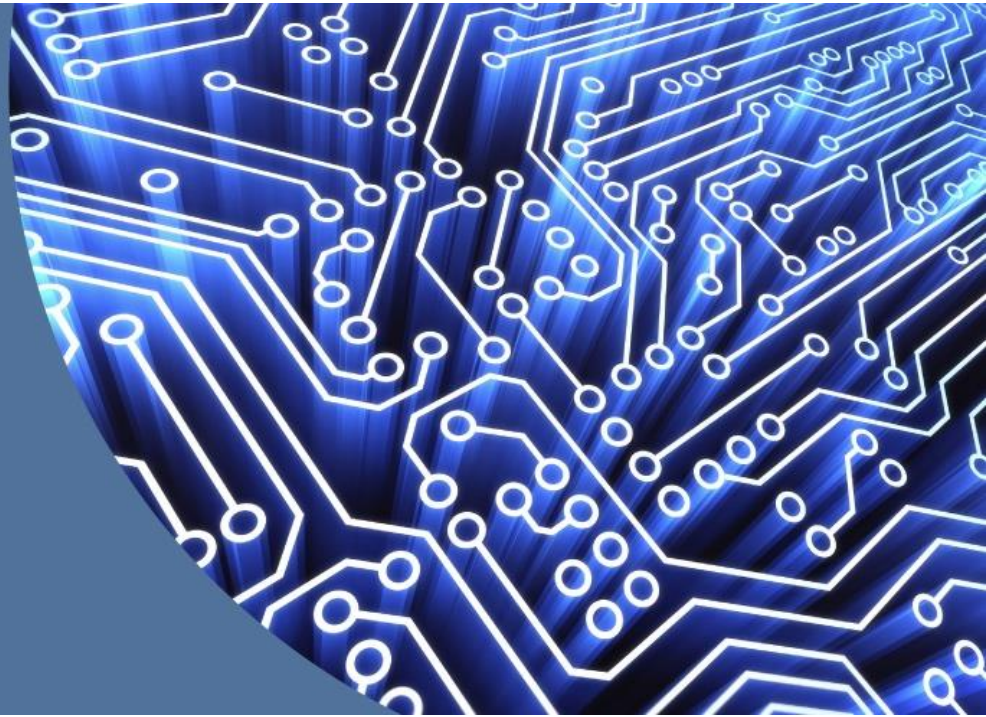
Every Cycle Counts

Gajinder Panesar, CTO, UltraSoC
gajinder.panesar@ultrasoc.com
Iain Robertson, VP Engineering, UltraSoC
Iain.robertson@ultrasoc.com

RISC-V Summit December 2019

- Overview
- Retirement Delta Trace
- Trace Encoder Interface
- Algorithm
  - Filtering
- 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 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

[1] The Economic Impacts of Inadequate Infrastructure for Software Testing, http://www.rti.org/pubs/software_testing.pdf, 2002

Retirement Delta Trace

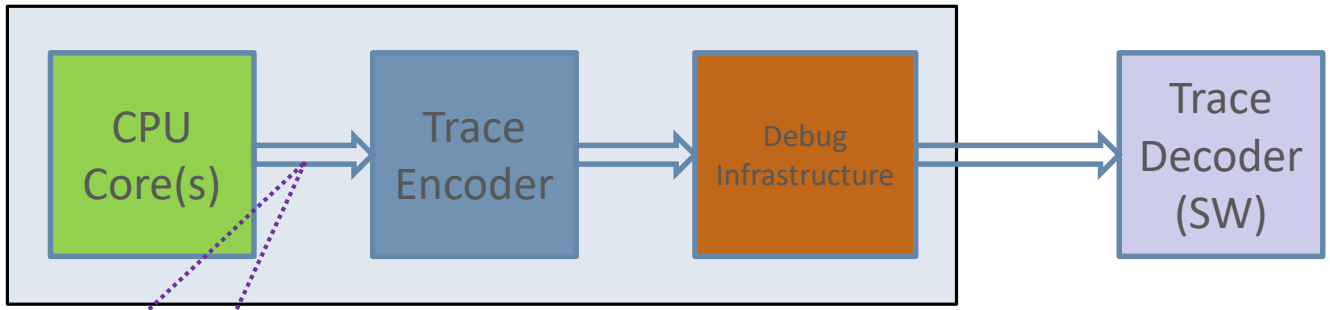
- Stated objective is to be able to determine when each instruction retires, but more importantly when the CPU stalls and does not retire instructions
- Can be achieved by reporting the number of clocks cycles between each successive retirement
- The expectation is the CPU will be able to retire one 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

