

Agilent W1160C

Agilent Fault Detective

Getting Started Guide



Agilent Technologies

Notices

© Agilent Technologies, Inc. 1996-2010

No part of this manual may be reproduced in any form or by any means (including electronic storage and retrieval or translation into a foreign language) without prior agreement and written consent from Agilent Technologies, Inc. as governed by United States and international copyright laws.

Edition

Third Edition, January 2010

Printed in USA

Agilent Technologies, Inc.
3501 Stevens Creek Blvd.
Santa Clara, CA 95052 USA

Trademark Information

Adobe® is a trademark of Adobe Systems Incorporated.

Pentium® is a trademark of Intel Corporation in the U.S. and other countries.

Visual Studio® is a registered trademark of Microsoft Corporation in the United States and/or other countries.

Microsoft® is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

Windows and MS Windows are U.S. registered trademarks of Microsoft Corporation.

Windows Vista® is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

Software Revision

This guide is valid for Revisions 4.x of the Fault Detective software, where x refers to minor revisions of the software that do not affect the technical accuracy of this guide.

Warranty

The material contained in this document is provided “as is,” and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

Restricted Rights Legend

U.S. Government Restricted Rights. Software and technical data rights granted to the federal government include only those rights customarily provided to end user customers. Agilent provides this customary commercial license in Software and technical data pursuant to FAR 12.211 (Technical Data) and 12.212 (Computer Software) and, for the Department of Defense, DFARS 252.227-7015 (Technical Data - Commercial Items) and DFARS 227.7202-3 (Rights in Commercial Computer Software or Computer Software Documentation).

Safety Notices

CAUTION

A **CAUTION** notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a **CAUTION** notice until the indicated conditions are fully understood and met.

WARNING

A **WARNING** notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a **WARNING** notice until the indicated conditions are fully understood and met.

Contents

Introduction 7

- System Requirements 8
 - For Fault Detective Model Development 8
 - For the Fault Detective Runtime 9
- Installing Fault Detective 10
 - Modifying or Repairing Fault Detective 10
 - Contacting Agilent 11
- What is Agilent Fault Detective? 12
- Modeling Overview 13
 - Components 14
 - Subcomponents 15
 - Functions 17
 - Tests 17
- Modeling Guidelines 19

The Fault Detective User Interface 21

- The Screen Layout 22
- The Opening View 23
 - The Task Guide 24
- The Components and Functions View 25
 - Adding Model Elements 26
 - Components and Functions Summary 26
 - Exploring Components, Subcomponents and Functions 27
 - Detail Panes 27
 - Viewing Coverage 29

Creating a Model Report	29
The Tests View	30
Adding Model Elements	31
Tests Summary	31
Exploring Tests	32
Creating a Model Report	32
The Predicted Performance View	33
Creating a Predicted Performance Report	34
The Suspect Faults View	35
Diagnosing a Syndrome	36
Creating a Suspect Faults Report	38
Digital Modeling Tutorial	39
About This Tutorial...	40
Step 1. Create, Save, Close, and Re-Open the Model	42
Step 2. Add a Component	43
Step 3. Add More Components	46
Step 4. Add Subcomponents	49
Step 5. Add Bus Subcomponents	52
Step 6. Add Functions	55
Step 7. Begin Modeling Tests	59
Step 8. Add Coverage	63
Step 9. Verify the Model	65
Check for Unused Functions	65
Resolving Unused Functions	66
Check for Incomplete Tests	66
Resolving Incomplete Tests	67
Step 10. Use Predicted Performance to Improve the Model	69

Step 11. Add More Tests and Coverage 73

Step 12. Iteratively Improve the Model 76

Step 13. Create Reports 78

Where To Go Next 79

Diagnosing Suspect Faults 81

About Diagnosing Suspect Faults... 82

How Fault Detective Diagnoses Suspect Faults 83

Diagnosis Example 84

Running the Diagnosis 85

Scoring Algorithms 88

What's in the Online Help? 89

Predicted Performance 91

About Predicted Performance... 92

About Test Strategies... 92

Running the Predicted Performance Analysis 94

Using Analysis to Improve Test Coverage 97

Using Analysis to Optimize the Test Suite 101

What's in the Online Help? 106

Modeling with Microsoft Excel 107

Using Microsoft Excel with Fault Detective 108

Creating a New Model Using the Excel Template 109

Exporting a Model to Excel 111

Importing a Model from Excel 114

Entering and Editing Model Information in Excel 116

Contents

Export Options	120
One Test Group per Excel Worksheet	120
Multiple Test Groups per Excel Worksheet	122
What's in the Online Help?	123

Analog Modeling 125

About the Analog Model...	126
Analog vs. Digital Modeling	128
Selecting Components	129
Use of Subcomponents	130
Identifying Functions	131
Typical Amplifier Functions	131
Typical Filter Functions	132
Typical Switch Functions	132
Typical Attenuator Functions	132
Typical Mixer Functions	132
Global Functions	134
Tests	136
Differentiating Faults	137
Testing Components to Different Specifications	138

Glossary 141

Index 159



1

Introduction

System Requirements	8
Installing Fault Detective	10
What is Agilent Fault Detective?	12
Modeling Overview	13
Components	14
Subcomponents	15
Functions	17
Tests	17
Modeling Guidelines	19

This chapter contains installation information and an introduction to Fault Detective and modeling in Fault Detective.



System Requirements

The system requirements for Agilent Fault Detective 4.7 are as follows:

For Fault Detective Model Development

- Processor: minimum 2.0 GHz Pentium 4 or equivalent, Core 2 Duo recommended
- Operating system: One of the following 32-bit Microsoft Windows versions:
 - Windows® 7 (32-bit systems only)
 - Windows Vista® Enterprise (32-bit systems only)
 - Windows XP Professional Service Pack 3 or later
 - Windows Server 2008
 - Windows Server 2003, Service Pack 1 or later
- Browser: Internet Explorer 6.0 or later is recommended
- Available memory: 1 GB or greater
- Available disk space:
 - 225 MB required for installation (includes 160 MB for Microsoft .NET Framework)
 - 175 MB required for operation (includes 110 MB for Microsoft .NET Framework)
- Video: XGA (1024x768) or better, 256 colors or more
- Optional: Microsoft Excel version 10 (commonly known as Excel 2002) or higher to export/import models to/from Excel
- You will need a copy of the free Adobe® Reader application to read the product manuals and other product literature in PDF format online. The Reader can be obtained at:
<http://www.adobe.com>

For the Fault Detective Runtime

- Processor: minimum 2.0 GHz Pentium 4 or equivalent
- Operating system: One of the following 32-bit Microsoft Windows versions:
 - Windows 7 (32-bit systems only)
 - Windows Vista Enterprise (32-bit systems only)
 - Windows XP Professional Service Pack 3 or later
 - Windows Server 2008
 - Windows Server 2003, Service Pack 1 or later
- Available memory: 1 GB or greater
- Available disk space:
 - 225 MB required for installation (includes 160 MB for Microsoft .NET Framework)
 - 175 MB required for operation (includes 110 MB for Microsoft .NET Framework)

Installing Fault Detective

- 1 Close all other applications on your PC, insert the Fault Detective 4.7 CD into your CD-ROM drive, and follow the instructions on your screen.
- 2 If the installation does not start automatically, select **Start > Run** (on the Windows Start menu) and type `<drive>:\setup.exe`, where drive is your CD-ROM drive.
- 3 Leave the **Run the Agilent Fault Detective License Activator** check box selected. Following the installation, follow the instructions to license your software. You will need the Order Number and Certificate Number from your Fault Detective License Entitlement Certificate.

NOTE

If you are using floating (concurrent) licenses, do not use the Agilent Fault Detective License Activator. See the online help for directions: Click **Help > Contents and Index** on the Fault Detective main menu; search for the help topic on **Using Floating Licenses**.

- 4 If you have difficulty with the installation, contact Agilent as shown below.

Modifying or Repairing Fault Detective

When you Repair or Modify Agilent Fault Detective, you may see a dialog with a message similar to the following:

The feature you are trying to use is on a CD-ROM or other removable disk that is not available.

Insert the 'Agilent Fault Detective 4.7' disk and click OK.

If you see this message, insert the Agilent Fault Detective CD into the CD-ROM drive of your PC. Windows will use the files on the CD to make repairs.

If you are using Windows Vista or Windows 7 with User Access Control (UAC) enabled, you may see this dialog box even if the CD is in the CD-ROM drive. You will be able to browse to the files on the CD, but you will get the dialog box above again and again when you click **OK**.

In order to complete the repair in this case, you must do one of the following two procedures:

- Perform the repair with User Access Control disabled:
 - 1 Close the Windows Installer dialog box.
 - 2 Disable User Access Control.
 - 3 Restart the PC.
 - 4 Perform the Repair or Modify operation.
 - 5 Enable User Access Control and follow any further instructions from Windows. (You may be asked to restart your PC.)
- Run the installer from your desktop:
 - 1 Do not close the Windows Installer dialog box shown above.
 - 2 Copy the *Installer* folder from the Fault Detective CD to your PC's desktop.
 - 3 Use the **Browse** button in the Windows Installer dialog box to browse to the location of the *Installer* folder on your desktop.
 - 4 Select the installer file, *Agilent Fault Detective 4.7.msi*. Click **Open**.

Contacting Agilent

In the United States, the primary contact center phone number is 1-800-829-4444.

To contact Agilent Technologies for technical and application assistance, consulting services, product selection and purchasing assistance, training and education, visit:

<http://www.agilent.com/find/assist>

What is Agilent Fault Detective?

Agilent Fault Detective is a model-based software tool that diagnoses functional test failures, so repair and rework costs are greatly reduced. Given a set of functional test results, Fault Detective attempts to identify the faulty component or components based on a model of the test suite and the device under test (DUT).

Fault Detective can also perform analysis on your model and test suite by simulating defective components (faults) in your model, and by simulating the various patterns of test results (syndromes) that may be exhibited by these defective components. The results of this simulation are used to predict the detection and isolation performance of your test suite against the modeled DUT.

Modeling Overview

Agilent Fault Detective uses a model to describe the interaction between a functional test suite and the device under test (DUT). When the functional test suite is applied to a DUT, each test returns a result of pass or fail. These functional test results, or a suitable subset of them, are then applied to the Fault Detective model for the DUT to produce the diagnosis.

The model is a means of capturing what you already know about how your tests verify your DUT—which components, buses, operations and characteristics are used by each test.

Within Fault Detective, you create a model which is a functional representation of your DUT and the tests performed on the DUT. This functional representation consists of components, subcomponents, functions and tests.

[Figure 1](#) summarizes the Fault Detective modeling process.

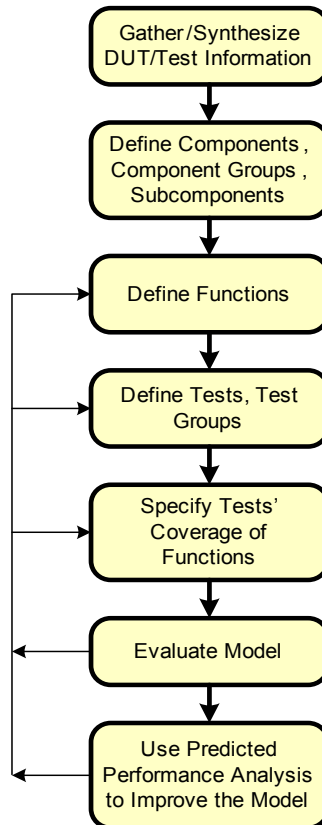


Figure 1 The modeling process

Components



Components represent the functionality provided by the DUT's physical parts. Often, several physical parts will be treated as a single component because they work together to provide a specific function and cannot be separated. In some cases, a physical part and a component will map to each other directly.

A component is an abstraction of the physical parts of the DUT. A component represents a structural element of a DUT performing one or more related functions. A component may be:

- Many physical parts—This is commonly used when a DUT has many discrete parts (for example, a power supply section), or when a test uses a function to exercise several indistinguishable physical parts.
- A single physical part—This is commonly used when the DUT has large complex parts (like ASICs).
- A block of parts.
- An entire assembly of parts—This is used when the component represents a complete printed circuit board.

Subcomponents

- Subcomponents are logical subsets of a component. For example, a microprocessor chip could contain a CPU, memory, bus interface, and serial I/O elements. If these elements are tested in different functional tests (as they typically are), then model a component for the microprocessor chip and a subcomponent for each of these distinct elements. This finer level of granularity lets you define test usage for each subcomponent, giving you a higher degree of accuracy in your Fault Detective results.

Using subcomponents is highly encouraged and typically results in more accurate models. Since adding subcomponents adds to the complexity of the model, you can apply these more specific rules:

- A subcomponent represents a structural element of a component performing one or more related functions. Subcomponents are used to model DUT features such as:
 - A bus with associated read and write functions
 - A signal group with associated functions such as “output enable”, “output disable”

- A CPU with very separate functional blocks such as “floating point section” and “graphics section”
- For reasoning purposes, Fault Detective regards subcomponents as separate possible explanations for the failed tests. If one subcomponent works (for example, bus1), this does not imply that another subcomponent of the same component is working (for example, bus2).

You should use subcomponents if:

- The parent component represents a complex part (or perhaps a combination of several complex parts).
- The component can be broken into logical subcomponents.
- Small combinations of subcomponents are tested in different tests.

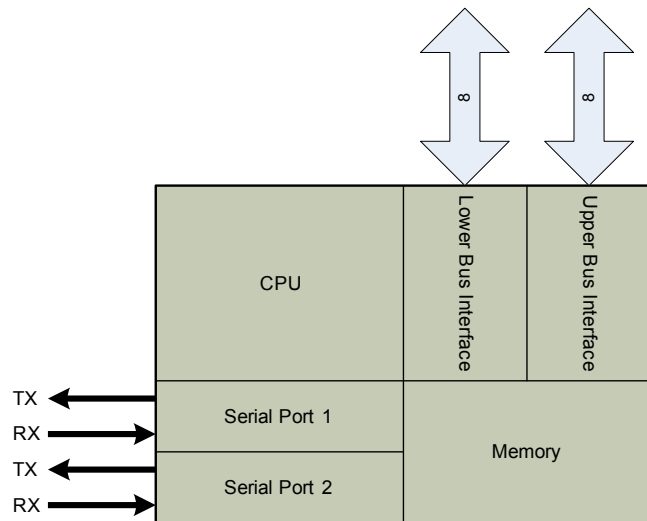


Figure 2 Typical subcomponents of a microprocessor component

Functions

- ◆ Functions are attributes, characteristics, or behaviors of components or subcomponents.

From a digital standpoint, functions are often operations performed by the component. For example, a bus transceiver will typically have read and write functions.

From an analog standpoint, functions are often characteristics of the component or a set of components. For example, a filter will typically have flatness and insertion loss characteristics that can be modeled as functions.

Tests



A test can be defined as one or more measurements or observations on a DUT that are evaluated to return information about the operational state of that DUT. A functional test suite is a set of functional tests that are run to determine whether a DUT is fully functional. Functional tests generally make one or more measurements on the DUT, and then pass judgment about some piece of functionality of the DUT. The result is pass or fail. If a test passes, it indicates a particular aspect of the DUT has been shown to be functional; if it fails, it indicates some portion of the functionality of one or more components isn't working.

In Fault Detective, you can model each functional test that can return a pass or fail judgment about some portion of the functionality of the DUT. In most cases, every test in a functional test suite should have a corresponding test element in the Fault Detective model. If there are tests in the test suite that do not evaluate the functionality of the DUT, you do not need to enter these tests into the Fault Detective model.

As shown below, tests use functions and functions exercise components/subcomponents.

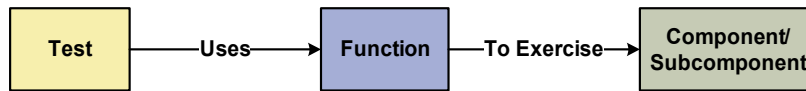
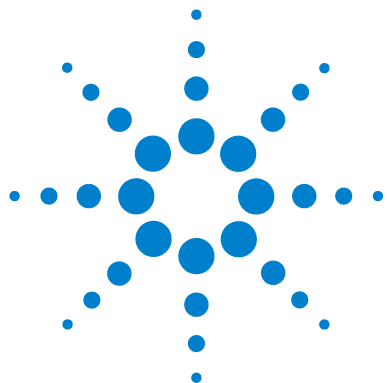


Figure 3 Tests use functions and functions exercise components/subcomponents

Modeling Guidelines

- Create the model incrementally. Start small, enter syndromes, run analysis, make diagnoses, gain confidence, and add more model elements later. Your results will improve as you add detail to the model.
- Carefully name your model elements, especially functions. You do not want to have to remember the content of every function or open them to understand what they do. The name should imply the content clearly.
- Mark up a block diagram and enter only the components needed for a model. Do not enter the DUT's net list, bill of materials, or schematics directly. Typically, you will not need all that information. Select a section of the DUT and an interesting subset of tests. Group parts into logical blocks to hide unnecessary detail. For example, you can model a complex RF device at the amplifier, filter, and mixer level.
- Model at the logical level, not physical level. Do not model individual data and control lines. Instead, bundle them together into logical groups—for example, buses.
- Enter your selected components first with obvious subcomponents like buses and major control signals. These are the things that you probably already know will be important in describing test paths.
- For each component/subcomponent, enter obvious functions. Buses will almost always have read and write functions. Passive elements typically have only one function. Complex amplifiers will likely have several functions.
- Always model the tests and DUT the way they are and leave the diagnoses to Fault Detective. Do not try to model to force some perceived correct answer.



2 The Fault Detective User Interface

The Screen Layout	22
The Opening View	23
The Components and Functions View	25
The Tests View	30
The Predicted Performance View	33
The Suspect Faults View	35

This chapter contains an overview of the Fault Detective user interface.



The Screen Layout

The screen layout, used throughout the application, consists of menus and toolbars, an explorer pane, a detail pane, navigation buttons, and a Task Guide.

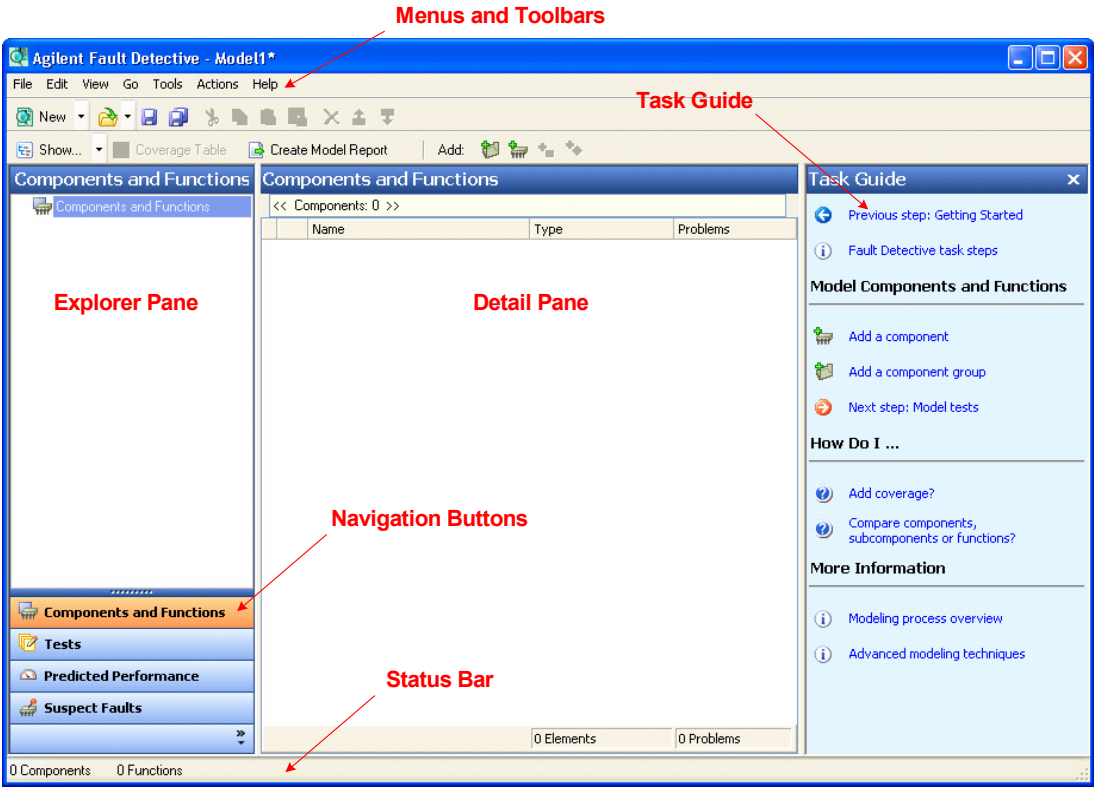


Figure 4 The Fault Detective user interface

The Task Guide



The Task Guide is located on the right side of the screen and is particularly useful for new Fault Detective users. The information in the Task Guide changes as you move through the various views and provides shortcuts to operations, features and information. For example, in the Opening view, the Task Guide contains this information:

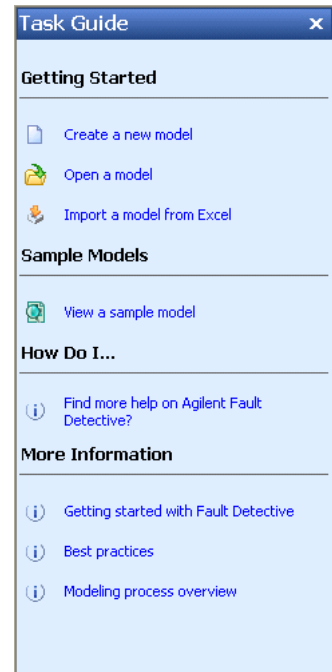
Getting Started Create a new model, Open a model, or Import a model from Excel.

Sample Models Choose from a model of an analog device or a digital device. These models are used in this getting started guide and show modeling fundamentals.

How Do I... Tasks associated with the current view appear here.

More Information Contains links to related information.

Next and Previous Steps As you move through the recommended model development steps, this icon  in the Task Guide points the way to the next logical task step. Click this icon  to backtrack to the previous step in the process.



The Components and Functions View



The Components and Functions view appears after you create a new model, open an existing model, or import model information from Excel. This view is where you add, delete, and edit component groups, components, subcomponents, and functions. You can access this view at any time by clicking the **Components and Functions** button.

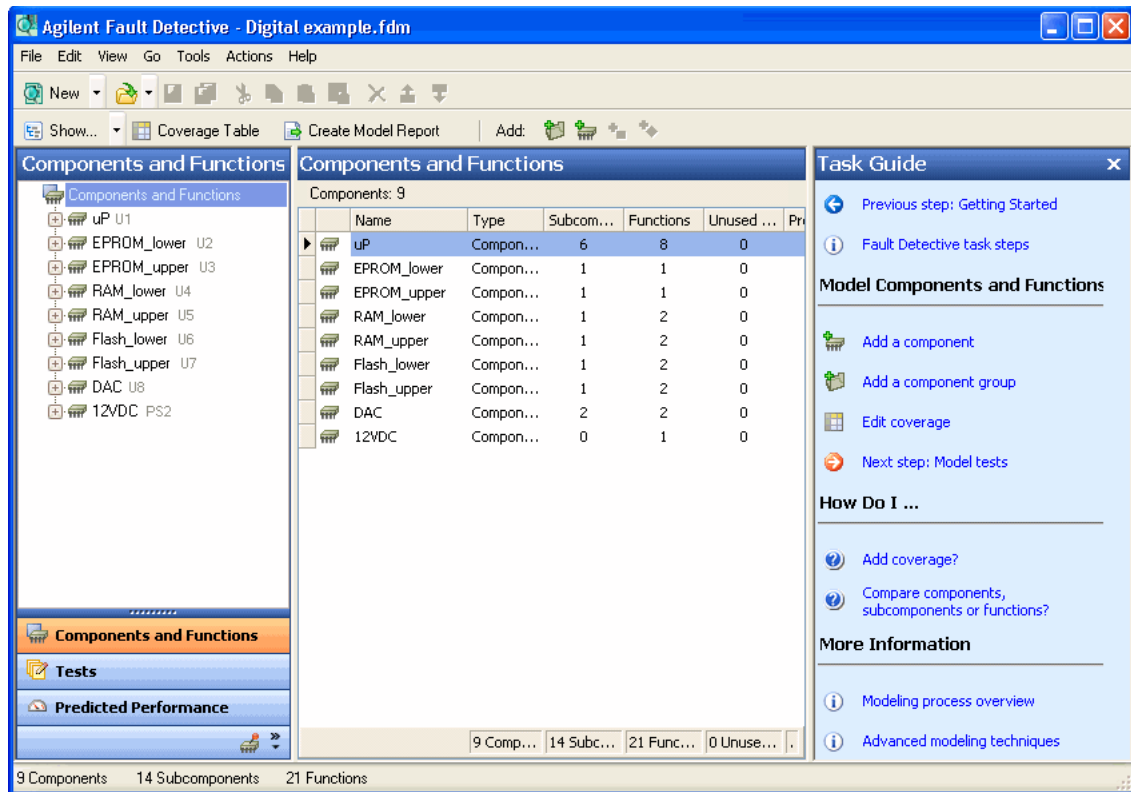


Figure 6 The *Digital example.fdm* model shown in the Components and Functions view

Adding Model Elements

You can use the **Add:** buttons in the toolbar or links in the Task Guide to add component groups, components, subcomponents and functions.

Components and Functions Summary

Selecting the top level in the explorer pane shows a summary of the components and functions in the model.

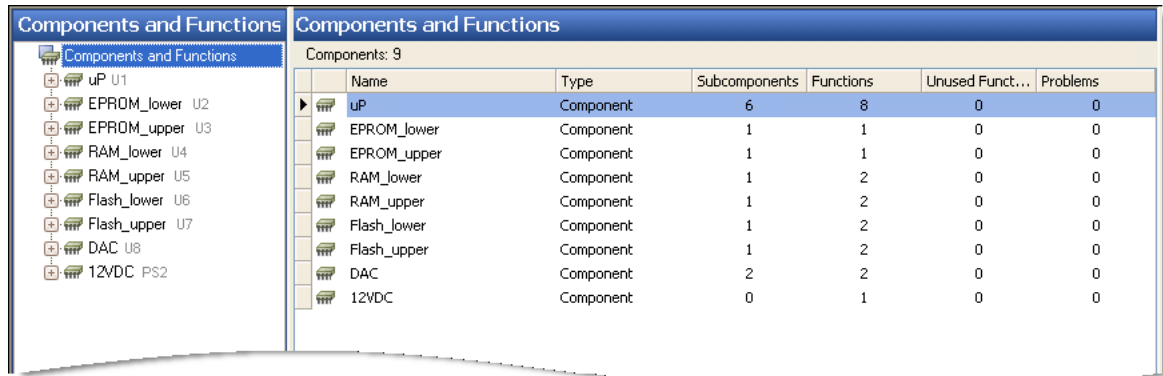


Figure 7 A summary of the components and functions in the *Digital example.fdm* model

Exploring Components, Subcomponents and Functions

Expanding the tree in the explorer pane shows the underlying structure of your model. This hierarchical view shows component groups, components, subcomponents and functions. Click on any of these model elements to see more information in the detail pane.

