ANTICS 1.3 User Guide

ANAFINS v2.5
ANAFAME v1.2
ANACOV v2.5

Stephen Spinks
Ian Bell
19-Sep-96
# Table of Contents

1. INTRODUCTION ......................................................................................................................... 4

2. OVERVIEW OF THE ANTICS AFS PROCESS ........................................................................ 5

3. THE ANTICS ENVIRONMENT .................................................................................................. 7

4. ANALYSIS TYPES SUPPORTED AND STRUCTURE OF SPICE INPUT FILE .......... 8

5. ANAFINS - FAULT INJECTION .................................................................................................. 10
   5.1. OVERVIEW ...................................................................................................................... 10
   5.2. FAULT MODEL DEFINITIONS ....................................................................................... 10
   5.3. FAULT INJECTION SPECIFICATIONS ......................................................................... 13
   5.4. THE FAULT INJECTION DATABASE ............................................................................. 15
   5.5. THE ANTICS SIMULATION BATCH FILE ........................................................................ 15
   5.6. HSPICE SYNTAX AND ANAFINS ................................................................................ 16
   5.7. PROGRAM EXECUTION .................................................................................................... 16
   5.8. LIST OF ANAFINS QUALIFIERS ..................................................................................... 17

6. ANAFAME - FAULT SIMULATION MANAGEMENT .......................................................... 22
   6.1. OVERVIEW ..................................................................................................................... 22
   6.2. SIMULATION ON REMOTE HOSTS ............................................................................... 22
   6.3. PING COMMAND CHECK ............................................................................................ 23
   6.4. PROGRAM OPERATION .................................................................................................. 23
   6.5. LIST OF ANFAME QUALIFIERS .................................................................................... 24

7. ANACOV - POST-PROCESSING ANALYSIS .......................................................................... 26
   7.1. OVERVIEW ..................................................................................................................... 26
   7.2. DETECTION MODES ....................................................................................................... 26
       7.2.1. Fixed ..................................................................................................................... 27
       7.2.2. Variable .............................................................................................................. 28
       7.2.3. Threshold ............................................................................................................ 28
       7.2.4. Digital ............................................................................................................... 28
       7.2.5. Alldata ................................................................................................................. 29
       7.2.6. Data .................................................................................................................... 31
       7.2.7. Threshdata ......................................................................................................... 31
   7.3. STATISTICAL ANALYSIS ............................................................................................. 31
   7.4. STROBE DEFINITIONS .................................................................................................. 35
   7.5. GSI PLOTS .................................................................................................................... 36
   7.6. RAW FILES .................................................................................................................. 37
   7.7. PRINT OPTIONS .......................................................................................................... 38
   7.8. THRESHOLD SWEEP MODE ........................................................................................ 38
   7.9. TRANSFORMATIONS ..................................................................................................... 38
   7.10. MEASURE STATEMENT ANALYSIS ............................................................................ 38
   7.11. STRUCTURE OF COVERAGE.DEF FILE .................................................................... 40
       7.11.1. Global commands ............................................................................................... 41
           7.11.1.1. Equivalent options .................................................................................... 41
       7.11.2. Analysis type commands .................................................................................... 41
           7.11.2.1. Print Options ............................................................................................. 41
           7.11.2.2. Statistical Options ...................................................................................... 41
           7.11.2.3. Rawfile Options ......................................................................................... 42
           7.11.2.4. Points Command ....................................................................................... 42
       7.11.3. Field commands .................................................................................................. 42
           7.11.3.1. Detection Modes ......................................................................................... 42
           7.11.3.2. Quantization Options .................................................................................. 44
           7.11.3.3. Strobe Modes ............................................................................................. 44
           7.11.3.4. Histogram Modes ...................................................................................... 45
1. Introduction

The ANTICS software suite is a set of programs for the development and evaluation of structural testing, DFT and BIST for analogue circuits. The Analogue Fault Simulation (AFS) which is performed produces an indication of the fault coverage of a given test technique for a given circuit. ANTICS has many useful features for AFS including several detection modes which enable new test methodologies (particularly Built In Self Test and Design for Test) to be evaluated by its modelling of the pass/fail decision process. In addition to simply providing a list of those faults which are undetectable and those which are detectable, ANTICS also produces information on the level of detectability and additional statistics about the faults. AC, DC and transient analysis are all supported as well as Monte Carlo simulation options, which, although they require a longer simulation time, can be used to produce more accurate results.

The fault modelling and fault injection languages which are provided for the description of faults have been designed to be as flexible as possible. This enables new fault models to be easily described using the SPICE-like parameterisable fault modelling language. The injection of faults may be into hierarchical circuits and redundant and equivalent short faults are automatically detected.

For the repeated simulation of faulty circuits (a requirement for AFS), ANTICS provides the option of parallel simulation of faults over a network of workstations which greatly reduces the fault simulation time. The software uses HSPICE as a core simulator to provide circuit simulation.

The ANTICS software provides a tool for the evaluation of fault coverage of structural test methods for analogue circuits. Its applications are in any area where analogue fault simulation is a requirement. In particular ANTICS is suited for use in DFT and BIST evaluation and the evaluation of structural test methodologies. Other applications include the optimisation and ordering of test sets for production testing (test time minimisation) and production test evaluation.
2. Overview of the ANTICS AFS Process

This section provides a basic overview and introduction to the ANTICS Analogue Fault Simulation software which has been developed to aid the testing of analogue circuits. The ANTICS software is a suite of programs which create and share a common simulation environment. An interface to the Cadence Design Framework II has also been developed. The basic structure of the software is shown below in Fig. 2-1.

ANTICS uses the standard analogue fault simulation procedure of fault injection, repeated simulation and post-processing with HSPICE as the kernel simulator for the simulation of the nominal and faulted circuits.

The main features of ANAFINS are that it supports fully hierarchical circuits and allows user-defined and parameterisable fault models. Any HSPICE component or model parameter may be replaced by a set of parameterisable HSPICE components as a fault model. This allows the fault simulator to be used for a wide variety of fault models, not just simple catastrophic fault models.

<table>
<thead>
<tr>
<th>Program Name</th>
<th>AFS Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANAFINS</td>
<td>Fault Injection</td>
</tr>
<tr>
<td>ANAFAME</td>
<td>Simulation Management / Network Fault Simulation Distribution</td>
</tr>
<tr>
<td>ANACOV</td>
<td>Post-Processing Analysis</td>
</tr>
<tr>
<td>MC_RAND</td>
<td>Monte Carlo Simulation Random Number Generator</td>
</tr>
</tbody>
</table>

Figure 2-1
ANAFAME manages repeated simulation of the faulted and nominal HSPICE files, on a network of UNIX workstations. Simulations are allocated to hosts as they become free, for efficient parallel simulation.

ANACOV performs post-processing analysis on three types of HSPICE analyses: DC Sweep, AC small-signal and Transient. The nominal and faulted responses are compared according to one of 6 detection algorithms to determine the fault coverage. The detection algorithms feature variable pass/fail thresholds to allow for component tolerances. Analysis based on the HSPICE measure command output is also supported. A strobe function allows a subset of all output points to be defined which will be used exclusively during ANACOV analysis. This allows for sharp transient waveform spikes and BIST schemes etc. The strobe points may be viewed graphically using the HSPICE Graphical Simulation Interface.

The MC_RAND program is used to generate sets of random numbers for Monte Carlo analysis using data-driven analysis in HSPICE. This approach is more flexible than the standard “built-in” Monte Carlo analysis in HSPICE. Various distributions of parameters are available and statistics of the generated numbers are available for checking purposes. Use of the MC_RAND program is only necessary if Monte Carlo analysis is to be performed.
3. The ANTICS Environment

In order to allow the user to keep track of simulations and results etc., a simulation environment for the UNIX file system has been developed. The basic structure of this is shown in Fig 3-1.

```
startdir/ _____ basedir/ ______ .anafins.env
|                        |
|___  anafins.mdb         |
|___  run1/ ___________ anafins.idb     
|                        |_____ faulted files..
|___  run2/...
|___ other run directories..
```

**Figure 3-1**

The environment is created by running ANAFINS fault injection program from the start directory with the -init qualifier. This does not perform fault injection but creates the subdirectory structure. Subsequent executions of the ANAFINS program (without the -init qualifier) cause the program to create separate numbered run directories (run1, run2...) every time the program is run. This allows the separation of different fault simulations on the same circuit. The environment file (.anafins.env) is used to keep track of the current environment information e.g. the most recently used run number.

The base directory contains control files, results files and is the directory where ANTICS programs are executed if the environment is being used. It is possible to run ANAFAME and ANACOV without using the environment structure.

The run directories contain spice simulation files for the nominal and faulted netlists (input and output files) and a fault injection database file, created by ANAFINS.
4. Analysis Types Supported and Structure of Spice Input File

ANTICS supports the use of three standard HSPICE simulation analysis types: AC small signal, DC sweep and Transient analysis. The fundamental input to the ANTICS software is an HSPICE input file that describes the circuit under test (netlist) with some form (or forms) of input stimulus and up to three types of analyses to be performed. The HSPICE .PRINT or .MEASURE commands cards must be used to produce a text output of the analyses results in the HSPICE output file. All three types of analysis may be analysed using the same HSPICE input file, with a .PRINT statement determining which fields (nodal voltages or branch currents) are to be observed for each analysis type. The structure of a suitable input file is shown below:

```
BASIC 2-STAGE OPAMP AC AND DC TEST
* .OPTIONS NUMDGT=7 NOPAGE NMOD INGOLD=2 .WIDTH OUT=132
* Fault Free Model Cards NMOD,PMOD
  .model nmod nmos level=2 vto=0.8 kp=78e-6 cgso=110e-11 cgdo=110e-12
  + rsh=20 cj=280e-6 mj=0.48 cjsw=190e-12 mjsw=0.16 nsub=2.1e+15
  .model pmod pmos level=2 kp=28e-6 pb=0.51 cgso=180e-12 cgdo=180e-12
  + rsh=25 cj=280e-6 mj=0.48 cjsw=290e-12 mjsw=0.28 tox=315e-10 nsub=3.9e+15
* D G S B L W AD AS PD PS NRD NRS
*--------------------------------
*input pair
   MN1 5 1 3 3 NMOD 5u 3u
   MN2 6 2 3 3 NMOD 5u 3u
*current source
   MN5 3 4 0 0 NMOD 4u 18u
*output current source
   MN7 7 4 0 0 NMOD 14u 56u
   MN8 4 4 0 0 NMOD 45u 14u
*load pair
   MP3 5 5 99 99 PMOD 15u 10u
   MP4 6 5 99 99 PMOD 15u 10u
*output
   MP6 7 6 99 99 PMOD 5u 400u
   CC 12 7 2.2pF
   MP19 4 4 8 99 PMOD 4u 1u
   MP20 8 8 9 99 PMOD 42u 10u
   MP21 9 9 99 99 PMOD 42u 10u
   MN9 10 4 99 0 NMOD 20u 10u
   MP10 10 10 11 99 PMOD 5u 100u
   MP11 11 11 99 99 PMOD 5u 100u
   MP12 12 10 6 99 PMOD 50u 650u
   CL 7 0 10pF
** voltage sources
   VINP 2 0 2.5V
   VINN 1 0 DC 2.4995V AC 1V
   VSUPPLY 99 0 5V
** control and print cards
  .AC OCT 10 1 1K
  .PRINT V(7) V(3) I(VSUPPLY)
  .DC VINN 0 5 0.1
  .PRINT V(7) V(3) I(VSUPPLY)
.END
```
Analysis may only be performed on the fields specified in the `.PRINT` statement(s) or in `.MEASURE` statements for a particular type of analysis. In the example above, only the voltages at nodes 7 and 3 and the supply current (VSUPPLY) may be used.

