

Ashling Software Engineering Tools: Overview

Contents

- 1. Introduction.....1
- 2. Real-time analysis using PathFinder and Vitra1
 - 2.1 Timing Performance Analysis2
 - 2.2 Call-Return Trace2
 - 2.3 Code Coverage Measurement in PathFinder.....3
 - 2.4 PathFinder Real-time analysis measurement capacity.....3
- 3. PathTrace Real-Time Software Quality Assurance System option.....4
 - 3.1 PathTrace Performance Analysis4
 - 3.2 Extended Code Coverage Measurement using PathTrace4
 - 3.3 PathTrace measurements5
 - 3.4 PathTrace data-capture modes5
 - 3.5 Regression Testing.....5
 - 3.6 PathTrace Measurement capacity.....5
 - 3.7 Interface to Software Quality Analysis tools.....6
 - 3.8 PathTrace Report Generator Toolkit6
 - 3.9 PathTrace Remote Execution Access Server7
 - 3.10 PathTrace Language support.....7
 - 3.11 System Requirements7
- 4. LDRA Testbed Software Quality Assurance framework7
 - 4.1 Testbed Product Summary.....7
 - 4.2 Static Analysis using Testbed7
 - 4.3 Dynamic Analysis using Testbed.....8
 - 4.4 Software Standards Conformance.....8

1. Introduction

Ashling supplies an expanding range of tools for Software Engineering, Software Test and Software Quality Assurance in performance-critical and safety-critical embedded software development. This document summarizes key-features of Ashling’s tools range.

2. Real-time analysis using PathFinder and Vitra

Ashling’s Vitra Networked Emulators and PathFinder Source Debugger include a set of powerful analysis and display functions for software analysis.

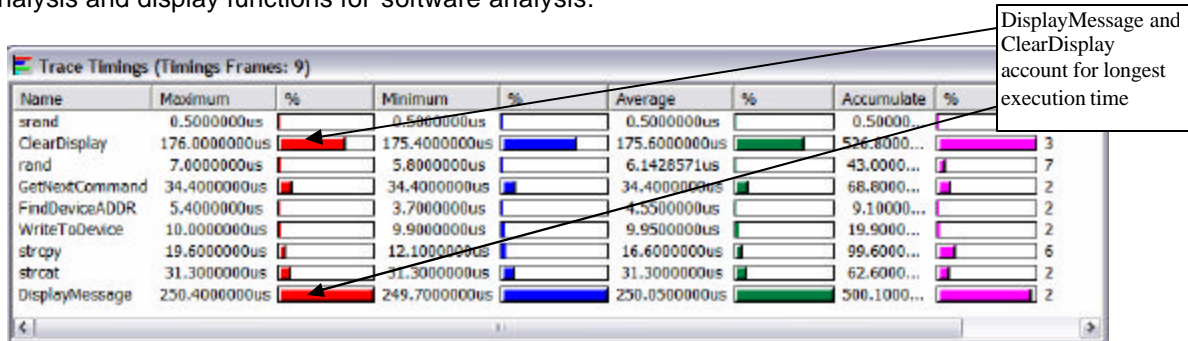


Figure 1: The Timing Performance Analyzer display in the PathFinder Source Debugger

Call-Return Trace and **Timing Performance Analysis** are included in:

- PathFinder-5500 Source Debugger for PowerPC MPC5554;
- PathFinder-PPC Source Debugger v1.0.7 and later, for PowerPC MPC56x series;
- PathFinder-TriCore Source Debugger v1.1.4 and later, for TriCore TC1765, TC1775, TC1776 and TC1796;
- PathFinder-ARM Source Debugger v1.1.2 and later, for ARM 7 and ARM9 cores.

Code Coverage Measurement is included in:

- PathFinder-5500 Source Debugger v1.0.1 and later, for PowerPC MPC5554;
- PathFinder-PPC Source Debugger v1.1.0 and later, for PowerPC MPC56x series.

Upgrading an existing system to add real-time analysis just requires a PathFinder update download; running PathFinder will automatically update the FPGA firmware in the Vira Networked Emulator.

By utilizing the Vira Emulator's non-intrusive capture of real-time trace data using the microprocessor's trace port, the PathFinder Debugger displays performance analysis data in real-time while the target systems is executing; it does not interrupt or slow-down execution, and it requires no target system resources.

2.1 Timing Performance Analysis

The Trace Timing Performance Analyzer window shows detailed timing information at C function level for all captured code trace information in the Vira trace buffer (see Figure 1).

