU TIME= 1,361.579, TOTAL CPU TIME= 1,501.570 MILLISECONDS.
After each FAUST statement the amount of C.P.U. time required is tabulated, plus, the total amount of CPU time used since the beginning of execution is given. The first line of output indicates that the CPU timer has been initialized properly.

The source statements of the FAUST deck listed above appear in the output with an eight digit number which is immediately followed by 'SOURCE.'.

The FAUST control card causes a variable number of actions to take place depending on the parameters used.

In this example the name 'CIRCUIT1' along with the names of its associated structures are fetched from the file directory into core. Next, for those names which have associated storage, their structures are fetched from the record file into core. Notice that with these structure names a number of bytes, a starting region, and a starting word are given, also for all structure names a node number is given. The usefulness of this information and an explanation of how the file system works is found in the FILE SYSTEM USER'S GUIDE.

Next the circuit operators table, $COPT, along with the FAUST operator's table, FAUST.$OPTAB, are fetched from the file into core. Similar information about them is given as was given for the other structures.

At this point all the necessary tables for CIRCUIT1 and for FAUST instructions are in core.
The following output is a result of the FAUST instructions.

```
= DIAGNOSTIC TREE INITIALIZED.
CPU TIME= 14.165, TOTAL CPU TIME= 1,515.735 MILLISECONDS.

0> CPU TIME= 2.499, TOTAL CPU TIME= 1,518.234 MILLISECONDS.
    SORT;

1> CPU TIME= 113.326, TOTAL CPU TIME= 1,631.560 MILLISECONDS.
    ACTIVATE ALL;

2> CPU TIME= 101.660, TOTAL CPU TIME= 1,733.220 MILLISECONDS.
    TRACE ALL;

3> CPU TIME= 68.328, TOTAL CPU TIME= 1,801.548 MILLISECONDS.
```

Since the instruction INITREE, was not used the diagnostic
tree was initialized automatically.

The instruction SORT; creates the a first level of the tree.

A glance at the tree below will show that this sort gives 0% diagnostic
resolution.

The instruction ACTIVATE ALL; sets activity bits of all leads
on. Unless the user is very familiar with the behaviour of circuit
models ; he should use this instruction always.

The instruction TRACE ALL; indicates that after each run we
wish to see the $LVBS for all leads of the circuit.

The next two instructions give the following input,

```
RUN FAULTS=NO; SHOW ALL;
04580000SOURCE.

PROPOGATION: 1, 2, 3, 4, 5, 6,

===DTMUX== DONE.
```
The instruction RUN FAULTS=NO; causes the $LVBS table to be printed. The 'PROPOGATION' statement indicates which leads have undergone any changes since the last run, since this is the first run all six lead numbers are printed. The $LVBS is similar to the one described in SECTION 2) except for the following additions:

There is one column for the activity bit vector headed by @.

There is an extra column in the table headed by the number zero.
This is a buffer column and can be ignored by the user. There are five rows above the lead names. These rows are used for workspace and they are printed only to aid debugging. They should be ignored by the user also.

The next instruction SHOW ALL; displays the $LVBS. Since nothing has been done since the last showing it is the same. Therefore this statement is redundant.

The next three instructions give the following output.

```
RESET X1,X2; RUN; SORT;
6>
CPU TIME= 70.828, TOTAL CPU TIME= 2,095.694 MILLISECONDS.
PROPOGATION: 3, 4, 5, 6,
==DIMUY== DONE.
```

```
1 NO. NAME @ 012345678901234567
-4 0000000000000000000000
-3 0000000000000000000000
-2 0000000000000000000000
-1 0000000000000000000000
 0 111111011111111111
 1 X1 0 0000000000000000000000
 2 X2 0 0000000000000000000000
 3 A1 0 0001000000000000000000
 4 N1 0 111111011111111111
 5 A2 0 00000000000100000000
 6 ø1 0 11111110.1111110111
7>
CPU TIME= 79.954, TOTAL CPU TIME= 2,175.688 MILLISECONDS.
```

```
8>
CPU TIME= 49.996, TOTAL CPU TIME= 2,225.684 MILLISECONDS.
```
The FAUST language allows more than one statement per line.
In order that the CPU time can be given for each statement the number of a statement is printed immediately before the CPU time for that statement.

The $LVBS here is the same as was described earlier for inputs \((X_1, X_2) = (0, 0)\). The sort statement adds another level to the tree. See LEVEL 1 of the tree below.

The next card causes this output.

```
SET X1; RUN; SORT;
9>
CPU TIME= 88.327, TOTAL CPU TIME= 2,314.011 MILLISECONDS.
PROPOGATION: 3, 4, 5, 6,
==DIMUY== DONE.
1
NO. NAME @ 012345678901234567
  4  00000000000000000000
  3  00000000000000000000
  2  00000000000000000000
  1  00000000000000000000
  0  111111110110111111111111
 1 X1  0 11111111111111111111
 2 X2  0 00000000000000000000
 3 A1  0 00010100000000000000
 4 N1  0 11111101101111111111
 5 A2  0 00010100000010100000
 6 Ø1  0 11111110110111101111
10>
CPU TIME= 79.161, TOTAL CPU TIME= 2,393.172 MILLISECONDS.
11>
CPU TIME= 56.663, TOTAL CPU TIME= 2,449.835 MILLISECONDS.
```

These three instructions perform the same actions as the previous three. Now, however, lead X1 is set to one. For results see LEVEL 2 of the tree below.
The output for the next statement is:

```
SAVE RECNME=TEST;
==STORE==CIRCUIT1.TEST.$LVBS
STRUCTURE: $LVBS OF NODE NO.87,STORED 74 BYTES,STARTING REGION NO.12 WORD NO.612
==STORE==CIRCUIT1.TEST.$DIATR
STRUCTURE: $DIATR OF NODE NO.88,STORED 347 BYTES,STARTING REGION NO.12 WORD NO.699
12>
CPU TIME= 176.655, TOTAL CPU TIME= 2,626.490 MILLISECONDS.
```

At this point we wish to save the results under the recname TEST.

Both the $LVBS table and the diagnostic tree are saved under the names TEST.$LVBS and TEST.$DIATR. File information is given about these structures as well. If during some future run we wish to start testing again at this point we say.

```
BACKUP RECNME=TEST;
```

that is, of course, if we are still dealing with CIRCUIT1.

Instructions 13, 14 and 15 given output similar to instructions 9, 10 and 11. The values of X1 and X2 are changed.

```
FLIP X1,X2;   RUN;   SORT;
```

```
13>
CPU TIME= 94.160, TOTAL CPU TIME= 2,720.650 MILLISECONDS.
```

```
PROPAGATION: 3, 4, 5, 6,
==DIMUY== DONE.
```

```
1
NO. NAME @ 012345678901234567
-4 000000000000000000
-3 000000000000000000
-2 000000000000000000
-1 000000000000000000
0 11111111111111111
1 X1 0 000000000000000000
2 X2 0 11111111111111111
3 A1 0 000110000000000000
4 N1 0 11111101011111111
5 A2 0 000000000001000000
6 Ø1 0 11111101011111111
```
CPU TIME= 79.994, TOTAL CPU TIME= 2,800.644 MILLISECONDS.
CPU TIME= 51.663, TOTAL CPU TIME= 2,852.307 MILLISECONDS.

