Methodizing Functional Test Programs

by Rick Wagner, Digalog Systems

Complete a test project even while the product is being developed? It's possible if you follow the generic approach outlined here.

Often, there is a temptation to start writing test code before there are adequate test and process specifications, leading some to ask, "Why do we not have time to do it right, but we have time to do it over?" While there is a legitimate need to complete test projects quickly, a more accurate and less costly way to achieve the same goal is to start writing the test and process specifications early—even while the product is being developed.

In *An Introduction to Mixed-Signal IC Test and Measurement*, Mark Burns and Gordon Roberts cautioned, "If a test plan is not clearly documented before coding begins, then the test engineer lacks the necessary overview of the test program that allows the tests to fit together efficiently."

Christine Turskey, author of *Test System Design*, also noted, "A vague or ambiguous test description will result in [the test developers] taking longer to deliver a solution as they clarify your requirements, or delivering a solution which is not quite what you require."

The process of writing a functional test program can be streamlined by adopting a standardized approach to the planning. This approach involves identifying the requirements, building consensus, and reducing risk by getting advance approvals. Getting everyone on the same page greatly reduces the variation in the process.

One generic approach includes seven key process steps that have been thoroughly tested and can be readily applied to any application (**Figure 1**). Two of the most critical steps are the test-specification and the test-process description. The test specification, the central document around which the entire functional test is written, outlines how the actual product is to be tested. The process description describes the interface from the tester to the outside world—the operator, the handling equipment, and the information systems.

Continued on page 20

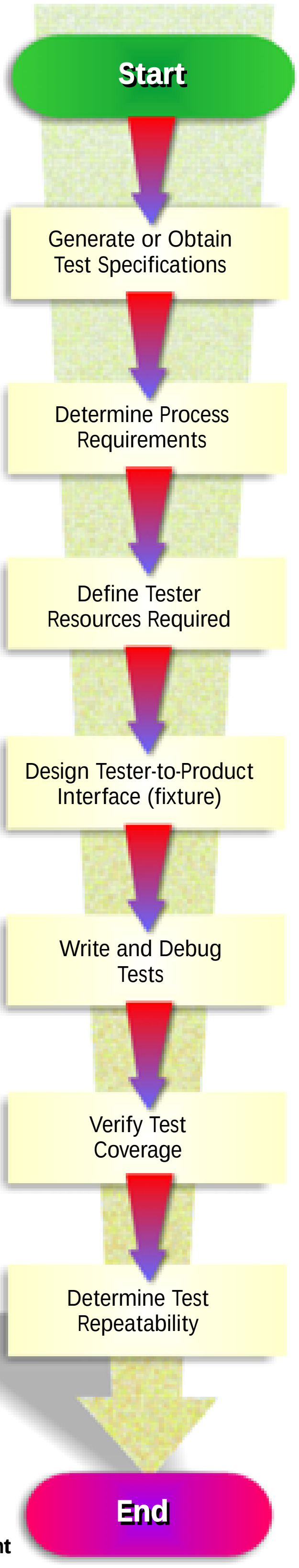

Start
Generate or Obtain Test Specifications
Determine Process Requirements
Define Tester Resources Required
Design Tester-to-Product Interface (fixture)
Write and Debug Tests
Verify Test Coverage
Determine Test Repeatability
End

Figure 1. Seven Key Process Steps for Application Software Development

Test Specification

As a minimum, the test specification contains the high and low test limits as well as an index number that identifies the test. However, much more information typically is required, as shown by the following fields. (Also see the testinfo.mdb database at www.digalogsys.com/mdftp.html.) **Figure 2** represents a completed test-specification record.

• Index—a unique test label used throughout the documentation. Tests related either by board function or logical sequence should be placed within the same major group via a major.minor notation scheme that allows for the insertion/deletion of tests without major renumbering, such as 100.10 Power On Voltage Check, 100.20 Power On Current Check, 100.30 High Product Voltage Current Check, and 100.40 Low Product Voltage Current Check.

• Identification String—a maximum of 32 characters for use in data reporting, such as short test descriptions.

• Test Description—description of the UUT parameter being measured. This consists of the name of the signal, the connector, and the pin. The measurement technique such as trigger voltage settings, rising edge, falling edge, and measurement resolution also should be included. Avoid specifying values covered by the high and low limit fields because the limits would need to be updated in more than one location. Using generic instrument names makes it easier to reapply the test specification.

For example, verify the specified voltage from P1.20 (Vcc) to P1.10 (Gnd) using a voltmeter with a resolution of at least 0.010 V. Place a positive lead to P1.20 and a negative lead to P1.10.

• Conditions—the state of the UUT when the test is being conducted, including the numeric spec values as well as additional conditions needed to hold a state or prevent an undesired side effect, such as open inputs causing oscillation. Conditions can be selected from a lookup table with preset information or added independently.

