Test Generation System (TGS)
User's Manual — Version 1.1

Kuen-Jong Lee Melvin A. Breuer

Department of Electrical Engineering-Systems
University of Southern California

Technical Report No. CENG 89-03
June 27, 1988

Preface

This manual describes the operation of a suite of programs used to generate test for combinational logic networks. Currently, the main modules in this suite consist of a test pattern generator, a fault simulator, a good circuit simulator, and a fault collapser. The test generator was obtained from the University of Nebraska, Lincoln. All other modules were developed at the University of Southern California. Rajesh Gupta wrote the fault collapser and its documentation (Appendix of this manual). Kayhan Kucukcakar and Rajiv Gupta developed the fault simulator and the logic simulator. Kuen-Jong Lee was responsible for integrating the various modules into a single system and writing this manual. Part of this work was supported by the Defense Advanced Research Projects Agency under contract no. N00014-84-K-0649.

Any question or comment please direct to:

Professor Melvin A. Breuer
Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089-0781
(213) 743-2308
E-mail: mb@usc-cse.usc.edu

or

Kuen-Jong Lee
Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089-0781
(213) 743-3794
E-mail: kjlee@usc-cse.usc.edu

Melvin A. Breuer
Project Director

Contents

[1] Written by Rajesh Gupta

1 General Description

This manual describes the Test Generation System (TGS) developed by the USC Test Group. The organization of this document is as follows. Section 1 gives a general description of the system. Each function provided by the system is described. The system environment and the default values of several system parameters such as the maximum number of gates, the maximum number of fanouts, etc., are also given. Section 2 summarizes the input/output specifications. The exact format of each file used in the system is described, followed by the input/output requirements and the user interface for each function. Section 3 illustrates the system through an example circuit containing 38 gates.

1.1 System Functions

TGS is designed for generating test vectors for combinational circuits described at the gate level. The gate types supported by the system are AND, OR, NAND, NOR, INV(inverter), BUF(buffer) and INPT(input gate). The single stuck-at-0 and stuck-at-1 fault model is assumed. At present, the system provides the following functions:

1. Fault collapsing
2. Test vector generation
3. Fault simulation
4. Logic simulation
5. Integration of 1), 2), 3) to derive a complete set of test patterns.

A brief description of each function is next.

1.1.1 Fault Collapsing

The main objective of fault collapsing is to classify the set of all possible faults. Typically, a test for an arbitrary fault detects several other faults in the circuit. Two faults F1 and F2 are said to be equivalent if any test for F1 also detects F2, and vice versa. A fault F2 is said to dominate F1 if all tests for F1 detect F2. Under the stuck-at fault model, all lines (gate output stems and fanout branches are considered separately) in a circuit can have two possible faults, viz., stuck-at-1 and stuck-at-0. Hence the total number of possible single faults is equal to twice the number of lines. The result of fault collapsing is a set of

fault classes, each of which consists of two sets – an **equivalence set** and a **dominance set** – which are defined below.

A test vector which can detect any fault in an equivalence set will also detect all other faults in the same set. A dominance set contains all the faults which dominate the faults in the corresponding equivalence set, i.e., a test vector which detects a fault in an equivalence set will also detect all the faults in the corresponding dominance set. However, the converse is not true. An equivalence set is said to be **prime** if it is not a subset of any other fault class. Otherwise it is **non-prime**.

The objective of fault collapsing is to organize the set of all possible faults into maximal fault classes such that every fault is present in the equivalence set of exactly one fault class. There are two fault collapsing approaches: equivalence merging and dominance merging, which are based on the equivalence and dominance relationships among faults. Fault collapsing is described in detail in Appendix A.

1.1.2 Test Vector Generation

The test vector generation process provides a test vector (consisting of $0, 1$ or $\times$ (don't care) entries) for any given detectable fault. The system currently uses the **PODEM** test generation algorithm [GOE 81] adopted from Bonebrake's work at the University of Nebraska, Lincoln, with modifications by Kosch and Hudli.

PODEM is a complete test generation algorithm, i.e., given enough time, it will eventually find a test vector if the fault is detectable. However, for some hard to detect faults the algorithm requires exponential search times. A time-out mechanism is provided in the system to avoid such prohibitive searches.

1.1.3 Fault Simulation

Fault simulation attempts to identify all faults that can be detected by a given input vector. It provides a list of faults and an associated primary output with the following property. If any fault in this list is injected in the circuit, then the logic values of the good circuit and the faulty circuit will differ at the associated primary output under the given input vector. We use the **Critical Path Tracing** method developed by Abramovici et al. [ABR 84].

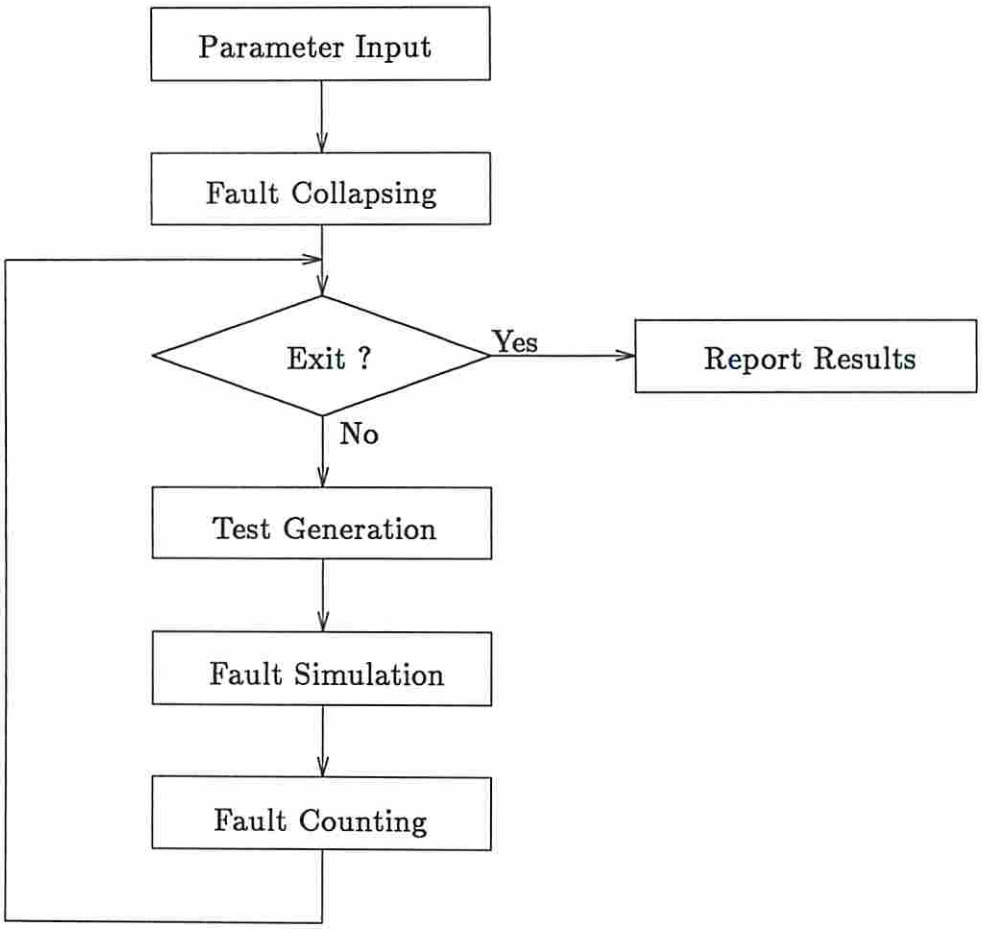

Figure 1: The Integrated System

1.1.4 Logic Simulation

Critical Path Tracing for fault simulation requires logic simulation of the good circuit. We have separated out the good circuit simulator from the fault simulator and it can be executed in a stand alone mode.

1.1.5 Integrated System

Besides the functions described above, an integrated system which combines fault collapsing, test generation and fault simulation into one complete test system is provided. The purpose of this system is to execute a complete test generation procedure without user's intervention, once the required input parameters are set up. Figure 1 shows the flow chart of the integrated system.

