Debug and Trace Infrastructure for RISC-V SoC

Hanan Moller, UltraSoC
hanan.moller@ultrasoc.com

RISC-V Service Tools Virtual Meetup
23 April 2020
• Overview
• RISC-V Instruction Trace
• Cycle Accurate Instruction Trace
• Holistic System
• Summary
Overview

• It is all about the system - not about ISAs or Cores
• Understanding today’s systems’ behaviour is hard... very hard
• Surprisingly, software sometimes does not behave as expected
  • Software developers spend between 50-75% of development time debugging[1], this will only increase as systems become even more complex
• Realtime performance, safety and security are increasingly important
• In such systems non-intrusive visibility is crucial
  • Processor branch trace alone is not enough
  • Understanding CPU stalling behaviour is key

RISC-V Standardised Instruction Trace
• Trace execution progress from a known start address by communicating deltas taken by the program

• Deltas are typically due to instructions such as branch, jump, call and return; interrupts and exceptions are also types of deltas

• The instructions between the deltas are assumed to be executed sequentially
  • True for ISAs, such as RISC-V, where all instructions are executed unconditionally
  • True when instruction execution can be determined based on the program (e.g., not predicated)

• The trace needs to report only:
  • Whether or not a branch was taken – destination address known from execution binary
  • The destination address of taken indirect branches or jumps – destination address not known statically
Interrupts generally occur asynchronously to the program’s execution rather than intentionally as a result of a specific instruction or event.

Exceptions can be thought of in the same way, even though they can be typically linked back to a specific instruction address.

The decoder generally does not know where an interrupt occurs in the instruction sequence, so the trace encoder must report the address where normal program flow ceased, as well as give an indication of the asynchronous destination which may be as simple as reporting the exception type.

When an interrupt or exception occurs, or the processor is halted, the final instruction executed beforehand must be traced.
This shows the mandatory interface signals.

Optional information can include the instruction context.

The interface supports sideband signals, for example, to indicate core is in reset, halted, or stalled.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>iretire</td>
<td>Instruction has retired</td>
</tr>
<tr>
<td>itype</td>
<td>Type of instruction</td>
</tr>
<tr>
<td>cause</td>
<td>Exception cause</td>
</tr>
<tr>
<td>tval</td>
<td>Exception data</td>
</tr>
<tr>
<td>priv</td>
<td>Privilege mode during execution</td>
</tr>
<tr>
<td>iaddr</td>
<td>The address of the instruction</td>
</tr>
</tbody>
</table>
This shows the mandatory interface signals.

Optional information can include the instruction context.

The interface supports side-band signals, for example, to indicate core is in reset, halted, or stalled.
**Standardised Hart to Trace Encoder Interface**

CPU Core(s) → Trace Encoder → Debug Infrastructure → Trace Decoder (SW)

**Signal**

| caretime[CARETIRE-1:0] |

**Function**

Number of instructions retired in this block

Optional for multi-retirement CPUs for improved efficiency.

23 April 2020
• The Trace Encoder sends a packet containing one of the following:
  1. Update – a branch map with optional differential address
  2. Update – a branch map with optional full address
  3. Update – address only (differential)
  4. Synchronise - a context with or without a full current address

The above ensures an efficient packing to reduce data being routed on and subsequently transported off-chip.
Instruction Trace Algorithm

- Formats 0 and 1: branch map and address
  - Differential or full address
- Format 2: address only
- Format 3: sync packet
  - Subformat 0: for when starting, resynchronizing or resuming from halt. No `ecause`, `interrupt` and `tval`
  - Subformat 1: for exception. All fields present
  - Subformat 2: for context change. No `address`, `ecause`, `interrupt` and `tval`.

See spec in: https://github.com/riscv/riscv-trace-spec
### Benchmark Results

#### RISC-V Standard Compressed Branch Trace

<table>
<thead>
<tr>
<th>program</th>
<th>BPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>coremark</td>
<td>0.1825</td>
</tr>
<tr>
<td>dhrystone</td>
<td>0.1816</td>
</tr>
<tr>
<td>median</td>
<td>0.3106</td>
</tr>
<tr>
<td>mm</td>
<td>0.0333</td>
</tr>
<tr>
<td>mt-matmul</td>
<td>0.0917</td>
</tr>
<tr>
<td>mt-vvadd</td>
<td>0.0789</td>
</tr>
<tr>
<td>multiply</td>
<td>0.1584</td>
</tr>
<tr>
<td>pmp</td>
<td>0.3555</td>
</tr>
<tr>
<td>qsort</td>
<td>0.2450</td>
</tr>
<tr>
<td>rsort</td>
<td>0.0042</td>
</tr>
<tr>
<td>spmv</td>
<td>0.1119</td>
</tr>
<tr>
<td>towers</td>
<td>0.0895</td>
</tr>
<tr>
<td>vvadd</td>
<td>0.0911</td>
</tr>
<tr>
<td>hello_world</td>
<td>0.0713</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>0.1433</strong></td>
</tr>
</tbody>
</table>
• Controlling when trace is generated is important
  • Helps reduces volume of trace data