Some valid HSPICE input files, particularly where formats not described in the HSPICE manual are used, may cause problems for the parser in ANTICS. For further details refer to section 5.6.
5. ANAFINS - Fault Injection

5.1. Overview

Fault injection is the process whereby a nominal (fault-free) circuit netlist is altered, replacing a faulted component with a simulation fault model. Thus with a list of faults to inject on a set of specified components (faultinject.def file) and the appropriate simulation fault model definitions (faultmodel.def file) a set of faulted SPICE netlists can be produced (set of spice input files - faultx.Hspc.) As well as comprising a set of faulted SPICE input files, the output from this program includes a batch file which may be run to provide spice simulation of the faulted circuits and a fault injection database (cctname.idb). This process is illustrated below in Fig. 5-1.

![Fault Injection Diagram](image)

**Figure 5-1**

5.2. Fault Model Definitions

The lack of a standard fault model for analogue circuits is a well known limitation in this field. For this reason, ANAFINS has no built-in fault models. All fault models are user-defined in a library of faults (faultmodel.def). This allows flexibility as new fault models may be added as they are developed to allow them to be included in fault simulation. Parameterisable fault models may also be created so that one fault model may be called with any number of different component or model values.

The syntax by which a fault is described is very similar to that of the SPICE language. The basic structure of the fault model definition is illustrated below. A fault model definition file may contain one or more such descriptions.

```
.fault modelname element type [arg] [arg..] [fault_model_commands...] .endf
```
Modelname is the name given to the fault model; element is the initial letter used by HSPICE to indicate the type of element; and type is one of a basic category or class of fault. The fault_model_commands may be any SPICE netlists or commands, which are added to the SPICE description of the nominal circuit to produce a faulted circuit. In addition to standard SPICE syntax, it is also possible to use special syntax constructs and commands to allow the fault model to be parameterised and to reference items from the nominal circuit e.g. node numbers of the component under fault. The constructs and commands are described below:

<table>
<thead>
<tr>
<th>Fault Model Definition Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%param</code></td>
</tr>
<tr>
<td><code>#n</code></td>
</tr>
<tr>
<td><code>#keyname</code></td>
</tr>
<tr>
<td><code>##m</code></td>
</tr>
<tr>
<td><code>##p</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fault Model Definition Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.fault modelname element type</code></td>
</tr>
<tr>
<td><code>.fparam param=value</code></td>
</tr>
<tr>
<td><code>.fmodel modelname keyname=expression [keyname=expression]</code></td>
</tr>
<tr>
<td>`.connected nodes... [model</td>
</tr>
<tr>
<td><code>.type name</code></td>
</tr>
<tr>
<td><code>.endf</code></td>
</tr>
</tbody>
</table>
Note that lines beginning "**" are ignored as comments. The first line is ignored as in a SPICE netlist.

An example of a fault model definition of a Drain Open fault is shown in Fig.5-2.

```
**
** MDOP - Mosfet Drain Open
** Parameters are rfault and cfault
**
.fault mdop1 m open
.fparam %cfault=1.8e-16
.fparam %rfault=10 Meg
.connected #1 #2 onode #4 ##m ##p
Rf onode #3 r=%rfault
Cf onode #3 c=%cfault
.endf
```

**Figure 5-2**

The first command line of the fault model definition defines a fault called mdop1 applicable only to elements beginning with the letter m, i.e. MOS transistors. The fault model is of class open (basic category of fault). The fault is only applicable to elements beginning with the letter m. Two parameters, %rfault and %cfault are defined, using the .fparam command, with default values 1.8e-16 and 10 Meg respectively. The default values may be overridden when the fault model is called by the fault injection definition. The .connected statement is used to specify that the existing component is present in the faulted netlist, in this case with an altered node name - the previous node 3 is replaced by an additional ONODE, since that is the third parameter. The next two lines define the instances of the circuit components in the simulation fault model. The final .endf command ends the fault model definition.

It is possible to write a fault model for any component in a HSPICE netlist. Furthermore, component model parameters (HSPICE .model statement) may also be modified. An example would be the changing of a transistor beta value or oxide thickness. Similarly, fault models are not limited to catastrophic models; a parametric resistor fault is described below:

```
**
** RPARAM - Resistor Parametric
** resistor is scaled by a factor of k
** default is fault free
**
.fault rparam r parametric
.param k=1.0
.connected #1 #2 r='r*%k'
.endf
```

The arguments ([arg] [arg..]) in the fault model definition are for use with fault redundancy and equivalence checking. They are currently only used for short circuit models to which they may be applied and are essential for equivalence and redundancy checking. The arguments should correspond to the two nodes n1 and n2 involved in the short circuit model definition:
An example is shown below for a MOSFET gate-drain short which has nodes 1 and 2 shorted.

```
**
** MGDS - Mosfet Gate-Drain Short
** Parameters is rfault
**
.fault mgds1 m short 1 2
.fparam %rfault=1
.connected #1 #2 #3 #4 ##m ##p
Rf #1 #2 r=%rfault
.endf
```

If net names or element names used in fault modes happen to match those already present in the circuit, then the name of the fault nets and elements are automatically modified during fault injection.

### 5.3. Fault Injection Specifications

In order to describe which components within a netlist are to be replaced with a particular fault model, a fault injection specification file is used (faultinject.def). The fault injection specification is made with reference to the original circuit netlist and the fault model definition. The basic structure of a fault injection specification is shown below. A fault injection file may contain any number of these specifications.

```
.inject faultmodel [param=value...] [(sweep param from to by)...
[faultinjection commands]
.endi
```

Faultmodel is the name of a fault model definition; param=value are optional specifications of parameter values (the parameters must be present in the fault model definition). If no parameters are specified in the fault injection specification then the default values from the fault model definition will be applied. The fault injection commands allow the user to specify explicitly which HSPICE components are to be replaced. The injection commands are described below.

| `.inject faultmodel [param=value] [(sweep..)]` | This is the start of a fault injection specification |
| `.endi` | This specifies the end of the fault injection specification. |
| `.only instance [instance]` | This specifies that the particular fault is to be injected only on the specific named component instances. Component instances may be named explicitly or using wildcards. The standard HSPICE notation using full stops for defining component instances in hierarchical circuits is |
The "*" wildcard matches all appropriate instances at a particular level e.g. x1.* matches only those components within the x1 subcircuit. To match all instances in subcircuits below x1 in the hierarchy, the "..." wildcard is used.

| .except instance [instance] | This is similar to the .only command, but specifies that all instances of components at all levels are to be injected except that the named instances are excluded. The syntax for the specification of the instances to be matched is the same as for the .only command. |
| .all | This is the default which specifies all instances of components at all hierarchical levels. Its use is effectively redundant |

An example fault injection specification is given below showing 2 fault injection specifications.

```
** catastrophic drain open fault
.inject mdop1 rfault=1G cfault=1e-15
.endi

** parametric resistance fault R-10%
.inject rparam k=0.9
.only x1.r1   x2...   x3.*
.endi
```

The first fault injection specification will be applied to all instances in all hierarchies to which the mdop fault model (Fig.5-2) can be applied i.e. all MOS transistors. The parameters rfault=1G and cfault=1e-15 are used in formation of the fault. The second specifies resistor r1 in subcircuit x1 (by using x1.r1), all resistors in subcircuit x2 and all resistors within subcircuits of x2 (using x2...), and all resistors in subcircuit x3 but not those in subcircuits below x3 (with x3.*).

It is also possible to sweep fault parameter values, which avoids the need to repeatedly specify fault injections of the same type with different parameter combinations. Using the sweep keyword followed by the initial value, final value and sweep step size provides a sweep of parameter values.

```
** sweep for gos fault
.inject mpgos1 sweep k 0.2 0.8 0.2
.endi
```

In this case, 4 faulty circuits will be generated with values of k from 0.2 to 0.8 with a step size of 0.2. If multiple sweep statements are used, every combination of parameters is generated. Thus,

```
** sweep for gos fault
.inject mpgos1 sweep k 0.2 0.8 0.2 sweep rfault 1 100 25
.endi
```
generates 16 fault files.

Additional comments:
1. Any number of injection specifications may be present in a fault injection file.
2. Lines beginning with “#” are comments and are ignored.
3. The first line of the fault injection file is ignored.

