

Integrating the Parasoft C++-test and MDK-ARM for achieving higher quality and safety verification

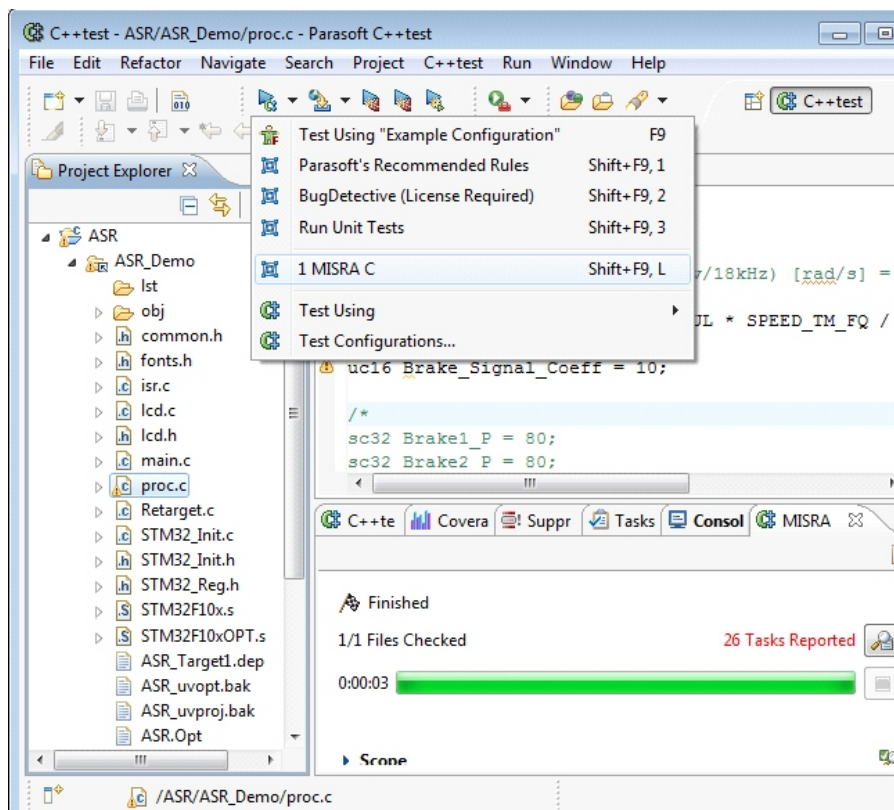
PARASOFT WHITE PAPER

Parasoft C++-test is an integrated solution for automating a broad range of best practices proven to improve software development team productivity and software quality. C++-test facilitates:

- ▶ **Static analysis** – static code analysis, data flow static analysis, and metrics analysis
- ▶ **Peer code review process automation** – preparation, notification, and tracking
- ▶ **Unit testing** – unit test creation, execution, optimization, and maintenance
- ▶ **Runtime error detection** – memory access errors, leaks, corruptions, and more

This provides teams a practical way to prevent, expose, and correct errors in order to ensure that their C and C++ code works as expected. To promote rapid remediation, each problem detected is prioritized based on configurable severity assignments, automatically assigned to the developer who wrote the related code, and distributed to his or her IDE with direct links to the problematic code and a description of how to fix it. For embedded and cross-platform development, C++-test can be used in both host-based and target-based code analysis and test flows.

Automate Code Analysis for Monitoring Compliance



continued on page 2

A properly implemented coding policy can eliminate entire classes of programming errors by establishing preventive coding conventions. C++test statically analyzes code to check compliance with such a policy. To configure C++test to enforce a coding standards policy specific to their group or organization, users can define their own rule sets with built-in and custom rules. Code analysis reports can be generated in a variety of formats, including HTML and PDF.

Hundreds of built-in rules—including implementations of MISRA, MISRA 2004, and the new MISRA C++ standards, HIS source code metrics as well as guidelines from Meyers' Effective C++ and Effective STL books, and other popular sources—help identify potential bugs from improper C/C++ language usage, enforce best coding practices, and improve code maintainability and reusability. Custom rules, which are created with a graphical RuleWizard editor, can enforce standard API usage and prevent the recurrence of application-specific defects after a single instance has been found.

Identify Runtime Bugs without Executing Software

BugDetective, the advanced interprocedural static analysis module of C++test, simulates feasible application execution paths—which may cross multiple functions and files—and determines whether these paths could trigger specific categories of runtime bugs. Defects detected include using uninitialized or invalid memory, null pointer dereferencing, array and buffer overflows, division by zero, memory and resource leaks, and various flavors of dead code. The ability to expose bugs without executing code is especially valuable for embedded code, where detailed runtime analysis for such errors is often not effective or possible.

