Test Designer

Analog and Mixed-Signal Simulation, Fault Analysis and Testability

Provides a Repeatable and Structured Approach to Failure Analysis and Test Program Development

What is Test Designer?

Test Designer is a fully integrated toolset that adds test diagnostics to Intusoft's ICAP/4Professional:

1. Creates, simulates and verifies designs.
2. Defines product acceptance tests.
3. Builds sophisticated fault-diagnostic tree to identify failed parts.
4. Defines required built-in self tests.
5. Tracks and maintains production quality.

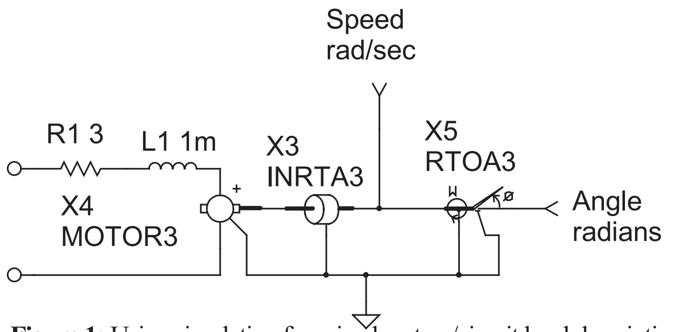


Figure 1: Using simulation for mixed system/circuit level description.

How is Test Designer used?

Test Designer's primary function is to develop a set of automated tests that can be used by unskilled technicians to test technologically complex end items. These end items include elevators, trains, aircraft, ships, pacemakers, operating room equipment, automobiles and perhaps slot machines, to name a few. The theory of fault diagnostic testing was initially developed by the commercial aircraft industry, and subsequently applied to military systems. Test Designer encapsulates that technology using Intusoft's IsSpice4 simulator to build a fault dictionary; that is, the measured response to each fault across a design. The simulator works from the transistor level through system level so that many design views are possible (Figure 1).

The design entry system (SpiceNet) used in Test Designer was developed especially for organizing and managing the immense amount of data needed to automate the test design process. A simple power supply, for example, might have only 30 electrical parts, containing on the average of 3 failure modes each. With 10 measurements, there are 1,800 test limits. In addition, the user needs to account for test equipment in the design, perhaps using several test set-ups. Traditionally, each failure mode and each setup would require a separate simulation netlist. Each of these netlists would have to be modified if the circuit was changed. However, here is what SpiceNet performs to solve these problems:

1. Creates test configurations made up of combined schematic layers. Special test configurations can be swapped out while using common layers to describe the unit under test (Figure 2).
2. Automates fault simulations so that the common underlying topology is reused with just the fault of interest employed for any given simulation.