### 5.4. The Fault Injection Database

To enable the ANTICS programs to keep track of the faults that have been injected and their filenames, a fault injection database file is created. Information as to the type of fault injected, parameter values etc. are stored in this file. This file is normally only accessed by the tools, but has a text based format, allowing the user to check the fault injection process. The structure of a typical file is shown below.

```
ANTICSFIDB 2.5 Wed Aug 28 15:16:35 1996
Good_circuit_name= shold
Number_of_faulty_circuits= 152
No. model inst filename ext start end class type equiv redun status nparams
0   mgds1 xa7.xs3.m8 xa7_xs3_m8_mgds1 Hspc 0 0 default 0 -1 0 0 1 rfault 1
1   mgss1 xa7.xs3.m8 xa7_xs3_m8_mgss1 Hspc 0 0 default 1 -1 0 0 1 rfault 1
3   mgpl1 xa7.xs3.m8 xa7_xs3_m8_mgpl1 Hspc 0 0 default 1 -1 0 0 1 rfault 1
4   mdop1 xa7.xs3.m6 xa7_xs3_m6_mdop1 Hspc 0 0 default 1 -1 0 0 1 rfault 1
5   msop1 xa7.xs3.m6 xa7_xs3_m6_msop1 Hspc 0 0 default 1 -1 0 0 1 rfault 1
6   mgds1 xa7.xs3.m6 xa7_xs3_m6_mgds1 Hspc 0 0 default 0 -1 0 0 1 rfault 1
7   mgss1 xa7.xs3.m6 xa7_xs3_m6_mgss1 Hspc 0 0 default 0 -1 0 0 1 rfault 1
8   mdop1 xa7.xs3.m6 xa7_xs3_m6_mdop1 Hspc 0 0 default 1 -1 0 0 1 rfault 1
9   msop1 xa7.xs3.m6 xa7_xs3_m6_msop1 Hspc 0 0 default 1 -1 0 0 1 rfault 1
10  mgds1 xa7.xs3.m5 xa7_xs3_m5_mgds1 Hspc 0 0 default 0 -1 0 0 1 rfault 1
11  mgss1 xa7.xs3.m5 xa7_xs3_m5_mgss1 Hspc 0 0 default 0 -1 0 0 1 rfault 1
12  mdop1 xa7.xs3.m5 xa7_xs3_m5_mdop1 Hspc 0 0 default 1 -1 0 0 1 rfault 1
13  msop1 xa7.xs3.m5 xa7_xs3_m5_msop1 Hspc 0 0 default 1 -1 0 0 1 rfault 1
14  mgds1 xa7.xs3.m8 xa7_xs3_m8_mgds1 Hspc 0 0 default 0 -1 0 0 1 rfault 1
15  mgss1 xa7.xs3.m8 xa7_xs3_m8_mgss1 Hspc 0 0 default 0 -1 0 0 1 rfault 1
16  mdop1 xa7.xs3.m8 xa7_xs3_m8_mdop1 Hspc 0 0 default 1 -1 0 0 1 rfault 1
17  msop1 xa7.xs3.m8 xa7_xs3_m8_msop1 Hspc 0 0 default 1 -1 0 0 1 rfault 1
18  mgds1 xa7.xs3.m8 xa7_xs3_m8_mgds1 Hspc 0 0 default 0 -1 0 0 1 rfault 1
```

The `redun` column is used to keep track of redundant faults. A 1 in this column indicates a redundant short. The `equiv` column stores equivalent fault information. A -1 indicates that the fault is not equivalent to any other in the current simulation run directory. A -2 in this column indicates that the fault is used to represent a fault equivalence class. Any other number indicates that the fault is part of a fault equivalence class but is not used to represent the fault equivalent group. The number in the `equiv` column refers to the fault number of the fault used to represent the fault equivalence class. For example in the database shown above, faults 12, 15 and 5 are all part of an equivalent class and fault 5 is used to represent this class. Only fault 8 is redundant.

Note that equivalencies are only checked for the set of faults that are injected by ANAFINS in a given run. For example, if ANAFINS was run on a circuit just using gate drain short faults and subsequently using gate source short faults, any equivalent gate drain short and gate source short faults would not be detected as equivalent and the fault would effectively be duplicated in the different run directories.

Information from both the `equiv` and `redun` column are used by both ANAFAME and ANACOV.

### 5.5. The ANTICS Simulation Batch File
If ANAFAME is not to be used for repeated simulation e.g. only one host is available, it is possible to use a batch file created by ANAFINS to simulate the files on one host. This file may be sourced or run as a batch job. The default filename of the simulation batch file for run directory $x$ is $\text{spicefsim\_runx}$.

### 5.6. HSPICE Syntax and ANAFINS

ANAFINS parses the input HSPICE netlist as part of its operation, and there will occasionally be times when ANAFINS fails to recognise language structures that are recognised by HSPICE. HSPICE does not seem to stick rigidly to the syntax defined in the HSPICE user guide and thus will tolerate input descriptions that differ from the examples and definitions given. The exact parsing process used by HSPICE is unknown.

If difficulty arises, one work-around is to change the layout of the input file to one which is simpler and uses language constructs which conform more directly with those in the manual.

Other effects of the ANAFINS parser are as follows:

1. The formatting of the original HSPICE input file is not preserved in the ANAFINS-generated SPICE input files. In particular, most blank lines are removed and spacing (tabs and spaces) are generally reduced to one space. The spacing and formatting of comments is not always preserved.

2. Comments may move slightly in relation to the rest of the SPICE description in the ANAFINS-generated files. This is particularly the case where comments are in the middle of continuation lines or near .include statements.

3. Long lines in the original HSPICE input file will be replaced with continuation lines. If the original file contained continuation breaks, the position of these will not necessarily be preserved.

4. Full checking of sub-circuit scope and visibility is not performed by ANAFINS so if (for example) sub-circuit B is defined within sub-circuit A then an error will be generated by HSPICE if B is used at the top level, but not by ANAFINS.

**Note that ANAFINS should never be used on circuit descriptions which give HSPICE errors.** Moreover, HSPICE should always be used to simulate circuits to check for errors before ANAFINS is used to inject faults.

### 5.7. Program Execution

Before fault injection can take place, the ANTICS environment should be set-up using the -init qualifier:

```
anafins -init circuitname
```
Subsequent execution of the program performs fault injection and is performed using the command:

```
anafins circuitname
```

A set of qualifier commands and flags can be used to control the operation of the program. Qualifiers may be specified in three ways; in a user initialisation file, in an initialisation file in the working directory, or on the command line. They are assigned in the following order:

- Default program value
- Environment variable
- Command in `~/.anafins.ini`
- Command in `[current_directory]/anafins.ini`
- Command line qualifiers

For example, a command line qualifier will overwrite any qualifier set in the anafins.ini files. Current_directory is the directory in which anafins is run. The complete set of qualifiers is listed in Section 5.7. The format of the anafins.ini files is as follows:

Each line starts with a valid qualifier expression in exactly the same form as would be found on a command line, except that no qualifier character, “-”, is required. Anything after a qualifier expression on a line is ignored, allowing for comments after each qualifier. Lines beginning with '!' are ignored to allow for longer comments. Blank lines are also ignored.

The default values used may not be applicable to all users and thus, the default values may be changed using a number of environment variables. Setting the environment variable in the users `.cshrc` file (or the global `.cshrc` file for all users) effectively alters the program default value. An example would be setting the ANAFINSMODELDEF environment variable (using the `csh` command “setenv”) to point to a fault model definition file used by a research group. This ensures that all users use the same models. The command line qualifier `-modeldef` could be used to override the default if, for example, new fault models were being developed.

The following environment variables are recognised by ANAFINS:

- `ANAFINSMODELDEF` Fault model definition file
- `ANAFINSINJECTDEF` Fault injection specification file
- `ANAFINSSPICEINEXT` SPICE input file extension
- `ANAFINSSPICEOUTEXT` SPICE output file extension
- `ANAFINSRUNROOT` Run directory name
- `ANAFINSMODELLDB` Compiled fault model database
- `ANAFINSDEFCCLASS` Default class name for user-defined fault classes
- `ANAFINSBATCH` Fault simulation batch file name
- `ANAFINSENVFILE` ANAFINS environment file

### 5.8. List of ANAFINS Qualifiers
Standard Qualifiers:

- **[no]afi[=filename]**
  Specifies the name of the text file containing reports from the fault insertion process. These reports are produced if the appropriate qualifiers (e.g. -pisf and -piml) are used. The default is cctname.afi. If -noafi is used the report is produced on the standard output. The default is -afi.

- **[no]errorlog[=filename]**
  Specifies whether or not to log error messages etc. to a file, and the name of the logfile if required. If no error log file is specified the default is anafins.err. The default is -noerrorlog, that is no errorlog file is produced.

- **injestspec=filename**
  Specifies the name of the fault injection specification file. The default is faultinject.def unless the environment variable ANAFINSINJECTDEF is set, in which case that provides the default value.

- **spiceinext=ext**
  Specifies the file-name extension, used for SPICE input files. The default is .Hspc unless the environment variable ANAFINSSPICEINEXT is set in which case that provides the default.

- **spiceoutext=ext**
  Specifies the file-name extension, used for SPICE output files. The default is .out unless the environment variable ANAFINSSPICEOUTEXT is set in which case that provides the default.

- **rundir=filename**
  Specifies the run directory root name. The default is run unless the environment variable ANAFINSSRUNROOT is set in which case that provides the default. This name is used to form the run directory name by adding the run number. The run directories contain the faulted circuit spice files.

- **basedir=filename**
  Specifies the base directory. This is the directory containing the environment information, compiled fault models and run directories. The run directories contain the faulted circuit spice files.

- **initialise**
  Initialise rather than run. ANAFINS must be run in this mode first before faulted circuit files can be generated. During initialisation the base directory is created if it does not already exist; the fault model definitions are compiled; and the environment information is set up.

- **batchfile=filename**
  Specifies the batch file root name. The batch file can be sourced or run as a batch job in order to simulate the faulted circuits. This will run on the machine on which it is started and, unlike using ANAFAME, does not provide the option of distributed
processing. The default is `spicefsim` unless the environment variable ANAFINSBATCH is set in which case that provides the default.

`-faultmodeldb=filename`
Specifies the fault model database file name. This is created in initialisation mode upon successful compilation of the fault model definition file. The database file is placed in the base directory. Its default filename is `anafins.mdb` unless the environment variable ANAFINSMODELEDB is set in which case that provides the default.

`-modeldef=filename`
Specifies the fault model definition file. This is compiled when ANAFINS is run in initialisation mode. The default filename is `faultmodel.def` unless the environment variable ANAFINSMODELDEF is set in which case that provides the default.

`-[no]overwrite`
Allow overwrite of existing run directory. The default is `nooverwrite`.

`-runnumber=n`
Sets n as the current run number. This is used to form the run directory name by adding it to the run directory base name. The run directories contain the faulted circuit spice files. The default is previous run number plus 1, or 1 for the first run.

`-[no]rerun`
Specifies whether or not to overwrite the previous run directory for this run. The default is `norerun`.

`-[no]reinit`
Specifies whether or not to allow reinitialisation of existing base directory. The default is `noreinit`.

`-[no]createspice`
Specifies whether or not to create fault- injected spice files in the run directory. The default is `createspice`.

`-[no]checkredund`
Specifies whether or not to check for redundant faults. The default is `checkredund` (i.e. check for redundant faults).

`-[no]checkequiv`
Specifies whether or not to check for equivalent faults. The default is `checkequiv` (i.e. check for equivalent faults).

`-[no]supequivm`
Specifies whether or not to suppress warning messages about fault equivalences and redundancies. Default is not to suppress them (`nosupequivm`).

**Diagnostic qualifiers:**
-\[no\]pxil
Specifies whether or not to print the expanded instance list diagnostic report to the "afi" file (see -afi).

-\[no\]pdfdf
Specifies whether or not to print the fault definitions diagnostic report to the "afi" file (see -afi). Default is -nopdfdf.

-\[no\]pisf
Specifies whether or not to print the internal spice file diagnostic report to the "afi" file (see -afi). Default is -nopisf.

