Automating Analog Test Design

by Lawrence G. Meares
Intusoft, San Pedro California USA
email info@intusoft.com, http://www.intusoft.com

Abstract

A method of performing analog/mixed signal test will be described based on the assumption that:

1. All failure modes do not have to be defined.
2. The majority of failures are described by catastrophic faults [1-3].

Working forward from these assumptions, a comprehensive method will be presented that results in identifying a test suite that can serve either as a product acceptance test or a field test for fault isolation and repair. The resulting test suites are characterized by their percent fault coverage and number of replaceable components required for repair. The role of tolerance faults, correlated process faults and traditional IC defects are folded into the test methodology.

Introduction

Historically, product acceptance testing has stopped at specification compliance testing. While quality and safety considerations suggest that more testing is required, product cost has driven the testing toward a minimal solution. The ubiquitous desktop computer has relentlessly driven down the cost of design and test. It is now possible to use widely available Electronic Design Automation (EDA) tools to provide the specification for additional tests at a reasonable cost.

This paper will illustrate, using existing EDA technology, how it is possible to apply long-established techniques, which were originally developed for the aerospace industry, to define faults and sequence measurements in order to produce an optimal test sequence. Most tests require the design engineer to establish pass/fail limits. Methods are discussed to set these limits and to create tests that are robust, that is, tests which aren't likely to give incorrect pass/fail indications. Then a method will be described to establish optimal test sequences that will either maximize the likelihood of fault isolation, or minimize the effort that is required in order to demonstrate fault-free performance.

Defining Faults

What is a failure? A good design can accept part tolerances that are far wider than the tolerance of the individual components. For example, a 2k pull-up resistor may work just as well as the specified 1k resistor. Clearly, we want to accept the out-of-tolerance part in order to take advantage of the increased yield which is provided by a robust design. Production engineers should be able to substitute parts based on cost and availability.

In the pull-up resistor example, an open resistor could actually pass a test which is based upon functional requirements, and fail in the next higher assembly when the noise environment increases.

It's reasonable to conclude that we want to detect and reject products which contain catastrophic component failures; but accept products which may contain parametric "failures" that do not affect functional performance. Actually, these parametric failures are part of the tolerance distribution of the parts which we are using. Monte Carlo analysis will show the robustness of the design when we compare the resulting performance predictions with the product specification.

Defining failure modes: Failures are well characterized by a finite number of catastrophic failure modes. For digital circuits, there are 2 failure modes; the outputs may be stuck at logic one, or stuck at logic zero. Film and composition resistors are characterized by open circuit failures. The US Navy characterizes the catastrophic failures for many common parts for its Consolidated Automated Support System [3] and are the default failure modes used by several EDA vendors.

You may define failure modes using common sense experience, historical data or physical analysis. Depending on your application, you might want to consider additional failures; for example, IC bridging or interconnect failures which result in shorts between devices. In a PWB design, you may wish to simulate high resistance "finger print shorts" which are caused by improper handling of sensitive components.

Abstracting subassembly failure modes to the next higher assembly is a common error. For example, consider the case of an IC op-amp. Failures detected and rejected by the op-amp foundry are primarily caused by silicon defects. Once eliminated, these defects won't reappear. At the next higher assembly, failures may be caused by electrical stress due to static electricity, operator error in component testing, or environmental stress in manufacturing. Therefore, a new set of failure modes at the next production level is required. Again, we can usually describe these failure modes in terms of catastrophic events at the device or assembly interface, e.g. open, short or stuck failure modes for each interface connection.

Unusual failure modes are infrequent: Unusual failure modes get a lot of attention. For example, it is unusual to find a PNP transistor die in an NPN JANTXV package. Although these things do happen, they are very rare. If acceptance testing detects only 99% of all failed parts, then the quality of the product increases 100 fold after these tests are performed. For many products, the increased quality guarantees that products with undetected failures will not be delivered to the customer.

Detecting process faults: Process faults could cause the shift of many parameters simultaneously. When this is a consideration, as is usually the case for Ic's, the process parameters are monitored separately. If the process fails, the unit is rejected before the acceptance test is performed. Therefore, acceptance testing does not need to account for multiple parametric failure modes.

