Generating Workload Models from TLM-2.0-based Virtual Prototypes for Efficient Architecture Performance Analysis

Tim Kogel
NASCUG 13
June 13, 2010
Outline

• Motivation and TLM-2.0 Virtual Prototyping Use Cases

• Generation of Workload Models from Virtual Prototypes

• Case Study
Hardware and Software Challenges

Architectural challenges
- Dynamic application workloads
- Resource sharing and arbitration
- High risk
  - Risk of under-designing
  - Risk of over-designing

Software challenges
- Multi-core
- Exploding SW content
- Late integration

Top 5 Tasks Causing Project Delays

<table>
<thead>
<tr>
<th>Task</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project management/planning</td>
<td>44.7%</td>
</tr>
<tr>
<td>System architecture design &amp; spec.</td>
<td>43.4%</td>
</tr>
<tr>
<td>Firmware development/test</td>
<td>34.9%</td>
</tr>
<tr>
<td>Board-level eng. /test</td>
<td>32.9%</td>
</tr>
<tr>
<td>System integration/test</td>
<td>28.9%</td>
</tr>
</tbody>
</table>

Source: VDC
Software Debug and Analysis

Using Loosely Timed Virtual Prototypes

+ High simulation speed
+ SW debug and analysis
+ Some level of timing in AT mode
  – Timing not reliable for architecture analysis
Hardware/Software Validation

Using cycle accurate Virtual Prototypes

+ High accuracy
+ Joint HW and SW analysis
  - Real SW required
  - Difficult to set up performance critical corner cases
  - Low simulation speed
Architecture Analysis

Using partial virtual prototypes and non-functional workload models

+ Reduced effort to capture prototype
+ Requires profiling information, but porting of real SW not required
+ Ideal for performance optimization of SoC backbone (interconnect/memory)
+ Faster simulation than cycle-accurate ISS

– How to model the traffic?
Outline

• Motivation and TLM-2.0 Virtual Prototyping Use Cases

• Generation of Workload Models from Virtual Prototypes

• Case Study
Workload Generation Flow

VP for Software Analysis

IA ISS  L1$

IA ISS  L1$

L2$

Mem

LCD

Uart

Software Analysis Results

simulation

post-processing

VP for Hardware Performance Analysis

Hardware Performance Analysis

stl

VP for Hardware Performance Analysis

Mem

simulation

exploration
Generating "Elastic" Trace Files

Socket Transaction Language (STL, from OCP-IP)

- Generate read and write transactions
- Wait for outstanding transactions or cycles
- Synchronize with other STL files

main.stl

```plaintext
... 0 read 0x37c
  wait response = -1
idle 10
#C: call start..
raise swi=4
wait swi=126
#C: returned from..
0 write 0x4039c 0x1
...```

© Synopsys 2010
Generating "Elastic" Trace Files

main.stl

0 read 0x37c
wait response = -1
idle 10
#C: call start_initial_dma_ch0.stl
raise swi=4
wait swi=126
#C: returned from ..
0 write 0x4039c 0x1
...

start_initial_dma_ch0.stl

wait swi=4
0 read 0x444
wait response=-1
...
#C: call SetChannelDmaSize.stl
raise swi=5
wait swi=124
#C: returned from ..
...
raise swi=126

SetChannelDmaSize.stl

wait swi=5
...
0 read 0xcf8
wait response=-1
0 write 0xc0000044 0x100
wait response=-1
0 read 0xcfc
...
raise swi=124
Outline

• Motivation and TLM-2.0 Virtual Prototyping Use Cases

• Generation of Workload Models from Virtual Prototypes

• Case Study
  – Simple Example Platform
  – ARM Versatile Board
Simple AHB Example Platform

• Goal
  – Validate expressiveness of performance results by comparing FRBM and ISS based virtual prototypes

• Scenario
  – CPU programs DMA
  – DMA loads picture
  – CPU filters and stores picture

• Experiment
  – Record STL trace with ARM 968 LT ISS and AT bus
  – Replace AT bus with cycle accurate AHB bus and modify memory latency
  – Replace ISS and DMA with 3 GFRBMs
  – Compare analysis results and measure accuracy of traces
## AHB Results

<table>
<thead>
<tr>
<th>ISS</th>
<th>FRBM</th>
<th>ISS</th>
<th>FRBM</th>
<th>FRBM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM. p_AHB-20131.0</td>
<td>dma_DMA_0-41758.0</td>
<td>dma_DMA_1-no data</td>
<td>cpu_POST=23068.0</td>
<td>dma_DMA_0-20480.0</td>
</tr>
<tr>
<td>20131.0</td>
<td>14758.0</td>
<td>23068.0</td>
<td>20480.0</td>
<td>22342.0</td>
</tr>
<tr>
<td>15626.0</td>
<td>7856.0</td>
<td>12376.0</td>
<td>1048.0</td>
<td>5468.0</td>
</tr>
<tr>
<td>22590.0</td>
<td>21378.0</td>
<td>21576.0</td>
<td>20480.0</td>
<td>22342.0</td>
</tr>
<tr>
<td>7421.0</td>
<td>15469.0</td>
<td>7421.0</td>
<td>15469.0</td>
<td>7421.0</td>
</tr>
<tr>
<td>15932.0</td>
<td>256.0</td>
<td>5832.0</td>
<td>256.0</td>
<td>5832.0</td>
</tr>
</tbody>
</table>

Time increases with FRBM.

~11.5% ~25%
AHB Results

~11.5%

~25%

DMA finishes, then CPU starts next DMA

Use functional DMA controller

No synchronization between DMA read + write and between DMA + CPU Trace

Add waiting for HW interrupt to CPU trace
Architecture Exploration Results

- Elastic STL traces predict the correct performance trend
- Missing synchronization points cause increasing error

Adding synchronization points increases accuracy of results

Memory latency in cycles:
- 5 cycles
- 10 cycles
- 20 cycles

Total simulation time ms:
- ISS + DMA
- 3 GFRBM
- GFRBM + DMA
Case Study Results

• Effort: Preparation based on existing platforms
  – 1 week setup to create post-processing script for generating traces from analysis
  – 1 week for running and analyzing results

• Outcome
  – Trace-driven traffic generation methodology is confirmed for performance analysis

• Simulation speed
  – 1-10 MIPS for LT platform with analysis recording
  – 0.1-1 MIPS for AT platform with analysis recording
  – ~1 min of post-processing for 10ms of real-time traffic per initiator
  – 10s to 100s of KCPS of GFRBM-based performance model

• Benefits
  – Generate realistic workloads for architecture analysis and exploration
  – Very good adaptability of traces to Interconnect/Memory
  – Very good accuracy

• Limitations
  – For now system-level synchronization points need to be manually inserted
Ongoing Project: Versatile Platform

Validated STL generation with realistic platform

Next Step: Validate accuracy for different architecture variants

Switch from LT/DMI to AT

CPU transaction trace

CPU context trace

CPU function trace (per context)