• Filters are required

• Using filters the following trace examples are available:
  • Trace in an address range
  • Start trace at an address; end trace at an address
  • Trace particular privilege level
  • Trace interrupt service routines

• Other examples
  • Trace for fixed period of time
  • Start trace when external (to the encoder) event detected
  • Stop trace when an external (to the encoder) event detected
Cycle Accurate Instruction Trace
Using Retirement Delta Trace
• Stated objective is to be able to determine when each instruction retires, but more importantly when the CPU stalls and does not retire instructions

• Can be achieved by reporting the number of clock cycles between each successive retirement

• The expectation is the CPU will be able to retire one instruction every cycle most of the time, so it will also be advantageous to be able to report the number of back-to-back retirements without stall

• Based on this, the basic reported unit (or group) should be a pair of numbers: the first being the number of contiguous retirements, the second being the number of stall cycles that follow

• For CPUs that support multiple retirement, need to be able to report details of how many instructions retire per cycle as well
• Key is to efficiently represent a sequence of count values
• Various encoding formats were investigated
• Elias Delta[2] is most efficient but for values under 32 Elias Gamma has the same or better efficiency
  • Coding number of cycles so large numbers will be less frequent so coding efficiency is less important
  • Gamma will result in more efficient coding when it matters
• Amortising short stalls below some threshold will also improve efficiency

Packets comprise packed groups of count tokens. Five token types are defined:

1. Token A – Elias gamma encoded count of contiguous retire or amortised stall cycles
2. Token B – Elias gamma encoded count of missed retirement opportunities
3. Token C – Elias gamma encoded count of unamortised contiguous stall cycles
4. Token D – Count of retirements in a single cycle (usually binary, though Elias gamma is an option)
5. Token E – Elias gamma encoded count of number of cycles to next non-stall cycle

Tokens are assembled into 3 group formats:

- {A, C}
- {A, B, C}
- {D, E}

This ensures most efficient packing to reduce data being routed on and subsequently transported off-chip.
• Basic retirement delta for single retirement harts
• Uses packed groups of \{A, C\} tokens

Trace Encoder Output – Single Retirement Example

- Group 1: A=4, C=1 (6 bits)
- Group 2: A=3, C=1 (4 bits)
- Group 3: A=2, C=4 (8 bits)

• Three groups, 18 bits in total (2 bits per instruction)
• Amortise single stalls with retirements
• Uses groups of \{A, B, C\} tokens

---

**Trace Encoder Output – Stall Amortisation Example**

```
R R R R S R R R S R R S S S S R
```

- **Group 1**
  - A=12, B=3, C=3
  - (13 bits)

- One group, 13 bits in total – more efficient (1.4 bits per instruction)
- Precise location of single stalls not known – less accurate

**Symbols**

- **R**: retire
- **S**: stall

---

23 April 2020
• Basic retirement delta for multi retirement harts (4 instructions per cycle in this example)
• Uses packed groups of \{A, B, C\} tokens

- Group 1: A=4, B=6, C=1 (11 bits)
- Group 2: A=3, B=0, C=1 (5 bits)
- Group 3: A=2, B=6, C=4 (13 bits)

• Three groups, 29 bits in total (1.2 bits per instruction)
• With single stalls amortised: A=12, B=24, C=3; 19 bits (0.8 bits per instruction)

R3 retire 3 instructions  S stall
Per cycle detail for multi retirement harts (4 instructions per cycle in this example)

Uses packed groups of \{D, E\} tokens

Nine groups, 30 bits in total (1.25 bits per instruction)

Highest precision detail of retirement activity

R3 retire 3 instructions  S stall
Retirement Delta Trace Algorithm

- Shows the \{A,C\} token case
- Some boundary conditions omitted for clarity
- \{A,B,C\} and \{D,E\} cases are similar
## Benchmark Results

