



White paper

**Improving your software quality with
embedded test automation**

COPYRIGHT NOTICE

© Copyright 2011 Atollic AB. All rights reserved. No part of this document may be reproduced or distributed without the prior written consent of Atollic AB.

TRADEMARK

Atollic, Atollic TrueSTUDIO, Atollic TrueINSPECTOR, Atollic TrueVERIFIER and Atollic TrueANALYZER and the Atollic logotype are trademarks or registered trademarks owned by Atollic. ECLIPSE™ is a registered trademark of the Eclipse foundation. All other product names are trademarks or registered trademarks of their respective owners.

DISCLAIMER

The information in this document is subject to change without notice and does not represent a commitment of Atollic AB. The information contained in this document is assumed to be accurate, but Atollic assumes no responsibility for any errors or omissions. In no event shall Atollic AB, its employees, its contractors, or the authors of this document be liable for any type of damage, losses, costs, charges, claims, demands, claim for lost profits, fees, or expenses of any nature or kind.

DOCUMENT IDENTIFICATION

ASW-WPTA June 2011

REVISION

First version

Atollic AB

Science Park
Gjuterigatan 7
SE- 553 18 Jönköping
Sweden

+46 (0) 36 19 60 50

E-mail: sales@atollic.com

Web: www.atollic.com

Atollic Inc

115 Route 46
Building F, Suite 1000
Mountain Lakes, NJ 07046-1668
USA

+1 (973) 784 0047 (Voice)

+1 (877) 218 9117 (Toll Free)

+1 (973) 794 0075 (Fax)

E-mail: sales.usa@atollic.com

Web: www.atollic.com

Contents

Abstract	1
Introduction.....	2
Find bugs as early as possible.....	3
Unit testing	4
Automatic generation of unit tests	5
Automatic execution of unit tests	6
How good were the tests?	8
Summary.....	10

Tables

No table of figures entries found.

ABSTRACT

It is an unfortunate fact that many embedded systems are released to market without sufficient testing. Most embedded developers do not use testing tools of any kind to support this crucial activity. Further compounding the problem is that very few tools for in-target testing of embedded systems software are available on the market.

This white paper outlines how modern tools can support embedded developers in the important areas of automated in-target testing which reduces the amount of engineering hours needed while at the same time increasing both completeness of the test coverage and the software quality.

INTRODUCTION

Releasing a product with bugs is potentially very expensive when costs of field upgrades, recalls, repairs, etc are considered. Less quantifiable, but equally important is the diminution of a company's reputation and loss of good will. Yet, many products based on embedded systems are released without all of the testing which is necessary and/or desirable to minimize these problems.

For those developers of embedded projects who choose to employ automated in-target software testing, the challenge is to find tools that are well integrated into their embedded development environment. Tools with this level of integration lend themselves to more efficient testing and rectification of discovered problems.

Most existing software testing tools can only execute the tests on a PC. This is of limited use when testing an embedded application because the actual system will have different hardware interfaces, different timing issues, memory constraints, target specific #pragmas, inline assembly, incompatible linker configuration files and other differences.

Additionally, ANSI-C compliant source code often behaves differently when compiled using different compilers or run on different CPU architectures. Typical examples are packing and ordering of structure fields, whether a "char" data type becomes 8 or 32 bit by default, etc.

Recompiling embedded code on a PC often affects the runtime behavior in ways that may not be easily understood or may produce misleading results. A test that passes on a PC might well fail when it is run on the target board.

Consequently, it is of considerable importance to run as many software tests as possible on the actual embedded hardware so as to minimize inter-platform differences. For this reason, tools that facilitate on-target testing are inherently superior to those which rely on external platforms such as PCs to simulate the application.

FIND BUGS AS EARLY AS POSSIBLE

Finding and fixing bugs is a fact of life in software development, but it takes on special importance in embedded applications development. It is well known that finding errors early in the cycle is much cheaper than finding them later. However, with traditional test and debug strategies, many bugs remain hidden until late in the development and test cycle by which time the cost of fixing them is high. Development teams should thus strive to find and correct bugs as early as possible in the development cycle.

