Software Verification Using COTS Tools

Issues and Solutions Surrounding GATM and DO-178B

Freescale Semiconductor
Authors, Nat Hillary and Tammy Reeve

Document Number: SWVERIFICATIONWP
Rev. 0
11/2005

freescale semiconductor

CONTENTS

1. Introduction

Air Force Instruction 63-1301 dated 9 May 2001, “Assurance of Global Air Traffic Management Certifications” states that the purpose of Global Air Traffic Management (GATM) and Navigation Safety certification is to ensure Air Force aircraft and Air Force managed aircraft acquisitions, and modifications, conform to appropriate civil requirements.

The GATM certification process described in this document references the civil standards used by the Federal Aviation Administration (FAA), including Advisory Circulars. FAA Advisory Circular AC 20-115B recognizes DO-178B as a means of demonstrating compliance to the Federal Aviation Regulations for the software aspects of airborne systems and equipment. This means that DO-178B will need to be used as a means of evaluating software for GATM certification.

Due to the safety-critical nature of many airborne systems, the Federal Aviation Administration requires that airborne software be developed to the highest levels of quality. To this end, the FAA provides guidance for the development, test and certification of software in the RTCA DO-178B document. DO-178B contains guidelines and provisions that result in disciplined software development processes—if followed.

Software development can be a very repetitive and human-labor intensive process. This can often result in errors, as well as high costs. For these reasons various tools have been developed to automate portions of this process. If the tools are dependable, then improvements in productivity and lower numbers of in-service errors may be realized. DO-178B describes “Software Verification Tools” as ones that replace or automate a manual verification process of DO-178B (e.g., coverage analysis tools) and that cannot introduce errors, but may fail to detect them.

So how do we meet the guidelines described in DO-178B? This document answers this question by describing three key lessons, and a golden rule, that can be learned from the development of avionics systems in accordance with DO-178B.

Starting with a brief look at what is meant by that infamous term, “software quality”, this paper takes a quick look at the validation and verification activities that are used to achieve it. This paper will not dwell on the details of DO-178B (if you want this, then please visit www.amc.com) but will cut to the chase with three key DO-178B software verification lessons and a golden rule that you can start implementing today, and will consider the role that Software Validation tools play when used for structural coverage analysis.

Lessons Learned from DO-178B	Best Possible Software Test and Verification Practices
Lesson One	All testing needs to be performed at the system level and be driven by software requirements. Use structural coverage analysis to measure the effectiveness of your functional testing.
Lesson Two	There is no “one size fits all” coverage analysis metric. The appropriate level of coverage needs to be based on the role of the embedded software within your system.
Lesson Three	To have high quality code, you need high quality tools. Your choice of tool should depend on its availability, support, compatibility, multi-level coverage capabilities and a track record with systems seeking industry certification such as DO-178B or ED-12B and ISO.
The Golden Rule	Concentrate on the testing process first and using software verification tools second.

2. Software Quality: Objectivity Versus Subjectivity—Where Are We?

Software quality is determined by answering two questions: does the software successfully fulfill the task that it was intended to do, and is it free from defects?

Two activities are used to ensure software quality—validation and verification. Validation is the name assigned to activities used to answer the question “are we building the right system?” Verification is the name assigned to activities used to answer the question “are we building the system right?”

For example, consider developing a word processing application. Tests are run to ensure that no errors resulted from all of the inputs that the software (written to match the system requirements) was written to accept. What good is this word processor to a user if requirements and code were not written to accept and handle a carriage return? The code could be flagged as flawless by the tests, but it would still be defective to the user.

In this scenario, the tests used to determine that the implementation was error-free (i.e., built right) are an example of software verification. Checking to ensure that the system design includes a requirement to accept a carriage return (i.e., the right system) is an example of software validation.

Both validation and verification activities are required in order to build quality software. However, the focus of this article is software verification.

The predominant method of software verification is testing. Software testing occurs at all stages of software development, from implementation (where developers test and debug the code in the components they've written) through to system testing (where all components are integrated and the software is tested in its target environment).

The primary objective of testing is to identify flaws in the software implementation. There are many testing techniques and methods, from the debugging methods used by developers, through to white and black box testing. Typically, during testing, software is exercised using all conceivable permutations of inputs in a range of operating conditions, in an attempt to identify defects. Testing resources are limited, however, and a difficult question to answer is when is testing complete?