Retirement Delta Trace

- 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

[2] https://commons.wikimedia.org/wiki/File:Fibonacci,_Elias_Gamma,_and_Elias_Delta_encoding_schemes.GIF

Current Hart to Trace Encoder Interface



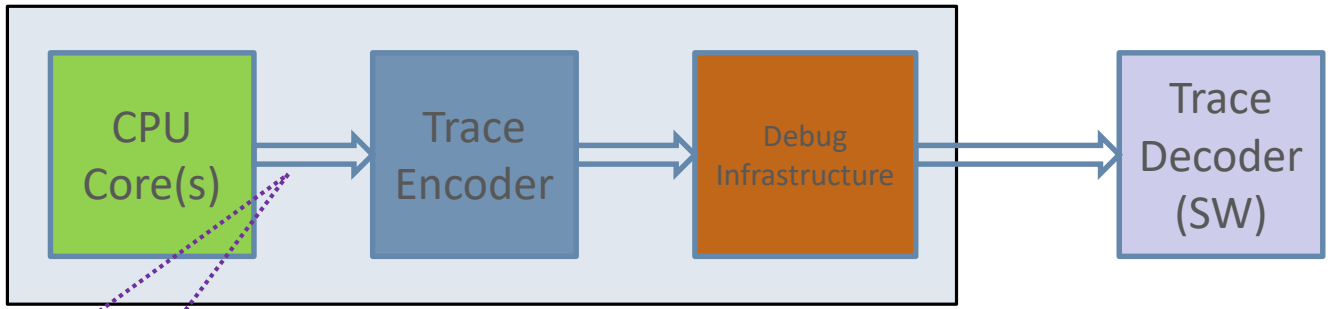
Signal	Function
iretire	Instruction has retired
itype [ITYPELEN-1:0]	Type of instruction
cause [CAUSELEN-1:0]	Exception cause
tval [XLEN-1:0]	Exception data
priv [PRIVLEN-1:0]	Privilege mode during execution
iaddr [XLEN-1:0]	The address of the instruction

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.

Extra Hart to Trace Encoder Interface Signals



Signal	Function
`caretire[CARETIRE-1:0]`	Number of half code words retired in this block

Optional for multi-retirement CPUs for improved efficiency

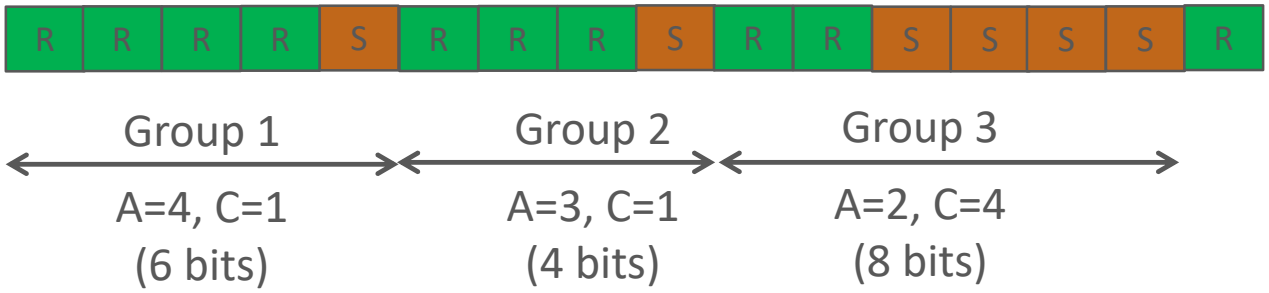
Trace Encoder Output

- 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

Trace Encoder Output – single retirement example

- Basic retirement delta for single retirement harts
- Uses packed groups of {A, C} tokens