All occurrences for the functions captured in the in the buffer are measured and their timings are compared in the display. The columns in the display window show:

Name	Function name.
Maximum	Maximum recorded execution-time for this function.
Minimum	Minimum recorded execution-time for this function.
Average	Mean execution-time for this function, averaged over all recorded occurrences.
Cumulative	Total execution time for this function, accumulated over all recorded occurrences.
Count	Total number of recorded executions for this function.

All of the graphs in the % columns are displayed relative to the *largest* value for that column; for example, the function with the largest **Maximum** execution time is shown as 100% and the maximum execution-time for all others functions are shown as percentages of the largest.

2.2 Call-Return Trace

The Call-Return Trace window displays real-time execution of the program-under-test as a sequence of high-level function calls and returns.

Frame	Time	Type	Address	Symbol
00000723	211.1000000us	->	4000055C	rand
00000744	218.4000000us	<-	400005A4	rand
00000749	218.9000000us	<-	4000053C	GetNextCommand
00000763	222.3000000us	->	4000023C	WriteToDevice
00000775	224.7000000us	->	40000288	FindDeviceADDR
00000797	230.5000000us	<-	400002E8	FindDeviceADDR
00000807	232.7000000us	<-	40000284	WriteToDevice
00000812	234.8000000us	->	400002EC	DisplayMessage
00000820	236.8000000us	->	4000040C	ClearDisplay
00001440	414.8000000us	<-	40000458	ClearDisplay
00001453	416.9000000us	->	40000650	strcpy
00001521	430.5000000us	<-	400006D8	strcpy
00001532	433.2000000us	->	400005B4	strcat
00001593	444.9000000us	->	4000065	strcpy

Figure 2: The Call-Return Trace display in the PathFinder Source Debugger

The columns in the display window (Figure 2) show:

Frame	Position of this Call or Return Event in the Vitra Trace Buffer.
Timestamp or Cycle-count	Shows a Timestamp (relative to the start of tracing) for this call/return event. (You can select Timestamp or cycle-count in the Trace Trace Window Display dialog).
Event Type	→ Indicates a function Entry; ← indicates a function Exit.
Address	Shows the code address of the function Entry or Exit event.
Symbol	Shows the Name of the function.

The Call-Return Trace window shows you an exceptionally clear and easy-to-understand real-time record of the sequence of function-calls in the program-under-test.

2.3 Code Coverage Measurement in PathFinder

PathFinder Code Coverage (see Figure 3) identifies and displays all tested, untested and partially-tested source lines, function and modules in the program under test. Summary and detailed reports are provided on depth of code test coverage by function, module and program.

All unused or unreachable code fragments are identified. PathTrace Code Coverage identifies branches that require additional test-cases so as to progress towards 100% code test coverage.

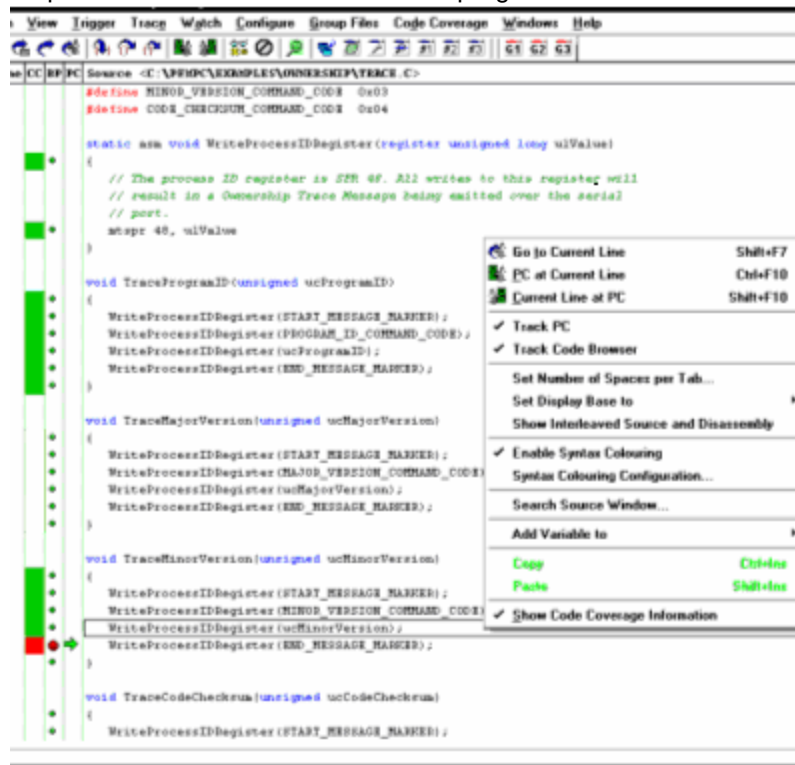


Figure 3: The real-time Code Coverage display in the PathFinder Source Debugger; source lines with a Green block marker show all Tested lines of code