Setting Test Limits

Test limits are established using information from the product specification, simulation, analysis, laboratory testing and instrumentation capability. When parts fail or age, the measurement value for a given test may be near the test limit. Variations in circuit parameters, environment and test equipment can migrate these results across the test limit, thereby invalidating the test conclusion. Figure 1 shows why this happens when failed parts are included in the Unit Under Test, UUT.

By setting test limits suitably wide, the movement from pass to fail can be eliminated. In most cases, there isn't a product requirement which sets each test limit; test limits are determined via analysis, simulation or lab testing. Here are a few tricks we use:

Include a liberal input voltage tolerance for power sources
Perform a Monte Carlo analysis; include test set tolerances
Vary ambient temperature +- 20 degrees celsius

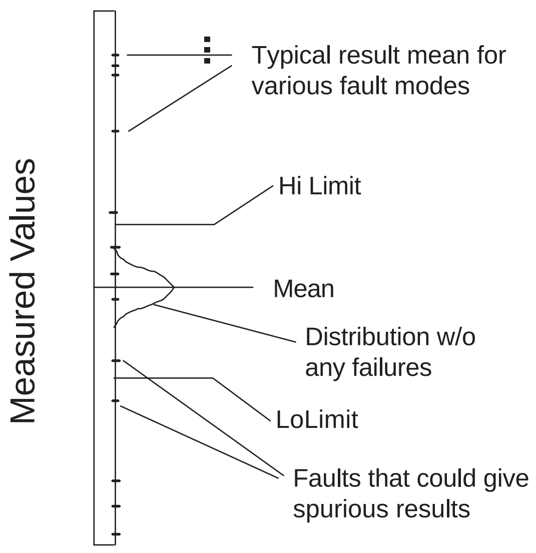

Measured Values
Typical result mean for various fault modes
Hi Limit
Mean
Distribution w/o any failures
LoLimit
Faults that could give spurious results

Figure 1, Failed parts can result in measurements that are near the measurement limits for normal operation, leading to false conclusions if the test is used for diagnosing those nearby faults

After subjecting the circuit to these simulation environments, expand the measurement tolerances so that the UUT passes all of the above tests with a liberal margin (5 sigma or higher). Remember: any lack of knowledge about component models, test equipment or environment will ALWAYS necessitate a larger test limit. Increasing the limit accounts for unforeseen conditions.

Movement of nearby failures across the limit, from fail to pass are handled by rejecting tests that could fail in this manner.

Seting Alarm Conditions

Another kind of simulation limit is the stress limit or alarm condition which is produced by some EDA tools [6]. When a part fails, it is frequently possible to overstress other parts, or cause an undesired circuit condition. It is recommended that power and/or current limits are set to the manufacturer's limit before part failure mode simulations are performed. In many instances, there are product safety issues such as firing an airbag or launching a rocket; these events must be prevented. Since these conditions are caused by special circuit states, you must devise measurements in order to catch these problems and group them with the stress alarms. The line between acceptance testing, built-in test and product quality begins to blur as we consider safety issues. For example, not only do we want to prevent unnecessary damage during an acceptance test, we also want to eliminate or detect safety hazards.

Resolving Alarms

Part failure modes which cause stress alarms should be detected as soon as possible. Tests can be sequenced in a manner which reveals these destructive failure modes early on, and therefore avoid the possibility of damage to any other circuit parts. Safe-to-start tests are among the tests which can reveal potentially damaging failure modes, e.g. shorts from power nodes to ground. After performing the tests which detect destructive failure modes, tests that could otherwise cause damage (if one of the destructive failure modes were present) can then be run. Using EDA tools for simulation is an ideal method for understanding circuit operation and specifying the new test limits.

Organizing a Test Sequence

Some test ordering has been described, based upon resolving stress alarms and simulation failures. The next level of ordering is by test difficulty or cost. Simple and inexpensive tests are performed first. Tests which use the same setup are grouped together. Tests which validate product performance are performed after eliminating the overstories failure modes. Product performance tests should not be used for fault detection because the tolerances are not set with failure modes in mind; they should only lead to a pass or out-of-tolerance conclusion.

