RISC-V® Summit

December 8-10 | Virtual Event

CONTRIBUTE. COLLABORATE. COMMERCIALIZE.

Brought to you by informa tech

riscvsummit.com #RISCVSUMMIT

RISC-V® Summit

December 8-10 | Virtual Event

Overview of RISC-V Trace

Robert Chyla
Lead Engineer Debug & Trace, Chair of Nexus Trace TG
IAR Systems

#RISCVSUMMIT

Overview of Trace for RISC-V

Agenda & Introduction

Welcome ...

- Introduction
- Quick streaming trace demo (to set top-level focus)
- A bit of inside peak into trace pipeline
- Trace encoding principles
- Trace testing
- E-Trace and Nexus encoding principles
- Trace (compression) benchmarks and bandwidth
- Collection of useful RISC-V trace links
- Trace for RISC-V today
- What will/should happen next?
- IAR Systems ambitions ...
- Q&A session (voice or chat?)

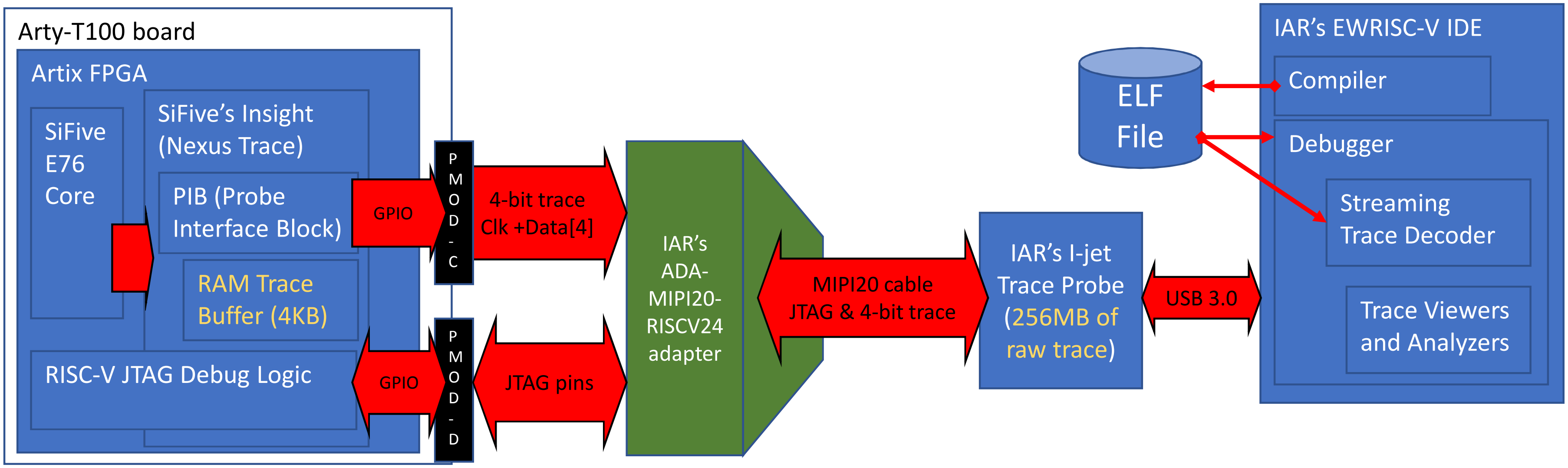
Robert Chyla | Debug & Trace Lead, IAR Systems