At the extreme, testing is complete when every possible execution path has been exercised, error free, at least once. An observation made by G. J. Myers in 1976 explains why this is impossible to achieve. Myers described a 100-line program that had 10^{18} unique paths. For comparative purposes, he noted that the universe is only about 4×10^{17} seconds old. With this observation, Myers concluded that complete software execution path testing is impossible, so an approximation alternative and another metric are required to assess testing completeness. Structural coverage analysis has been accepted as an excellent metric for assessing testing effectiveness.

So what does a world-class testing process that may be implemented with only limited resources look like? The lessons learned from developing avionics software help to paint a clear picture of how this may be achieved. Before considering these lessons, however, let us look at what structural coverage analysis is all about.

3. Structural Coverage Analysis: How Covered Are You? How Covered Do You Need to Be?

Structural coverage analysis is a process whereby the code structures exercised by a given test or tests are measured. An example of source level coverage results generated from Freescale's CodeTEST™ Coverage Analysis tool can be seen in Figure 1.

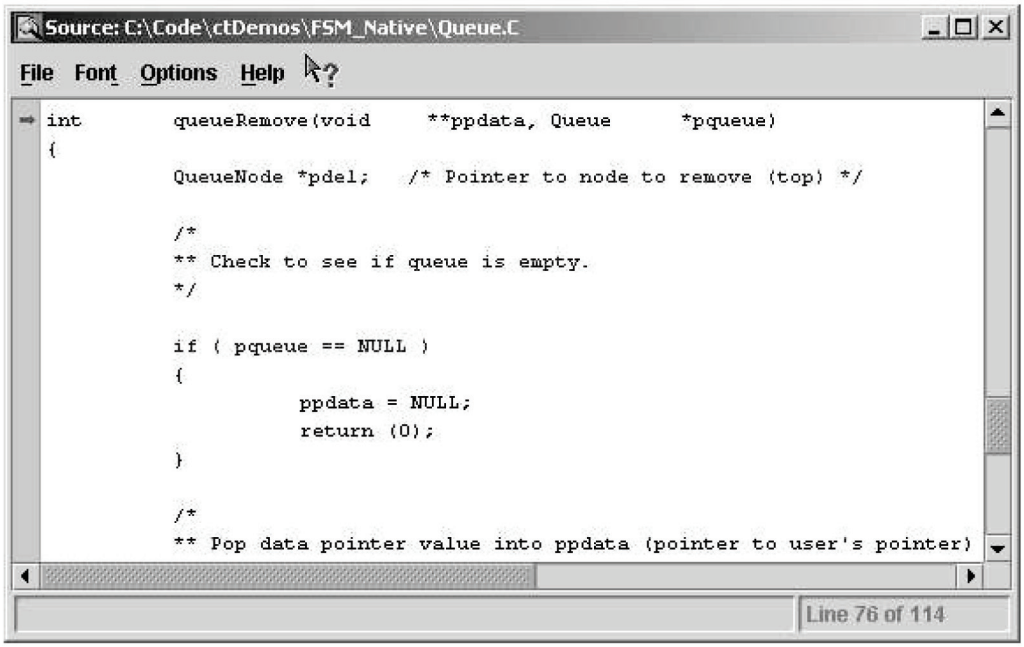
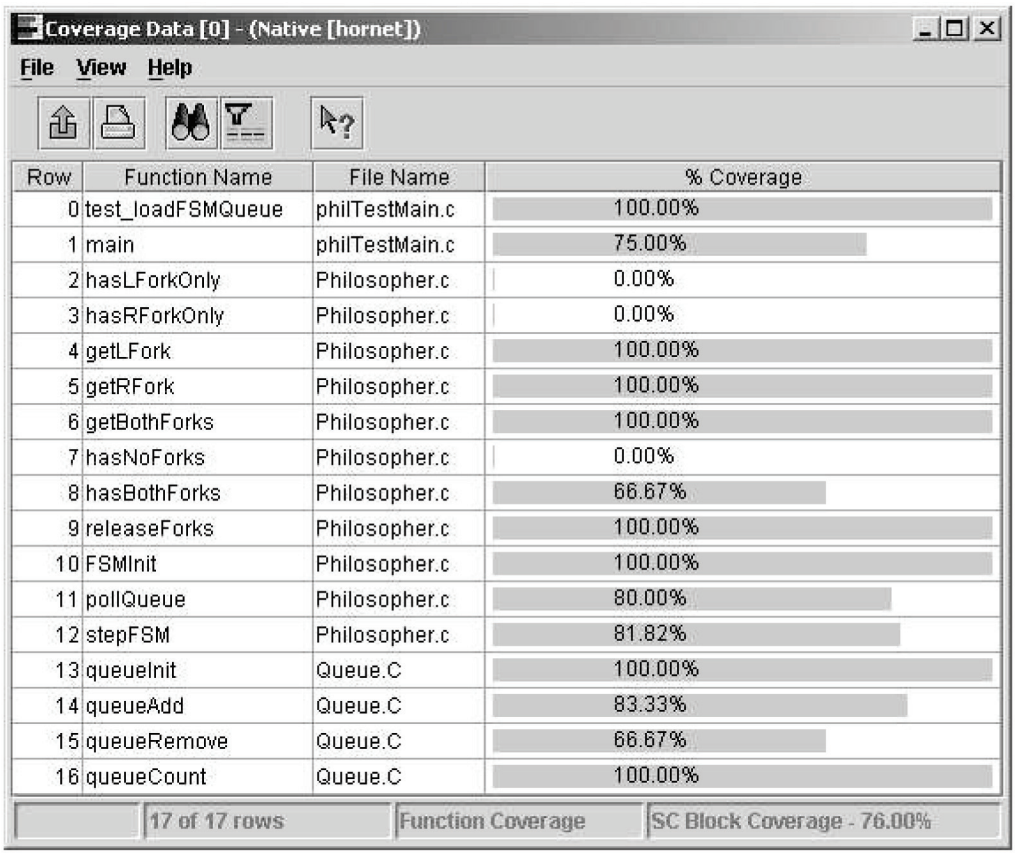