• Test Units—test value units.

• Failure Action—action the program should take if a particular test fails; for example, abort, continue, or move to some other point in the program.

• Coverage—the sections of the product that are covered; helpful with engineering reviews and diagnostics of failed product.

• Application—identification of specific testing environments such as temperature; for example, where elevated temperatures have different limits that must be noted.

The process description can be greatly simplified by the proper choice of the test-development language.

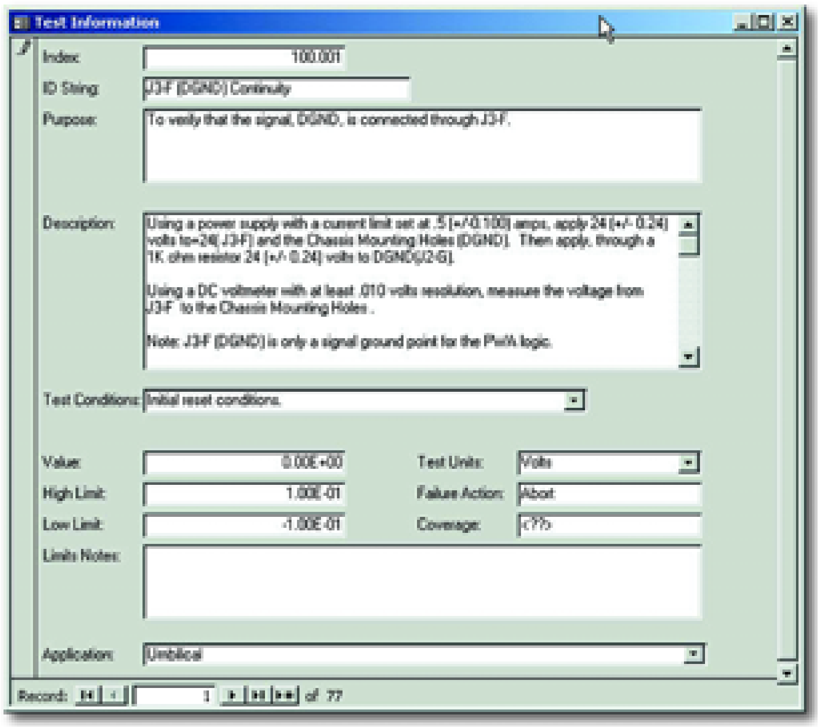
One tool that automates this process is a Microsoft Access™ database, which allows records to be added easily. Its standard format also makes changes easier to track.

Access offers many helpful features. The cut-and-paste capability reduces time and chance of error. Unique index labels automatically are verified, and password-protected access can be established. Field validation prevents you from omitting information and allows the designer to set information requirements. Lookup tables offer consistency in data entry by limiting the choices.

Once data is entered, you can extract it for use in the test program via an assortment of subroutines available for the Microsoft Windows environment. Other development environments such as Visual C or LabWindows/CVI are just as easy to implement. Most significantly, the programmer does not distribute the source code to facilitate limit adjustments on the production floor. The operator or technician can adjust the limits within the database, virtually guaranteeing test-source code integrity because it does not reside on every machine.

To capture specifications, *Building a Successful Board Test Strategy*

Continued on page 22



Index:	100.001
ID String:	J3-F (DGND) Continuity
Purpose:	To verify that the signal, DGND, is connected through J3-F.
Description:	Using a power supply with a current limit set at .5 (+/-0.100) amps, apply 24 (+/- 0.24) volts to~24(J3-F) and the Chassis Mounting Holes (DGND). Then apply, through a 1K ohm resistor 24 (+/- 0.24) volts to DGND(J2-G). Using a DC voltmeter with at least .010 volts resolution, measure the voltage from J3-F to the Chassis Mounting Holes . Note: J3-F (DGND) is only a signal ground point for the PWA logic.
Test Conditions:	Initial reset conditions.
Value:	0.00E+00
High Limit:	1.00E-01
Low Limit:	-1.00E-01
Test Units:	Volts
Failure Action:	Abort
Coverage:	<??>
Limits Notes:	
Application:	Umbilical

Record: 1 of 77

Figure 2. Test Specification Record

by Stephen F. Scheiber and *Test System Design* recommend examining the schematics and block diagrams and placing inputs and outputs on a matrix. This gives the test developer an idea of the complexity of the task at hand.

The applicable stimulus for inputs and limits should be entered. Outputs should be categorized by type, digital or analog, and by the loading required under normal operating conditions. An Excel spreadsheet is a good tool for organizing this data.

Once output and input types are entered, the developer can segment the tests into functional blocks. Typically, the UUT must be set to a state in which some of the inputs not being tested are set to a known condition for a certain block of tests to prevent unknown events from interfering.