Building a Test Fault-Tree

Tests are used to detect faults using the logic which is illustrated in Figure 2. Each test has one input and 2 outputs [2],[4]. The input contains a list of failure modes, and the test performs the logic which is necessary in order to classify the outcome as pass or fail. Each outcome has a list of failure modes that can be passed-on to successive tests. The process of selecting the best test in each ordered group results in a binary fault tree. After selecting a test, successive tests are placed on the pass and fail nodes of the tree until no more useful information is gained. When a new test configuration is selected, it may be possible to further isolate failure modes by expanding the tree from each node that has more than one fault. We will develop these concepts in considerable detail.

Since the world of simulation has more resources, that is, more observable information and more elaborate test capability, the job is generally acceptable at this point, although consuming excessive resources. We can gradually eliminate the most expensive test configurations, measurements and test points until the overall test objectives are met. These objectives are generally stated as a percent fault detection and if fault isolation is required, some goals regarding replaceable group size and distribution. Of course, if the objectives weren't met, you need to devise more tests, or if all else fails -- change the objectives.

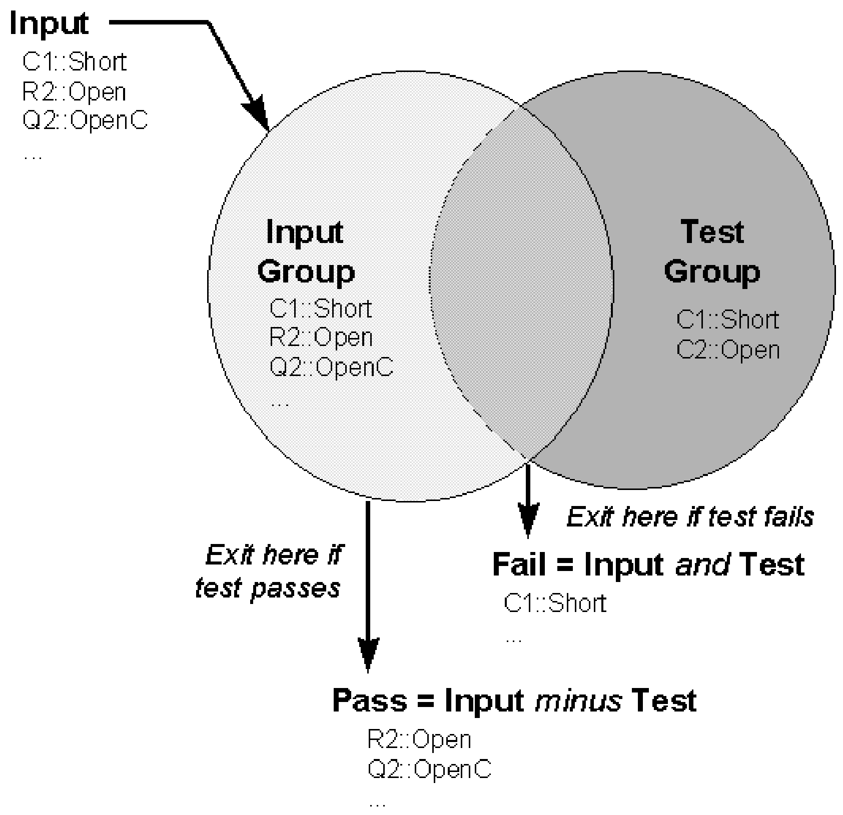


Figure 2: Test definition showing the logic used to sequence tests

Test Synthesis

Next, we lay out a method for test synthesis that is based on standards developed for the aerospace industry [1,4,5]. Before proceeding, several concepts are required to understand the synthesis procedure.

Ambiguity Group: An ambiguity group is simply a list of failure modes. Since the failure mode references a part, all of a parts properties are implicitly included. The most important for this discussion is the failure weight.

Failure Weight: The failure weight of a part failure mode is an arbitrary number proportional to the failure rate; it is set to 1.0 for each default failure mode. The failure weight will be used to grade a test. Using these weights, each ambiguity group can be assigned a probability which is the sum of all failure weights. A selection procedure will be developed that favors tests that split these cumulative weights equally between their pass and fail outcomes.