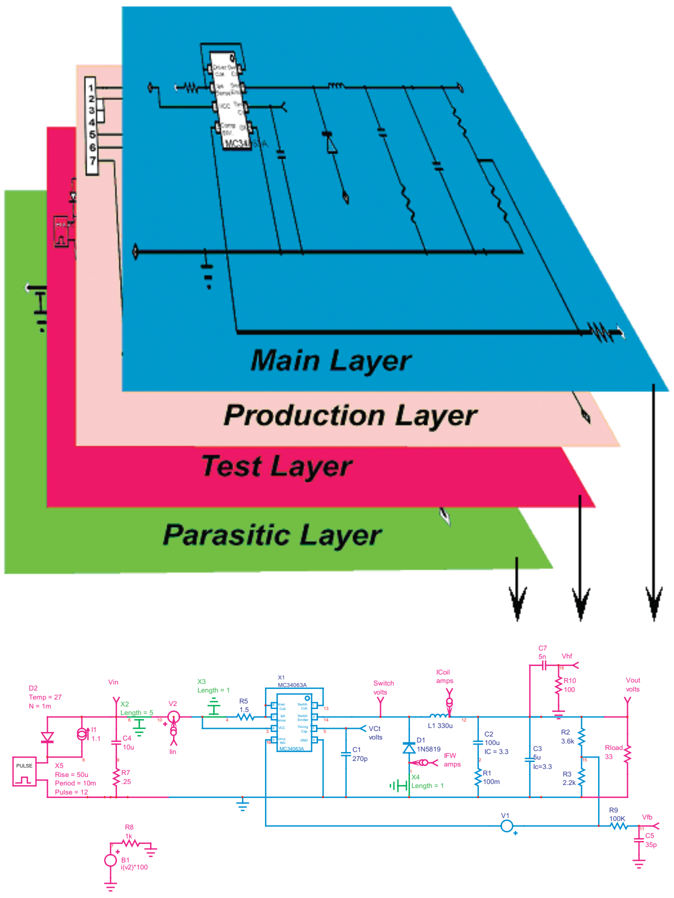


Figure 2: Schematic configured by layers lets test and production configurations share common parts.

3. Uses a built in graphical interface to define faults for each part including open, short, stuck and tolerance failures.
4. Provides default tolerances and failure modes for most parts.
5. Embeds user-defined electrical measurements and simulation results in the design database (Figure 4).

The state-of-the-art IsSpice4 simulator includes new convergence algorithms that virtually eliminate failed simulations when faults are injected. Even power train shorts in a power supply simulate successfully. Intusoft's powerful waveform viewer (IntuScope), shares a common script language with IsSpice4. Measurements made using IntuScope can be turned into scripts, saved in SpiceNet, and applied to the simulation directly without running the waveform post processor. This technological advance allows complex test sequences to be fully automated within the simulator.

What is Structural Testing?

If a circuit is built in accordance with its drawing specification, that is, all parts are working and connected properly, then the circuit should work as intended (Figure 3). That's the basic assumption used in structural testing. Of course one must also account for design "errors" that cause the circuit to perform outside of its design specification. In fact, there are cases for which the central region of a part's performance, or product distribution, is selected and sold for a premium price. Given that accuracy, there are many cases in which a failed part is not detected by performance-based testing. These cases can sometimes lead to unfortunate and unpredictable consequences in the field. For example, a fault-tolerant design for a pacemaker could have critical "backup" circuitry unavailable just when it's needed, or an aircraft control system could have a latent part failure that causes significant signal instability, resulting in a violent upset, consequent to wake turbulence or atmospheric disturbance.

My design is redundant, isn't that good enough?

The more fault tolerant a design, the greater the requirement is to periodically confirm that all of the parts are working. That's because the odds of a latent failure increase over time. With fault-tolerant designs, there are even more parts that can fail. One example is a "drive" in an automotive application that uses 100 parts in a critical steering system. If 10 million of these are deployed and "Space" quality parts are used, the number of failures in one year would be:

FR = 100.001%/1000hrs*8760hrs/year*

FR = 0.00876 per vehicle

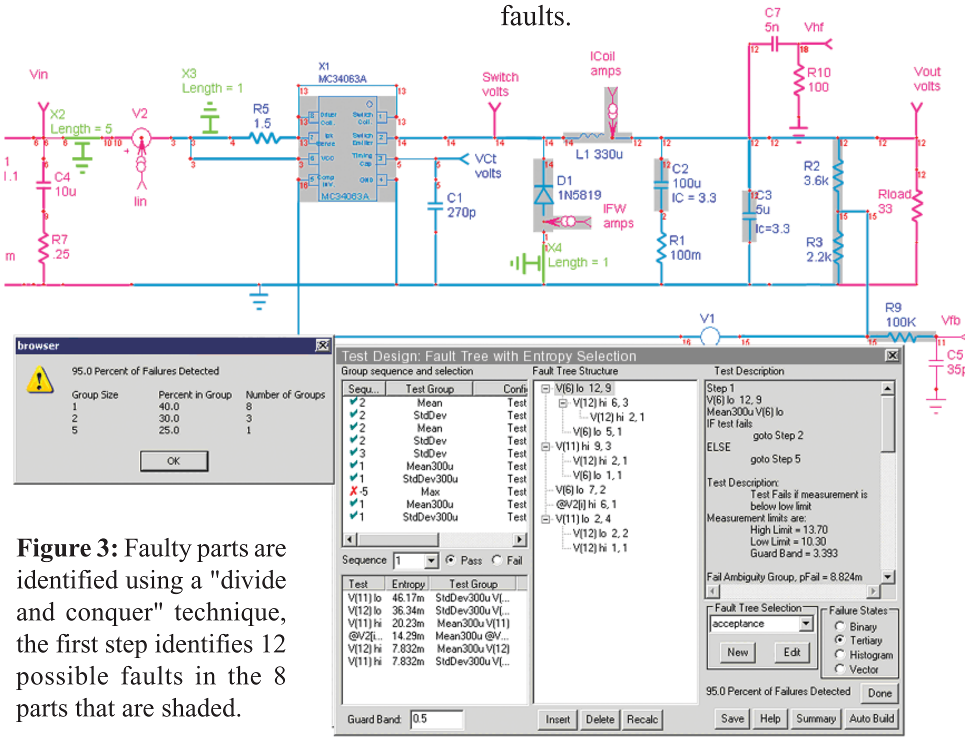
*Vehicle Failures/year = FR*10million =* ***87,600!***

You can see the huge deployment makes the risk unacceptable so that fault tolerant or redundant design is needed. But if 2 failures are needed to cause a mission failure and the system redundancy was never tested, then the following calculation holds:

*FR = 0.00876*0.00876 per vehicle*

*Failures/year = FR * 10Meg = 767*

That's still not very good. Even worse, in 2 years there will be 4 times as many mission critical failures. The solution is to test the system every time it's turned on, plus warn the driver that repairs are needed when latent defects are detected. That way you don't accumulate hours and FR drops by over a factor of 1,000. Plus, FR*FR drops by a million! And of course Test Designer gives you the assurance that the tests are detecting ALL structural faults.



browser

95.0 Percent of Failures Detected

Group Size	Percent in Group	Number of Groups
1	40.0	8
2	30.0	3
5	25.0	1

OK

Test Design: Fault Tree with Entropy Selection

Group sequence and selection

Segu...	Test Group	Confi
✓2	Mean	Test
✓2	StdDev	Test
✓2	Mean	Test
✓2	StdDev	Test
✓3	StdDev	Test
✓1	Mean300u	Test
✓1	StdDev300u	Test
✗-5	Max	Test
✓1	Mean300u	Test
✓1	StdDev300u	Test

Sequence 1 [✓ Pass] Fail

Test	Entropy	Test Group
V(11) lo	46.17m	StdDev300u V(...
V(12) lo	36.34m	StdDev300u V(...
V(11) hi	20.23m	Mean300u V(11)
@V2[i...	14.29m	Mean300u @V...
V(12) hi	7.832m	Mean300u V(12)
V(11) hi	7.832m	StdDev300u V(...

Guard Band: 0.5

Fault Tree Structure

```
V(6) lo 12, 9
    V(12) hi 6, 3
        V(12) hi 2, 1
    V(6) lo 5, 1
V(11) hi 9, 3
    V(12) hi 2, 1
    V(6) lo 1, 1
V(6) lo 7, 2
@V2[i] hi 6, 1
V(11) lo 2, 4
    V(12) lo 2, 2
    V(12) hi 1, 1
```

Test Description

```
Step 1
V(6) lo 12, 9
Mean300u V(6) lo
IF test fails
        goto Step 2
ELSE
        goto Step 5

Test Description:
        Test Fails if measurement is
        below low limit
Measurement limits are:
        High Limit = 13.70
        Low Limit = 10.30
        Guard Band = 3.393

Fail Ambiguity Group, pFail = 8.824m
```