Similarly, some outputs may need to be set to known values for the device to operate properly. Once these conditions are met, the testing of the functional block proceeds. Sometimes it is necessary to test a group of outputs such as an address or data bus, but a specific evaluation of data requires a single test specification. For that reason, test specifications need a unique number to identify them.

As developers write the test specification for each output, they can specify the setup condition in the database as well as high limits, low limits, and expected reading. Entering an expected reading can be of great use off-line if any instrument drivers can be disabled; that is, the program can be run without errors being returned from the drivers.

Evaluation subroutines then can be programmed to use the expected data instead of the data passed from the program. Entering values to cause failures also will allow the developer to examine that program flow. A flag can be used to abort when a critical test has failed so testing does not continue. Other program enhancements could include a field for directing program flow or indicating an alternate printout for analog vs. digital tests.

A key purpose for the test specification is to communicate to other departments what is being tested and how. The following guidelines help facilitate the communications:

• Use terms that are as generic as possible.

• Use generic standards such as voltmeter instead of amplitude measurement system (or other tester resource name).

• Specify the resolution and precision of the measuring instrument and stimulus being used.

• Use only standard units such as volts, amps, or ohms.

• Use appropriate nomenclature; for example, tester resource may not mean anything to someone from the quality department.

It's important to note test resolution, or you will not be able to determine if tester resources will be sufficient. If tester resources have not been defined, this value will help specify them. *Test System Design* cautioned: "Accuracy, in conjunction with the degree of complexity of your test requirements, will always be a cost driver for any test system." Overspecifying test resolution also will present a problem in quality reviews.

The test specification also serves to get the design-engineering, quality-engineering, and production departments to approve the specification before writing one line of code. Being able to conduct inquiries and generate reports further improves communications among the team members.

Having the engineering and quality departments review and ultimately approve the test specification greatly reduces the chance of rewriting test code. There can be product changes at any time, but having the test spec in a standardized form makes it easy for anyone to update.

Production engineering benefits by understanding how the test will be performed. Things like test time can be better estimated by adding a field

CIRCLE 58

and summing all the records. Production engineering, in particular, further benefits from a second, standardized document that defines the process requirements.

Process Requirements

A process-requirements document describes the environment required for the test program and tester. Significant time and effort must be put into this document or the impact on the test program can be devastating. The same environment often is maintained, allowing the document to be reused and potentially become a standard. If no company or site standard exists, simple things like the format of pass-and-fail print slips may need to be detailed.

The most significant component is the test executive. It is the code that forms the structural framework for individual test code. It is the program that calls or directs all these separate functions or subroutines. The test executive also interfaces with external entities such as operators, device handlers, printers, information systems, and bar-code readers.

This section of code may be purchased or developed internally. Prewritten test executives for various program development languages are readily available. Either way, failure to clearly define requirements in advance almost guarantees rewriting or heavily modifying the executive code, which can be a huge and expensive undertaking.

Continued on page 24

CIRCLE 30

CIRCLE 44

Many companies standardize the test-executive program because its development is so time-intensive—standardizing things such as failure tickets, information system communications, and the user interface, which reduces training time for operators.

The user interface encompasses several key aspects that must be evaluated before coding starts: a log-in feature; on-screen and printed information in languages other than English; pre-test, mid-test, and post-test operator instructions; and special graphics such as an adjustment screen or oscilloscope window. Other items to specify in the process description include safety and ergonomics, methods of starting the test, test abort conditions, data reporting, the test language, and auxiliary equipment interfaces.

Equipment external to the tester can easily be forgotten, potentially resulting in unexpected code changes. Test stations often use handling equipment to position the UUT and connect it to the tester, which requires code for communications to the handlers.

The process description can be greatly simplified by the proper choice of the test-development language. Of the many different test-development languages to choose from, the best one is the one that production engineering, design engineering, and quality engineering agree to use. Having the entire engineering staff supporting the same language in advance of writing the test program—and documenting that—may save rewriting later.

Conclusion

While your particular test scenario may involve other factors, this checklist has been applied to many test applications industry-wide and represents the fundamental issues that must be addressed. Thorough planning in advance of writing any code, especially with respect to the test specification and process description, will help organize the test developer. Gaining consensus of other departments on the contents of these documents will further minimize risks.

About the Author

Rick Wagner is CEO of Digalog Systems. He has been with the company since 1983, previously holding the positions of engineering manager and design engineer. Digalog Systems, 3180 S. 166th St., New Berlin, WI 53151, 262-797-8000, e-mail: rwagner@digalogsys.com

For More Information,
visit our website at
www.evaluationengineering.com
and access the Article Archives.

EE

CIRCLE 63