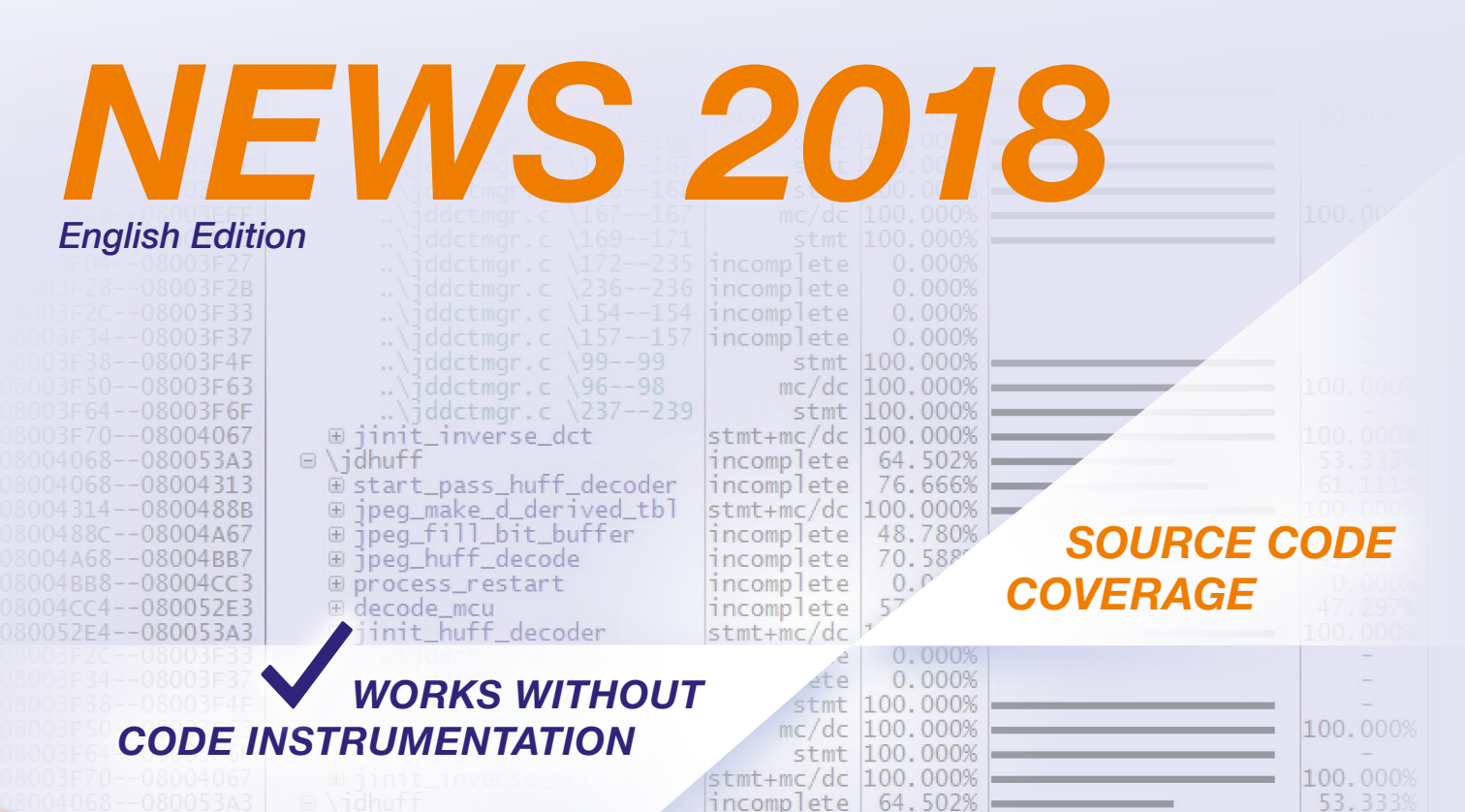
NEWS 2018

English Edition



SOURCE CODE COVERAGE

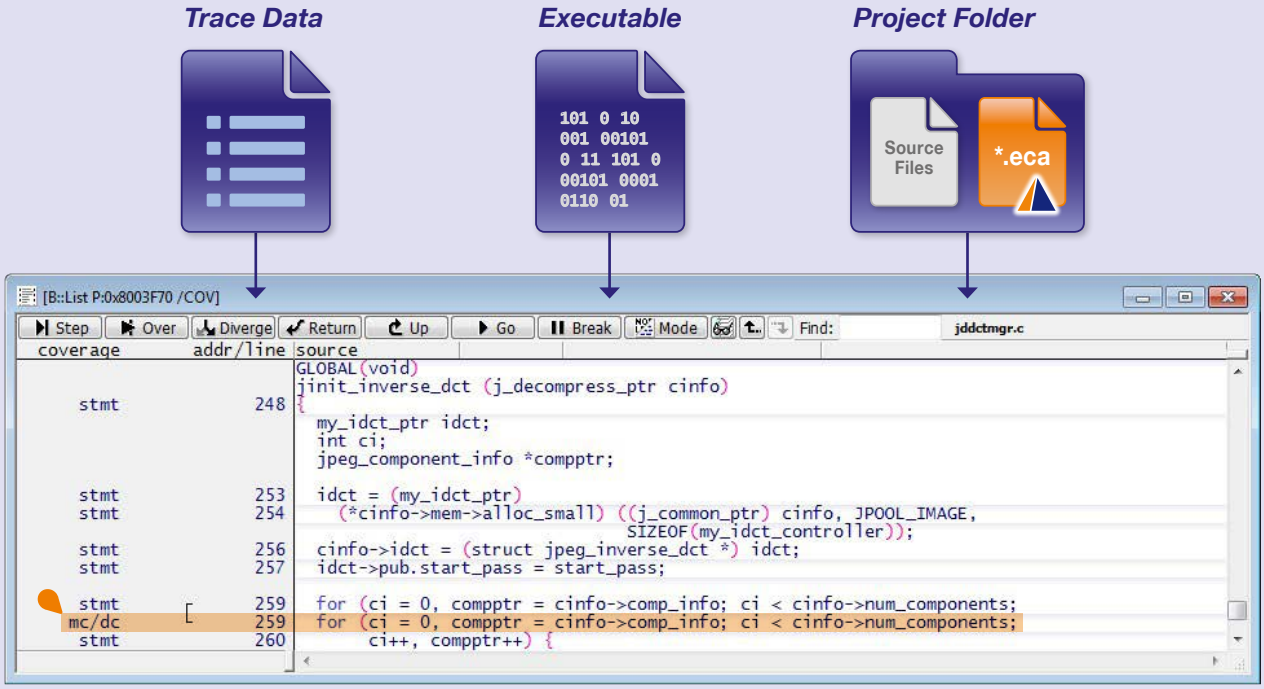
WORKS WITHOUT CODE INSTRUMENTATION

CONTENTS

LAUTERBACH
DEVELOPMENT TOOLS

Trace-based MCDC Coverage

MCDC Coverage



In March 2018, Lauterbach will unveil its trace-based MCDC coverage. TRACE32 now supports all important code coverage metrics at the source code level.

Definitions According to DO-178C [3]

Statement Coverage: Every statement in the program has been invoked at least once.

Decision Coverage: Every point of entry and exit in the program has been invoked at least once and every decision in the program has taken on all possible outcomes at least once.

Modified Condition/Decision Coverage: Every point of entry and exit in the program has been invoked at least once, every condition in a decision in the program has taken all possible outcomes at least once, every decision in the program has taken all possible outcomes at least once, and each condition in a decision has been shown to independently affect that decision's outcome. A condition is shown to independently affect a decision's outcome by: (1) varying just that condition while holding fixed all other possible conditions, or (2) varying just that condition while holding fixed all other possible conditions that could affect the outcome.