Overview of Trace for RISC-V

Introduction

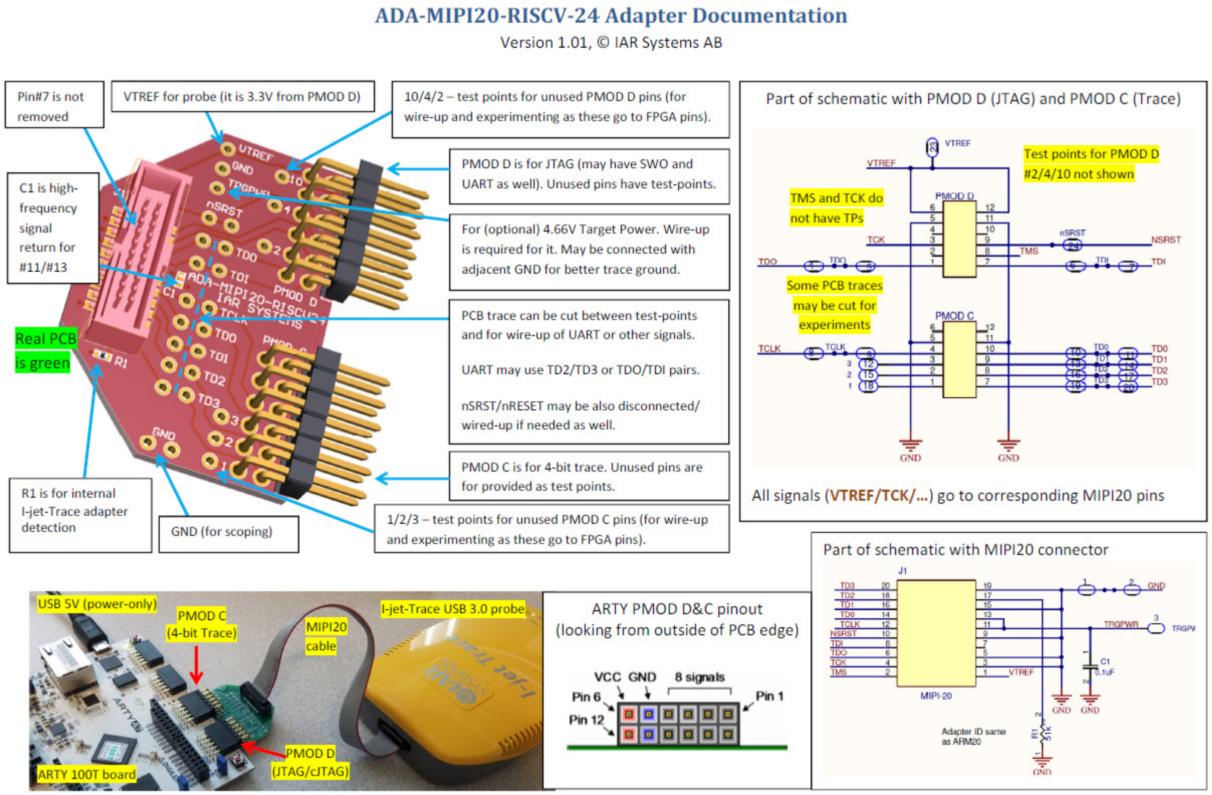
- Audience
  - This is NOT ‘trace-tutorial’ for SW developers (how to use trace)
  - This is trying to outline what to do to deploy trace inside of SoC or FPGA.
- Type of trace (quick …)
  - On-chip (dedicated RAM or shared), Off-chip parallel, Off-chip slow-serial (UART/Manchester), Off-chip High-speed Serial (monster trace ☺).
- Why two different trace (E-Trace and Nexus Trace)?
  - From same reason that we have Linux and Windows and MAC ☺.
  - Or FreeRTOS and Zephyr (maybe that’s better)?
  - One-size fits all is not practical …
- How to fully utilize trace tools which are available for Arm® architecture:
  - Why not? We are not building another RTOS for RISC-V – right?
  - There is 25+? years of trace-development to be utilized ..
- I may skip over some ‘parts’ - I want to leave some time for Q&A.

Quick streaming trace demo (to set focus)



- Trace may be also collected in dedicated RAM Trace Buffer (but is limited in size and cannot be streamed 'live').
- This (simpler) flavor was shown by other vendors at RISC-V Summit.

Quick streaming trace demo (adapter)



ADA-MIPI20-RISCV-24 Adapter Documentation

Version 1.01, © IAR Systems AB

Pin#7 is not removed

VTREF for probe (it is 3.3V from PMOD D)

10/4/2 – test points for unused PMOD D pins (for wire-up and experimenting as these go to FPGA pins).

PMOD D is for JTAG (may have SWO and UART as well). Unused pins have test-points.

For (optional) 4.66V Target Power. Wire-up is required for it. May be connected with adjacent GND for better trace ground.

C1 is high-frequency signal return for #11/#13

PCB trace can be cut between test-points and for wire-up of UART or other signals.

UART may use TD2/TD3 or TDO/TDI pairs.

nSRST/nRESET may be also disconnected/ wired-up if needed as well.

Real PCB is green

PMOD C is for 4-bit trace. Unused pins are for provided as test points.

R1 is for internal I-jet-Trace adapter detection

GND (for scoping)

1/2/3 – test points for unused PMOD C pins (for wire-up and experimenting as these go to FPGA pins).

Part of schematic with PMOD D (JTAG) and PMOD C (Trace)

TMS and TCK do not have TPs

Test points for PMOD D #2/4/10 not shown

Some PCB traces may be cut for experiments

All signals (**VTREF/TCK/…**) go to corresponding MIPI20 pins

Part of schematic with MIPI20 connector

Adapter ID same as ARM20

USB 5V (power-only)

PMOD C (4-bit Trace)

MIPI20 cable

I-jet-Trace USB 3.0 probe

ARTY 100T board

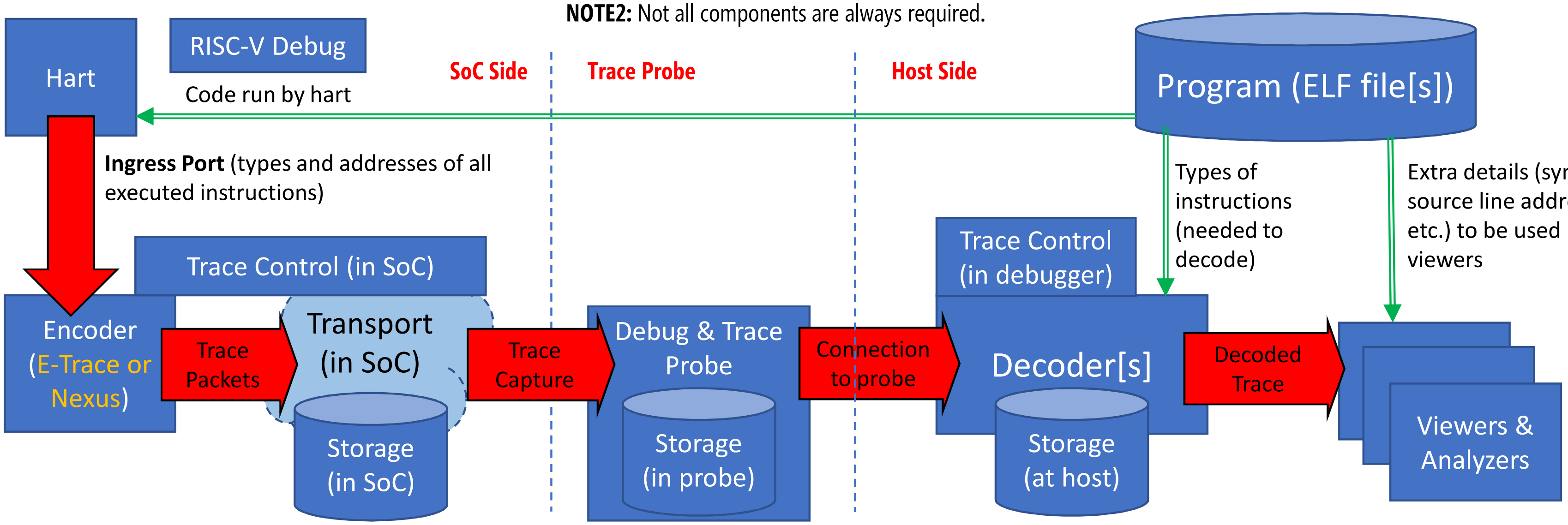
PMOD D (JTAG/cJTAG)