FIGURE 1: IN THIS SOURCE CODE COVERAGE VIEW, EXECUTED CODE IS HIGHLIGHTED IN BLUE, WHILE NON-EXECUTED CODE REMAINS IN BLACK. COVERAGE INFORMATION LIKE THIS MAY BE USED.

There are many different types of structural coverage analysis. DO-178B uses the following three coverage measurements, ordered from simple to complex:

Statement Coverage: Functional test execution paths are analyzed to ensure that every statement in the program has been invoked at least once during testing. Figure 2 shows a typical statement coverage report.



Row	Function Name	File Name	% Coverage
0	test_loadFSMQueue	philTestMain.c	100.00%
1	main	philTestMain.c	75.00%
2	hasLForkOnly	Philosopher.c	0.00%
3	hasRForkOnly	Philosopher.c	0.00%
4	getLFork	Philosopher.c	100.00%
5	getRFork	Philosopher.c	100.00%
6	getBothForks	Philosopher.c	100.00%
7	hasNoForks	Philosopher.c	0.00%
8	hasBothForks	Philosopher.c	66.67%
9	releaseForks	Philosopher.c	100.00%
10	FSMInit	Philosopher.c	100.00%
11	pollQueue	Philosopher.c	80.00%
12	stepFSM	Philosopher.c	81.82%
13	queueInit	Queue.C	100.00%
14	queueAdd	Queue.C	83.33%
15	queueRemove	Queue.C	66.67%
16	queueCount	Queue.C	100.00%

FIGURE 2: IN THIS FUNCTION COVERAGE VIEW, IT IS POSSIBLE TO TELL AT A GLANCE WHICH FUNCTIONS HAVE BEEN ADEQUATELY TESTED AND THOSE THAT HAVE NOT. IMPROVING FUNCTIONAL TESTS ON THOSE FUNCTIONS WITH LOW COVERAGE RESULTS LEADS TO GREATER TEST EFFECTIVENESS.

Decision Coverage: Functional test execution paths are analyzed to ensure that every point of entry and exit in the program has been invoked at least once during testing and every decision in the program has taken on all possible outcomes at least once during testing.

Modified Condition/Decision Coverage: Functional test execution paths are analyzed to ensure that every point of entry and exit in the program has been invoked at least once during testing, every condition in a decision in the program has taken all possible outcomes at least once during testing, every decision in the program has taken all possible outcomes at least once during testing, and each condition in a decision has been shown to independently affect that decision's outcome. A condition is shown to independently affect a decision's outcome by varying just that condition while holding fixed all other possible conditions.