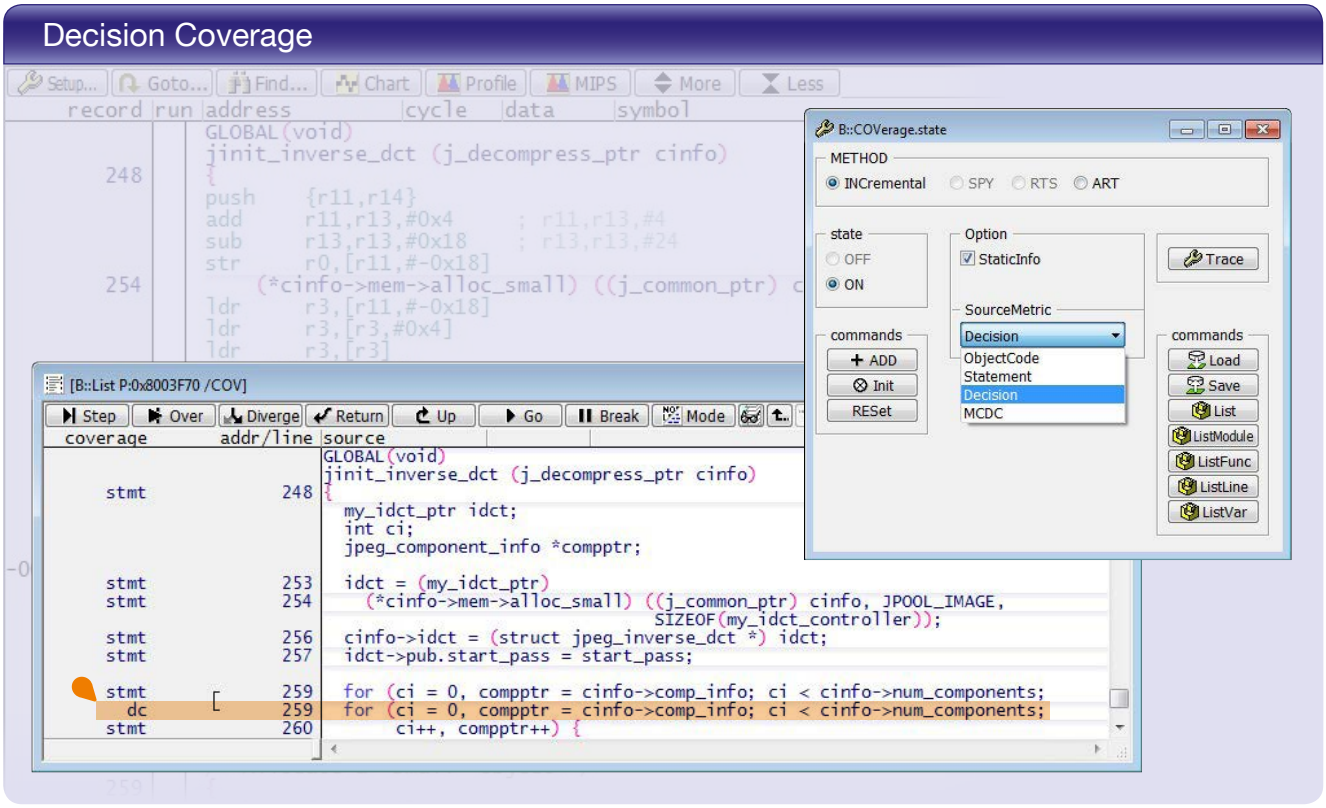
During the development of safety-critical systems, code coverage processes are used in order to demonstrate that the software was tested thoroughly and comprehensively. Software development standards such as ISO 26262 and DO-178C have stipulated that proof of code coverage is a mandatory part of the development cycle.

Trace-based Coverage

As a market leader in real-time trace tools, Lauterbach provides trace-based code coverage measurements which do not require any instrumentation of the target code. The information on the program execution is at first tracked at object code level. This allows the following coverage metrics to be easily proven:

- **Object statement coverage** proves that each assembler instruction has been executed at least once during the test.
- **Object branch coverage** proves that each conditional jump was taken at least once and that it was also not taken at least once.

Code coverage measurements at object code level have always been part of the capabilities of all



TRACE32 trace tools. With the addition of statement and decision coverage in February 2017, Lauterbach tools also provide proof of source code level coverage (see “Decision Coverage” figure on the top of this page). Many customers now also want to use their TRACE32 trace tools for performing the MCDC coverage measurements which are increasingly being mandated.

MCDC Code Coverage

Previously, it was the opinion of many that the proof of object branch coverage was an adequate replacement for the MCDC coverage measurement at the source code level. However, in the aviation industry, the Commercial Aviation Safety Team (CAST) has shown a clear opposition to this view (refer to [1]).