Code Coverage information is shown in both the Source and Disassembly (reverse-assembly) windows. You can select any module, function or address range in your program and view coverage statistics for that unit; you can save Code Coverage data to disk after each test session and thus build up a test-history over a wide range of test conditions and stimuli.

2.4 PathFinder Real-time analysis measurement capacity

Real-time storage capacity of PathFinder real-time analysis is determined by the Vitra Emulator's Trace Memory capacity with which it is used; with Vitra-T512K (512K-Frames memory option), PathTrace captures 512K executed branches or returns with no intrusion on real-time execution and no software-instrumentation.

3. PathTrace Real-Time Software Quality Assurance System option

PathTrace is an optional software add-on for Ashling's PathFinder Source Debugger and Vitra Networked Emulator with Trace that captures, stores, analyzes and reports on the executed performance of a real-time embedded application.

PathTrace expands the program-analysis features in the PathFinder Source Debugger by adding Selective and Global capture modes for Timing Performance Analysis, Code Coverage Measurement and Call-Return Trace; a Report Generator Toolkit for customized reports; user-defined reports; and an interface to software analysis tools.

The hardware-interface to the executing target-under-test uses the Ashling Vitra Universal Emulator and Trace.

PathTrace integrates with the leading static and dynamic software analysis environments to capture real-time data from the target embedded system.

3.1 PathTrace Performance Analysis

PathTrace Performance Analysis measures real-time code execution performance, identifies functions or paths for timing-optimization, and measures user-specified critical timing paths that can consist of one or more functions modules or code fragments.

The PathTrace Performance Analyzer is an invaluable tool for:

Optimization: You can identify at-a-glance the most time-consuming functions in your program; you can then focus on optimizing the code in these functions to reduce your overall system execution-time or response-time.

Verification: You can confirm that the Minimum, Maximum and Average execution-times for each function conform to the requirements of your specification.

System-integration: You can identify 'trashing' or 'lockout' conditions, where your program gets stuck in a repeated loop; 'runaway' functions (excessive executions); Initialization functions that were executed more than once; or functions that called themselves.

Execution testing: You can identify functions that *were* or *were not* executed during your test session.

3.2 Extended Code Coverage Measurement using PathTrace

PathTrace extends the capacity and capability of PathFinder's Code Coverage by adding options for Selective measurement, by Module, Function or Routine.

All unused or unreachable code fragments are identified. PathTrace Code Coverage identifies branches that require additional test-cases so as to progress towards 100% code test coverage.

3.3 PathTrace measurements

PathTrace provides interactive and hard-copy Displays and Reports on:

- Code Coverage:** Execution coverage map for a test session on the software function, module or program under test.
- Cumulative Code Coverage:** Execution coverage map, accumulated over a series of different test sessions, on the software unit (function, module or program) under test.
- Function Trace:** Execution Trace of software under test, showing time-stamped history of function calls and returns, interrupt executions and returns, and exception executions and returns.
- Performance Analysis** Graphical and text display of minimum and maximum execution-time of functions-under-test, allowing measurement, verification, optimization and documentation of timing performance.

3.4 PathTrace data-capture modes

PathTrace can capture and analyze execution data in either of two Modes:

Global Capture Mode:

PathTrace captures *all* branches and returns in the program-under-test. Global Mode adds no software-instrumentation to the code-under-test. It is suitable for testing short test sessions; however, its real-time execution capacity is limited because it is not selective: capturing execution of all branches and returns in all functions will cause the trace capture memory to fill-up over a relatively short execution session.

Selective Capture Mode:

Using PathTrace's Program Structure Dialog, the user can select a set of Modules, Functions or Assembler-routines for analysis; **PATHTRACE** automatically adds software-instrumentation instructions to all entry- and exit-points in the selected set at a 'C' level requiring that the application be recompiled and linked. Selective Mode is suitable for testing longer or more complex test sessions, because it captures execution data for the selected entry- or exit-points only; thus, a given size of trace memory can capture a longer execution session before pausing to upload the data to the host PC.