ARTY PMOD D&C pinout (looking from outside of PCB edge)

Basic Trace Pipeline

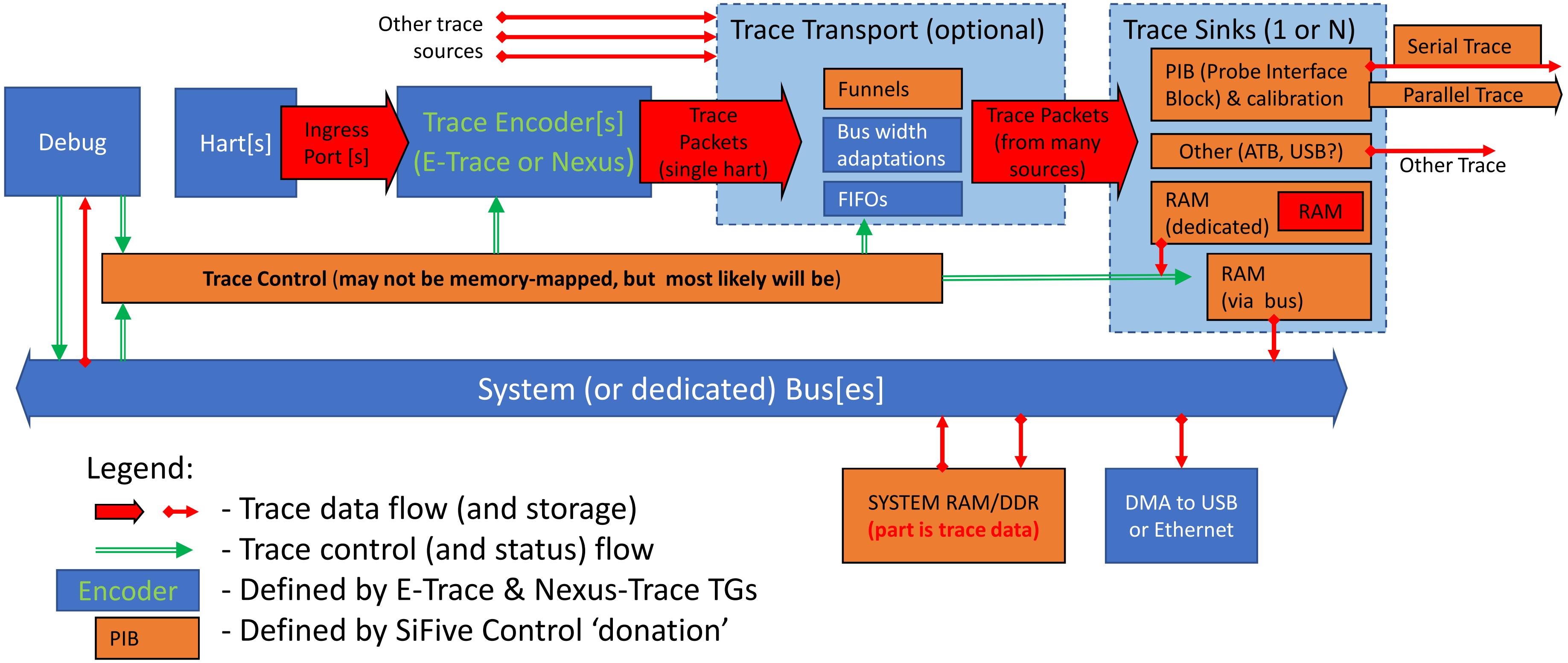
NOTE1: Simplified for single core/hart.

NOTE2: Not all components are always required.



- Hart/core is reporting <instruction type> and <PC address> streams (at every cycle!).
- Trace Encoder is compressing these parallel streams into sequence of 'Trace Packets'.
- Sequence of Trace Packets are decoded into <PC address> stream (looking at ELF-file for types of encountered instructions).