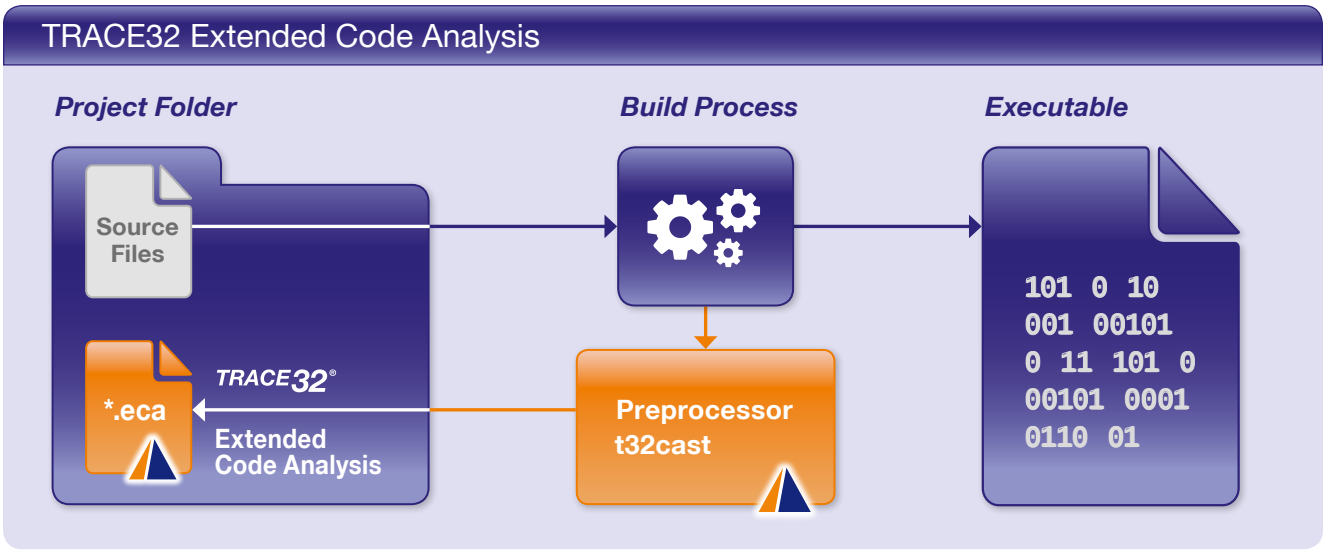
Currently, most people rely on source code instrumentation in order to be able to collect MCDC coverage data. The Lauterbach engineers set themselves the goal of providing MCDC coverage without having to alter the target code in any way. They studied many white papers and publications available on this subject. Each champions its own approach; however, across all of them, there are some fundamental similarities:

1. In order to ensure that MCDC coverage can be proven based on trace data, the structure and the position of a decision within the source code must both be known.
2. At the same time, each condition in the source code must be represented at the object code level by a conditional jump or by a conditional instruction and not by an arithmetic representation of the condition.
3. When the MCDC coverage analysis is performed, the structure and the position of a decision within the object code must be known.

For the implementation of an MCDC coverage measurement without code instrumentation, according to requirement #1, additional information about the source code structure is required. This is currently not part of the debug information generated by the compiler. In addition to this, it must be ensured that the compiler generates the object code in such a way that it fits requirement #2.

AdaCore

MCDC coverage can be proven easily with a trace recording if the compiler and the software performing the MCDC coverage analysis are from the same



provider. The compiler can include the new required information on the structure and the position of each decision within the source code into the debug information. The company AdaCore (refer to [2]) offers such a solution. In addition to this, AdaCore also offers a target emulation solution for generating trace data.

For customers who have to additionally prove an MCDC coverage for the final implementation on the target hardware, AdaCore offers an interface which allows trace data recorded with TRACE32 to be imported for the required analysis.

t32cast

As of March 2018, Lauterbach customers can also prove an MCDC coverage in TRACE32 based on a real-time trace recording. Lauterbach offers the t32cast command line tool for this purpose, which analyzes the C/C++ source code. As a result, the information on the decision structure required for the execution of the MCDC coverage measurements is generated for each source code file. Consequently, the build process must be adjusted in order to allow this information to be generated (refer to "TRACE32 Extended Code Analysis" figure on the top of the page). The t32cast command line tool is compiler-independent and it can be easily integrated into existing build environments.

As soon as the user starts the MCDC coverage measurement in TRACE32, TRACE32 automatically loads all required .eca files, generated by the t32cast tool. Subsequently, TRACE32 is able to map the decisions on source code level to the object code with the help of the debug information.

However, during this process, the user must ensure that the selected compiler represents each condition in the source code by a conditional jump or by a conditional instruction at the object code level, e.g. by disabling optimizations.

