

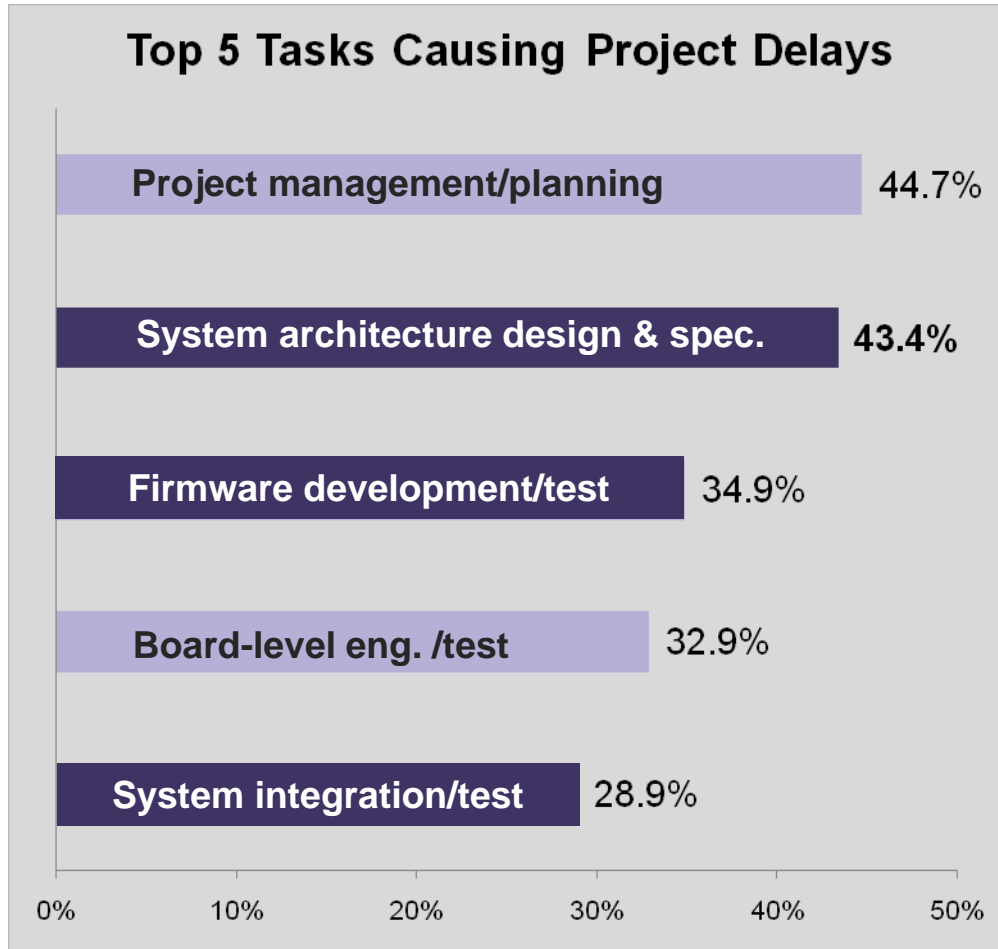
Generating Workload Models from TLM-2.0-based Virtual Prototypes for Efficient Architecture Performance Analysis