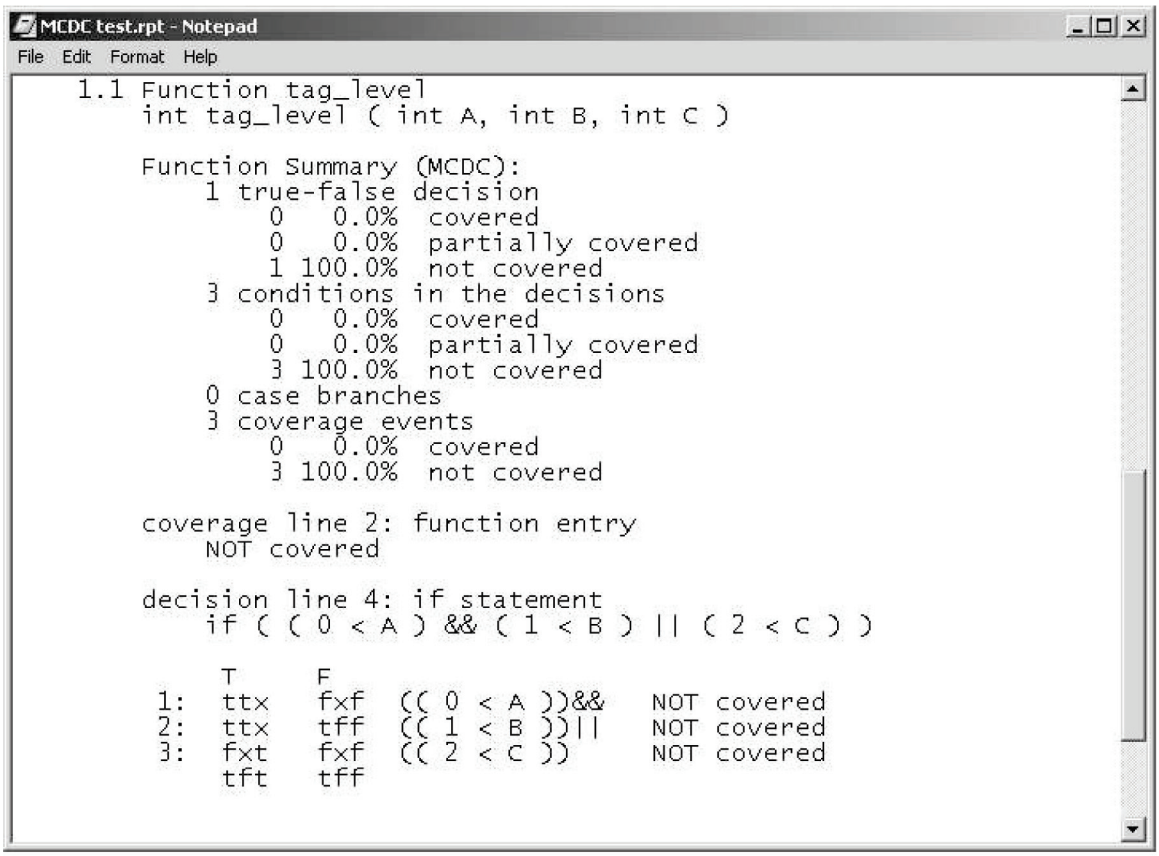


FIGURE 3: WHEN DEVELOPING SOFTWARE IN ACCORDANCE WITH DO-178B, FORMAL REPORTS ARE REQUIRED TO PROVE THE COVERAGE LEVELS ACHIEVED THROUGH TESTING. THIS MODIFIED CONDITION/DECISION COVERAGE REPORT SUMMARY IS A TYPICAL EXAMPLE.

Manual structural coverage analysis is a very repetitive and human-labor intensive process. The use of structural coverage analysis tools helps to guarantee the accuracy and cost effectiveness of making coverage measurements. Although DO-178B allows for the use of coverage analysis tools, their use is not mandated. Care should be taken to ensure that when coverage analysis tools are used, it is to streamline the software verification process and not to replace knowledge about structural coverage analysis.

The choice of which type of coverage analysis to use is based on how safety critical the system under development is.

4. Lessons Learned from DO-178B

DO-178B states that “Software verification process objectives are satisfied through a combination of reviews, analyses, the development of test cases and procedures, and the subsequent execution of those test procedures. Reviews and analyses provide an assessment of the accuracy, completeness, and verifiability of the software requirements, software architecture, and Source Code. The development of test cases may provide further assessment of the internal consistency and completeness of the requirements. The execution of the test procedures provides a demonstration of compliance with the requirements.”

