Processor Trace in a Holistic World

Gajinder Panesar, CTO, UltraSoC
gajinder.panesar@ultrasoc.com

RISC-V Workshop
May 2018 – Barcelona, Spain
• Overview
• Processor Branch Trace
• Trace Encoder Interface
• Algorithm
  • Filtering
  • Encoding efficiency
• Holistic System
• Demo Summary
• Summary and Next Steps
In complex systems understanding program behaviour is not easy

Surprisingly, software sometimes does not behave as expected

- This may be due to a number of factors, for example, interactions with other cores’ software, peripherals, realtime events, poor implementation or some combination of all of the above.
- Usually because engineers write code with bugs
- Hiring better software engineers is not always an option

Using a debugger is not always possible

- Realtime behaviour is affected

Providing visibility of program execution is important

- This needs to be done without swamping the system with vast amounts of data

One method of achieving this is via Processor Branch Trace
- Works by tracking execution from a known start address and sending messages about the deltas taken by the program.
- Deltas are typically introduced by jump, call, return and branch type instructions, although interrupts and exceptions are also types of deltas.
- For instruction set architectures, such as RISC-V, all instructions are executed unconditionally or at least their execution can be determined based on the program, the instructions between the deltas are assumed to be executed sequentially.
- This characteristic means that there is no need to report them via the trace, only whether the branch was taken or not and the address of taken indirect branches or jumps.
- If the program counter is changed by an amount that cannot be determined from the execution binary, the trace decoder needs to be given the destination address (i.e. the address of the next valid instruction). Examples of this are indirect branches or jumps, where the next instruction address is determined by the contents of a register rather than a constant embedded in the source code.
Interrupts and Exceptions

- 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.
Trace Encoder Ingress Port

- For cores retiring N instruction per clock cycle the interface is replicated N times.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>iv:valid</td>
<td>Instruction has retired or trapped (exception).</td>
</tr>
<tr>
<td>iexception</td>
<td>Exception</td>
</tr>
<tr>
<td>interrupt</td>
<td>0 if the exception was synchronous; 1 if interrupt</td>
</tr>
<tr>
<td>cause [CAUSELEN-1:0]</td>
<td>Exception cause</td>
</tr>
<tr>
<td>tv:val[XLEN-1]</td>
<td>Exception data</td>
</tr>
<tr>
<td>priv[PRIVLEN-1:0]</td>
<td>Privilege mode during execution</td>
</tr>
<tr>
<td>i:addr [XLEN-1]</td>
<td>The address of the instruction</td>
</tr>
<tr>
<td>instr [ILEN-1:0]</td>
<td>The instruction</td>
</tr>
</tbody>
</table>
The Encoder sends a packet containing one of the following:

1. Update – a branch map with or without a differential destination address/next address
2. Update – a full destination/next address and branch map
3. Update – a differential destination/next address with no branch or instruction related fields.
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 send branch map and address
- Format 2 is address only
- Format 3 is a sync packet
  - Subformat 0 for when starting or resume 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`.
• 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
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Instructions</th>
<th>Packets</th>
<th>Payload Bytes</th>
<th>Bits per instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>dhrystone</td>
<td>215015</td>
<td>1308</td>
<td>5628</td>
<td>0.209</td>
</tr>
<tr>
<td>hello_world</td>
<td>325246</td>
<td>2789</td>
<td>10642</td>
<td>0.262</td>
</tr>
<tr>
<td>median</td>
<td>15015</td>
<td>207</td>
<td>810</td>
<td>0.432</td>
</tr>
<tr>
<td>mm</td>
<td>297038</td>
<td>644</td>
<td>2011</td>
<td>0.054</td>
</tr>
<tr>
<td>mt-matmul</td>
<td>41454</td>
<td>344</td>
<td>953</td>
<td>0.184</td>
</tr>
<tr>
<td>mt-vvadd</td>
<td>61072</td>
<td>759</td>
<td>2049</td>
<td>0.268</td>
</tr>
<tr>
<td>multiply</td>
<td>55016</td>
<td>546</td>
<td>1837</td>
<td>0.267</td>
</tr>
<tr>
<td>pmp</td>
<td>425</td>
<td>7</td>
<td>39</td>
<td>0.734</td>
</tr>
<tr>
<td>qsort</td>
<td>235015</td>
<td>2052</td>
<td>8951</td>
<td>0.305</td>
</tr>
<tr>
<td>rsort</td>
<td>375016</td>
<td>683</td>
<td>2077</td>
<td>0.044</td>
</tr>
<tr>
<td>spmv</td>
<td>70015</td>
<td>254</td>
<td>1154</td>
<td>0.132</td>
</tr>
<tr>
<td>towers</td>
<td>15016</td>
<td>72</td>
<td>237</td>
<td>0.126</td>
</tr>
<tr>
<td>vvadd</td>
<td>10016</td>
<td>111</td>
<td>316</td>
<td>0.252</td>
</tr>
<tr>
<td><strong>Mean</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>0.252</strong></td>
</tr>
</tbody>
</table>

- Table shows encoding efficiency of the algorithm
- Does not include any overhead for encapsulating into messages or routing
- Different program types will have different overheads
Holistic System
Demo Details
Demo System Architecture

- Zynq ZC706 FPGA platform
  - ARM
    - Plus RV32 RISC-V
    - Plus custom logic
- Demo shows:
  - Bus state
  - Traffic
  - Performance histogram
  - Memory
  - Processor control
  - Bus deadlock detection
  - RISC-V Processor trace
Decoded trace showing source code and assembly

Bus activity

Trace Packets

Control configuration
Determining Program behaviour is not always possible using source level debugging

Understanding program behaviour in-field and realtime is needed

An efficient Branch Trace scheme provides this

- An encoder providing this has been presented
- A number of different filtering and triggering schemes are supported

Couple this with a Holistic non-intrusive monitoring infrastructure provides the means of understanding complete SoC behaviour
Next Steps

• Set up a dedicated task group to:
  • Standardize Processor to Encoder interface
  • Standardize compressed branch trace packet format

• Work on data trace