Trace Architecture (inside SoC)



Legend:

🟥 ↔ - Trace data flow (and storage)

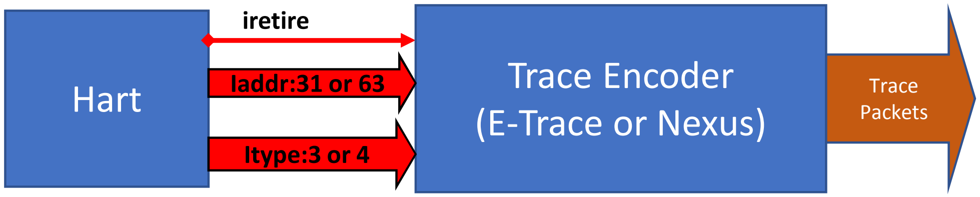
→ - Trace control (and status) flow

Encoder - Defined by E-Trace & Nexus-Trace TGs

PIB - Defined by SiFive Control ‘donation’

Trace Ingress Port Overview

- This is in details defined in "**Chapter 3 Hart to encoder interface**" in E-Trace Ratified Spec
- Signal bus which reports instructions encoder (complicated for harts which may retire more than 1 instruction/cycle). **Let's look at simple case only:**



- **iretire:** Number of retired instructions (usually 0 or 1). **IMPORTANT:** Only instructions retired i.e. fully completed are reported.
- **iaddr:** Address of retired instruction 31/63-bits (LSB is 0 for RISC-V, so always skipped to save 1 signal/bit).
- **itype:** Type of instruction (0=other/1=exception/2=interrupt/3=exception-or-interrupt-return/4=branch-non-taken/5=branch-taken/6=uninferable-jump)
  - For above 3-bit field, there is not possible to distinguish jump from linear instruction or indirect jump/call from normal return.
  - For 4-bit variant/extension, where call/jump/return/co-routine-swap are provided (to allow better compression). **IMPORTANT:** Chapter 3.1.1. provides qualification of [c.]jal[r] opcode variants into above cases. **These details are ABI defined and NOT defined by RISC-V architecture**.
  - **NOTE:** There is an option (extra signal) to support '**sequentially inferable jump mode**' where inferable jump is defined by two adjacent instructions.
- Exception or interrupt cause and privilege level (not shown – defined by spec as 'mandatory', but may be 0-size fields).
- Other (optional) signals (related to mode changes and context) and used when hart may retire more than one instruction on same cycle.

Overview of Trace for RISC-V

Trace Encoding Principles

This section will elaborate on trace encoding principles – it provides necessary background for understanding RISC-V trace encoding and decoding.

NOTE: During presentation some topics may be skipped – some more elaborated parts can be used while implementing own trace encoder inside of SoC.

PC Trace Encoding/Decoding Basic Principles

- **Do not emit too much ☺ -** key idea is to do NOT send information which may be known/available by looking at code (self-modifying code?).
  - Encoder emits enough information, so decoder may follow PC instruction after instruction and 'calculate' next PC based on trace packets.
  - Full PC is available at start and periodically.
- **Linear instruction** - if we have N linear instructions one after another, these usually run together - encoder must advance PC addresses by instruction sizes.
- **Direct (=statistically inferable) jump/call** – as destination PC address is known from opcode these are traced same as linear instructions.
  - RISC-V headache' here: Direct jumps have limited address range (+-20 bits). Far jump/call is performed as indirect-jump (two instructions).
- **Conditional Branch (**direct only for RISC-V architecture**)** – decoder needs to know if branch was taken or not-taken so next PC can be determined.
  - **Conditional Instruction** –usually traced similar to branches, but there are no such instructions in RISC-V architecture (unless in future extensions ☹ ...).
- **Indirect/non-inferable jump/call/return** – for all these, an address cannot be 'deducted by looking' at code itself, so an address must be sent by encoder.
  - Often small 'address-delta' (in different form ...) is sent – if destination PC is 'close' trace packet with an address is small.
  - Full address is sent periodically – this is called SYNC-packet. It may also serves as 'starting-point' for decoding (in case part of trace is lost).
  - RISC-V 'headache' here: There is no special opcode for 'ret' (but 'mret' is) – it may make encoder/decoder to be ABI-specific.
- **Exceptions & interrupts** - these are by definition 'unpredictable' (but fortunately rare, so do not affect trace compression much).
  - These are usually traced as specially marked 'indirect-calls' with full address given.
  - These provide a bit if 'chaos' as normal program-flow must break and encoder must know where interrupt/exception happened.
  - Entry into debug mode (stop of core/hart) is usually handled as special 'exception' code.
- **Lost/Missed Trace** – it may/will happen (from many reasons), that part of trace is lost. Encoder must start from next SYNC-packet (with full PC & context).

Basic Trace Encoding/Compression Techniques

- **Linear instructions and inferable jumps** – one may to count them as needed for exception/interrupt.
- **Indirect PC-change/Exception** - each indirect PC changes must provide an address
  - Address (either full or 'delta/partial') is reported together with 'reason'.
  - Start (or restart after filter-off...) of trace must provide FULL PC address as otherwise decoder will not know where to start.
- **Direct Branch Trace** - packet is emitted at taken branches
  - Non-taken branches are considered as 'linear-code' (as it is linear instruction in such a case ...).
  - Every taken branch will emitting trace packet (with number of instructions) - this is easiest to implement SoC trace encoder (just instruction counter?).
- **Branch History Trace** - each branch is adding single bit 'taken-or-not' into 'history-map' (does not generate new packet!)
  - More 'costly' to implement on SoC side, but provides MUCH better compression as we have 1-bit/branch. Otherwise we have 1-packet/taken-branch.
- **Other Important Data**
  - Despite pure PC-flow, encoder should emit changes in mode of and some other 'context'.
  - It is also desired (if possible to have each markings for important events - like interrupt entry/leave/sleep/reset etc.– it will allow good filtering).