-\[no\]pctd
Specifies whether or not to print the circuit topology data diagnostic report to the "afi" file (see -afi). Default is nopctd.

-\[no\]pafm
Specifies whether or not to print the available fault models diagnostic report to the "afi" file (see -afi). Default is -nopafm.

-\[no\]ptap
Specifies whether or not to print the transient analysis points diagnostic report to the "afi" file (see -afi). Default is -noptap.

-\[no\]piml
Specifies whether or not to print the instance match list diagnostic report to the "afi" file (see -afi). A match list will only be present if the only or except commands have been used in the fault injection specification. Default is -nopiml.

-\[no\]pspl
Specifies whether or not to print the spice parameter list diagnostic report to the "afi" file (see -afi). Default is -nopspl.

-\[no\]pdca
Specifies whether or not to print the dc analysis requests diagnostic report to the "afi" file (see -afi). Default is -nopdca.

-\[no\]pmod
Specifies whether or not to print the SPICE models report to the "afi" file (see -afi). Default is nopmod.

-\[no\]paca
Specifies whether or not to print the ac analysis requests to the "afi" file (see -afi). Default is nopaca.

-\[no\]ptra
Specifies whether or not to print the transient analysis requests to the "afi" file (see -afi). Default is noptra.
-[no]pfnl
Specifies whether or not to print the flat SPICE netlist to the "afi" file (see -afi). Default is nopfnl.

-[no]plfn
Specifies whether or not to print the list of flattened nets to the "afi" file (see -afi). Default is noplfn.
6. **ANAFAME - Fault Simulation Management**

6.1. **Overview**

The simulation of the faulted and nominal spice input files is achieved using ANAFAME. This is the most time consuming part of the AFS process, requiring \( n+1 \) simulations, where \( n \) is the number of faults injected. In order to reduce the amount of “user” simulation time, the simulation of the faulted and nominal circuits can be distributed over a network cluster of workstations. A remote shell is started on each host and the simulation is executed. The inputs and outputs to the program are shown graphically below (Fig.6-1)

![Figure 6-1](image)

6.2. **Simulation on Remote Hosts**

The hosts database file (hosts.db) is used to store the names of the hosts used to run the remote hspice processes. The structure of the hosts database file is evident from the example shown below:

```
3
bart
crystal
dex
```

The first line indicates the number of hosts to be used and the subsequent lines are the names of the hosts. This file must specify the name of the host on which the ANAFAME program is executed as one of the hosts if it is to be used for simulation.
If the name of the local host is not found in the /etc/hosts.equiv file on the remote machine, and the local username and hostname are not found in the remote user's .rhosts file, then the access to the host will be denied and ANAFAME will fail to allocate the faults.

6.3. Ping Command check

It is possible that not all machines listed in the hostname file are available at the time of running the program. If this is the case, the program would hang while it waits to start a shell on a remote machine. Using the (default) -pinghosts qualifier helps to ensure that the machines are available to simulate the faulty files. The UNIX ping command is used to establish a list of hosts which are available; any machine not responding to the ping command is not used in simulation.

In order to cope with the different messages from the ping command, ANAFAME uses a file (ping.msg) which contains a list of possible responses from hosts to the ping command. A default file is available, however, it may be necessary to add additional responses to this file as they occur. If ANAFAME encounters a message in the ping response from a host that is undefined then it will look at the next line of the response until there are no more and then stop with an error message and the undefined ping response. The undefined response should then be added to the ping.msg file with a field to say if it is available for use or not for subsequent runs. This is to aid portability between systems.

The structure of the ping.msg file is as follows:

The first line should contain the number of responses. Subsequent lines should contain a response (or a unique part of a response line) in quotes followed by a space and then either the number 1 or 0. 1 represents the host can be used for the given response 0 if it cannot. An example is given below:

```
2
"is alive" 1
"no response from" 0
```

6.4. Program Operation

After performing the ping command if applicable, ANAFAME initially distributes the fault simulation processing until there are no processors that have not been allocated a SPICE simulation. The software will then poll for finished simulations and allocate simulations accordingly.

ANAFAME obtains the names of the fault files to be simulated from the fault injection database file (cctname.idb file). Redundant faults are not simulated and only one fault of a fault equivalence class is simulated. This is consistent with ANACOV which does not look for the output from these files by default.
During simulation a set of SPICE error output files (*.err) is produced (these are automatically deleted if the -tidyup qualifier is used). The files contain the standard error output from the spice simulations and include information as to the completion of the SPICE simulation (i.e. concluded/failed) and the simulation time.

A file (anafame.status) is written to disk in the run directory when the program is started and deleted on successful completion of the program. If the program is terminated (e.g. using ^Z) before completion this file will still be present. Subsequent runs of ANAFAME require that this file is removed. This is to remind users that the run directory must be cleared of all *.err and *.batch files before ANAFAME is re-run.

The following table contains a list of file output extensions:

<table>
<thead>
<tr>
<th>Extension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.batch</td>
<td>These files are used by ANAFAME and should be deleted after the program has run.</td>
</tr>
<tr>
<td>*.err</td>
<td>These are the standard SPICE error output files - they give information on runtime and concluded/aborted information.</td>
</tr>
<tr>
<td>*.out [or default spice output file extension]</td>
<td>These are SPICE output files.</td>
</tr>
<tr>
<td>*.Hspc [or default spice input file extension]</td>
<td>These are SPICE input files.</td>
</tr>
<tr>
<td>*.st0</td>
<td>These are the SPICE standard output files with runtime information for each simulation.</td>
</tr>
</tbody>
</table>

### 6.5. List of ANFAME Qualifiers

- **-spice_extn="extn"**
  Changes the default SPICE output file extension to extn. The default is out.

- **-ip_file_extn="extn"**
  Change the default SPICE input file extension to extn. The default is Hspc.

- **-db_extn="extn"**
  Change the default database file extension to extn. The default is idb.

- **-host_file="filename"**
  Change the default hostfile filename (and path) to filename. The default is hosts.db.

- **-[no]convert**
  Specifies that ANAFAME is to convert the faultlist.txt file produced from the OLDANAFINS program to a database file. This is for backwards compatibility with OLDANAFINS and is not needed for use with version 2. The default is -noconvert.

- **-[no]batchjob**
NOT CURRENTLY SUPPORTED. Specifies that ANAFAME is to simulate the SPICE files as a batch job rather than a process running in the background. This should be left as the default. The default is -batchjob.

-\[no\]use_env
Use environment information set up by ANAFINS. If this qualifier is set the environment information from ANAFINS ".anafins.env" file is used to determine the run directory for the circuit. With this qualifier set, the ANACOV program should be run from the working directory rather than the run directory. Without this qualifier set the program should be run from the run directory. The default is -use_env.

-sim_goodfile="always|never|iffound"
Specifies whether to simulate the fault-free file.

- always - ANAFAME looks for the fault-free spice input file and if it is present it is simulated else an error message is produced.
- never - do not try to simulate good file
- iffound - only simulate file if it is found

The default is -iffound.

-\[no\]run_number[="n"]
Specify run directory in which to run ANACOV when using the antics environment directory structure. This should only be used with the "use_env" qualifier. "n" is the run number containing the files to perform analysis on. If the "use_env" qualifier is used without the "run_dir="n"" qualifier then the run directory is set to the value in the .anafins.env file. The default is -norun_number.

-\[no\]ping_hosts
Specifies whether to check if the hosts in the hosts.db file are alive using the UNIX ping command. The default is -ping_hosts.

-\[no\]tidyup
Specifies whether to clear the run directory of *.err and _batch files after all the files have simulated. The default is -tidyup.

-ping_messages_file="filename"
Changes the name (and path) of the ping messages file searched for to “filename”. The default is ping.msg.

-ping_path="filename"
Changes the (full) path to the UNIX ping command. This must include the command itself. The default is /usr/etc/ping.
7. ANACOV - Post-processing Analysis

7.1. Overview

The final post-processing analysis is achieved using ANACOV. This performs a decision on whether a given fault condition of a circuit is detectable or undetectable for a given input and type of analysis. The inputs and outputs to the program are shown graphically in Fig. 7-1.

![Figure 7-1](image)

The SPICE output responses of the faulted and nominal circuit (*.out) are compared using one of several post-processing detection algorithms. The coverage definition file (coverage.def) is used to specify which analysis detection algorithms are to be applied to which analysis types (i.e. transient, DC sweep or AC small signal) and includes additional control statements such as thresholds and strobe point definitions. The strobe points used and a fault coverage histogram may be viewed graphically using the HSPICE Graphical Simulation Interface. A file with results from a sweep of thresholds is also available. The syntax and structure of the coverage definition file (coverage.def) is discussed in section 7.11.

7.2. Detection Modes
The detection algorithms described below are applied to the nominal and faulted spice output file responses to determine the fault coverage. To allow for the fact that two fault-free circuits within specification may produce two different measured output responses due to factors such as process parameter variations, noise, component tolerances, etc., a threshold envelope may be applied around the response of the good circuit. Since this is also true of a circuit under fault conditions, a threshold envelope around a faulty response is also justified and may be applied.

The detection algorithms essentially compare the nominal and fault-free responses at a number of sample points in the simulation response output file. Depending on the response values, detection mode and threshold envelope criteria, a particular sample point in a response may be regarded as detectable or undetectable. The number of points required to be detectable before a fault is classed as detectable can be set to more than one to increase the confidence in the results.

### 7.2.1. Fixed

![Figure 7-2](image)

Fig. 7-2 shows the use of detection threshold envelopes around a fault-free and faulted circuit response. Sample points (1 to 3) where the threshold envelopes do not overlap are classed as detectable. If more than \( x \) points are detectable, where \( x \) is a variable user-defined cut-off, the fault is detectable. In addition to these measurements of detectability, the distance between the threshold bound values at a given sample point may be used as a confidence measure; \( F_L[i] - G_U[i] \) (or \( x[i] \)) for the case of sample points 1 to 3. A larger distance implies that at that sample point the circuit under a particular fault condition is more easily detectable since it deviates from the fault-free case by a greater amount. This algorithm may be applied equally to any type of analysis supported by ANTICS, so a sample points could represent a time (transient), a frequency (AC) or an input value (DC).
7.2.2. Variable

One of the main drawbacks to the fixed detection algorithm is that the tolerances are assumed to be fixed throughout the entirety of the waveform. In practice, different parts of a response would require more or less tolerance. The variable detection algorithm allows the upper and lower bounds on a fault-free circuit to be derived from a previously obtained SPICE output response. This facilitates the use of best/worst case analysis, sensitivity analysis, temperature variations, parametric deviation estimates, best/worst foundry to be used directly as a detection envelope threshold. An additional fixed threshold may also be applied around the fault-free and faulted responses to allow for other effects such as measurement noise. Detection criteria and confidence measures are the same as those for the fixed algorithm.