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



Source: VDC

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

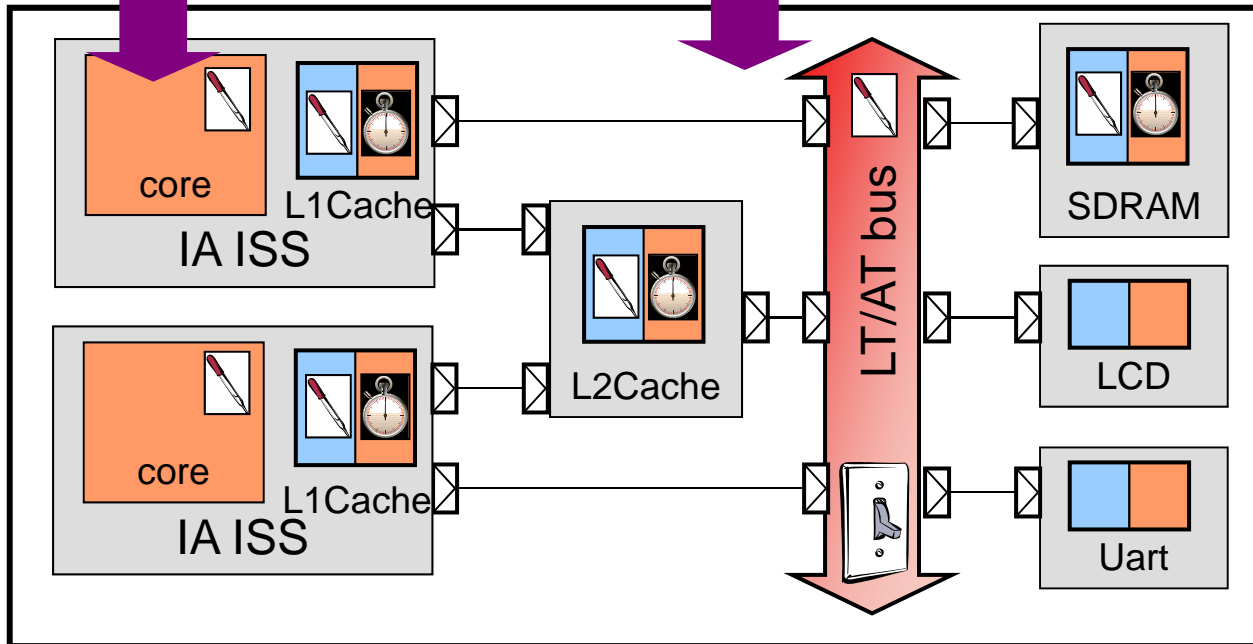
Software Debug and Analysis

Software Debugger

The screenshot shows a debugger window with a code editor at the top displaying assembly-like code. Below it are registers and memory windows. A thread window at the bottom shows the current thread's state.

Software Analysis

The screenshot shows a hardware analysis tool with a central component list including 'HARDWARE_LARM926' and '/kernel'. To the right, there are execution flow diagrams and timing charts.



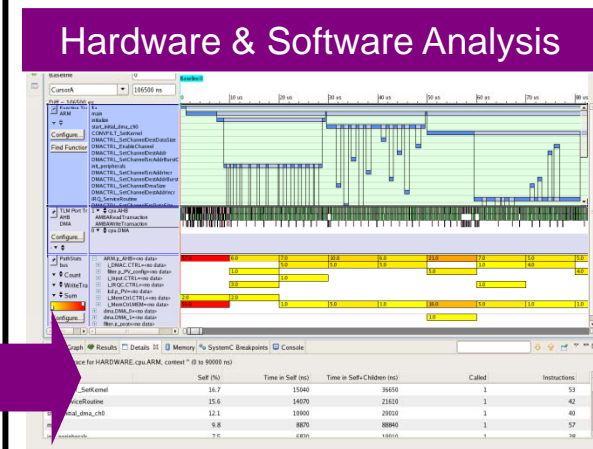
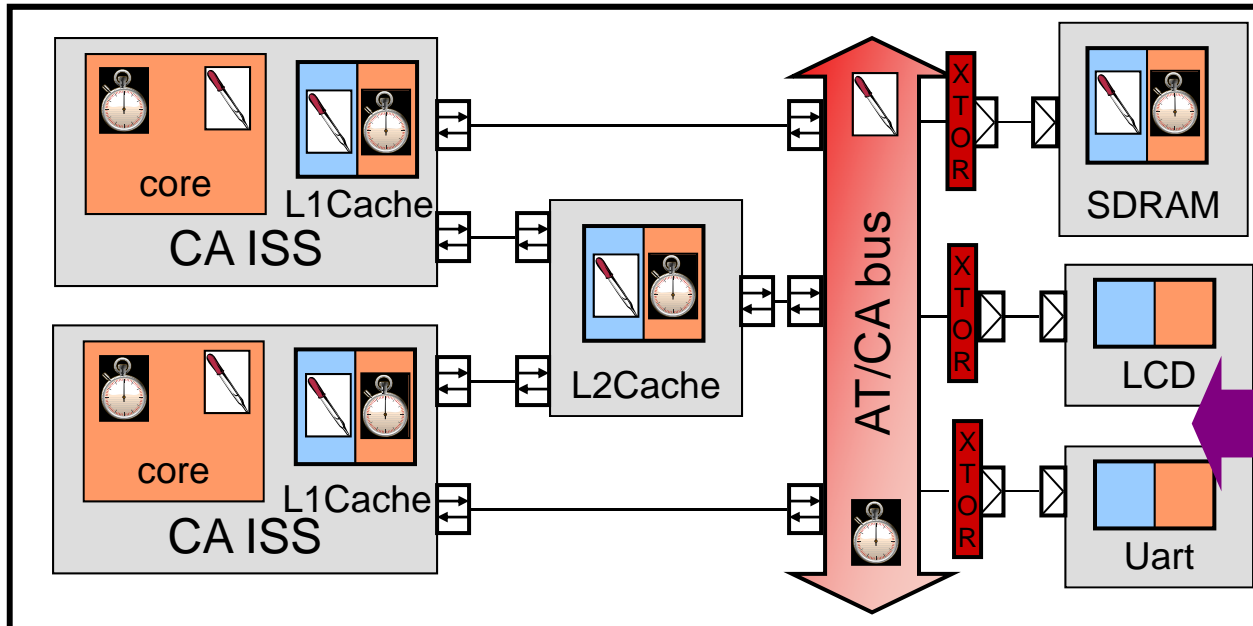
Using Loosely Timed Virtual Prototypes