The next instruction TREE; will print out the entire tree which should have 4 levels (4 sort statements). The output is as follows,

TREE;
CPU TIME= 82.494, TOTAL CPU TIME= 2,934.801 MILLISECONDS.
==TREE== FL= 0 HS= 0 LL= 4
FP= 18 HP= 0 LP= 19;
LEVEL# 0 1 NODES, 1 SUBSETS.
LEVEL# 1 3 NODES, 1 SUBSETS.
LEVEL# 2 4 NODES, 2 SUBSETS.
LEVEL# 3 4 NODES, 3 SUBSETS.

===== THE DIAGNOSTIC TREE =====

RESOLUTIONS:

LEVEL# 0
*001*
0.0%DIA.
0.0%DTC.

*****

LEVEL# 1
*001*-----------*002*---*003*
66.6%DIA.
* 2 -----------* 3 *--* 0 *
17.6%DTC.

*****

LEVEL# 2
*001*---*002*---*003*
33.3%DIA.
* 2 *---* 3 *--* 0 *
28.5%DTC.

*****

LEVEL# 3
*001*---*002* *003*
11.1%DIA.
* 2 *--* 0 * * 2 *
10.0%DTC.

*****

MACHINER: # 1# # 8# # 3# # 9# # 11# # 6#
# 2# # 5# # 14#
OVER ALL # 4# # 13#
31.2%DIA. # 7#
50.0%DTC. # 10#
# 12#
# 15#
# 16#
# 17#

16>
The first part of the output lists the parameters given by the user, by default in this example. This is followed by a list of levels, their number of nodes, and the number of subsets at each level. To the left of the tree an indication is given as to the diagnostic resolution (DIA) and the detection resolution (DTC) (i.e. how many faulty machines have been distinguished from the good machine). In this example we have completely isolated 3 machines out of 17.

On each level of the tree the nodes are numbered from left to right. The top number in each node indicates the position of the node on its level. We can identify any node of the tree by giving this node number and the proper level number.

The lower number in each node indicates the output pattern of the faulty circuits associated with that node. This decimal number when changed to its binary equivalent indicates the output of OUTPUT#1, OUTPUT#2, ... in that order.

There is no confusion as to which output is number one etc. The ordering is assigned previously by TRAIZE (see SECTION 8), example 1). Since CIRCUIT1 has only 2 outputs the lower number can be one of 0, 1, 2, or 3.

The failed circuits associated with a node are all those included in the subtree formed with the particular node as root.
The next output is a listing of the file directory tree and will be discussed in the file system's USER'S GUIDE.

****** DIRECTORY LISTING ******
48 AVAIL POINTER.
40 REGIONS USED.
24 STRUCTURES RECOGNIZED.
141 NODES IN THE DIRECTORY.
141 DESIRED NO. OF NODES.
22 KEY-WORDS RECOGNIZED.
700012 DATE OF LAST CHANGE.

****** DIATORA KEY-WORDS TABLE ******

<table>
<thead>
<tr>
<th>NO.</th>
<th>KEY-WORD</th>
<th>INDEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>9</td>
<td>$C10S</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>$C0P</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>$C10T</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>$C10M</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>$C10R</td>
<td>4</td>
</tr>
<tr>
<td>14</td>
<td>$C10S</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>$C10A</td>
<td>13</td>
</tr>
<tr>
<td>16</td>
<td>$C10B</td>
<td>14</td>
</tr>
<tr>
<td>17</td>
<td>$C10C</td>
<td>6</td>
</tr>
<tr>
<td>18</td>
<td>$C10D</td>
<td>9</td>
</tr>
<tr>
<td>19</td>
<td>$C10E</td>
<td>7</td>
</tr>
<tr>
<td>20</td>
<td>$C10F</td>
<td>5</td>
</tr>
<tr>
<td>21</td>
<td>$C10G</td>
<td>12</td>
</tr>
<tr>
<td>22</td>
<td>$C10H</td>
<td>8</td>
</tr>
</tbody>
</table>

****** THE DIRECTORY TREE ******

#NL. #FC_NAME:
# 5172 $C10G $C10F SIS DATA
# * 9 REC_FILE
# * -1 -1 1 

# NL. #FC_NAME:

************
# 1 HEADFILE 00
# 1 -1 0 88 0
# 03 $FREE, 029
# 710 12 0 87 0
# 87 $FREE, 769
# 512 12 0 89 0
# 89 $FREE, 139
# 218 12 0 77 0
# 77 $FREE, 30
# 564 12 0 65 0
# 85 $FREE, 749
# 882 11 0 80 0
# 80 $FREE, 379
# 603 12 0 70 0
# 70 $FREE, 319
# 523 12 0 75 0
# 75 $FREE, 759
# 86 10 0 76 0
# 76 $FREE, 559
# 730 10 0 73 0
# 73 $FREE, 1294
# 303 10 0 72 0
# 72 $FREE, 349
# 493 12 0 67 0
# 57 $FREE, 299
# 402 10 0 96 0
# 56 $FREE, 249
# 744 10 0 59 0
# 55 $FREE, 659
# 69 10 0 26 0
# 26 $FREE, 269
# 794 10 0 23 0
************
- 19A -

***
* 63 B0JATR  554*
* 1731 11 11 -1 -1*


SECTION 4) FAUST control card

The basic rules for writing the FAUST control card are the same as for the TRAIZE control card. (See TRAIZE user's guide.) Referring to diagram 6 of SECTION 3): a 'Z' sign must appear in column one, followed by a runname, followed by a keyword (SIMULATE instead of TRAIZE), followed by a circuit name which must be known, followed by parameters.

The DISP parameter explains disposition of the run results:* DISP can have four values. If we wish to save these results unconditionally, we put DISP = YES. If we do not wish to save the run results in the file we put DISP = NO. If we wish to create new entries in the file directory for the run results we put DISP = NEW. If we wish to overwrite already existing records of run results with the same name we put DISP = OLD.

The FAULTS parameter explains the fault option desired; FAULTS can have four values. If we wish to simulate all single faults of a circuit we put FAULTS = ALL. If we wish to simulate the circuit with no faults, we put FAULTS = NO. If we wish to simulate the circuit with selected faults which do not exist in the DIALATOR file for that circuit (i.e. we have not made a previous run with these same selected faults of the circuit and stored them), we put FAULTS = GET. Note, when this value of the parameter is specified statements defining the selected faults must follow the control card. * If we wish to simulate the circuit with selected faults previously run and stored in the DIALATOR file, we put FAULTS = name, where name is the runname of the previous

* These statements are described in SECTION 6)

* 'run results', meaning the diagnostic tree and the $LVBS.
run referred to above. For instance the control card:

    % SAMP17: SIMULATE CIRCUIT=CIRCUIT1,FAULTS=SAMP2;

refers to a previous runname SAMP2, under which some selected faults,
now desired, are stored in the DIALATOR file.