Conclusion

The TRACE32 MCDC coverage can be used independently from the compiler and the processor architecture. For those who can't abandon code optimization, even when implementing safety-critical systems, there is no way around reducing the optimization for the trace-based MCDC coverage.

In the future, it would hold significant value, if compilers could be configured so that, at the object code level, decisions are translated into conditional jumps or conditional instructions only, independently from the optimization level selected, and that optimizations selected are performed entirely for all remaining areas.

References

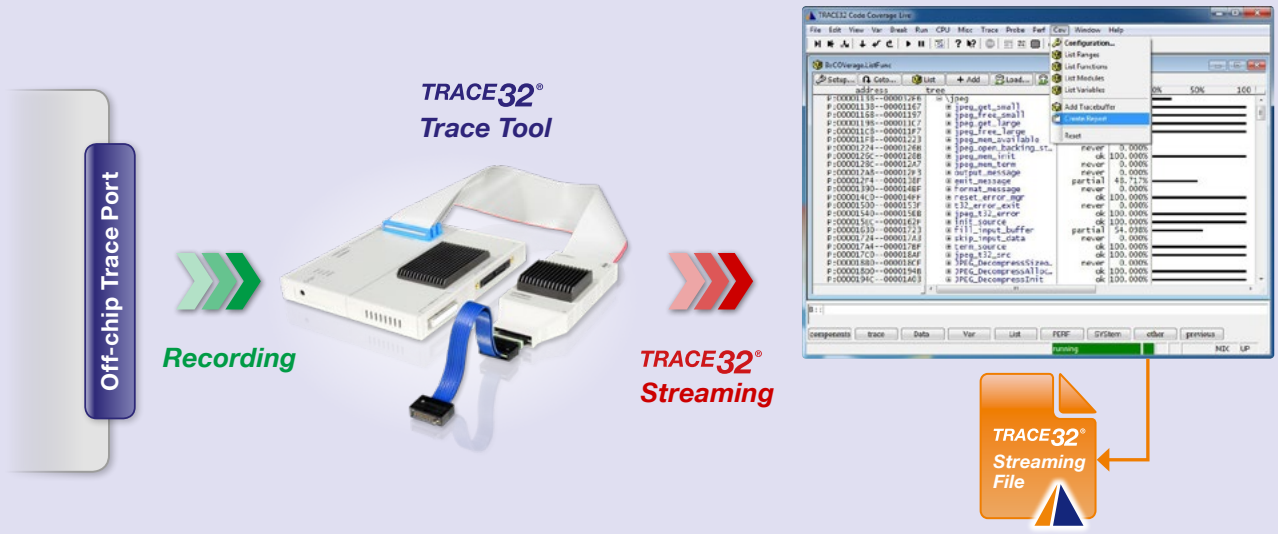
[1] CAST-17 Position Paper (2003, January). Structural Coverage of Object Code

[2] Comar, C., Guitton, J., Hainque, O., & Quinot, T. (2012, May). Formalization and comparison of MCDC and object branch coverage criteria. In ERTS (Embedded Real Time Software and Systems Conference).

[3] RTCA Inc. (2011, December) RTCA/DO-178C Software Considerations in Airborne Systems and Equipment Certification

Code Coverage Live

Real-Time Processing of Trace Data



Real-time Profiling (RTS) is the name given by Lauterbach to the trace mode in which trace data is streamed to the host during recording and analyzed there immediately. This allows the results of the code coverage analysis to be followed live on the screen. At the end of 2017, the RTS mode, which could already be used for Arm-ETMv3 since 2009, was rolled out on further trace protocols. The “RTS - Supported Trace Protocols” table shows an overview of all currently supported trace protocols.

Basic Conditions

The following basic conditions apply to all supported trace protocols:

1. The program code is required for decoding trace data. Given that it would be too slow to read the code from the memory during program execution, the code must be loaded into TRACE32 before starting the live analysis. This means that a live analysis can be performed for static programs only.
2. The live analysis of trace data works only in cases where the average data rate at the trace port does not exceed the maximum data transmission rate to the host computer. Given that current TRACE32 trace tools are equipped with a USB3 interface, the maximum data transmission rate to the host computer is around 180 MByte/s. The transmission rate has tripled compared to data rates that were available in 2009.
3. The live analysis of trace data can currently be performed for single core and for SMP multicore trace streams. The implementation for AMP multicore trace streams is currently still in the development phase.
4. The live code coverage measurement can be used for the metrics Object Statement Coverage and Object Branch Coverage as well as for Statement Coverage and Decision Coverage.