Advanced Trace Compression Techniques

- **Return Stack**

  Each 'return' is PC change, which is unknown by looking at 'ret' instruction, so partial address must be sent. But if you know from where call was made you know where it will return. Both encoder and decoder maintain a stack of call/return. In case return address is matching an address at top of stack that particular 'ret' is traced as 'linear-instruction' as decoder will know next address as well.

- **Repeated Packet**

  Consider loop running a lot of times – single packet may encode limited number of repetitions as 'taken branch' must be part of a loop. Instead of sending same packet N times, short packet '**Repeat previous packet N times**' is sent instead.

  This technique (part of Nexus standard) may dwarf any other compression method!

- **Recent address[es] cache:**

  If certain (indirect!) address is used often, it may be put into 'cache of recent addresses' and later sent as small index. This technique may be useful in case of C++ code with virtual methods, where indirect addresses (for calls) are used often.

- **Branch Prediction**

  Core may be using some branch-prediction algorithm. In case (direct) branch is correctly predicted, it is considered as direct jump (i.e. does not generate any extra trace).

Other Trace Encoding Details

- **Interrupts and Exceptions** – these must provide a reason (at least rough …) – encode must provide PC where exception/interrupt occurred.
- **Trace Filtering/Overflows** –restarted trace (filtered-off or 'dropped' due to bandwidth) must emit full PC and context, so decoding may be reliably resumed.
- **Out-of-order and dual-issue cores** – ratified RISC-V trace ingress port (core ➔ encoder interface) is only providing RETIRED=FULLY FINISHED instructions.
  - As such there is no need to produce any 'cancel' trace packets (it makes decoder simpler …).
  - In case of core, which may retire more than one instruction at particular cycle, parts of ingress port is repeated N times.
  - In case of out-of-order execution (but not 'speculative') encoder must take this into account.
  - Obviously this is not very easy to implement this well ☺ - I would strongly recommend to implement trace for simple core first.
- **Trace and Security** - It may be desired to limit trace for parts of the system (for example machine mode).
  - It may be either done on core-side (just do not provide any instructions …) or encoder side (just do not emit packets).
- **Multi-core/multi-hart trace** - trace packets may (optionally) provide source of trace.
  - It is assumed each of N cores/hart will have own encoder with own ingress-port.
  - Trace packets from different encoders may be merged together (N different decoders will be able to split them based on trace source ID).
- **Mixed-core trace** – it may be often the case when RISC-V core is used in a system together with other cores implementing CoreSight™ trace.
  - In this case, output from RISC-V trace encoders should be routed via ATB (Arm Trace Bus) bridge and connected to trace funnel.
  - In case of system with different Nexus-compatible trace, each component on the system should have different SRC-ID.

Time-stamp and cycle-accurate trace

- **Timestamp** - trace packets may also (optionally) provide ‘timestamp’, so PC flow can be time-correlated with other events in the system.
- **Cycle-accurate trace** - If small packet (1 byte?) represents say 20 individual instructions, it is hard to efficiently expose timing details of every single instruction
  - It can be done, but it will limit trace encoding efficiency.
- Some trace probes (I-jet Trace is doing it) provide timestamping of trace packets ‘on-the-wire’
  - It is not ideal (timestamp is not at the source …), but saves on bandwidth.
  - Works well if source is NOT providing timestamp (for example early Arm® Cortex-M-class ETM did not provide timestamp packets).
  - Works well if trace is ‘rare’ as particular packet is not being delayed inside of SoC FIFOs
- Good timestamp of trace is challenging topic ☹.

E-Trace Encoder Principles

- E-Trace packets are ‘hand-crafted’ sequence of bytes with almost every bit ‘defined’ (**goal was to provide max compression**).
  - Packets are starting at byte-boundary (appropriate bits=0 are appended if needed).
- Full address is emitted as 31-bit or 63-bit field (on RISC-V LSB of PC is always 0 and is skipped).
- Branch-map (1-bit for taken/not-taken branch) can be max 31-bits in size (dedicated 5-bit counter, with a bit of ‘waste’ is provided).
  - There is a version of 38-bit encoding.
- Additional compression is achieved by ‘prediction’ so more instructions provide ‘no-trace’ (inside of encoder and repeated by decoder).
- Exceptions/interrupts and other (rare-packets) are not so important from compression point of view
  - But VERY important from handling ‘corner-cases’!
- Packet ‘framing’ is not part of spec, but is affecting overall bandwidth.
- Two initial part of entire trace pipeline are defined
  - Ingress port (input to encoder).
  - Format of trace packets out of encoder.
- This is certainly good start, but not enough to have ‘out-of-the-box’ tools working with it.

