Leveraging on the Nexus 5001 Debug Standard for Debugging Manycore Systems

Presented by:
Mats Fredriksson
Andreas Magnusson
Outline

• Manycore debug & optimization challenges
• Solutions at hand & why Ericsson chose Nexus
• Nexus 5001 Debug Standard Overview
• Manycore specific problems
• Resolving the problems
• Nexus in Ericsson’s Baseband SoCs
• Summary
• Looking forward
• Cooperations & references
Manycore debug & optimization challenges

- SoCs with very high degree of integration with ~100 cores
- Heterogeneous core types
  - different ISAs, I/Fs, etc.
- Very low observability as all ‘important’ busses are internal

So how do you debug & performance optimize such a complex system?
Solutions at hand & why Ericsson chose Nexus

Overall Debug needs

✔ Stop mode debugging – JTAG based Target Debugging
☐ Non intrusive observability – HW supported Real Time Trace

So we needed multicore HW Real Time Trace support!

While investigating existing debug solutions, we found the Nexus Standard. It had exactly what we needed & more, so why reinvent the wheel!? – Ericsson joined Nexus 2009!

- Open standard
- Heterogeneous Multicore support
- Time stamping
- Extendable

- DT (Data Trace)
- PT (Program Trace)
- (Nexus Trace Probes)
Nexus 5001 Debug Standard Overview

The Nexus 5001 Forum™ Standard for a Global Embedded Processor Debug Interface

- based on member companies’ long debug & trace expertise
- several compliance classes
- data compression techniques to minimize bandwidth
- heterogeneous multicore support
- standardized PUBLIC and VENDOR defined messages
- parallel and high speed serial trace ports
- aim is to lower costs for probe & SW tools
- versions 1.0 1999, 2.0 in 2004, 3.0 in 2012
- www.nexus5001.org
Nexus Feature Classes

Debug functionality in our baseband SoCs before & after joining Nexus
Nexus Trace Messages example

venus1/dsp7> trdecode tb 0 0xffff
Trace messages found in tb[0x0..0xffff]:

1 BTM TCODE=56 SRC=dsp0 TTYPE=LOAD NWORDS=87
   LADDR=0x0 CADDR=0x300 TSTAMP=0xf9

2 PTBM TCODE=28 SRC=dsp0 BTYPE=IBR ICNT=11
   FADDR=0xd HIST={F,T} TSTAMP=0x12b

3 TRM TCODE=57 SRC=trc0 BFN_HC=0xda10
   BFN=0xe09

*These messages are 'Vendor defined' messages.

Nexus trace msgs

00 1110000000111000
00 000000000010101
00 000000000000000
00 000000110000000
00 000000000000000
11 0000000011111001
00 011000000011100
00 0000001101000001
00 000000001000000
11 010010101100000
00 000011111111001
00 1100111100010101
11 000000011100000

↑
11 => end of msg
Manycore specific problems

- Trace bandwidth bottleneck!
- Potentially massive amounts of trace data to analyze!
- Non-deterministic scheduling of tasks onto cores
- Synchronized global time stamping absolutely necessary for correlation of events when processing is very dynamic
- Time stamping potentially consumes 30% of trace bandwidth
- Stop mode debugging often impossible on complex real-time systems - ‘you can’t stop the world’
- Profiling based on traditional Real Time Trace messages (PT & DT) impossible due to limited bandwidth and dynamic scheduling
Resolving the problems

We deal with the manycore problems by providing:

- Advanced dynamic runtime trace filtering features!
- Flexible trace generation/collection HW architecture
- Global synchronized time stamping with bandwidth optimization
- Stop mode debugging enabled by selective system freeze
- HW Profiling counters count events on all cores using limited bandwidth

The above topics will be discussed in more detail on the following slides
Nexus in Ericsson’s Baseband SoCs
Advanced dynamic runtime trace filtering (1)

‘Only trace what is relevant’ with:

- Local HW supported dynamic runtime filtering
  - 3 runtime filters can selectively control what is traced
  - SW provides info for filtering during runtime, debugger sets match and mask registers
    Ex. trace the processing of a specific mobile phone call on the manycore system

- Global runtime filtering using cross triggers
  - trace/profiling windows

**Note:** Trace filtering features can also be used in advanced BPs that only trigger when executing in a particular context.
HW supported runtime filtering of SW traces

SW trace filtering is needed since there are many types of SW traces that consume a significant amount of trace BW. Additionally are the SW traces too static as SW enabling of traces is too costly.

- Nexus DQM (Data Acquisition Msg) carry SW traces
- 32 bit DQM mask filters between 32 SW trace types

Ex. DQM mask = 0x000100ff => only SW trace types 0-7 & 16 will create DQM messages
### Nexus in Ericsson’s Baseband SoCs

Advanced dynamic runtime trace filtering (3)

#### Hardware Registers

<table>
<thead>
<tr>
<th>Static RTTCNF</th>
<th>PT</th>
<th>DT</th>
<th>OT</th>
<th>..</th>
<th>WP</th>
<th>DQMSMASK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic RTTCNF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DYN enable**

**Ex. PT is on if (static PT || (DYN && dyn PT))**

#### Static Trace Mode
- ‘Background’ trace mode, ex.
  - \( DQMSMASK = 0x000300f \)
  - \( PT=0, \; DT=0 \)
- \( \Rightarrow \) Only 6 types of SW traces