- + 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

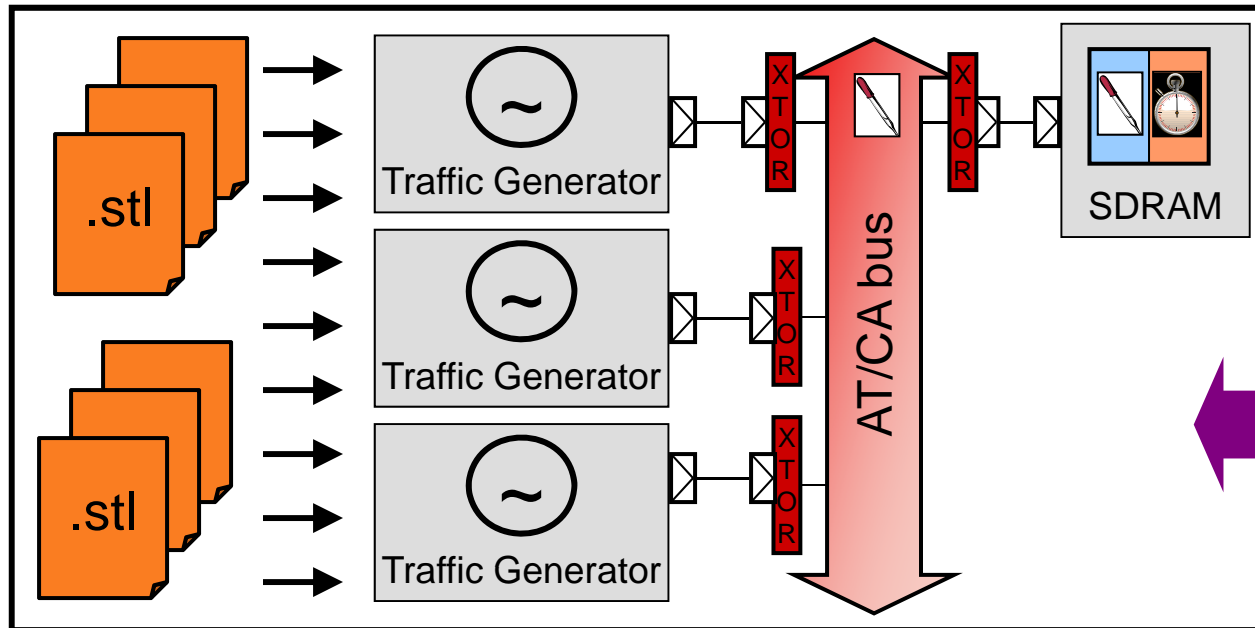
- + High accuracy
- + Joint HW and SW analysis
- Real SW required
- Difficult to set up performance critical corner cases
- Low simulation speed