DO-178B describes the software verification activities independently from the use of software verification tools. The objective of this is to ensure that development processes are designed to focus on testing first and automation tools second. There are three lessons that may be learned from the DO-178B approach.

4.1 Lesson One

The first and most important lesson to be learned about software verification from DO-178B is that all testing needs to be performed at the system level and be driven from the software requirements. Structural coverage analysis is then used to measure the effectiveness of the functional testing.

This means that requirements-based functional testing must be executed first, before any structural coverage analysis metrics are gathered. Essentially, this is saying that it is important to understand how the software should operate before attempting to identify flaws in it.

Once requirements-based functional testing is complete, structural coverage analysis is done to determine which code structures were not exercised by the requirements-based test procedures. The requirements-based test cases may not have completely exercised the code structure, so structural coverage analysis is performed and additional verification produced to provide structural coverage. This helps to identify shortcomings in requirements-based test cases or procedures, inadequacies in software requirements, dead code (i.e., code that cannot be reached) or deactivated code (e.g., #ifdef'd code that is not included in all software configurations).

Note: Unit testing is not explicitly mentioned. However, coverage analysis can also be beneficial to unit testing. In many circumstances, the defects that testers find can be a direct result of the fact that they are the first to execute portions of code (in one instance, the number of defects discovered this way amounted to more than 75 percent of the total number of defects found!). Using coverage analysis at the debugging stage ensures that the developers uncover the maximum numbers of bugs. This enables testers to focus on what they do best—identifying the more esoteric defects. In the business case described at the end of this article, coverage analysis was employed during development.

The target that DO-178B sets for the coverage analysis of avionic software is 100 percent. In practice, this is a difficult target to achieve through testing alone. To put this in perspective, industry guru Chuck House cites one leading software consultant's experience of coverage analysis in the industry in general. Dick Bender's experience (based on the measurements that he has made) is that the best coverage analysis results he has seen are 70 percent. So how do avionics companies achieve 100 percent coverage?

Avionics companies achieve 100 percent coverage through a combination of functional testing and analysis. Requirements-based functional testing is performed, and then coverage analysis measurements are made. From the results, those sections that have not yet been tested are analyzed to determine whether they have not been tested due to shortcomings in the requirements-based test cases or procedures, inadequacies in the software requirements or dead code.

If there are shortcomings in the requirements-based testing, or the software requirements are incomplete, then the requirements and/or the tests are modified to address the omission. Any dead code identified is removed from the software. Tests are then repeated. These changes result in improved coverage measurements. Even this rarely enables 100 percent of the code base to be executed, however. For much of the code that is not covered, it is prohibitively expensive to create the system faults and conditions that will cause the code to be executed. For this code, additional analysis methods (e.g., code walkthroughs) must be used to ensure that there are no defects in the code.

One hundred percent coverage is not necessarily achieved through code execution, then, but all of the code needs to be executed, inspected and analyzed for defects.

4.2 Lesson Two

The second lesson to be learned about software verification from DO-178B is that there is no one-size-fits-all coverage analysis metric. The coverage objectives for each system must be chosen based on the role of the system. For some systems, coverage analysis may not be required, while for others it is essential. For instance, DO-178B requires that MC/DC coverage, one of the most strenuous coverage analysis metrics, be performed on the most safety-critical systems. For those systems that do not affect the safe flight of aircraft (e.g., the In-Flight Entertainment system), coverage analysis is not required. How critical the software is to the continued safe operation of an aircraft is determined as a result of a System Safety Assessment that is performed on the whole airborne system. From this assessment, a software criticality level is assigned, which drives a coverage analysis objective (Table 1).

TABLE 1: DO-178B SOFTWARE CRITICALITY LEVELS AND REQUIRED COVERAGE ANALYSIS LEVELS

Software	Definition	Associated Structural Coverage Level Objectives, (DO-178B 6.4.4.2 and Table A-7)
Level A	Software that could cause or contribute to the failure of the system resulting in a catastrophic failure condition.	Modified Condition/Decision Coverage, Decision Coverage and Statement Coverage
Level B	Software that could cause or contribute to the failure of the system resulting in a hazardous or severe-major failure condition.	Decision Coverage and Statement Coverage
Level C	Software that could cause or contribute to the failure of the system resulting in a major failure condition.	Statement Coverage
Level D	Software that could cause or contribute to the failure of the system resulting in a minor failure condition.	None required
Level E	Software that could cause or contribute to the failure of the system resulting in no effect on the system.	None required

4.3 Lesson Three

The third lesson to be learned about software verification from DO-178B is that because high quality code is required, any tools used to measure coverage should be high quality tools.