Streamline Code Review

Code review is known to be the most effective approach to uncover code defects. Unfortunately, many organizations underutilize code review because of the extensive effort it is thought to require. The C++test Code Review module automates preparation, notification, and tracking of peer code reviews, enabling a very efficient team-oriented process. Status of all code reviews, including all comments by reviewers, is maintained and automatically distributed by the C++test infrastructure.

Monitor the Application for Memory Problems

Application memory monitoring is the best known approach to eliminating serious memory-related bugs with zero false positives. The running application is constantly monitored for certain classes of problems—like memory leaks, null pointers, uninitialized memory, and buffer overflows—and results are visible immediately after the testing session is finished.

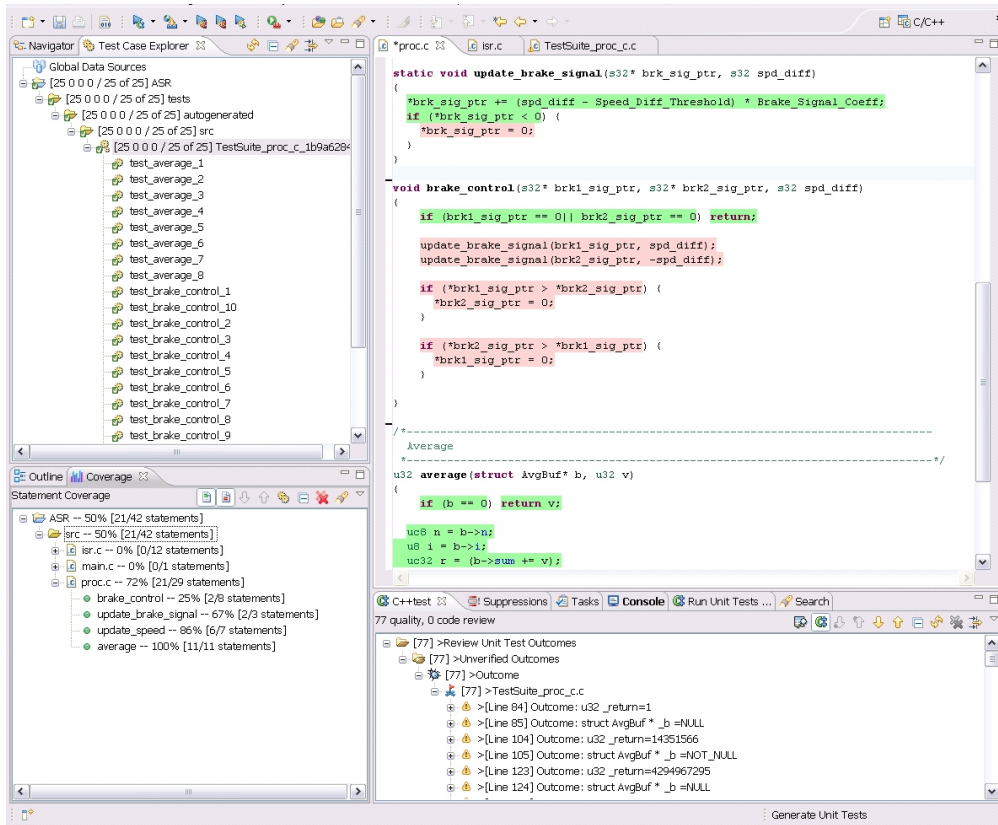
Without requiring advanced and time-consuming testing activities, the instrumented application—additional code is added for the monitoring purposes—goes through the standard functional testing and all existing problems are flagged. The application can be executed on the target device, simulated target, or host machine. The collected problems are presented directly in the developer's IDE with the details required to understand and fix the problem (including memory block size, array index, allocation/deallocation stack trace etc.)

Coverage metrics are collected during application execution. These can be used to see what part of the application was tested and to fine tune the set of regression unit tests (complementary to functional testing).

Unit and Integration Test with Coverage Analysis

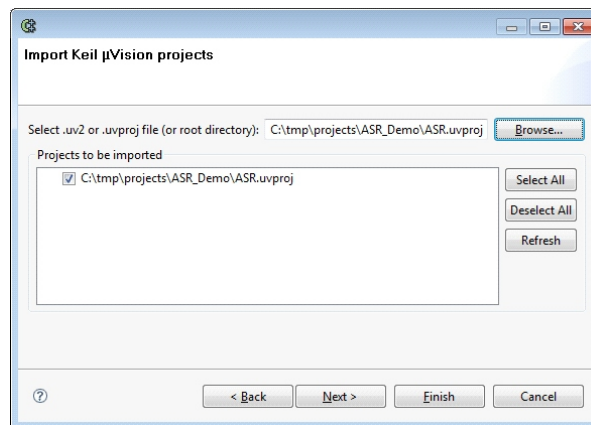
C++test's automation greatly increases the efficiency of testing the correctness and reliability of newly-developed or legacy code. C++test automatically generates complete tests, including test drivers and test cases for individual functions, purely in C or C++ code in a format similar to CppUnit. These tests, with or without modifications, are used for initial validation of the functional behavior of the code. By using corner case conditions, these automatically-generated test cases also check function responses to unexpected inputs, exposing potential reliability problems.

