

New Trends in Real-Time Software Debugging

Using test points for monitoring signals is a basic part of hardware debugging. Continuously collecting and visualizing signals in real time from a working environment provides the foundation for understanding the system's behavior. Be it an electronic board, a data network or even the human body, the concept of monitoring a set of test points in real-time is the essence of trouble-shooting. SurroundView is a new software development tool, which facilitates the instrumentation of real time software applications with test points. It performs its tasks utilizing an agent-host architecture. SurroundView provides a highly efficient method for adding numerous monitoring test points within a real time target application, utilizing these test points to monitor, visualize and analyze the application's behavior while it is running.

MONITORING TEST POINTS

Instrumenting a real time application with monitoring test points provides a significant increase in testing and debugging capabilities. The test points serve the debugging process of an embedded application by providing the ability to visualize the application's execution, dynamically collect and record application data for on-line examination and for further off line analysis. The test points can also provide the means for assigning new values to the running application's variables, without neither halting its execution nor interfering with its natural execution flow. Monitoring test points can provide for observing the logical path of the application execution. They can also serve as a triggering mechanism, capable of triggering data collection based on user specified target events, thereby facilitating capturing of misbehavior functions of the embedded software application.

DEBUGGING TOOLS

Current debugging methods vary from one tool to another. Many currently available software development tools are utilizing the concept of test points to accomplish their task. These tools include emulators, debuggers, memory analyzers and profilers.

Memory analyzers instrument a software application by inserting test points in the locations where memory is allocated and it is being freed. Collecting data from these test points allows for understanding of the ways the application is utilizing its memory resources and for detection of memory leaks.

Profilers instrument Operating System's objects such as tasks and semaphores with test points for data collection and analysis. Profilers enable monitoring of the operating system's performance and provide tools for analysis of tasks' and functions' execution time.

Debuggers enable visualization of still snap-shots of the application's variable values and behavior at pre-defined debugging Break Points.

The capabilities that these software development tools provide are important and useful, but not good enough. All of these tools do not provide the essential features required for locating numerous bugs that are

related, and are the result of the dynamic behavior of a running software application.

Current emulators, debuggers, memory analyzers and profilers provide a very limited understanding of the application dynamic behavior, since they do not provide the essential capability of collecting and viewing application variables and application events in real-time.

PROBLEMS DEVELOPERS FACE

Embedded programmers are usually faced with both standard "desktop" bugs and with bugs that are related to the dynamic behavior of the application. These dynamic bugs include:

- Timing related misbehaviors
- Inter-process and inter-processor communication related bugs
- Hardware-software interface related bugs
- Algorithmic bugs
- Application resource leaks (beyond memory leaks)

Developers often try to tackle these problems by inserting simple test points that enable data logging. These "print" like methods allow the developer to collect application data from locations defined prior to compilation. Target resources usually limit the size and volume of the collected data in this approach. Therefore, it is usually possible to collect data over relatively short time periods and to monitor only few variables at any one-collection session. Due to these limitations, the data logging process is performed iteratively, by changing the test points in each iteration. A change in the test points requires going through a process of compile- link- load & run- collect- analyze, over and over again until locating the problem. This debugging method also requires the developer to develop custom special analysis tools to decipher the collected data.

Test points can be a reliable, minimally intrusive mechanism that provides a broad application level view of the target system provided they are coupled with a powerful, dynamic host visualization, management and analysis tool. SurroundView includes such a visualization, analysis and debugging environment.

INTRODUCING SURROUNDVIEW

SurroundView is a tool for monitoring and debugging embedded applications in real-time. It equips developers with a highly flexible visualization tool for monitoring the dynamic behavior of embedded applications and for debugging real-time software. Using SurroundView you can detect dynamic misbehaviors of your application, isolate the problematic variables and pinpoint to the erroneous code.

SurroundView can dynamically select application variables to be monitored, assign values to these variables, all in run-time without using any break points or halting the system.

SurroundView is based on source code instrumentation. Using a simple API, a developer can add powerful and efficient test points to the source code. Utilizing these test points, the user can continuously collect the values of all of the target application variables, trigger events and export text prints.

A typical application could include hundreds or even thousands of test points, depending on the complexity of the application and the number of its variables. Once the application is running, the developer can monitor all of the application variables or can dynamically choose to monitor a selected group of variables. Test points, which are not activated, consume negligible amount of CPU and no memory resources. Therefore, the developer is encouraged to leave the test points in the source code for future debugging use. Releasing an application with integrated test points provides a solid base for further product testing and customer support at later stages.