7.2.3. Threshold

The threshold detection mode uses two fixed reference levels to produce a detectable/undetectable decision. Points from a faulty response lying outside of these levels are classed as detectable - see Fig. 7-3. Faulty responses may also have an additional fixed threshold applied as in the fixed mode. The confidence measurements are applicable as in the fixed detection mode.

![Figure 7-3](image)

Examples of typical applications of this detection mode are when evaluating the fault coverage of supply current measurements for $I_{DDQ}$ quiescent supply current monitoring, and any BIST scheme that uses fixed reference levels e.g. an on-line safety-critical system test.

Any conveniently low value may be used as the low threshold if only the upper threshold is required e.g. there is no low threshold in $I_{DDQ}$ testing.
7.2.4. Digital

The digital detection mode was included to enable a digital ATE set-up to be modelled. Although ANTICS was primarily designed as a fault simulator for analogue circuits, there is no reason why digital or mixed signal circuits cannot be simulated.

When the digital detection mode is utilised, a decision is made as to whether the field under test is logic high(1), logic low(0) or floating(X), at a specific point. This digitisation is applied to both the nominal and faulted netlists. The test is thus purely Boolean. The thresholds for logic 1 and 0 are user definable. Sample points lying between the upper and lower logic thresholds (i.e. floating) for either the faulty of fault-free circuit are not used in comparison.

Realistically, this detection mode is only applicable to a transient analysis and in general, the cutoff value will be 1 and the strobe option will be used with strobe points occurring after the system clock.

7.2.5. Alldata

The detection modes discussed in the previous sections all use some form of fixed threshold or fixed reference level to be applied during the post-processing detection. In practice, these levels will not be fixed and the alldata and data detection algorithms are designed to account for this by using Monte Carlo analysis.

The alldata detection mode uses a set of HSPICE output responses in its algorithm. The good circuit netlist and every faulty netlist is simulated using the HSPICE sweep .data command. Thus the output responses (*.out files) contain the response to the data driven analysis. This output takes the form of several circuit responses in succession. An example of part of the spice input netlist is shown below:

```plaintext
** data driven monte carlo analysis
.DATA mc_data MER
FILE= 'mc_data.inc'
+ox=1
+latdiff=2
+uop=3
.ENDDATA
* put circuit description here...

* define models with parameters to be altered by mc_data
* Note these are not complete definitions
* they are just examples

* PTYPE TRANSISTOR
.MODEL MOSP PMOS
+LEVEL = 2
+VTO= vtp
+TOX= ox
+UO= uop
+UCRIT= 4.2K

* NTYPE TRANSISTOR
```
In this example, the data for the parameters to be varied during the simulation is stored in the file "mc_data.inc". The MC_RAND program (see section 8) may be used to generate this file. The parameter values in each line of the "mc_data.inc" file will produce a separate "index" in the SPICE output file - thus a set of output responses ("indices") will be obtained for each fault and the fault-free circuit.

The alldata detection algorithm reads every response from the set of responses in the fault-free output file. For each sample point (time, frequency or dc value), the highest and lowest values from the set of responses are recorded. Thus an upper and lower threshold are generated from the set of responses. This technique is similarly applied to every faulty response in turn to generate an upper and lower envelope for each fault. An example is shown below in Fig. 7-4:

Comparison is then the same as the fixed detection algorithm, using the upper and lower envelopes $F_U[i]$, $F_L[i]$, $G_U[i]$, $G_L[i]$. A fixed threshold may also be applied to the
faulty and good upper and lower envelopes. This allows additional flexibility and could be used to model noise for example

### 7.2.6. Data

The **data** detection algorithm is similar to the **alldata** algorithm except that the fault-free output response is derived from a data-driven Monte Carlo Simulation but the faulty circuit outputs use single responses. Thus the “single” faulty responses are compared against the upper and lower bounds of a data-driven Monte Carlo Simulation. Again, upper and lower fixed thresholds can be applied to the good Monte Carlo response and the faulty single responses.

### 7.2.7. Threshdata

The **threshdata** detection algorithm is similar to the **threshold** algorithm except that data-driven Monte Carlo responses are compared to fixed reference levels, instead of single responses. Faulty upper and lower thresholds may also be applied to the worstcase upper and worstcase lower from the Monte Carlo Simulations.

![Graph](image)

**KEY**
- ■ Faulty Upper
- ● Faulty Lower

### 7.3. Statistical Analysis

As well as indicating the detectability of a the set of faults, further statistical analysis may also be produced which gives information about the distribution of sample points...
as a confidence measure. The information is available for each fault and as overall figures for all of the faults considered.

**Fixed mode:**

Let $G[i]$ be the set of samples from a good circuit response

Let $F[i]$ be the set of samples from a faulty circuit response

Define upper and lower bounds using fixed thresholds, $\delta_{G_L}, \delta_{G_U}, \delta_{F_L}, \delta_{F_U}$:

$$
G_L[i] = G[i] - \delta_{G_L} \quad \text{Lower bound on good response}
$$

$$
G_U[i] = G[i] + \delta_{G_U} \quad \text{Upper bound on good response}
$$

$$
F_L[i] = F[i] - \delta_{F_L} \quad \text{Lower bound on faulty response}
$$

$$
F_U[i] = F[i] + \delta_{F_U} \quad \text{Upper bound on faulty response}
$$

Distance confidence measure $x[i]$ between sample points is:

$$
x[i] = \begin{cases} 
G_L[i] - F_U[i] & \text{for } G_L[i] > F_L[i] \\
F_L[i] - G_U[i] & \text{for } F_L[i] > G_U[i] \\
0 & \text{otherwise}
\end{cases}
$$

**Variable mode:**

Upper and lower bounds for the good circuit are taken from two existing circuit responses. Bounds for faulty circuit are defined using fixed thresholds $\delta_{F_L}, \delta_{F_U}$ around the faulty response:

$$
G_L[i] \quad \text{Derived from lower raw file circuit response}
$$

$$
G_U[i] \quad \text{Derived from upper raw file circuit response}
$$

$$
F_L[i] = F[i] - \delta_{F_L} \quad \text{Lower bound on faulty response}
$$

$$
F_U[i] = F[i] + \delta_{F_U} \quad \text{Upper bound on faulty response}
$$

Distance confidence measure $x[i]$ between sample points is:

$$
x[i] = \begin{cases} 
G_L[i] - F_U[i] & \text{for } G_L[i] > F_L[i] \\
F_L[i] - G_U[i] & \text{for } F_L[i] > G_U[i] \\
0 & \text{otherwise}
\end{cases}
$$

**Threshold mode:**

“Good” upper and lower bounds taken as fixed reference levels. Bounds for faulty circuit are defined using fixed thresholds $\delta_{F_L}, \delta_{F_U}$ around faulty response.

$$
G_L[i] \quad \text{Lower fixed reference level}
$$
**GU[i]** Upper fixed reference level

F_L[i]=F[i]-ΔF_L Lower bound on faulty response

F_U[i]=F[i]+ΔF_U Upper bound on faulty response

Distance confidence measure x[i] between sample points is:

\[
x[i] = \begin{cases} 
G_L[i] - F_U[i] & \text{for } G_L[i] > F_L[i] \\
F_L[i] - G_U[i] & \text{for } F_L[i] > G_U[i] \\
0 & \text{otherwise}
\end{cases}
\]

**Digital mode:**

Statistics and confidence measures are meaningless for this mode.

**Alldata mode:**

Define upper and lower values for the good and faulty circuits from Monte Carlo simulations at each sample point i as:

- G_LMC[i] = Good lower Monte Carlo bound
- G_UMC[i] = Good upper Monte Carlo bound
- F_LMC[i] = Faulty lower Monte Carlo bound
- F_UMC[i] = Faulty upper Monte Carlo bound

Define upper and lower bounds using fixed thresholds, ΔG_L, ΔG_U, ΔF_L, ΔF_U:

- G_L[i]=G_LMC[i]-ΔG_L Lower bound on good response
- G_U[i]=G_UMC[i]+ΔG_U Upper bound on good response
- F_L[i]=F_LMC[i]-ΔF_L Lower bound on faulty response
- F_U[i]=F_UMC[i]+ΔF_U Upper bound on faulty response

Distance confidence measure x[i] between sample points is:

\[
x[i] = \begin{cases} 
G_L[i] - F_U[i] & \text{for } G_L[i] > F_L[i] \\
F_L[i] - G_U[i] & \text{for } F_L[i] > G_U[i] \\
0 & \text{otherwise}
\end{cases}
\]

**Data mode:**

Define upper and lower values for good circuit from Monte Carlo simulation at each sample point i as:

- G_LMC[i] = Good lower Monte Carlo bound samples
- G_UMC[i] = Good upper Monte Carlo bound samples
- F[i] = Faulty response samples
Define upper and lower bounds using fixed thresholds, $\delta G_L, \delta G_U, \delta F_L, \delta F_U$:

- $G_L[i] = G_{LMC}[i] - \delta G_L$  
  Lower bound on good response
- $G_U[i] = G_{UMC}[i] + \delta G_U$  
  Upper bound on good response
- $F_L[i] = F[i] - \delta F_L$  
  Lower bound on faulty response
- $F_U[i] = F[i] + \delta F_U$  
  Upper bound on faulty response

Distance confidence measure $x[i]$ between sample points is:

$$x[i] = \begin{cases} 
    G_L[i] - F_U[i] & \text{for } G_L[i] > F_L[i] \\
    F_L[i] - G_U[i] & \text{for } F_L[i] > G_U[i] \\
    0 & \text{otherwise}
  \end{cases}$$

**Threshdata mode:**

Define upper and lower values for the good and faulty circuits from Monte Carlo simulations at each sample point $i$ and good upper and lower bounds from fixed reference levels:

- $F_{LMC}[i]$ = Faulty lower Monte Carlo bound
- $F_{UMC}[i]$ = Faulty upper Monte Carlo bound
- $G_L[i]$ = Lower fixed reference level
- $G_U[i]$ = Upper fixed reference level

Define upper and lower bounds using fixed thresholds, $\delta F_L, \delta F_U$:

- $F_L[i] = F_{LMC}[i] - \delta F_L$  
  Lower bound on faulty response
- $F_U[i] = F_{UMC}[i] + \delta F_U$  
  Upper bound on faulty response

Distance confidence measure $x[i]$ between sample points is:

$$x[i] = \begin{cases} 
    G_L[i] - F_U[i] & \text{for } G_L[i] > F_L[i] \\
    F_L[i] - G_U[i] & \text{for } F_L[i] > G_U[i] \\
    0 & \text{otherwise}
  \end{cases}$$