The integrated system functions in two phases: initialization phase and test generation/evaluation phase. During initialization, the user is asked to select options from several menus and provide all the relevant file names. The system then determines all possible single stuck-at-faults in the input circuit description file and collapses them into fault classes.

The test generation/evaluation phase is a loop consisting of test vector generation, fault simulation and fault counting. Depending on the user-specified test vector generation method, it can be executed in three different ways. The user can either use algorithmically generated test vectors (PODEM in the current system), random test vectors, or prestored test vectors (in a file). In the first case, a fault selection procedure must be invoked to iteratively select an undetected fault for the test vector generator.

Once a test vector is obtained (see Test Generation in Figure 1), all faults that can be detected by this test vector are identified (Fault Simulation). Then the newly detected faults are counted and the new fault coverage is computed (Fault Counting).

The program exits this loop whenever the user-specified requirement is satisfied, or when no more untested faults are left. The results of execution are then written to several output files.

The integrated system is menu-driven. The user must select the exit condition, test-generation method and fault-selection method (if using algorithmic test vector generation) from the corresponding menu. The exit condition can be a user-specified fault coverage or based on the number of test vectors. Test vectors can be generated by PODEM, randomly or prestored in a file. Currently, only in-order selection method (described below) is used to select a fault. The program, however, can be easily modified to provide other alternatives.

The in-order fault-selection procedure is as follows. The integrated system first selects a fault from the prime equivalence sets. If the required fault coverage is not achieved after all prime equivalence sets have been processed (due to the undetectability of some faults), the system selects faults from non-prime equivalence sets and processes them next. If the requirement is still not satisfied, all the untested faults in the fault list are selected in turn to achieve the required fault coverage.

The fault counting procedure marks out all the detected faults. It functions as follows: whenever a fault is identified by the fault simulator, all other faults in the same fault class (in both equivalence and dominance sets) will be marked as detected. Normally, the selected fault (when using algorithmic test generation method) should be identified by the fault simulator also. However, sometimes the selected fault is not detected by the fault simulator because the Critical Path Tracing is an approximate algorithm. The system marks the selected fault for which test was generated as detected, irrespective of whether fault simulation detects it or not.

The integrated system is designed to execute the complete test procedure. Depending on the application, it can be executed in various ways. Following are some useful applications.

1. To derive a set of test vectors which satisfies the user-specified fault coverage requirement.
2. To compute the fault coverage for a given set of test vectors prestored in a file.
3. To compute the fault coverage for a user-specified number of random test vectors.

1.2 System Environment

The program is written in PASCAL and runs on a SUN Workstation (SUN 3) under BSD UNIX 4.2. The program is fully portable. Certain parameters have been made constants in the program, and are set during compilation. The maximum values of these parameters are limited by actual main memory availability and the limits of the PASCAL compiler here at USC. Currently these values are:

- maximum number of gates (including the primary inputs): 10,000
- maximum number of faults: 20,000
- maximum number of primary inputs: 500
- maximum number of fanout branches per stem: 20
- maximum number of inputs per gate: 10
- maximum number of primary outputs: 300
- maximum number of characters for a file name: 30
- maximum number of characters for a gate name: 8
- maximum time in milliseconds allowed to test for a fault when using PODEM: 10,000

If it is desired to run the system with different parameters, the user must modify these constants and re-compile the program.

2 Input/Output & User Interface

2.1 I/O File descriptions

The system allows the user to enter desired file names during execution. The file name must be a standard Unix file name. Its maximum length is set to 30. For convenience, the system also provides default file names for all but the input circuit file. A default file name is formed by appending a dot "." and a default file extension to the input circuit file name. When inquired for a file name, the user can simply enter <RETURN> to use the default name. If the user doesn't want to generate some output file, he can enter "/" to suppress the file generation.

The following 13 files are used by the system. The strings in paratheses are the default file extensions.

- Input circuit description file (no default extension) — contains the gate-level description of a circuit. It must be provided whenever the system is invoked.
- Fault class file ("cls") — contains information about fault classes. It is generated during fault-collapsing, either by the fault collapser itself or when the fault collapser is called by the integrated system.
- Input fault file for test generation ("tgi") — contains a set of faults to be tested during test vector generation. It is provided by the user.
- Output test file of test generation ("tgo") — contains the resulting test vectors generated by the test generator.
- Input vector file for fault simulation ("fsi") — has the same format as the "tgo" file and is used for fault simulation.
- Output fault file of fault simulation ("fso") — contains the vectors obtained by assigning "0" or "1" to the "x" (*don't care*) for the vectors in a "fsi" file, and the corresponding fault lists generated during fault simulation.
- Input vector file for logic simulation ("lsi") — has the same format as the "tgo" and "fsi" files and is used for logic simulation.
- Output vector file of logic simulation ("lso") — contains the same input vectors as in the "fso" file, and the output vectors of the user-specified observation points.
- Output test vector file of integrated system ("tst") — contains the test vectors, the corresponding output vectors, the numbers of detected/undetected faults, and the fault coverage of these test vectors. It is generated by the integrated system.

- Fault information file ("flt") — contains information about each fault such as sourae gate number, destination gate number, stuck-at value, class number, detected or not, etc. It is generated by the integrated system.
- Complete result file ("res") — contains most information obtained during the execution of the integrated system. This includes test vectors, good-value simulation results, all faults detected by each test vector, the new faults detected by the current test vector, fault coverage, etc.
- Time file ("tim") — contains the information about execution time which includes the time for fault collapsing and the time for each iteration of the integrated system. It also provides accumulated fault coverage information.
- Input test file for integrated system ("int") — has the same format as the "tgo", "fsi" and "lsi" files. It is used only when the integrated system is required to acquire test vectors from a file.

The format of each file is described next. Each file will be described in three parts separated by dash lines. The first part is the exact file format, the second part is the file description, and the third part gives an example. The circuit, called "c60", for the example is shown in Figure 2.