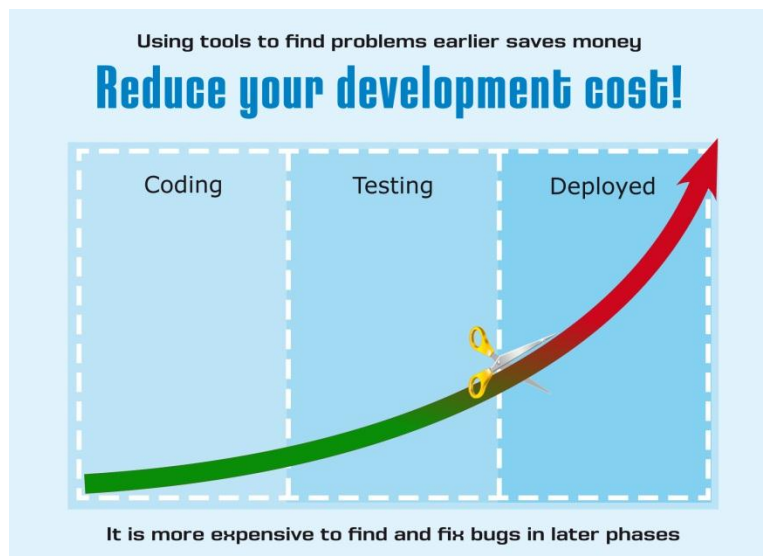


Figure 1 - It is cheaper to correct problems earlier

By deploying tools that simplify and expand the coverage of testing, it becomes easier to create better tests earlier in the development process. It is also much easier to perform regression tests that ensure that code changes do not introduce new bugs in existing code.

UNIT TESTING

Most embedded applications are not the subject of a formal test methodology, although more and more teams are looking into the use of this methodology. Unit tests are those tests which exercise your C functions, by calling them with different combinations of input parameter values, to drive the code through different execution paths of the function.

Writing unit tests by hand is labor intensive, costly, tedious, and still leaves the possibility that many of the possible execution paths are not tested. To guarantee that as many as possible of the important execution paths throughout a function would be exercised by a test suite, requires a detailed analysis of how the input parameters drive the code in various directions. This is very time consuming and difficult or impossible for anything but trivial functions.

Maintaining the synchronization of unit tests with code under development is another challenge that development teams face. This problem is often exacerbated by the fact that schedules are tight and frequently become compressed as the project progresses. If source code and unit tests thus get out-of-synch, the unit tests will become less useful, especially at the time that they are most needed.

Embedded systems present their own special challenge in testing, as all testing is inherently remote. The most effective unit tests are those built into the application and run on the target hardware in its native application environment, rather than on an ancillary platform (such as a PC) that only simulates the application.

The unit test tools available to PC developers are less useful to embedded developers as they rarely manage compilation, downloading and execution of the test suites in embedded boards. Tools that create the unit tests automatically, build them into the embedded application and allow you to run them on the target via a convenient pathway such as a JTAG debugger will result in optimal usage of time and maximum productivity.

A more effective approach is to use a full embedded test automation system that are integrated into the IDE so that synchronization is easy to maintain, and ancillary tools such as JTAG debug probes can be used to full effect. Such tools have not been common in the embedded industry previously, but new tools like **Atollic TrueVERIFIER™** now meet these demanding criteria and bring very powerful in-target test automation capabilities to embedded software developers.

AUTOMATIC GENERATION OF UNIT TESTS

The new breed of professional embedded systems test automation tools can analyze the source code and generate and execute suitable test suites automatically.

Atollic TrueVERIFIER™ auto-generates a test suite (implemented in C source code), that makes many function calls with different combinations of input parameters, thus driving a large number of different execution paths in the functions to be tested.

The illustration below shows how a trivial C function can be tested, by calling it many times with different input parameters. The selection of input parameter values (the test data) is selected and generated automatically by the testing tool.