The list parameter explains that we do or do not want the
description of the specified circuit printed. If we wish to have the
circuit description printed, we put LIST = YES, if we do not, we put
LIST = NO.

The default values of DISP, FAULTS and LIST are NO.

SECTION 5) FAUST Statements

All FAUST statements must begin with an instruction word.
Some instruction words have qualifiers following. All statements must
end with a semicolon. Diagramatically statements are of the form:

    instruction word [qualifiers];

The following is a list of valid instruction words for
FAUST statements, presently.

    == FAUST OPTAB ==
    ACTIVATE
    BACKUP
    DEACTIVATE
    END
    FLIP
    IN1FAULT
    INITREE
    RESET
    RUN
    SAVE
    SET
    SHOW
    SORT
    TRACE
    TREE
    UNTRACE
Rules for FAUST qualifiers

1) A lead name can be any valid name (maximum of 6 characters) referring to a known lead.

2) A lead number can be any valid lead number and must be preceded immediately by a '#' symbol.

3) A lead list can be the word
   a) ALL meaning all leads.
   or b) INPUTS meaning all primary inputs.
   or c) OUTPUTS meaning all primary outputs.
   or d) FBKS meaning all feedbacks.
   or e) a lead name
   or f) a lead number
   or g) any combination of b), c), d), e), f), separated by commas.

4) A faulty machine number can be any number referring to a valid fault number.

5) A list of faulty machines can be
   a) a faulty machine number
   or b) a faulty machine number followed by 'TO' followed by a larger faulty machine number
   or c) any combination of (a) and (b)

NOTE: Any list of faulty machines must be enclosed in round brackets.
   e.g. (1, 3 TO 6, 12, 14) is valid

NOTE: Any words not separated by a delimiter must have at least one blank between them.
Description of FAUST statements with the instruction word

1) SET

This instruction sets the value of specified leads to one.

Format of this statement type is as follows:

a) SET followed by
b) a list of faulty machines followed by *
c) a list of leads followed by
d) the keyword ACTIVE or UNACTIVE followed by
e) ;

The list of faulty machine options would be of value if we had wished to set the specified leads only for particular faults. If this phrase is omitted the specified leads will be set for all faults. The keywords ACTIVE and UNACTIVE allow the user to manipulate the activity bits ** of the successors. The default is ACTIVE. This option will not be fully described in this manual.

2) RESET

This instruction sets the value of any specified leads to one.

Format of this statement is the same as for SET.

3) FLIP

This instruction changes the values of specified leads from one to zero or from zero to one. Format of this statement is the same as for SET.

* all phrases in this script signify optional qualifiers.

** activity bits are discussed in the FAUST programmer's guide.
4) TRACE

This instruction adds the specified leads to the list of
leads to be included in the $LVBS displays. Format of this statement
is as follows:

a) TRACE followed by

b) lead list followed by

c) ;

TRACE followed by no list reactivates the tracing routine,
(this is used after a blank UNTRACE statement has been used).

5) UNTRACE

This instruction removes the specified leads from the list
of leads to be included in the $LVBS displays. Format of this state-
ment is the same as for TRACE.

6) SHOW

This instruction displays the $LVBS including the specified
leads. Format of this statement is the same as for TRACE except b)
is not optional.

7) SAVE

This instruction saves the diagnostic tree and the $LVBS
table. These data structures are stored under the fully qualified
names: CIRCUITNAME.RECNAME.$LVBS and CIRCUITNAME.RECNAME.$DIATR
the circuit name as specified on the control card and RECNAME is the
record name specified by the RECNAME= option of the SAVE statement.
Format of this statement is as follows:

a) SAVE followed by
b) RECNAME='	recname' followed by
c) ;

If the RECNAME= option is omitted $CTRES.runname is used where the
runname from the control card is used.

8) BACKUP

This instruction retrieves the diagnostic tree and $LVBS
under the fully qualified names used in the SAVE instruction. Format
of this statement is the same as for SAVE.

9) RUN

This instruction activates the simulation routine to update
the $LVBS display. Format of this statement is as follows:

a) RUN followed by

b) FAULTS=NO or FAULTS= ALL followed by
c) ' , ' when b) and d) are used followed by
d) MAXIT= maxit followed by
e) ;

The FAULTS option can be used to specify simulation runs with
a different set of faulty machines, in its absence FAULTS option of the
control card is used. The MAXIT option specifies the maximum number
of iterations through the feedback nodes during the simulation. The
default value for MAXIT is 8.

10) SORT

This instruction looks at the output and sorts the faulty
machines within a group according to their output thus creating a
new level on the diagnostic tree. Format of this statement is as follows:

a) TREE followed by
b) FIRST= fl followed by
c) 's' when b) and d) are used followed by
d) ROOT= hs followed by
e) 's' when d) and f) are used followed by
f) LAST= ll followed by
g) ;

If all the parameters are left out the entire tree is printed.

FIRST option: fl is the first level to be printed.

ROOT option: if hs is greater than zero the hs subset on the fl level is used as the root of the printed tree, namely the tree is printed only from that node down. If hs is zero (default value) then the complete fl level is printed as the first level of the printed tree.

LAST option: ll is the last level of the tree to be printed (if omitted the whole remainder of the tree is printed).

Suppose we have a tree of the following form:

LEVEL 0

LEVEL 1

LEVEL 2

LEVEL 3

DIAGRAM 8
If we put `TREE FIRST=2,LAST=3;` the entire second and third levels will be printed. If we put `TREE ROOT=1;` the entire tree will be printed. If we put `TREE FIRST=2,ROOT=5;` the following will be printed.

```
LEVEL 2
    5

LEVEL 3
    6---7
```

DIAGRAM 9

12) INITREE

This instruction can be used to initialize the diagnostic tree in core. This statement need not appear at the beginning of FAUST statements since the tree will be initialized automatically. Format of this statement is as follows:

a) INITREE followed by

b) UB= upper bound on the tree followed by

c) ;

The option b) is present so the programmer can control the amount of core taken by the tree. The default for UB is three times the number of faulty machines. If the INITREE statement is used after the first FAUST statement, the current version of the diagnostic tree in core is deleted and a new data structure is allocated with appropriate upper bound. This instruction combined with SAVE and BACKUP facilitate the creation of several diagnostic trees for the same
circuit at the same run.

13) INIFAULT

This instruction causes the $LVBS table to be initialized with the stuck at one and stuck at zero faults. Format of this statement is as follows:

a) INIFAULT followed by

b) ;

14) ACTIVATE

and 15) DEACTIVATE

These instructions allow the user to manipulate the activity bits* and thus change the model for which this program is designed. Therefore the user must do so with care. For normal circuit runs the user should put near the beginning of his FAUST statements

ACTIVATE ALL;

The uses of these instructions will not be described further in this manual. Format of these instructions is as follows:

a) ACTIVATE or DEACTIVATE followed by

b) lead list followed by

c) ;

* activity bits are discussed in the FAUST programmer's guide.
SECTION 6: Selected faults definition statements

The purpose of the facility to create a set of selected faults for a circuit, as opposed to all single faults, is twofold. First, if we are not interested in all faults the option will save space and time on the computer, and second; the option has the capability of creating multiple faults.