Nexus Encoder Principles

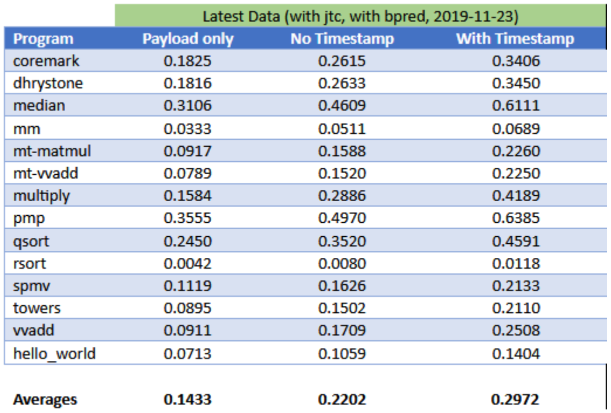
- Nexus generates messages consisting of two bit-fields:
  - MSEO - 2 status bits (idle, start if message=packet, end of message, end of variable field)
  - MDO - 6 'data'-bits (either fixed in size or variable 0-expanded till end)
- Above '2+6' split is not mandated by Nexus, but is very good (and was chosen by SiFive ...) because:
  - SW guys love work on bytes (36-bit memory is HW invention from good reason ☺ – but we still do not have 36-bit registers!).
  - It provides 12.5% (=2/8) of 'waste', but it is NOT 'pure-waste' as it provides benefits.
    - There are no 'counters' describing sizes at all as variable size-fields are
    - There is no need for any packet framing (I will get back to this later on ...)
- Most of fields are 'vendor-specific' (by 'vendor' we will mean 'RISC-V' – Nexus Trace TG is trying to define what that means ...)
- It is highly extendible format (64 types of messages, but only few are really needed for trace – Nexus defines debug messages as well)
  - If you utilize this, one may get really good compression (it may NOT be 100% Nexus-compatible, but 'Nexus-inspired' ...).
- Examples of messages can be found in manual for NexRv tool (as it provides message dump capability).

Ways of testing trace ‘pipeline’ implementation

- We (IAR) have in-house technology to test ‘trace correctness’
  - It was ported from Arm® trace to RISC-V trace with basically no effort.
  - We use it to validate correctness of our trace decoder implementation (and viewers as well), but it actually tests entire ‘pipeline’ (including core!).
  - We run a program on HW/FPGA and compare decoded trace (PC sequences) with trace generated by simulator (which runs same program).
  - We do so for 100-s of programs (long and short …) - as we are compiler company, we have 1000-s of program to test C/C++ compiler correction. Imagine what mixture of ‘ugly-code’ this provides ☹ … We also trace different library calls (often hand-written …).
  - We also do it a bit more complex (filtering, stepping etc.) – although not for all programs …
    - We also ‘kick’ these programs with interrupts – assuming interrupts are ‘well-behaving’ main PC flow should not be affected.
  - It was proven to be good-enough to find problems which were NOT in decoder, but actually in encoder (implemented in FPGA).
- That technology was inspiration for NexRv tool (reference encoder & decoder) I provided here: https://github.com/riscv/tg-nexus-trace/tree/master/refcode/c
  - Run the program and produce “PC sequence” AND “encoded-trace”
  - Decode that “encoded-trace” – if PC sequence is NOT the same either decoder (or encoder …) is incorrect.
  - For ANY deterministic program “input PC sequence ➔ encoder ➔ decoder ➔ output PC sequence”, output must match an input.
  - This technology is elaborated more in this PDF: https://github.com/riscv/tg-nexus-trace/blob/master/refcode/c/NexusTraceTG-RefCode.pdf
- **PS. Just added today.** It all well correlates to (great) Summit 2020 talk by Bill McSpadden from Seagate (he is active member on trace groups).
  - He was actually comparing 3 flows – ‘Instruction Retirement Monitor, Decoded Trace, Simulator’ and look for differences …
    - PS: Even if none of them can be considered ‘**100% good**’ chance that each method has same problem is very small …
  - He was also ‘bombarding’ code with ‘asynchronous’ events to make it just harder.

Trace Compression Benchmarks

- Very often people ask me (not so easy …) question:. **What is particular trace method compression ratio?**
- I am in this field long-enough to provide the following answer: "**Give me your program and I will tell you ☺**".
  - Seriously – this is **very dependent on particular program** - Here are benchmarks officially published for E-Trace (bits/instruction):



	Latest Data (with jtc, with bpred, 2019-11-23)		
Program	Payload only	No Timestamp	With Timestamp
coremark	0.1825	0.2615	0.3406
dhrystone	0.1816	0.2633	0.3450
median	0.3106	0.4609	0.6111
mm	0.0333	0.0511	0.0689
mt-matmul	0.0917	0.1588	0.2260
mt-vvadd	0.0789	0.1520	0.2250
multiply	0.1584	0.2886	0.4189
pmp	0.3555	0.4970	0.6385
qsort	0.2450	0.3520	0.4591
rsort	0.0042	0.0080	0.0118
spmv	0.1119	0.1626	0.2133
towers	0.0895	0.1502	0.2110
vvadd	0.0911	0.1709	0.2508
hello_world	0.0713	0.1059	0.1404
Averages	**0.1433**	**0.2202**	**0.2972**

As you see, compression 'ratio' is very different for different programs:

- Sometimes 'amazing' (0.0042 for 'rsort')
- Sometimes 'so-so' (0.3555 for 'pmp')
- Compression range is 0.3555/0.0042= **84 times!**
- Seems that 'packet-framing' provides 0.22/0.14 - 1 = **57% overhead**

- Two 'bigger-programs' (coremark and dhrystone …) are very different (as far as C-code), but have consistent (and above average …) compression.
- Still that 'pmp' is nearly 100% worst!
- I have some basic questions: Are these program representative enough? What to do to check these results? How to run it with different encoder or options?