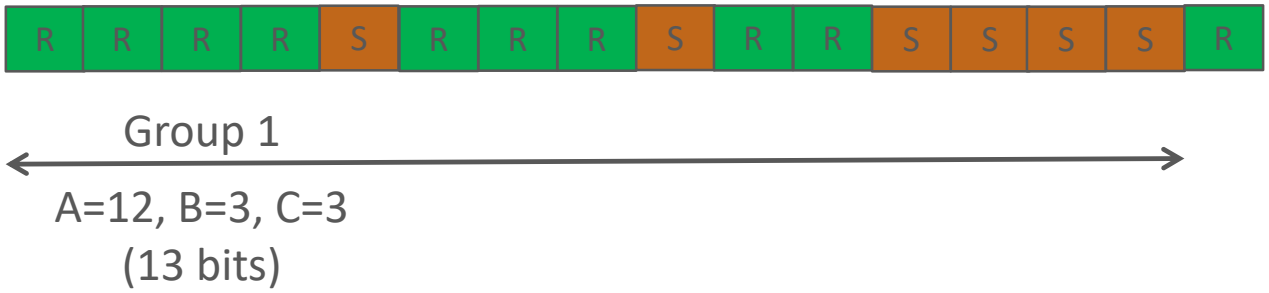


- Three groups, 18 bits in total (2 bits per instruction)

R retire S stall

Trace Encoder Output – stall amortisation example

- Amortise single stalls with retirements
- Uses groups of {A, B, C} tokens

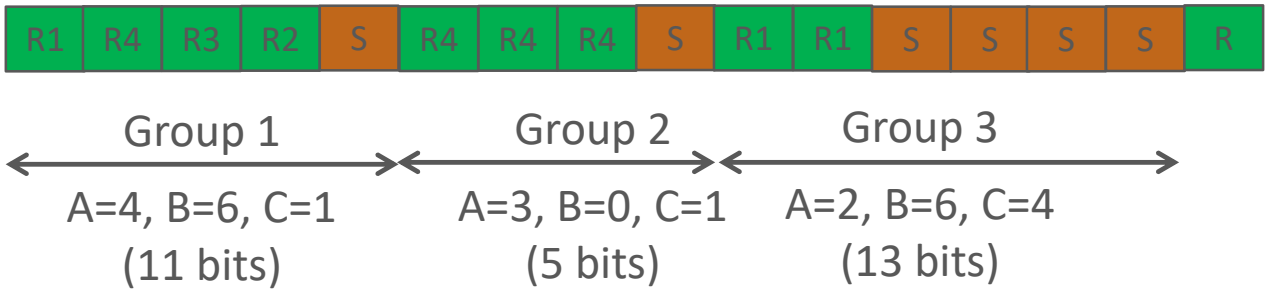


- One group, 13 bits in total – more efficient (1.4 bits per instruction)
- Precise location of single stalls not known – less accurate

R retire S stall

Trace Encoder Output – multi retirement example

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

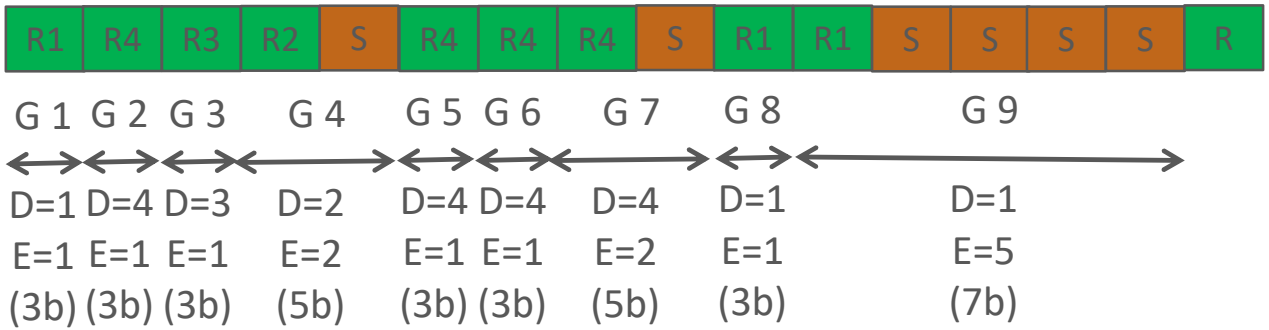


- Three groups, 29 bits in total (1.2 bits per instruction)
- With single stalls amortised: A=12, B=24, C=3; 19 bits (0.8 bits per instruction)