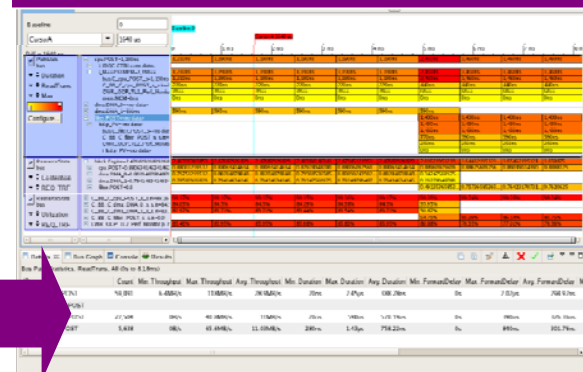
Architecture Analysis

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?**



Hardware Performance Analysis

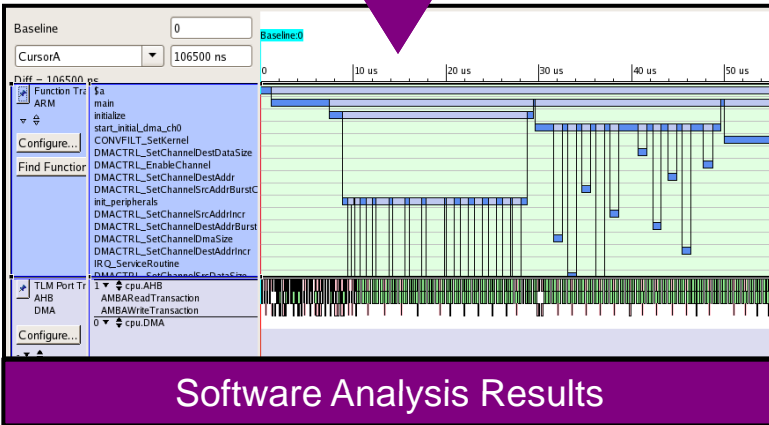
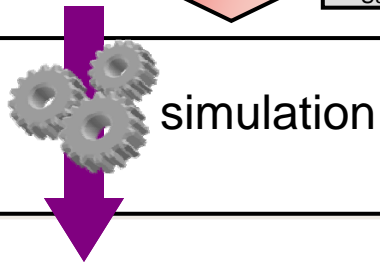
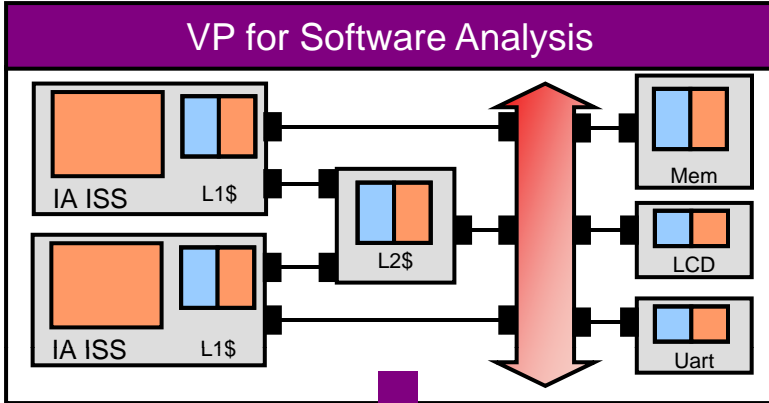


Outline

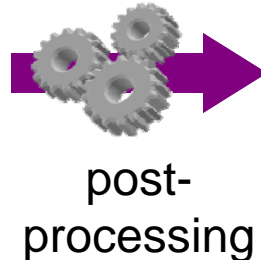
- Motivation and TLM-2.0 Virtual Prototyping Use Cases
- Generation of Workload Models from Virtual Prototypes
- Case Study

