Techniques and Tools for Software Analysis

Freescale Semiconductor

freescale™
semiconductor

Understanding how software development can be optimized through the use of software analysis tools and techniques beyond standard debuggers—including advanced debug techniques, performance tools and code coverage functionality—can ensure higher system quality.

Software analysis tools and techniques can help tackle the costly, hard-to-find errors that frustrate and delay development, testing and deployment. They help to identify issues before they cause problems downstream or for the customer. This is important because the later in the development process an error is found, the more it costs to fix. Finding a bug after a product is deployed to the field can be costly in revenue, and can damage customer goodwill as well as corporate reputation in the industry.

This paper examines how software analysis tools may affect each stage of the development cycle so that developers have the information they need to evaluate their analysis options.

CONTENTS

1. What Are Software Analysis Tools?

Just as there are software tools available to assist in the basic building of software code, there are tools that monitor how software is behaving as it runs. These software analysis tools offer visibility into the execution history of an application. There are four basic types of software analysis tools:

> Code Coverage—Measures the amount of the software that has been executed

> Instruction Trace—Creates a record of exactly what happens as the code is executed

> Memory Analysis—Tracks the code's memory usage and identifies possible errors

> Performance Analysis—Identifies performance bottlenecks and other issues allowing fine-tuning of the application for higher performance

The primary difference between software analysis tools and traditional debugging is that software analysis tools do not require you to stop the application to test it. Debugging involves starting and stopping the software repeatedly to examine the code that was executed in order to understand the control flow inside the application.

The debugging method is problematic in an embedded environment, where systems cannot always be stopped, or where stopping the system inhibits or skews the analysis. Consider the powertrain software that controls an automobile's transmission system. With a real-time system such as this one, it is important to use tools that can gather information and monitor the control flows as the application runs, so the developer can be assured that performance and other operational design specifications are met. This technique helps ensure that the system will not fail once deployed.

Traditional debugging also comes up short in environments that have multiple processors within the same system. Increasingly common in today's systems, multiple threads have multiple tasks running at any given time. It is important to monitor what is happening in each thread—and how the threads are interacting—without disturbing the other threads by stopping the application. The ideal software analysis tool attaches to a running system with as little intrusion as possible.

2. The Benefits of Using Software Analysis Tools

2.1 Accelerate the Development Process

The primary benefits of using software analysis tools are a deeper understanding of how an application is really performing and the identification of errors earlier in the development process. It is crucial to fix errors before they can cause problems in downstream development. These tools find problems that cannot be detected with stop-mode debug operations.

To truly understand the interactions between multiple threads performing simultaneous tasks, one must be able to examine how the application interacts with the RTOSes being used. Monitoring the real-time data flows also aids in the detection of memory issues, making troubleshooting easier. The end result is an accelerated development process, as potential problems are found and dealt with in a timelier manner. In particular, Memory Analysis and Instruction Trace are the two tools that have the greatest direct impact on the speed of development.

2.2 Tune for Maximum Performance

Software analysis tools can help verify the performance accuracy of an application. By eliminating unused code and tightening processing loops, the code can be tuned for maximum performance, ensuring that the whole performs better than the sum of the parts. Performance analysis can also be used to ensure that real-time specifications are met. Instrumentation has a distinct advantage here, because it identifies every execution of a particular function. Conversely, sampling tools may miss the single occurrence that happens to fall outside the design tolerances.

2.3 Improve Software Quality

By eliminating potential errors and memory leaks, larger problems that may arise in real-world situations can be prevented. Software analysis tools can track down hard-to-find problems ranging from memory leaks to strange interactions that develop during execution. The ideal Memory Analysis tool can reset its data during an execution without affecting the software, making it easier to identify where in the source code memory leaks are occurring.

A Code Coverage tool can provide metrics that are useful for improving the QA process, and thereby the quality of current and future releases. Coverage analysis is a good way to measure the thoroughness of QA efforts.

While testing is a must, verifying the quality of the test itself can increase confidence in the application. It is important to verify the performance of any test harnesses used, particularly for software used in mission-critical systems.
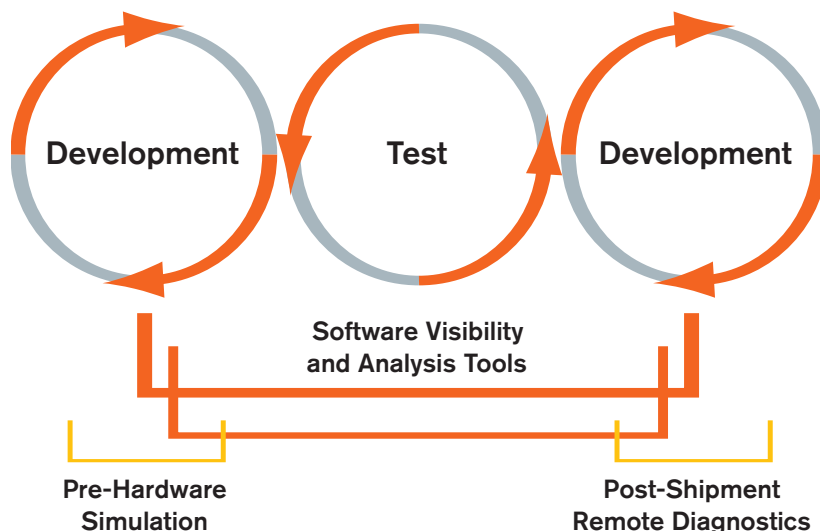
2.4 Standards Compliance and Certification

In markets such as aviation and automotive safety, there are stringent certification guidelines for all involved software. Software analysis tools can supply the depth of data needed to satisfy these requirements.