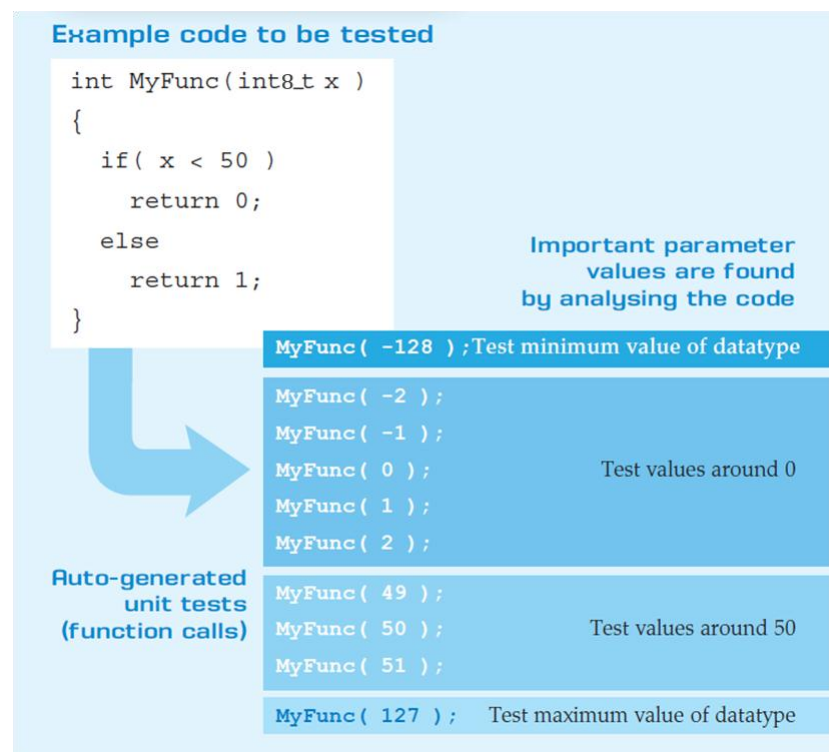


Figure 2 - Automatic generation of unit tests

Atollic TrueVERIFIER™ is a powerful tool for embedded test automation. It analyzes the source code of the application and automatically generates unit tests with important combinations of input parameter values, in order to drive as many important combinations of execution paths in the function as possible. To maintain full flexibility in case special cases need to be addressed, it is also possible to create test cases manually with custom sets of input parameter values.

AUTOMATIC EXECUTION OF UNIT TESTS

Once the test suite has been generated, it must be compiled, linked and executed on the target system. Many low-cost unit test tools exist that only run unit tests on a Windows PC. The inability of these tools to integrate with the embedded tool development environment, results in additional inefficiency of not being able to execute the tests on the target board.

This type of tool is not ideally suited to the needs of professional embedded systems testing. Test automation tools that are a part of, and not an ad-hoc addition to, the development tools suite offer the most efficient way to maintain test synchronization with code development, keep up with changes effectively, and give developers the best chance of finding errors as early as possible in the cycle before bug fixing and problem resolution become excessively expensive.

Atollic TrueVERIFIER™ is exactly such a tool that integrates seamlessly into a professional embedded IDE, enables easy synchronization of test with code development and takes advantage of integration with ancillary tools such as JTAG debug probe.

TrueVERIFIER™ integrates directly into **Atollic TrueSTUDIO®**, a modern state-of-the-art IDE designed explicitly to serve the needs of professional developers, especially those operating in teams. Once the unit tests have been generated in C source code, they are compiled automatically using the integrated embedded build tools. They are then downloaded to the target board using the same JTAG probe that works with the IDE that is being used for normal debugging. Finally, execution is performed in the target board with dynamic execution flow analysis to measure the achieved code coverage.