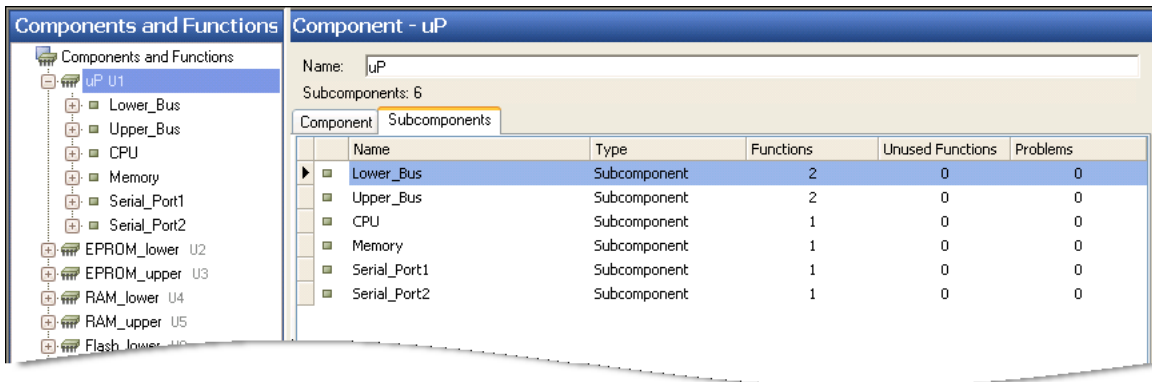


Figure 8 Click an item in the explorer pane to see more information in the detail pane

Detail Panes

Selecting a function in the explorer pane exposes detail regarding the function and how it exercises the component/subcomponent.

Notice the Notes: area near the bottom in [Figure 9](#). You can describe the function details here or add links to supporting documentation such as HTML files and URLs. This is an important Fault Detective feature allowing you to provide a transfer of knowledge from the model designer to the production floor. When necessary, test personnel can locate supporting documents such as schematics, block diagrams or parts cost information. The Notes: field is available for components, functions and tests.

2 The Fault Detective User Interface

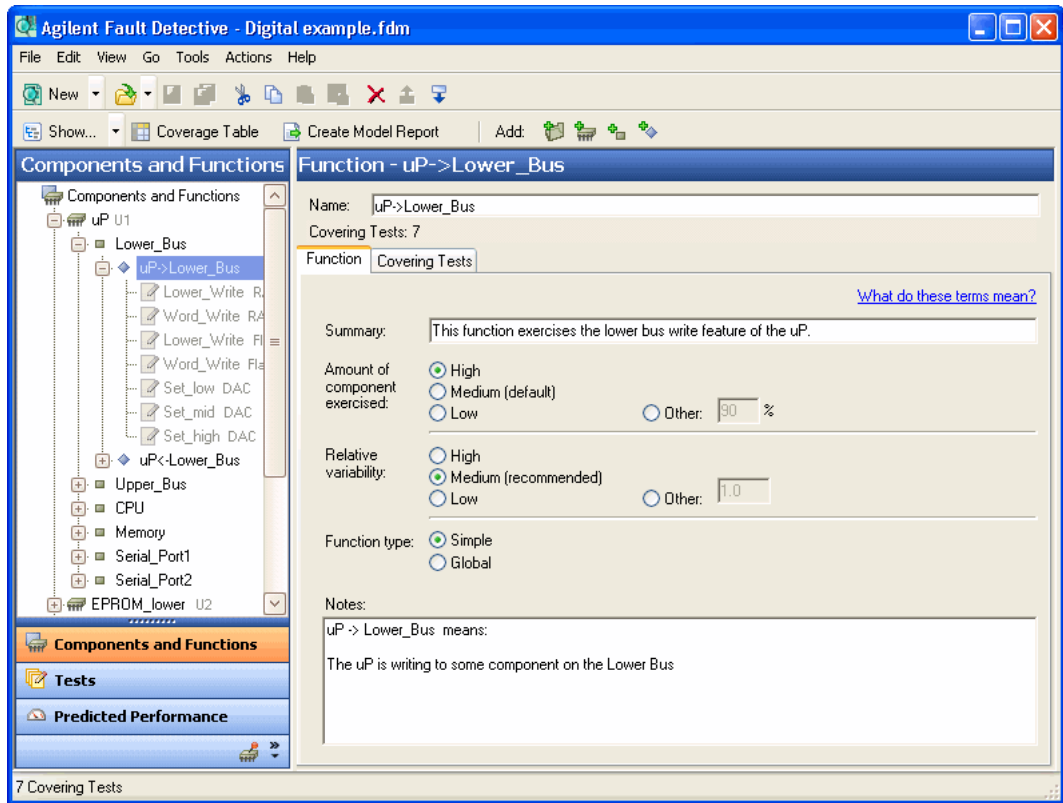


Figure 9 Detail pane showing the *uP->Lower_Bus* function

Viewing Coverage

Click the **Covering Tests** tab for any function and get a listing of the test(s) using this function (an x in the Used column indicates the test(s) using this function).

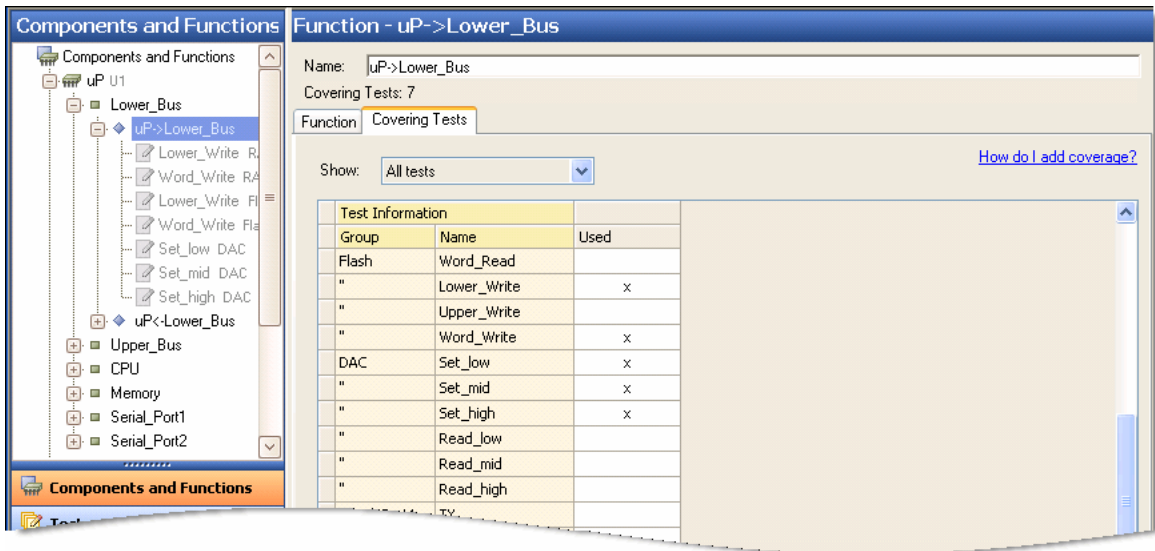
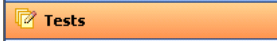


Figure 10 Tests covering this function

Creating a Model Report

Click the **Create Model Report** button on the toolbar to create an HTML report for the model. Refer to the Fault Detective online help for details regarding this report.

The Tests View



Click the **Tests** button on the bottom left side of the screen to open the Tests view. The Tests view is where you add, delete or modify test groups and tests. Test groups and tests are listed hierarchically in the left explorer pane. Details for the selected test group or test are shown in the detail pane. The Task Guide shows test-related actions and shortcuts to supporting information.

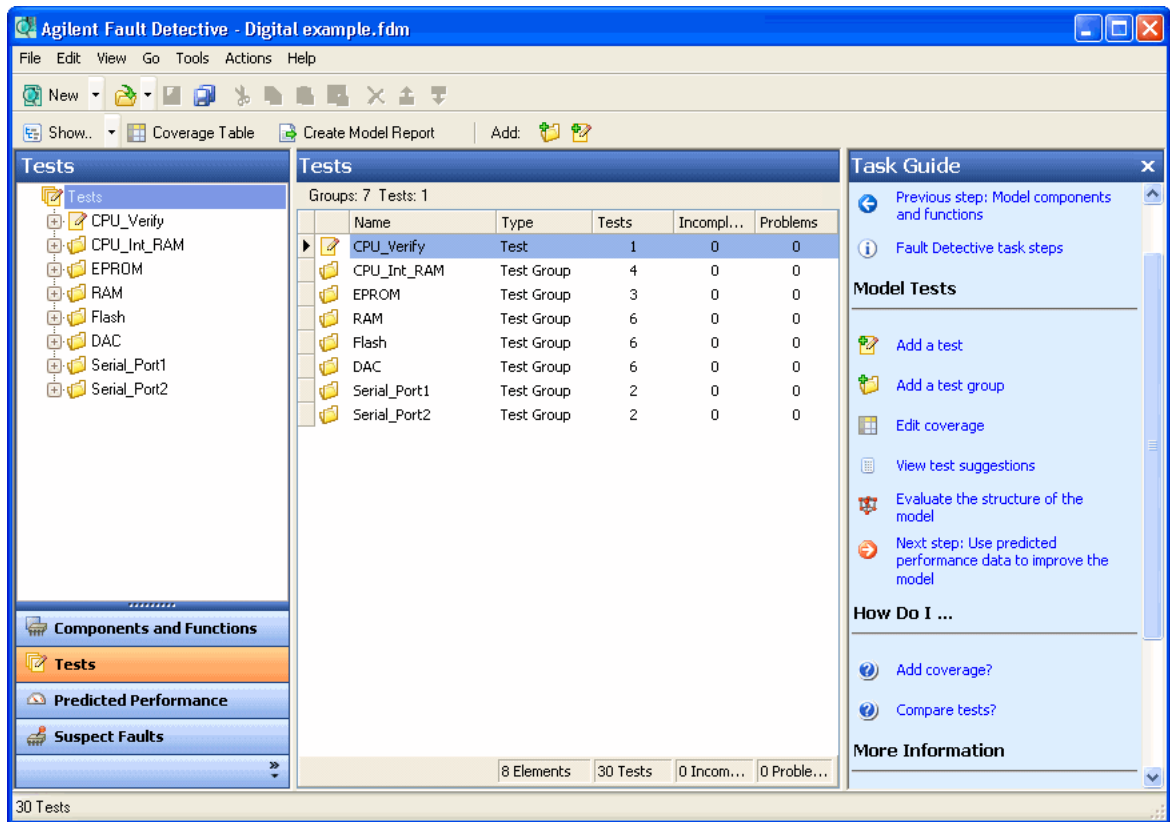


Figure 11 The *Digital example.fdm* model shown in the Tests view

Adding Model Elements

You can use the **Add:** buttons in the toolbar or links in the Task Guide to add test groups and tests.

Tests Summary

Selecting the top level in the explorer pane shows a summary of the test groups and tests in the model. This allows you to easily locate any incomplete tests or problems relating to tests.

Tests		Tests				
<div> <div>Tests</div> <div> <div>CPU_Verify</div> <div>CPU_Int_RAM</div> <div>EPROM</div> <div>RAM</div> <div>Flash</div> <div>DAC</div> <div>Serial_Port1</div> <div>Serial_Port2</div> </div> </div>		Groups: 7 Tests: 1				
		Name	Type	Tests	Incomplete Tests	Problems
		CPU_Verify	Test	1	0	0
		CPU_Int_RAM	Test Group	4	0	0
		EPROM	Test Group	3	0	0
		RAM	Test Group	6	0	0
		Flash	Test Group	6	0	0
		DAC	Test Group	6	0	0
		Serial_Port1	Test Group	2	0	0
		Serial_Port2	Test Group	2	0	0

Figure 12 Test groups and tests in the *Digital example.fdm* model

Exploring Tests

Clicking on a test in the explorer pane and clicking the **Covered Functions** tab shows the function(s) used by the test.

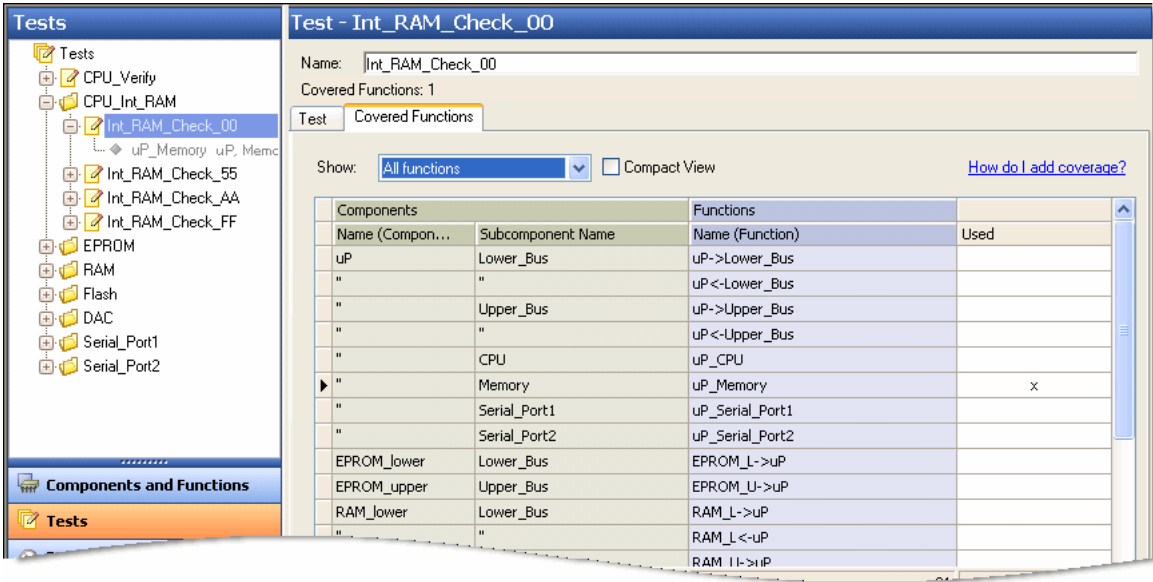


Figure 13 Test details for the *Int_RAM_Check_00* test

Creating a Model Report

As with the Components and Functions view, from the Tests view, you can click the **Create Model Report** button on the toolbar to create an HTML report for the model. Refer to the Fault Detective online help for details regarding this report.

The Predicted Performance View



Click the **Predicted Performance** button on the bottom left side of the screen to open the Predicted Performance view. This view allows you to create a test strategy, run the analysis, and create an analysis report.

The test strategy is listed in the left explorer pane. Details for the test strategy are shown in the detail pane. The Task Guide shows test analysis-related actions and shortcuts to supporting information.

The test strategy consists of a set of selected tests and a set of selected components. You select these elements from lists of all the tests and components in the model. By default, when you create or import a new model, there is one test strategy consisting of all the tests and components in the model.

You can analyze the test strategy by clicking the **Analyze** button in the toolbar or from a link in the Task Guide. The analysis creates predicted performance data consisting of:

- Predicted performance of the test strategy
- Predicted performance of individual faults
- Predicted performance of individual tests
- Suggestions for additional tests

When Fault Detective analyzes the test strategy, only the selected tests and components are considered in the analysis results. Therefore, the predicted performance displayed for the test strategy predicts what would result from running only the selected tests, on a device under test that consists of only the selected components. Tests and components that are not selected are not used in the analysis.

Creating a Predicted Performance Report

Click the **Create Predicted Performance Report** button on the toolbar to create an HTML report for the selected test strategy. Refer to the Fault Detective online help for details regarding this report.

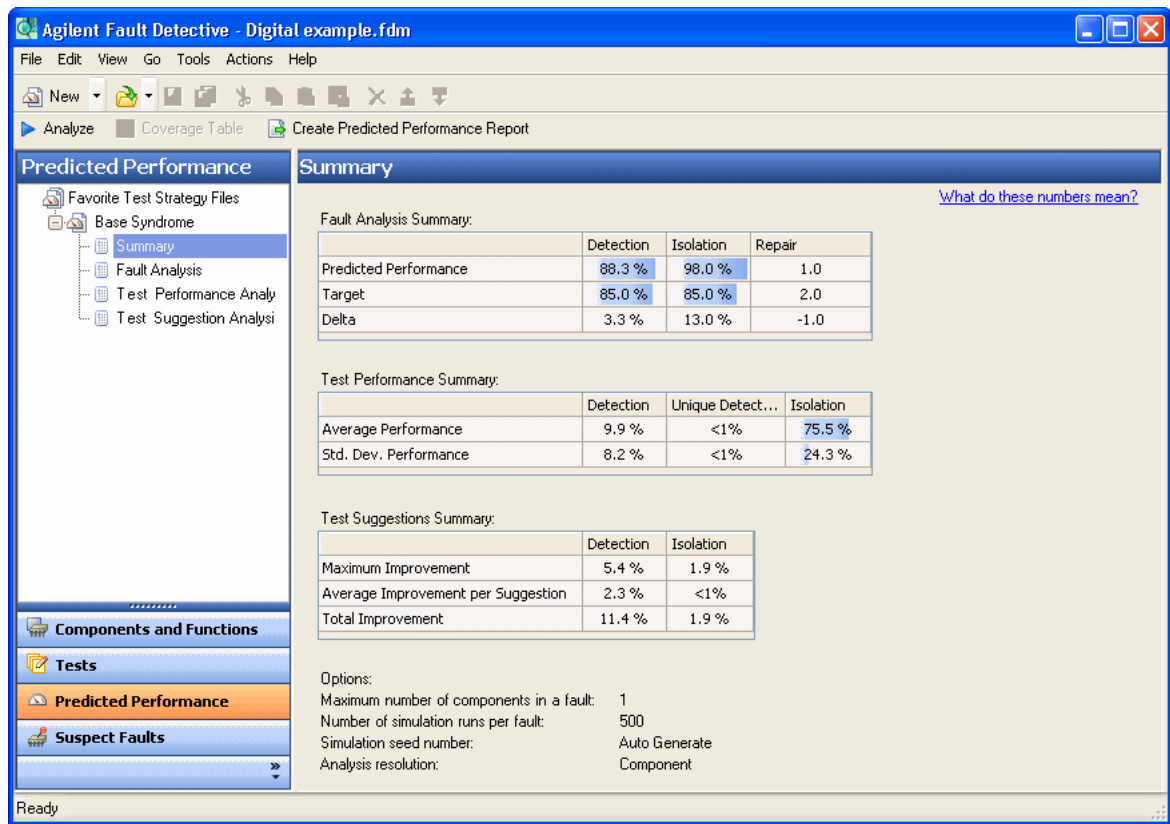


Figure 14 The Predicted Performance view

The Suspect Faults View



Click the **Suspect Faults** button on the bottom left side of the screen to open the Suspect Faults view. This view is where you create, edit, save and load syndromes (a set of test results). Syndromes are listed in the explorer pane. Details for the selected syndrome are shown in the detail pane. The Task Guide shows syndrome-related actions and shortcuts to supporting information.

A syndrome is a set of unordered test results, with each test's result recorded as *Pass*, *Fail*, or *Skip*. Fault Detective stores syndromes in files with the extension *.tr*. You can create, edit, save and load syndromes in the Suspect Faults view.

Clicking the **Diagnose** button produces a list of suspect faults based on the pass, fail and skip information in the selected syndrome.

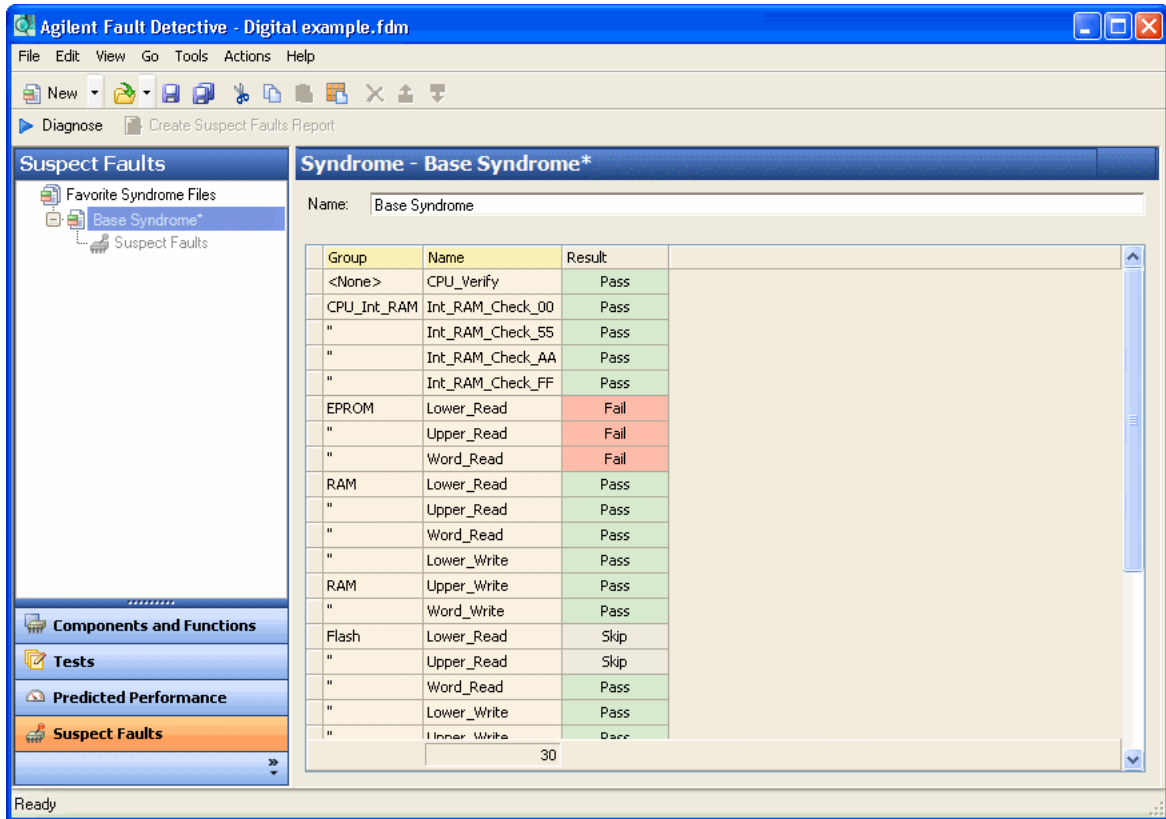


Figure 15 Syndrome details include the test groups, test names and test results (Pass, Fail or Skip)

Diagnosing a Syndrome

Diagnosing a syndrome is the process whereby Fault Detective detects and isolates faults to determine the most likely cause of a failing test suite. You can diagnose a syndrome by clicking the **Diagnose** button on the toolbar or from a link in the Task Guide. Syndromes are critical data for Fault Detective's diagnosis and analysis processes. The diagnosis process uses a single syndrome that you specify in

the Suspect Faults view. From the test results in this syndrome, Fault Detective produces a list of suspect faults in order of their likelihood. Clicking **Suspect Faults** under a syndrome in the explorer pane shows an ordered list of suspect faults in the detail pane.

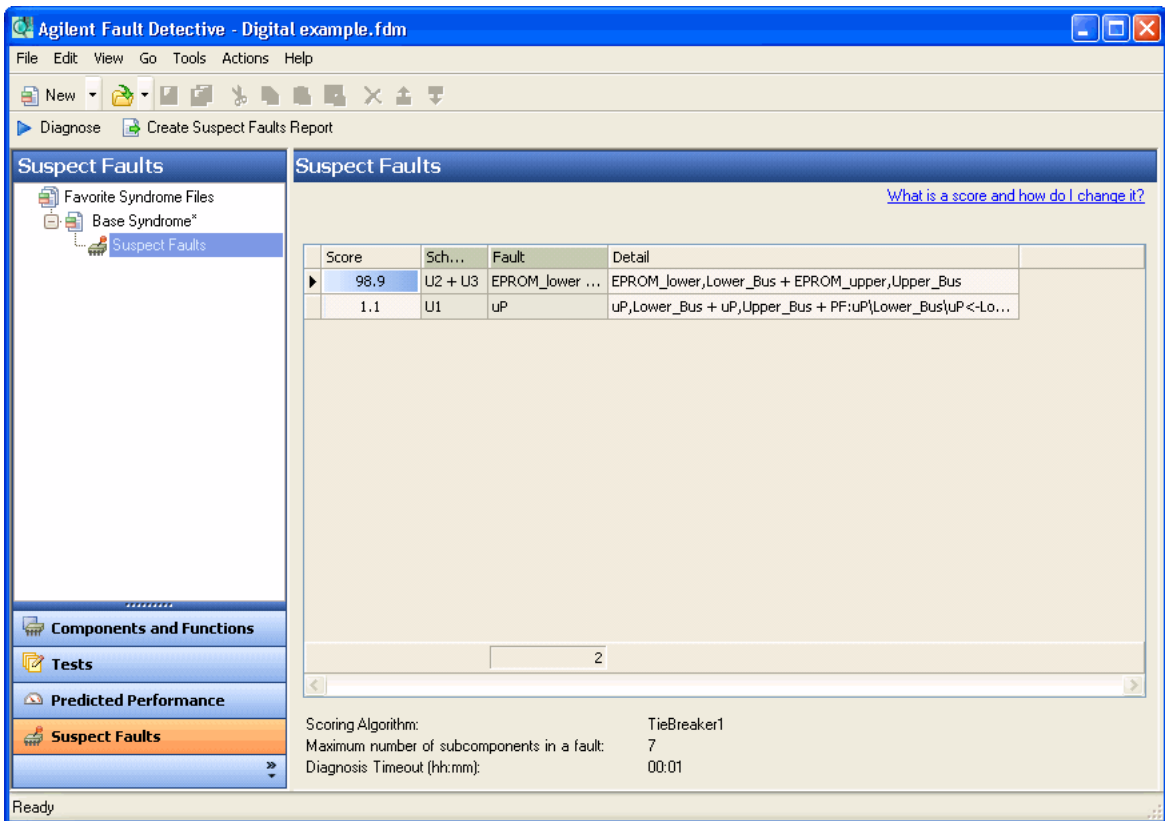
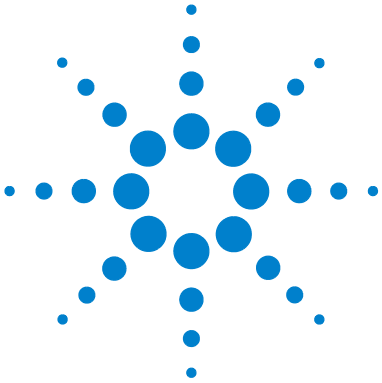


Figure 16 Suspect faults in order of likelihood for *Digital example Default Syndrome*

Creating a Suspect Faults Report

Click the **Create Suspect Faults Report** button on the toolbar or at the bottom of the suspect faults detail pane to create an HTML report for the selected syndrome. Refer to the Fault Detective online help for details regarding this report.