Trace Compression Benchmarks (part2)

- My goal with trace benchmarking is to provide INFRASTRUCTURE to do so.
  - Seeing is believing …
  - Run program ➔ Create PC sequences (ingress port …) ➔ Run 'reference encoder' (with appropriate options/modes/parameters) ➔ Know
  - This is what NexRv tool is able to do.
- I did run NexRv tool (which reports Nexus compression …) on these programs (which I compiled myself using GCC and IAR compiler) and got very good results, but still I do not want to 'publish' these from two reasons:
  - Providing only results (without anyone being able to verify it) is not good – I may be making simple mistake.
  - Running trace compression on test of C-programs does not mean much. If you do NOT have an ELF (=binary code) at hand compiler switches/options/versions may make big-difference.
- I joined code size/speed groups to get 'representative' benchmarks:
  - Recently Zephyr was posted …
- I will provide Nexus trace benchmarks once I will be able to run E-Trace reference encoder/decoder on any ELF file
  - Currently it is not possible, as reference code for E-Trace cannot be run as these are not complete – hoping to have it fixed.
  - Any open source implementations of E-Trace encoder/decoder out there?

Trace Bandwidth (related to benchmark)

- Let’s assume (for simplicity of mental math …) 0.33bits/instruction, i.e. 3 instructions/bit of trace.
  - For on-chip trace this is not a problem (bandwidth to memory is usually enough)
- Let’s assume we have 100MHz core executing 1 instruction/cycle. So, to trace that core one will need 33Mbps trace bandwidth.
- Let’s look at bandwidth of some existing trace probes (I will use well known to me IAR I-jet Trace probes):
  - MIPI20 trace (4-bit, ~100MHz, double edge) = 800Mbps ➔ **Enough for single 2.4GHz or 24 100MHz cores!**
  - Serial UART/Manchester 60Mbps ➔ **Enough for full PC trace over single ‘plain-wire’.**
  - If above is not enough, one can do 1-bit MIPI20 = 200Mbps ➔ **Should be enough even for 5x worst trace compression!**
  - Mictor-36 (16bit, ~350MHz, double edge) = 11.2Gbps ➔ **14 (=11.2G/800M) times more than 4-bit (300+ 100MHz cores!)**
- Conclusions:
  - Assuming trace compression benchmark is right, we should be OK.
  - It may be possible to run full-PC trace over slow-serial interface (it was NOT possible for Arm® trace).

RISC-V® Summit

Collection of Useful RISC-V Trace-related Links

- Ratified Debug Specification (v 0.13.2, March 2019): https://riscv.org/wp-content/uploads/2019/03/riscv-debug-release.pdf
  - Reserves 3 trigger actions for trace use (no trace-related changes in recent working version)
- Ratified RISC-V Processor Trace Spec. (v1.0, March 2020): https://github.com/riscv/riscv-trace-spec/blob/e372bd36abc1b72ccbff31494a73a862367cbb29/riscv-trace-spec.pdf
  - Includes ingress port specification which - shared with Nexus Trace for RISC-V.
  - By mistake link on this official page https://riscv.org/technical/specifications/ points to working, not yet approved version. Link above is for ratified spec.
  - “RISC-V Processor Trace” was recently renamed to “E-Trace for RISC-V” (E-Trace stands for Efficient Trace).
- Nexus IEEE-ISTO-5001 Trace Specification: http://nexus5001.org/wp-content/uploads/2018/05/IEEE-ISTO-5001-2012-v3.0.1-Nexus-Standard.pdf
- MIPI Alliance Recommendations for Debug and Trace Connectors: https://mipi.org/sites/default/files/MIPI-Alliance-Recommendation-Debug-Trace-Connectors.pdf
- SiFive/Nexus related links:
  - Insight Debug & Trace main page (it is first Nexus trace implementation for RISC-V): https://www.sifive.com/soc-ip/sifive-insight
  - Reference decoder (called as-is by Freedom Studio debugger to decode trace): https://github.com/sifive/trace-decoder
  - SiFive trace decoder tests (for reference decoder): https://github.com/sifive/trace-decoder-tests
- Benchmarks:
  - Code used for some trace benchmarking already: https://github.com/embench/embench-iot
  - Code Size TG (ambition – some benchmarks recently posted): https://github.com/riscv/riscv-code-size-reduction/tree/master/benchmarks
  - Code Speed SIG (ambition – no benchmarks posted): https://github.com/riscv/riscv-code-speed-optimization

Collection of Useful RISC-V Trace-related Links (part2)

RISC-V Summit

- E-Trace TG main page: https://lists.riscv.org/g/tech-trace
  - Github with working specification (Latex format): https://github.com/riscv/riscv-trace-spec
    - Reference encoder & decoder (needs some work to be externally used): https://github.com/riscv/riscv-trace-spec/tree/master/te_codec/src
  - Some early benchmarking and patch for Spike simulator: https://github.com/gajinderpanesar/spike-instruction-trace-patch
- Nexus TG main page: https://lists.riscv.org/g/tech-nexus
  - Github with working specification (Asciidoc format): https://github.com/riscv/tg-nexus-trace
    - Trace Control (based on proposal by SiFive): https://github.com/riscv/tg-nexus-trace/blob/master/docs/RISC-V-Trace-Control-Interface.adoc
    - Nexus Message (Details): https://github.com/riscv/tg-nexus-trace/blob/master/docs/NexusTrace-TG-MessageDetails.adoc
    - Proposal for Trace Connectors (MIPI-based): https://github.com/riscv/tg-nexus-trace/blob/master/docs/NexusTrace-TG-Connectors.adoc
  - Reference Nexus encoder/decoder in C (NexRv tool provided by IAR Systems): https://github.com/riscv/tg-nexus-trace/tree/master/refcode/c
    - Documentation for above (PDF only): https://github.com/riscv/tg-nexus-trace/blob/master/refcode/c/NexusTraceTG-RefCode.pdf

RISC-V Trace ‘in-the-wild’ as of today - literally ☺!