The testing tool system is shown in the illustration below:

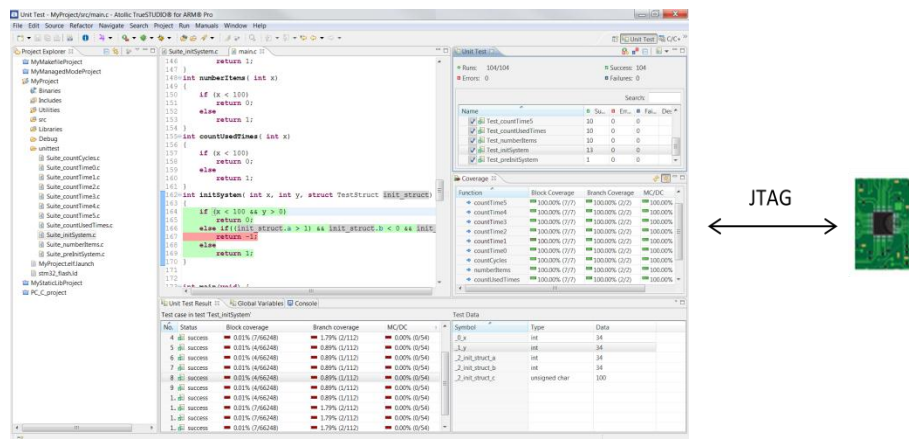


Figure 3 – The embedded systems test automation tool with target connection

Once the test suite is completed, test results and code coverage information is uploaded to the IDE. **Atollic TrueVERIFIER™** measures code coverage on MC/DC level (see the next section for details), the test quality level required for flight control system software.

In effect, a professional embedded systems test automation tool like **Atollic TrueVERIFIER™** automates the following tasks:

- Analyzing the source code to find out what combinations of input parameter values affect the execution flow.
- Generation of test suites (in C source code) that calls the functions to be tested many times with different combinations of input parameter values, thus driving different execution paths in the functions being tested.
- Building (compilation and linking) of the test suite source code.
- Downloading the built test suite into the target board, using the same JTAG probe as is used for debugging.
- Execution of test suites in the target board with execution flow monitoring, thus enabling advanced code coverage analysis to measure the achieved test quality.
- Uploading and visualization of test results and achieved code coverage.

HOW GOOD WERE THE TESTS?

While it is very useful to get a test suite to be generated and executed automatically, this is of little value unless you can trust that the test suite actually test your software well enough. The best way to develop confidence in unit tests is to examine the results to actually see what happens when they are executed. This means that code coverage is necessary.

Code coverage analysis is achieved using dynamic execution flow analysis, and is commonly used to study what parts of the code have been tested, and hence measure the test quality. There are many different types of code coverage analysis, from very simple analysis up to very stringent types:

- Statement or block coverage
- Function coverage
- Function call coverage
- Branch coverage
- Modified condition/decision coverage (MC/DC)

Even simple code sections are very difficult to test rigorously. To make a thorough test, all code blocks should be executed, all branches should be exercised, and all sub-expressions in complex branch decisions ought to be tested to ensure that each sub-expression has been driving the branch decision independently of the other sub-expressions.

Different types of code coverage analysis are often classified formally, such as modified condition/decision coverage (MC/DC) listed above.

The more advanced types of code coverage analysis are often used for measuring test quality of safety critical software. As an example, RTCA DO-178B (a standard for development of flight safety critical software) requires MC/DC testing of software on "Level-A criticality", the most critical part of airborne software, where a software error can lead to a catastrophic situation with loss of aircraft or human lives.

Many projects outside of the aerospace industry can also benefit from more rigorous testing and quantification of what has been tested. In for the case of companies that ship products with high volumes, defects from software errors are seen by a larger audience and are commensurately more expensive to fix.

Some products that are deployed in remote locations or have other use cases making them difficult or expensive to upgrade in the field certainly would benefit from more thorough testing. In all cases where suppliers want to keep their good reputation and software, defects can be costly for a company's reputation and market share.

For these reasons, code coverage analysis ought to be used if at all possible to verify whether the software has been tested well enough or not before delivery to customers.