Generally, trace data is no longer required after it have been analyzed. However, TRACE32 offers the option to save the trace data during the analysis into a streaming file. This allows the trace data to be verified one more time, in a detailed way, even after the code coverage measurement has been completed. Just like conventional TRACE32 code coverage measurement, RTS mode allows comprehensive test reports to be created.

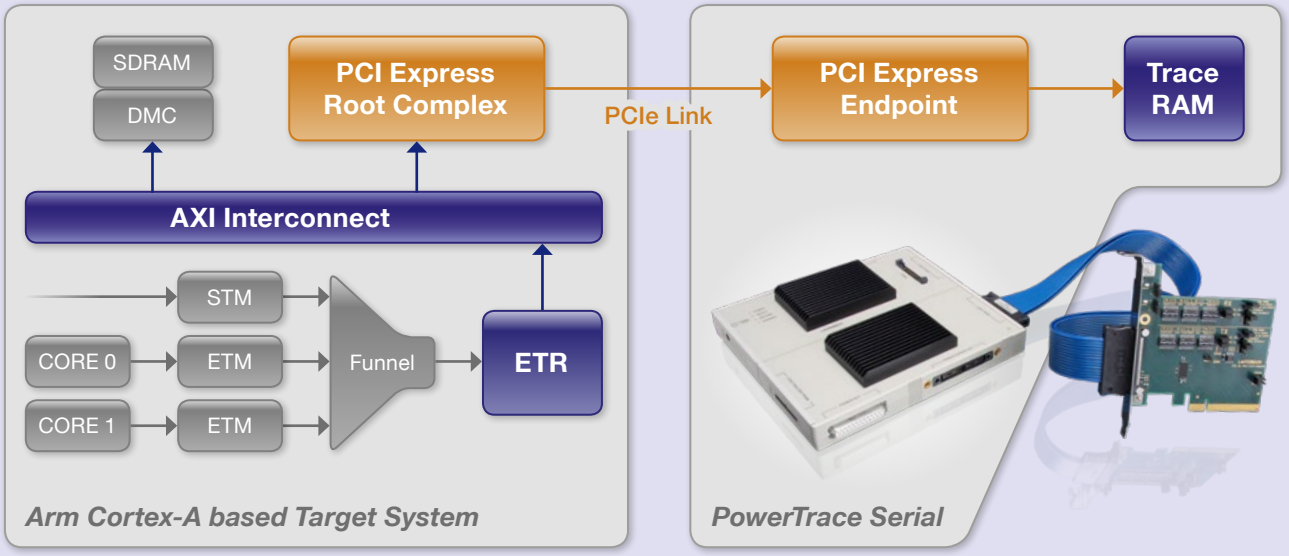
RTS – Supported Trace Protocols

- ETM v3 on Arm / Cortex®
- PTM on Arm / Cortex®
- ETM v4 on Arm / Cortex®
- MCDS on Infineon TriCore™
- Nexus on NXP MPC5xxx / STM SPC5xx
- Nexus on NXP PPC QorIQ®

others on request

Tracing via PCI Express

TRACE32 PowerTrace Serial as PCIe Endpoint



When a chip is equipped with a PCIe interface, it is possible to use it for recording and analyzing trace data with an external trace tool. Some early adopting Lauterbach customers already use this trace technique for Arm Cortex-A® or NXP Power Architecture-based QorIQ® processors.

Trace Tool as a PCIe Endpoint

First of all, let's briefly outline how tracing via PCI Express works. The trace tool, TRACE32 PowerTrace Serial, is designed to operate as a PCIe endpoint. As such, the trace tool must be connected to the target and already running before the software (e.g. the boot loader) starts enumerating and configuring the endpoints. During this configuration, each endpoint is allocated an address range in the system memory. Thereafter, data for this endpoint can be simply written directly to this location.

The trace infrastructure of the target system must then be configured to write the trace data into the address range which has just been allocated to the TRACE32 PowerTrace Serial endpoint. This is applicable to the vast majority of processors. After this, tracing can begin.