PathTrace software-instrumentation causes a very low level of intrusion on the program under test (just ten assembly instructions per monitored function). Execution of this instrumentation code is predictable; it causes no wait-states or interrupts. Because it has negligible impact on execution of the application (other than a very minor and predictable increase in code-size and execution-time), the software-instrumentation code may be safely left in the final released code build.

3.5 Regression Testing

All PathTrace measurements and reports can be stored and executed as Scripts, thus providing automated, repeatable regression testing. Measurement and display scripts can detect timing differences and verify timing conformance for a new software release, by reference to measurements on the previous software build.

3.6 PathTrace Measurement capacity

Real-time storage capacity of PathTrace is determined by the Vitra Emulator's Trace Memory capacity with which it is used, and by the choice of Global or Selective Capture Mode:

- In *Global Capture Mode* with Vitra-T512K (512K-Frames memory option), PathTrace captures 512K executed branches or returns with no intrusion on real-time execution and no software-instrumentation.
- In *Selective Capture Mode* with Vitra-T512K (512K-Frames memory option), PathTrace captures 512K executed events (usually entry- or exit-points) using software-instrumentation on the selected software events.

To analyze execution-times that exceed the capacity of the Vitra trace-buffer memory, PathTrace can be instructed to either:

- Halt data-capture and continue execution after the trace memory capture-limit is reached, or
- Briefly pause execution while the captured data is uploaded to the host PC, and then resume real-time execution and data capture. This allows effectively unlimited capture of execution data.

3.7 Interface to Software Quality Analysis tools

The PathTrace Remote Execution Access Server interface is used to interface to LDRA Testbed™ and other leading Static and Dynamic software analysis environments. PathTrace supplies real-time data to the analysis application from the target embedded system-under-test.

3.8 PathTrace Report Generator Toolkit

PathTrace graphically shows the captured information in windows within the PathFinder user interface. In addition, PathTrace saves finished Reports or raw Data in HTML, CSV, Excel (.XLS) and plain-text formats.

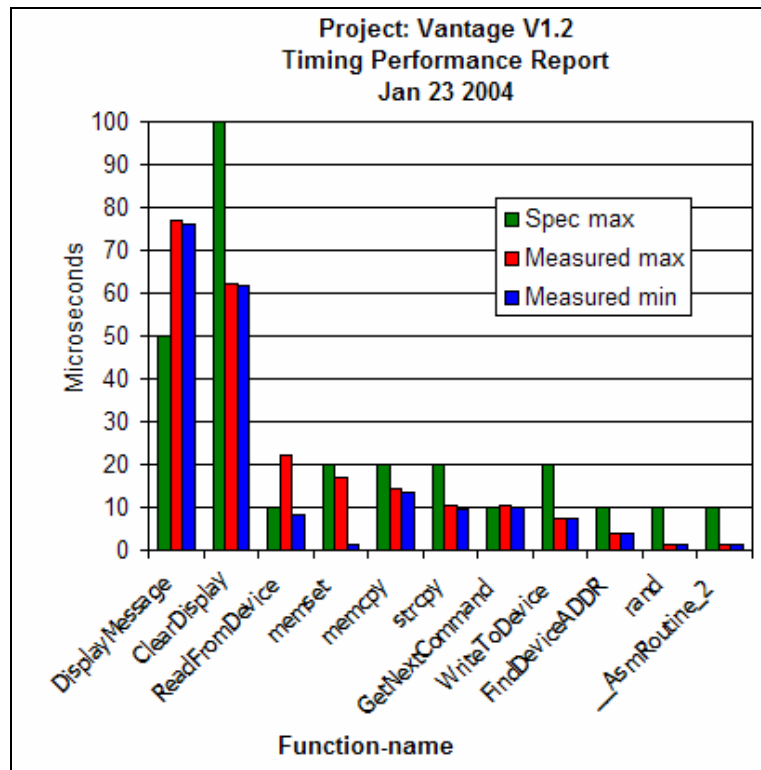


Figure 4: Using the Report Generator Toolkit you can create customized Charts (left) and Reports within Microsoft Excel®. This example compares the Minimum and Maximum function execution-times recorded by PathTrace with the specification's limits

The Report Generator Toolkit allows external data-processing applications (such as Microsoft's Excel Spreadsheet and Access Database) to access execution data from the system-under-test. Thus, users can generate customized graphs, reports, and analyses from the real-time execution data collected by the Vitra Emulator and Trace.