No Fault Weight : When we begin a test, there exists an input ambiguity group that contains all of the failure modes we will consider, it's the fault universe for the UUT. It is useful to add a dummy failure to this universe, the no fault probability. The no fault probability will change depending on where, in the product life cycle, the test is being performed. An initial production test should have a high yield so we use a high no fault weight. If the product is returned from the field, it's expected to be faulty so that the no fault weigh is low. For built-in tests, the no fault weight depends on the accumulated failure rate. It turns out that the no fault weight will change the grade for tests in the "go" line; that is, the product acceptance test list. The no fault weight will tend to reduce the number of tests needed to arrive at the expected conclusion.

Test Definition

What's a test? In order to compare one test with another there must be a precise definition of a test.

A test consists of a comparison of a resultant value against limits that classify the UUT as good (pass) or bad (fail).

The resultant value can be the combination of one or more measurements and calculations. Each test has an associated ambiguity group. At the beginning of the test, there isa set of failure modes that have not been tested; these are defined by the Input Ambiguity Group. The test itself is capable of detecting a number of failure modes, these modes are grouped into a set called the Test Ambiguity Group. The pass and fail outcomes, then carry a set of failure modes that are the result of logical operations carried out between the Input Ambiguity Group and the Test Ambiguity Group such that:

Fail Ambiguity Group = Input Ambiguity Group AND Test Ambiguity Group
Pass Ambiguity Group = Input Ambiguity Group MINUS Fail Ambiguity Group

Where AND represent the intersection of the lists, and MINUS removes the elements of one group from the other. Using MINUS here is a convenient way of avoiding the definition of NOT (Input…), since we really aren't interested in a universe that's greater than the union of the Input and Test groups. Figure 2 illustrates this logic.

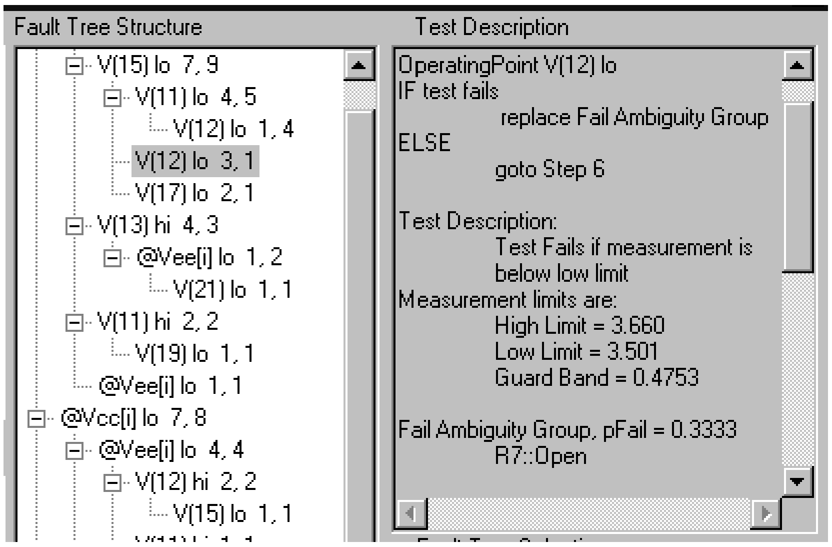
The fail or pass outcomes can then be the input for further tests. If further tests are only connected to the pass output, then a product acceptance test is created. If tests are connected to both the pass and fail outcomes, then a fault tree is created which isolates faults for failed products. In either case, the object of each test is to reduce the size of output ambiguity groups. When no tests can be selected to further reduce these ambiguity group sizes, the test design will have been completed.

Test Strategy

The strategy used to select test and test sequences is as follows:

> To arrive at a conclusion; that is, the smallest ambiguity group using the least number of tests.

The least number of tests is actually the smallest mean number of tests to reach the terminal nodes of the diagnostic fault tree. The fault tree is made by interconnecting the tests, illustrated here, in a binary tree.