Each statement defines a set of faults to be associated with a specified lead. Format of the statements are as follows.

a) lead name followed by
   b) ALL
   or c) s@1
   or d) s@0
   or e) OFD
   or f) any combination of (c), (d), (e) followed by
      g) / ALL
   or h) / selected list of input references followed by
      i) ;

If all the optional phrases are missing, all faults for the specified lead are assumed. The '/' sign is to separate the basic faults from the input faults.

Multiple faults

If we wish to associate more than one fault with same faulty machine we can do so using the '+' sign. For example, suppose we
wish to consider the fault, A1 is stuck at one and A2 has an open input diode from X1. We would describe this fault as follows:

   A1 s @ 1; + A2/ X1;

The plus sign will combine the faults of first statement with that of the second. In general, as long as each statement specifies only one fault, the plus sign will combine into one fault, the faults of all of the statements with the statement immediately preceding them. In the case that more than one fault is specified per lead the first fault of a statement with a plus sign is combined with the last fault of the statement ahead.

Consider the following fault descriptions for CIRCUIT1.

e.g. 1. A1 s @ 1, s @ 0; + N1.s @ 1, s @ 0/X1,X2;

   + A2 s @ 0/X1,A1;

The selected faults list would be

1. A1 s @ 1.
2. A1 s @ 0 and N1 s @ 1.
3. N1 s @ 0.
4. N1 open input diode from X1.
5. N1 open input diode from X2 and A2 s @ 0.
6. A2 open input diode from X1.
7. A2 open input diode from A2.

e.g. 2. A1 s @ 1; + N1 s @ 1, s @ 0/X1,X2;

   A2 s @ 0/X1;
The selected faults list would be

1. \( A_1 \ s @ 1 \) and \( N_1 \ s @ 1 \).
2. \( N_1 \ s @ 0 \).
3. \( N_1 \) open input diode from \( X_1 \).
4. \( N_1 \) open input diode from \( X_2 \).
5. \( A_2 \ s @ 0 \).
6. \( A_2 \) open input diode from \( X_1 \).

e.g. 3. \( A_1 \ s @ 1; +N_1 \ s @ 1; +A_2 \ s @ 1; +\emptyset_1 \ s @ 0; \)

The selected faults list would be

1. \( A_1 \ s @ 1 \) and \( N_1 \ s @ 1 \) and \( A_2 \ s @ 1 \) and \( \emptyset_1 \ s @ 0 \).

At the end of any selected faults list the keyword END, followed by a semicolon, must appear.
SECTION 7) **Control Cards for FAUST**

The following deck is an example of the necessary control cards used to make a FAUST run on a system 360 under OS/360.

```plaintext
//FAUST1 JOB 'P0785SYSTRUC,TIME=(,59),PAGES=999',CLASS=A,MSGLEVEL=1
//JOBLIB DD DSN=PO785.LMAN2,DISP=SHR
//GO EXEC PGM=FAUST,REGION=250K
//SYSPRINT DD SYSOUT=A
//GO.RECDIR DD DSN=U0664.RECDIR,DISP=OLD
//GO.RECFILE DD DSN=U0664.RECFILE,DISP=OLD
//SYSIN DD *

Place your FAUST deck here.

/*

*/
SECTION 8) FAUST Examples

The following examples of FAUST runs are provided to help the user become more familiar with the FAUST language. The examples are runs on CIRCUIT1. These FAUST statements

