XDS560 Trace
Technology Showcase

Daniel Rinkes
Texas Instruments
Agenda

• AET / XDS560 Trace Overview
• Interrupt Profiling
• Statistical Profiling
• Thread Aware Profiling
• Thread Aware Dynamic Call Graph
Agenda

- AET / XDS560 Trace Overview
- Interrupt Profiling
- Statistical Profiling
- Thread Aware Profiling
- Thread Aware Dynamic Call Graph
What is XDS560Trace?

- XDS560 Trace is a combination of hardware and software that allows non-intrusive logging of CPU Program and Data Bus activity.
- Data is gathered and compressed on the target, and passed up to the XDS560 Trace Pod via an additional 15 EMU Pins (Currently uses up to 12 total).
- The data is then decompressed on the host and displayed for the user.

XDS560 Trace is Not

- Visibility into CPU Register Values/Reads/Writes
- Visibility into EDMA Activity
- Multi Core
Trace Architecture

- CPU
- DSP
- XDS560T POD
- Trace & AET Jobs
  - Comparators
  - Compressor
  - Cycle counter
- Additional JTAG Emulation Pins
- • Current Buffer Size: 224K
  • Future: 64 MB
- XDS560T POD RECORDING UNIT
  - To/from host PC
OK...so what is AET

- Advanced Event Triggering (AET)
  - On-Chip programmable logic that can be programmed to generate triggers based on application events or combinations of application events
AET Hardware Overview

- State Resources
  - State Machines
  - Counters

- Event Detection Resources
  - Processor Address and Data
    - Pipeline Flattener
  - Auxiliary Event Input Circuits
  - Auxiliary Event Generators

- Event Generation
  - Event Combination
    - Trigger Generation
      - Through Event Combination (Trigger Builders)
      - Trigger Generation Resources

- Triggers
  - Reload/Count
    - Next State
  - Save Events
    - Trace Enables (PC, Read, Write)
  - Trace End
  - Program Halt
  - External Triggers
AET Job Examples

• Trace PC whenever the PC is between location A and B (PC Trace in Range)
• Trace all writes to memory location 0x12345678 (Data Trace)
• Halt whenever the application executes the instruction at location 0x801000, but only after it has executed the instruction at 0x804060 (Hardware Breakpoint qualified by a Sequencer State)
• Halt when the value 0x0001234 is written to location 0x80500000 (Halt on Data Write qualified by value)
Required Software and Hardware

- 60 Pin emulation header
- Target must support Trace (Full-Gem)
- Blackhawk USB 560
- XDS560T Trace Pod/Cable
- CCS 3.30 or higher
XDS560 Trace Use Cases

Debugging

• Branch to uninitialized memory
• Invalid Writes to specific locations in memory
• Order of execution
• Interrupts Disabled
• ...
Profiling & Application Analysis

• Find the most cycle intensive functions in an application
• Display a Cycle Accurate Thread Execution Graph
• Provide Function Profile information in a thread specific context

All in a matter of minutes
Profiling Advantages Over Traditional Methods

• Memory/Cache/Peripheral Effects all inherently considered
• Real-Time, Cycle Accurate, and Completely Non-Intrusive to the application
• It’s fast…results in minutes
Agenda

- AET / XDS560 Trace Overview
- Interrupt Profiling
- Statistical Profiling
- Thread Aware Profiling
- Thread Aware Dynamic Call Graph
Interrupt Profiling Overview

- Capture Program Address and Timestamp whenever the PC is within the Interrupt Vector Table
- Generate a cycle accurate picture of when each interrupt starts executing
- Graphically display interrupt cycle accurate interrupt servicing frequency
Cycle Count 102460

Trace Log

<table>
<thead>
<tr>
<th>PC</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00897C60</td>
<td>102456</td>
</tr>
<tr>
<td>0x00897C64</td>
<td>102457</td>
</tr>
<tr>
<td>0x00897C68</td>
<td>102458</td>
</tr>
<tr>
<td>0x00897C6C</td>
<td>102459</td>
</tr>
<tr>
<td>0x00897C70</td>
<td>102460</td>
</tr>
<tr>
<td>0x00897C74</td>
<td>102461</td>
</tr>
<tr>
<td>0x00897C78</td>
<td>102462</td>
</tr>
<tr>
<td>0x00897C7C</td>
<td>102463</td>
</tr>
</tbody>
</table>
Data Captured

Raw Trace Data

Processed Data

Technology for Innovators™

Texas Instruments
Results
### Results - 2

The image shows a computer interface with various windows and panels. The interface appears to be related to a software tool, possibly for debugging or monitoring. The main window displays a timeline with data points labeled `hmx14`, `hmx9`, and `hmx7`. Below the timeline, there is a statistics table that lists symbols such as `hmx14`, `hmx7`, and `hmx9` with their respective minimum, maximum, average, and total values.

```
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Count</th>
<th>Min</th>
<th>Max</th>
<th>Average</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>hmx14</td>
<td>2192</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2208199035.00</td>
</tr>
<tr>
<td>hmx7</td>
<td>649</td>
<td>412834.00</td>
<td>12129868.00</td>
<td>3510691.99</td>
<td>2272194139.00</td>
</tr>
<tr>
<td>hmx9</td>
<td>1301</td>
<td>172247.00</td>
<td>1272947.00</td>
<td>178661.61</td>
<td>2272367725.00</td>
</tr>
</tbody>
</table>
```
Agenda