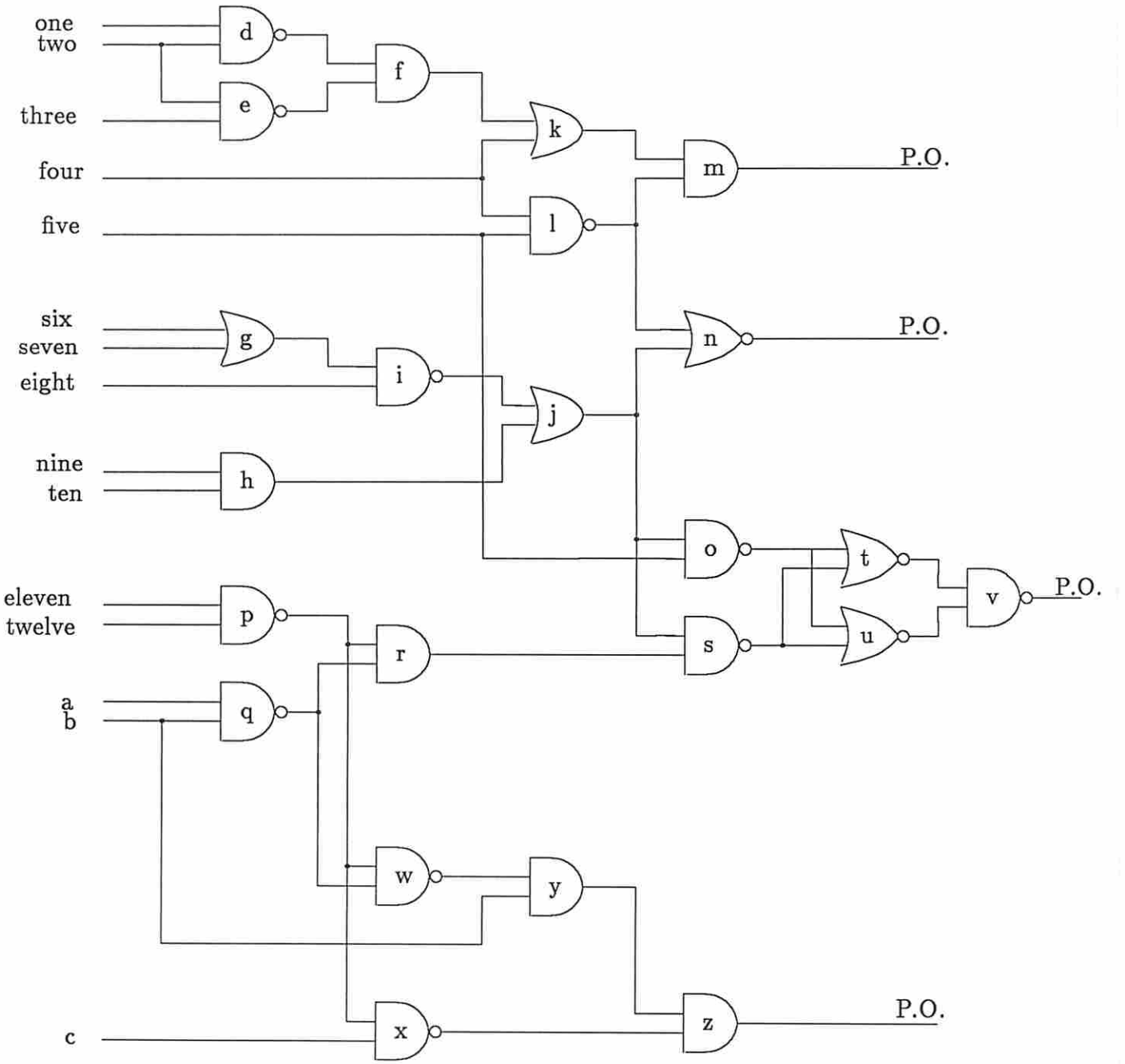

Figure 2: Circuit diagram of c60

2.1.1 Input circuit description file

```
-----------------------------------------------------------------
        #gates
         gtype   gname   #fout  #fin    ing1    ing2    ...
         gtype   gname   #fout  #fin    ing1    ing2    ...
               ...
         gtype   gname   #fout  #fin    ing1    ing2    ...
-----------------------------------------------------------------
where   #gates    = number of gates in circuit, each primary input
                    is represented as a special gate.
        gtype     = one of the following strings:
                     "inpt" -- primary input
                     "and"  -- AND gate
                     "buf"  -- buffer, the same as 1-input OR gate
                     "or"   -- OR gate
                     "nand" -- NAND gate
                     "nor"  -- NOR gate
                     "inv"  -- inverter, the same as 1-input NOR
                               gate
        gname     = user-specified gate name.
        #fout     = number of fanouts.
        #fin      = number of fanins.
        ing1,ing2 ... = names of input gates which feed the current
                        gate.

Note:  1) Each primary input is represented as a special gate of
          type "inpt". The value of #fin for a primary input
          must be 0.
       2) The value of #fout for a primary output must be 0.
          If a primary output in the original circuit is a fanout
          branch from previous stage, then a buffer must be inserted
          (see Figure 3) to denote the primary output.
       3) Words/numbers in the same line can be separated by
          one or more spaces or tabs.
-----------------------------------------------------------------
Example: c60

38
inpt     one        1    0
inpt     two        2    0
inpt     three      1    0
```

```
inpt    four      2   0
inpt    five      2   0
inpt    six       1   0
inpt    seven     1   0
inpt    eight     1   0
inpt    nine      1   0
inpt    ten       1   0
inpt    eleven    1   0
inpt    twelve    1   0
inpt    a         1   0
inpt    b         2   0
inpt    c         1   0
nand    d         1   2   one       two
nand    e         1   2   two       three
nand    l         2   2   four      five
or      g         1   2   six       seven
and     h         1   2   nine      ten
nand    p         3   2   eleven    twelve
nand    q         2   2   a         b
and     f         1   2   d         e
nand    i         1   2   g         eight
nand    x         1   2   p         c
and     r         1   2   p         q
nand    w         1   2   p         q
or      k         1   2   f         four
or      j         3   2   h         i
and     y         1   2   w         b
and     m         0   2   k         l
nor     n         0   2   l         j
nand    o         2   2   five      j
nand    s         2   2   j         r
and     z         0   2   x         y
nor     t         1   2   o         s
nor     u         1   2   o         s
nand    v         0   2   t         u

------------------------------------------------------------------
```

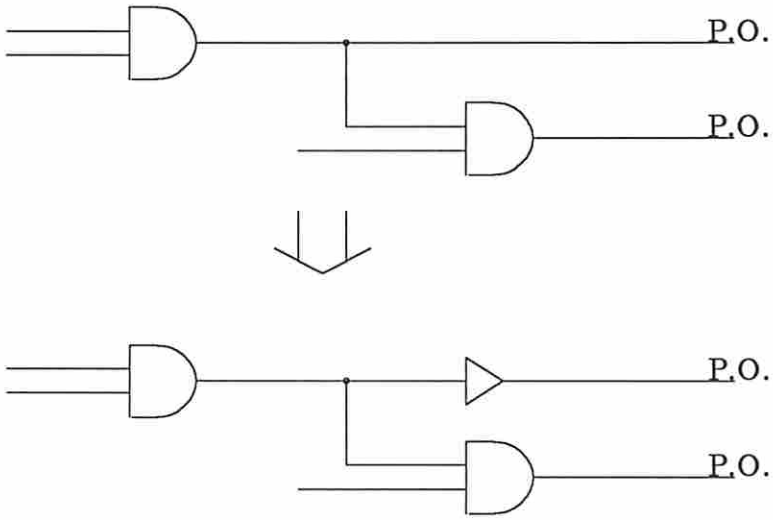

Figure 3: Inserting a buffer at a primary output

2.1.2 Fault class file ("cls")

```
-----------------------------------------------------------------
        nclasses
                   < one blank line >
         class-type   neqs1
                 src    dest    s-a-fault
                 src    dest    s-a-fault
               ...
                        ndos1
                 src    dest    s-a-fault
                 src    dest    s-a-fault
               ...
                   < one blank line >
         class-type   neqs2
               ...
                   < one blank line >
         class-type   neqsn
                 src    dest    s-a-fault
                 src    dest    s-a-fault
               ...
                        ndosn
                 src    dest    s-a-fault
               ...
-----------------------------------------------------------------
```

```
where   nclasses =  number of fault classes after fault-collapsing.
        class-type = "p" if the class is prime, "n" otherwise.
        neqs1, neqs2, ...
                =  number of faults in equivalence set 1, 2, ...
        ndos1, ndos2, ...
                =  number of faults in dominance set 1, 2, ...
        src     =  source gate name for the fault.
        dest    =  destination gate name for the fault,
                   dest="0" if the fault is on a stem or a primary
                   output
        s-a-fault   =  stuck-at value of the fault.

Note: 1) There is one blank line before each class.
      2) Some equivalence sets don't have any dominance sets,
         in which case only the equivalence sets are listed.
-----------------------------------------------------------------
Example: c60.cls

74

p  3
    one     d               0
    two     d               0
    d       f               1
   4
    four    k               1
    f       k               1
    k       m               1
    m       0               1

p  1
    one     d               1
   6
    d       f               0
    e       f               0
    ...

p  1
    s       0               0
   0

p  1
```

```
    z       0               1
   0

p  1
    v       0               0
   0
-----------------------------------------------------------------
```

2.1.3 Fault file for test generation ("tgi")

```
-----------------------------------------------------------------
       src      dest        sa-value
       src      dest        sa-value
          ...
       src      dest        sa-value
-----------------------------------------------------------------
where  src  = source gate name of the fault
       dest = destination gate name of the fault, dest="0" if the
              fault is on the stem or a primary output
       sa-value = stuck-at value of the fault
-----------------------------------------------------------------
Example: c60.tgi

one 0 1
two 0 0
two d 1
q x 1
q w 0
-----------------------------------------------------------------
```