```plaintext
ZSAMP2: SIMULATE CIRCUIT=CIRCUIT1,DISP=NO,LIST=YES,FAULTS=ALL;
    INITREE;
    ACTIVATE ALL;
    INIFault;
    RUN; SHOW ALL ;
    SET X1;
    TRACE A2,Ø1;
    RUN; SORT;
    TREE;
    BACKUP RECNAME=TEST;
    TRACE N1;
    FLIP X1,X2;
    RUN; SORT;
    UNTRACE Ø1;
    RUN; SORT;
    TREE;
    END;
```

give the following output:
EXAMPLE 2

CPU TIME= 260.460.000; TOTAL CPU TIME= 0.000 MILLISECONDS.

%SAMP2: SIMULATE CIRCUIT=CIRCUIT1, DISP=NG, LIST=YES, FAULTS=ALL; COO80000SOURCE.
CPU TIME=   142.490; TOTAL CPU TIME= 142.490 MILLISECONDS.

==LCID==CIRCUIT NAME: CIRCUIT1, OPTIONS: GOOC
FAULTS OPTION: ALL

==FCIR==CLD DIRECTORY Fetched.

===== CIRCUIT1 CIRCUIT RECORDS =====

CIRCUIT1 NODE NO.  79
$CIDS  NODE NO.  71  86 BYTES, STARTING REGION NO.  12 WORD NO.  525
$LDSC  NODE NO.  78 168 BYTES, STARTING REGION NO.  12 WORD NO.  547
$FMCS  NOT FOUND.
$SYMTA NODE NO.  81 144 BYTES, STARTING REGION NO.  12 WORD NO.  492
$REFS  NODE NO.  82  56 BYTES, STARTING REGION NO.  12 WORD NO.  589
$SLCS  NODE NO.  83  75 BYTES, STARTING REGION NO.  0 WORD NO.  2C
$TERM  NOT FOUND.
### STRUCTURES LOCATIONS ####

<table>
<thead>
<tr>
<th>Structure</th>
<th>Description</th>
<th>Size</th>
<th>Region No.</th>
<th>Word No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SCIDS</td>
<td>CF NODE NO. 71, FETCHED</td>
<td>86 bytes</td>
<td>12</td>
<td>525</td>
</tr>
<tr>
<td>$LOSC</td>
<td>OF NODE NO. 78, FETCHED</td>
<td>168 bytes</td>
<td>12</td>
<td>547</td>
</tr>
<tr>
<td>$SYMTA</td>
<td>CF NODE NO. 81, FETCHED</td>
<td>124 bytes</td>
<td>12</td>
<td>492</td>
</tr>
<tr>
<td>$REFS</td>
<td>OF NODE NO. 82, FETCHED</td>
<td>56 bytes</td>
<td>12</td>
<td>589</td>
</tr>
<tr>
<td>$SUICS</td>
<td>OF NODE NO. 83, FETCHED</td>
<td>75 bytes</td>
<td>0</td>
<td>20</td>
</tr>
</tbody>
</table>

EV=01101110 R=110;

===FETCH===

<table>
<thead>
<tr>
<th>Structure</th>
<th>Description</th>
<th>Size</th>
<th>Region No.</th>
<th>Word No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SCOPT</td>
<td>OF NODE NO. 60, FETCHED</td>
<td>2612 bytes</td>
<td>0</td>
<td>114</td>
</tr>
</tbody>
</table>

#### SINGLE FAULT CIRCUITS LISTING ####

<table>
<thead>
<tr>
<th>No.</th>
<th>NAME</th>
<th>FAULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>FAULT FREE CIRCUIT.</td>
</tr>
<tr>
<td>2</td>
<td>A1</td>
<td>OUTPUT STUCK AT ZERO</td>
</tr>
<tr>
<td>3</td>
<td>A1</td>
<td>OUTPUT STUCK AT ONE</td>
</tr>
<tr>
<td>4</td>
<td>A1</td>
<td>OPEN INPUT DIODE FROM X1</td>
</tr>
<tr>
<td>5</td>
<td>A1</td>
<td>OPEN INPUT DIODE FROM X2</td>
</tr>
<tr>
<td>6</td>
<td>N1</td>
<td>OUTPUT STUCK AT ZERO</td>
</tr>
<tr>
<td>7</td>
<td>N1</td>
<td>OUTPUT STUCK AT ONE</td>
</tr>
<tr>
<td>8</td>
<td>N1</td>
<td>OPEN INPUT DIODE FROM X1</td>
</tr>
<tr>
<td>9</td>
<td>N1</td>
<td>OPEN INPUT DIODE FROM X2</td>
</tr>
<tr>
<td>10</td>
<td>A2</td>
<td>OUTPUT STUCK AT ZERO</td>
</tr>
<tr>
<td>11</td>
<td>A2</td>
<td>OUTPUT STUCK AT ONE</td>
</tr>
<tr>
<td>12</td>
<td>A2</td>
<td>OPEN INPUT DIODE FROM X1</td>
</tr>
<tr>
<td>13</td>
<td>A2</td>
<td>OPEN INPUT DIODE FROM A1</td>
</tr>
<tr>
<td>14</td>
<td>U1</td>
<td>OUTPUT STUCK AT ZERO</td>
</tr>
<tr>
<td>15</td>
<td>U1</td>
<td>OUTPUT STUCK AT ONE</td>
</tr>
<tr>
<td>16</td>
<td>U1</td>
<td>OPEN INPUT DIODE FROM A1</td>
</tr>
<tr>
<td>17</td>
<td>U1</td>
<td>OPEN INPUT DIODE FROM N1</td>
</tr>
</tbody>
</table>

**CIRCUIT:** $SCIDS, $FMC

---

**STORE:**

<table>
<thead>
<tr>
<th>Structure</th>
<th>Description</th>
<th>Size</th>
<th>Region No.</th>
<th>Word No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SCIDS</td>
<td>CF NODE NO. 71, STUCKED</td>
<td>86 bytes</td>
<td>12</td>
<td>525</td>
</tr>
<tr>
<td>$SCIDS</td>
<td>OF NODE NO. 98, STORED</td>
<td>132 bytes</td>
<td>12</td>
<td>450</td>
</tr>
</tbody>
</table>

**STORE:**

<table>
<thead>
<tr>
<th>Structure</th>
<th>Description</th>
<th>Size</th>
<th>Region No.</th>
<th>Word No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FMC</td>
<td>CF NODE NO. 98, STORED</td>
<td>132 bytes</td>
<td>12</td>
<td>450</td>
</tr>
</tbody>
</table>
CIRCUIT SIMULATION TABLE

6 LEADS.
2 OUTPUTS : 01, A2
2 INPUTS : X1, X2
0 FEEDBACKS:
17 SINGLE FAULTS.
12 SUCCESSORS IN SS.
0 TERMINALS.
6 SYMBOLIC NAMES.
0 FALST RUNS.

3 LOGIC LEVELS.

<table>
<thead>
<tr>
<th>NO</th>
<th>NAME</th>
<th>ATOM</th>
<th>LVL</th>
<th>INPUT</th>
<th>REFERENCES</th>
<th>FANOUT</th>
<th>OUTPUT REFERENCES</th>
<th>ASSOCIATED FAULTS</th>
<th>NO. S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X1</td>
<td>INP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>X2</td>
