It's not about the core, it’s about the system

Gajinder Panesar, CTO, UltraSoC
gajinder.panesar@ultrasoc.com
RISC-V Workshop
18 – 19 July 2018 – Chennai, India
Overview

Architecture overview
Example Scenarios
In-field Analysis/ML
Summary
Demos
• In complex systems understanding the behaviour is not easy
• Surprisingly, systems sometimes do not behave as expected
  • This may be due a number of factors, for example, interactions with cores’ software, peripherals, realtime events, poor implementation or some combination of all of the above
  • Hiring better software engineers is not always an option: you have done that already
  • Oh, RTL engineers introduce bugs too
• Providing visibility of SoC behaviour is important
  • This needs to be done in an intelligent manner and without swamping the system with vast amounts of data
  • Remember the core is a very small part of the overall SoC
Some obvious statements

- SoCs have become increasingly complicated and they are not going to get simpler
  - Contain several (even 1000s) processors, from different vendors
  - Contain 100s of SIP
  - Contain complex interconnects
  - Software created by large disparate teams
  - All this has to successfully work together

- Debugging is more than just Run-control
  - It is more than just CPU centric information such as instructions trace
  - These are important but are only parts of the problem
  - In order for RISCV to be successful it must be useable in systems constructed as above
**Key requirements**

- A vendor-neutral debug, monitoring and analytics infrastructure
  - One that enables access to different proprietary debug schemes used today by various cores
  - Allows for monitors into interconnects, NoCs, interfaces and custom logic
  - These need to be run-time configurable
    - Re-use the hardware to provide visibility for different scenarios
    - Run-time configuration of cross-triggering
    - Support 10s if not 100s of cross-triggering events
  - These can be interrogated after a problem to determine actual status
  - Need to be power aware
  - Security built-in
  - Can be used during the whole development flow and more importantly in the field
Corporate overview

- Founded 2009
- VC-funded start-up
  - 2017 D-round ($7M)
  - New Chairman October 2017
    Alberto Sangiovanni-Vincentelli
- Headquarters in Cambridge UK
- 44 patents
- 32 employees
- Industry leaders adopting UltraSoC
- Silicon-proven with multiple customers
Advanced debug/monitoring for the whole SoC

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

- Bus Mon
- Trace Receiver
- PAM
- Trace Encoder
- PAM
- Static Instrumentation
- DMA
- Status Monitor
- Message Engine
- Message Engine
- Message Engine

- AXI Comm
- JTAG Comm
- USB Comm
- Universal Streaming Comm
- System Memory Buffer

Portfolio of Analytic Modules
Flexible & Scalable Message Fabric
Family of Communicators

System Block
UltraSoC IP
Software tools for data-driven insights

Eclipse based UltraDevelop IDE

- Script based

- Single step & breakpoint
- CPU code & decoded trace

- RISC-V CPU
- Multiple other CPUs
- SW & HW in one tool
- Real-time HW Data
- RISC-V instruction packets
- Real-time HW Data
- SW & HW in one tool
- RISC-V instruction packets
Overview
Architecture overview
Example Scenarios
In-field Analysis/ML
Summary
Demos
Example of UltraSoC Enabled SoC
Example problems UltraSoC solves

- Why is the CPU not performing as fast as expected?
- Why do some DMA transfers take too long?
- What is going on with my memory controller?
- Why does the system hang or deadlock on rare occasions?
- What is the mismatch between the host & the DSP?
Example 1: “Where have my MIPS gone?”

Why is the CPU not performing as fast as expected?

CPU spent cycles

- Compute: 80%
- Stall 1 outstanding: 12%
- Stall 2 outstanding: 8%

Why is the CPU not performing as fast as expected?
Why do some DMA transfers take too long?

What is going on with my memory controller?