R3 retire 3 instructions S stall

Trace Encoder Output – intra-cycle example

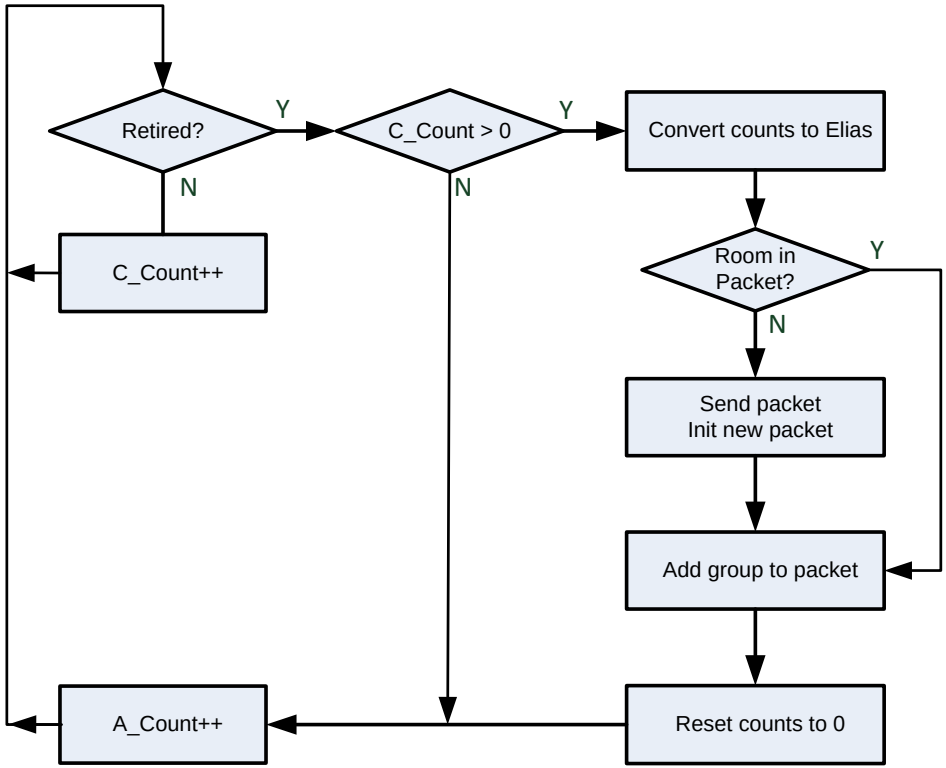
- Per cycle detail for multi retirement harts (4 instructions per cycle in this example)
- Uses packed groups of {D, E} tokens



- Ten 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

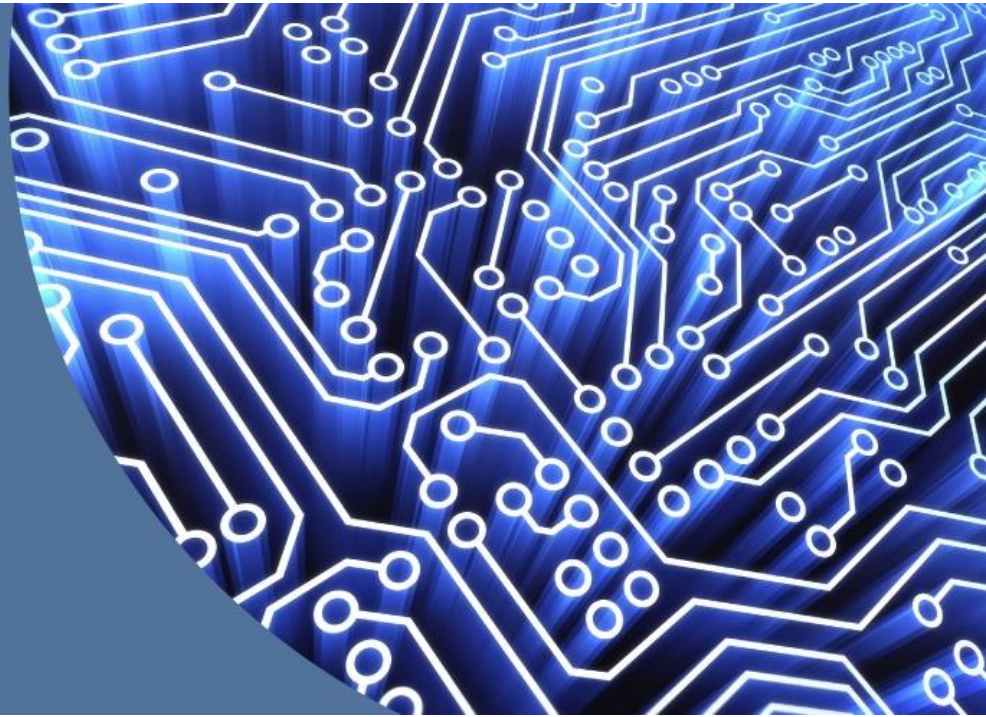


- Shows the {A,C} token case
- Some boundary conditions omitted for clarity
- {A,B,C} and {D,E} cases are similar