### Retirement Delta Trace

<table>
<thead>
<tr>
<th>program</th>
<th>BPI</th>
<th>CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>coremark</td>
<td>1.0120</td>
<td>4.0328</td>
</tr>
<tr>
<td>dhrystone</td>
<td>1.2741</td>
<td>4.6771</td>
</tr>
<tr>
<td>median</td>
<td>1.3347</td>
<td>5.1407</td>
</tr>
<tr>
<td>mm</td>
<td>1.2192</td>
<td>5.1468</td>
</tr>
<tr>
<td>mt-matmul</td>
<td>1.2774</td>
<td>5.3007</td>
</tr>
<tr>
<td>mt-vvadd</td>
<td>1.2966</td>
<td>5.1938</td>
</tr>
<tr>
<td>multiply</td>
<td>0.8720</td>
<td>3.5071</td>
</tr>
<tr>
<td>pmp</td>
<td>0.8538</td>
<td>3.3465</td>
</tr>
<tr>
<td>qsort</td>
<td>1.2064</td>
<td>4.5891</td>
</tr>
<tr>
<td>rsort</td>
<td>1.1924</td>
<td>4.4238</td>
</tr>
<tr>
<td>spmv</td>
<td>1.1248</td>
<td>4.6240</td>
</tr>
<tr>
<td>towers</td>
<td>1.3873</td>
<td>4.6516</td>
</tr>
<tr>
<td>vvadd</td>
<td>1.3291</td>
<td>5.0886</td>
</tr>
<tr>
<td>hello_world</td>
<td>1.0544</td>
<td>4.3205</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>1.1739</strong></td>
<td><strong>4.5745</strong></td>
</tr>
</tbody>
</table>

### RISC-V Standard Compressed Branch Trace

<table>
<thead>
<tr>
<th>program</th>
<th>BPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>coremark</td>
<td>0.1825</td>
</tr>
<tr>
<td>dhrystone</td>
<td>0.1816</td>
</tr>
<tr>
<td>median</td>
<td>0.3106</td>
</tr>
<tr>
<td>mm</td>
<td>0.0333</td>
</tr>
<tr>
<td>mt-matmul</td>
<td>0.0917</td>
</tr>
<tr>
<td>mt-vvadd</td>
<td>0.0789</td>
</tr>
<tr>
<td>multiply</td>
<td>0.1584</td>
</tr>
<tr>
<td>pmp</td>
<td>0.3555</td>
</tr>
<tr>
<td>qsort</td>
<td>0.2450</td>
</tr>
<tr>
<td>rsort</td>
<td>0.0042</td>
</tr>
<tr>
<td>spmv</td>
<td>0.1119</td>
</tr>
<tr>
<td>towers</td>
<td>0.0895</td>
</tr>
<tr>
<td>vvadd</td>
<td>0.0911</td>
</tr>
<tr>
<td>hello_world</td>
<td>0.0713</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>0.1433</strong></td>
</tr>
</tbody>
</table>
### Profile of Instructions Used

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Property</th>
<th>Probability</th>
<th>Additional Stall Cycle Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>STORE</td>
<td>L1 cache hit</td>
<td>85%</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>L2 cache hit</td>
<td>10%</td>
<td>5 ... 10</td>
</tr>
<tr>
<td></td>
<td>Cache Miss (posted-write)</td>
<td>5%</td>
<td>20 ... 50</td>
</tr>
<tr>
<td>LOAD</td>
<td>L1 cache hit</td>
<td>85%</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>L2 cache hit</td>
<td>10%</td>
<td>5 ... 10</td>
</tr>
<tr>
<td></td>
<td>Cache Miss (DRAM fetch)</td>
<td>5%</td>
<td>100 ... 200</td>
</tr>
<tr>
<td>FENCE</td>
<td>...</td>
<td>always</td>
<td>100 ... 1,000</td>
</tr>
<tr>
<td>WFI</td>
<td>...</td>
<td>always</td>
<td>0 ... 20,000</td>
</tr>
<tr>
<td>I$ MISS</td>
<td>...</td>
<td>2%</td>
<td>0 ... 200</td>
</tr>
<tr>
<td>Pipeline Flush</td>
<td>...</td>
<td>2%</td>
<td>3 ... 5</td>
</tr>
</tbody>
</table>
Holistic System
Monitoring, Reliability, and Security for the Whole SoC

Interconnect (AXI, ACE, ACE-lite, OCP, CHI NoC)

- Bus Sentinel
- Lockstep Manager
- Bus Sentinel

Portfolio of Analytic Modules
- Message Engine
- Message Engine
- Message Engine

Flexible & Scalable Message Fabric
- UltraSoC IP
- UltraSoC IP With Lock

System Block

- AXI Comm
- JTAG Comm
- EBC Comm
- USB 2 Comm
- Aurora Comm
- System Memory Buffer
- Analysis Sub System

UltraSoC with Lockstep Manager

23 April 2020
• Today’s systems are too hard to understand with yesterday’s tools
• The systems must do what they were designed to do: safely and securely
• Understanding system behaviour is key: both in-field and realtime
• One dimension to this is to understand where and how CPUs stall
  • An encoder providing this has been presented
  • A number of different filtering and triggering schemes are supported
• Coupling this with a holistic non-intrusive monitoring infrastructure provides the means of understanding complete SoC behaviour
Contact details:

Name
hanan.moller@ultrasoc.com
www.ultrasoc.com
@UltraSoC