For DO-178B, the quality of a coverage analysis software tool is assured through a process called “qualification,” where the tool is tested in the target environment to ensure that it provides accurate and reliable results.

Once a tool has been qualified for use in the development of airborne software, the results that the tool produces may be used with confidence and without further verification.

5. Summary

We have seen that there are many lessons to be learned about software verification from the approach that is used in the development of avionic systems. Each of these lessons embodies a golden rule, however, to concentrate on the testing process first and using software verification tools second.

Testing must focus first on requirements-based functional tests conducted at the system level. Structural coverage analysis should then be used to analyze the effectiveness of the requirements-based testing. The level of coverage analysis chosen should be based on the resources available, and how critical the software is to the operation of the overall system. Finally, to ensure the highest quality code, any coverage tool used should be a high quality tool. DO-178B sets the example for this by requiring that any coverage tool whose results are not going to be independently verified needs to be tested to ensure that it provides accurate and reliable results.

This approach to software verification has been refined over the past 20 years, starting with the development of the original DO-178 document, and evolving through two additional revisions, DO-178A and DO-178B. The contents of the latter two versions captured the experience gained during the application of the preceding versions; with DO-178B being a complete rewrite of DO-178A.

6. Contact and Resources

Key Contact:

Alice Gannon-McKinley Marketing Communications Manager Applied Microsystems Corporation 2050 148th Ave NE Redmond, WA 98052

Phone: 425-882-5233 Fax: 425-883-3049 e-Mail: aliceg@amc.com

The following resources provide great next steps to understanding software verification, DO-178B and best possible testing practices.

Other Resources

> Tool Qualification for DO-178B White Paper, November 2001, Applied Microsystems Technology Center, http://www.amc.com/techcenter/whitepapers/

> DO-178B & ED-12B Software Verification Using CodeTEST® ACT, http://www.amc.com/news/features/feature3.html

> Software Considerations in Airborne System and Equipment Certification—RTCA/DO-178B, RTCA Inc., Washington D.C., December 1992

> FAA Notice N8110.91, Guidelines for the Qualification of Software Tools Using RTCA/DO-178B, Jan 16, 2001

> http://av-info.faa.gov/software/Policy%20&%20Guidance/N8110_91.pdf

> FAA Advisory Circular AC 25-1309-1A and/or the Joint Aviation Authorities AMJ 25-1309 FAA AC-20-115B

> Cem Kaner, et al, Testing Computer Software, Second Edition, Massachusetts, International Thomson Computer Press, 1993

> Myers, G.J., Software Reliability: Principles and Practices. New York: John Wiley & Sons. 1976

> Chuck House, Development Dilemmas and the SEI Model-Case Studies in Software Process Development, presented to the 9th IEEE International Software Quality Week, San Francisco, May 23rd, 1996

> Arnold Berger, Steve Dearden, Improving Software; Quality and Productivity with Coverage Analysis. RTC Magazine Industry Watch feature article, August 2001

How to Reach Us:

Home Page:
www.freescale.com

e-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor

Technical Information Center, CH370

1300 N. Alma School Road

Chandler, Arizona 85224

1-800-521-6274

480-768-2130

support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH

Technical Information Center

Schatzbogen 7

81829 Muenchen, Germany

+44 1296 380 456 (English)

+46 8 52200080 (English)

+49 89 92103 559 (German)

+33 1 69 35 48 48 (French)

support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.

Headquarters

ARCO Tower 15F

1-8-1, Shimo-Meguro, Meguro-ku,

Tokyo 153-0064, Japan

0120 191014

+81 3 5437 9125

support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.

Technical Information Center

2 Dai King Street

Tai Po Industrial Estate,

Tai Po, N.T., Hong Kong

+800 2666 8080

support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor

Literature Distribution Center

P.O. Box 5405

Denver, Colorado 80217

1-800-441-2447

303-675-2140

Fax: 303-675-2150

LDCForFreescaleSemiconductor

@hibbertgroup.com

Document Number: SWVERIFICATIONWP
Rev. 0
11/2005

freescale semiconductor