Workload Generation Flow

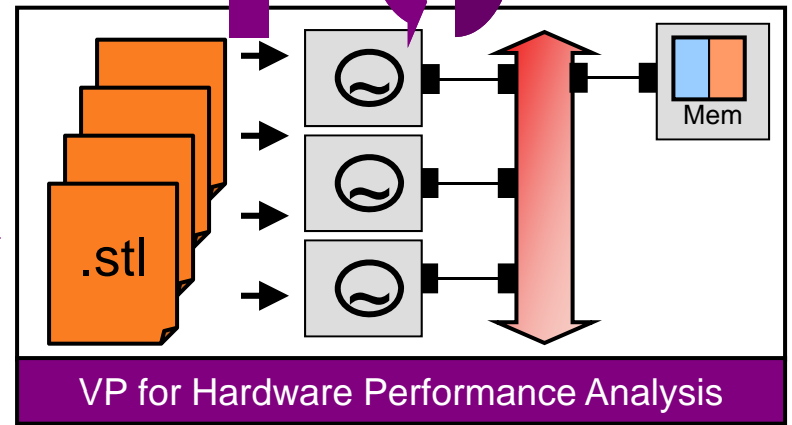
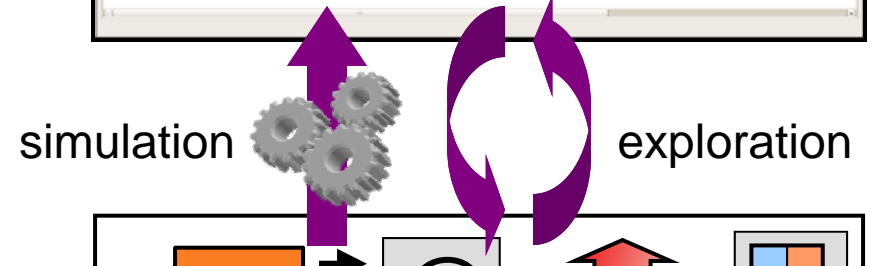
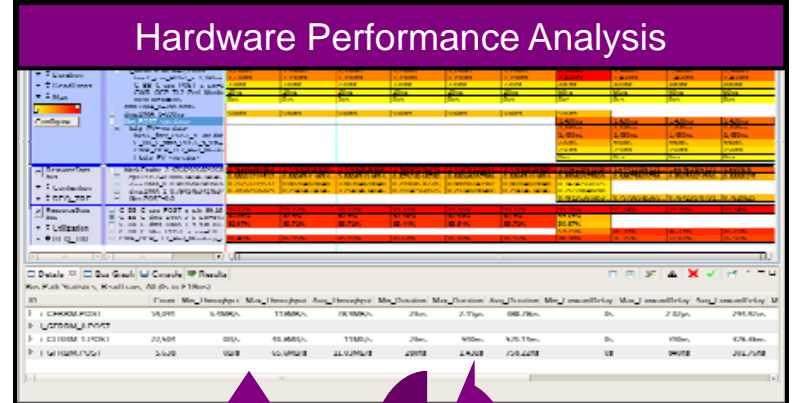
VP for Software Analysis