3 Digital Modeling Tutorial

About This Tutorial...	40
Step 1. Create, Save, Close, and Re-Open the Model	42
Step 2. Add a Component	43
Step 3. Add More Components	46
Step 4. Add Subcomponents	49
Step 5. Add Bus Subcomponents	52
Step 6. Add Functions	55
Step 7. Begin Modeling Tests	59
Step 8. Add Coverage	63
Step 9. Verify the Model	65
Step 10. Use Predicted Performance to Improve the Model	69
Step 11. Add More Tests and Coverage	73
Step 12. Iteratively Improve the Model	76
Step 13. Create Reports	78
Where To Go Next	79

This chapter contains a step-by-step tutorial that shows you how to model a digital DUT (Device Under Test). Although the model is for a digital DUT, the tutorial describes basic modeling techniques that apply to both digital and analog DUTs. You should perform this tutorial even if you plan to model analog DUTs. If you are modeling an analog DUT, refer to [Chapter 7](#) for modeling information specific to analog DUTs after completing this tutorial.



About This Tutorial...

The tutorial on the following pages takes about an hour to complete and shows how to model a Device Under Test (DUT) containing mostly digital components. The tutorial uses the *Digital example.fdm* model. This model file is located in:

<drive>:\Program Files\Agilent\Fault Detective\Fault Detective 4.7\Samples\Models\Digital

Model files for completed tutorial steps are located in:

<drive>:\Program Files\Agilent\Fault Detective\Fault Detective 4.7\Samples\Models\Tutorial

As shown in [Figure 17](#), the DUT contains a microprocessor, a data bus, EPROMs, RAM, Flash, a DAC and a power supply.

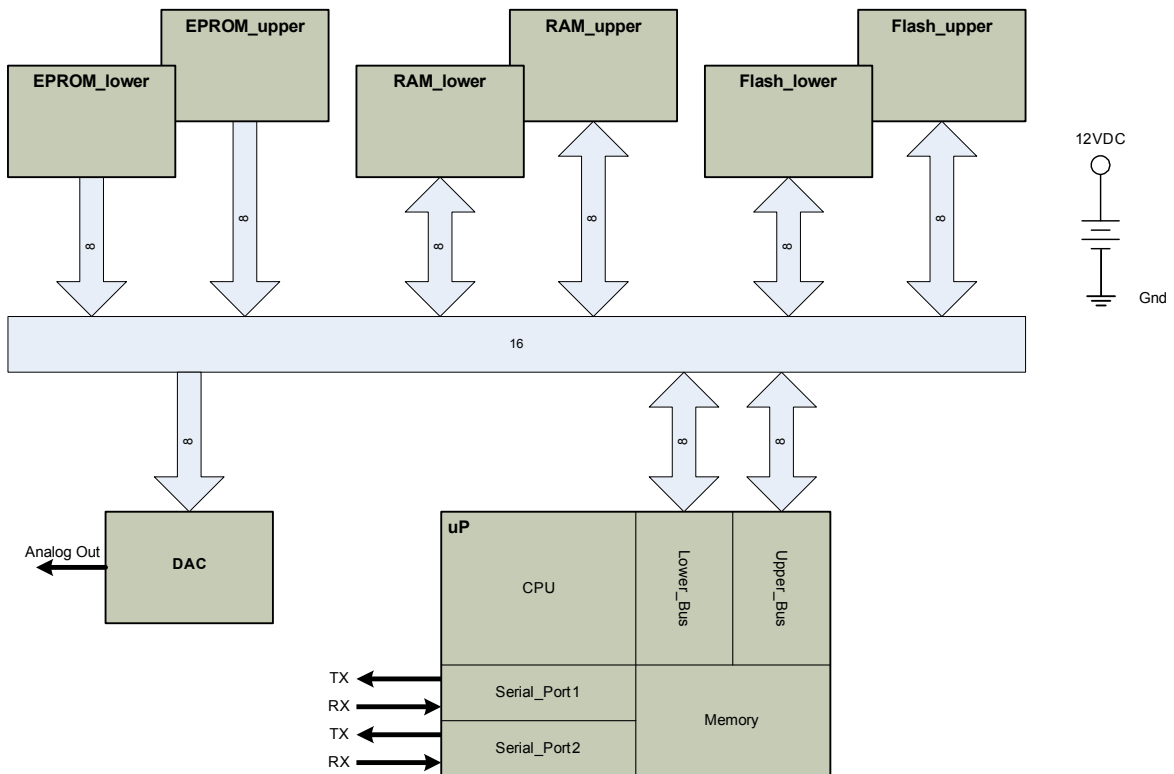


Figure 17 Block diagram of the DUT used in *Digital example.fdm*

Tip

Notice that we are using a block diagram—not a schematic. Modeling is done at a functional test level which usually best correlates to a block diagram. If a block diagram does not exist, you can use a schematic. However, schematics typically have a large number of low-level parts such as resistors, capacitors and inductors. In most cases, these low-level parts are not tested directly by the functional tests and should not be part of the Fault Detective model.

Step 1. Create, Save, Close, and Re-Open the Model



1 To open the Fault Detective application, click **Start > All Programs > Agilent Fault Detective > Fault Detective 4.7 > Fault Detective 4.7**.

2 Create a new model. Click **File > New > Model**.

3 Save and name the model. Click **File > Save As...** and navigate to:

<drive>:\Program Files\Agilent\Fault Detective\Fault Detective 4.7\Samples\Models\Tutorial

Enter the file name *tutorial_step1.fdm* and click **Save**.

4 Close the model. Click **File > Close All**.

Tip

Clicking **Close All** closes and saves the model file (*.fdm*), syndrome files (*.tr*), and test strategy files (*.ts*). The model file holds the components, subcomponents, functions and tests for your model. The syndrome and test strategy files are discussed later in this manual; you will notice that Fault Detective creates a folder (in this case, *tutorial_step1_files*) to hold these files.


5 Click **File > Open > Model**. Navigate to the saved *tutorial_step1.fdm* file and click **Open** to open the model.

Step 2. Add a Component

In Fault Detective, a component represents the functionality provided by the DUT's physical parts. In our model, components are the high-level circuit elements shown in the block diagram. These are the microprocessor, EPROMs, RAM, Flash, DAC, and power supply.

- 1 With the *tutorial_step1.fdm* model open, click the **Components and Functions** button



- 2 Click the add component button  in the task toolbar.

- 3 In the explorer pane, type over *NewComponent* with the name *uP*.

- 4 Enter the following component information (this information is optional but it is good practice to enter):

- **Schematic Reference:** *U1*
- **Part Number:** *68MC302*
- **Diagnostic Comment:** Microprocessor
- **Notes:** *This is a very fast, low-power, 16-bit microprocessor.*

It has 32MBytes of internal RAM.

It also has two serial ports.

NOTE

For this model, leave the Relative Failure Rate set to Medium (the default) for this component, and all components. The Relative Failure Rate can be important in modeling. Refer to the Fault Detective online help for more information (search for *relative failure rate*).

Similarly, leave the Component Cost set to its default, 0.00, for the purposes of this tutorial. You can use the Component Cost to help you determine the repair order of suspect faults. Search for *component cost* in the online help for more information.

- 5 When finished entering the information, click on the component name (*uP*) in the explorer pane (left pane). By moving focus from the detail pane to the explorer pane, you are indicating to Fault Detective that you are done entering the information for this component.

NOTE

Notice that a default function (*NewFunction*) is automatically created for each component. Ignore these default functions for now. You will learn more about functions in “[Step 6. Add Functions](#)”.

- 6 Save and name the model. Click **File > Save As...**, enter the file name *tutorial_step2.fdm* and click **Save**.

Tips

- Instead of using buttons in the task bar, you can add model elements by right-clicking from anywhere in the explorer pane.
- The Task Guide supplements this tutorial and provides how-to information and links to additional help topics relating to the particular view. Keep the Task Guide open (click **View > Task Guide**) while doing this tutorial.

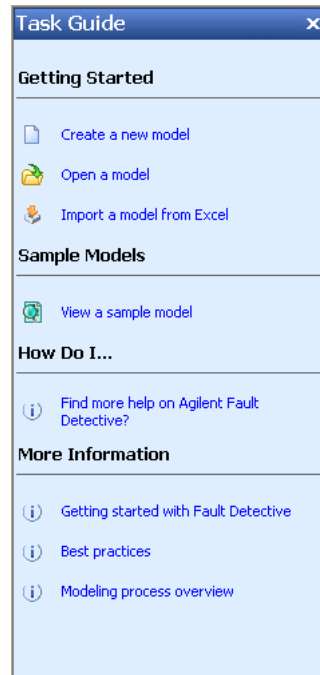



Figure 18 The Task Guide supplements this tutorial

Step 3. Add More Components

When modeling, keep in mind that a component is a *replaceable* part or group of parts. A component will often equate to an integrated circuit such as the microprocessor, EPROM, or DAC. Often, a component consists of a group of parts that are replaced as an assembly—a power supply, for example. You will learn more about making decisions regarding components as we work through the tutorial.

- 1 With the *tutorial_step2.fdm* model open, click the

add component button  .
- 2 In the explorer pane, type over *NewComponent* with the name *EPROM_lower*. Add the additional information for this component from the table below.
- 3 Continue adding the remaining components shown in [Table 1](#). [Figure 19](#) shows the explorer pane with all components added.

NOTE

After you have added a few components from [Table 1](#), you can close the model and open *tutorial_step3.fdm* in this directory path:

<drive>:\Program Files\Agilent\Fault Detective\Fault Detective 4.7\Samples\Models\Tutorial

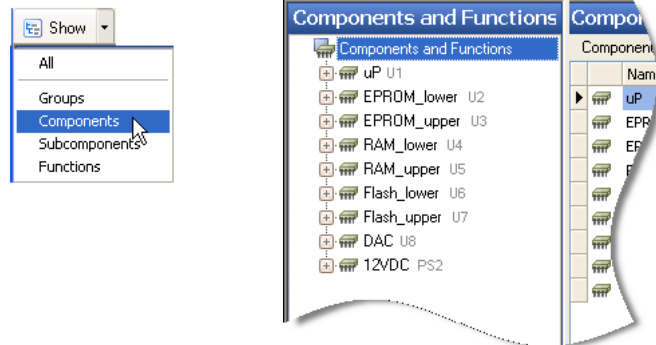
This model file has all the components added for you.

Table 1 Components for the Digital Example

Name (no spaces in name)	Schematic Reference	Part Number	Diagnostic Comment
uP	U1	68MC302	Microprocessor
EPROM_lower	U2	EP-512	EPROM, lower
EPROM_upper	U3	EP-512	EPROM, upper
RAM_lower	U4	DDR2-512	RAM, lower

Table 1 Components for the Digital Example (continued)

Name (no spaces in name)	Schematic Reference	Part Number	Diagnostic Comment
RAM_upper	U5	DDR2-512	RAM, upper
Flash_lower	U6	28F512	Flash, lower
Flash_upper	U7	28F512	Flash, upper
DAC	U8	DAC8	8-bit Digital-to-Analog Converter (DAC)
12VDC	PS2	PS-12VDC	+12 VDC power supply for Flash programming

**Figure 19** Explorer pane showing components

Tips Modeling Components

- The goal when modeling components is to model the relationship between the tests and the DUT by modeling components whose functionality is actually being used in the functional tests.
- To identify components to be modeled, start by going through your tests one by one, asking the question, "Which components must operate to some degree for this test to pass?"
- When choosing components, select those that must be at least partially functional for the tests to pass. Do not model components that cannot cause functional test failures. Typically, a DUT made up of hundreds of parts can be represented by a model with tens of components.



The Notes: Field

You can enter supporting details in the Notes: field or add links to supporting documentation, such as HTML files and URLs. This allows you to provide a transfer of knowledge from the model designer to the production floor. When necessary, test personnel can locate supporting documents such as schematics, block diagrams or parts cost information. The Notes: field is available for components, functions and tests.

Adding Model Elements

This tutorial uses buttons in the task toolbar to add model elements. You can also add model elements from the Task Guide or by right-clicking from anywhere in the explorer pane.

Moving Model Elements in the Explorer Pane

You can drag and drop model elements in the explorer pane or you can use the move up  and move down  buttons in the toolbar to arrange model elements.

Step 4. Add Subcomponents

Subcomponents are logical subsets of a component. For example, if an integrated circuit provides multiple distinct operations, and these operations are tested in different functional tests, you should create a component for the integrated circuit and a subcomponent for each of these operations. This finer level of granularity allows you to define test usage for each subcomponent, giving you a higher degree of accuracy in your Fault Detective results.

In our model, the microprocessor's CPU, upper and lower bus interface, memory, and serial ports are good examples of subcomponents. The microprocessor subcomponents are shown in [Figure 20](#).

As another example, the DAC (Digital-to-Analog Converter) contains a bus interface and a converter subcomponent.

NOTE

Ignore the bus interface subcomponents for now; we will model the bus separately in the next step, "[Step 5. Add Bus Subcomponents](#)".

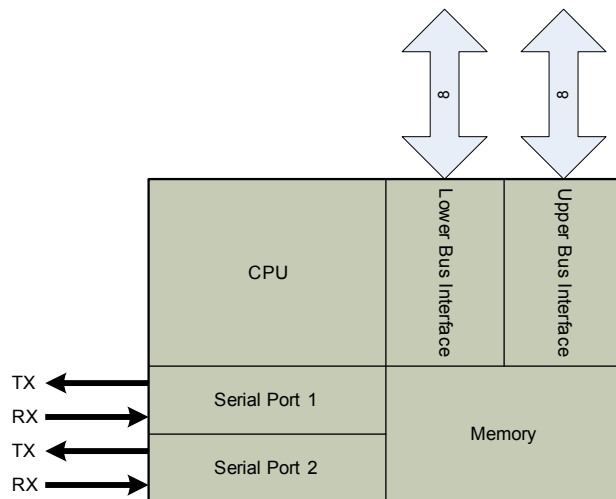



Figure 20 uP subcomponents

- 1 In the explorer pane, click the component *uP*.
- 2 Click the add subcomponent button  . Type over *NewSubcomponent* with the name *CPU*.

NOTE

When you add the first subcomponent to a component, you will see a message box notifying you that the default function for the component has been moved to the subcomponent. Fault Detective is automatically managing the hierarchy of components, subcomponents and functions. To disable this notification, click the **Don't show this dialog box again** checkbox. The function moving operation will still occur even with the notification disabled. You will learn more about functions in [“Step 6. Add Functions”](#).

- 3 Repeat step 2 for the remaining subcomponents in [Table 2](#). [Figure 21](#) shows the explorer pane with these subcomponents added.

NOTE

After you have added a few components from [Table 2](#), you can close the model and open *tutorial_step4.fdm*. This model file has all the uP subcomponents added for you.

Table 2 uP Subcomponents

Component	Subcomponents
uP	CPU
	Memory
	Serial_Port1
	Serial_Port2
DAC	Converter
12VDC	None

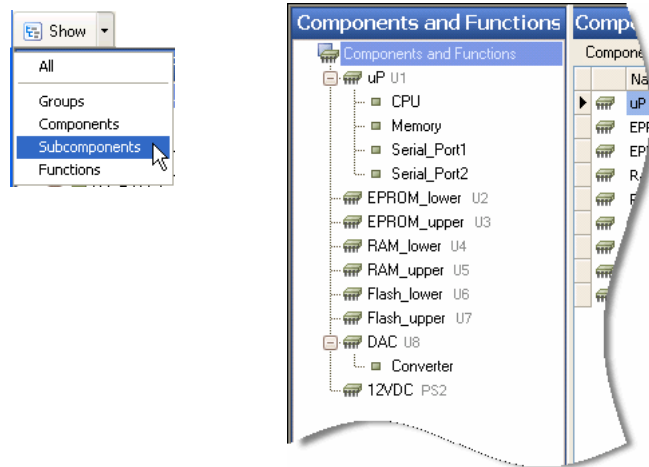


Figure 21 uP and DAC subcomponents in the explorer pane

Tips Why model the power supply (12VDC) as a component with no subcomponents?

The purpose of diagnosing faults is to identify which parts need to be replaced so that a DUT can be repaired. Therefore, you should model replaceable parts of the DUT as individual components. If there is a replaceable module that is made up of many physical parts (such as the power supply), you can model it as a single component. Conversely, if the module provides more than one piece of the functionality of the DUT, it may be best to model it as a single component with subcomponents representing the varied functionality.

Avoid modeling subcomponents unnecessarily.

Subcomponents that are always tested together should not be modeled separately. Modeling unnecessary subcomponents can overly complicate your model.

Step 5. Add Bus Subcomponents

Subcomponents are typically used when modeling a data bus. In our example DUT, we have an 8-bit lower data bus and an 8-bit upper data bus. Each component on the data bus has a bus interface with read and/or write capability. We will model each bus interface as a subcomponent. Later, in “[Step 6. Add Functions](#)”, we will add the appropriate read and/or write functions to these subcomponents.

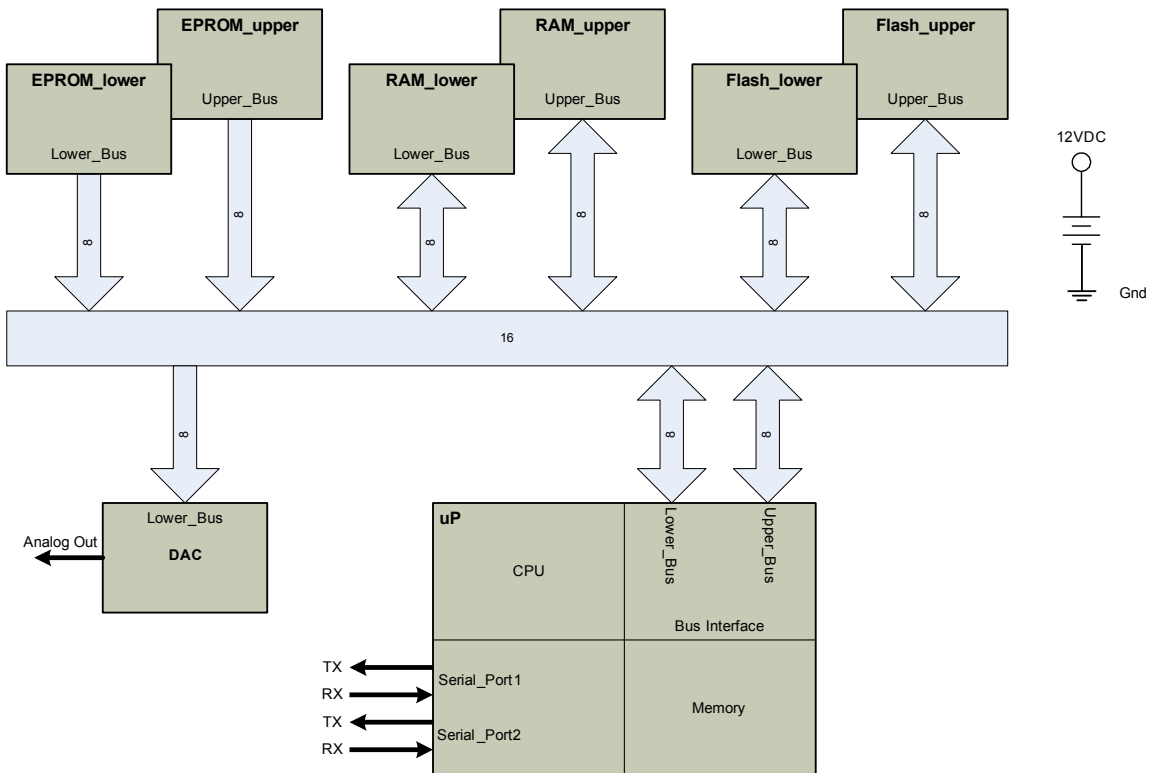




Figure 22 Data buses and bus interface subcomponents

- 1 In the explorer pane, click the component *uP*.
- 2 Click the add subcomponent button . Type over *NewSubcomponent* with the name *Lower_Bus*.
- 3 Click the add subcomponent button . Type over *NewSubcomponent* with the name *Upper_Bus*.
- 4 Repeat steps 1 and 2 for the remaining bus subcomponents in Table 3. Figure 23 shows the explorer pane with all subcomponents added.

NOTE

After you have added a few components from Table 3, you can close the model and open *tutorial_step5.fdm*. This model file has all the bus interface subcomponents added for you.

Table 3 Bus Interface Subcomponents

Component	Subcomponents
uP	Lower_Bus Upper_Bus
EPROM_lower	Lower_Bus
EPROM_upper	Upper_Bus
RAM_lower	Lower_Bus
RAM_upper	Upper_Bus
Flash_lower	Lower_Bus
Flash_upper	Upper_Bus
DAC	Lower_Bus

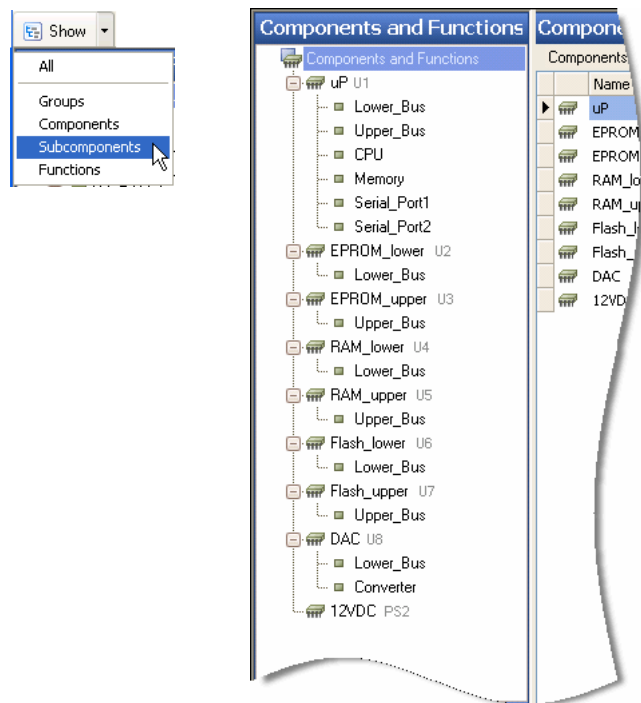


Figure 23 All subcomponents in the explorer pane

Tip Why not model the data bus as a component?


You could consider a data bus to be a single component with read/write capability. However, by making each bus interface a subcomponent, you will be much better able to isolate read/write faults. Also, typically, the data bus is not a replaceable part.

Step 6. Add Functions

Functions are Fault Detective model elements that represent capabilities, operations, or characteristics of a component or subcomponent. For example, a bus interface will typically have read and write functions. Functions are very important in the Fault Detective model. Functions define the coverage relationship between tests and components.

NOTE

At this point in the modeling process, knowledge of your DUT, its functions and the tests covering those functions is required. Before modeling functions, examine the tests in your test suite and understand their coverage of the components/subcomponents in question. You may also have to study the block diagram, schematics, and interview the product designers. To get the best results from Fault Detective, model your functions in such a way that each test uses one or more functions in their entirety. You need not model functions that are not tested.

- 1 In the explorer pane, click the subcomponent *Lower_Bus* under component *uP*. This subcomponent already has a default function. Click this function and, in the center pane, type over the name *NewFunction* with the name *uP->Lower_Bus* (see the Tips at the end of this step for details on the naming conventions used).
- 2 Click the subcomponent *Lower_Bus* again. Click the add function button  to add another function to this subcomponent. Type over the name *NewFunction* with the name *uP<-Lower_Bus*.
- 3 Repeat steps 1 and 2 for the remaining subcomponents and functions listed in [Table 4](#). [Figure 24](#) shows the explorer pane with functions added.

NOTE

After you have added a few functions from [Table 4](#), you can close the model and open *tutorial_step6.fdm*. This model file has all the functions added for you.

Table 4 Components, Subcomponents and Functions

Component	Subcomponent	Function
uP	Lower_Bus	uP->Lower_Bus uP<-Lower_Bus
	Upper_Bus	uP->Upper_Bus uP<-Upper_Bus
	CPU	uP_CPU
	Memory	uP_Memory
	Serial_Port1	uP_Serial_Port1
	Serial_Port2	uP_Serial_Port2
EPROM_lower	Lower_Bus	EPROM_L->uP
EPROM_upper	Upper_Bus	EPROM_U->uP
RAM_lower	Lower_Bus	RAM_L->uP RAM_L<-uP
RAM_upper	Upper_Bus	RAM_U->uP RAM_U<-uP
Flash_lower	Lower_Bus	Flash_L->uP Flash_L<-uP
Flash_upper	Flash_upper	Flash_U->uP Flash_U<-uP
DAC	Lower_Bus	DAC_L<-uP
	Converter	DAC_Converter
12VDC	--	Program_Voltage_for_Flash

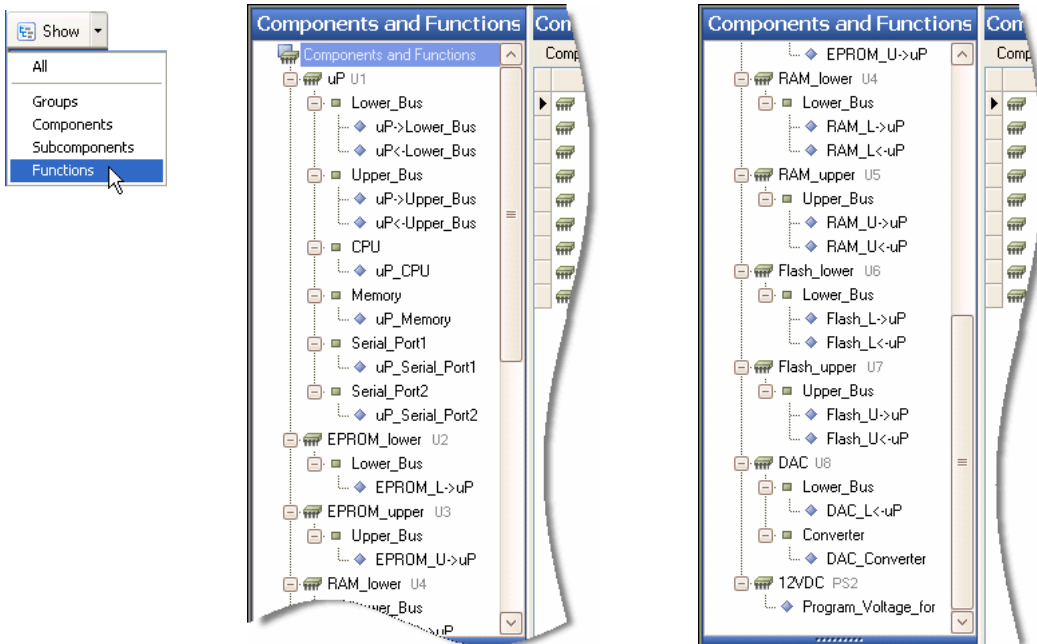


Figure 24 Functions in the explorer pane (scroll down to see all functions)

Tips **How do I determine the Amount of component exercised?**

The **Amount of component exercised** is the degree to which a function exercises the capabilities of a component or subcomponent. To determine whether to set this to *High*, *Medium*, or *Low*, consider the model overall. Review each test, the components that affect it, and how the tests might interact to measure the overall functionality of the DUT. In general, each test that covers a particular component should have a corresponding function to exercise that component; this function represents the functionality of the component that is used by the test. To understand the amount of functionality used by each test (and therefore the amount of the component exercised by the corresponding function), you should understand how that test contributes to measuring the functionality of the DUT.

