Unleashing the Power of RISC-V E-Trace with a Highly Efficient Software Decoder

Marcel Zak[1, *], Mat O'Donnell[1] and Vivek Chickermane[2]

[1]Tessent Embedded Analytics, Siemens EDA, UK
[2]Tessent Embedded Analytics, Siemens EDA, USA
* Corresponding author: `marcel.zak@siemens.com`

Abstract

Debugging program misbehaviour that impacts large scale deployment of highly connected systems requires a robust debug infrastructure to monitor the instructions, data, and transactions between system components. The rapid adoption of RISC-V processors in mission critical applications has compelled system designers to rely on embedded trace monitors based on the RISC-V E-Trace specification to collect instruction trace data to analyse the trajectory of the transactions involving the CPU, memory, I/Os, peripherals, and other sub-systems. In this paper we describe a highly efficient software E-Trace Decoder that allows trace data to be non-intrusively captured at-speed. We describe two case studies to highlight the power of this implementation and provide quantitative data to show the efficiency of this implementation.

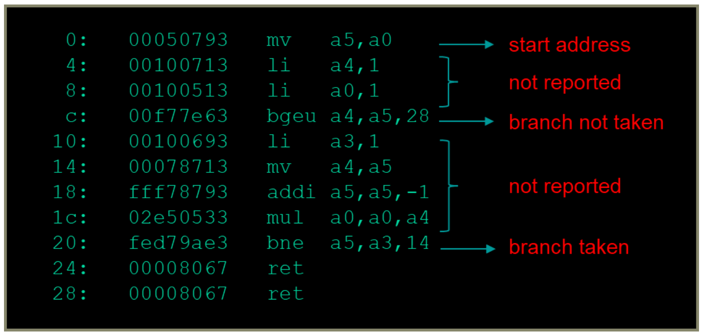
Introduction

A key requirement for deploying RISC-V based architectures in mission critical applications is the ability to quickly isolate program misbehaviour due to software bugs, and latent silicon defects. For new and unproven architectures there are additional areas of concern such as efficiency of multi-threading, cache coherence, branch prediction, I/O throughput, asynchronous HW/SW interfaces, concurrency impacts, etc. An embedded trace encoder that complies with the RISC-V E-Trace specification [1] provides real-time capture of the instructions being executed by the RISC-V processor that can be combined with other monitoring data to provide a comprehensive system debug functionality.

The term *trace* is defined as the non-intrusive monitoring of an application's execution so that the trajectory of the program execution can be tracked over time. It provides the ability to quickly and robustly isolate the exact sequence of events and data updates that led to a functional failure. It can be coupled with other procedures such as run-control (stop-and-go debug), code coverage analysis, reverse debuggers, and quality assurance tools. Trace is particularly valuable when the errors are infrequent and irregular such as those triggered by latent hardware defects.

Figure 1 shows a generic working principle of an embedded trace system such as described in [1]. The embedded trace encoder monitors instructions and data. Due to the vast amount of data that is generated, instruction trace data is only encoded and stored when the trace is triggered. It only logs the starting address and whether a branch - jumps, calls, returns, interrupts, exceptions etc., - is taken or not. Sequentially executed instructions are not reported as these can be easily inferred offline.

Program counter addresses that cannot be inferred such as asynchronous interrupts, exceptions, and indirect jumps require the trace encoder to report the destination address in full. The ability to trace at-speed, set filtering criteria and achieve very high compression ratios enables lossless trace in most use cases.



```
 0:   00050793   mv   a5,a0
 4:   00100713   li   a4,1
 8:   00100513   li   a0,1
 c:   00f77e63   bgeu a4,a5,28
10:   00100693   li   a3,1
14:   00078713   mv   a4,a5
18:   fff78793   addi a5,a5,-1
1c:   02e50533   mul  a0,a0,a4
20:   fed79ae3   bne  a5,a3,14
24:   00008067   ret
28:   00008067   ret
```

Figure 1: Example Instruction Trace

Trace data can be transported off-chip via dedicated high-speed parallel or serial interfaces or using functional interfaces such as PCIe or USB. In other cases, the trace data may be stored on-chip using a circular buffer built using high-performance memory arrays and unloaded to off-chip storage systems when needed.

Offloaded data can be processed by a software Trace Decoder in batch mode on a huge trace data file (terabytes or higher) or by the real-time processing of the trace packet messages coming off a high-speed I/O and extracting the time-stamped trace data. The decoded trace data can be visualized in real time or stored in a time-series database that feeds an AI/ML trace data analytics engine. In a closed loop real time system, the analytics may steer the trace monitor to log specific events or filter out what is unneeded. A robust system monitoring solution is designed to concurrently gather data from silicon sensors that track process, voltage and temperature (PVT), slack monitors, in-system test comparators and other instruments to detect anomalous behaviour such as irregular and infrequent functional failures or performance faults. The trace capability is an essential part of the forensic debug toolkit required to root-cause the erratic failures and take corrective action before the failure becomes permanent and leads to catastrophic errors.