2.1.4 Output test file of test generation ("tgo"), Input vector file for fault simulation ("fsi"), Input vector file for logic simulation ("lsi"), Input test file for integrated system ("int")

```
-----------------------------------------------------------------
testvector / comment line
testvector / comment line
   ...
testvector / comment line     .
-----------------------------------------------------------------
```

```
where  each line can be a test vector or a comment line.
       each testvector is a vector consisting of "0", "1" or "x".
       each comment line must start with a "*".
       for "tgo" file, a comment line may represent
         1) an undetectable fault,
         2) a "hard" fault (i.e., the test generator cannot
            find the test in the specified time), or
         3) an undefined fault.

Note: For convenience, the system adds one space between every 5 bits
      for each test vector when generating the "tgo" file. It is
      recommended that the user provide the same format for the other
      files to increase readability.
-----------------------------------------------------------------------
Example: c60.tgo

0100x xxxxx xxxxx
0110x xxxxx xxxxx
10x0x xxxxx xxxxx
*** 3: Fault undefined!***
xxxxx xxxxx x0010
-----------------------------------------------------------------------
```

2.1.5 Output fault file of fault simulation ("fso")

```
-----------------------------------------------------------------------
   < one blank line >
inputvector1
   [ po1 ]= val11
     src     dest     sa-value
     src     dest     sa-value
       ...
     src     dest     sa-value
   [ po2 ]= val12
     src     dest     sa-value
       ...
   < one blank line >
inputvector2
   [ po1 ]= val21
     src     dest     sa-value
```

```
       ...
-------------------------------------------------------------------
where inputvectori = the i-th input vector, each bit of the vector
                     must be either "0" or "1". There is one
                     space between every 5 bits.
      poi  = name of the i-th primary output gate.
      valji = the value of i-th primary output under the j-th
              input vector.
      src  dest  sa-value = the source gate name, the destination
              gate name and the stuck-at value of the fault.

Note: There must be one blank line before every input vector.
-------------------------------------------------------------------
Example: c60.fso

00000 00000 00000
  [m       ]= 1
    d        f        0
    e        f        0
    f        k        0
    k        m        0
    l        0        0
    m        0        0
    l        m        0
  [n       ]= 0
    n        0        1
  [z       ]= 0
    y        z        1
    z        0        1
  [v       ]= 1
    v        0        0

11111 11111 11111
  [m       ]= 0
    five     0        0
    l        0        1
    four     l        0
    five     l        0
     ...

11111 00000 11111
  [m       ]= 0
```

```
   five      0         0
   l         0         1
     ...
 [v        ]= 1
   v         0         0
-----------------------------------------------------------------
```

2.1.6 Output vector file of logic simulation ("lso")

```
 -----------------------------------------------------------------
      1: obv1
      2: obv2
         ...
      n: obvn
      /
      inputvector1
      obv-vector1
      inputvector2
      obv-vector2
      ...
 -----------------------------------------------------------------
 where obvi = name of the i-th observation point
       inputvectori = i-th input vector
       obv-vectori  = i-th observation vector

 Note: 1) The number before each observation point is its position
          in the observation vector.
       2) There must be one line containing "/" between the name
          of the last observation point and the first input vector.
       3) There is a space between every 5 bits in each vector.
-----------------------------------------------------------------
 Example: c60.lso

1:m
2:n
3:z
4:v
5:one
6:two
/
```

```
00000 00000 00000
10010 0
11111 11111 11111
00111 1
01010 01010 01010
10010 1
10101 10101 10101
10011 0
10101 01010 10101
10001 0
11111 00000 11111
00111 1
 ----------------------------------------------------------------
```

2.1.7 Output test vector file of integrated system ("tst")

```
 ----------------------------------------------------------------
    1: pi1
    2: pi2
    ...
    n: pin
    /
    1: po1
    2: po2
    ...
    m: pom
    /
    testvector1
    outputvector1
    testvector2
    outputvector2
    ...

    /
       nvectors
       nfaults   ndetected     fc
 ----------------------------------------------------------------
 where
    pii  = the name of the i-th primary input.
    poi  = the name of the i-th primary output.
```

```
    testvectori    = the i-th derived test vector
    outputvectori = the output vector corresponding to the i-th test
                    vector
    nfaults        = total number of stuck-at faults
    ndetected      = number of detected faults by the test vectors
                    in this file
    nvectors       = number of test vectors in the file
    fc             = fault coverage

 Note: 1) The number before each name is its position in the
          corresponding vector.
       2) There must be three lines containing "/" to mark the end of
          primary input names, primary output names and the test
          vectors respectively.
       3) There is a space between every 5 bits in each vector.
-----------------------------------------------------------------------
 Example: c60.tst

1:one
2:two
3:three
4:four
5:five
6:six
7:seven
8:eight
9:nine
10:ten
11:eleven
12:twelve
13:a
14:b
15:c
/
1:m
2:n
3:z
4:v
/
01000 00000 00000
1001
01100 00000 00000
```

```
0001
10000 00000 00000
1001
00000 00000 00010
1001
/
     4
     120       31     25.8333
```

2.1.8 Fault information file ("flt")

```
 ------------------------------------------------------------------
  fault no.    source    destination  s-a-value     class     detected
       1         src1        dest1        sa-1       class-1      det-1
       2         src2        dest2        sa-2       class-2      det-2
          ...
 ------------------------------------------------------------------
 where     srci  =  source gate name of the i-th fault
           desti =  destination gate name of the i-th fault,
                    dest = "0" means that fault is on a stem or
                    a primary output.
           sa-i  =  stuck-at value of the i-th fault (s-a-0 or s-a-1).
           class-i  =  equivalence fault class number that the i-th
                       fault belongs to.
           det-i =  "yes" if the fault is detected, "no" if the fault
                    is undetectable, "hard" if the test generator
                    cannot give the result before time-out.

 Note: For easy reference, the first line of this file
       is the string "Fault no. source  destination   ...   detected".
       When using this file as an input file (for later
       processing), the first line must be skipped.
 ------------------------------------------------------------------
 Example: c60.flt

  fault no.     source     destination  s-a-value      class      detected
       1        one          d              0           1          yes
       2        one          d              1           2          yes
```

```
      3        two          0                0          57          yes
         ...
    118        u            v                1          56          no
    119        v            0                0          74          yes
    120        v            0                1          52          yes
-------------------------------------------------------------------
```

2.1.9 Complete result file ("res")

```
-------------------------------------------------------------------
testvector-1
outputvector-1
  num-faults-1
      f11   f12   f13 ...
  new-faults-1
      nf11  nf12  nf13 ...
testvector-2
outputvector-2
  num-faults-2
      f21   f22   f23 ...
  new-faults-2
      nf21  nf22  nf23 ...
testvector-3
outputvector-3
  .
  .
  .
testvector-n
outputvector-n
  num-faults-n
      fn1   fn2   fn3 ...
  new-faults-n
      nfn1  nfn2  nfn3 ...
/
  nvectors
  nfaults  ndetected  fc
-------------------------------------------------------------------
where
      testvector-i    = i-th test vector consisting of 0's and 1's
      outputvector-i = good circuit output for testvector-i
```

```
      num-faults-i    = total number of faults detected by the i-th
                        test vector
      new-faults-i    = total number of new faults detected by the
                        i-th test vector
      fij             = faults detected by the i-th test vector
      nfij            = new faults detected by i-th test vector
      nvectors        = number of test vectors in the file
      nfaults         = total number of faults
      ndetected       = total number of detected faults
      fc              = percentage (%) of faults covered

 Note: 1) There must be one line containing "/" to signify
          the end of test vectors.
       2) Each fault is represented by its fault number given by the
          system. User can refer to the fault information file for
          more information.
-----------------------------------------------------------------------
 Example: c60.res

11001 10000 00010
0000
36
    1       5      48      16      76      86      98       3     100
   61      38      69      67      73      84      96     114     104
  106     110     112     115     117     120      17      21      91
  102      28      77      87      65      71      81      93     108
36
    1       5      48      16      76      86      98       3     100
   61      38      69      67      73      84      96     114     104
  106     110     112     115     117     120      17      21      91
  102      28      77      87      65      71      81      93     108
01000 00011 10100
1001
13
    2      47      49      53      75      85      97      10      51
  100      96     114     119
10
    2      47      49      53      75      85      97      10      51
  119
  ...
01111 01010 00110
0011
```

```
24
   17      13      19      52      98      54      28      77      87
  100      89      46      43      79      83      95     113      37
   41      70      74      39     119      82
1
   82
/
   19
   120     112     93.3333
-----------------------------------------------------------------
```