ARCHITECTURE

SurroundView is based on an Agent-Host architecture. The Agent is a low priority task that provides reliable and effective communication with the host. The Agent is responsible for collecting data from the application's test points, packing the collected data and sending it to the host. It also receives data from the host for depositing values to application variables and receives user commands for activating and/or deactivation test points. The host is a Windows application running on a PC workstation. Its tasks are to receive data from the target application via the agent, record the data to a history-file, analyze the incoming data and continuously display raw and processed data through a set of powerful visualization tools. The SurroundView host has the capabilities of replaying the recorded data for off-line analysis.

The target/agent architecture is depicted in figure 1. When the execution of the target application reaches a test point, it checks whether the user had activated this particular test point via the host. If activated, data at this test point is copied from the application to a cyclic buffer on the target and the application execution is permitted to continue. The agent, working as a low

priority task, collects the data from the cyclic buffer, packs it and sends it to the host via the target-host connection. This architecture provides minimal intrusiveness to the target application and does not interfere with the application's real-time execution. The agent task is also responsible for depositing data arriving from the host into the application, to assign values to the application variables.

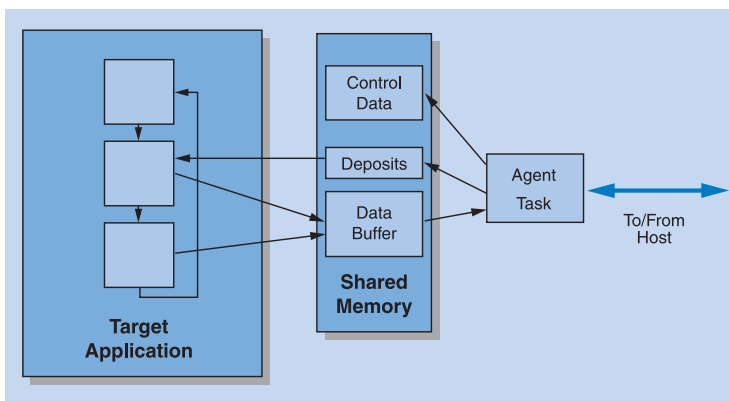


Figure 1. Agent Architecture

SurroundView host provides the means for managing the target application data collection, displaying and analyzing the collected data on the host side of the system. The SurroundView host supports connections to multiple targets simultaneously, where each target can be a different processor running a different RTOS. The host enables dynamic activation and/or deactivation of target test points by the user through a graphical filtering mechanism.

SurroundView allows for automatic instrumentation of the target application source code with test points. The instrumentation utility provides a static view of the target application hierarchy from the system level, through the source files and down to the test points. It allows for adding prints, events or data test points to selected source code functions, according to user specifications.

SurroundView analyzes image files before they are downloaded to a target and extracts the debug information. This enables a symbolic user-interface where the developer can symbolically specify what data to collect and display.

APPLICATION VISUALIZATION

The usage of test points may best be exploited with powerful visual tools. SurroundView's visualization environment providing a rich variety of visualization capabilities to analyze and display the data as can be viewed in Figure 2. Supported display formats include charts of various types, structure/class views, table comparison views and event log and graph views. Users can select a different display format for each monitored variable and can simultaneously view data from multiple sources as this data is synchronized to provide a consistent view of the system.

SurroundView enables triggering of various actions upon the state and value of the collected data. Such

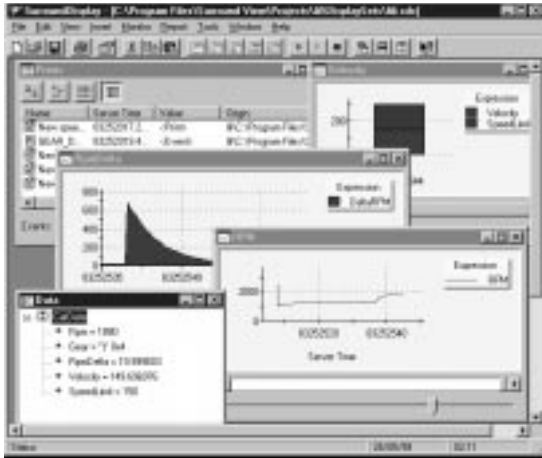


Figure 2. Visualization Capabilities

actions include alerting on a value change through a graphic representation (chart coloring), beeping, e-mail initiation, etc. Triggering may be based on complex mathematical and logic conditions. The definition process is extremely short, simple and performed symbolically during run time. An expression editor is available to perform mathematical and statistical expression evaluations. These expressions may be displayed in all available display formats.