Simple vs. Global Functions

Our model uses simple functions exclusively. Simple functions are the most common type of function and are subordinate to, or owned by, each component/subcomponent.

Global functions typically exercise two or more components/subcomponents in an identical manner. Global functions tend to be used more often when modeling *analog* DUTs where a testable characteristic of the DUT involves several components/subcomponents.

Relative Variability

In most cases, you should leave **Relative Variability** set to the default value (*Medium*). **Relative Variability** indicates the degree to which a function may pass some tests and fail others. This may happen, for example, because the function's performance in some tests is data-dependent, or because the function is used slightly differently by two tests. (In the latter case, you may want to consider modeling two functions instead of one, if that leads to a more accurate representation of your DUT's functionality.)

Function Naming





Notice we are using a dash and left or right angle brackets to indicate the data flow direction in our bus functions—for example, *uP->Upper_Bus* for a microprocessor write operation or *uP<-Upper_Bus* for a microprocessor read operation. It is important to be as descriptive as possible in the naming of your model elements, especially functions.

Step 7. Begin Modeling Tests

Tests in Fault Detective represent the actual functional tests you perform on your DUT. Functional tests generally make one or more measurements on the DUT, and then pass judgment about some piece of functionality of the DUT. The result is usually pass or fail. If a test passes, it indicates a particular aspect of the DUT has been shown to be functional. If a test fails, it indicates some portion of the functionality of one or more components is not working.

In Fault Detective, you should model each functional test that can return a pass or fail judgment about some portion of the functionality of the DUT. In most cases, every test in a functional test suite should have a corresponding test element in the Fault Detective model. If you have some tests that do not measure the functionality of the DUT, you do not need to include them in the Fault Detective model.

For our model, we will start adding the tests that apply to the CPU, CPU RAM, DAC, and serial ports.

- 1 Click the **Tests** button  .
- 2 Click the add test button  . Type over the name *NewTest* with the name *CPU_Verify*. In the Summary field, enter *Performs internal CPU check*.
- 3 Click the add test group button  . Test groups allow you to organize your tests by grouping related tests. Type over the name *NewTestGroup* with the name *CPU_Int_RAM*.
- 4 With the *CPU_Int_RAM* test group still selected in the explorer pane, click the add test button  to add a test to the test group. Type over the name *NewTest* with the name *Int_RAM_Check_00*. In the Summary field enter *Writes & reads "00" pattern*.

- 5 Repeat steps 3 and 4 for the remaining test groups and tests listed in Table 5. Figure 25 shows the explorer pane with these test groups and tests added.

NOTE

After you have added a few tests from Table 5, you can close the model and open *tutorial_step7.fdm*. This model file has these tests added for you.

Table 5 CPU, CPU_Int_RAM, DAC, Serial_Port1, and Serial_Port2 Test Groups and Tests

Test Group	Test	Summary
--	CPU_Verify	Performs internal CPU check
CPU_Int_RAM	Int_RAM_Check_00	Writes & reads "00" pattern
	Int_RAM_Check_55	Writes & reads "55" pattern
	Int_RAM_Check_AA	Writes & reads "AA" pattern
	Int_RAM_Check_FF	Writes & reads "FF" pattern
DAC	Set_low	uP writes to DAC to set value
	Set_mid	uP writes to DAC to set value
	Set_high	uP writes to DAC to set value
	Read_low	Read output of DAC with external DMM
	Read_mid	Read output of DAC with external DMM
	Read_high	Read output of DAC with external DMM
Serial_Port1	TX	Serial Port 1 (uP) sends data out
	RX	Serial Port 1 (uP) receives data
Serial_Port2	TX	Serial Port 2 (uP) sends data out
	RX	Serial Port 2 (uP) receives data

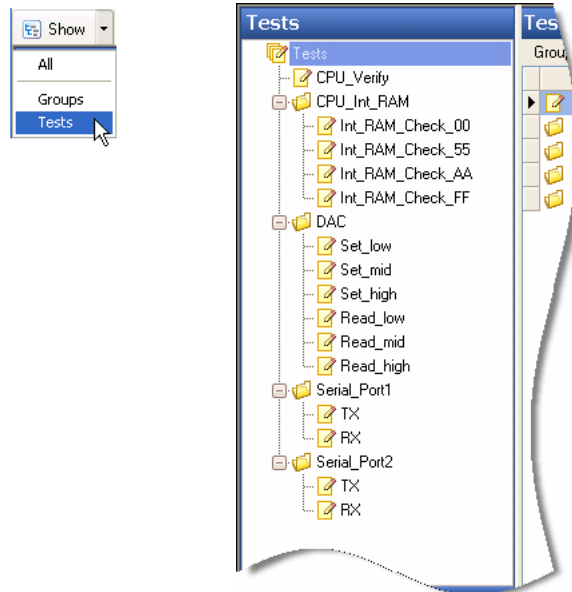


Figure 25 CPU_Verify, CPU_Int_RAM, DAC, Serial_Port1 and Serial_Port2 test groups and tests

Tips Modeling Complex Tests

If a test in your test suite returns more complex information (such as failure codes), you can model it as multiple pass/fail tests in Fault Detective.

Using Groups

Grouping model elements can help you to develop and organize your model. You can group components into component groups, tests into test groups, and global functions into function groups. Groups are shown with a folder icon. Grouping does not affect Fault Detective's diagnosis and analysis processes. Some uses of groups include:


- Component groups to indicate the hierarchy of your block diagram
- Test groups to indicate related tests within your test suite
- Test groups to segregate optional tests (such as manual steps performed by a technician)
- Groups to indicate submodels that are owned or edited by different modelers

Step 8. Add Coverage

Adding coverage to a test means you are specifying that a particular function is used by that test. Coverage defines the relationships between your DUT and your functional test suite. When you add or edit coverage, you are defining the functions that are used by each test in the test suite. Functions exercise components and subcomponents, so the coverage relationship ultimately indicates the components that are tested, or covered, by each modeled test.

NOTE

This step in the modeling process requires specific knowledge of the relationship of your functional test suite to the functions and components/subcomponents of the DUT. Typically, to complete this step you will need to do some research. This usually involves studying the test specifications, interviewing test developers, actually running the tests on a DUT, and so on.

- 1 Click the **Tests** button .
- 2 Click the *CPU_Verify* test in the tests explorer pane.
- 3 In the detail pane, select the **Covered Functions** tab. Make sure **Show: All Functions** is selected and **Compact View** is *not* selected.
- 4 Locate *uP_CPU* in the Name (Function) column of the table.
- 5 In the Used column next to *uP_CPU*, click once to select the cell, then click again to specify usage of this function by the test (denoted by an x in the Used column).

(To see this coverage displayed in the Components and Functions view, you can now return to the Components and Functions view, select the function *uP_CPU* under the *CPU* subcomponent of the *uP* component, and click the **Covering Tests** tab to verify that the *CPU_Verify* test is shown as used.)

6 Repeat step 5 for the remaining test groups, tests and functions shown in Table 6.

Table 6 Test Groups, Tests and Covered Functions

Test Group	Test	Covered Function
--	CPU_Verify	uP_CPU
CPU_Int_RAM	Int_RAM_Check_00	uP_Memory
	Int_RAM_Check_55	uP_Memory
	Int_RAM_Check_AA	uP_Memory
	Int_RAM_Check_FF	uP_Memory

Step 9. Verify the Model

When modeling, you should periodically check for errors and problems in the model. For example, omitting the information that *Test A* uses *Function B* can cause incorrect fault diagnosis results, whether Test A passes or fails.

Since we are just beginning to model tests in our model, we are quite certain we have unused functions and unused tests. In modeling, we will do an iterative process of verifying the model and adding tests and coverage.

Check for Unused Functions

Unused functions are those functions not used by any test.

- 1 Click the **Components and Functions** button



- 2 Select **Components and Functions** at the top of the explorer pane. Examine the Unused Functions column in the detail pane. The number of unused functions in a component or group, if nonzero, is listed in red (see Figure 26).

Components and Functions		Components and Functions			
Components: 9		Name	Type	Subcomponents	Unused Functions
+	uP U1	uP	Component	6	6
+	EPROM_lower U2	EPROM_lower	Component	1	1
+	EPROM_upper U3	EPROM_upper	Component	1	1
+	RAM_lower U4	RAM_lower	Component	1	2
+	RAM_upper U5	RAM_upper	Component	1	2
+	Flash_lower U6	Flash_lower	Component	1	2
+	Flash_upper U7	Flash_upper	Component	1	2
+	DAC U8	DAC	Component	2	2
+	12VDC PS2	12VDC	Component	0	1

Figure 26 Unused functions shown in red


Resolving Unused Functions

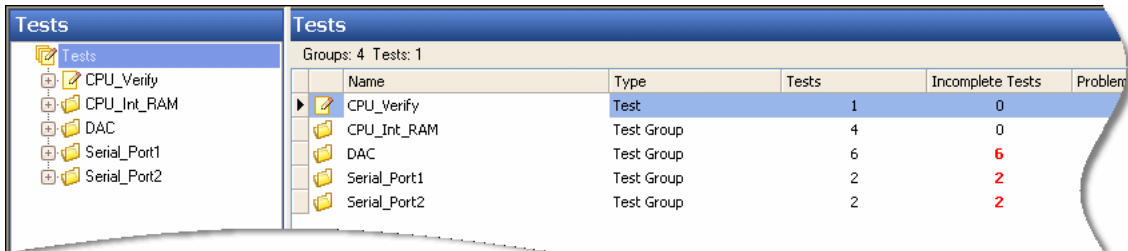
Unused functions can indicate holes in your functional test suite. Typically, you should resolve any unused functions by adding tests and coverage to the model and, if necessary, by adding tests to the test suite.

In the following tutorial steps we will add more tests and coverage to the model to resolve the unused functions.

Check for Incomplete Tests

Incomplete tests are tests that do not use any functions.

- 1 Click the **Tests** button .
- 2 Select **Tests** at the top of the tests explorer pane. Notice the Incomplete Tests column in the detail pane. The number of incomplete tests, if nonzero, is listed in red for each test or test group (see [Figure 27](#)).



Tests		Tests			
Tests		Groups: 4 Tests: 1			
	Name	Type	Tests	Incomplete Tests	Problems
▶	CPU_Verify	Test	1	0	
▶	CPU_Int_RAM	Test Group	4	0	
▶	DAC	Test Group	6	6	
▶	Serial_Port1	Test Group	2	2	
▶	Serial_Port2	Test Group	2	2	

Figure 27 Incomplete tests shown in red

Resolving Incomplete Tests

Typically, incomplete tests simply indicate that you are not finished defining the model, although they may also signify an omission in the test suite itself. If you believe your model is complete but it still contains incomplete tests, you should investigate those tests to determine whether they are correctly modeled and whether the test suite is sufficient.

For this model, we just need to add coverage to the tests in the *DAC*, *Serial_Port1*, and *Serial_Port2* test groups.

- 1 Click the *DAC* test group in the detail pane to show all the tests in the test group.
- 2 Click *Set_low* and, on the **Covered Functions** tab, make sure **Show: All Functions** is selected and **Compact View** is not selected.
- 3 Add coverage for the *DAC_L<-uP* and *uP->Lower_Bus* functions.
- 4 Repeat step 2 for the remaining tests and functions shown in [Table 7](#).

NOTE

After you have added a few tests from [Table 7](#), you can close the model and open *tutorial_step9.fdm*. This model file has these tests added for you.

Table 7 DAC and Serial Port Test Groups, Tests and Covered Functions

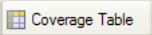
Test Group	Test	Covered Function
DAC	Set_low	DAC_L<-uP uP->Lower_Bus
	Set_mid	DAC_L<-uP uP->Lower_Bus
	Set_high	DAC_L<-uP uP->Lower_Bus

Table 7 DAC and Serial Port Test Groups, Tests and Covered Functions (continued)

Test Group	Test	Covered Function
	Read_low	DAC_Converter
	Read_mid	DAC_Converter
	Read_high	DAC_Converter
Serial_Port1	TX	uP_Serial_Port1
	RX	uP_Serial_Port1
Serial_Port2	TX	uP_Serial_Port2
	RX	uP_Serial_Port2

Tip Defining Coverage

As shown in this tutorial, you can define coverage in the detail pane for a test. You can also define coverage in the detail pane for a function, or in the Coverage Table dialog box by clicking

 Coverage Table . The editor you choose depends on the size of your model, the type of changes you are making (whether you are making changes to the coverage of a single test, or defining coverage for your whole model), and your personal preferences. The coverage editors always remain synchronized, so you can edit in whichever one is appropriate at the moment.

Step 10. Use Predicted Performance to Improve the Model


Fault Detective performs the predicted performance analysis by simulating defective components (faults) in your model and simulating the various patterns of test results (syndromes) that may be exhibited by these defective components. The results of this simulation are used to predict the performance of your test suite against the modeled device under test.

Analysis is done on a *test strategy* consisting of a set of selected tests and a set of selected components—only the selected tests and components are considered in the analysis results. Test strategy analysis can help you locate missing coverage and other modeling errors, and improve your model's performance. Analysis helps you to optimize your use of subcomponents and functions, and fine-tune your model.

- 1 Click the **Predicted Performance** button



Click on *Base Test Strategy [Default]* in the explorer pane to select the default test strategy for your initial analysis.

- 2 Click the **Analyze** button  to begin the analysis. When prompted, click **OK** and save the files. You will see messages showing the progress of the analysis. Since we have just added a few tests, we are not expecting good performance at this point.

NOTE

When you save the *Base Test Strategy* file, the *[Default]* indicator disappears. *[Default]* indicates an element (test strategy or syndrome) that was created automatically by Fault Detective, and has been neither saved nor edited.

- 3 Click **Summary** in the explorer pane. As shown in the Fault Analysis Summary in the detail pane, our detection performance, as we expected, is poor.

NOTE

To see definitions for terms such as detection and isolation, press F1 or click on [What do these numbers mean?](#)

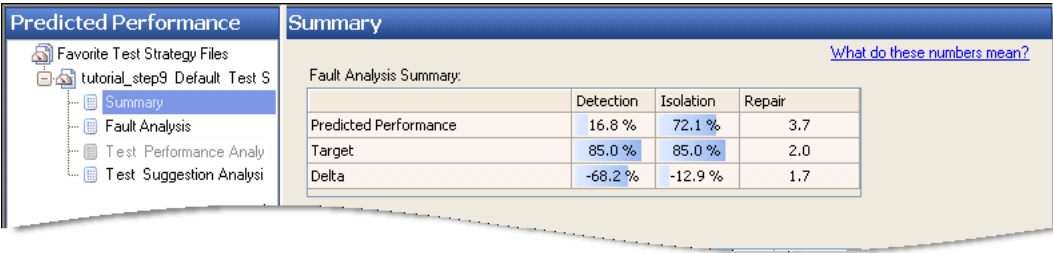


Figure 28 Fault Analysis Summary

NOTE

You will see slightly different analysis results than those shown here when you run the analysis. You will also see slight differences when successively running the analysis without having changed the test strategy. This is because, by default, the simulation starts from a randomly generated number (seed number). You can set the starting number to a fixed value in the **Tools > Options** dialog box. Click the **Predicted Performance** tab, click **User Specified** under the **Simulation seed number** and enter a starting number. When using the same starting number for each analysis, the analysis numbers will be identical for successive analysis runs on the same test strategy. However, you will achieve more accurate results by using random seed numbers and averaging the results of multiple analysis runs.

- 4 Click **Fault Analysis** in the explorer pane. We now get more information on how our model can be improved. We currently have 0% detection and isolation for all components except *uP* and *DAC*.

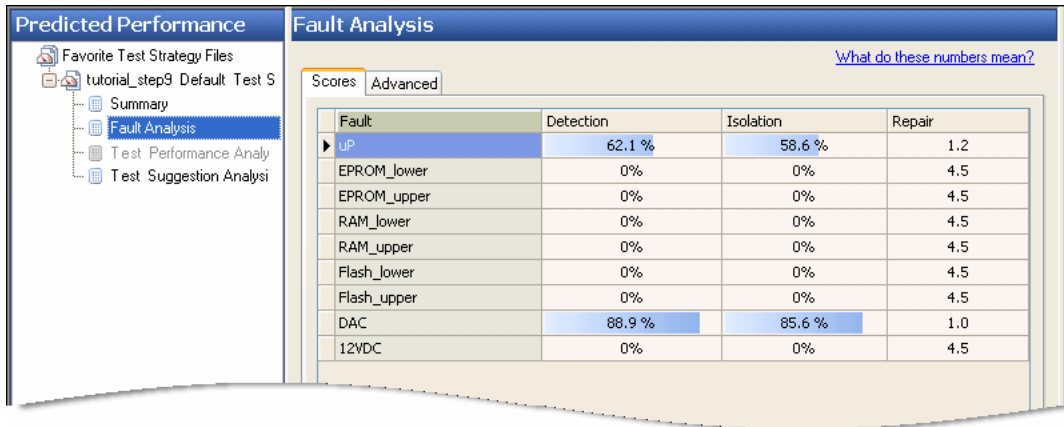


Figure 29 Fault Analysis

- Click **Test Suggestion Analysis** in the explorer pane. Fault Detective shows us which faults can be improved to get better performance. As we suspected, the components that we have not yet added tests and coverage for need 100% improvement. We will add tests and coverage for the faults in “[Step 11. Add More Tests and Coverage](#)”.

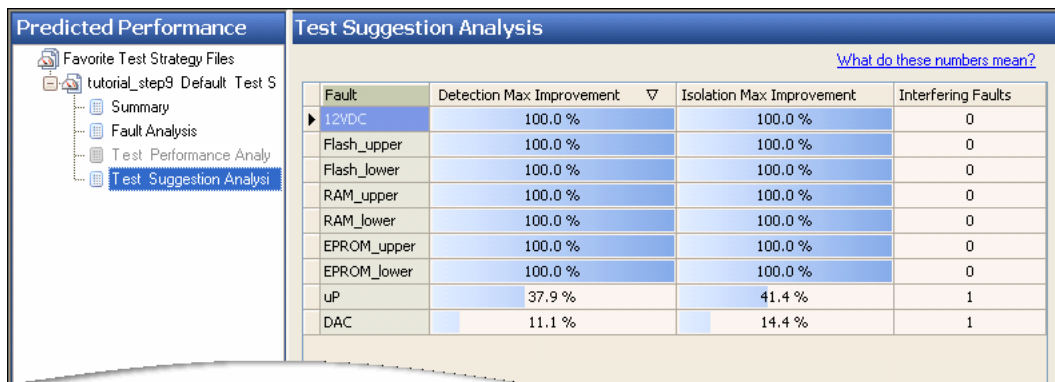


Figure 30 Test Suggestion Analysis

Tips

- Notice the [What do these numbers mean?](#) help link. You can find these help links throughout the application to help you with the terminology and numbers. You can also, at any time, press the F1 key to get help on the current window.
- Do not (except temporarily, as an experiment) modify your model to improve analysis results if this means the model no longer accurately represents your DUT and test suite.

Step 11. Add More Tests and Coverage

We will now add the remaining tests that cover the functions associated with the data bus, DAC, and power supply.

- 1 Add the test groups and tests listed in [Table 8](#).
- 2 To add coverage of many tests at once, click **Tests** at the top of the Tests explorer pane, then click the **Coverage Table** button in the toolbar.

NOTE

After you have added a few of the tests from [Table 8](#) and defined their coverage, you can open *tutorial_step11.fdm*. This model file has all these model elements added for you.

Table 8 Bus, DAC, and Power Supply Test Groups, Tests and Covered Functions

Test Group	Test	Summary	Function
EPROM	Lower_Read	uP reads from EPROM on Lower bus	uP<-Lower_Bus EPROM_L->uP
	Upper_Read	uP reads from EPROM on Upper bus	uP<-Upper_Bus EPROM_U->uP
	Word_Read	uP reads from EPROM on Lower & Upper bus	uP<-Lower_Bus uP<-Upper_Bus EPROM_L->uP EPROM_U->uP
RAM	Lower_Read	uP reads from RAM on Lower bus	uP<-Lower_Bus RAM_L->uP
	Upper_Read	uP reads from RAM on Upper bus	uP<-Upper_Bus RAM_U->uP
	Word_Read	uP reads from RAM on Lower & Upper bus	uP<-Lower_Bus uP<-Upper_Bus RAM_L->uP RAM_U->uP

Table 8 Bus, DAC, and Power Supply Test Groups, Tests and Covered Functions (continued)

Test Group	Test	Summary	Function
	Lower_Write	uP writes to RAM on Lower bus	uP->Lower_Bus RAM_L<-uP
	Upper_Write	uP writes to RAM on Upper bus	uP->Upper_Bus RAM_U<-uP
	Word_Write	uP writes to RAM on Lower & Upper bus	uP->Lower_Bus uP->Upper_Bus RAM_L<-uP RAM_U<-uP
Flash	Lower_Read	uP reads from Flash on Lower bus	uP<-Lower_Bus Flash_L->uP
	Upper_Read	uP reads from Flash on Upper bus	uP<-Upper_Bus Flash_U->uP
	Word_Read	uP reads from Flash on Lower & Upper bus	uP<-Lower_Bus uP<-Upper_Bus Flash_L->uP Flash_U->uP
	Lower_Write	uP writes to Flash on Lower bus	uP->Lower_Bus Flash_L<-uP Program_Voltage_for_Flash
	Upper_Write	uP writes to Flash on Upper bus	uP->Upper_Bus Flash_U<-uP Program_Voltage_for_Flash
	Word_Write	uP writes to Flash on Lower & Upper bus	uP->Lower_Bus uP->Upper_Bus Flash_L<-uP Flash_U<-uP Program_Voltage_for_Flash

Tip**Overlapping Tests**

Fault Detective uses passing tests, failing tests, and overlapping tests (multiple tests covering the same components) to diagnose faults. Notice that, in our model, we have multiple tests using the same functions involving the data bus. The importance of overlapping tests is discussed in greater detail in [Chapter 4](#), “Diagnosing Suspect Faults”.

Step 12. Iteratively Improve the Model

We have completed the model for the digital DUT. In actual practice, the final modeling “step” involves many iterations of adding additional tests and coverage, verifying the model, and running the analysis. The point is to improve the model until it truly reflects your functional test suite and, if necessary, improve your functional test suite (see Chapter 5, “Predicted Performance” for more information). Set your Predicted Performance targets appropriately to help you determine whether your model and test suite are performing adequately for your needs. As you continually improve the model, you may also find it necessary to add or remove some components, subcomponents, functions, and tests.

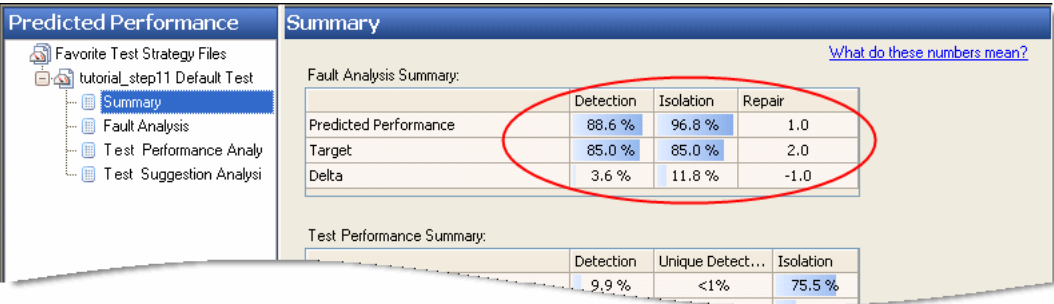


Figure 31 Actual and target values in the Predicted Performance Summary

Tips **Component Cross-Checks**

When you have modeled all your tests and the tested components, subcomponents, and functions, there will probably be many parts that are not included in any of the modeled components. You should ask yourself the following questions about each of these parts:

- **Can the part cause any of the tests to fail?** If it can, then re-examine the tests to determine which tests require the part to operate and to what degree, and then add the part to the model.
- **Can the part cause a failure in actual use?** If the part can cause a functional failure, but cannot cause any of the existing tests to fail, then the existing test suite is not fully exercising the DUT. Additional tests are needed to diagnose failures caused by this part.

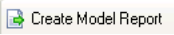
For more information, see *How Do I Know When My Model is Done?* in the Fault Detective online help.

Step 13. Create Reports

The model report and predicted performance report are particularly useful for documenting your model and its performance.


Creating a Model Report

From the Components and Functions view or the Tests view,

click the **Create Model Report** button  on the toolbar to create an HTML report for the model.

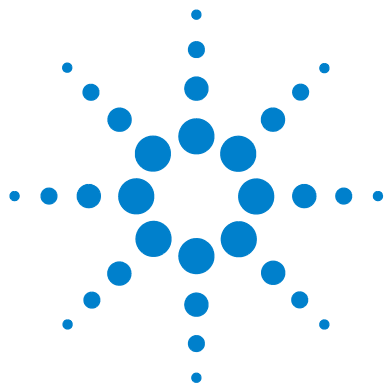
Creating a Predicted Performance Report

From the Predicted Performance view, click the **Create Predicted Performance Report** button

 on the toolbar to create an HTML report for the selected test strategy.

Where To Go Next

[Chapter 4](#), “Diagnosing Suspect Faults,” starting on page 81 uses the model we just created to show how Fault Detective performs diagnosis and how the model can influence the diagnosis.



4 Diagnosing Suspect Faults

About Diagnosing Suspect Faults...	82
How Fault Detective Diagnoses Suspect Faults	83
Running the Diagnosis	85
What's in the Online Help?	89

This chapter describes how to use Fault Detective to diagnose suspect faults and describes how Fault Detective uses passing, failing, and overlapping tests (multiple tests covering the same components) in the diagnosis.



About Diagnosing Suspect Faults...

When you run a diagnosis, Fault Detective identifies suspect faults to determine the most likely cause(s) of the failing test(s) in a particular syndrome (a set of test results). Fault Detective stores syndromes in files with the extension *.tr*.

After the diagnosis process runs, Fault Detective displays an ordered list of suspect faults in the Suspect Faults detail pane. Each suspect fault has a diagnosis score which is a relative score that shows how well this suspect fault explains the syndrome being diagnosed. The total of all the scores for all the suspect faults is always 100.

Fault Detective also displays the repair cost and recommended repair order for each suspect fault (not shown in this tutorial). The repair cost for a suspect fault is the total of the component costs of all components in the suspect fault. The recommended repair order is based on the repair cost and the repair score. For more information, search for *repair cost* and *repair order* in the Fault Detective online help.

How Fault Detective Diagnoses Suspect Faults

When performing a diagnosis, Fault Detective considers which sets of components can explain all of the failed tests. For each set of components, Fault Detective reviews the passing tests to determine if any of the suspect components are covered by one or more passing tests. A component that is covered by a passing test is at least partially functional. To determine the diagnosis score of a suspect fault, Fault Detective considers both the failing tests and the passing tests that involved each component. Failing tests increase the score of the suspect fault, passing tests decrease the score.