Characteristics

Variable data rate
- Gen1 250 MByte/s per lane
- Gen2 500 MByte/s per lane
- Gen3 984 MByte/s per lane

Variable port width (1, 2, 4 or 8 lanes)

Full-size card adapter available,
mini-PCIe card adapter planned

4 GigaByte of trace memory,
trace decoding for all standard protocols

Summary

Tracing via PCIe is easy to do. For target systems without a dedicated trace interface, this technique offers an excellent method for recording large amounts of trace data. The main differences between tracing via an external interface such as PCIe and a dedicated trace port can be summarized as:

- The trace recording can be started only after the target software has configured the PCIe root complex. This is a task assigned to the operating system.
- At the same time, tracing via PCIe is, strictly speaking, no longer "non-intrusive". It requires a portion of the system memory. Furthermore, the trace competes with other endpoints for the bandwidth of the PCIe bus.

Seamless Transition from Wind River to TRACE32

Since 2014 Wind River has stopped offering JTAG debuggers, and existing users are increasingly switching to TRACE32 for maintenance and further development of their products. In close cooperation with Wind River, Lauterbach has enhanced its TRACE32 software, and any basic adjustment to the system is implemented seamlessly in order to ensure that users can keep working as per usual.

Support for All Processors

Most clients changing over to TRACE32 use processors of the Power Architecture™ family. Support for this architecture has been part of the Lauterbach product portfolio since 1997, and clients changing over to TRACE32 can rely on proven debug solutions.

Script Converter

Before being able to start with debugging, the program code is usually programmed into target flash. TRACE32 uses a script for this. The setting of the processor configuration register, in particular the SDRAM configuration, is an important component of the script. Wind River defined the required settings in a "Register Configuration File". Lauterbach provides a special converter for translating this file into a TRACE32 script. Over the years of professional debugger use, customers have often accumulated a large number of test scripts for various purposes. Lauterbach provides a script converter allowing the transfer of these complex test scripts to TRACE32.

Wind River Workbench

Switching to a new debugger after using the same tried and tested system for many years can be an anxious moment for many engineers. For this reason, in 2015, Lauterbach enhanced its debuggers to act as a TCF agent. It is now possible to use the Wind River Workbench as a debug IDE with the TRACE32 debugger acting as the debug backend.

Support for All Wind River Products

In most projects, Wind River software products are also used, for example: Wind River VxWorks, Wind River Linux, or Wind River Hypervisor. Comprehensive debug solutions for these products were obviously provided when using a Wind River debugger. Lauterbach is an extremely experienced tool provider in this area as well, and TRACE32 has supported the debugging of Wind River VxWorks since 1996. Full Wind River Linux support was added in 2000. Currently TRACE32 also offers support for Wind River products which no longer have Workbench JTAG debug capability. This includes VxWorks 7, VxWorks 653 v4 as well as Wind River Hypervisor v3.

TRACE32 Debugger for RISC-V

JTAG Debugger

Lauterbach launched its new RISC-V debugger in November 2017. The first chips currently supported are the E31 Core Complex (32-bit) and the E51 (64-bit) Core Complex by SiFive.

RISC-V is an Open Instruction Set Architecture (ISA), based on the established RISC principles, and is organized, specified and developed under the direction of the RISC-V Foundation (https://riscv.org). Initially established for academic research, RISC-V is currently gaining traction in the embedded market, making a professional hardware debugger indispensable.

The Lauterbach implementation for the RISC-V debugger is based on the open source specification "RISC-V External Debug Support" that is expected to be adopted by the RISC-V Foundation in 2018. The objective of the specification is a flexible halt mode debugging. Each hardware thread of a RISC-V core shall be debugged from the reset directly.

To be able to debug RISC-V processors with TRACE32, they currently must be equipped with the JTAG DTM (Debug Transport Module). The DTM is specified as an independent and replaceable module, allowing chip manufacturers to have the freedom to implement the access to the Debug Module via a different communication interface. Thanks to its vast experience with the different debug communication interfaces, Lauterbach is a competent implementation partner.

Via the Debug Module, the TRACE32 debugger has access to all standard debug functions of the processor. These are part of the specification as Debug Register and Abstract Commands. Furthermore, the specification also allows the design of proprietary debug functions. The TRACE32 RISC-V debugger already supports different standard ISA extensions as, for example, compressed instructions or floating-point instructions but it also supports customer-specific ISA extensions.