Fault Tree Structure

```
  ⊟ V(15) lo  7, 9
     ⊟ V(11) lo  4, 5
          V(12) lo  1, 4
       V(12) lo  3, 1
       V(17) lo  2, 1
  ⊟ V(13) hi  4, 3
     ⊟ @Vee[i] lo  1, 2
          V(21) lo  1, 1
  ⊟ V(11) hi  2, 2
       V(19) lo  1, 1
     @Vee[i] lo  1, 1
⊟ @Vcc[i] lo  7, 8
  ⊟ @Vee[i] lo  4, 4
     ⊟ V(12) hi  2, 2
          V(15) lo  1, 1
```

Test Description

```
OperatingPoint V(12) lo
IF test fails
          replace Fail Ambiguity Group
ELSE
          goto Step 6

Test Description:
          Test Fails if measurement is
          below low limit
Measurement limits are:
          High Limit = 3.660
          Low Limit = 3.501
          Guard Band = 0.4753

Fail Ambiguity Group, pFail = 0.3333
          R7::Open
```

The Best Test

Without performing an exhaustive search, the best test tends to be the test with the highest entropy. Variations on the exhaustive search technique, such as looking ahead one test, rarely produce better results. Exhaustive search has been shown to consume computational resources so rapidly that it is not a viable method.

Selecting the best test: A terminal conclusion is defined as the pass or fail conclusion for which no more tests can be found. Then if "no fault" is present, then it is the product acceptance test result, with all remaining faults being the ones that are undetectable. Otherwise the parts in the resulting Ambiguity Group would be replaced to repair the UUT.

In general, our goal is to reach the terminal pass-fail conclusions by performing the fewest numbers of tests to reach each conclusion. The general solution to this problem using an exhaustive search technique expands too rapidly to find a solution during the lifetime of our universe[4]. Several heuristic approaches are possible [5], one of which follows.

If we take the idea of failure modes one step further, we can give each failure mode a failure weight that is proportional to the failure rate. To avoid looking up failure rates, we can default these weights to 1.0, and some time later fill in a more precise number. For each test candidate, we can compute the probability of a pass outcome and a fail outcome. From a local point of view, the summation of the pass and fail probabilities must be unity; that is, the UUT either passes or fails a particular test. Borrowing from information theory, we can compute the test entropy using the following equations [4],[5]].

$$\text{Entropy} = -q * \log(q) - p * \log(p)$$

where p and q are the pass and fail probabilities

and:

$$p = \Sigma \text{ pass weights} / (\Sigma \text{ pass weights} + \Sigma \text{ fail weights})$$

$$q = \Sigma \text{ fail weights} / (\Sigma \text{ pass weights} + \Sigma \text{ fail weights})$$

The highest entropy test contains the most information. We select the best test as the test having the highest entropy. For the case when failure weights are defaulted to unity, this method will tend to divide the number of input failures into 2 equal groups. Since no fault can only be in the pass group, a high no fault weight will steer the tests through the pass leg fastest; making the best product acceptance test. The rationale for a high no fault probability is the expectation that most units will pass the production acceptance test, a condition of an efficient and profitable business. If, on the other hand, we want to test a product that is broken, we would give the no fault probability a lower value. Then the test tree would be different, having a tendency to isolate faults with fewer tests.

Robust Tests

Tolerances can cause measurement results to migrate across the test limit boundary. As a result, a fault could be classified as good or a good part could be classified as a failure. Tolerances include:

> UUT part tolerances, computer model accuracy, measurement tolerances, UUT noise, test set noise and fault prediction accuracy.

Previously; we showed that avoiding false test conclusions for tolerance failures requires setting the test limits as wide as possible. Now we will show how to set the limits as far away from expected failed results as possible. We will do this by a unique test selection procedure. But first, to compare one test with another, we need to define a measure of test robustness.

Guard Band *(the fudge factor)*

A test measurement for a good UUT has a range of values that define acceptable performance which we call a tolerance band.

> The measure of test robustness with respect to a failure mode is then the distance between the failed measurement result and the nearest test limit divided by the tolerance band. We call this value a guard band.

The test limit can be safely placed in the guard band as long as no other faults have results in this band. Normalizing all measurements using their tolerance band allows us to compare guard bands of different tests. We can then modify the entropy selection method to reject tests with small guard bands. Figure 4 shows how this works.

In this example we have 2 operating point tests. Failure modes are identified as NoFault,F!, F2, …F6. Assume test A is performed first, and test B is performed on the pass group of test A as shown in figure 5.. Test A divides the failures into a pass group containing (F4,F5,F6) and a fail group containing (F1,F2,F3) . Connecting test B to the test A pass outcome eliminates F1 from the test B failure input and the guard band for test B extends from the hi limit to F4. If test B were done first the guard band would be smaller, from the test B hi limit to F1.