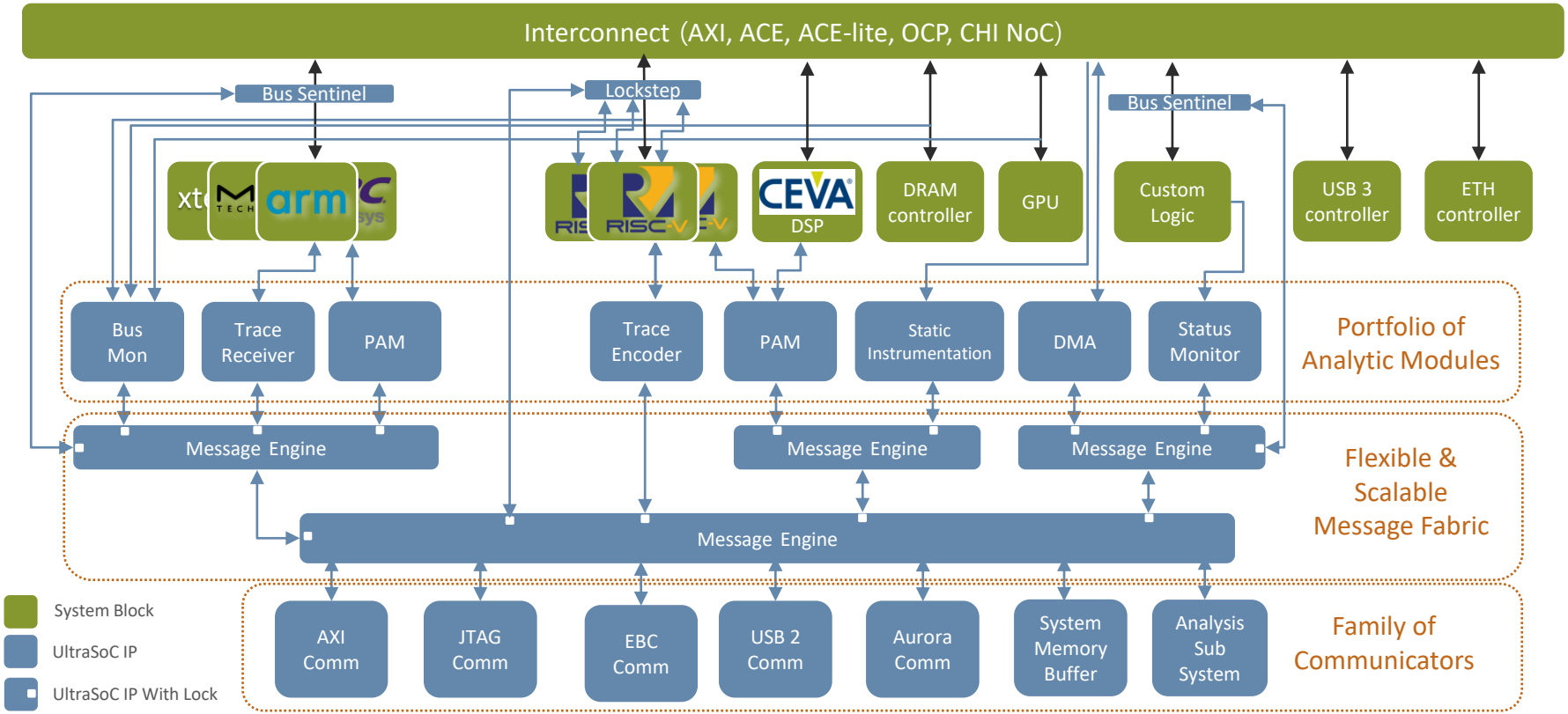
Trace Control

- 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

Holistic System



Monitoring, Reliability, and Security for the Whole SoC



Summary

- 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