For all detection modes

$$\text{number of detectable sample points} = \sum_{i=1}^{n} \det[i]$$

$$\text{fault detectable / undetectable flag} = \begin{cases} 
    1 & \text{when } \sum_{i=1}^{n} \det[i] > \text{cutoff value} \\
    0 & \text{when } \sum_{i=1}^{n} \det[i] \leq \text{cutoff value}
  \end{cases}$$
mean number of sample points detectable = \[ \frac{\sum_{i=1}^{n} (x[i] \text{det}[i])}{\sum_{i=1}^{n} \text{det}[i]} \]

normalised error area = \[ \frac{\sum_{i=1}^{n} (x[i] \text{det}[i])}{\sum_{i=1}^{n} \text{det}[i]} \times \frac{1}{\text{max} - \text{min}} \]

\[ \sum_{i=1}^{n} \text{det}[i] x[i]_i^2 = \frac{\left(\sum_{i=1}^{n} \text{det}[i] x[i]\right)^2}{\sum_{i=1}^{n} \text{det}[i]} \]

Standard deviation = \[ \sqrt{\frac{\sum_{i=1}^{n} \text{det}[i] x[i]_i^2 - \left(\sum_{i=1}^{n} \text{det}[i] x[i]\right)^2}{\sum_{i=1}^{n} \text{det}[i] - 1}} \]

where the \text{det} function:

\text{det}[i] = 1 \quad \text{sample point i is detected}
\text{det}[i] = 0 \quad \text{sample point i is undetected}

and

\text{min} = \text{minimum fault-free value (see below)}
\text{max} = \text{maximum fault-free value (see below)}

The max or min values obtained from the fault-free circuit are used unless \text{min} and \text{max} have been specified using the .quantization statement in the coverage definition file. For the case of data and alldata, this is the max and min for all the indices in the good output HSPICE file. For any other analysis this is simply the max and min values that appear in the fault-free output files. Note that this is the case for threshold, variable and threshdata modes even though the good output file is not used in the analysis. For these modes, if the Normalised Error Area is to be used then the min and max should be specified using the .quantization statement.

7.4. Strobe Definitions

It is often undesirable to include every simulation sample point from an output response in a calculation. For example, consideration should be taken as to the shape
of the waveform for a transient analysis. Some waveforms may have sharp spikes which could not be captured accurately with some ATE. In addition, other circuits require a period of setting up or locking e.g. a PLL before sampling should begin. I\textsubscript{DDQ} monitoring and any BIST scheme that performs measurements at a certain time in a transient waveform also fall into the category of circuits which do not require all points to be used. Similarly in an AC frequency response test, it is unlikely that every frequency will be used, it is more common to test at critical frequencies. For DC testing, measurements may also only occur at specific operating points.

These considerations may be met by using a subset of all response points during post-processing analysis. The strobe option within ANACOV allows a number of strobe points to be specified from the set of all response points and only these points are used in analysis. The strobe points may be specified in one of three ways:

| From-to-by | This allows specific points to be defined as strobe points or groups of points from an instance, to another instance using a particular step size. Any number of these definitions may be used for the same response. The points specified must be within 10\% of the simulation step size or else an error will be generated. Using from “x” to “y” by “z” specifies points up to but not including “y”. |
| File | This uses a strobe file containing a list of times/ frequencies/dc sweep points where strobing should occur. |
| All | This indicates all analysis points should be used during post-processing and is the default. |

The strobe points selected may be viewed graphically alongside (for example) the nominal response so that correct selection of strobe points may be checked. This is described in Sect. 7.5. An example of a selection of strobe points used to avoid fast transients is shown in Fig.7-5.

### 7.5. GSI Plots

ANACOV can create two HSPICE files which may be simulated by the user (the simplest way is to use the \texttt{.include SPICE} statement to include the file and resimulate the good circuit) and then viewed using the HSPICE Graphical Simulation Interface. The first file is a strobe file which allows the user to view the strobe points selected graphically. A comparison should be made with the nominal output response to check the suitability of the strobe points chosen. This is response (2) in Fig.7-5. The nominal response (in this case supply current waveform) is shown (1)
7.6. Raw Files

When running ANACOV, the data from each output response must be extracted from HSPICE output files before analysis can take place. This process can be quite time-consuming, especially for larger files. ANACOV allows the use of “raw files” to reduce the post-processing time after the first analysis. Data from the fields specified in the coverage.def file is saved to a set of raw files in a simplified form. A separate file is saved for each field, for each type of analysis for each faulty response. The naming convention is as follows:

Faultname_typename_field.rawextn

where Faultname is the name of a fault, typename is the type of SPICE analysis, field is the name of the field and rawextn is the rawfile extension (default is “RAW”). Subsequent analyses after the generation of the raw files can use the data.
from the raw files immediately, resulting in a reduced post-processing time. The raw file may consist of all the points from the responses or only the strobed points. The latter is useful when performing more than one analysis using a small number of strobe points from a large response. However, subsequent analyses may only be performed at the corresponding strobe points, or a smaller subset.

### 7.7. Print Options

The output file for a large number of faults could potentially contain a large amount of information. For this reason, print options may be used to specify the format of the output file. Several options are used with the `.printopt` command to specify exactly what information is produced as an output.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userdata</td>
<td>Prints information about the simulation e.g. date, time, run directory and a summary of options and parameters from the definition file (coverage.def)</td>
</tr>
<tr>
<td>allfaults</td>
<td>Prints fault coverage information for each fault, including number of sample points detectable and confidence values.</td>
</tr>
<tr>
<td>allstats</td>
<td>Prints overall statistical analysis for each field for each type.</td>
</tr>
<tr>
<td>coverage</td>
<td>Prints overall fault coverage for each field for each type of analysis</td>
</tr>
<tr>
<td>undetect</td>
<td>Prints a list of undetectable faults for each type of analysis</td>
</tr>
<tr>
<td>separate</td>
<td>Prints results for each field separately.</td>
</tr>
<tr>
<td>newline</td>
<td>Prints a new line between each field which produces a tidier output. Output is grouped according to faults.</td>
</tr>
</tbody>
</table>

### 7.8. Threshold Sweep Mode

In this mode, the fixed thresholds in `fixed` and `threshold` detection modes are swept using a “from-to-by” definition and an output file is produced with two columns: threshold size and fault coverage. This file can be used to generate a graph of threshold vs. fault coverage. This is only applicable to `fixed` and `threshold` mode and the number of points for detection is always assumed to be 1.

### 7.9. Transformations

ANACOV has the ability to change a waveform before the detection algorithm is applied to it using a transformation. Currently only one transform is available (sub_avg) which subtracts the average value from every point on in a waveform. The DC component is thus removed from the waveforms. This is applied to every waveform (good and faulty). For the `alldata` and `data` detection modes, this is applied to every index in the output files containing Monte Carlo values, before the detection algorithms are applied.

### 7.10. Measure Statement Analysis
ANACOV has the ability to perform analysis from results in output files generated using the HSPICE .measure format (see HSPICE manual for mode details). Currently, the only detection mode this is valid for is the **alldata** mode. Analysis is performed on the faulty and fault-free responses using probability distributions obtained from the Monte Carlo simulation data - see Fig. 7.6

![Figure 7-6](image)

The mean and standard deviation value ($\sigma$) are calculated for the fault-free circuit for each .measure variable specified. The +/- $3\sigma$ points are calculated for the fault-free circuit. This is repeated for every faulty circuit in turn. The $3\sigma$ points are used as detection thresholds for each fault. If the $3\sigma$ points overlap then the fault is said to be undetectable (as in Fig. 7-6). If the $3\sigma$ points do not overlap then the fault is undetectable. This may be expressed more clearly as follows:

Let $G_{LMC}$ = the lower good $3\sigma$ point  
Let $G_{UMC}$ = the upper good $3\sigma$ point  
Let $F_{LMC}$ = the lower faulty $3\sigma$ point  
Let $F_{UMC}$ = the upper faulty $3\sigma$ point

The fault is detectable if ($G_{LMC} > F_{UMC}$) or ($F_{LMC} > G_{UMC}$) else it is undetectable.

In addition, it is also possible to specify a fixed limit (or fixed limits) in place of using the good $3\sigma$ Monte Carlo values as good detection threshold points. This corresponds to the case of say a specification test where the limits of the specification are set. These effectively become the $G_{LMC}$ and $G_{UMC}$ points. If only the upper or lower specification is given the only the corresponding decision function [either ($G_{LMC} > F_{UMC}$) or ($F_{LMC} > G_{UMC}$)] is used. For example if the specification was (say) offset voltage and upper=1.5V then a fault would only be detectable if the lower 3-sigma point of the offset voltage was greater than 1.5V.

In addition to a detectable/undetectable decision, the mean, $\sigma$ and distribution test are printed for each fault and the fault-free circuit. The statistical information is NOT calculated for this analysis and should be ignored in the output file.
7.11. Structure of coverage.def File

The coverage definition file (coverage.def) is used to set options and control the analysis technique of the program. This section provides a description of the structure of the file and a summary of the available commands. The commands should be read in conjunction with the corresponding section in the program description.

Commands can be used that either pertain to a type of analysis e.g. transient, AC or a particular field within an analysis. The commands must be placed in the correct part of the file e.g. field commands must be nested within a field declaration. The general structure of the file is as follows:

```
* Coverage Specification file - 1st line is ignored
.analysis [analysis name1]
[commands relating to analysis type 1]
.field [fieldname1]
[commands relating to field1 within analysis1]
.endfd
[commands relating to analysis type1]
.
.field [fieldname n]
[commands relating to field n within analysis1]
.endfd
[commands relating to analysis type1]
.enda
.
.analysis [analysis name n]
[commands relating to analysis type n]
.field [fieldname1]
[commands relating to field 1 within analysis n]
.endfd
.
[commands relating to analysis type n]
.field [fieldname n]
[commands relating to field n within analysis n]
.endfd
[commands relating to analysis type n]
.enda
```

Where [analysis name n] is the name of a type of SPICE analysis (three are currently supported):

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transient</td>
<td>SPICE transient analysis</td>
</tr>
<tr>
<td>AC</td>
<td>SPICE AC analysis</td>
</tr>
<tr>
<td>DC</td>
<td>SPICE DC sweep</td>
</tr>
</tbody>
</table>

[Fieldname n] is the name of a field (i.e. nodal voltage or branch current) specified with the SPICE .print statement in the input file. Analysis is performed for each field specified for each analysis specification.
7.11.1. Global commands

There is currently only one global command. This applies to all fields for all analysis types specified in the coverage definition file.