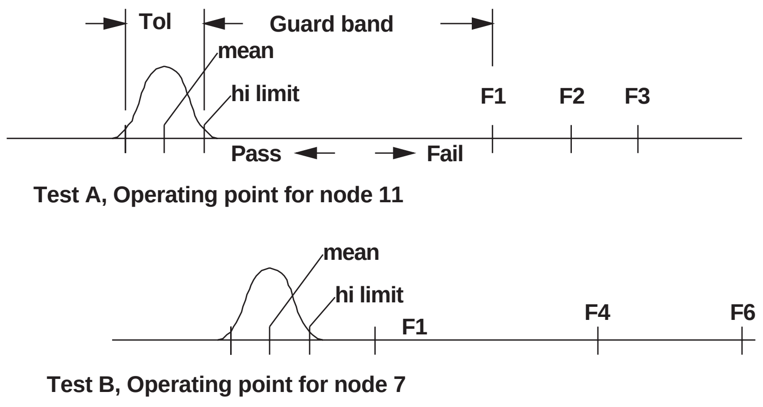

Tol
Guard band
mean
hi limit
F1
F2
F3
Pass
Fail
Test A, Operating point for node 11
mean
hi limit
F1
F4
F6
Test B, Operating point for node 7

Figure 4, Adjusting a test sequence achieves a robust test

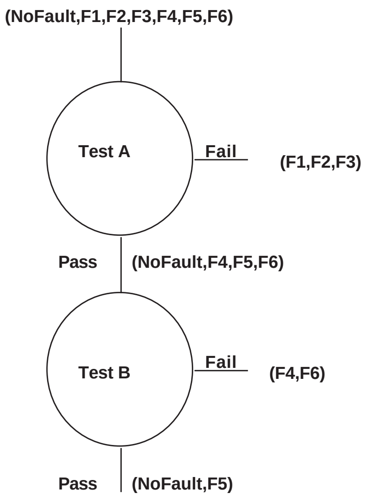


Figure 5, Fault tree

An incorrect test outcome will invalidate subsequent test decisions. In order to be right most often, tests with large guard bands should be performed first because they are less likely to be wrong. Moreover, tests which were previously rejected may turn out to be excellent tests later in the sequence (as illustrated in the example). Tests with small guard bands simply shouldn't be used.

While a model of the statistical distribution was shown in figure 4, you should be aware that there usually isn't sufficient information to have a complete knowledge of the statistics of a measurement result. In particular the mean is frequently offset because of model errors; for example, a circuit that is operating from different power supply voltages than was expected. The statistics of failed measurements are even less certain because the failure mode and the failed circuit states are less accurately predicted. It is therefore necessary to increase the tolerance band as much as possible. We avoided saying exactly where in the guard band the measurement limit should be placed because it's a judgment call, depending on how much the tolerance band was widened and on the quality of the fault predication.

Summary

We have shown how to use EDA tools with various test configurations to describe normal and failed circuit behavior. Methods for establishing test tolerances for circuits without failed parts were presented. Assuming that failures mechanisms are reasonably well known, a method to recognize the existence of a single fault and further diagnose which part was responsible was developed. All of these procedures can be closely integrated in existing EDA tools to provide a cost effective method for test design [6].

References

[1] Cheng-Ping Wand and Chin-Long Wey, "Efficient Testability Design Methodologies for Analog/Mixed Signal Integrated Circuits", 1997,3rd International Mixed Signal Testing Workshop, pp 68-74

[2] H. Dill et. al., "Application of Analog & Mixed Signal Simulation Techniques to the Synthesis and Sequencing of Diagnostic Tests", 1977 Autotestcon Proceedings, pp425-434

[3] CASS Red Team Package data item DI-ATTS-80285B, Fig. 1 -SRA/SRU Fault Accountability Matrix Table, pg. 11

[4] W.R. Simpson and J.W. Sheppard, "Fault Isolation in an Integrated Diagnostic Environment, IEEE D&T, vol. 10 No. 1, March 1993

[5] Sharon Goodal, "Analog/Mixed Signal Fault Diagnosis Algorithm and Review", 1994 AutoTestCon Proceedings, pp 351-359

[6] Test Designer, A software program from Intusoft, 1998.