2.1.10 Time file ("tim")

```
 ---------------------------------------------------------------
 Fault collapsing time =       fct
 Vector     1   fc1            time1
 Vector     2   fc2            time2
   ...
 ---------------------------------------------------------------
 where fct   = fault collapsing time
       fci   = fault coverage after the i-th iteration
       timei = the elapsed cpu time in seconds up to the i-th
               iteration.
---------------------------------------------------------------
 Example: c60.tim

 Fault collapsing time =       0.800
 Vector     1   30.00          1.983
 Vector     2   38.33          2.033
 Vector     3   50.00          2.133
 Vector     4   57.50          2.216
 Vector     5   59.17          2.316
 Vector     6   75.00          2.416
 Vector     7   76.67          2.483
 Vector     8   81.67          2.583
 Vector     9   82.50          2.666
 Vector    10   85.00          2.733
 Vector    11   85.83          2.800
 Vector    12   86.67          2.883
 Vector    13   88.33          2.933
```

```
Vector     14    89.17          3.033
Vector     15    90.00          3.116
Vector     16    90.83          3.200
Vector     17    91.67          4.633
Vector     18    92.50          4.700
Vector     19    93.33          4.766
-----------------------------------------------------------------
```

2.2 I/O specifications

The input circuit description file must be provided whenever the system is invoked. After obtained the circuit file, the system will display a main menu showing the options provided. For each option in the menu, inputs and outputs are as follows:

Fault collapsing:

Input:

1. Name of the fault class file.
2. Option to carry out equivalence merging only or both equivalence and dominance merging.
3. Option to see the result on screen.

Output:

1. An internal array of all faults (in memory) which can be used in the integrated system.
2. Class file (optional).
3. Fault collapsing result on screen (optional).

Test vector generation:

Input:

1. Name of Output test file of test generation.
2. Options for providing faults.

- if manually, fault representations.
- if using a file, name of file and option to see result on screen.

Output:

1. Output test file of test generation (optional).
2. If faults are specified manually, test results are always displayed. If faults are in a file, test result display is optional.

Fault simulation:

Input:

1. Name of Input vector file for fault simulation.
2. Name of Output fault file of fault simulation.
3. How to assign values to "x" terms in input vectors. If random assignment, the probability of assigning "1".
4. Option to see the result on screen.

Output:

1. Fault simulation result on screen (optional).
2. Output fault file of fault simulation (optional).

Logic Simulation:

Input:

1. Name of Input vector file for logic simulation.
2. Name of Output vector file of logic simulation.
3. How to assign values to "x" terms in input vectors. If random assignment, the probability of assigning "1".
4. Option to observe all primary outputs.
5. Names of all observation points of interest, in addition to the primary outputs.

6. Option to see result on screen.

Output:

1. Logic simulation result on screen (optional).
2. Output vector file of logic simulation (optional).

Integrated system:

Input:

1. Option to use default values of parameters. If not using default values, then
   (a) Option for Test Generation method. If using PODEM, fault selection method. If using random test generation, the probability of a primary input being "1". If using a file containing test vectors, file name.
   (b) Exit condition.
2. Name of Fault class file.
3. Name of Output test vector file of integrated system.
4. Name of Fault information file.
5. Name of Complete result file.
6. Name of Time file.
7. How to assign values to "x" terms in test vectors. If random assignment, the probability of assigning "1".
8. Option to carry out equivalence merging only or both equivalence and dominance merging.

Output:

1. Test process status on screen.
2. Fault class file (optional).
3. Output test vector file of integrated system (optional).
4. Fault information file (optional).
5. Complete result file (optional).
6. Time file (optional).

3 Sample run

This section illustrates the system through an example circuit as shown in Figure 2 in Section 2. Each function of the system will be illustrated in a subsection. C60 contains 38 gates (15 primary inputs plus 23 internal gates), 120 single stuck-at faults (on 60 lines), and 4 primary outputs. The circuit description can be found in 2.1.1.

When the program is invoked, the system first asks for the name of the input circuit file. This file must have the format described earlier. After obtaining a legal file name, the system creates a temporay file with name "temp.temp" and uses this file for all subsequent operations. Thus before using the system, the user must make sure that there is no file with name "temp.temp" in the current directory. This temporary file will be deleted when exiting the system. Let "c60" be the input circuit file name.

```
Please enter circuit file name: c60
```

The system then displays the following Main Menu:

```
                MAIN MENU

    0. Exit
    1. Fault-collapsing
    2. Test-generation
    3. Fault-simulation
    4. Logic-simulation
    5. Integrated System

  Please enter your choice:
```

We will illustrate each function in the following subsections.

3.1 Fault Collapsing

When fault collapsing is selected, the system first asks for the fault class file name. The user can either use the default file name (here it is "c60.cls"), select another file name, or choose not to generate any output file at all (by entering "/"). Throughout this section, we will use the default file names for all functions. Thus a <RETURN> is always entered.

```
              <   Main menu  >
            Please enter your choice: 1

 Please enter output file name for fault classes.
  <RETURN> --- use default name: c60.cls
     "/"   --- do not generate class file.
         Enter: <RETURN>
```

The system then asks the user whether he wants to carry out equivalence merging only, or both equivalence and dominance merging.

```
 Please choose one option for fault collapsing:
    1  Equivalence merging only (for circuits with feedback).
    2  Equivalence as well as dominance merging.
 Enter option: 2
```

After the option is selected, the system starts the fault collapsing process based on equivalence merging. If option 2 is selected, fault collapsing based on dominance merging is executed after equivalence merging. When the collapsing process finishes, the system produces a message and inquires whether it should display the fault collapsing information.

```
 Fault collapsing completed.
 Do you want to see the result on screen? y
```

If the answer is "y", the system will first display the number of all classes and the number of prime classes, and then display the information about each class.

```
 All the faults have been collapsed into 74 fault classes,
 64 of which are prime.

 Class #1  (prime)
 --------
    Equivalent fault(s):
         Fault no.   1:  Src one     ,  Dest d       ;  Stuck at "0".
         Fault no.   5:  Src two     ,  Dest d       ;  Stuck at "0".
```

```
          Fault no.  48:  Src d       ,  Dest f       ;  Stuck at "1".
     Dominating fault(s):
          Fault no.  16:  Src four    ,  Dest k       ;  Stuck at "1".
          Fault no.  76:  Src f       ,  Dest k       ;  Stuck at "1".
          Fault no.  86:  Src k       ,  Dest m       ;  Stuck at "1".

  Class #2  (prime)
  --------
     Equivalent fault(s):
          Fault no.   2:  Src one     ,  Dest d       ;  Stuck at "1".
...

  Class #22  (non-prime)
  ---------
     Equivalent fault(s)
          Fault no.  47:  Src d       ,  Dest f       ;  Stuck at "0".
       ...
  Class #74  (prime)
  ---------
     Equivalent fault(s):
          Fault no. 119:  Src v       ,  Dest 0       ;  Stuck at "0".

 <RETURN> to continue
```

As shown above, each fault class consists of an equivalence set and a dominance set. A class may be prime (e.g., Class 1, 2) or non-prime (e.g, Class 22). Each fault is represented by its fault number, source gate name, destination gate name and the stuck-at value. The fault number is assigned by the system and used for easy-reference.