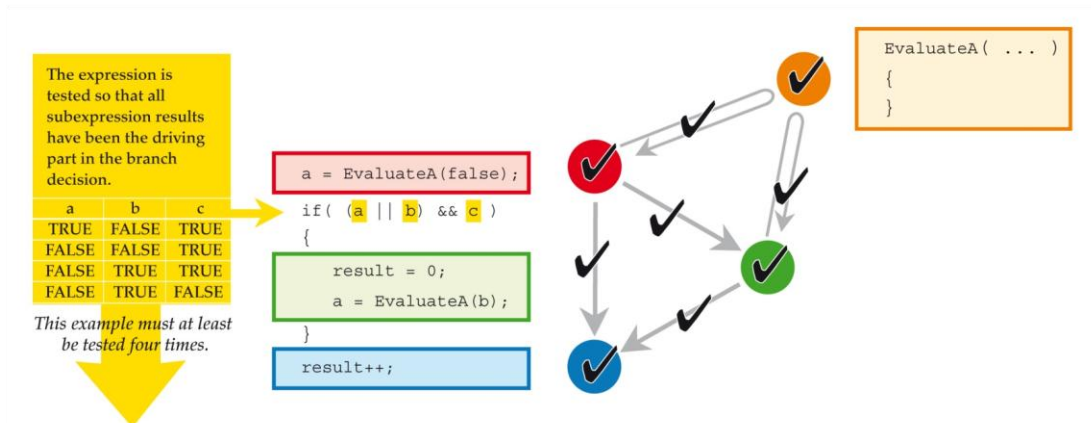


Figure 4 - Measuring test quality using code coverage analysis

The illustration above visualizes how **Atollic TrueVERIFIER™** monitors the execution flow in the functions during execution of a test suite. It also shows that even a relatively simple code construct generates many possible paths of execution.

In real-world examples code is often more complex, posing greater testing and coverage challenges. **Atollic TrueVERIFIER™** measures the code coverage on MC/DC-level (modified condition/decision coverage), which is the test quality level required for flight control system software.

For each function, **Atollic TrueVERIFIER™** detects if all code blocks are executed; if all branches have been exercised; and if all functions have been called and if all function calls have been exercised. Whether or not all sub-expressions in complex branch decisions have been the driving part in the overall branch decision is also checked. Results are collated and displayed for easy and positive verification of test efficacy.

Atollic TrueVERIFIER™ not only generates unit tests automatically, it works seamlessly with other tools in the IDE to download instrumented code to the target, run and collect test results for analysis.

Not only does this information give you a very high confidence in the actual test coverage and quality achieved, it is doing so by measuring code coverage at the same level as is required for flight control system software. In other words, using **Atollic TrueVERIFIER™** not only speeds up the process, but considerably raises the level of rigor in testing and code analysis.

SUMMARY

The use of a professional embedded systems test automation tool like **Atollic TrueVERIFIER™**, gives developers automatic generation and execution of unit tests cases in the most realistic scenario possible, the target board itself.

By cutting out intermediate steps of creating test cases for platforms that only simulate, but do not reproduce the functioning of the device under test, the time required to analyze the source code for testing, writing and maintaining the unit tests, as well as building and executing them in the target board is substantially reduced.

With automatically generated unit test suites, the tests are always synchronized with the latest code changes, and the test suites normally exercise a larger set of the potential execution paths in the functions being tested. In short, the result is better software quality, faster, and with less work.

In order to cope with time to market pressure, shrinking profit margins and rising costs of maintenance and bug fixing, it is time to give embedded system developers more powerful and effective testing tools.

Highly integrated products like **Atollic TrueVERIFIER™** that can substantially cut the time from unit test case generation to results evaluation will give development teams better possibilities to deliver well designed software, on time and on budget, with improved quality.

Atollic provides a family of well integrated tools for professional embedded systems development and debugging, static source code analysis, test automation and test quality measurement.

More information about Atollic, **Atollic TrueSTUDIO®**, **Atollic TrueINSPECTOR®**, **Atollic TrueANALYZER®** and **Atollic TrueVERIFIER™** products is available here:

www.atollic.com

www.atollic.com/truestudio

www.atollic.com/trueinspector

www.atollic.com/trueanalyzer

www.atollic.com/trueverifier