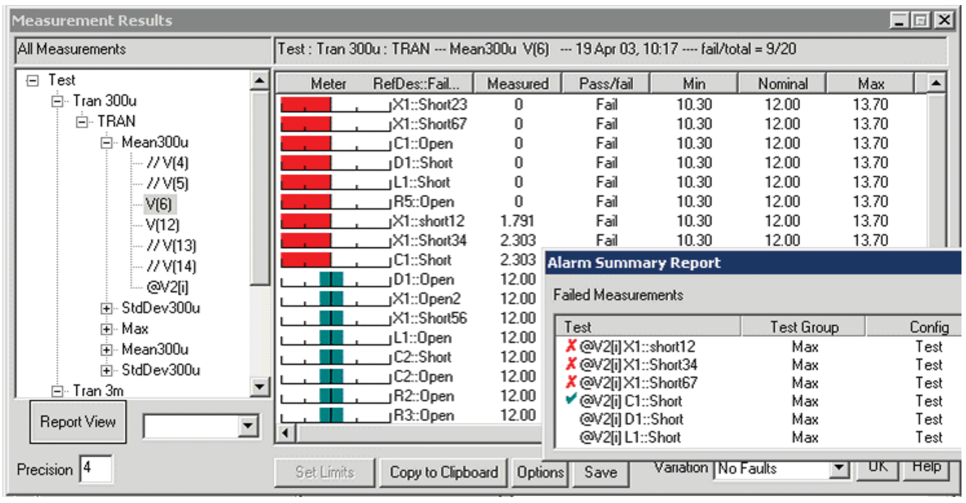
Fault Tree Selection: acceptance

Failure States: Binary [✓ Tertiary] Histogram Vector

95.0 Percent of Failures Detected

Insert Delete Recalc Save Help Summary Auto Build Done

Figure 3: Faulty parts are identified using a "divide and conquer" technique, the first step identifies 12 possible faults in the 8 parts that are shaded.



Measurement Results

All Measurements

Test : Tran 300u : TRAN --- Mean300u V(6) --- 19 Apr 03, 10:17 ---- fail/total = 9/20

```
Test
  Tran 300u
    TRAN
      Mean300u
        // V(4)
        // V(5)
        V(6)
        V(12)
        // V(13)
        // V(14)
        @V2[i]
      StdDev300u
      Max
      Mean300u
      StdDev300u
  Tran 3m
```

Meter	RefDes::Fail...	Measured	Pass/fail	Min	Nominal	Max
🟥	X1::Short23	0	Fail	10.30	12.00	13.70
🟥	X1::Short67	0	Fail	10.30	12.00	13.70
🟥	C1::Open	0	Fail	10.30	12.00	13.70
🟥	D1::Short	0	Fail	10.30	12.00	13.70
🟥	L1::Short	0	Fail	10.30	12.00	13.70
🟥	R5::Open	0	Fail	10.30	12.00	13.70
🟥	X1::short12	1.791	Fail	10.30	12.00	13.70
🟥	X1::Short34	2.303	Fail	10.30	12.00	13.70
🟥	C1::Short	2.303				
🟩	D1::Open	12.00				
🟩	X1::Open2	12.00				
🟩	X1::Short56	12.00				
🟩	L1::Open	12.00				
🟩	C2::Short	12.00				
🟩	C2::Open	12.00				
🟩	R2::Open	12.00				
🟩	R3::Open	12.00				

Alarm Summary Report

Failed Measurements

Test	Test Group	Config
✗ @V2[i] X1::short12	Max	Test
✗ @V2[i] X1::Short34	Max	Test
✗ @V2[i] X1::Short67	Max	Test
✓ @V2[i] C1::Short	Max	Test
@V2[i] D1::Short	Max	Test
@V2[i] L1::Short	Max	Test

Report View

Precision 4

Set Limits Copy to Clipboard Options Save

Variation No Faults

OK Help

Figure 4: Fault dictionary for the first test eliminates all of the over-stressed failure modes using a short 300usec, safe to start test.