• AET / XDS560 Trace Overview
• Interrupt Profiling
• Statistical Profiling
• Thread Aware Profiling
• Thread Aware Dynamic Call Graph
Statistical Profiling

GOAL
Get a quick overall view of which functions in an application consume the most cycles

- Sampling every Program Counter in an application quickly consumes Trace Buffer Bandwidth, preventing analysis of the entire application
- We can eliminate this problem by only capturing a statistical sample of application execution
Statistical Profiling Overview

- The Program Address is sampled at regular intervals
- Statistical Analysis is performed on the captured samples
- As in any statistical analysis, the determinations made on the statistical sample can be related to the general population
Statistical Profiling - 2

- AET contains all of the hardware needed to capture trace samples at a specified interval
- Interval should be carefully chosen so as not to coincide with a periodic function
- Application instrumentation can switch AET off in locations that are not of interest
Statistical Profiling - Results

- Comma Separated Value Format
- Sorted from most intensive functions to least
Agenda

• AET / XDS560 Trace Overview
• Statistical Profiling
• Interrupt Profiling
• Thread Aware Profiling
• Thread Aware Dynamic Call Graph
Thread Aware Profiling

GOAL
Generate a cycle accurate execution graph of a Thread/Task based application
Solution

• Instrument the task/thread switch function to write the task/thread ID to a well known location (global variable)
  – Operating systems typically provide hooks to insert functions in this location

• Trace all of the writes to that location, and get a timestamp with each.
Results
Agenda

• AET / XDS560 Trace Overview
• Statistical Profiling
• Interrupt Profiling
• Thread Aware Profiling
• Thread Aware Dynamic Call Graph
Thread Aware Dynamic Call Graph

GOAL
Display a Thread Based representation of actual function execution in an application.
Capturing the Data

• Thread/Task
  – Hook function writes address of task

• Function
  – Entry/Exit points instrumented with Mark 0/Mark 1 instructions*
    • Mark 0 inlined at each function entry point
    • Mark 1 inlined at each function exit point
  – Trace captures each of these locations with timestamp
    * CGT 6.0.1 enables function hooks
• Graphical Displays become impractical as the number of functions increase

• At right is a sample call graph displayed by the Guess graphing package
GPROF Like Format

- Modeled after Unix GPROF
- Each Thread separated into it’s own subsection
- Each function section contains only immediate callers and callees
## A Closer Look

Thread: 0x00828194

<table>
<thead>
<tr>
<th>index</th>
<th>excl_cycles</th>
<th>called</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>14491050</td>
<td>388/388</td>
<td>_DEC_tcp2PreProc [2]</td>
</tr>
<tr>
<td></td>
<td>14491050</td>
<td>388</td>
<td>_DEC_tcp2DeintUnpunctSoft3 [0]</td>
</tr>
<tr>
<td>32580</td>
<td>14491050</td>
<td>388</td>
<td>_dst0Isr [8]</td>
</tr>
<tr>
<td>12154</td>
<td>48/48</td>
<td>389</td>
<td>_edmaIsr [7]</td>
</tr>
<tr>
<td></td>
<td>9049502</td>
<td>389</td>
<td>_DEC_tcp2QuantizeSoft [1]</td>
</tr>
<tr>
<td>26587</td>
<td>49/146</td>
<td>389</td>
<td>_edmaIsr [7]</td>
</tr>
<tr>
<td></td>
<td>4598471</td>
<td>389</td>
<td>_DEC_tcp2PreProc [2]</td>
</tr>
<tr>
<td></td>
<td>14491050</td>
<td>389</td>
<td>_DEC_tcp2DeintUnpunctSoft3 [0]</td>
</tr>
<tr>
<td></td>
<td>9049502</td>
<td>389/389</td>
<td>_DEC_tcp2QuantizeSoft [1]</td>
</tr>
<tr>
<td></td>
<td>3590771</td>
<td>389/389</td>
<td>_varianceEstim [3]</td>
</tr>
<tr>
<td>86691</td>
<td>3590771</td>
<td>389</td>
<td>_COM_spoolPost [5]</td>
</tr>
<tr>
<td>12342</td>
<td>49/146</td>
<td>389</td>
<td>_edmaIsr [7]</td>
</tr>
<tr>
<td></td>
<td>3590771</td>
<td>389</td>
<td>_varianceEstim [3]</td>
</tr>
</tbody>
</table>
Future Display – Tree View

Thread: 0x828194

_DEC_tcp2DeintUnpunctsoft3

_CALLERS

_DEC tcp2QuantizeSoft

_DEC tcp2PreProc

_variance Cycles = 388/388

_COM_spoolTask

_Cycles = 14491050

_COM_spoolPost

_DEC_tcp2Edmalsr
Questions?