The score is a function of the component's relative failure rate, the number of passing and failing tests, the relative variability of each function, and the amount of the component exercised by each function.

Diagnosis Example

Figure 32 shows some of the lower bus-related tests, functions, and components from the *Digital example.fdm* model. In this example, the microprocessor's *uP<-Lower_Bus* function is used by six tests that also cover the EPROM, RAM, and Flash modules of the DUT. This example is used in the procedure “Running the Diagnosis” on page 85.

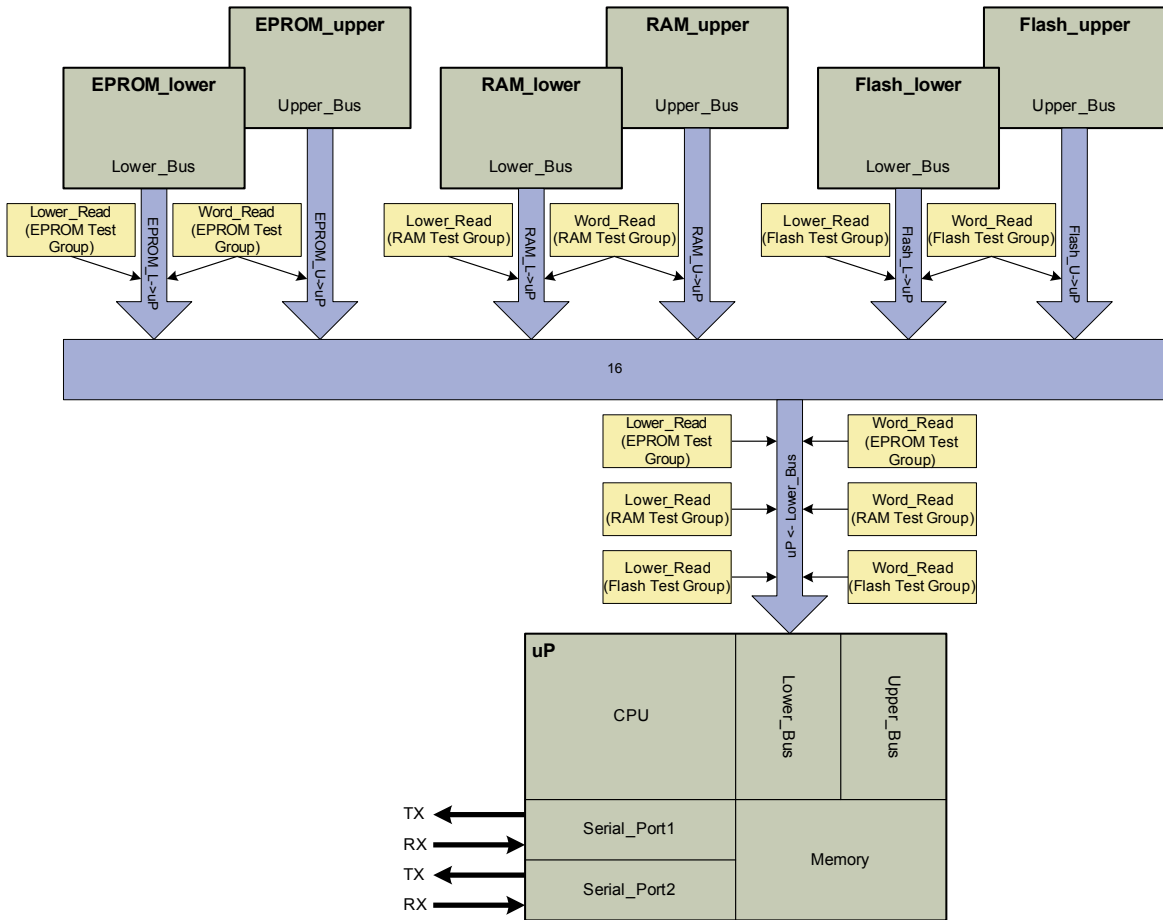


Figure 32 Tests using the *uP<-Lower_Bus* function in the *Digital example.fdm* model


Running the Diagnosis

This procedure shows how to run the diagnosis on a particular syndrome. A syndrome is simply a set of test results, with each test result set to Pass, Fail, or Skip. The procedure also shows how Fault Detective uses passing, failing, and overlapping tests to diagnose suspect faults.

- 1 Click **File > Open > Model**. Navigate to the *Digital example.fdm* file located in:

<drive>:\Program Files\Agilent\Fault Detective\Fault Detective 4.7\Samples\Models\Digital

Click **Open** to open the model.

- 2 Click the Suspect Faults button . Click the *Base Syndrome* in the explorer pane. (You can create additional syndromes by

clicking  .)

NOTE

Notice that Fault Detective set the result of the first test to Fail and all others to Pass. This is to avoid a warning that occurs when all tests are set to Pass (at least one test result must be set to Fail in order to run the diagnosis).

- 3 Modify the base syndrome to show passing and failing tests. To start, set all cells in the Result column to *Pass*. Note that an asterisk (*) appears after the name *Base Syndrome* to indicate that the syndrome has been changed, but the changes have not been saved.

NOTE

To select multiple cells, click the column header, or drag the mouse over a number of cells in the Result column, or hold down the **CTRL** or **Shift** key and click cells. After selecting multiple cells, right-click and select **Set to Pass**.

- 4 Click in the **Result** column next to the *Lower_Read* test in the *RAM* test group until the result is set to *Fail*. Leave all other tests set to *Pass*.

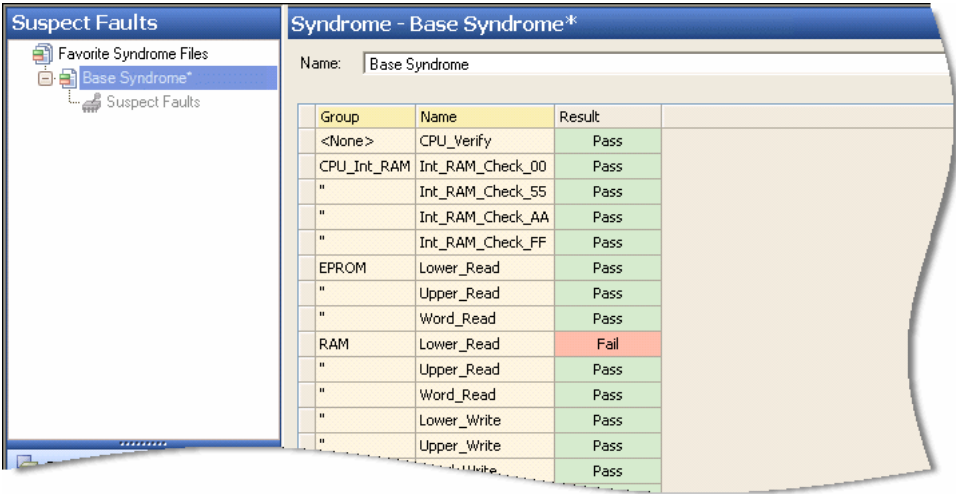



Figure 33 Set the *RAM Lower_Read* test to Fail, all others to Pass

- 5 Click the **Diagnose** button . Click **OK** > **Save** when prompted to save the files. Fault Detective runs the diagnosis. Following the diagnosis, Fault Detective displays an ordered list of suspect faults in the Suspect Faults detail pane.

As shown in [Figure 32](#), the *Lower_Read* test uses the microprocessor's *uP<-Lower_Bus* function and the *RAM_L->uP* function. Since only the *Lower_Read* test failed, it is unclear whether the fault lies with the U1 microprocessor component or the U4 RAM component. Fault Detective gives each component U4 and U1 a 50.0 diagnosis score.

Suspect Faults		Suspect Faults		
<div> <div>Favorite Syndrome Files</div> <div>Base Syndrome</div> <div>Suspect Faults</div> </div>		What is a score and how do I change it?		
Score	Sch...	Fault	Detail	
50.0	U1	uP	uP_Lower_Bus + PF:uP_Lower_Bus{uP<-Lower_Bus	
50.0	U4	RAM_lower	RAM_lower_Lower_Bus + PF:RAM_lower_Lower_Bus{RAM_L...	

Figure 34 Only one test failed, all others passed; either U1 or U4 could be at fault

- 6** The *Word_Read* test and the *Lower_Read* test are overlapping tests (overlapping because both tests use the *uP<-Lower_Bus* and *RAM_L->uP* functions). In the *RAM* test group, set the *Word_Read* and *Lower_Read* tests to *Fail* and set all other tests to *Pass*. Run the diagnosis.

Since two tests relating to U4's lower bus failed, and all other tests passed, Fault Detective gives U4 a 99.5 score and U1 a <1 score. This shows the value of overlapping, but different, tests. Fault Detective has determined that U4 is almost certainly faulty by examining both the passing and failing tests.

Suspect Faults		Suspect Faults		
<div> <div>Favorite Syndrome Files</div> <div>Base Syndrome</div> <div>Suspect Faults</div> </div>		What is a score and how do I change it?		
Score	Sch...	Fault	Detail	
99.5	U4	RAM_lower	RAM_lower_Lower_Bus + FO:RAM_lower_Lower_Bus{RAM_L...	
<1	U1	uP	uP_Lower_Bus + PF:uP_Lower_Bus{uP<-Lower_Bus	

Figure 35 Two overlapping tests failed, all other passed; U4 is most likely at fault

- 7 In the *RAM* test group, leave *Lower_Read* and *Word_Read* set to *Fail* and set all other tests to *Skip*. (Right-click the column header, click **Set to Skip**, then change the two tests back to *Fail*.) Run the diagnosis.

Notice that Fault Detective did not give U4 a high score as it did when the other tests were set to *Pass*. This is because, with all other tests set to *Skip*, Fault Detective cannot use the results of passing tests in the diagnosis.

- 8 Continue experimenting with *Pass*, *Fail*, and *Skip* to see how different combinations of test results affect the diagnosis.
- 9 You can create an HTML diagnosis report for the selected

syndrome by clicking  .

Scoring Algorithms

Fault Detective has different scoring algorithms to detect and score (rank) suspect faults. You can choose a scoring algorithm in the Options dialog box (click **Tools > Options...** > **Suspect Faults** tab). The default algorithm is Tie Breaker 1. Refer to the Fault Detective online help for details regarding each scoring algorithm.

What's in the Online Help?

Here are some of the topics in the Fault Detective online help relating to diagnosing suspect faults (search for *diagnosis* or *suspect faults*):

- About Fault Details
- About Scoring Algorithms
- About Syndromes
- Creating and Loading Syndrome Files
- Reading the Suspect Faults Report
- When to Use Tie Breaker 2 Scoring



5 Predicted Performance

About Predicted Performance...	92
Running the Predicted Performance Analysis	94
Using Analysis to Improve Test Coverage	97
Using Analysis to Optimize the Test Suite	101
What's in the Online Help?	106

You have already seen some of the benefits of Fault Detective's predicted performance analysis capabilities in developing a model ([Chapter 3](#), "Digital Modeling Tutorial"). This Predicted Performance chapter shows some of the many ways you can use the predicted performance analysis to improve the test suite.



About Predicted Performance...

The Predicted Performance view allows you to create a test strategy, run the analysis, and create a predicted performance report.

Fault Detective performs the predicted performance analysis by simulating defective components (faults) in your model and simulating the various patterns of test results (syndromes) that may be exhibited by these defective components. The results of this simulation are used to predict the performance of your test suite against the modeled device under test.

About Test Strategies...

The predicted performance analysis is performed on a test strategy containing selected tests and components. The analysis can help you locate modeling errors, and improve the performance of your model and the test suite. The analysis creates predicted performance data consisting of:

- Predicted performance of the test strategy
- Predicted performance of individual faults
- Predicted performance of individual tests
- Suggestions for additional tests

Each test strategy consists of a set of selected tests and a set of selected components. When Fault Detective analyzes a test strategy, only the selected tests and components are considered in the analysis results. The predicted performance displayed for a test strategy predicts what would result from running only the selected tests, on a DUT that consists of only the selected components. Excluded tests are not simulated, and excluded components are not considered as possible faults. Fault Detective stores test strategies in files with a *.ts* extension.

You can use a test strategy to analyze:

- The tests and components that are the focus of a given test stage in the testing process, such as process test, in-circuit test, functional test, and repair station test.
- A proposed or experimental approach to minimizing the number or duration of tests in a test suite while optimizing some outcome, such as fault detection at functional test or fault isolation at the repair station.
- A subset of the model that may contain a modeling error or problem that you are trying to find.

Fault Detective's analysis process provides a great deal of information to assist you in improving your model. In the Options dialog box, you can set target values for the predicted fault detection and fault isolation performance of your model. The predicted performance report tells you whether you have achieved these targets with your current model and test strategy, and if not, where your model and strategy fall short of the targets. This is one way to decide whether your model is ready to deploy.

Running the Predicted Performance Analysis

This procedure shows how to run the predicted performance analysis on a particular test strategy.

- 1 Click **File > Open > Model**. Navigate to the *Digital example.fdm* file located in:

<drive>:\Program Files\Agilent\Fault Detective\Fault Detective 4.7\Samples\Models\Digital

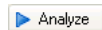
Click **Open** to open the model.

- 2 Click the predicted performance button



. Click the *Base Test Strategy* in the explorer pane.

- 3 By default, the base test strategy has all tests and components selected. Run the analysis by clicking



. Click **OK > Save** when prompted to save the files.

- 4 The analysis produces a predicted performance summary similar to that shown in [Figure 36](#).

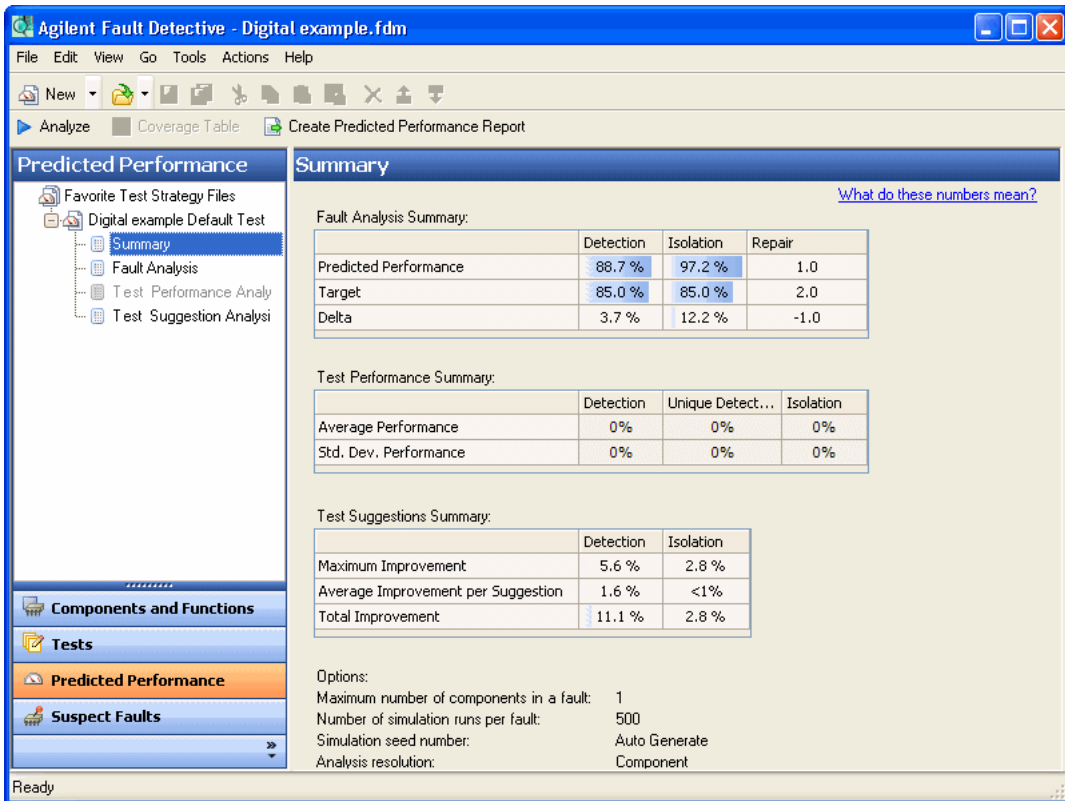


Figure 36 Typical Predicted Performance Summary for the *Digital example.fdm* model

NOTE

You will see slightly different analysis results than those shown here when you run the analysis. You will also see slight differences when successively running the analysis without having changed the test strategy. This is because, by default, the simulation starts from a randomly generated number (seed number). You can set the starting number to a fixed value in the **Tools > Options** dialog box. Click the **Predicted Performance** tab, click **User Specified** under the **Simulation seed number** and enter a starting number. When using the same starting number for each analysis, the analysis numbers will be identical for successive analysis runs on the same test strategy.

Using Analysis to Improve Test Coverage

Although the analysis numbers in Figure 36 are good, there are still opportunities for improvement.

- 1 Click **Fault Analysis** in the explorer pane. The power supply (12VDC) has a low detection score.

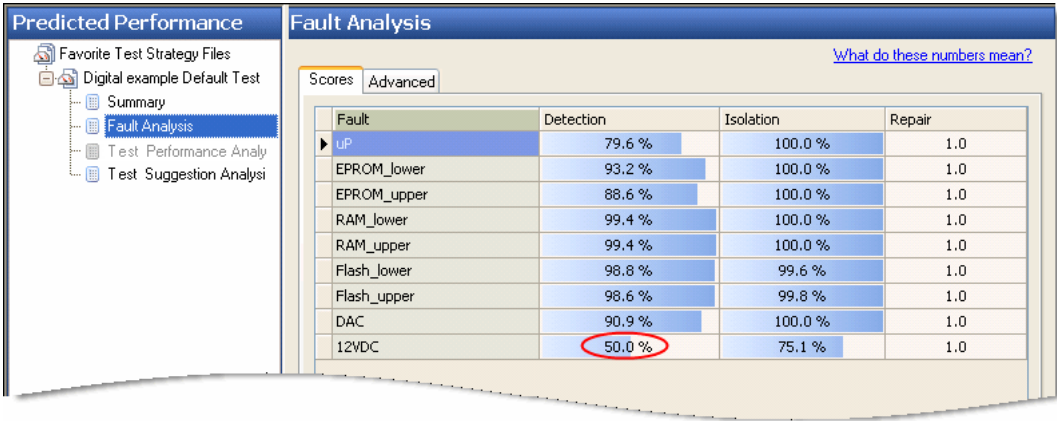


Figure 37 12VDC has a low Detection score

Click **Test Suggestion Analysis** in the explorer pane. Fault Detective is suggesting improved detection performance by adding additional tests for the power supply.

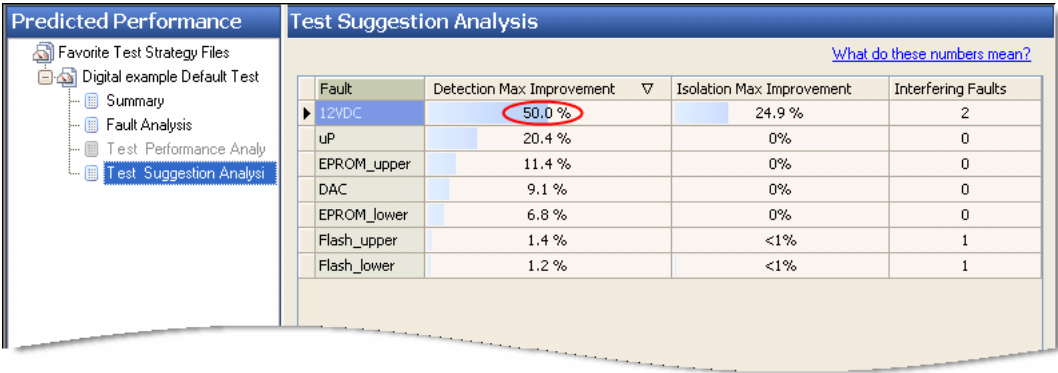


Figure 38 12VDC has the best opportunity for Detection Max Improvement

- 2 By adding a test that checks the power supply’s output voltage under load, we can be much more certain the power supply is operating within specifications. This test uses the same function *Program_Voltage_for_Flash* used to program the Flash.

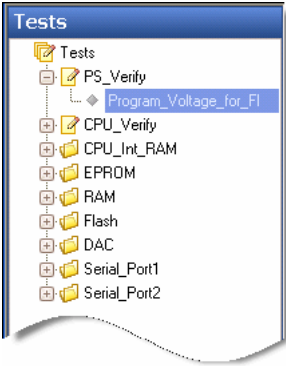


Figure 39 Adding a test to verify the power supply voltage

Since this new test fully verifies the power supply’s output specification, the **Amount of component exercised** can be increased to *High*.

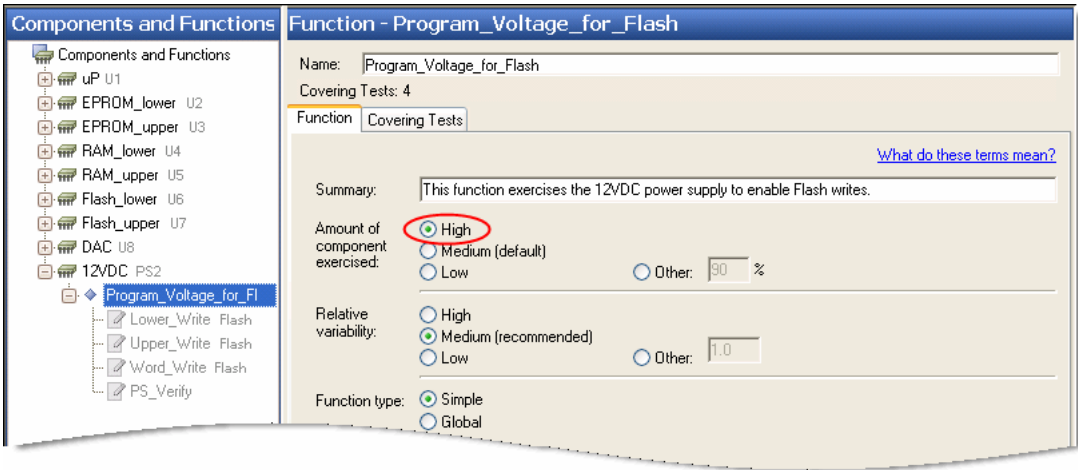


Figure 40 Set the **Amount of component exercised** to *High*

3 Run the analysis again. Figure 41 and Figure 42 show typical analysis results.

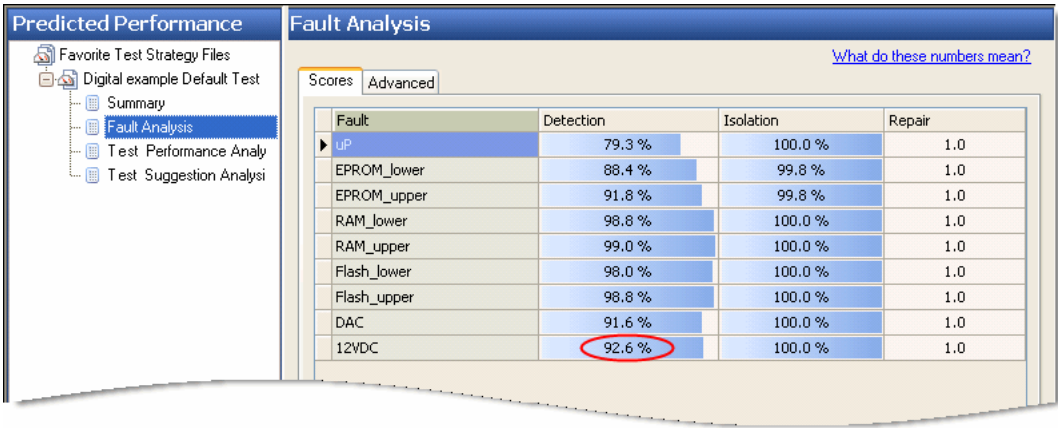


Figure 41 Improved Detection score for 12VDC

5 Predicted Performance

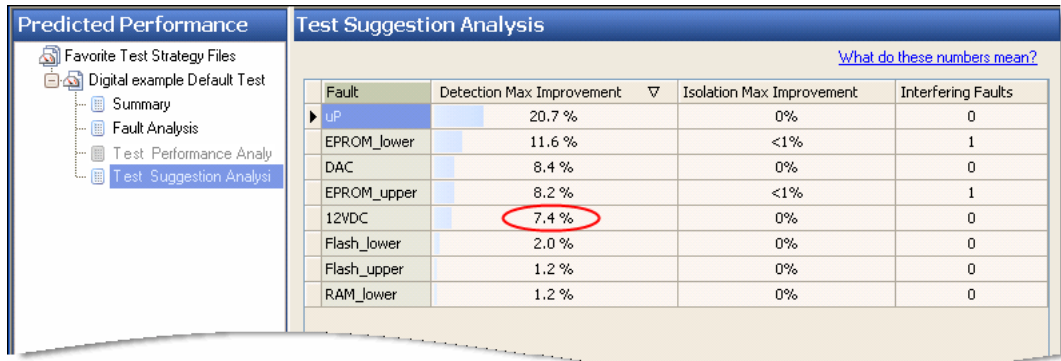


Figure 42 12VDC is no longer the top candidate for Detection Max. Improvement

- 4 You can create an HTML predicted performance report for the selected test strategy syndrome by clicking

 Create Predicted Performance Report

Tip Remember that your model must reflect reality.

If you add tests or extend existing tests in your model, you must implement these tests or extensions in your test suite. It's okay to try adding tests to your model first and then using Fault Detective's analysis to determine whether the results are worth the effort of the additional implementation, but ultimately you must implement the tests to obtain any benefit from the test suggestions.

Using Analysis to Optimize the Test Suite

You can use Fault Detective's Test Performance Analysis to help you optimize your test suite. By eliminating tests that provide marginal contribution to your test suite's detection and isolation performance, you can speed up the running of your test suite and minimize the cost of equipment and fixtures. For example, after a DUT has been in production for some time and the design has been proven, it is often possible to eliminate some of the production tests to speed production. This procedure shows how the analysis can help you decide which tests can be removed without severely impacting test suite performance.

- 1 Click **File > Open > Model**. Navigate to the *Digital example.fdm* file located in:

<drive>:\Program Files\Agilent\Fault Detective\Fault Detective 4.7\Samples\Models\Digital

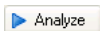
Click **Open** to open the model.

- 2 Click the predicted performance button



. Click the *Base Test Strategy* in the explorer pane. By default, the base test strategy has all tests and components selected.

- 3 By default, Test Performance Analysis is excluded (turned off). This is because it can take a long time, especially if your model is large. Turn Test Performance Analysis on by clicking **Tools > Options...** and selecting the check box labeled **Include test performance analysis** (on the Predicted Performance tab of the Options dialog box). Click **OK**.