Simulation measurements vs. Real world test equipment

There are fundamental differences between simulation measurement and test equipment measurements in accuracy, capability and measurement strategy. These differences will influence the way you map the simulation tests and measurements into real world test sets and measurement equipment. The following summary describes the most important differences:

Transient Simulation Accuracy:

Simulation accuracy is frequently no better than 1% to 3%; about what you expect from an oscilloscope. Accuracy limitations occur because of model accuracy, numerical errors and the user defined topology description. The implicit topology description is probably the largest source of error. Interconnect imperfections and parasitic coupling are assumed absent. You must explicitly include these effects. The designer usually adds these models sparingly because they are difficult to describe and may adversely impact the simulation. Fortunately, most parasitic elements have no important effects on a design; the trick is to identify and model the "important" ones.

Model Range:
Component models are made to be accurate in the neighborhood of a components expected useful operating point. Part failures will often place other components in an abnormal operating condition, one for which their model does not apply. To mitigate these problems, current limited power supplies should be used in the simulation. Moreover, when a simulation fails because the simulator cannot converge; it should be a warning that the failure mode cannot be studied with that particular test. It will be necessary to isolate the failure before running that particular test so that the failure mode does not need to be addressed anymore.

Visual Tests:
For items returned from the field, some component failures could have caused catastrophic damage that can be identified by visual inspection. Don't test a smoked board!

Multimeter Tests:
Consider running "dead" circuit tests to isolate failures that result in simulation convergence failures. These tests are run without the power supplies and can be used to find shorted or opened power components. Isolation of open or shorted bypass capacitors can be done in this manner using an AC multimeter.

Measurement Range:

You don't need to set the range or limits. There is no simulation range limitation. You automatically get IEEE double precision results.

Units:

Units are always converted to standard engineering postfix notation; for example 1.23e-9 volts will be shown as 1.23n volts. You may also use this notation for parameter input.

Coupling:

There is no need to worry about AC coupling when making measurements. You can always subtract the average value. In fact the stdDev function does just that. The simulator will work just fine at offsets of thousands of volts and your virtual instrument will not be overloaded.

Current vs. Voltage:

Current and voltage are properties of the quantity being measured. There is no need to select a current or voltage meter in the simulation.

Measurement Time:

Simulation time is a valuable resource. Measurement of period and frequency can be done on a much smaller data sample since there is no noise. Frequency measurement includes the specification of the number of events to be included so precise timing “gate” can be set in order to eliminate quantizing effects. Remember to let the circuit settle to its steady state condition before making measurements that assume the circuit has be running awhile.

AC Simulation Accuracy:

Frequency domain measurements made from AC analysis data are frequently more accurate than can be achieved with test equipment. The reason for this is that there is no noise to corrupt measurement accuracy and the numerical equations are solved exactly, rather than using the iteration procedure used in the transient simulation. For example; highly accurate estimates of oscillator frequency can be made through phase measurements. These would then map into counter/timer based measurements for the automated test.

Noise:

Thermal noise must be explicitly modeled, otherwise it is absent. This allows measurements to be made accurately without averaging them over many cycles. There is generally no benefit to adding noise unless the noise influences the value of the parameter you want to measure; for example, the lock time of a phase locked loop.

Numerical Artifacts:

Numerical artifacts are sometimes encountered in the Transient Simulation. You will have to pay attention to simulation settings such as RELTOL and integration METHOD in the simulation setup that have no counterpart in real world test equipment.

Fault Tree Design:

Simulators accumulate large chunks of data for each simulation run. These datum can be thought of as a state vector; representing the UUT state under some setup condition. These states then become a number of conventional tests; where a test is a measured value compared to some set of limits. The difficulty in acquiring data for the simulator, and probably for the hardware test are equivalent for each test. It is therefore convenient to lump all of these simulation measurements together when designing a fault tree; if you need to separate these measurement groups, you can name them differently and separate test vectors between these new measurement groups. Test sequencing is then performed by selecting the groups for a sequence, the test mode (binary, tertiary, …) and either manually or automatically completing the fault tree using the selected groups and mode.