continued on page 3



A multi-metric test coverage analyzer, including statement, branch, path, and MC/DC coverage, helps users gauge the efficacy and completeness of the tests, as well as demonstrate compliance with test and validation requirements, such as DO-178B. Test coverage is presented via code highlighting for all supported coverage metrics—in the GUI or color-coded code listing reports. Summary coverage reports including file, class, and function data can be produced in a variety of formats.

Integration with MDK-ARM



C++test provides a wizard for easy uVision project importing. Wizard allows importing single project or collection of projects. Wizard is available in GUI mode and also in command line mode for automated projects import. Once imported, C++test will be synchronizing changes to original project whenever it detects such.

C++test is using the build bat file generated by uVision IDE to acquire the compiler/linker flag. If the build bat file generation is not enabled in uVision project import wizard will issue a warning and block project import. To enable build bat file generation in uVision simply mark the check box in uVision project properties.

continued on page 4

After wizard settings are confirmed C++test will proceed with project import. In effect user will get the new C++test project that is synchronized with uVision project. All basic settings are automatically set so it is possible to start the static analysis without any additional settings.

With unit testing the situation is bit more complicated. To successfully run unit tests on the target C++test needs to:

- ▶ prepare the test components (C/C++ source files)
- ▶ build a test executable basing on generated test components
- ▶ generate a special debugger script for uVision that will automate the test execution
- ▶ start uVision with generated debugger to execute scheduled portion of tests
- ▶ process collected results and provide the execution statistics to the users.

But C++test provides a solution to this. All testing actions, whether this are standard steps or user custom actions, are handled by so called "test configurations", and more precisely a test flow definition that is a part of the test configuration. C++test ships with number of built-in test configurations. For testing uVision4 projects with ULINKPro debug and trace adapter the best starting point is to go with **"Test Configurations>Builtin>Embedded Systems>Keil uVision>Run Keil uVision Tests- ULINKPro"** test configuration. Then user can freely modify test configuration and tailor flow definition to suit individual project needs.

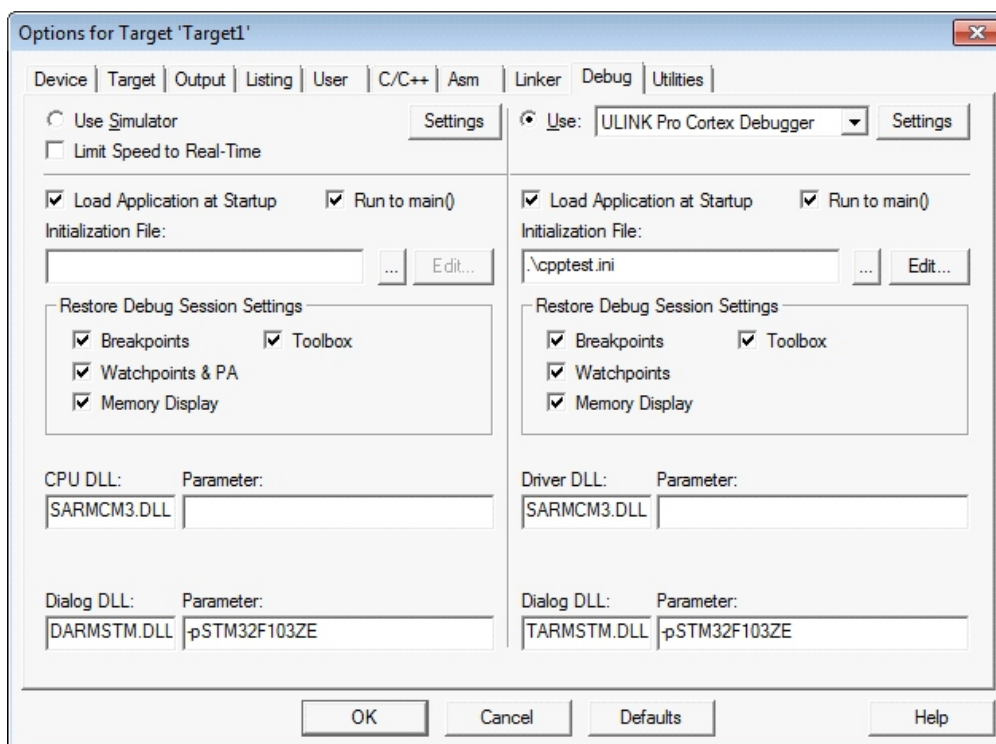
Besides the configurations done in the C++test project there are some adjustments that need to be done also in uVision project in order to achieve fully automated tests execution. Typically there are the following things to customize:

▶ **Path to the C++test generated debugger script.**

C++test uses the following command to run uVision IDE to automatically execute a prepared test executable :

```
uv4.exe -d <tested project uVision project file> -t <name of the uVision project target>
```

"-d" option enforces uVision to run Debugging mode and execute all commands that are specified in the debugger script specified for the <name of the uVision project target> target. To assure that C++test generated script will be used the path to this script needs to be set in the Project Properties>Debug tab. See the screen shot below:



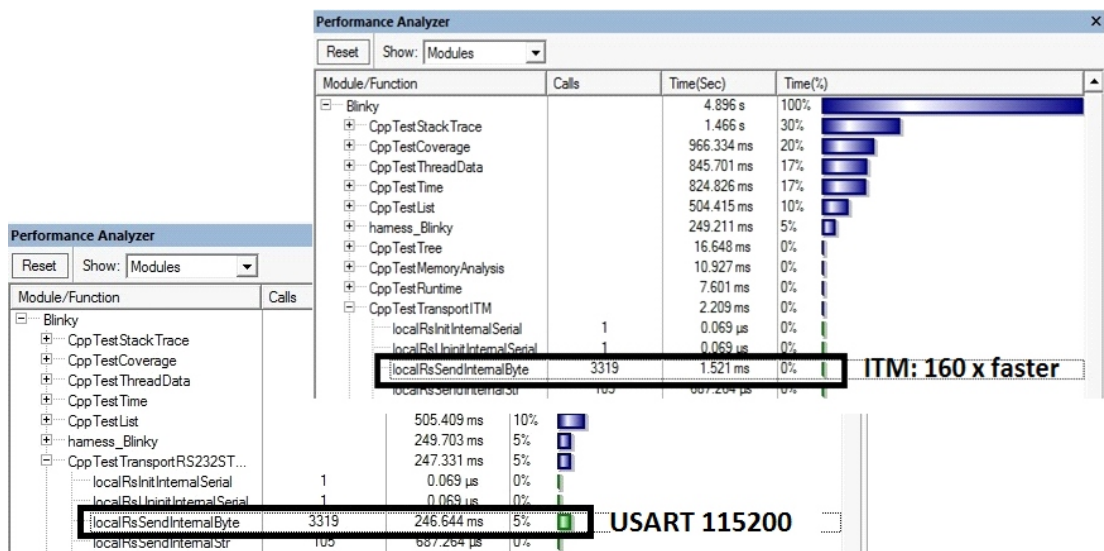
continued on page 5

► Heap and Stack memory

The unit testing framework brings some additional consumption of the memory, affecting the stack and heap usage. The amount of the stack and heap memory that is taken by testing framework depends on many factors like: instrumentation configuration, C++test runtime library configuration, code of test cases and stubs. C++test provides several configuration points to limit stack/heap. For a relatively simple projects one may assume that the 0x450 for stack and about 0x900 for heap is far enough.

► Communication channel

The configuration of the results transmission requires selecting the suitable communication channel in the C++test runtime library. The default setting for running tests with MDK-ARM 4.1 and ULINKPro is the communication based on the Instrumentation Trace Macrocell, a part of ARM CoreSight debug and trace technology. In this mode, C++test writes the test messages directly to ITM port, ULINKPro assures the data transport to the host machine, where it is captured by uVision IDE and stored into the file for analysis by C++test. Communication through ULINKPro and ITM port is even 100 times faster than traditional through UART (as seen on screenshot below)



Summary

Parasoft C++test provides a comprehensive suite of functionality for assuring code and application quality. Integration with MDK-ARM allows engineers to effectively employ this functionality to their projects. That may concern quality in general as well as safety certification. Utilization of fast ITM communication channel makes data transfer to and from target board a no-brainer. A broad range of analysis types—including coding standards compliance analysis, data and control flow analysis, unit testing, application monitoring and automated peer code review process—together with the configurable test reports containing high level of details, significantly facilitates the work required for the software verification process.

www.parasoft-embedded.com

Contact info:

Parasoft Corporation, 101 E. Huntington Dr., 2nd Flr., Monrovia, CA 91016
Ph: (888)305.0041, Fax: (626)256.6884, Email: info@parasoft.com