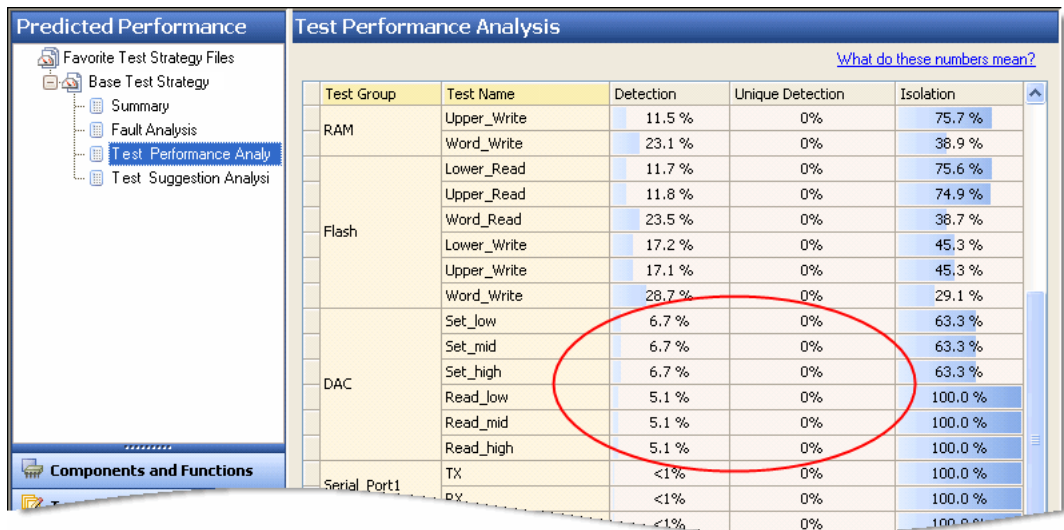
- 4 Run the analysis by clicking  .

- 5 There are a number of tests relating to the DAC functionality. Can some of these tests be eliminated without adversely affecting the test suite?

Click **Test Performance Analysis** in the explorer pane.

5 Predicted Performance

Notice that the DAC tests have fairly low detection scores and 0% unique detection. (Unique detection is the portion of a test's detection performance that can only be achieved by including this test in the test strategy, and cannot be achieved by a combination of other tests.) We can use the predicted performance analysis to determine how the test suite performs without some of these tests.



Predicted Performance

Favorite Test Strategy Files

- Base Test Strategy
- Summary
- Fault Analysis
- Test Performance Analysis**
- Test Suggestion Analysis

Test Performance Analysis

[What do these numbers mean?](#)

Test Group	Test Name	Detection	Unique Detection	Isolation
RAM	Upper_Write	11.5 %	0%	75.7 %
	Word_Write	23.1 %	0%	38.9 %
Flash	Lower_Read	11.7 %	0%	75.6 %
	Upper_Read	11.8 %	0%	74.9 %
	Word_Read	23.5 %	0%	38.7 %
	Lower_Write	17.2 %	0%	45.3 %
	Upper_Write	17.1 %	0%	45.3 %
DAC	Word_Write	28.7 %	0%	29.1 %
	Set_low	6.7 %	0%	63.3 %
	Set_mid	6.7 %	0%	63.3 %
	Set_high	6.7 %	0%	63.3 %
	Read_low	5.1 %	0%	100.0 %
	Read_mid	5.1 %	0%	100.0 %
Serial_Port1	TX	<1%	0%	100.0 %
	RX	<1%	0%	100.0 %

Components and Functions

Figure 43 The DAC tests have fairly low Detection scores and 0% Unique Detection

- Click the *Base Test Strategy* in the explorer pane. Click the **Selected Tests** tab. Scroll down if necessary and click in the column to the right of the *Set_low*, *Set_mid*, *Read_low*, and *Read_mid* tests until the **x** disappears. These tests will no longer be used in the analysis—we are now relying on the *Set_high* and *Read_high* tests to test the DAC.

Test Strategy

Selected Tests

Selected Components

Select the tests to be included in the analysis of this test strategy.

Group	Name	Base Test ...	
Flash	Lower_Write	x	
"	Upper_Write	x	
"	Word_Write	x	
DAC	Set_low		
"	Set_mid		
"	Set_high	x	
"	Read_low		
"	Read_mid		
"	Read_high	x	
Serial_Port1	TX	x	
		x	

Figure 44 DAC Set_low, Set_mid, Read_low, and Read_mid tests are no longer part of the analysis

- 7 Run the analysis. Compare the analysis summary results with the two tests removed (Figure 45 below) to those with all tests selected (Figure 36 on page 95). The overall performance has changed very little.

5 Predicted Performance

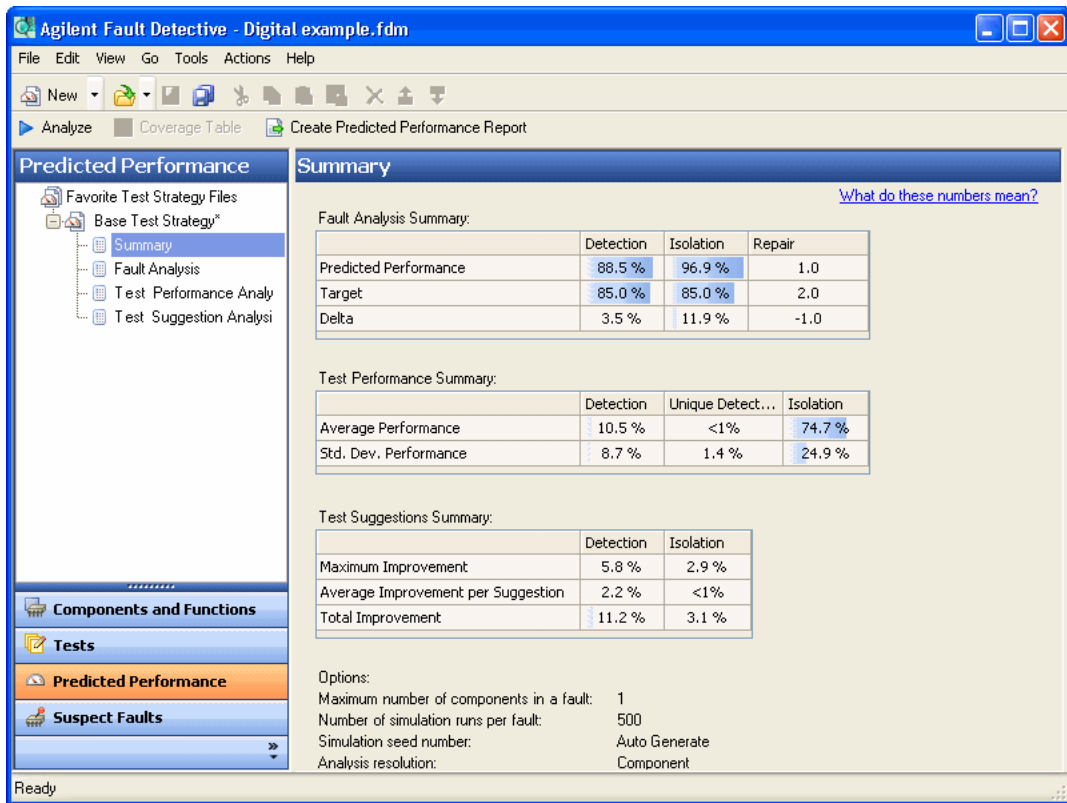


Figure 45 Typical Predicted Performance Summary without the *Set_low*, *Set_mid*, *Read_low*, and *Read_mid* tests

- 8 Compare the test performance analysis results with the two tests removed (Figure 46 below) to those with all tests selected (Figure 43 on page 102). The detection performance has changed very little but the unique detection has increased substantially. Since these two tests are uniquely detecting DAC faults, they should not be removed from the test suite.

Predicted Performance		Test Performance Analysis		
Favorite Test Strategy Files Base Test Strategy" Summary Fault Analysis Test Performance Analy Test Suggestion Analysis		What do these numbers mean?		
Test Group	Test Name	Detection	Unique Detection	Isolation
RAM	Lower_Read	11.7 %	0%	75.4 %
	Upper_Read	11.5 %	0%	75.2 %
	Word_Read	23.2 %	0%	38.7 %
	Lower_Write	11.7 %	0%	75.4 %
	Upper_Write	11.5 %	0%	75.9 %
Flash	Word_Write	23.2 %	0%	38.8 %
	Lower_Read	11.7 %	0%	75.4 %
	Upper_Read	11.8 %	0%	75.8 %
	Word_Read	23.5 %	0%	38.8 %
	Lower_Write	17.1 %	0%	45.7 %
DAC	Upper_Write	16.8 %	0%	45.6 %
	Word_Write	28.7 %	0%	29.3 %
	Set_high	6.7 %	5.0 %	62.4 %
	Read_high	5.1 %	5.1 %	100.0 %
	TX	1.0 %	0%	100.0 %
		1.0 %	0%	100.0 %

Figure 46 Test Performance Analysis results with the *Set_low*, *Set_mid*, *Read_low*, and *Read_mid* tests removed

What's in the Online Help?

Here are some of the topics in the Fault Detective online help relating to analysis (search for *analysis* or *predicted performance*):

- Analysis to Improve the Test Suite
- Analysis to Improve the Model
- About Fault Isolation Scores
- About Test Strategies
- About Test Suggestions
- Reading the Predicted Performance Report



6 Modeling with Microsoft Excel

Using Microsoft Excel with Fault Detective	108
Creating a New Model Using the Excel Template	109
Exporting a Model to Excel	111
Importing a Model from Excel	114
Entering and Editing Model Information in Excel	116
Export Options	120
What's in the Online Help?	123

This chapter describes how to create Fault Detective models in Microsoft Excel and how to export/import Fault Detective models to/from Excel.

Fault Detective supports Excel 2002 (Excel Version 10) and later. Note that if you are using Excel 2007 or later, you must save your workbook in Excel 97-2003 Workbook (.xls) format, using **Save As....**



Using Microsoft Excel with Fault Detective

Fault Detective has full import/export compatibility with Microsoft Excel. You can use Excel along with Fault Detective to:

- **Define models**—capture component, function and test information
- **Validate models**—automatically verify that rows, columns, and cells are properly ordered and syntactically correct
- **Define and validate coverage**—the Excel template makes it easy to look for coverage patterns

Fault Detective has these Excel-related features:

- Ability to import partial or complete models from an Excel worksheet (.xls) file into Fault Detective
- Ability to export your model from Fault Detective to an .xls file
- Ability to fully define components, functions, tests and their relationships and related fields (such as relative failure rate and coverage)

NOTE

Any model developed in any version of Fault Detective can be opened in Fault Detective 4.7. However, if you add component cost data to a model (either in the Fault Detective 4.7 application or in an Excel worksheet), and you save that information in your model file or worksheet, you will not be able to open that model in an earlier version of Fault Detective. You can choose to save your model in backward compatibility mode (that is, Fault Detective 4.5 format). If you do so, however, you will lose the component cost data.

Creating a New Model Using the Excel Template

An Excel template is provided with Fault Detective. This template contains predefined rows and columns that help you to enter your model information in the correct order.

NOTE

- Instead of starting with the Excel template, you may want to start with a subset or your DUT's bill of materials (without capacitors and resistors, for example), format it to fit the Excel template, and then import it into Excel. Refer to *Formatting Model Data for Import Into Fault Detective* in the Fault Detective online help for more information.
- You may also find it easier to create a partial model in Fault Detective, and then export the model to Excel. This generates an Excel spreadsheet containing some of your model's components, subcomponents, functions, and tests. This is often a better starting point for modeling in Excel than starting with the empty Excel template. See ["Exporting a Model to Excel"](#) on page 111 for details.

- 1 Make a copy of the Fault Detective Excel template. The template is located in:

<drive>:\Program Files\Agilent\Fault Detective\Fault Detective 4.7\Samples\ModelTemplate.xls

- 2 Rename the copy appropriately for your model and save it to your working folder (outside of the ...*\Program Files* folder). Note that if you are using Excel 2007 or later, you must save your workbook in Excel 97-2003 Workbook (.xls) format, using **Save As....**
- 3 Double-click the copy to open it in Excel. [Figure 47](#) shows the Fault Detective Excel template as it appears in Excel.

	A	B	C	H	J	K	L	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
1																			
5	Component Information				Function Information			Group (Test)	MyTest1	MyTest2	MyTest3	MyTest4	MyTest5	MyTest6	MyTest7	MyTest8	MyTest9	None	
6	Group	Name (Component)	Schematic Reference	Name (Subcomponent)	Name (Function)	Type	Amount Exercised	Name (Test)											
7																			
8	Enter Component and Function Information Here																		
9																			
10																			
11																			
12																			
13																			
14																			
15																			
16																			
17																			
18																			
19																			

Figure 47 An Excel template is provided with Fault Detective

- 4 You can now work with the model as described in “Entering and Editing Model Information in Excel” on page 116.

Tips

Use the provided Fault Detective Excel template (or export to the template as described in “Exporting a Model to Excel” on page 111). DO NOT begin with a standard Excel worksheet. When importing a worksheet, Fault Detective depends on named ranges in the worksheet to correctly interpret the data. To ensure that you have the correct named ranges, you must begin with the Fault Detective Excel template.

What are those cells with the “#” in front of the name?

These cells are included in the provided empty worksheets to give you a range of cells to start with; they can be deleted or changed. Any name with a leading “#” will not be parsed, nor will it be imported. Note that the “#” character is not allowed in Fault Detective names.

Exporting a Model to Excel

You can export an existing Fault Detective model to an Excel file. The export produces an Excel file having the same features as the template described previously in this chapter.

- 1 In Fault Detective, click **File > Open > Model**. Navigate to the *Digital example.fdm* file located in:

<drive>:\Program Files\Agilent\Fault Detective\Fault Detective 4.7\Samples\Models\Digital

Click **OK** to open the model.

- 2 Export the model from Fault Detective to Excel by clicking **Tools > Export model to Excel...**. Click **Next** to begin the export process.
- 3 When you encounter the Export Options dialog box, you must decide whether you want to model with **One test group per Excel worksheet** or **Multiple test groups per Excel worksheet** (see “[Export Options](#)” on page 120 for details).

For this tutorial, select **One test group per Excel worksheet**. Refer to “[Export Options](#)” on page 120 for more information and examples of the two export options.

- 4 Follow the instructions in the export wizard to export and save the model to an Excel file. The last step in the wizard allows you to automatically open the Excel file after export.
- 5 If you did not automatically open the Excel file, navigate to the saved Excel file and double-click the file name to open it in Excel. You will see the worksheet shown in [Figure 48](#). Components and functions are displayed in rows, and test groups and tests in columns. An *x* at the intersection of a row and column indicates coverage. The worksheets for the test group appear as tabs near the bottom of the screen. Notice that the one test not in a test group is placed in a worksheet entitled *None*.

	A	B	C	H	J	K	L	Q	R	S
1										
5	Component Information				Function Information			Group (Test)	[None]	
6	Group	Name (Component)	Schematic Reference	Name (Subcomponent)	Name (Function)	Type	Amount Exercised	Name (Test)	CPU Verify	
7										
8	[None]	uP	U1	Lower_Bus	uP->Lower	Simple	High	0		
9	[None]	uP	U1	Lower_Bus	uP<-Lower	Simple	High	0		
10	[None]	uP	U1	Upper_Bus	uP->Upper	Simple	High	0		
11	[None]	uP	U1	Upper_Bus	uP<-Upper	Simple	High	0		
12	[None]	uP	U1	CPU	uP_CPU	Simple	High	1	x	
13	[None]	uP	U1	Memory	uP_Memo	Simple	High	0		
14	[None]	uP	U1	Serial_Port	uP_Serial	Simple	Medium	0		
15	[None]	uP	U1	Serial_Port	uP_Serial	Simple	Medium	0		
16	[None]	EPROM	U2	Lower_Bus	EPROM_L	Simple	High	0		
17	[None]	EPROM	U3	Upper_Bus	EPROM_U	Simple	High	0		
18	[None]	RAM_low	U4	Lower_Bus	RAM_L->u	Simple	High	0		
19	[None]	RAM_low	U4	Lower_Bus	RAM_L<-u	Simple	High	0		
20	[None]	RAM_upper	U5	Upper_Bus	RAM_U->u	Simple	High	0		
21	[None]	RAM_upper	U5	Upper_Bus	RAM_U<-u	Simple	High	0		
22	[None]	Flash_low	U6	Lower_Bus	Flash_L->u	Simple	High	0		
23	[None]	Flash_low	U6	Lower_Bus	Flash_L<-u	Simple	High	0		
24	[None]	Flash_upper	U7	Upper_Bus	Flash_U->u	Simple	High	0		
25	[None]	Flash_upper	U7	Upper_Bus	Flash_U<-u	Simple	High	0		
26	NoTestGroup CPU_Int RAM EPROM RAM Flash DAC Serial_Port1 Serial_Port2									

Figure 48 Digital example.fdm model exported to Excel (first worksheet)

- Click the **RAM** tab to see the worksheet associated with this test group. Excel is particularly useful for showing coverage patterns in the model.
- After exporting the model to Excel, you can work with the model as described in “[Entering and Editing Model Information in Excel](#)” on page 116.

Tips**Does the exported information include all the information in the model file?**

The exported information is sufficient to define tests, functions, components and their relationships. There is additional information in the model file that is not exported to Excel. For example, information about the test strategies used in the Predicted Performance area of the application is not exported.

Creating more worksheets

The last worksheet (right-most tab) in the exported file contains your component information and dummy tests. You can duplicate this worksheet to create more test groups.

Importing a Model from Excel

- 1 In Fault Detective, click **File > Close All** to close the open model.

NOTE

You do not have to close a model to import from Excel. The model is being closed in this step so you can see the effects of the model import. You can, for example, import new information from Excel into an existing model. See the Tips below for details.

- 2 Click **Tools > Import model from Excel...**
- 3 Follow the instructions in the import wizard to navigate to, and open, the Excel file.
- 4 The import wizard parses the Excel file and checks the file for problems, such as syntax errors. After parsing, a summary is displayed.

You may see some warning messages that problems were discovered in the import. Upon import, Fault Detective attempts to repair any syntactical problems it finds.

For example, an *undefined* function that has a value for relative variability causes a warning on import and is marked as a problem. In this case, to retain the relative variability information, Fault Detective changes the function type to *defined*.

Refer to the Fault Detective online help for more information on import problems and how to resolve them.

- 5 Click **Finish** to complete the import operation.

Tips Syntax checking

Fault Detective performs syntax checking whenever you import a model. When warnings or errors are found, they are reported and you have the option to cancel the import.

- **A warning** indicates that Fault Detective has made modifications to prevent the loss of data. Do not be alarmed by a large number of warnings—most are probably the import algorithm setting default values. Before completing the import operation, you can view any warning by clicking the **Warning Detail** button. After importing, warnings are displayed as problems in the center pane. These warnings will be erased when you save and re-open the model or the next time you export to Excel.
- **An error** is something that Fault Detective cannot automatically modify and repair. If you decide to import a model containing errors, those errors are saved and left unchanged. These errors will be re-exported when you export the model. This gives you the capability to import/export partial models during model development.

Fault Detective prevents loss of data.

On import, Fault Detective will make modifications to prevent the loss of data. For example, if you have a function group defined containing only an *undefined* function, the function is changed to *defined* so as not to lose the function group information.

Last entry wins.

Since the Excel spreadsheet will be divided across three different Fault Detective trees, some information, such as relative failure rate, is repeated in the spreadsheet. If the repetitive information is different, upon import, the last entry is kept, and any differences are noted with warnings.

What happens when I import Excel data into a model that is already populated (not empty)?

- Fault Detective adds the imported model data to the existing model. An element that has the same name as an existing element will assume the imported element's definition. Element properties (such as relative failure rate and summary) will assume the imported values.
- To import entirely new model elements into the model, you should make those elements unique. Use unique component and test names or put these elements into uniquely named component groups or test groups.

Entering and Editing Model Information in Excel

Whether you are starting from the Fault Detective Excel template or have exported your model to an Excel file, you can use the following methods to work with the model information in Excel:

- Decide whether you want to model with **One test group per Excel worksheet** or **Multiple test groups per worksheet** (see “Export Options” on page 120 for details).
- To add component groups or components, click a row in the components and functions area (Figure 47 on page 110) and select **Insert > Rows** from the Excel menu. Enter the applicable component group name (if used), component name, subcomponent name, and function in each row.

As a minimum, each row needs a component name or a global function name (see the online help for details). The other cells will assume default values when imported into Fault Detective. Rows without component or global function names are ignored.

- To add test groups or tests, click a column in the test names area (Figure 47 on page 110) and select **Insert > Columns** from the Excel menu. Enter the applicable test group name (if used) and test name(s) in each column.

As a minimum, you need only to enter the test name. The other cells will assume default values when imported into Fault Detective.

- *Copy and paste* is a very useful feature in Excel. Copy and paste entire rows or columns to create additional component groups/components and test groups/tests.
- Notice that some of the tables in Fault Detective are very similar to the information in your Excel spreadsheet. You can copy and paste from Fault Detective to your Excel spreadsheet.
- To add coverage, type an *x* in the intersection of the applicable function and test.

- To create more worksheets, right-click on the worksheet tab that you want to copy and use the Excel *Move and Copy* feature.
- Position the mouse on any column header to see summary details for the column, such as acceptable values and default values.

The screenshot shows an Excel spreadsheet with a data table. The table has columns for Group, Name (Component), Name (Subcomponent), Name (Function), Type, and Amount Exercised. A tooltip is displayed over the 'Amount Exercised' column header, providing details about the data type and constraints for that column.

	A	B	H	I	J	K	Q	R	S	T	U	V	W	X	Y
6	Component Information			Function Information											
7	Group	Name (Component)	Name (Subcomponent)	Name (Function)	Type	Amount Exercised									
9	none	C1	none	F1	<Simple>	High	2	x	x						
10	none	C1	none	< High >	<Undefined>	High	2	x		x					
11	none	C2	Sc1	F2	<Distribute>	High	1		x						
12	none	C2	Sc1	< High >	<Undefined>	High	0								
13	none	C2	Sc1	< Medium	<Undefined>	Medium	1			x					
14	none	C2	Sc2	F2	<Distribute>	High	1		x						
15	none	C3	none	< High >	<Undefined>	High	1	x							

Figure 49 Position the mouse on a column header to see details about the column

- Sorting and filtering—use the drop-down menus in each column for predefined sorting and filtering choices. Or, click **Custom** to customize the filtering algorithm.

	Group	Name (Component)	Name (Subcomponent)	Name (Function)	Type	Amount
7						
8						
9	none	U1	Sort Ascending	uP->Lower	<Simple>	High
10	none	U1	Sort Descending	uP<-Lower	<Simple>	High
11	none	U1	(All)	uP->Upper	<Simple>	High
12	none	U1	(Top 10...)	uP<-Upper	<Simple>	High
13	none	U1	(Custom...)	uP<-Upper	<Simple>	High
14	none	U1	Converter	uP_CPU	<Simple>	High
15	none	U1	Memory	uP_Memory	<Simple>	High
16	none	U1	Serial_Port1	uP_Serial	<Simple>	Medium
17	none	U1	Serial_Port2	uP_Serial	<Simple>	Medium
18	none	U2	uP_Lower_Bus	EPROM_L	<Simple>	High
19	none	U3	EPROM_Upper_Bus	uP_Upper_Bus	<Simple>	High

Figure 50 Sorting and filtering columns in Excel

Tips **Import and export often.**

To avoid extensive rework, import your model into Fault Detective and export it back to Excel often. This allows you to take advantage of Fault Detective's error checking to find problems early. In addition, during the export cycle, Fault Detective fills in the default values for most blank fields.

Which rows and columns MUST be filled in?

As a minimum, you must enter component names and test names. All other empty cells will assume default values upon import into Fault Detective. You may get a series of warnings upon import, indicating the assumptions that Fault Detective has made about the values of empty cells.

Can I work on a model in Excel and Fault Detective simultaneously?

Changes made in Excel will not be immediately visible in Fault Detective, and vice versa. Use Fault Detective to export to Excel, then use Excel to import to Fault Detective. If you make changes in both applications simultaneously without importing/exporting in between, you will not be able to merge those changes, and you will lose information.

Why are some of the rows and columns missing?

By default, some of the less frequently used model information is hidden. You can determine hidden columns or rows by looking for missing letters or numbers identifying those columns or rows. For example, in the Excel template, rows 2 through 4 and columns E through H are hidden.

- To display hidden columns, select the columns on either side of the hidden ones (in this case columns D and I), right-click and select **Unhide**.
- Similarly, to unhide rows 2 through 4, select rows 1 and 5, right-click and select **Unhide**.
- To hide a column or row, select the row(s) or column(s), right-click and select **Hide**. Do not hide row 1 or column A.

If possible, print spreadsheets on a large-format printer or plotter—this makes it much easier to see all the spreadsheet information.

Export Options

When exporting a model to Excel, you can choose from these two export options:

- One test group per Excel worksheet
- Multiple test groups per Excel worksheet

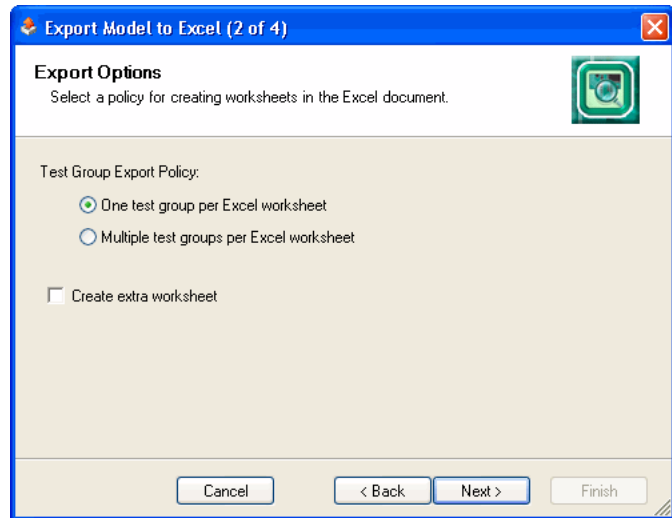


Figure 51 Export Options dialog box

One Test Group per Excel Worksheet

This export option creates an Excel worksheet for each test group in your model. The name for each worksheet is taken from the name of the first test group placed on that worksheet.

This option allows you to easily distinguish between test groups in your model. For example, you could use this option when you have multiple engineers, each with

responsibility over a different set of test groups. With this option, each engineer can work on selected worksheets and ignore the other worksheets.

Figure 52 shows the *Digital example.fdm* model exported using **One test group per Excel worksheet**. Each test group in the model is now a worksheet (shown as tabs on the bottom).

NOTE

If you are not using test groups or have independent tests outside of test groups, those tests will be placed in a worksheet labeled *NoTestGroup*. For example, in Figure 52 the tab labeled *NoTestGroup* contains a single test that was not part of a test group.