<td>INP</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>A1</td>
<td>AND</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>N1</td>
<td>NAND</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>A2</td>
<td>AND</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td></td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>C1</td>
<td>OR</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td></td>
<td>14</td>
<td>0</td>
</tr>
</tbody>
</table>

==FETC==FALST. &OPTAB
STRUCTURE: &OPTAB OF NODE NO. 35, FETCHED 324 BYTES, STARTING REGION NO. 10 WORD NO. 408
CPU TIME= 1.591.593, TOTAL CPU TIME= 1.444.073 MILLISECONDS.
==== FAULST STATEMENTS ==== 
= DIAGNOSTIC TREE INITIALIZED. 
CPU TIME= 13.332, TOTAL CPU TIME= 1,457.495 MILLISECONDS.

0>
CPU TIME= 2.499, TOTAL CPU TIME= 1,459.904 MILLISECONDS.

INITREE;
= DIAGNOSTIC TREE INITIALIZED. 
CPU TIME= 98.327, TOTAL CPU TIME= 1,558.231 MILLISECONDS.

1>
CPU TIME= 3.333, TOTAL CPU TIME= 1,561.564 MILLISECONDS.

ACTIVATE ALL;

2>
CPU TIME= 72.495, TOTAL CPU TIME= 1,634.059 MILLISECONDS.

INIFALT;
==INIFL==CCNE.

3>
CPU TIME= 79.954, TOTAL CPU TIME= 1,714.053 MILLISECONDS.

RUN; SHOW ALL;

PRODCGATION: 1, 2, 3, 4, 5, 6,
==DIMLY== CCNE.

4>
CPU TIME= 116.659, TOTAL CPU TIME= 1,830.712 MILLISECONDS.
1

NG. NAME 012345678901234567
-4 CCCCCCCCCCCCCCCCCC
-3 CCCCCCCCCCCCCCCCCC
-2 CCCCCCCCCCCCCCCCCC
-1 CCCCCCCCCCCCCCCCCC
0 1111110111111111
1 X1 0 CCCCCCCCCCCCCCCCCC
2 X2 0 CCCCCCCCCCCCCCCCCC
3 A1 0 CCCCCCCCCCCCCCCCCC
4 N1 0 1111110111111111
5 A2 0 CCCCCCCCCCCCCCCCCC
6 O1 0 1111111111111111

5>

CPU TIME= 80.828, TOTAL CPU TIME= 1,911.546 MILLISECONDS.

SET X1;
6>

CPU TIME= 69.995, TOTAL CPU TIME= 1,981.535 MILLISECONDS.

TRACE A2, O1;
7>

CPU TIME= 72.495, TOTAL CPU TIME= 2,054.030 MILLISECONDS.

RUN; SURT;

PROPAGATION: 3, 4, 5, 6,

==DIMUY== DONE.

1

NG. NAME 012345678901234567
5 A2 0 0010100101000000
6 O1 0 1111110111111111
8>

CPU TIME= 110.826, TOTAL CPU TIME= 2,164.856 MILLISECONDS.

00130000SOURCE.
00140000SOURCE.
00150000SOURCE.
CPU TIME = 50.830, TOTAL CPU TIME = 2,215.686 MILLISECONDS.

TREE;
CPU TIME = 88.327, TOTAL CPU TIME = 2,304.913 MILLISECONDS.

==TREE== FL = 0 HS = 0 LL = 1
FP = 18 HP = 0 LP = 19;
LEVEL# 0 3 NODES, 1 SUBSETS.

===== THE DIAGNOSTIC TREE =====

RESOLUTIONS:

***** ***** *****
LEVEL# 0 *001*---*C02*---*C03* 3DIA. * 2 *---* 3 *---* C *
41.1%DTC. ***** ***** ******

MACHINE# 1# 3# 6# 2# 5# 9# 4# 11# 14# 7# 13#

OVER ALL 12.5%DIA. 8# 10# 12# 15# 16# 17#

CPU TIME = 142.490, TOTAL CPU TIME = 2,446.503 MILLISECONDS.
BACKUP  RECNAM=TEST;

==FETCH==CIRCUIT1.TEST.SLVBS
STRUCTURE:  $LVBS  OF  NODE  NO.  86,  FETCHED  74  BYTES,  STARTING  REGION  NO.  12  WORD  NO.  699

==FETCH==CIRCUIT1.TEST.$DIATR
STRUCTURE:  $DIATR  OF  NODE  NO.  97,  FETCHED  347  BYTES,  STARTING  REGION  NO.  12  WORD  NO.  966

CPU  TIME= 161.656,  TOTAL  CPU  TIME= 2,608.159  MILLISECONDS.

TRACE N1;

CPU  TIME= 107.493,  TOTAL  CPU  TIME= 2,715.652  MILLISECONDS.

FLIP  X1,X2;

CPU  TIME= 74.161,  TOTAL  CPU  TIME= 2,789.813  MILLISECONDS.

RUN;  SURF;

PROPAGATION: 3, 4, 5, 6,

==DIMUY==  DCNE.

<table>
<thead>
<tr>
<th>NC.</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>N1</td>
</tr>
<tr>
<td>5</td>
<td>A2</td>
</tr>
<tr>
<td>6</td>
<td>01</td>
</tr>
</tbody>
</table>

CPU  TIME= 112.492,  TOTAL  CPU  TIME= 2,902.305  MILLISECONDS.

CPU  TIME= 52.495,  TOTAL  CPU  TIME= 2,954.801  MILLISECONDS.
UNTRACE CI;

CPU TIME= 95.827, TOTAL CPU TIME= 3,050.628 MILLISECONDS.

RUN: SORT;

PROPAGATION:

==DIMLY== CCNE.

<table>
<thead>
<tr>
<th>NO.</th>
<th>NAME</th>
<th>012345678901234567</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>N1</td>
<td>111111010111111111</td>
</tr>
<tr>
<td>5</td>
<td>A2</td>
<td>000000000010000000</td>
</tr>
</tbody>
</table>

CPU TIME= 104.993, TOTAL CPU TIME= 3,155.621 MILLISECONDS.

CPU TIME= 54.163, TOTAL CPU TIME= 3,209.784 MILLISECONDS.

TREE:

CPU TIME= 90.827, TOTAL CPU TIME= 3,300.611 MILLISECONDS.

==TREE== FL= 0 HS= 0 LL= 5
FP= 18 HP= 0 LP= 19;

LEVEL# 0 1 NODES, 1 SUBSETS.
LEVEL# 1 3 NODES, 1 SUBSETS.
LEVEL# 2 4 NODES, 2 SUBSETS.
LEVEL# 3 4 NODES, 3 SUBSETS.
LEVEL# 4 3 NODES, 3 SUBSETS.
The presence of LIST=YES in the control card statement causes the circuit description of CIRCUIT1 to be printed.

The initial sort was not done in this run. Note the affect on the diagnostic tree; it does not have a singular root.

The $LVBS was not printed after the first RUN statement. The reason is that a TRACE statement must appear before the automatic display appears after each run.

The statement