After all the classes have been displayed, a <RETURN> will cause the system to display a minimal fault set. This set consists of the first fault in each equivalence set of each prime class.

```
  Minimal fault set:
  -----------------

          Fault no.   1:  Src one     ,  Dest d       ;  Stuck at "0".
          Fault no.   2:  Src one     ,  Dest d       ;  Stuck at "1".
          Fault no.   6:  Src two     ,  Dest d       ;  Stuck at "1".
                     ...
          Fault no. 114:  Src z       ,  Dest 0       ;  Stuck at "1".
```

```
        Fault no. 119:  Src v        ,  Dest 0        ;  Stuck at "0".

<RETURN> to continue
```

The user must type another <RETURN> to continue the process. The system now asks whether the user wants to see the result again. It will continue printing the result until the user's answer is no (type 'n'). The system then goes back to the Main Menu.

```
Do you want to see the result again? y
     ...
 < print the result again >
     ...
Do you want to see the result again? n
 < go back to Main Menu >
```

3.2 Test Vector Generation

To generate test vectors for specific faults, the user selects option 2 from the Main Menu. The system then asks for the name of the file to store the generated test vectors. Again, the user can specify a new file name, use the default file name, or elect not to generate an output file.

The faults to be tested can be provided interactively or prestored in a file. For the interactive mode, the system repeatedly asks for the fault representation, generates the test and displays the result on the screen, until the user want to exit test vector generation. If the selected fault is undefined (e.g., a non-existig line stuck-at-fault), undetectable, or can not be tested within the specified maximum time, an appropriate message is displayed.

```
                < Main Menu >
     Please enter your choice: 2

How will you provide the fault(s)?
    1. Manually
    2. Using a file
 Enter your selection (1 or 2): 1

 Please enter output file name for test vectors.
   <RETURN> --- use default name: c60.tgo
```

```
      "/"   --- do not generate test vector file.
    Enter: <RETURN>

Enter source gate name: one
Enter destination gate name ("0" for output stem): d
Enter stuck-at-fault (0/1): 1

Fault Source[one      ] , Destination[d       ] s-a-1 is detected by:
  input #  1 [one      ],  test bit: 0
  input #  2 [two      ],  test bit: 1
  input #  3 [three    ],  test bit: 0
       ...
  input # 14 [b        ],  test bit: x
  input # 15 [c        ],  test bit: x

Would you like to process another fault?: [y/n]y

Enter source gate name: one
Enter destination gate name ("0" for output stem): two
Enter stuck-at-fault (0/1): 1

Fault Source[one      ] , Destination[two     ] s-a-1 is undefined.

Would you like to process another fault?: [y/n]y

Enter source gate name: o
Enter destination gate name ("0" for output stem): t
Enter stuck-at-fault (0/1): 0

Fault Source[o        ] , Destination[t       ] s-a-0 is undetectable!!

Would you like to process another fault?: [y/n]n
      < go back to Main Menu >
```

One may also generate tests for faults prestored in a file (e.g., c60.tgi). The system asks whether or not to display the results on screen (in the interactive mode, the result is always displayed). A test vector for each detectable fault specified in the file is generated. If a specified fault is undefined, undetectable or is too hard to detect, an appropriate message is produced.

```
How will you provide the fault(s)?
    1. Manually
    2. Using a file
 Enter your selection (1 or 2): 2

 Please enter file name containing faults.
  <RETURN> --- use default name: c60.tgi
     Enter: <RETURN>
 Do you want to see the result on screen? y

Fault Source[one     ] , Destination[0       ] s-a-1 is detected by:
  input #  1 [one     ],  test bit: 0
  input #  2 [two     ],  test bit: 1
  input #  3 [three   ],  test bit: 0
       ...
  input # 14 [b       ],  test bit: x
  input # 15 [c       ],  test bit: x

Fault Source[two     ] , Destination[0       ] s-a-0 is detected by:
  input #  1 [one     ],  test bit: 0
  input #  2 [two     ],  test bit: 1
      ...
Fault Source[two     ] , Destination[d       ] s-a-1 is detected by:
  input #  1 [one     ],  test bit: 1
  input #  2 [two     ],  test bit: 0
      ...
Fault Source[q       ] , Destination[x       ] s-a-1 is undefined.
 Skip this fault!

Fault Source[q       ] , Destination[w       ] s-a-0 is detected by:
  input #  1 [one     ],  test bit: x
  input #  2 [two     ],  test bit: x
  input #  3 [three   ],  test bit: x
      ...
  input # 15 [c       ],  test bit: 0

           <go back to the Main Menu>
```

3.3 Fault Simulation

Option 3 selects fault simulation which uses the Critical Path Tracing algorithm. The input vectors to be fault-simulated must be stored in a file (e.g., c60.fsi). The system also asks for the name of the file to store the simulation result.

```
Please enter the name of file containing the
test vectors to be fault-simulated.
 <RETURN> --- use default name: c60.fsi
  Enter: <RETURN>

Please enter the name of file to store the result.
 <RETURN> --- use default name: c60.fso
  / --- do not generate the output file.
   Enter: <RETURN>
```

Since the Critical Path Tracing algorithm requires that the value of each primary input be specified, the "x" terms in the input vectors must be assigned either "1" or "0". These x's can be assigned binary values in one of three ways: 1) always assign "1", 2) always assign "0", or 3) randomly assign "1" and "0". In the last case, the system allows the user to specify the probability of assigning "1". After this, the system asks whether to display the result on screen. Assume the answer is yes.

```
If there is any "x" (don't care) in the input vector,
 what value should it be assigned?
    1. always assign 1
    2. always assign 0
    3. randomly assign 1 or 0
 Enter your selection:  3
 Enter the probability of assigning 1: 0.5

Do you want to see the result on the screen? y
```

The system now starts fault simulation. Whenever it finishes processing an input vector, it will display the results on screen. This process repeats until the input vectors are exhausted. The system then returns to the Main Menu.

```
INPUT VECTOR
```

```
00000 00000 00000

FAULTS CHANGING THE OUTPUT OF GATE [m       ] FROM 1 TO 0:
      Src [d       ],   Dest[f       ];  Stuck at "0"
      Src [e       ],   Dest[f       ];  Stuck at "0"
      Src [f       ],   Dest[k       ];  Stuck at "0"
      Src [k       ],   Dest[m       ];  Stuck at "0"
      Src [l       ],   Dest[0       ];  Stuck at "0"
      Src [m       ],   Dest[0       ];  Stuck at "0"
      Src [l       ],   Dest[m       ];  Stuck at "0"
FAULTS CHANGING THE OUTPUT OF GATE [n       ] FROM 0 TO 1:
      Src [n       ],   Dest[0       ];  Stuck at "1"
FAULTS CHANGING THE OUTPUT OF GATE [z       ] FROM 0 TO 1:
      Src [y       ],   Dest[z       ];  Stuck at "1"
      Src [z       ],   Dest[0       ];  Stuck at "1"
FAULTS CHANGING THE OUTPUT OF GATE [v       ] FROM 1 TO 0:
      Src [v       ],   Dest[0       ];  Stuck at "0"

INPUT VECTOR
11111 11111 11111
      ...
INPUT VECTOR
11111 00000 11111

FAULTS CHANGING THE OUTPUT OF GATE [m       ] FROM 0 TO 1:
      Src [five    ],   Dest[0       ];  Stuck at "0"
      Src [l       ],   Dest[0       ];  Stuck at "1"
         ...
      Src [l       ],   Dest[m       ];  Stuck at "1"
FAULTS CHANGING THE OUTPUT OF GATE [n       ] FROM 0 TO 1:
      Src [i       ],   Dest[j       ];  Stuck at "0"
         ...
      Src [j       ],   Dest[n       ];  Stuck at "0"
FAULTS CHANGING THE OUTPUT OF GATE [z       ] FROM 1 TO 0:
      Src [x       ],   Dest[z       ];  Stuck at "0"
         ...
      Src [z       ],   Dest[0       ];  Stuck at "0"
FAULTS CHANGING THE OUTPUT OF GATE [v       ] FROM 1 TO 0:
      Src [v       ],   Dest[0       ];  Stuck at "0"
             <go back to Main Menu>
```