7.11.1.1. Equivalent options

Syntax:

.equivalent

This option instructs ANACOV to treat equivalent faults as normal and not as equivalent faults. By default this option is off, and equivalent fault groups are treated as one fault. Note that the output files must be present for the all the equivalent faults if this option is set.

7.11.2. Analysis type commands

Commands relating to analysis types are applied to all fields within that analysis. The two commands control the output (print options) and the statistical analysis:

7.11.2.1. Print Options

Syntax:

.printopt [userdata] [allfaults] [allstats] [coverage] [undetect] [separate] [newline]

Description:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userdata</td>
<td>Prints information about the simulation</td>
</tr>
<tr>
<td>allfaults</td>
<td>Prints fault coverage information for each fault</td>
</tr>
<tr>
<td>allstats</td>
<td>Prints overall statistical analysis.</td>
</tr>
<tr>
<td>coverage</td>
<td>Prints overall fault coverage.</td>
</tr>
<tr>
<td>undetect</td>
<td>Prints a list of undetectable faults</td>
</tr>
<tr>
<td>separate</td>
<td>Prints field output separately</td>
</tr>
<tr>
<td>newline</td>
<td>Prints field output on a new line</td>
</tr>
</tbody>
</table>

These are all off by default.

7.11.2.2. Statistical Options

Syntax:

.statistics [sd_per_fault] [det_fault]

Description:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd_per_fault</td>
<td>Calculates standard deviation information for each fault.</td>
</tr>
<tr>
<td>det_fault</td>
<td>Only use detectable faults in standard deviation calculations</td>
</tr>
</tbody>
</table>
By default these are both true.

### 7.11.2.3. Rawfile Options

Syntax:

```
.saveraw [strobeonly]
```

Description:

| .saveraw | Specifies that the files are to be saved as raw files |
| strobeonly | Specifies that only the strobed points are to be saved in the rawfiles |

By default these are both false.

### 7.11.2.4. Points Command

Syntax:

```
.points [number=N]
```

Description:

| .points | Overrides the number of points calculated by ANACOV and expected in the output files. This should be used when ANACOV calculates the wrong number of points automatically. This has been known when two numbers in the SPICE .tran statement divide exactly. |
| number=N | Specifies the number of points. |

This is off by default.

### 7.11.3. Field commands

Commands relating to fields are only applicable to a specific field for a specific analysis type. Six commands have been implemented. Where more than one is given only one should be selected; the syntax changes according to for example the detection mode selected. N is an integer, n is a floating point number.

#### 7.11.3.1. Detection Modes

Syntax:

```
.detection fixed [gut=n] [glt=n] [fut=n] [flt=n] [npoints=N] [threshold_file="filename" from=n to=n by=n]
```

```
.detection variable [ugfile="filename"] [lgfile="filename"] [fut=n] [flt=n] [npoints=N]
```

```
.detection threshold [lower=n] [upper=n] [fut=n] [flt=n] [npoints=N] [threshold_file="filename" from=n to=n by=n]
```
.detection digital [high=n] [low=n] [npoints=N]
.detection alldata [gut=n] [glt=n] [fut=n] [flt=n] [npoints=N]
.detection data [gut=n] [glt=n] [fut=n] [flt=n] [npoints=N]
.detection threshdata [lower=n] [upper=n] [fut=n] [flt=n] [npoints=N]

Description:

<table>
<thead>
<tr>
<th>fixed</th>
<th>variable</th>
<th>threshold</th>
<th>digital</th>
<th>data</th>
<th>alldata</th>
<th>threshdata</th>
</tr>
</thead>
<tbody>
<tr>
<td>gut=n, glt=n</td>
<td>Upper and lower fixed threshold levels respectively to be placed around the fault-free circuit response (or the upper and lower Monte Carlo bounds for data or alldata modes). The default is 0.0. Note that lower thresholds are expressed as positive numbers which are then subtracted.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fut=n, flt=n</td>
<td>Upper and lower fixed threshold envelope levels respectively to be placed around the faulted circuit response (or the upper and lower Monte Carlo bounds for data or alldata modes). The default is 0.0. Note that lower thresholds are expressed as positive numbers which are then subtracted.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ugfile=&quot;filename&quot;</td>
<td>Specifies name of output file for upper threshold in variable detection mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lgfile=&quot;filename&quot;</td>
<td>Specifies name of output file for lower threshold in variable detection mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lower=n</td>
<td>Specifies lower fixed detection value for threshold and threshdata detection modes. This should be the actual value of the fixed threshold (it is not subtracted - it is a fixed value so do not make negative voltages/currents positive.) The default is 0.0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>upper=n</td>
<td>Specifies upper fixed detection value for threshold and threshdata detection modes.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>high=n</td>
<td>Specifies logic 1 threshold value for digital detection mode. The default is 0.0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>low=n</td>
<td>Specifies logic 0 threshold value for digital detection mode. The default is 0.0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>npoints=N</td>
<td>Specifies the cutoff value for the detection i.e. the number of sample points that must be detected before the fault is classed as detectable. The default is 1.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[threshold_file=&quot;filename&quot;]</td>
<td>Specifies that the fixed threshold in fixed or threshold...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
mode is to be swept from n1 to n2 by n3. The results are stored in a file “filename”. All other thresholds should be 0 (e.g. gut=0.0, glt=0.0, fut=0.0, fjt=0.0) for fixed mode and equal to the mid-point of the envelope sweep for threshold mode (make upper=xxx and lower=xxx where xxx is the midpoint of the sweep).

NOTE the number of points for detection is always 1 for this mode. It should only be used in fixed or threshold ANACOV modes. For threshold mode the envelope is swept out symmetrically about the mid-point defined. The threshold in the file should be doubled to give the actual size of the window for the threshold detection mode.

7.11.3.2. Quantization Options

Syntax:

.quantization [min=n] [max=n] [levels=N | bits=N]

Description:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>min=n</td>
<td>Specifies the lower limit for quantization. Any values lower than this will saturate at zero.</td>
</tr>
<tr>
<td>max=n</td>
<td>Specifies the upper limit for quantization. Any values lower than this will saturate at the maximum quantization level.</td>
</tr>
<tr>
<td>levels=N</td>
<td>Specifies the number of quantization levels.</td>
</tr>
<tr>
<td>bits=N</td>
<td>Specifies the number of quantization levels using a number of bits.</td>
</tr>
</tbody>
</table>

The .quantization option is off by default. If max or min are not specified then the max or min values obtained from the fault-free circuit are used. For the case of data and alldata, this is the max and min for all the indices in the good output HSPICE file. For any other analysis this is simply the max and min values that appear in the fault-free output files. This is the case for variable and threshdata mode even though the good output file is not used in the analysis. Thus for the variable and threshdata modes min and max should be specified if data quantization is required.

The options min and max are also used to specify the bounds for the Normalised Error Area. For the threshdata and threshold modes, which do not read in a fault-free file, this is the only way of setting these bounds. If it is necessary to set these bounds but quantization is not required then this is possible by only specifying the min and max and not specifying levels or bits. See also Section 7.3.

Note that threshold and digital mode do not have the provision for data quantization.

7.11.3.3. Strobe Modes

Syntax:

.strobe all [write="filename"]
.strobe specific [n1] [n2] [n3] ... [n][n][n] [write="filename"]
.strobe file="filename" [write="filename"]

Description:

<table>
<thead>
<tr>
<th>all</th>
<th>Use all sample points</th>
</tr>
</thead>
<tbody>
<tr>
<td>specific n n n</td>
<td>Use specified sample points. Points are specified from n1 to n2 by n3.</td>
</tr>
<tr>
<td>file=&quot;filename&quot;</td>
<td>Specifies name of file containing strobe points. The file should contain a list of times/frequencies/dc-levels in ascending order with the spice output file, each separated by a newline. Standard spice scale factors and exponential form is allowed.</td>
</tr>
<tr>
<td>write=&quot;filename&quot;</td>
<td>Specifies name of file to write strobe points to allow viewing with the GSI</td>
</tr>
</tbody>
</table>

The default is .strobe all.

NOTE:
Any strobe points which are not quantizable to a simulation time/frequency/dc_level within 10% of the step time will result in an error message. Using from "x" to "y" by "z" specifies points up to but NOT including "y".

7.11.3.4.Histogram Modes

Syntax:
.histogram [points][accumulate]

Description:

| .histogram                        | Specifies that a histogram is to be produced to be viewed using the GSI |
| points                           | Specifies do a non-cumulative histogram |
| accumulate                       | Specifies do a cumulative histogram |

This option is off by default and both arguments are off by default. The options are not exclusive, so both can be specified to be on at the same time to produce both outputs.

7.11.3.5.Storefile Options

Syntax:
.store [accuracy=n]

Description:

| .store [accuracy=n] | Specifies that the field is to be stored as a PWL waveform |
| accuracy=n         | Specifies the accuracy to which the field is stored. Sample points in the stored waveform will be to within n of their original value. This is used to reduce the number of points |
This is off by default and the accuracy is set to max (0.0) by default.

7.11.3.6. Transform Options

Syntax:

.transform [sub_avg]

Description:

<table>
<thead>
<tr>
<th>transform</th>
<th>Specifies that a transformation should be applied to a field</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub_avg</td>
<td>Specifies that the average value of the waveform for the faulty and fault-free circuits should be subtracted from the results. This effectively removes the DC component of every waveform.</td>
</tr>
</tbody>
</table>

This is off by default.

7.11.3.7. Measure Options

Syntax:

.measure [upper=n] [lower=n]

Description:

<table>
<thead>
<tr>
<th>measure</th>
<th>Specifies that the field is a .measure output variable and that .measure analysis is to be performed. This uses $3\sigma$ analysis on the distributions of the faulty and fault-free circuits.</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper=n</td>
<td>Specifies the upper value of a specification parameter which is used in the analysis in place of the good circuit upper $3\sigma$ point.</td>
</tr>
<tr>
<td>lower=n</td>
<td>Specifies the lower value of a specification parameter which is used in the analysis in place of the good circuit lower $3\sigma$ point.</td>
</tr>
</tbody>
</table>

NOTES:

1) Measure analysis can only be used with the alldata detection mode.
2) The ANACOV output for the .measure statement is of a different format to the normal output.
3) The overall statistical analysis should be ignored for this mode.

Measure mode is off by default.