3. Software Analysis Throughout the Development Cycle

When selecting which software analysis tools to use, it is important to consider the entire software development process. From emulation and simulation in the pre-hardware phase to remote diagnostics after the product has shipped, thorough data streams are crucial.

Software analysis tools can provide this data at every stage of the cycle.

3.1 Development

Software analysis tools augment a debugger when stop mode is not an option. An Instruction Trace tool can provide performance measurements while clarifying RTOS interactions, context switches and performance bottlenecks. The most effective tools not only trace function pathways, but provide visibility inside the functions as well.

Software Trace tools are critical in dealing with failures. Often, with hard-to-detect faults, it is difficult to determine the string of events that led to the failure. After the fact, it is often impossible to reconstruct an identical situation. A snapshot of the system at the time of failure is often insufficient; a more complete history is needed.

Trace tools reveal all of the events that led to the failure. Hardware-based analysis tools can provide an advantage in most cases, as they collect and store trace data off the system. Therefore, if the system crashes or experiences an unexpected problem, there is significantly less risk of corruption or loss of trace data.

3.2 Tuning Phase

Just because the software is up and running, that does not mean it is ready to ship. It has to be tuned for optimum performance. Software analysis tools are an invaluable aid during this phase, when the application's performance is being measured and tweaked. There are two software analysis methods that are commonly used:

3.2.1 Instrumentation

By applying tags to specific lines of code, the instrumentation method offers extremely accurate measurements of application performance. However, the method does increase the size of the code. It is important to find tools that permit a tight focus on specific problems. There are hardware solutions available that have minimal impact on the final code.

3.2.2 Sampling Method

The sampling method is non-intrusive to code size, but it cannot offer the 100 percent accuracy of instrumentation. Sampling also requires some form of monitor, which drains system resources and impacts its performance, affecting the accuracy of the test.

3.3 Check-In/Hand-Off to QA

Software analysis tools streamline the transitions from design to development, from development to test and from test to deployment by identifying resolved and unresolved issues as they occur in the development cycle.

Unit tests must be developed to cover all new code. It is also important to check the code that calls new code to ensure it is functioning properly. There are many tools that can identify calls, but the challenge for any analysis tool is providing visibility into what is happening inside the function.

The test harness being used must be evaluated as well. It is important to note the level of coverage expected from the environment and ensure the test harness achieves that level of testing. In one instance, a company believed its test harnesses provided complete testing, but software analysis tools were able to prove they were in fact testing less than 30 percent of the code. Exception testing is essential to mission-critical code, and dead code stripping is an important by-product of the process.

3.4 Validation/Certification

Software analysis tools can provide objective metrics into code quality, giving project managers, engineers and marketing teams the data they need to feel confident in the quality of the system. The test harness can be measured to ensure 100 percent coverage. A by-product of a thorough validation process is the wealth of data needed for many certification programs. For example, aviation applications must receive Federal Aviation Administration (DO-178B) certification, which requires tests that show coverage of the code and called routines.

4. Types of Software Analysis

4.1 Instrumentation

This method of software analysis involves inserting markers within lines of code to measure the location and timing of executions. The obvious drawback is that this method increases the size of the code. However, some tools can be focused on a specific problem area, so you can start with low-level instrumentation and increase as needed. With a hardware solution, the impact is even lower, as most processors can handle the tags with just a one-cycle latency.

4.2 Code Sampling

Code sampling involves monitoring code as it runs and inspecting the PC at fixed intervals. It is not as accurate a method as instrumentation, since 100 percent accuracy requires prohibitively long test times. Another limitation is that code sampling does not catch instruction string or memory errors. In embedded systems, the method is essentially a faster and simpler version of a stop-mode debugger. The process is inherently intrusive.

4.3 Built-In SoC Capabilities

System-on-chip (SoC) systems present unique challenges. When a complete networking system is implemented within a single integrated circuit (IC), there are multiple modular components interconnected by a common infrastructure. All the various subsystems and blocks must be designed, verified and completed simultaneously, in parallel fashion. It is now possible to design SoCs with Instruction Trace capabilities, as well as cache monitors and performance timing information.

5. Summary

The proper use of software analysis can enhance the entire product development cycle and ease the transition between the phases. In industries where product performance and code quality are key, software analysis tools can make a significant difference.

The key to integrating software analysis tools into your development and testing process is choosing tools that fit your environment and the type of analysis you need to achieve. One industry-leading software analysis tool is Freescale's CodeTEST® Software Analysis Tool, and this tool is capable of addressing many of your software analysis needs. CodeTEST offers a flexible, scalable solution that provides consistency throughout your development process with a choice of software and hardware data collection methods so you can select the option that best meets your needs.

Another unique aspect of the CodeTEST Software Analysis Tools is the capability to provide several types of analysis with a single tool. CodeTEST offers software execution trace, memory analysis, performance profiling and code coverage analysis all within a single environment using a single data collection probe. Additionally, CodeTEST supports a broad array of hardware architectures, RTOSes and tool chains, so your analysis tools do not force you to change your development environment. CodeTEST is also portable across projects, allowing you to leverage your investment across numerous development efforts.

The key to integrating software analysis tools into your development and testing process is choosing tools that fit your environment and the type of analysis you need to achieve. These factors should be considered at the beginning of every project to ensure they are integral to the development process.

For more information about software analysis tools, read other Freescale white papers including:

> Beyond Profiling—Gaining Control of Software Performance

> Code Profilers: Choosing a Tool for Analyzing Performance

> CodeTEST Tool Qualification for DO-178B Qualification

> Using Performance Analysis Tools

Available online at http://www.freescale.com

freescale™
semiconductor