```
BACKUP RECNAM=TEST;
```

fetches the structures TEST.$LVBS and TEST.$DIATR into core, wiping out the previous $LVBS and $DIATR. The final tree is a 'super tree' of that printed in run SAMP1.

The next example simulates CIRCUIT1 under multiple faults. The statements are:

```
%SAMP3:  SIMULATE CIRCUIT=CIRCUIT1,DISP=NO,FAULTS=GET;
  A1 S@1,S@0;
  + N1 S@1,S@0/X1,X2;
  +  A2 S@0/X1,A1;
  END;
  SORT;
  TRACE Ø1,A2;
  ACTIVATE ALL;
  SET X1; RUN; SORT;
  SET X2; RUN; SORT;
  TRACE ALL;
  RESET (1,3 to 5) X2;
  RUN; SORT;
  END;
```

The resulting output is:
EXAMPLE 3

CPU TIME = 260,400,000, TOTAL CPU TIME = 0.000 MILLISECONDS.

%SAMP3: SIMULATE CIRCUIT=CIRCUIT1, DISP=NO, FAULTS=GET;
CPU TIME = 145.824, TOTAL CPU TIME = 145.824 MILLISECONDS.

==LCID==CIRCUIT NAME: CIRCUIT1, OPTIONS: 0000
FAULTS OPTION: GET
==FDIR==DIR EXPANDED FROM 136 TO 141 NODES.
==FDIR==OLD DIRECTORY FETCHED.

====== CIRCUIT1 CIRCUIT RECORDS ======

CIRCUIT1 NODE NO. 79
$SCIDS NODE NO. 71 86 BYTES, STARTING REGION NO. 12 WORD NO. 525
$LDSC NODE NO. 78 168 BYTES, STARTING REGION NO. 12 WORD NO. 547
$FMCS NODE NO. 84 132 BYTES, STARTING REGION NO. 12 WORD NO. 450
$SYMTA NODE NO. 81 124 BYTES, STARTING REGION NO. 12 WORD NO. 492
$SREFS NODE NO. 82 56 BYTES, STARTING REGION NO. 12 WORD NO. 589
$SUUGS NODE NO. 83 75 BYTES, STARTING REGION NO. 0 WORD NO. 20
$TERM NOT FOUND.
====== STRUCTURES LOCATIONS ======
STRUCTURE: $CIDS OF NODE NO. 71, FETCHED 86 BYTES, STARTING REGION NO. 12 WORD NO. 525
STRUCTURE: $LDSC OF NODE NO. 78, FETCHED 168 BYTES, STARTING REGION NO. 12 WORD NO. 547
STRUCTURE: $SYMTA OF NODE NO. 81, FETCHED 124 BYTES, STARTING REGION NO. 12 WORD NO. 492
STRUCTURE: $REFS OF NODE NO. 82, FETCHED 56 BYTES, STARTING REGION NO. 12 WORD NO. 589
STRUCTURE: $SUCS OF NODE NO. 83, FETCHED 75 BYTES, STARTING REGION NO. 0 WORD NO. 2C

EY=01101110*8 R= 110;

==FETCH==*$COPT
STRUCTURE: $COPT OF NODE NO. 60, FETCHED 2610 BYTES, STARTING REGION NO. 0 WORD NO. 114

====== SELECTED FAULTS EVALUATION ======
FRNM = CIRCUIT1.$CTRES.$AMP3.$SFMC

FAULTS DESCRIPTION SOURCE:
A1 S@1, S@0;
+ N1 S@1,S@0,X1,X2;
+ A2 S@0,X1,A1;
END;

==FLEV== 3 FAULTY LEADS.

==FETCH==FAUST.$OPTAB
STRUCTURE: $OPTAB OF NODE NO. 35, FETCHED 324 BYTES, STARTING REGION NO. 12 WORD NO. 408
CPU TIME= 1,674.059, TOTAL CPU TIME= 1,819.883 MILLISECONDS.
==== FAUST STATEMENTS =====
= DIAGNOSTIC TREE INITIALIZED.
CPU TIME= 10.832, TOTAL CPU TIME= 1,830.715 MILLISECONDS.
0>
CPU TIME= 3.333, TOTAL CPU TIME= 1,834.048 MILLISECONDS.
SURT;
1>
CPU TIME= 125.825, TOTAL CPU TIME= 1,959.873 MILLISECONDS.
TRACE 01,A2;
2>
CPU TIME= 100.826, TOTAL CPU TIME= 2,060.699 MILLISECONDS.
ACTIVATE ALL;
3>
CPU TIME= 69.995, TOTAL CPU TIME= 2,130.694 MILLISECONDS.
SET X1; RUN; SORT;
4>
CPU TIME= 68.328, TOTAL CPU TIME= 2,199.022 MILLISECONDS.
PROPOGATION: 1, 2, 3, 4, 5, 6,
==DIMUY== DONE.
NO. NAME 012345678
   5 A2  0 10011000
   6 01  0 11111111

5> CPU TIME= 54.996, TOTAL CPU TIME= 2,254.018 MILLISECONDS.

6> CPU TIME= 51.663, TOTAL CPU TIME= 2,305.681 MILLISECONDS.
   SET X2; RUN; SORT;
   7> CPU TIME= 89.166, TOTAL CPU TIME= 2,394.841 MILLISECONDS.

PROPOGATION: 3, 4, 5, 6,

==DIMUY== DONE.

NO. NAME 012345678
   5 A2  0 11011101
   6 01  0 11011111

8> CPU TIME= 54.163, TOTAL CPU TIME= 2,449.034 MILLISECONDS.

9> CPU TIME= 49.996, TOTAL CPU TIME= 2,499.030 MILLISECONDS.
   TRACE ALL;
10> CPU TIME= 89.166, TOTAL CPU TIME= 2,588.166 MILLISECONDS.
RESET (1, 3 TO 5) X2;

11>
CPU TIME: 77.495, TOTAL CPU TIME: 2,665.655 MILLISECONDS.

RUN: SORT:

PROPGATION: 3, 4, 5, 6,

==DIMUY== DONE.

<table>
<thead>
<tr>
<th>NO.</th>
<th>NAME</th>
<th>012345678</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>000000000</td>
<td></td>
</tr>
<tr>
<td>-3</td>
<td>000000000</td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td>000000000</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>000000000</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>000000000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X1</td>
<td>011111111</td>
</tr>
<tr>
<td>2</td>
<td>X2</td>
<td>000000000</td>
</tr>
<tr>
<td>3</td>
<td>A1</td>
<td>010000000</td>
</tr>
<tr>
<td>4</td>
<td>N1</td>
<td>000000000</td>
</tr>
<tr>
<td>5</td>
<td>A2</td>
<td>000000000</td>
</tr>
<tr>
<td>6</td>
<td>O1</td>
<td>000000000</td>
</tr>
</tbody>
</table>

12>
CPU TIME: 129.158, TOTAL CPU TIME: 2,794.813 MILLISECONDS.

13>
CPU TIME: 53.329, TOTAL CPU TIME: 2,848.142 MILLISECONDS.

END;
For a detailed description of the selected faults for this run, see section 6), example 1 of this manual.

SECTION 9) FAUST program error diagnostics

The error messages of FAUST source errors are self explanatory.

Here is output of an incorrect FAUST source deck.
EXAMPLE 4

CPU TIME= 260.400.000, TOTAL CPU TIME= 0.000 MILLISECONDS.

% SAMP4: SIMULATE CIRCUIT=CIRCUIT1, DISP=NC, FAULTS=ALL;
CPU TIME= 138.324, TOTAL CPU TIME= 138.324 MILLISECONDS.

==LCID==CIRCUIT NAME: CIRCUIT1, OPTIONS: 0000
FAULTS OPTION: ALL
==FDIR==GIR EXPANDED FROM 141 TO 143 NODES.

==FDIR==OLD CIRECTORY FETCHED.

====== CIRCUIT1 CIRCUIT RECORDS ======

CIRCUIT1 NODE NO. 79
$CIDUS NODE NO. 71 86 BYTES, STARTING REGION NO. 12 WORD NO. 525
$SLCSC NODE NO. 78 168 BYTES, STARTING REGION NO. 12 WORD NO. 547
$FMCS NODE NO. 98 132 BYTES, STARTING REGION NO. 12 WORD NO. 450
$SYMIA NODE NO. 81 144 BYTES, STARTING REGION NO. 12 WORD NO. 492
$REFS NODE NO. 82 56 BYTES, STARTING REGION NO. 12 WORD NO. 589
$SUCS NODE NO. 83 75 BYTES, STARTING REGION NO. 0 WORD NO. 20
$TERM NOT FOUND.
| STRUCTURE: $CID | CF NODE NO. | 71, FETCHED | 66 BYTES, STARTING REGION NO. 12 WORD NO. 525 |
| STRUCTURE: $LDSC | OF NODE NO. | 78, FETCHED | 168 BYTES, STARTING REGION NO. 12 WORD NO. 547 |
| STRUCTURE: $FMCS | OF NODE NO. | 98, FETCHED | 132 BYTES, STARTING REGION NO. 12 WORD NO. 450 |
| STRUCTURE: $SYMTA | OF NODE NO. | 81, FETCHED | 124 BYTES, STARTING REGION NO. 12 WORD NO. 492 |
| STRUCTURE: $REFS | OF NODE NO. | 82, FETCHED | 56 BYTES, STARTING REGION NO. 12 WORD NO. 589 |
| STRUCTURE: $SCUCS | OF NODE NO. | 83, FETCHED | 75 BYTES, STARTING REGION NO. 0 WORD NO. 20 |

$EV=#01111110'B  $R=126$

$== FETCH ==$CCPT
STRUCTURE: $CCPT | CF NODE NO. | 60, FETCHED | 2610 BYTES, STARTING REGION NO. 0 WORD NO. 114