3.4 Logic simulation

To use logic simulation, select option 4 from the Main Menu. For this option, the user must prepare the input vectors in a file, provide file names (both input and output files) to the system and decide how to assign values to "x" terms.

```
                < Main Menu >
            Please enter your choice: 4

Please enter the name of file containing the
test vectors to be logic-simulated :
  <RETURN> --- use default name: c60.lsi
    Enter: <RETURN>

Please enter the file name for the result.
  <RETURN> --- use default name: c60.lso
   / --- do not generate the output file.
    Enter: <RETURN>

If there is any "x" (don't care) in the input vector,
 what value should it be assigned?
    1. always assign "1"
    2. always assign "0"
    3. randomly assign "1" or "0"
 Enter your selection: 3
 Enter the probability of assigning "1": 0.4
```

The system then asks for the name of gates whose outputs are to be observed. Since the primary outputs are usually of interest, the system first asks whether or not to observe all the primary outputs, and then asks for any other observation points. To end the selection, type "0" for the gate name. If all the primary output are to be observed, the system will display the order in which these outputs appear in the output vector.

```
Do you want to observe all the primary outputs? y

The 4 primary outputs of the circuit will be displayed as the
first 4 bits of each output vector in the following order :
   Bit 1: m
   Bit 2: n
   Bit 3: z
```

```
   Bit 4: v

 Enter the names of other gates to be observed:
  Gate name ("0" to end the selection)  5: one
  Gate name ("0" to end the selection)  6: two
  Gate name ("0" to end the selection)  7: s
  Gate name ("0" to end the selection)  8: y
  Gate name ("0" to end the selection)  9: 0

 Do you want to see the result on the screen? y
```

The result to be displayed on screen is as below:

```
-----------------------------------------------
Input vector :
 00000 00000 00000
 Observed values :
 10010 000
 -----------------------------------------------
Input vector :
 11111 11111 11111
Observed values :
 00111 111
-----------------------------------------------
Input vector :
 01010 01010 01010
Observed values :
 10010 100
-----------------------------------------------
Input vector :
 10101 10101 10101
Observed values :
 10011 010
-----------------------------------------------
Input vector :
 10101 01010 10101
Observed values :
 10001 000
-----------------------------------------------
```

```
Input vector :
 11111 00000 11111
Observed values :
 00111 111
```

After the display, the system automatically returns to the Main Menu.

3.5 Integrated System

The Integrated System is used for carrying out various complex test procedures. There are several parameters to be specified by the user for various applications. The system provides a convenient default set of values for these parameters as shown below.

```
                <Main Menu>
     Please enter your choice: 5

  Would you like to use the following default values?

      Exiting Condition : fault coverage = 100%
      In-order fault selection
      Test Generation Method : PODEM

     Please enter: [y/n] y
```

If the default values are to be used, the system will set up these values and ask for the following file names:

```
For the following file names, enter
 <RETURN> to use default file name as shown in paratheses,
    "/"   to suppress file generation, or
  enter the desired name.

Please enter file name for fault classes.
 (default name: c60.cls) : <RETURN>
Please enter file name for resulting test vectors.
 (default name: c60.tst) : <RETURN>
Please enter file name for fault list.
```

```
 (default name: c60.flt) : <RETURN>
Please enter file name for complete output result.
 (default name: c60.res) : <RETURN>
Please enter file name for execution time.
 (default name: c60.tim) : <RETURN>
```

Since the default test generation algorithm is PODEM which may generate test vectors with "x" terms, the user must tell the system how to assign values to "x" terms.

```
If there is any "x" (don't care) in the test vector,
 what value should it be assigned?
    1. always assign 1
    2. always assign 0
    3. randomly assign 1 or 0
 Enter your selection: 3
 Enter the probability of assigning "1": 0.7
```

After this, the system calls the fault collapsing procedure and asks the user the same question as described before:

```
  Please choose one option for fault collapsing:
    1. Equivalence merging only (for circuits with feedback).
    2. Equivalence as well as dominance merging.
  Enter option: 2
```

Since the objective is to derive a complete set of test vectors, it is recommended to select option 2 for this question so that more information about faults can be obtained and used later. When the fault collapsing is completed, a message will appear on screen and the number of all faults is displayed.

```
 **** Fault Collapsing OK! ****

  Number of faults = 120
```

The system then enters the major loop which consists of fault selection, test generation, fault simulation and fault counting. For each iteration, the system prints the test vector generated by PODEM, the number of faults detected so far, and the fault coverage.

```
 *** Now enter the main loop ... ***

 Number of detected faults = 0
 Current fault coverage is 0.00%

**************************************************
**************************************************
 First selection iteration..
------------------------------------------
Vector[1] is
11001 10000 01011
Number of detected faults = 33
Current fault coverage is 27.50%
------------------------------------------
Vector[2] is
01001 01111 10100
Number of detected faults = 51
Current fault coverage is 42.50%
------------------------------------------
     ...
------------------------------------------
Vector[15] is
11011 01111 00110
Number of detected faults = 111
Current fault coverage is 92.50%

**************************************************
**************************************************
 Second selection iteration...
------------------------------------------
Vector[16] is
11111 11011 01111
Number of detected faults = 112
Current fault coverage is 93.33%

**************************************************
**************************************************
 Third selection iteration...
Number of detected faults = 112
Current fault coverage is 93.33%
**************************************************
**************************************************
```

```
 Total number of faults is 120.

 112 faults have been detected by 16 test vectors.
 The fault coverage is 93.3333 %.

*****************************************************
```

When the procedure is terminated, the system again reports the total number of faults. It also prints the number of detected faults, number of test vectors, and the fault coverage as shown in the last three lines of the above example. Note that we used 100% fault coverage as our exit condition but the best possible fault coverage for this circuit is 93.3333% as there are 8 undetectable faults.

We now run the system with values other than the default values.

```
  Would you like to use the following default values?

     Exiting Condition : fault coverage = 100%
     In-order fault selection
     Test Generation Method : PODEM

    Please enter: [y/n] n
```

The system asks the user to select the test vector generation method. If PODEM is selected, the system asks for the exit condition and fault-selection method. Currently the exit condition can only be based on fault coverage or the number of test vectors and only the in-order fault selection is provided.

```
Which Test Generation method should be used?
  1. PODEM
  2. Random Test Generation
  3. Test Vectors in a file
 Enter your selection: 1

 Which exiting condition do you wish to use ?
  1. fault coverage
```

```
  2. number of test vectors
  3. CPU time (not available)
 Enter your selection: 1

 Please enter the percentage fault coverage desired: 80

 Which fault selection method should be used ?
  1. In-order selection
  2. Random fault selection by user (not available)
  3. Fault class with maximum # of faults (not available)
  4. Determined by Topology of circuit (not available)
 Enter your selection: 1

For the following file names, enter
 <RETURN> to use default file name as shown in paratheses,
     "/"   to surpress file generation, or
  enter the desired name.

Please enter file name for fault classes.
 (default name: c60.cls) :
Please enter file name for resulting test vectors.
 (default name: c60.tst) :
    ...
If there is any "x" (don't care) in the test vector,
 what value should it be assigned?
    ...
 Please choose one option for fault collapsing:
    ...
**** Fault Collapsing OK! ****
    ...
************************************************
************************************************
 First selection iteration..
-------------------------------------------
Vector[1] is
11000 10001 00111
Number of detected faults = 16
Current fault coverage is 13.33%
-------------------------------------------
  ...
Vector[11] is
01011 01111 01111
```

```
Number of detected faults = 87
Current fault coverage is 72.50%
-----------------------------------------
Vector[12] is
00111 11101 00110
Number of detected faults = 99
Current fault coverage is 82.50%
*****************************************************
*****************************************************
 Total number of faults is 120.

 99 faults have been detected by 12 test vectors.
 The fault coverage is 82.5000 %.

*****************************************************
```

If random test generation is selected, the system will generate random test vectors (and thus no fault selection is necessary). If the test vectors are prestored in a file, then the system will ask for the file name (also no fault selection is required). After obtaining these parameters, the system executes a test procedure similar to the one described above. No matter which option is used, the system returns to the Main Menu after the processing is over. It is recommended that the user try various options to fully understand the operation of this system.