	A	B	C	H	J	K	L	Q	R	S	T	U	V
1													
5	Component Information				Function Information			Group (Test)	CF	CF	CF	CPU_Int_RAM	
		Name (Component)	Schematic Reference	Name (Subcomponent)	Name (Function)	Type	Amount Exercised	Name (Test)	Int_RAM_Check_00	Int_RAM_Check_55	Int_RAM_Check_AA	Int_RAM_Check_FF	
6	Group												
7													
8	[None]	uP	U1	Lower_Bus	uP->Lower	Simple	High	0					
9	[None]	uP	U1	Lower_Bus	uP<-Lower	Simple	High	0					
10	[None]	uP	U1	Upper_Bus	uP->Upper	Simple	High	0					
11	[None]	uP	U1	Upper_Bus	uP<-Upper	Simple	High	0					
12	[None]	uP	U1	CPU	uP_CPU	Simple	High	0					
13	[None]	uP	U1	Memory	uP_Memory	Simple	High	4	x	x	x	x	
14	[None]	uP	U1	Serial_Port	uP_Serial	Simple	Medium	0					
15	[None]	uP	U1	Serial_Port	uP_Serial	Simple	Medium	0					
16	[None]	EPROM_k	U2	Lower_Bus	EPROM_L	Simple	High	0					
17	[None]	EPROM_u	U3	Upper_Bus	EPROM_U	Simple	High	0					
18	[None]	RAM_low	U4	Lower_Bus	RAM_L->u	Simple	High	0					
19	[None]	RAM_low	U4	Lower_Bus	RAM_L<-u	Simple	High	0					
20	[None]	RAM_upper	U5	Upper_Bus	RAM_U->u	Simple	High	0					
21	[None]	RAM_upper	U5	Upper_Bus	RAM_U<-u	Simple	High	0					
22	[None]	Flash_low	U6	Lower_Bus	Flash_L->u	Simple	High	0					
NoTestGroup CPU_Int_RAM EPROM RAM Flash DAC Serial_Port1 Serial_Port2													

Figure 52 The **One test group per Excel worksheet** method puts each test group on a separate worksheet

Multiple Test Groups per Excel Worksheet

This export option places multiple test groups (totaling up to 200 tests) per Excel worksheet. Test groups are exported in the order listed in the Fault Detective model. This export option allows you to see all of your model (up to 200 tests) on a single tab. If your model has more than 200 tests, additional worksheets are created to accommodate the remaining test groups.

Figure 53 shows the *Digital example.fdm* model exported using **Multiple test groups per Excel worksheet**.

	A	B	C	H	J	K	L	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH		
1																											
5	Component Information				Function Information			Group (Test)	Int_CPU_Verify	Int_CPU_Check_00	Int_RAM_Check_55	Int_RAM_Check_AA	Int_RAM_Check_FF	Lower_Read	Upper_Read	Word_Read	Lower_Read	Upper_Read	Word_Read	Lower_Write	Upper_Write	Word_Write	Lower_Read	Upper_Read	Word_Read		
6	Group	Name (Component)	Schematic Reference	Name (Subcomponent)	Name (Function)	Type	Amount Exercised	Name (Test)																			
7																											
8	[None]	uP	U1	Lower_Bus	uP->Lower	Simple	High	7												x		x					
9	[None]	uP	U1	Lower_Bus	uP<-Lower	Simple	High	6						x		x	x						x		x		
10	[None]	uP	U1	Upper_Bus	uP->Upper	Simple	High	4													x	x					
11	[None]	uP	U1	Upper_Bus	uP<-Upper	Simple	High	6							x	x		x	x					x	x		
12	[None]	uP	U1	CPU	uP_CPU	Simple	High	1	x																		
13	[None]	uP	U1	Memory	uP_Memory	Simple	High	4		x	x	x	x														
14	[None]	uP	U1	Serial_Port	uP_Serial	Simple	Medium	2																			
15	[None]	uP	U1	Serial_Port	uP_Serial	Simple	Medium	2																			
16	[None]	EPROM_k	U2	Lower_Bus	EPROM_L	Simple	High	2						x		x											
17	[None]	EPROM_u	U3	Upper_Bus	EPROM_U	Simple	High	2							x	x											
18	[None]	RAM_low	U4	Lower_Bus	RAM_L->u	Simple	High	2									x			x							
19	[None]	RAM_low	U4	Lower_Bus	RAM_L<-u	Simple	High	2													x		x				
20	[None]	RAM_upper	U5	Upper_Bus	RAM_U->u	Simple	High	2									x	x									
21	[None]	RAM_upper	U5	Upper_Bus	RAM_U<-u	Simple	High	2														x	x				
22	[None]	Flash_low	U6	Lower_Bus	Flash_L->u	Simple	High	2															x		x		
CPU_Verify/																											

Figure 53 The **Multiple test groups per Excel worksheet** method puts up to 200 tests per worksheet

What's in the Online Help?

Here are some of the topics in the Fault Detective online help relating to modeling with Excel (search for *Excel*):

- Frequently Asked Questions about Excel Modeling
- Working with a Model in Excel
- Microsoft Excel Features to Help You Edit Your Model
- Formatting Model Data for Import Into Fault Detective



7 Analog Modeling

About the Analog Model...	126
Analog vs. Digital Modeling	128
Selecting Components	129
Use of Subcomponents	130
Identifying Functions	131
Global Functions	134
Tests	136

This chapter describes how modeling an analog DUT in Fault Detective differs from modeling a digital DUT. This chapter builds upon the lessons learned in the [Chapter 3](#), “Digital Modeling Tutorial”. You should perform the tutorial in [Chapter 3](#) before using this analog modeling chapter.



About the Analog Model...

This chapter references the *RF Example.fdm* model. This model file can be found in:

<drive>:\Program Files\Agilent\Fault Detective\Fault Detective 4.7\Samples\Models\RF

Figure 54 is the block diagram of the DUT used in the analog model. This DUT is the input circuitry (front end) for an RF analyzer.

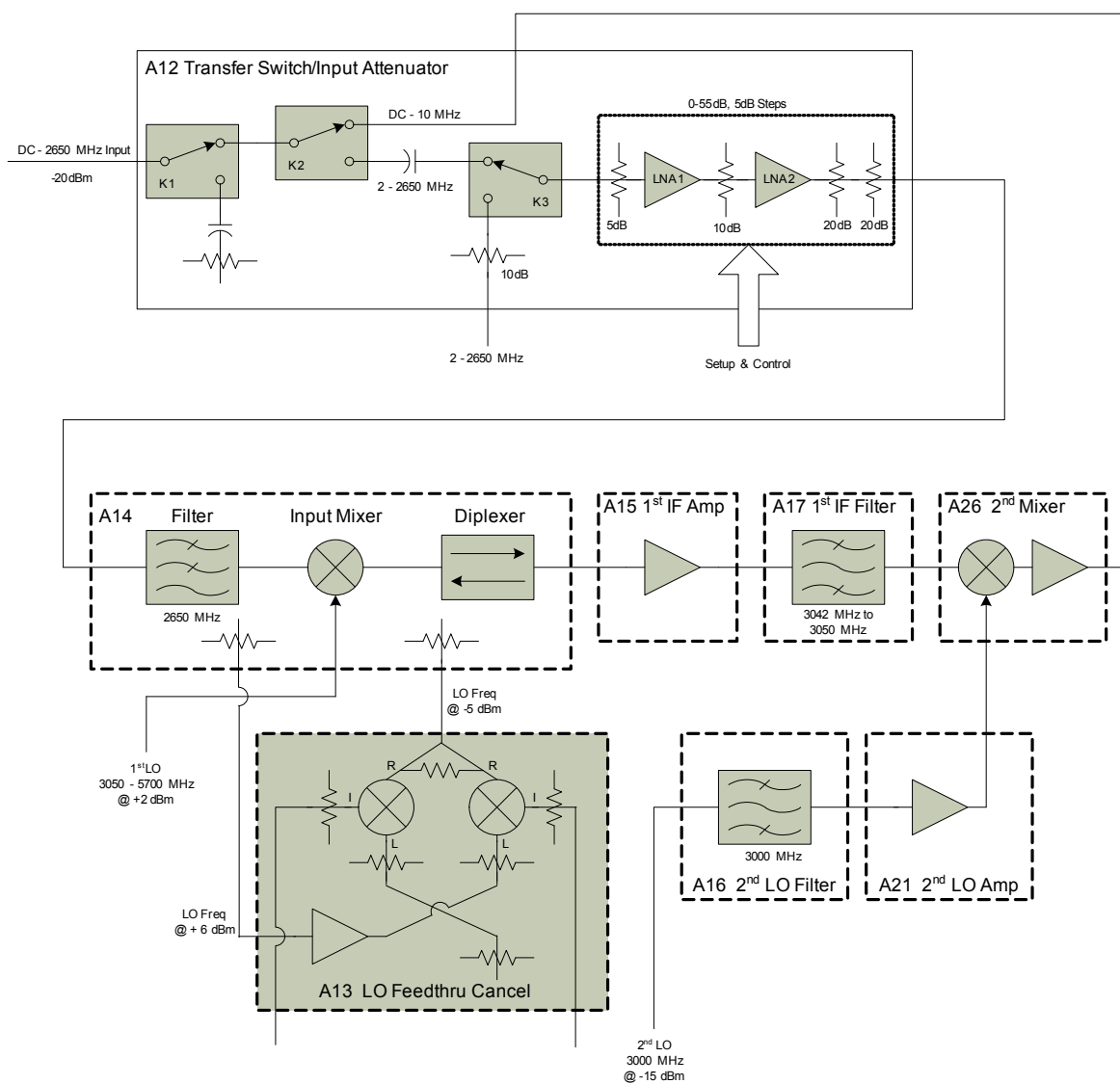


Figure 54 Analog DUT input circuitry block diagram

Analog vs. Digital Modeling

In general, there are a number of similarities between digital and analog modeling, such as modeling at the block diagram level and selecting components. There are also some key differences. The differences relate to the use of subcomponents, identifying functions, and how tests are performed.

Selecting Components

Component selection in the analog model is very similar to that in the digital model. Components relate to the block diagram level just as they did in the digital model. Typical analog components are block diagram elements, such as amplifiers, filters, mixers, and switches. Since the analog DUT has a number of printed circuit (PC) assemblies, each component name in the model is prefaced with its PC assembly number.

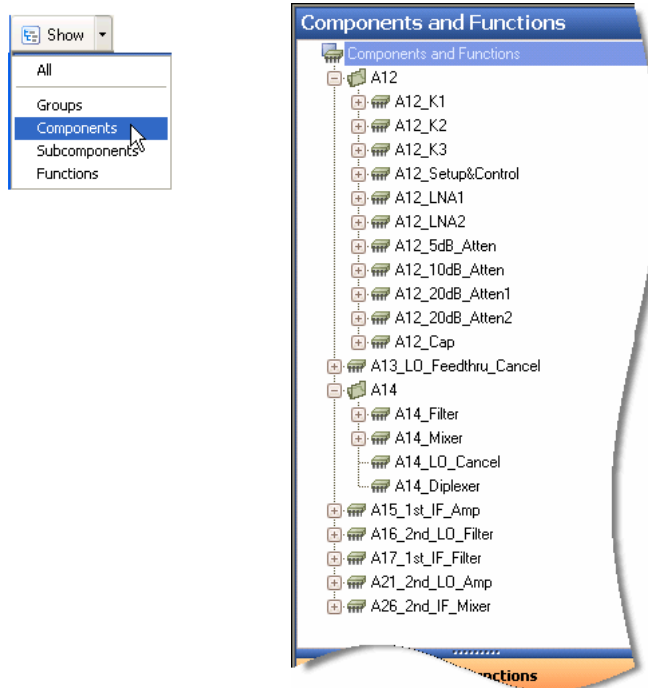


Figure 55 Components in the analog model

Use of Subcomponents

There are very few subcomponents in this analog model. In complex digital integrated circuits, sections that perform specific operations can easily be identified as subcomponents. Analog circuits tend to have a single purpose, such as amplification, filtering or mixing.

Each printed circuit (PC) assembly in the analog DUT performs just one or two operations. Had this DUT been built with more complex analog integrated circuits, subcomponents would have been more applicable.

Identifying Functions

In a model for a digital DUT, a function is typically an *operation* (or capability) of the component/subcomponent. For example, a bus transceiver will typically have read and write functions. In a model for an analog DUT, a function is typically a *characteristic*. For example, a filter will typically have flatness and insertion loss characteristics, which are modeled as functions.

The functions performed by analog components can sometimes be a little less obvious than their digital counterparts. For example, an amplifier's primary operation is gain. However, amplifiers do more to a signal than simply provide gain. While a digital multiplier only multiplies, an analog amplifier has gain, distortion, flatness, and noise characteristics. Similarly, an analog filter has passband gain, passband shape, stopband rejection, and center frequency characteristics.

In short, analog/RF components seem to have more characteristics that describe the imperfections in their behavior than they have describing their primary intended purpose. In addition, analog components are likely to be performing multiple functions and exhibiting multiple characteristics simultaneously. Some examples of the functions for the various components in the analog model are listed below.

Typical Amplifier Functions

- Gain—the primary operation of an amplifier
- Flatness—when the amplifier is used over a wide bandwidth
- Distortion—especially in applications where the overall gain is fairly high
- Noise—in applications where noise is critical

Typical Filter Functions

- Passband—nominal gain at the center of the passband
- Stopband—nominal rejection outside the passband
- Shape—for filters whose passband shape is critical and likely to be tested (such as narrow IF filters, where IF flatness significantly affects measurement accuracy)
- Center frequency—for tunable filters

Typical Switch Functions

- Activate, deactivate—for a two-position switch
- Normal, Cal—for a switch that selects between the normal signal path and a calibration path
- More complex switches simply have more positions to identify and name

Typical Attenuator Functions

- Activate—for when the attenuator is used
- Deactivate—for when the attenuator is not used
- Flatness—in wideband applications, where frequency rolloff is a concern

Typical Mixer Functions

- Gain, Flatness, Distortion

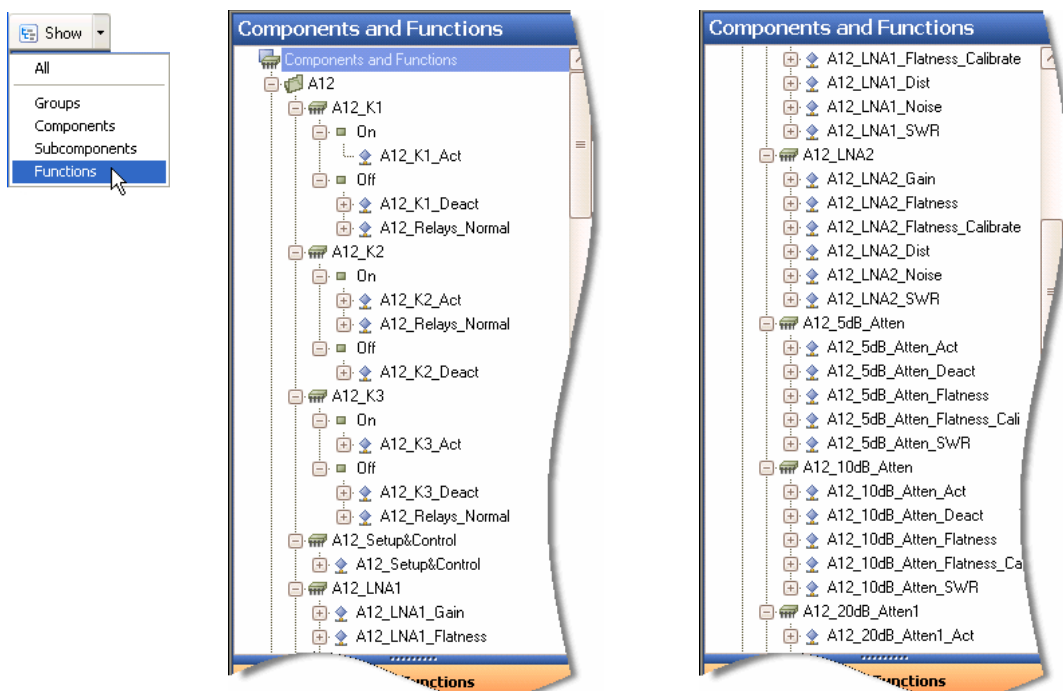



Figure 56 Some of the functions in the analog model

Global Functions

Global functions are functions that, unlike simple or undefined functions, are not "owned" by any component or subcomponent. A global function typically exercises more than one subcomponent or component.

Global functions are typically used in models of analog DUTs, where a testable characteristic of the DUT involves several components/subcomponents. They may also be needed in certain models of digital DUTs when a group of components/subcomponents acts together to perform a testable function.

Global functions are shown (as all functions are) in Fault Detective's Components and Functions view. Each function is shown in the Components and Functions explorer pane under its exercised component/subcomponent; if a global function exercises multiple components/subcomponents, it is shown under each. (If you edit a global function's properties in this view, the changes will be applied under all exercised components/subcomponents.) A global function is indicated

by this icon  .

There is also a Global Functions view available for you to create and manage these functions (refer to the Fault Detective online help for more information on global functions).

In the analog model, *A12_Relays_Normal* is an example of a global function. This function is used to exercise relays *A12_K1*, *A12_K2*, and *A12_K3*.

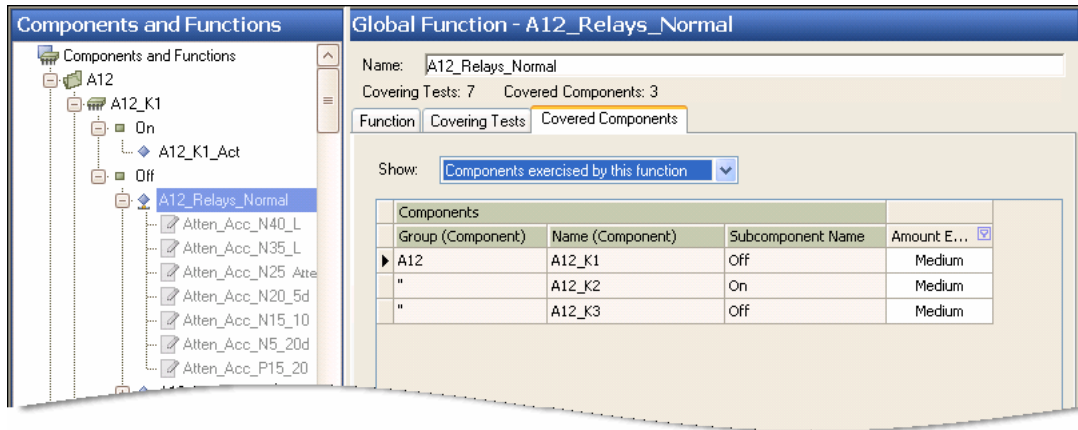


Figure 57 Global functions exercise multiple components in an identical manner

Tests

The tests in the analog model are the tests used in manufacturing to verify the performance of the DUT against a tightly controlled set of specifications. The tests measure signal characteristics, such as harmonic distortion, absolute accuracy, IF flatness, noise floor, and phase noise. The tests are not written with hardware diagnostics in mind. That is, there are no tests such as *Verify A12* or *Verify A14*.

Since tests are designed to test specifications and not the hardware paths being used, specific knowledge of how the tests are exercising the product's functions is necessary.

Figure 58 shows the RF analyzer's A12 Transfer Switch/Input Attenuator PC assembly. This assembly is used to describe how the tests use functions to exercise components. The A12 assembly contains:

- Several switches—used to select a path that bypasses the input section entirely, or to select a calibration signal input
- Amplifiers and attenuators—used to select the amplitude range over which the analyzer will make measurements

The amplitude range of the instrument is determined by the combination of attenuators and amplifiers used on the A12 assembly. LNA1 has a gain of 10dB; LNA2 has a gain of 15dB. The most sensitive range utilizes both amplifiers, with no attenuation. The second range utilizes both amplifiers with a 5dB attenuation, and so on.

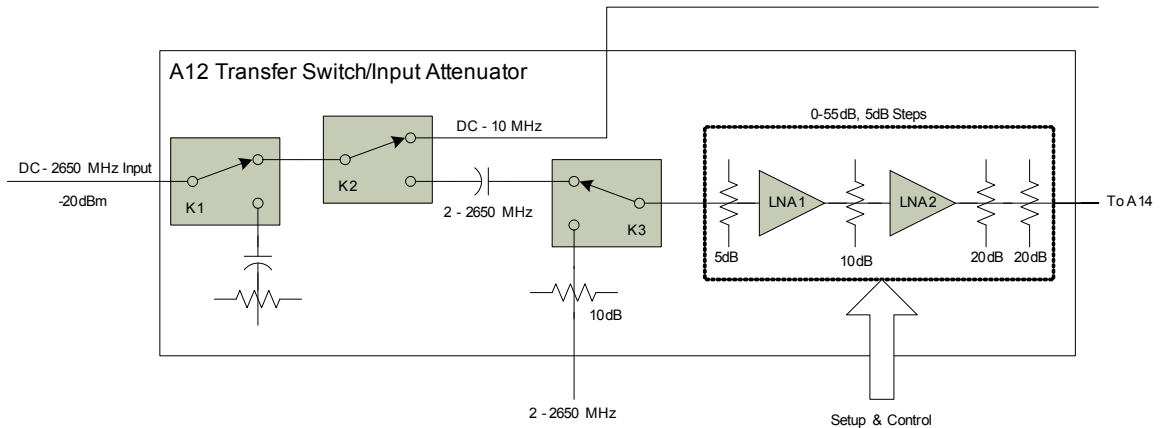


Figure 58 A12 Transfer Switch/Input Attenuator PC assembly

Differentiating Faults

Typically, an absolute accuracy test for an RF analyzer passes a signal through the entire front end, and verifies the measurement accuracy on all amplitude ranges and across all frequencies. The absolute accuracy test is actually a combined gain/flatness test, using multiple amplifier and attenuator settings. Because the test uses so many different signal paths, if the test passes, it says much about the inner workings of the hardware. If the test fails, however, it provides little insight into which circuit element(s) may have failed.

To better support the Fault Detective model, the absolute accuracy test is divided into multiple individual tests. Data points are selected from the ranges that utilized the amplifiers and attenuators individually, and the tests are named accordingly. For example, the *Abs_accuracy_N40_LNA2* test is based on the data points taken on the -40dBm range, using the LNA2 amplifier with

no attenuators; the *Abs_accuracy_N35_LNA1* test uses data from the -35dBm range, and uses only the LNA1 amplifier. By using multiple absolute accuracy tests, if a single test fails, Fault Detective can determine that a particular amplifier or attenuator is at fault.

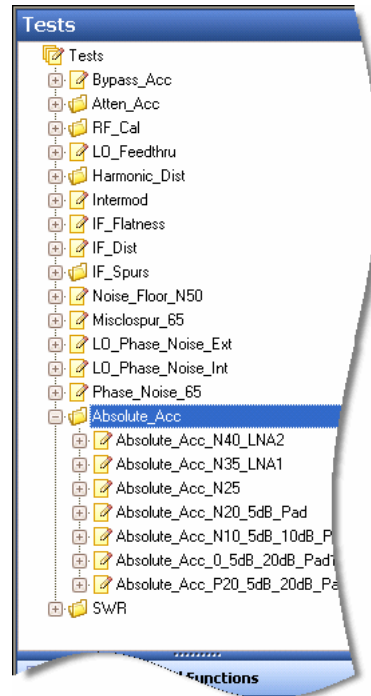


Figure 59 Absolute accuracy tests

Testing Components to Different Specifications

Unlike tests on digital circuitry, sometimes the same analog component is tested against multiple specifications. For example, the A14 mixer is tested against tighter flatness and

distortion specifications when no amplifiers or attenuators are in the signal path. This is because the characteristics of the mixer alone can be more tightly maintained.

In the analog model, two functions are used to differentiate between the tighter and looser specifications for the *A14_mixer*. The *A14_mixer_flatness* function is used by tests involving the looser specifications. The *A14_mixer_flatness_noamps* function is used by tests involving the tighter specification, such as the *Abs_accuracy_N25* test (-25dBm range, no amplifiers or attenuators).

This same concept is used to test calibrated vs. uncalibrated components. Throughout the model you will see duplicate functions, such as *A12_LNA1_Flatness* and *A12_LNA1_Flatness_Calibrated*.

Tips Design tests to isolate signal paths/components.

Manufacturing tests for analog devices are often written for testing conformance to specifications—not necessarily to diagnose component failures. Break down large tests into smaller tests to allow isolation of the various paths and components within the DUT.

Naming model elements

Descriptive naming of components and functions may be even more important in analog modeling. Since analog devices tend to have multiple amplifiers, filters, mixers and so on, include PC assembly, component type, and so on, to easily distinguish model elements later on.

Using global functions

You should model a multiple-component global function only if you are certain that the exercised components/subcomponents in question are truly used together in the same way by each test that uses them. If you find that your global function has high relative variability (often passes in some tests while failing in others), consider whether this function would be more appropriately represented as two different functions; in any case, always ensure that your model represents reality.



Glossary

Active

Currently selected to be acted upon. For example, the active component is the component currently selected in the Components and Functions explorer pane, and is the component whose details are displayed in the Component detail pane. There is only one loaded model at a time; it is the active model, and its file name is indicated in the title bar of the Fault Detective window.

Agilent Fault Detective

A model-based software tool that diagnoses the most-likely failing component or subassembly in a device under test. Once you have created a model, Fault Detective uses test results (independent of any language or test executive) to identify faulty components.

Ambiguity Group

A set of faults that can exhibit the same syndrome.

See also [“Interfering Fault”](#).

Ambiguous Syndrome

A syndrome that can be exhibited or explained by more than one fault; this makes it difficult to diagnose and repair the responsible fault on the first attempt. Also called a shared syndrome.



Analysis

The process whereby the Fault Detective analysis engine takes a test strategy and predicted performance options as input and generates predicted performance information about your model. Analysis is performed in the Predicted Performance view of the Fault Detective application.

Analysis Engine

The software module that takes a test strategy and predicted performance options as input and generates predicted performance information about your model. This software module is the heart of Agilent Fault Detective's predicted performance feature.

Component

In Fault Detective modeling, an abstraction of one or more physical parts of the device under test (DUT).

A component may represent:

- A single physical part. This type of component is commonly used when the DUT has large complex parts (such as ASICs).
- Many physical parts. This type of component is commonly used when a logical subsystem of the DUT (for example, a power supply section) has many discrete parts, or when a test uses a function that exercises several indistinguishable physical parts (for example a set of bypass capacitors).
- An entire assembly of parts, such as a complete PC board.

Component Cost

The replacement cost of a component in the device under test.

Component Element

A component or subcomponent.

Contributing Score

The score of a plausible fault in a diagnosis obtained using the Tie Breaker 2 algorithm. Tie Breaker 2 provides all the possible combinations of subcomponents and functions that may have caused a component to fail; each of these combinations, or plausible faults, is given a relative score, which is the contributing score. Contributing scores are displayed on the Advanced tab of the Suspect Faults detail view.

Contribution

Contribution is the product of a component's likely frequency of failure (based on its relative failure rate) and its detection loss, isolation loss, or repair score.