- Look at I$ from compute engines
- *Aggregate* bandwidth from each is within spec
- But at Time 2300 Combined peak I$ read request of >2GB/s, cf average of ~570MBs
Example 3: Deadlock detection

• Many different types but consider this as an example
  • CPU (master) asserts arvalid and issues a read address to the Slave
  • Slave asserts rvalid and outputs read data but never sees rready asserted

• Configure bus monitor trace to trigger when transaction duration exceeds threshold (programmable up to 16k cycles)
  • Trace not output until triggered
  • When triggered by deadlocked transaction, trace will output most recent transactions up to and including the deadlocked transaction
  • Trace identifies transaction ID and address, identifying both master and slave of deadlocked transaction
• The monitors continue to function when the system freezes
• The can operate by updating internal circular buffer
• When a system freeze is detected the trace buffers from all the monitors can be extracted
  • The detection of freeze can be done by the monitors themselves
  • For example no transaction in a window
  • Trace not output until triggered
  • When triggered by system freeze transaction, trace will output most recent transactions up to and including the deadlocked transaction
  • Trace identifies transaction ID and address, identifying both master and slave of deadlocked transaction
  • Similar for Status monitor
• Can be considered as a system-wide core dump
  • Use to create known state before hang
  • Send out core-dumps periodically
Metrics generation – Example 1

Runtime Configuration

- Status Monitor configured to count stall triggers from Processor
- Set period of Interval Timer
- Counter values snapshot on expiry of interval timer

Data Flow

1. Stall trigger observed on SM inputs
2. Counter data periodically output from SM
3. Data traced out via USB

![Status Monitor Counter Values](chart)

- Stall Triggers Observed vs Sample Time (ns)
- Data Flow Diagram showing integration with UltraSoC and Debug Hub
Cross-triggering – Example 1

Data Flow

1. Bus Monitor A outputs UltraSoC event when memory access detected
2. Status Monitor receives Stall trigger
3. Event output from SM after transitioning from DMA START -> STALL
4. Trace Receiver(s) and RISCV encoder enabled after receiving event
5. Processor Trace output via USC-Trace Receiver

Example ARM+RISCV System

Interface diagrams for ARM and RISCV cross-triggering.
• The SI provides independent memory-mapped channels (mailboxes)
• Software and hardware can post writes to these channels which can be used to understand system wide behaviour
• The data is timestamped
  • Or no data only timestamp
• The channels can be filtered
• Each channel can be enabled to provide events which can be used for cross-triggering
• The Virtual Console provides bi-directional channels
Simple SI visualization

<table>
<thead>
<tr>
<th>Time</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>100</th>
<th>110</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMA-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TURBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Event
Overview
Architecture overview
Example Scenarios
In-field Analysis/ML
Summary
Demos
Actionable insights across the whole SoC

UltraSoC delivers actionable insights

With system-wide understanding

From rich data across the whole SoC

UltraSoC enables full visibility of SoC
Non-intrusive latency-bandwidth correlation

- Shows how bandwidth and latency are cross-correlated
- Interested in masters: this is where latency is “consumed” affecting master operation
- Interested in reads mainly: master will have to wait for read results, writes less critical
- Presented in a heat map diagram
- For example: on the diagram shown, all CPU latencies are affected by DMA bandwidths
Non intrusive anomaly detection

- Three CPU plots below show CPU cache-like traffic for 3 CPUs configured with different miss rates
- Excessive (anomalous) latencies are shown in red
Non-intrusive profiling with anomaly detection

- Traditional profilers are inadequate:
  - Sampling = miss subtle or fast events (Nyquist)
  - Performance impact/intrusive
  - “Heisenbugs”
- UltraSoC is non-intrusive
- UltraSoC is wirespeed (100% coverage)
- Analytics and automated anomaly detection to make engineer more efficient
Non-intrusive stuck pixels detection

Fastest time to detection

Incoming image

Detected stuck pixels
Overview
Architecture overview
Example Scenarios
In-field Analysis/ML

Summary
Demos
• The challenge today is Systemic Complexity
  • Processor-processor interactions
  • HW/SW interactions
  • Interconnect, NoC & deadlock
  • Long-tail bugs dominate performance – but are hard to detect
• UltraSoC provides a completely scalable coherent analytics, monitoring and debug system
• UltraSoC is system wide, non-intrusive, wire-speed
• Analytics and ML help engineer identify subtle problems efficiently
Overview
Architecture overview
Example Scenarios
In-field Analysis/ML
Summary

Demos
Demo System Architecture

- **Zynq ZC706 FPGA platform**
  - ARM
  - Plus RV32 RISCV
  - 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

Control configuration

Bus activity

Trace Packets