7.11.4. Example Coverage Definition (coverage.def) file
An example coverage specification file is shown below:

```plaintext
* Coverage specification test input file
* transient analysis on fields 2 and vout
  .analysis tran
  .printopt userdata allfaults allstats coverage undetect separate
  .statistics sd_per_fault
  .field 2
  .detection fixed gut=1 glt=1 fut=1 flt=1 npoints=1
  .strobe all
  .saveraw strobeonly
  .histogram hist_file point
  .endfd

  .field vout
  .detection threshold upper=1.0 lower=0.5 fut=0.01 flt=0.02
  .strobe all
  .endfd
  .enda

* dc analysis on fields 2 and 5
  .analysis dc
  .printopt userdata allfaults allstats coverage undetect classes
  .statistics sd_per_fault
  .field 2 5
  .detection variable ugfile="upper.out" lgfile="lower.out"
  .strobe 0 1 0.1 1 2 0.2 write="strobe.out"
  .endfd
  .enda
```

7.12. Program Execution

ANACOV is invoked using the command:

```
anacov cctname
```

where cctname is the name of the HSPICE input file used to generate the faulty files with ANAFINS. The file extension of cctname must be omitted.

7.13. List of ANACOV Qualifiers

`-[no]acv[=filename]`
Specifies the name of the results file, containing the fault coverage and additional information. The default is cctname.acv. If -noacv is used the output is produced on the standard output. The default is -acv.

`-[no]coverage`
Specifies whether or not to print results of fault coverage. It enables ANACOV to be run to check the spice outputs are complete without writing results to the output file. The default value is -coverage.

`-overwrite|append`
Specifies whether to overwrite the existing ANACOV output results file or append the results to the end of it. The default value is -append.

-[no]convert
Specifies that ANACOV is to convert the faultlist.txt file produced from the OLDANAFINS program to a database file. This is for backwards compatibility with an old version of the ANAFINS program and need for use with version 2. The default is -noconvert.

-spice_output_extn="extn"
Changes the default SPICE output file extension to extn. The default is "out".

-spice_input_extn="extn"
Changes the default SPICE input file extension to extn. The default is "Hspc".

-raw_extn="extn"
Changes the default raw file extension to extn. The default is "raw".

-db_filename="filename"
Changes the default database filename to filename. The default is "cctname.idb".

-[no]use_env
Use environment information set up by ANAFINS. If this qualifier is set the environment information from ANAFINS ".anafins.env" file is used to determine the run directory for the circuit. With this qualifier set, the ANACOV program should be executed from the working directory rather than the run directory. Without this qualifier set the program should be executed from the run directory. The default is use_env

-[no]run_number[="n"]
Specifies the run directory in which to run ANACOV when using the ANTICS environment directory structure. This should only be used with the "use_env" qualifier. "n" is the run number containing the files to perform analysis on. If the "use_env" qualifier is used without the run_number=n qualifier then the run directory is set to the value in the .anafins.env file. The default is norun_number.

-[no]use_raw
Specifies that ANACOV should use raw files in place of SPICE output files. The raw files are created with the save_raw option. The default is -nouse_raw.

-an_def_filename="filename"
Specifies the name of the coverage definition filename. The default is “coverage.def”.

-histogram_extn="extn"
Specifies the file extension for the GSI histogram files. The default is “.hist”.

-[no]check_indices
Check that the spice responses using alldata analysis have the correct number of responses in the faulty files compared with the fault free file. The default is check_indices.

-old_version
Use the old_version algorithm when calculating the number of points to expect from a spice file. This is for backwards compatibility. The old version of SPICE (pre HSPICE95) produces one less point on transient analysis if the increment and stop times do not divide exactly. The default is -new_version.
NOTE: there is still some problem in calculating the exact number of points, so the number of simulation points expected by ANACOV can be overridden using the .points command in the coverage definition file.

-new_version
Use the new_version algorithm when calculating the number of points to expect from a SPICE file. This is the default is -new_version.
NOTE: there is still some problem in calculating the exact number of points, so the number of simulation points expected by ANACOV can be overridden using the .points command in the coverage definition file.
8. MC_RAND - Monte Carlo Random Number Generator

8.1. Overview

MC_RAND is a random number generator designed to be used for the generation of random numbers for use as data input to the HSPICE simulator. The file produced can be used as the data file for Monte Carlo simulations using the .data="FILE" statement in HSPICE. The HSPICE simulation response can be used with the ANACOV program running in the DATA, ALLDATA or THRESHDATA detection modes. Three distribution types can be generated - Normal, Uniform and Limit. The MC_RAND program allows a random seed value to be set so that unlike the inbuilt Monte Carlo analysis in HSPICE, different Monte Carlo simulation runs will produce different random data points.

8.2. Distributions Available

Three distributions are currently supported by MC_RAND. These are identical to those available using the HSPICE Monte Carlo simulation. Refer to the HSPICE user guide for more information.

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>MEAN, SIGMA (absolute variation), MULTIPLIER</td>
</tr>
<tr>
<td>Gaussian</td>
<td>MEAN, SIGMA (absolute variation), MULTIPLIER</td>
</tr>
<tr>
<td>Limit</td>
<td>MEAN, SIGMA (absolute variation)</td>
</tr>
</tbody>
</table>

Uniform Distribution
Gaussian (Normal) distribution

Limit Distribution

Uniform  Has a flat Probability Density Function (PDF) with centre MEAN and maximum variations +/- SIGMA. Note that SIGMA is absolute not relative to the MEAN.

Gaussian  Has a Gaussian (Normal) PDF with centre MEAN and standard deviation of SIGMA. Note that SIGMA is absolute not relative to the MEAN.

Limit  Has random limit distribution. SIGMA is either added or subtracted from MEAN depending on whether the outcome of a 0-1 uniform distribution is greater than or less than 0.5.

The multiplier is used for Gaussian and Uniform distribution to repeat the calculation a number of (MULTIPLIER) times with the furthest deviation saved. This produces a bimodal distribution. A MULTIPLIER value of 1 (the default) has no effect.

Note:
- For the Uniform distribution, the SIGMA keyword is used to set the max and min values for the distribution not the actual standard deviation.
- For the Limit distribution, the SIGMA keyword is used to set the values of the limits and is not the actual standard deviation.

8.3. Format of the Command File
The command file is used to describe the types of distributions, the number of rows and columns of data, and the means and absolute variations of the required distributions. Every new line within the command file generates a new column, which in turn relates to a new HSPICE parameter. The format for a line is as follows:

\[
\text{column uniform mean=}n \text{ sigma=}n \text{ multiplier=}N \text{ rows=}N
\]

or

\[
\text{column gaussian mean=}n \text{ sigma=}n \text{ multiplier=}N \text{ rows=}N
\]

or

\[
\text{column limit mean=}n \text{ sigma=}n \text{ rows=}N
\]

Where \(n\) represents a double and \(N\) an integer value. SIGMA is always specified as an absolute value and not relative to the size of MEAN.

For each line, the first 2 keywords must be specified and be in the correct place, otherwise a parser error will occur. Apart from that the order is arbitrary and the following default values are used if values are not specified:

\[
\begin{align*}
\text{Mean} &= 1.0 \\
\text{Sigma} &= 1.0 \\
\text{Multiplier} &= 1 \\
\text{Rows} &= 1
\end{align*}
\]

The number of rows must be the same for all columns and if there is more than one column and if the default value is not being used then the number of rows must be specified on each line. There should be no additional characters after the last command line e.g. newlines/spaces etc. Asterisks may be used to indicate a comment. This is useful for keeping track of which parameters correspond to which definition.

An example file is shown below which describes 30 rows and 3 columns.

\[
\begin{align*}
\text{column gaussian mean=}5.0 \text{ sigma=}1e-5 \text{ rows=}30 \quad \text{** oxide thickness}
\end{align*}
\]

\[
\begin{align*}
\text{column uniform mean=}10.3 \text{ sigma=}3.2 \text{ rows=}30 \quad \text{** beta}
\end{align*}
\]

\[
\begin{align*}
\text{column limit mean=}5.2u \text{ sigma=}1u \text{ rows=}30 \quad \text{** unit length}
\end{align*}
\]

Note the use of comments.

\section*{8.4. Program Operation}

The MC_RAND program reads a command file (the default is mc_rand.cmd) and then the seed file (default is ~/.mc_rand_seed). The seed file is updated by an increment by 1 every time the program is executed. This is to ensure the random numbers generated are truly random. Without the seedfile, MC_RAND would generate the same numbers each time. The program then generates an output file with columns of randomly generated numbers with distributions defined in the command file. The file should be
included within a HSPICE file using the HSPICE data driven analysis. An example input file is shown below:

```
Spice input file with data-driven Monte Carlo Analysis
* monte carlo (data driven) simulation parameters

.DATA mc_data MER
FILE='mc_data.inc'
+ox=1
+latdiff=2
+uop=3
.ENDDATA

* put circuit description here...

* define models with parameters to be altered by mc_data
* Note these are not complete definitions
* they are just examples

* PTYPE TRANSISTOR
 .MODEL MOSP PMOS
 +LEVEL = 2
 +VTO= vtp
 +TOX= ox
 +UO= uop
 +UCRIT= 4.2K

* NTYPE TRANSISTOR
 .MODEL MOSN NMOS
 +LEVEL = 2
 +VTO= vtn
 +TOX= ox
 +UO= uon
 +UCRIT= 12K

* define analysis relating to simulation parameters using sweep
 .tran 15ns 6.5us sweep data=mc_data

* define outputs to be analysed with ANACOV
 .print v(125) I(VDUM)
 .end
```

8.5. Program Execution

The program is executed using the command:

```
mc_rand filename
```

where filename is the name of the output file holding the random data.

A set of qualifier commands and flags can be used to control the operation of the program. Qualifiers may be specified in three ways: in a user initialisation file, in an initialisation file in the working directory, or on the command line. They are assigned in the following order:

```
Default program value
```
command in ~/mc_rand.ini
command in [current_directory]/mc_rand.ini
command line qualifiers

For example, a command line qualifier will overwrite any qualifier set in the anafins.ini files. Current_directory is the directory in which anafins is run. The complete set of qualifiers is listed in Section 8.6. The format of the mc_rand.ini file is as follows:

Each line starts with a valid qualifier expression in exactly the same form as would be found on a command line, except that no qualifier character, ";-", is required. Anything after a qualifier expression on a line is ignored, allowing for comments after each qualifier. Lines beginning with "!" are ignored to allow for longer comments. Blank lines are also ignored.

8.6. Program Qualifiers

-overwrite|append
Specifies whether to overwrite the existing MC_RAND output results file or append the results to the end of it. The default value is -overwrite.

-cmdfile="filename"
Specifies the name of the file to be used as a command file. The default is mc_rand.cmd.

-[no]errorlog="filename"
Specifies name of log file to which information about the distributions may be recorded. The default is outputfilename.log.

-seedfile="filename"
Specifies the name of the file to be used as the seed file. The seed file should contain a positive integer. Subsequent runs update the seed file, increasing the value of the seed by 1 each time. The default file is ~/.mc_rand_seed. The home directory (~) is derived using the environment variable HOME. Unless it is essential that the exact values of a run be repeated, this file should not be edited or reset.