- **This is based on my observations and participation in both RISC-V Trace group meeting – it is certainly not complete (RISC-V is very dynamic!).**
- UltraSoc’s trace infrastructure for RISC-V is available for some time and I assume it may be widely used (it is using USB as in/out transport media).
- SiFive Insight(=Nexus Trace) is publicly available since 2019/09 and is most likely deployed in many devices as it is standard SiFive IP offering.
  - More compressed variant (Nexus History Trace) was added few months ago.
  - It also provides Instrumentation Trace (via single pin using UART or Manchester encoding)
- NexRv tool (provided by IAR in May 2020) allows to validate Nexus trace on simulation level and provides decoding tool to see if trace encoder is correct.
- Trace Control proposal (provided by SiFive in June 2020) – it is embraced by Nexus and E-Trace TGs (to be clarified and extended – should remain common!)
- Trace-related talks at this RISC-V Summit 2020 (in ‘order of appearance’):
  - Seagate - Dec 8-th (2:30PM PTS): **Comprehensive Pre-Si Verification of RISC-V Cores in a Storage Controller** – using E-Trace seriously.
  - Lauterbach - Dec 8-th 3:00PM PST: **Debug and Trace of Complex Heterogeneous SOC** - SiFive Insight RAM trace (Nexus) + AMP debug.
  - Segger - Dec 8-th 3:50PM PST: **Debugging RISC-V based Embedded Systems -** SiFive Insight RAM trace (Nexus) + RTT Monitor.
  - UltraSoc/Mentor - Dec 9-th (1:30PM PST): **RISC-V in 5G New Radio Small Cell Base Stations**
    - Elaborated analytics for monitoring complex SoC (Processor Trace not mentioned, but it is implied as Andes RISC-V cores were shown)
  - IAR Systems - Dec 10-th (**this talk**). SiFive Insight(Nexus): Showing Off-chip (streaming!) trace & Instrumentation Trace (for printf and monitoring).
- As Nexus Trace TG Chair, I am aware of **several companies adding Nexus trace to own cores as we speak**.
- One adoption of E-Trace format I am aware of is Seagate (see above talk). **BTW: It is great example of use of trace in early pre-silicon stage!**

RISC-V Trace – what will/should happen next?

- **IMPORTANT:**
  - Board & TSC decided, that only '**RISC-V ISA**' maybe officially ratified – everything else will be 'Thing?? **for RISC-V**' and will be approved (never ratified).
  - It allows many different trace to co-exist - same was as many different OS/RTOS (**for RISC-V**) may co-exist.
  - It is a bit of blurred line (as debug & trace are connected with ISA a lot …)
- **"E-Trace for RISC-V" TG is working on:**
  - Data trace (definition of ingress port and packets) – there are challenges with out-of-order cores, delayed memory accesses and atomics.
  - Adoption of (parts of …) Trace Control.
  - Define of basic trace filtering capabilities.
- **"Nexus Trace for RISC-V" TG is/will be working on:**
  - Publish benchmarks (in a way everyone can run them using NexRv tool).
    - Compare with same benchmarks on E-Trace encoder (waiting for E-Trace encoder/decoder on github to be complete).
  - Finalize clarification of all RISC-V applicable Nexus messages.
  - Define 'profiles' (Nexus is reach format and allows many options – we want to set on two or 3 sub-sets).
  - Officially propose/approve physical connectors (MIPI20 for slow-trace and Mictor-38 for high-speed) – goal is to have existing trace probes to work as-is.
  - Expand on Trace Control spec and elaborate on filtering (both to be shared with E-Trace).
  - Add Instrumentation Trace (it is part of SiFive Insight). Needed for small devices – it was true success when introduced in Arm® Cortex-M™ cores!
  - Elaborate on timestamping (this is not very easy problem for compressed trace …).

RISC-V Trace – IAR ambitions

RISC-V Summit

- Provide more trace-related material into open-source/github
  - To make Nexus trace deployment (in SoC) easier (NexRv tool was useful).
  - Sorry it is too early to provide more details on that - but Christmas is coming ☺.
    - One of examples may be providing Arty ➔ MIP20 adapter (which I was showing/using during the demo).
- Provide all advanced/extra options for SiFive Insight Trace (including calibration of parallel and serial trace).
- Integration of E-Trace encoder (it is small portion of entire trace infrastructure).
    - **NOTE: We are looking for someone implementing E-Trace encode format – please contact me.**
- Provide pure command-like access to trace (both control and decoding)
  - We have it in-house.
  - It will enable validation of SoC-side implementation of trace (control and encoder) from scripts.
- Stay tuned for more details
  - I will broadcast any news/changes on Nexus Trace TG mailing list

Summary and Q&A …

- Trace made big ‘leap-jump’ since RISC-V Summit 2019.
    - It is good to have two groups working in parallel.
        - Help is needed (please join and contribute).
    - I expect 6 months from today Trace for RISC-V it will be widely used.
- Q&A
    - Should I re-run the demo (now with better context)?
    - PLACEHOLDER for live Q&A during the session

RISC-V®
Summit

December 8-10 | Virtual Event

Thank you for joining us.

Contribute to the RISC-V conversation on social!

#RISCVSUMMIT @risc_v

Brought to you by informa tech

riscvsummit.com #RISCVSUMMIT