$== FETCH ==$ALST.$UPTAB
STRUCTURE: $UPTAB | OF NODE NO. | 35, FETCHED | 324 BYTES, STARTING REGION NO. 10 WORD NO. 408
CPU TIME= 1,009.935, TOTAL CPU TIME= 1,148.259 MILLISECONDS.

$== FALST STATEMENTS ===
= DIAGNOSTIC TREE INITIALIZED
CPU TIME= 13.332, TOTAL CPU TIME= 1,161.591 MILLISECONDS.

$C>
CPU TIME= 2.499, TOTAL CPU TIME= 1,164.090 MILLISECONDS.

ACTIVATE:
* INVALID DELIMITER: : STATEMENT FLUSHED FROM: ACTIVATE
TRACE
SFTX1:

$1>
CPU TIME= 222.485, TOTAL CPU TIME= 1,336.575 MILLISECONDS.
RUN;

PROPAGATION:

==DIMLY== CCN6.

CPU TIME= 84.161, TOTAL CPU TIME= 1,470.736 MILLISECONDS.

RESET3;
* INVALID OPERATOR: RESET3 STATEMENT FLUSHED FROM: RESET3

CPU TIME= 95.827, TOTAL CPU TIME= 1,566.563 MILLISECONDS.

TREE RCCT=5;
CPU TIME= 63.329, TOTAL CPU TIME= 1,629.892 MILLISECONDS.

**TREE** HS= 5 0 ASSUMED.

==TREE== FL= 0 HS= 5 LL= 1
FP= 18 HP= 0 LP= 19;

**CLNT** INVALID PARM. P= 0

IHE5001 SUBSCRIPT RANGE IN STATEMENT CC105 AT OFFSET +014A8 FROM ENTRY POINT TREE

CONDITION ERR. OCCURRED IN STATEMENT CC105 AT OFFSET +014A8 FROM ENTRY POINT TREE

CALLED, IN STATEMENT CC250, FROM PROCEDURE WITH ENTRY POINT FAUST

**FAUST** FAILED.
The first error is the use of a colon instead of a semicolon. The second is due to the absence of a blank between SET and X1. Note however another error has been missed due to the first two. There should be a semicolon after TRACE.

The next error causes the program to halt. Here we have asked to have the diagnostic tree printed with the root at the 5th node on the first level. Since there is no fifth node the instruction fails and causes a PLI error.
ERROR MESSAGE GLOSSARY

For control card errors (i.e. errors of the form **CCINT) see the section on error messages in the TRAIZE user's guide.

**FAUST**FAILED

- if this message is not accompanied with any of the messages below the system has failed.

*INVALID DELIMITER. STATEMENT.

FLUSHED FROM: ________________________________

- check for missing semicolon etc.

*INVALID LEAD#: ________________________________

- lead number is too large i.e. greater than the number of leads.

*INVALID MACHINE#: ____________________________

- faulty machine number is greater than the number of faulty machines.

*INVALID NAME: ________________________________

- check valid lead names.

- remember a lead number must be preceded by a '#' sign.

*INVALID OPERAND. STATEMENT FLUSHED FROM: ____________

- check spelling of the instruction word, also, use of qualifiers.

*SURPLUS DELIMITER: ____________________________

- possible blank leadname.
APPENDIX A

Quick Reference

Statement

Guide
DECK SETUP

% runtime: SIMULATE CIRCUIT = circuitname

[ N ]
,FAULTS=ALL
,DISP=OLD
runname_

[ NO ]
[ YES ]
[ OLD ]
[ NEW ]

Faults definition statements iff FAULTS=GET,

.
.
.

END;

FAUST test description, simulation and printout statements,

.
.
.

END;

SEMANTICS FOR CONTROL CARD

runname - A symbolic name by which the results of this run would be identified. If DISP=NEW it must be different from any other run name of the simulated circuit, if DISP=OLD it must be a known name; otherwise an error will occur.

circuitname - The name of the circuit to be simulated. It must be a known name of a circuit whose description can be found in the DIALATOR file, or in core.

FAULTS - Defining faults to be used for this run:

NO - no faults (default value)

ALL - all faults

GET - following is a set of fault definition statements to be compiled into a selected faults list and to be saved in the DIALATOR file under the runname specified on this card.
runname - use the selected faults list which is stored under this runname in the DIALATOR file.

DISP - The disposition of the run results:

NO - do not save the run results.

YES - save the run results.

NEW - create new entries in the file for the run results.

OLD - overwrite already existing records of run results with the same name.
DEFINITIONS of SYNTACTIC PHRASES

In the following definitions of FAUST and Faults Definition Statements the following syntactic phrases will be used:

leadname ::= Any valid symbolic name (6 characters) referring to a known lead.

leadno ::= Any valid lead number [0 < N#] immediately preceded by a '#' symbol. (e.g. #34)

$1list ::= (leadname | leadno) [,leadname leadno, ,] |¥

$leads ::= ALL | [INPUTS | OUTPUTS | FBKS |$1list , , ,]

$1list and $leads are used for describing lists of leads referred to by their symbolic names, index numbers in the Description tables or by their group names inputs outputs or feedbacks. A $leads containing the member ALL will refer to all the circuit leads. (e.g. X23, IN5, #17, #78, INPUTS ).

machino ::= Any valid [0 < M#] faulty machine number.

$mlist ::= (machino | [machino TO machino], ,) |¥

$mlist is used for describing a list of faulty machines. It can be specified as a list of machine numbers separated by ',', or by specifying the range of the faulty machines number by using the 'TO' option. $mlist is always enclosed in '()' and its absence '¥' is always understood as ALL faulty machines known to the system at that time. (e.g. (12, 23, 24 TO 32, 35 TO 36) ).
$flist ::= \text{ALL} | \{[S@1],[S@0],[OFD]\} | \emptyset$

$flist$ is a list of fault types: stuck at one, stuck at zero and open feedback diode. ALL designates all the faults of the subject lead while a $\emptyset$ means none of these faults.
Faults Definition Statements

[+] leadname [ALL|$f$list]/\{ALL |$l$list\};
   - to define selected faults.

Faults Simulation Statements

\{SET|RESET|FLIP\}[m\list]leads[ACTIVE|UNACTIVE ;
   - to set, reset, or flip leads.

\{TRACE|UNTRACE|SHOW\} [\cdot|$\ leads];
   - to trace, untrace, or show leads.

\{SAVE|BACKUP\} [RECN\AME=rec\name];
   - to save or fetch in or from DIAL\ATOR FILE

RUN[FAULTS=\NO|\ALL],[MAXIT=max\it];
   - to simulate circuit

SORT;
   - to look at output, sort, add level to tree

TREE [FIRST=f\L],[ROOT=hs],[LAST=l\L];
   - to show diagnostic tree

INITREE [UB=upperbound on the tree];
   - to scratch old tree and initialize new one.

INIF\AULT;
   - to initialize $LVBS$ with strict at faults.

\{ACTIVATE|DEACTIVATE\}.|$leads;
   - to set or reset activity bits of leads.