Detection Loss Contribution = (1 - [Detection Score]) · [Frequency of Failure]

Isolation Contribution = (1 - [Isolation Score]) · [Frequency of Failure]

Repair Contribution = [Repair Score] · [Frequency of Failure]

Coverage

The relationship between tests and components, subcomponents or functions. The words "coverage" and "cover" are used in several ways in the field of product testing and troubleshooting; in common use, coverage can be:

- The degree to which a part is exercised by a test
- The degree to which a device under test is exercised by a test suite
- The percentage of possible faults that can be found by a test or a test suite

In Fault Detective, tests cover components by using functions which exercise the components or their subcomponents. This is equivalent to saying that the test provides coverage of the component/subcomponent in question. Fault Detective uses the terms *fault detection* and *fault isolation* to describe the

quantitative ability of a test suite to find a fault in the device under test, and to determine which component is faulty, respectively.

Detail Pane

The center pane of the Fault Detective window, which typically shows details of the item that is selected in the explorer pane at the left.

Detection

See [“Fault Detection”](#).

Detection Loss

A measure of the degree to which a test strategy or individual test falls short of perfect detection performance. Detection loss is equal to (one minus detection score).

Detection Score

A quantitative measure of a test strategy's or individual test's ability to detect faults in a device under test. The detection score is equal to (number of faults detected / number of possible faults) and is expressed as a percentage.

Device Under Test (DUT)

The device, product, board, assembly, or system being tested and modeled.

Diagnosis

The process whereby the Fault Detective diagnosis engine takes a syndrome and suspect faults options as input and generates diagnosis results in the form of a list of scored suspect faults. Diagnosis is performed in the Suspect Faults view of the Fault Detective application.

Diagnosis Engine

The software module that takes a syndrome and suspect faults options as input and generates diagnosis results (a list of scored suspect faults) as output. This software module is the heart of Agilent Fault Detective's diagnosis feature.

Diagnosis Score

A relative score, associated with a suspect fault, that shows how well this suspect fault explains the syndrome being diagnosed. The total of all the scores for all the suspect faults is 100.

Diagnostic Run

One run of Fault Detective's diagnosis engine.

DUT

Device under test.

Escape

A fault that escaped detection. An escape generally results in failed tests farther down the line, or product failures in the customer's hands.

Exercise

In Fault Detective, to make use of a component or subcomponent: Tests use functions, which exercise components. When you model a function in Fault Detective, you specify the amount of the component/subcomponent that the function exercises (High, Medium, or Low).

Explorer Pane

The left pane of the Fault Detective window, which shows a tree view of either model elements, syndromes and suspect faults, or test strategies and predicted performance data.

Fail

A test result that indicates this test has failed. In Fault Detective, a test uses functions, which exercise components; when one or more of the functions used by a test operates incorrectly, the test should fail.

Fail-Only Function

A function that is only used by failing tests (relative to a particular syndrome). A fail-only function is denoted by FO: in the fault detail string.

Fault

In Fault Detective, a set of one or more simultaneously defective components in a device under test. A fault is a possible explanation for all of the failing tests.

Fault Ambiguity

A measure of poor isolation (the inability to isolate a fault from other interfering faults). Fault ambiguity is equal to (one minus fault isolation).

Fault Condition

The condition of a device under test (DUT) when it is defective. A fault condition causes one or more syndromes. The fault condition of a DUT can be explained by one of the suspect faults generated by diagnosis of these syndromes. The fault that actually caused the fault condition is called the root cause.

Fault Detail

A string that represents detailed information about a suspect fault. The fault detail consists of a list of components/subcomponents and functions that may be implicated in the fault. See *About Fault Details* in the online help for a full discussion of the fault detail string format.

Fault Detection

A measure of the ability of a test suite, test, or test strategy to detect faults. Detection can be reported at the test suite level and at the fault level. This is also sometimes called coverage or fault detection coverage.

Fault Isolation

The ability of a test suite, test, or test strategy to isolate a fault from other suspect faults: that is, to eliminate potential faults from consideration to reduce the list of suspect faults. This is also sometimes called resolution or diagnosability.

F0:

A prefix given to a fail-only function in a fault detail string.


Function

An attribute or behavior of a component, subcomponent, or set of components/subcomponents. In Fault Detective, a function is said to exercise a component or subcomponent. Tests use functions in order to cover components.

In the model of a digital device under test (DUT), a function is typically a function of the component; for example, a bus transceiver will typically have read and write functions.

In the model of an analog DUT, a function is often a characteristic; for example, a filter will typically have flatness and insertion loss characteristics, which are modeled as functions.

Global Function

A function that, unlike a simple or undefined function, is not "owned" by any component or subcomponent. A global function typically exercises more than one subcomponent or component. Global functions are represented in Fault Detective by this icon .

Incomplete Function

A global function that exercises no components/subcomponents.

Incomplete Test

A test that uses no functions. Typically, this occurs because the modeler has not finished defining the model, although it may also signify an omission in the test suite itself. If you believe your model to be complete but it still contains incomplete tests, you should investigate those tests to determine whether they are correctly modeled and whether the test suite is sufficient.

Interfering Fault

When you use Fault Detective for analysis, you often focus on a fault of interest (the fault that has actually caused, or can cause, a failure of one or more tests). There may be a number of additional faults that interfere with this focus on the fault of interest, because they exhibit the same syndrome as the fault of interest and so could also be responsible for the observed failure or failures. These additional faults are called interfering faults. Interfering faults are signs of low isolation for the fault of interest.

See also [“Ambiguous Syndrome”](#).

Isolation

See [“Fault Isolation”](#).

Isolation Loss

A measure of the degree to which a test strategy or individual test falls short of perfect isolation performance. Isolation loss is equal to (one minus isolation score).

Model

A logical representation of the device under test and the test suite, which is used by Agilent Fault Detective to diagnose faults in the device under test and to predict the performance of the test suite in detecting and isolating faults. A Fault Detective model is stored in an *.fdm* file.

Model Development Environment

The part of the Agilent Fault Detective application that is used for model creation. Click the **Components and Functions** or **Tests** navigation button in the lower left corner of the Fault Detective window to go to the model development environment.

Model Element

An element of the Fault Detective model that represents an element of the device under test or of the test suite. Model elements include components, subcomponents, functions, tests, component groups, test groups, and function groups.

Model File

A Fault Detective model file has a *.fdm* suffix. It contains:

- Model element definitions
- Coverage information
- References to favorite syndrome files and favorite test strategy files

Multiple-Component Function

An attribute or behavior (i.e., a function) of multiple components and/or subcomponents. A multiple-component function is always global. It has a name, description, notes, and relative variability, and a list of exercised components and/or subcomponents, including the amount of each component/subcomponent that is exercised by the function.

No-Fail Syndrome

A syndrome consisting entirely of passing test results. Any faults attributed to this no-fail syndrome are undetected by the test suite, and consequently are likely to move to the next stage of production or to customers. Also called an escape syndrome.

Pass

A test result indicating that this test has passed. In Fault Detective, a test uses functions, which exercise components; if all the functions used by a test operate correctly, the test should pass.

Pass-Fail Function

A function that is used by both passing and failing tests (relative to a particular syndrome). A pass-fail function is denoted by PF: in the fault detail string.

PF:

A prefix given to a pass-fail function in a fault detail string.

Physical Part

Electronic components, such as ASICs, processors, and op amps; and non-electrical components, such as wires and connectors. A device under test (DUT) consists of physical parts.

Plausible Fault

The Tie Breaker 2 scoring algorithm identifies a set of suspect faults, each of which can be the result of multiple plausible faults. Plausible faults are different combinations of component elements and functions that can explain the failing tests. Each suspect fault may be explained by any one of its plausible faults. (The original and Tie Breaker 1 scoring algorithms do not support this feature.) Plausible

faults are displayed in the Detail column of the Advanced tab in the Suspect Faults view when Tie Breaker 2 scoring is in effect.

See also [“Scoring Algorithm”](#).

Predicted Performance

The area of the Fault Detective application where you can define test strategies and use the analysis process to predict the performance of your test strategy based on simulation of many fault conditions.

Predicted Performance Options

The set of user-configurable parameters that determine the behavior and performance of the analysis engine. Examples of these parameters include number of simulation runs per fault, component or subcomponent analysis resolution, and maximum number of components in a fault to be simulated. To set these options, click **Tools > Options** and then the Predicted Performance tab.

Predicted Syndrome

The predicted syndrome for a given fault is determined by assuming that, for each component in that fault, all tests that cover the component will fail.

Recommended Repair Order

A suggested order in which to repair suspect faults. The recommended repair order is calculated as (Repair score / Repair cost). It is intended to optimize the cost of the repair vs. the likelihood of resolving the failure.

Relative Failure Rate

In Fault Detective, a user-configurable property of a modeled component. The relative failure rate indicates the expected rate of failure of the physical part(s) represented by the

component. The use of absolute failure rates in Fault Detective models is discouraged; the use of relative failure rates (high, medium, and low) produces better results.

Relative Variability

A measure of how much a function varies from test to test. For example, assume Function A is verified to pass 999 out of 1000 times and Function B is verified to pass 9,999 out of 10,000 times. In this example, Function A has ten times the relative variability of Function B.

Repair Score

A measure of fault isolation.

- **For a fault condition:** The predicted average number of attempts that will need to be made to repair a defective device under test. The repair score is the weighted average of the number of repair attempts predicted to be made before and including the repair of the root cause.
- **For a test strategy:** The frequency-weighted average of the repair scores for the fault conditions associated with all possible syndromes.

Root Cause

The one suspect fault that is the actual cause of a fault condition.

Scoring Algorithm

The method used by Fault Detective's diagnosis engine to detect and score (rank) suspect faults. Fault Detective supports three scoring algorithms:

- The Original scoring method is supported only for backward compatibility. Agilent has deprecated this method.

- The Tie Breaker 1 scoring method is the preferred scoring method in most cases. It produces a better diagnosis than the Original scoring algorithm because it takes more factors into account and fits many more models than Tie Breaker 2 scoring.
- Tie Breaker 2 scoring differs in its handling of subcomponents and is better in certain cases; see *When to Use Tie Breaker 2 Scoring* in the online help for details.

You can choose a scoring algorithm in the Options dialog box. The default algorithm is Tie Breaker 1.


Selected

1. Active; refers to the item or items in a Fault Detective view that will be acted upon. (This is normally the highlighted item or items; you select an item by clicking it.)
2. As used by the analysis process and defined in the Predicted Performance view:
 - **Selected component:** A component that is defined in the active model and is used in a particular test strategy.
 - **Selected test:** A test that is defined in the active model and is used in a particular test strategy.

Shared Syndrome

A syndrome whose root cause may be any one of a group of interfering faults.

Simple Function

A defined function that exercises one component element; that is, an attribute or behavior of a single component or subcomponent. A simple function is considered to be owned by the component/subcomponent that it exercises. Most functions are simple functions. Simple functions are represented in Fault Detective by this icon .

Skip

A test result that indicates this test is not to be considered in the diagnosis process. Test results that are skipped are treated as if the test was not run. The skip result is used if the test actually has not been run, or if the test result is not dependable and should not be considered as part of the diagnostic process.

Subcomponent

A named subset of a component. You can define subcomponents in your model, and they will appear in the Components and Functions explorer pane underneath their parent components.

Although subcomponents add to the complexity of a model, their use typically results in more accurate models. You should use subcomponents if:

- The parent component represents a complex part or a combination of several complex parts,
or
- The parent component can be broken into logical subcomponents,
or
- Small combinations of subcomponents are tested in different tests.

Submodel

- When used in the context of Fault Detective analysis, a submodel is a carefully constructed subset of the model. Submodels, defined as test strategies, can be saved and used in future analysis activity.
- When used in the context of Fault Detective model development, a submodel is a casually constructed subset of the model used (typically temporarily) to focus on a portion of the model. In this context, submodels are combined into the model under development and are not saved separately.

Suspect Fault

One of the faults reported by the Fault Detective diagnosis process as a possible explanation for the syndrome being diagnosed. Any of the faults in the list of suspect faults can explain the failing tests in the syndrome; each suspect fault has a diagnosis score that indicates the relative likelihood of that fault being the actual problem (root cause).

Suspect Faults Options

The set of user-configurable parameters that determine the behavior and performance of the diagnosis engine. Examples of these parameters include the scoring algorithm, maximum number of subcomponents per fault, and diagnosis timeout. Diagnosis results only make sense in the context of the options used to generate those results.

Click **Tools > Options** and select the Suspect Faults tab to set these options.

Syndrome

A set of test results, with each test's result reported as *Pass*, *Fail*, or *Skip*. The syndrome is used as input to the Fault Detective diagnosis process, and can be saved to a syndrome file with the file extension *.tr*.

Target Fault

When doing a Fault Detective analysis, a particular fault is often the center of focus. Such a fault is called the target fault.

See also [“Interfering Fault”](#).

Test

In Fault Detective, a model element representing a real (or proposed) test in a test suite (typically, but not always, the functional test suite). A test uses functions, which exercise components. A modeled test consists of a name, description, notes, and a list of functions that are used by the test.

Test Result

The outcome of a single test. In Fault Detective syndromes, a test result can be *Pass*, *Fail*, or *Skip*.

Test Results File

A Fault Detective file that stores a syndrome. A test results file has a *.tr* suffix.

Test Strategy

A submodel, or subset of the loaded Fault Detective model. A test strategy consists of a set of selected tests and a set of selected components. It can be used to represent the tests and components that are the focus of a given stage in the testing process (e.g., process test, in-circuit test, functional test, repair station test). A test strategy can also be used when comparing and contrasting various approaches to minimizing the number or duration of tests in a test suite while optimizing some outcome, such as fault detection at functional test or fault isolation at the repair station. You can define test strategies in Fault Detective's Predicted Performance view.

Test Suggestion


A new test suggested by Fault Detective's analysis engine to improve fault detection or fault isolation. A test suggestion indicates a set of components whose detection and/or isolation can be improved by implementing a test to cover the components of the given fault while minimizing the coverage of the components that are part of the given fault's interfering faults list.

Test suggestions are displayed in Fault Detective's Test Suggestions view.

Test Suite

The set of tests used to verify a device under test, represented in the Fault Detective model.

Undefined Function

A function automatically created when you load a Fault Detective model from an earlier version of Fault Detective, if that model employed "direct coverage" (i.e., included one or more components or subcomponents which were used by tests, but were not exercised by any function). The displayed name for an undefined function is derived from its coverage value (for example, <High>). Undefined functions are provided for backward compatibility only. You can convert an undefined function into a simple function by giving it a name; you can convert all undefined functions in your model to simple functions by clicking **Tools > Convert Undefined Function....** Undefined functions are represented in Fault Detective by this icon  .

Unique Detection

The portion of a test's detection performance that can only be achieved by including this test in the test strategy, and cannot be achieved by a combination of other tests.

Unused Function

A function that is not used by any tests.

Index

A

about
 diagnosing suspect faults, 82
 predicted performance, 92
 test strategies, 92
 the analog model, 126
 this (digital modeling) tutorial, 40
active defined, 141
add
 bus subcomponents, 52
 coverage, 63
 functions, 55
 more components, 46
 more tests and coverage, 73
 subcomponents, 49
add a component, 43
adding model elements, 26, 31
advantages to using Excel, 111
Agilent Fault Detective defined, 141
Agilent, contacting, 11
algorithm, scoring, defined, 152
algorithms, scoring, 88
ambiguity group defined, 141
ambiguity, fault, defined, 146
ambiguous syndrome defined, 141
amplifier functions, typical, 131
analog model, 126
analog model, tests in, 136
analog modeling vs. digital modeling, 128
analysis
 defined, 142
 engine defined, 142
 predicted performance, running, 94
 using to improve test coverage, 97
 using to optimize the test suite, 101
attenuator functions, typical, 132

B

begin modeling tests, 59
bus subcomponents, adding, 52

C

cause, root, defined, 152
check for incomplete tests, 66
check for unused functions, 65
component defined, 142
component element defined, 142
component, selected, defined, 153
components, 14
 adding more, 46
 selecting, 129
 testing to different specifications, 138
components and functions summary, 26
components and functions view, 25
components, subcomponents and functions, exploring, 27
condition, fault, defined, 146
contacting Agilent, 11
contributing score defined, 143
contribution defined, 143
copying information from a Fault Detective table to Excel, 123
coverage defined, 143
coverage, adding, 63, 73
coverage, viewing, 29
create reports, 78
create, save, close, and re-open the model, 42

creating

 model report, 29, 32, 78
 new model in Excel, 115
 new model using the Excel template, 109
 predicted performance report, 34, 78
 suspect faults report, 38

D

detail pane defined, 143
detail panes, 27
detail, fault, defined, 146
detection
 defined, 144
 fault, defined, 147
 loss defined, 144
 score defined, 144
 unique, defined, 157
development environment, model, defined, 149
device under test (DUT) defined, 144, 145
diagnosing a syndrome, 36
diagnosing suspect faults, 82
diagnosis, 13
 defined, 144
 engine defined, 145
 example, 83
 running, 85
 score defined, 145
diagnostic run defined, 145
differentiating faults, 137
digital modeling vs. analog, 128
DUT (device under test) defined, 144, 145

E

element, component, defined, 142

Index

- element, model, defined, [149](#)
- engine, analysis, defined, [142](#)
- engine, diagnosis, defined, [145](#)
- entering and editing model information in Excel, [116](#)
- environment, model development, defined, [149](#)
- escape defined, [145](#)
- example, diagnosis, [83](#)
- example, one test group per Excel worksheet, [121](#)
- Excel
 - creating a new model, [115](#)
 - entering and editing model information, [116](#)
 - exporting a model to, [111](#)
 - importing a model from, [114](#)
 - multiple test groups per worksheet, [122](#)
 - one test group per worksheet, [120](#)
 - template, creating a new model using, [109](#)
 - using with Fault Detective, [108](#)
- exercise defined, [145](#)
- existing model, exporting to Excel, [111](#)
- explorer pane defined, [145](#)
- exploring components, subcomponents and functions, [27](#)
- exploring tests, [32](#)
- export options, [120](#)
- exporting a model to Excel, [111](#)

F

- fail defined, [146](#)
- fail-only function defined, [146](#)
- failure rate, relative, defined, [151](#)

- fault
 - ambiguity defined, [146](#)
 - condition defined, [146](#)
 - defined, [146](#)
 - detail defined, [146](#)
 - detection defined, [147](#)
 - interfering, defined, [148](#)
 - isolation defined, [147](#)
 - plausible, defined, [150](#)
 - suspect, defined, [155](#)
 - target, defined, [155](#)
- Fault Detective
 - defined, [141](#)
 - installing, [10](#)
 - model development, system requirements for, [8](#)
 - runtime, system requirements for, [9](#)
 - with Excel, [108](#)
- faults, differentiating, [137](#)
- faults, suspect, options defined, [155](#)
- faults, suspect, view, [35](#)
- file, model, defined, [149](#)
- file, test results, defined, [156](#)
- filter functions, typical, [132](#)
- FO: defined, [147](#)
- function
 - defined, [147](#)
 - fail-only, defined, [146](#)
 - global, defined, [147](#)
 - incomplete, defined, [148](#)
 - multiple-component, defined, [149](#)
 - pass-fail, defined, [150](#)
 - simple, defined, [153](#)
 - undefined, [157](#)
 - unused, defined, [157](#)
- functional test suite, [13](#)

- functions, [17](#)
 - adding, [55](#)
 - amplifier, typical, [131](#)
 - and components view, [25](#)
 - attenuator, typical, [132](#)
 - exercise components, [17](#)
 - filter, typical, [132](#)
 - global, [134](#)
 - identifying, [131](#)
 - mixer, typical, [132](#)
 - switch, typical, [132](#)
 - unused, checking for, [65](#)
 - unused, resolving, [66](#)

G

- global function defined, [147](#)
- global functions, [134](#)
- global functions view, [134](#)
- glossary, [141](#)
- group, ambiguity, defined, [141](#)
- guide, task, [24](#)
- guidelines, modeling, [19](#)

H

- help, online, what's in it, [89](#), [106](#), [123](#)
- how Fault Detective diagnoses suspect faults, [83](#)

I

- identifying functions, [131](#)
- importing a model from Excel, [114](#)
- incomplete
 - function defined, [148](#)
 - test defined, [148](#)
 - tests, checking for, [66](#)
 - tests, resolving, [67](#)
- installing Fault Detective, [10](#)
- interfering fault defined, [148](#)
- isolation defined, [148](#)
- isolation loss defined, [148](#)
- isolation, fault, defined, [147](#)
- iteratively improve the model, [76](#)

L

layout, screen, user interface, [22](#)
 loss, detection, defined, [144](#)
 loss, isolation, defined, [148](#)

M

Microsoft Excel, using with Fault Detective, [108](#)
 mixer functions, typical, [132](#)
 model
 adding a component, [43](#)
 analog, [126](#)
 creating, saving, closing, and re-opening, [42](#)
 defined, [149](#)
 development environment defined, [149](#)
 development, system requirements for Fault Detective, [8](#)
 element defined, [149](#)
 elements, adding, [26, 31](#)
 exporting to Excel, [111](#)
 file defined, [149](#)
 importing from Excel, [114](#)
 improving iteratively, [76](#)
 improving using predicted performance, [69](#)
 information, entering and editing in Excel, [116](#)
 new, creating using Excel template, [109](#)
 report, creating, [78](#)
 verifying, [65](#)
 modeling
 analog vs. digital, [128](#)
 guidelines, [19](#)
 overview, [13](#)
 tests, beginning, [59](#)
 multiple test groups per Excel worksheet, [115, 122](#)
 multiple-component function defined, [149](#)

N

new user interface, [22](#)

no-fail syndrome defined, [150](#)

O

one test group per Excel worksheet, [120, 121](#)
 online help, what's in it, [89, 106, 123](#)
 opening view, [23](#)
 options, export, [120](#)
 options, predicted performance, defined, [151](#)
 options, suspect faults, defined, [155](#)
 overview, modeling, [13](#)

P

pane, detail, defined, [143](#)
 pane, explorer, defined, [145](#)
 part, physical, defined, [150](#)
 pass defined, [150](#)
 pass-fail function defined, [150](#)
 performance, predicted, [92](#)
 performance, predicted, defined, [151](#)
 performance, predicted, options, defined, [151](#)
 PF: defined, [150](#)
 physical part defined, [150](#)
 plausible fault defined, [150](#)
 predicted performance, [92](#)
 analysis, [94](#)
 defined, [151](#)
 options defined, [151](#)
 report, creating, [78](#)
 summary, [94](#)
 using to improve the model, [69](#)
 view, [33, 34, 38](#)
 predicted syndrome defined, [151](#)

R

relative failure rate defined, [151](#)
 relative variability defined, [152](#)
 repair score defined, [152](#)

report
 creating model, [29, 32, 78](#)
 creating predicted performance, [34, 78](#)
 creating suspect faults, [38](#)
 reports, creating, [78](#)
 requirements, system, [8](#)
 resolving incomplete tests, [67](#)
 resolving unused functions, [66](#)
 result, test, defined, [156](#)
 results, test, file, defined, [156](#)
 root cause defined, [152](#)
 run, diagnostic, defined, [145](#)
 running, [94](#)
 running the diagnosis, [85](#)
 running the predicted performance analysis, [94](#)
 runtime, system requirements for Fault Detective, [9](#)

S

score
 contributing, defined, [143](#)
 detection, defined, [144](#)
 diagnosis, defined, [145](#)
 repair, defined, [152](#)
 scoring algorithm defined, [152](#)
 scoring algorithms, [88](#)
 screen layout, user interface, [22](#)
 selected component defined, [153](#)
 selected defined, [153](#)
 selected test defined, [153](#)
 selecting components, [129](#)
 shared syndrome defined, [153](#)
 simple function defined, [153](#)
 skip defined, [154](#)
 specifications, testing components to different, [138](#)
 strategies, test, [92](#)
 strategy, test, defined, [156](#)
 subcomponent defined, [154](#)

Index

- subcomponents, [15](#)
 - adding, [49](#)
 - bus, adding, [52](#)
 - use of, [130](#)
- submodel defined, [154](#)
- suggestion, test, defined, [156](#)
- suggestions, test, view, [156](#)
- suite, test, defined, [156](#)
- summary, components and functions, [26](#)
- summary, predicted performance, [94](#)
- summary, tests, [31](#)
- suspect fault defined, [155](#)
- suspect faults
 - diagnosing, [82](#)
 - how Fault Detective diagnoses, [83](#)
 - options defined, [155](#)
 - report, creating, [38](#)
 - view, [35](#)
- switch functions, typical, [132](#)
- syndrome
 - ambiguous, defined, [141](#)
 - defined, [155](#)
 - diagnosing, [36](#)
 - no-fail, defined, [150](#)
 - predicted, defined, [151](#)
 - shared, defined, [153](#)
- system requirements, [8](#)
- system requirements for Fault Detective model development, [8](#)
- system requirements for the Fault Detective runtime, [9](#)

T

- target fault defined, [155](#)
- task guide, [24](#)

- test, [13](#)
 - coverage, using analysis to improve, [97](#)
 - defined, [155](#)
 - group, one per Excel worksheet, [120](#)
 - groups, multiples per Excel worksheet, [122](#)
 - incomplete, defined, [148](#)
 - result defined, [156](#)
 - results file defined, [156](#)
 - selected, defined, [153](#)
 - strategies, [92](#)
 - strategy defined, [156](#)
 - suggestion defined, [156](#)
 - suggestions view, [156](#)
 - suite defined, [156](#)
 - suite, using analysis to optimize, [101](#)
- testing components to different specifications, [138](#)
- tests, [17](#), [136](#)
 - adding, [73](#)
 - exploring, [32](#)
 - in the analog model, [136](#)
 - incomplete, checking for, [66](#)
 - incomplete, resolving, [67](#)
 - modeling, beginning, [59](#)
 - summary, [31](#)
 - use functions, functions exercise components, [17](#)
 - view, [30](#)
- tutorial steps, [42](#)
- typical
 - amplifier functions, [131](#)
 - attenuator functions, [132](#)
 - filter functions, [132](#)
 - mixer functions, [132](#)
 - switch functions, [132](#)

U

- undefined function, [157](#)
- unique detection defined, [157](#)
- unused function defined, [157](#)
- unused functions, checking for, [65](#)
- unused functions, resolving, [66](#)
- use of subcomponents, [130](#)

- use predicted performance to improve the model, [69](#)
- user interface, [22](#)
- using analysis to improve test coverage, [97](#)
- using analysis to optimize the test suite, [101](#)
- using Microsoft Excel with Fault Detective, [108](#)

V

- variability, relative, defined, [152](#)
- verify the model, [65](#)
- view
 - components and functions, [25](#)
 - global functions, [134](#)
 - opening, [23](#)
 - predicted performance, [33](#)
 - suspect faults, [35](#)
 - test suggestions, [156](#)
 - tests, [30](#)
- viewing coverage, [29](#)

W

- what is Agilent Fault Detective?, [12](#)
- what's in the online help, [89](#), [106](#), [123](#)

www.agilent.com

© Agilent Technologies, Inc. 1996-2010

Third Edition, January 2010



Agilent Technologies