Software Trace Decoder

The key features of the RISC-V software trace decoder that we have implemented are described in this section. Additional details will be provided in the final presentation.

- **Speed of Decoding**: In the worst-case E-Trace only encodes 1 bit per executed branch leading to an average compaction of .141 bits per instruction, so the decoder must be able to unravel this highly optimized bit-stream by juxtaposing against the addresses in the ELF file.
- **Support for tracing custom instructions with E-Trace Optional Extensions**: A key feature of the RISC-V ISA is the support for custom instructions, making it essential to trace them. This task is more challenging with E-Trace optional extensions, requiring the decoding algorithm to support user-supplied instruction specifications.
- **Filters for Processor Trace**: To reduce the volume of trace data, we support easy-to-use filters to limit the amount of processor trace data. Modern processors can execute multiple instructions every clock cycle, so this helps reduce the trace data that is collected.
- **Integration Capabilities**: The Trace Decoder is a part of an SDK that can communicate with simulation environments, FPGA prototypes, and real hardware.
- **Supported Extensions**: Many RISC-V extensions such as RV32/RV64 IMAFDCV are supported by our instruction trace decoder.
- **Implicit Return Feature**: This instruction is highly prevalent in compiled code and requires some complex algorithmic support to correctly interpret the program trajectory.
- **API for Processor Trace Module**: To enable debug architects to access the details of the trace data it is coupled with a high-level abstraction layer and Asynchronous Python API. In the final presentation this will be illustrated with code snippets to explain how trace data is received. It includes not just a list of PC addresses but also traps, cycle accurate groups, and trace lost events. Users can easily determine the exact position of traps and pinpoint when trace loss occurs.
- **Impact of Optional Extensions on Decoding Speed**: E-Trace offers many optional extensions that can make the encoded payload smaller (better compression) but require the decoder to do more work to reconstruct the execution flow with the benefit of significant compaction. Examples include:

i. **Sequentially inferable jump mode:** Use the combination of consecutive instruction as an inferable jump address, e.g., LUIPC + JALR.

ii. **Implicit return mode:** Maintain stack of expected function return addresses, and do not report if return is as expected.

iii. **Branch Prediction mode:** Reports the number of correctly predicted branches before a mis-prediction. Benefits programs with lots of loops, in some cases dramatically by orders of magnitude.

iv. **Jump Target Cache mode:** Cache indirect jump target addresses and report index rather than target address.

Results and Case Studies

The RISC-V Trace Decoder has been implemented and tested using the Embench™ Benchmark programs with instructions counts ranging from 1.03 to 7.7 million. The compression results range from 0.38 bits per instruction (BPI) to 0.0007 BPI in the best case with the average at 0.14 BPI. The total trace payload size for the benchmark varies from 127.55 down to 0.34 KiB in the best case. With the use of optional extensions in the trace encoder the BPI can be reduced by an average of 38.18% highlighting the benefits of the extensions. Detailed results will be published in the final version.

The benefits of using RISC-V E-Trace with a highly efficient decoder can be seen from two real world case studies. The first case study [4] is a hard drive SoC where a RISC-V processor executes real time programs to position the head within a 2.4 nm accuracy by using disturbance detection filters and adaptive control algorithms to guard against vibrations. The trace features allow the engineers to debug the algorithms, improve parallelism and reduce latency.

The second case study [5] proposes integrating RISC-V trace with Silicon Lifecycle Monitoring applications for continuous monitoring of program behaviour to help identify the root cause of silent data corruption, irregular and frequent errors, and under-performance during its functional lifetime. Time stamped trace data can be aligned with other silicon health monitoring sensors and built-in self-test of logic and memories.

References

[1] G. Panesar and I. Robertson, “Efficient Trace for RISC-V,” Siemens, ver. 1.1.3-Frozen, Mar. 23, 2022. [Online]. Available: https://github.com/riscv-non-isa/riscv-trace-spec/blob/main/riscv-trace-spec.pdf

[2] “Embedded Trace Macrocell Architecture Specification,” ARM, ver. H, Nov. 2020.

[3] “Real Time Instruction Trace,” Intel, ver. 1.05, Dec. 2015.

[4] R. Bohm, “Debug & Optimization Strategy in Tomorrow’s Storage Technology,” Seagate Technologies, Siemens U2U Presentation, Available: https://eda.sw.siemens.com/en-US/ic/tessent/embedded-analytics/

[5] V. Chickermane, M. Zak, and M O’Donnell, “Embedded Trace: A Key Enabler for Silicon Debug and Continuous Monitoring,” Poster PO.42, IEEE Int Test Conference, Nov 2024