#### Dynamic Trace Mode
- SW assisted ‘detailed’ tracing
  - DYN trace propagated to child tasks
  - Tasks disable DYN upon completion
- trace ‘execution trees’, ex.
  - \( DQMDMASK = 0x0000927f \)
  - \( PT=1, \; DT=1 \) (system parameters)
- \( \Rightarrow \) PT + DT + 10 types of SW traces

---

**ERIČSSON**

Leveraging on the Nexus 5001 Debug Standard for Debugging Manycore Systems

**MULTICORE DEVELOPERS CONFERENCE**

12
Dynamic trace mode propagates to all child tasks (red)

Static background trace on everything NOT red marked

SW turns on ‘dynamic’ trace
Nexus in Ericsson’s Baseband SoCs
Flexible trace generation/collection HW architecture

Tracing in field
- Continuous trace to circular buffers (PMD)

Tracing in dev env
- stream to ETH or EM
- ‘probe less’ debugging using on board JTAG controller
- (Trace probe – cost & cabling issues)

Trace modes
- Non-intrusive (ERR msg)
- Minimally intrusive (halt exec during trace bursts)

Arch supports extreme high BW (short periods) to Low-Med BW trace (long periods)
Global synchronized time stamping with bandwidth optimization

- Global synchronized time
- Short timestamp from trace clients
- Time synchronization msgs ‘when needed’ save 30% trace BW!

Note: This is not part of Nexus Standard 2012 but something similar and improved is being discussed by the members for the next standard release.

### ‘Raw’ trace messages

1. ptbm src=dsp2 ... tstamp=0x201
2. otm src=dsp1 ... tstamp=0x21c
3. trm src=tc1 tsync=0x1304000
4. otm src=dsp22 ... tstamp=0x022

### Full timestamps after decoding

1. ptbm src=dsp2 ... tstamp=0x1303201
2. otm src=dsp1 ... tstamp=0x130321c
3. trm src=tc1 tsync=0x1304000
4. otm src=dsp22 ... tstamp=0x1304022
Nexus in Ericsson’s Baseband SoCs
Stop Mode Debugging Enabled by Selective System Freeze

System freeze feature are based on Cross Triggers

Cross Triggers

• can be set by BPs/WPs or SW
• global on the board level
• CTs can be used as/to:
  – synchronously stop selected cores
  – preserve mem state to debug (protection mask)
  – stop trace collection
  – SW interrupts
  – conditions in BPs
  – for external measurements
  – incoming events (e.g. from a Logic Analyzer)
Nexus in Ericsson’s Baseband SoCs

HW Profiling counters count events on all cores using limited BW

• Breakpoint counters can be reused for profiling

• Each core has 4 16bit counters that can be cascaded to count events for nanoseconds up to years

• Event counting in accumulated or sampled modes

• Counter overflows or sampling results be flushed out using RTT messages, thus reducing the number of counter bits needed

• Complex triggers & context qualifiers can be used as count windows

• Profiling counters only count when cores execute

• Supports the ability to count a large variety of events including: cycles, various stalls, WP hits, runtime context qualifiers, cross triggers
Summary (1)

Traditional Stop Mode Debugging – often too intrusive & limited but useful if you have:

- Advanced Program/Data breakpoints support
- Synchronous Execution Control – Start, Stop, Step
- HW support for freezing relevant parts of the debug state

RTT (Real Time Trace) is however the main debug/observability paradigm for debugging/optimizing our baseband SW!

Dynamic runtime filtering and profiling counters are required to tackle bandwidth/dataset problems

- Simplicity - Tools need to help hide the trace setup complexity
- Availability – RTT in simulators and prohless debugging
Benefits of joining Nexus

- gave Ericsson an excellent debug platform to build upon
- some trace issues that we and other member companies have seen, are being addressed in the Technical Committee and may very well be improvements to the next Nexus standard release
- the Technical Committee meetings have been a great way for members to share ideas and experience
Looking forward

Manycore Debug Area

- We expect to see more & more on-chip debug HW
- More advanced trace analysis tools
- HW supported analysis of live streaming trace data

Possible new features in coming Nexus standard releases

- New & more efficient time stamping schemes
- Alternative message framing mechanisms for more efficient high speed serial trace
- Standardize a trace message definition format to enable generic decoding tools and interchange of trace definitions
Cooperations & References

Ericsson is involved in many standards, working groups and in Open Source. Here are some references to such work and more information about on-chip instrumentation in general and the Nexus.

- Multicore Association Tool Infrastructure Working Groups

- For information & references to Ericsson’s involvement in the of these areas: GDB, CTF, CDT, LTTng, Eclipse Tracing & Monitoring Framework
  http://www.eclipsecon.org/.../EclipseCon%202013%20Troubleshooting%20Real%20World%20Software.ppt

- On-Chip Instrumentation – Design and Debug for Systems on Chip, Neal Stollon

- Nexus 5001 Debug Standard Overview, White Papers (including a white paper related to this presentation, hopefully available Q3 2013)
  http://www.nexus5001.org/
Backup Slides

Trace Usage Examples
Execution overview
Real Time Trace usage examples (2)

Time Synchronized Execution + trace log + search views

Leveraging on the Nexus 5001 Debug Standard for Debugging Manycore Systems
Real Time Trace usage examples (3)

Program flow view (reconstructed from RTT PT + object file)
Real Time Trace usage examples (4)

Program Flow Gantt Chart (reconstructed from RTT PT or thru SW instr)
Real Time Trace usage examples (5)

CPU Usage Report

Trace report examples