References

[GOE 81] P. Goel, *An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits,* IEEE Transactions on Computers, pp. 215-222, March, 1981.

[ABR 84] M. Abramovici, P. Menon, and D. Miller, *Critical Path Tracing: An Alternative to Fault Simulation,* IEEE Design and Test of Computers, pp. 83-93, February, 1984.

Appendix A: Fault Collapsing

Under the stuck-at fault model, all lines (gate output stems and fanout branches) in a circuit can have two possible faults, viz., stuck-at-1 and stuck-at-0. Hence the total number of possible single faults is equal to twice the number of lines. In order to simplify the process of test generation and fault simulation, the faults can be organized into fault classes.

Each fault class FC_i is represented by a tuple (E_i, D_i), where

E_i = set of faults such that any test for any fault in E_i detects all other faults in E_i.

D_i = set of faults such that any test for any fault in E_i detects all faults in D_i (but not *vice versa*).

E_i is called the **equivalence set** and D_i is called the **dominance set** of fault class FC_i.

Given two fault classes $FC_i = (E_i, D_i)$ and $FC_j = (E_j, D_j)$, we say that FC_j is *included* in FC_i if $(E_j \cup D_j)$ is a subset of $(E_i \cup D_i)$. If FC_j is not included in any other fault class, it is said to be a *prime class*.

The objective of fault collapsing is to organize the set of all possible faults into maximal fault classes, such that every fault is present in the equivalence set of exactly one fault class. We shall illustrate the process of generating fault classes for two simple circuits, one without fanout and the other with fanout.

Fanout-free Circuits

Consider the circuit shown in Figure 4. Let the abbreviations $x/0$ and $x/1$ represent x stuck-at-0 and x stuck-at-1, respectively. There are ten possible single faults in this circuit: *a/0, b/0, c/0, d/0, e/0, a/1, b/1, c/1, d/1, e/1.*

Since d is the output of an AND gate with inputs a and b, the faults *a/0, b/0* and *d/0* are equivalent. This is because any test for any of these faults will automatically be a test for the other two. Hence we can set up the fault class

$$FC_1 = (\{a/0, b/0, d/0\},\ \{\}).$$

Further, any test for either of the faults *a/1* and *b/1* will automatically be a test for *d/1*. Hence we have the fault classes

$$FC_2 = (\{a/1\},\ \{d/1\});$$

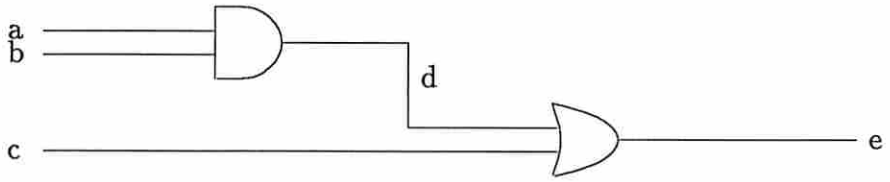

Figure 4: Fanout-free circuit

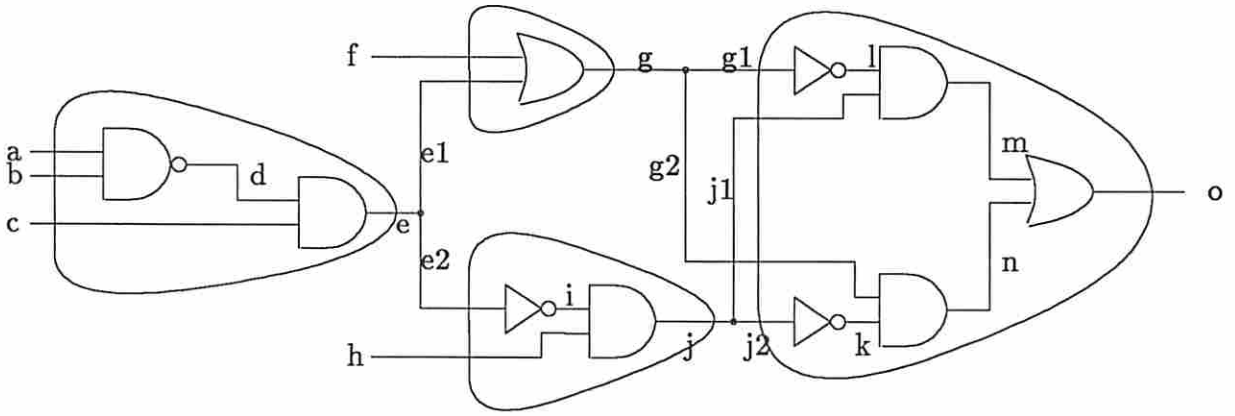

Figure 5: Circuit with fanout

$$FC_3 = (\{b/1\},\ \{d/1\}).$$

Finally, in order to ensure that the fault *d/1* is also present in the equivalence set of one fault class, we set up the class

$$FC_4 = (\{d/1\},\ \{\}).$$

Similarly, based on the OR gate with output e we set up the following classes.

$$\begin{aligned} FC_5 &= (\{c/1, d/1, e/1\},\ \{\}); \\ FC_6 &= (\{c/0\},\ \{e/0\}); \\ FC_7 &= (\{d/0\},\ \{e/0\}); \\ FC_8 &= (\{e/0\},\ \{\}). \end{aligned}$$

Using the structure of the circuit, we have set up eight primitive fault classes based on the original ten faults. However, there exist equivalence and dominance relationships between these classes and hence further class reductions are possible. This collapsing is done in two phases: *equivalence merging* and *dominance merging*.

Consider $FC_i = (E_i, D_i)$ and $FC_j = (E_j, D_j)$. **Equivalence merging** can be done on FC_i and FC_j if E_i and E_j have an element (fault) in common. If so, we can eliminate FC_i and FC_j and replace them by

$$FC_{i+j} = (E_i \cup E_j,\ D_i \cup D_j).$$

Classes 1 and 7 can be merged in this way.

After carrying out equivalence merging in all possible ways, we have six resulting fault classes. FC_2, FC_3, FC_6 and FC_8 are as before; and there are two merged classes, namely

$$\begin{aligned} FC_{1+7} &= (\{a/0, b/0, d/0\},\ \{e/0\}); \\ FC_{4+5} &= (\{c/1, d/1, e/1\},\ \{\}). \end{aligned}$$

It can be shown that the following properties hold over the set of fault classes produced by equivalence merging:

- Every fault is present in the equivalence set of exactly one fault class.
- No class is included in another, i.e., all the classes are *prime classes*.

Given fault classes $FC_i = (E_i, D_i)$ and $FC_j = (E_j, D_j)$, **dominance merging** can be done if D_i contains a fault which is also in E_j. If so, we replace FC_i by a new class

$$FC_{i+j} = (E_i,\ D_i \cup E_j \cup D_j).$$

This new class is also a *prime class*. The class F_j is now included in FC_{i+j}, and should be marked *non-prime*. It can be seen that after the dominance merging step, every fault is still present in the equivalence set of exactly one fault class.

For example, FC_2 and FC_{4+5} can be combined to yield a new prime fault class

$$FC_{2+(4+5)} = (\{a/1\},\ \{c/1, d/1, e/1\});$$

FC_2 is eliminated, and FC_{4+5} is retained after being marked non-prime.

After dominance merging has been carried out in all possible ways, we end up with five prime fault classes and one non-prime fault class. The five **prime classes** are

$$\begin{aligned} FC_{2+(4+5)} &= (\{a/1\},\ \{c/1, d/1, e/1\}); \\ FC_{3+(4+5)} &= (\{b/1\},\ \{c/1, d/1, e/1\}); \\ FC_6 &= (\{c/0\},\ \{e/0\}); \\ FC_8 &= (\{e/0\},\ \{\}); \\ FC_{1+7} &= (\{a/0, b/0, d/0\},\ \{e/0\}), \end{aligned}$$

and the **non-prime class** is

$$FC_{4+5} = (\{c/1, d/1, e/1\},\ \{\}).$$