The recorded data may be replayed at any given time to provide a systematic examination of the target behavior. Since the user's ability to monitor the system's behavior in real-time is limited, the replay feature allows both a step-by-step view of the target variables values; and a broad graphic presentation of the collected data. All analysis capabilities are available in off-line mode to enrich the possibilities of data examination.

EXAMPLE

Figure 3 depicts a SurroundView trace of a cruise control application. This application controls a vehicle's RPM (Revolutions Per Minute) and the correct gearbox setting necessary to establish the required velocity settings commanded by the driver. The brown line in the upper graph of figure 3 represents the commanded velocity settings and the blue line the actual velocity. It is evident that after a decrease in the commanded velocity, the actual velocity slowly decreases to match the new required velocity. The lower graph represents the car's actual RPM. A decrease in velocity is achieved by the cruise control system lowering the RPM. At a certain time (83249893), the RPM is decreased below 1000, so a change in gear occurs to keep the engine from stalling. After the gear is lowered an immediate raise in RPM occurs in order to keep the velocity at the commanded level. We designated the engine's allowed RPM range to be limited to 2500 RPM. Since the curve picks above that level, SurroundView immediately responds to the trigger and notifies the user that the cruise control algorithm forced an illegal scenario.

Note: the values of the RPM and velocity variables are monitored simultaneously and are synchronized, while the application is running.

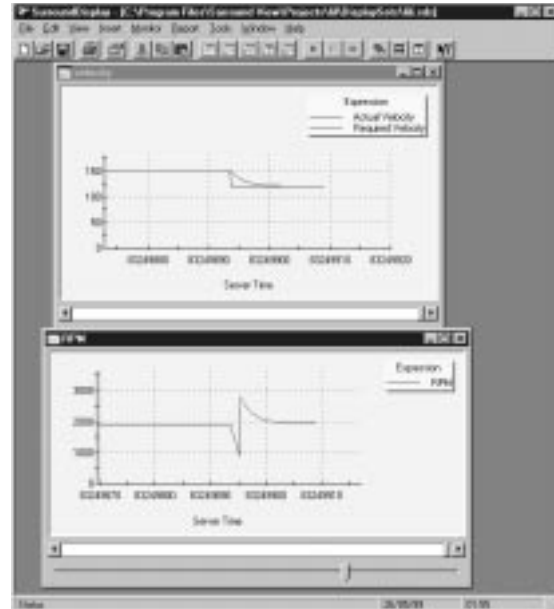


Figure 3. Cruise Control Example

SUPPORTED PLATFORMS

SurroundView Agent is available for a variety of platforms including Windows NT, pSOS (Integrated Systems), Nucleus PLUS (Accelerated Technology), OSE (ENEA) and Windows CE. Call to inquire about support for additional RTOS.

CONCLUSION

Embedded software instrumentation with dynamically controllable test points provides essential debugging capabilities. The growth in complexity in embedded applications is turning the usage of test points to standard practice for the embedded software industry. Test points provide an infrastructure for comprehensive run-time debugging, testing and customer support for embedded software applications. SurroundView is a unique tool, which is tuned to function in demanding embedded applications and is optimized for minimal affect on the run-time target application.

Detecting, locating and correcting dynamic time-related bugs is one of the latest stages of product development. These tasks are to be performed under extremely demanding time constraints. SurroundView provides the means to accomplish these tasks in significantly shorter time and less effort than traditional debugging tools. SurroundView enables shorter time to market, reduced costs, improved product reliability and provides an infrastructure for efficient system diagnostics and customer support. ■

Eldad Maniv is the founder and CEO of RTview Ltd. He has previously managed large scale embedded projects in the Israeli Aircraft Industries heading a team of 100 engineers in a \$50M development project. Mr. Maniv was involved in developing navigation, signal processing, flight control and electro-optical systems. He received his bachelor's degree in Physics, Mathematics and Computer Science in 1989 from the Hebrew University in Jerusalem. He may be contacted at eldad@rtview.com.