The Report Generator Toolkit can create test and V&V Reports in accordance with specific project requirements and standards, and can present graphical and text display of execution path, code coverage and timing performance

3.9 PathTrace Remote Execution Access Server

The PathTrace Remote Execution Access Server allows you to control execution of your embedded target and to access its real-time execution data from external applications.

You can integrate PathTrace with LDRA Testbed™ and other leading static and dynamic software quality assurance environments. You can also use the Remote Execution Access Server to back-annotate a high-level design representation (in state charts or UML, for example) with real-time execution data.

3.10 PathTrace Language support

PathTrace includes built-in displays of software timing, execution path and code coverage, annotated by reference to the program's C or assembler source code.

In Global Capture Mode, PathTrace captures execution of *all* branches (that is: all branch and return instructions and all interrupt-service and processor-exception-handling execution) throughout the application's code. PathTrace can therefore provide Code Coverage maps for both C and Assembly code, and can allow (for example) comparison of the C-level with the native-code execution-level coverage maps so as to verify that they are in agreement.

In addition, the Remote Execution Access Server provides an interface between PathTrace's real-time execution data, and alternative high-level design representations such as state charts, SDL, UML, Ada or other high-level design representations or languages.

3.11 System Requirements

PathTrace is an optional software extension to Ashling's PathFinder-PPC Source Debugger. Ashling's Vitra Networked Emulator and Trace provides the hardware interface to the executing real-time target-under-test. PathTrace runs on PCs with Windows XP/NT/2K/9x/Me.

4. LDRA Testbed Software Quality Assurance framework

Ashling distributes and supports LDRA Ltd.'s "Testbed" software analysis framework and "TBrun" automatic test harness generator.

4.1 Testbed Product Summary

Testbed is a unique quality assurance tool that provides powerful source-code testing and analysis facilities for Verification and Validation of software applications. It is an essential tool where software is required to meet measurable, documented criteria for the software-development and -test process, and its use brings substantial time, cost and efficiency savings.

LDRA Testbed's powerful analysis facilities may be applied in the two main testing domains of Static Analysis and Dynamic Analysis.

- *Static Analysis* analyses the source-code, provides an understanding of the code structure and ensures conformance with in-house-, industry- or customer-mandated standards for coding or language conformance.
- *Dynamic Analysis* executes the application-code on the target hardware using the Ashling PathTrace debugger option and the Vitra Emulator with Trace to capture execution data, compile execution history, define and measure Actual versus Expected test results, and pinpoint defects at run time.

4.2 Static Analysis using Testbed

Testbed's Static Analysis enables a program management team to ensure that a uniform set of programming standards are enforced, software is properly structured and complexity and other quality attributes are controlled within a configurable quality model. Static Analysis can also detect a significant number of software defects.

Testbed's Static Analysis is approved for the analysis of safety-critical code by many regulatory authorities.

4.3 Dynamic Analysis using Testbed

Dynamic Analysis uses test data sets to execute software in order to observe its behavior and produce test coverage reports. This assessment of source code ensures consistently high levels of quality assurance and correct use of regression-test and capture/playback tools.

LDRA Testbed Dynamic Analysis provides the facilities to achieve quality standards for critical code, improve code efficiency, minimize regression test costs, and detect software defects. When used during software development and maintenance, Dynamic Analysis techniques make a significant contribution to a program's robustness and reliability.

Dynamic Analysis is particularly effective for the analysis of software applications that are required to achieve high levels of reliability. It is the primary requirement for the testing of safety-critical avionics software and is widely used in all military, safety and mission critical software.

Testbed integrates with Ashling's PathTrace real-time software analysis product to capture data from the target embedded system-under-test using the Ashling Vitra Networked Emulator and Trace.

4.4 Software Standards Conformance

Testbed provides documentation and support towards conformance to the following standards, among others:

- DO-178B (Levels A, B & C)
- DEF STAN 00-55 (Static & Dynamic)
- MISRA C

In addition to the above any user-definable model can be configured based on recognized standards, against which all source code analysis can be compared.

Ashling Microsystems, Inc.

18612 Devon Avenue
Saratoga, CA 95070
Tel: (408) 884 3020
Fax: (408) 8843026
Email: support.usa@ashling.com

Doc: APB188-V3U