Software Analysis Results

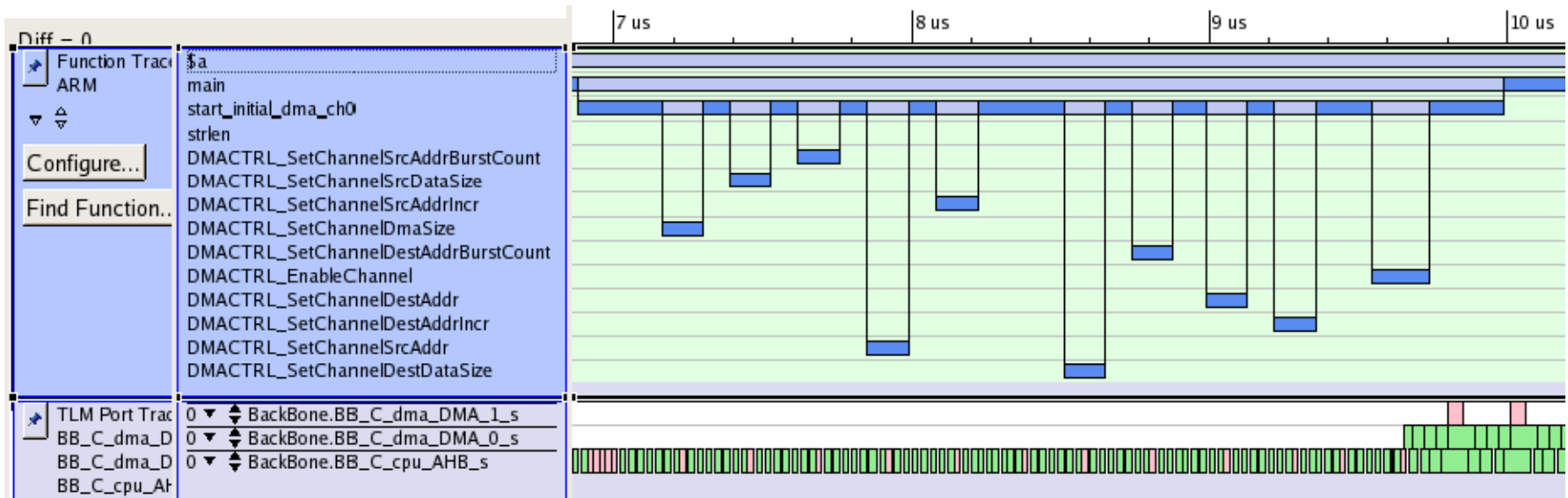


Hardware Performance Analysis



VP for Hardware Performance Analysis

Generating "Elastic" Trace Files



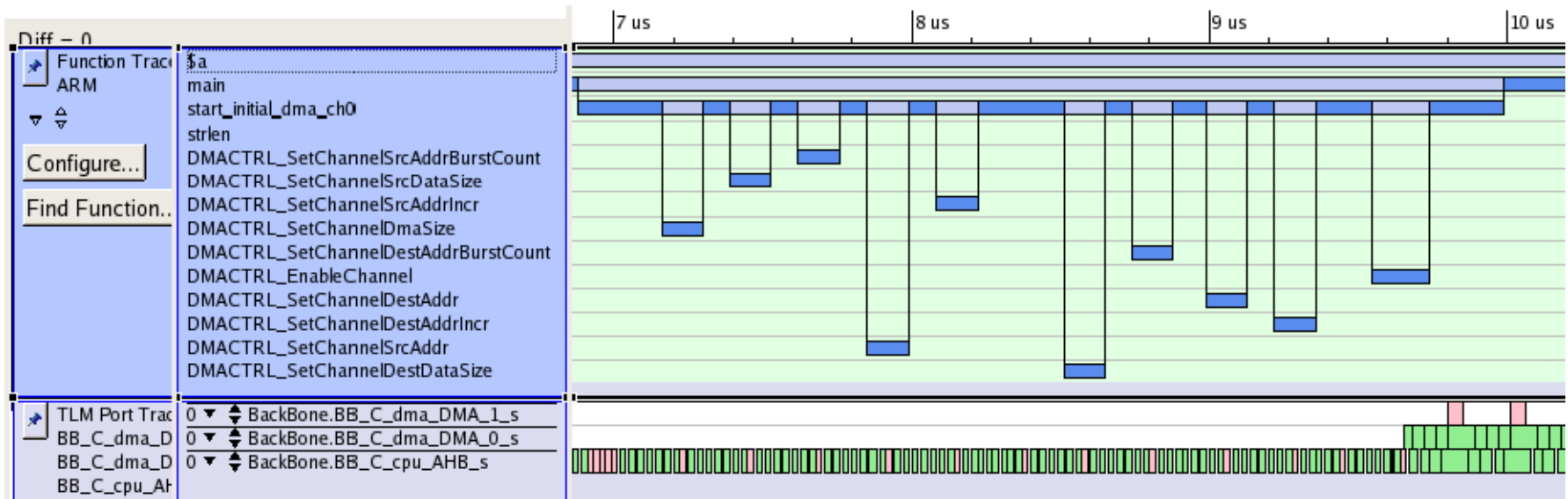
main.stl

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

Socket Transaction Language (STL, from OCP-IP)

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

Generating "Elastic" Trace Files



main.stl

start_initial_dma_ch0.stl

SetChannelDmaSize.stl

```

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

```

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

```

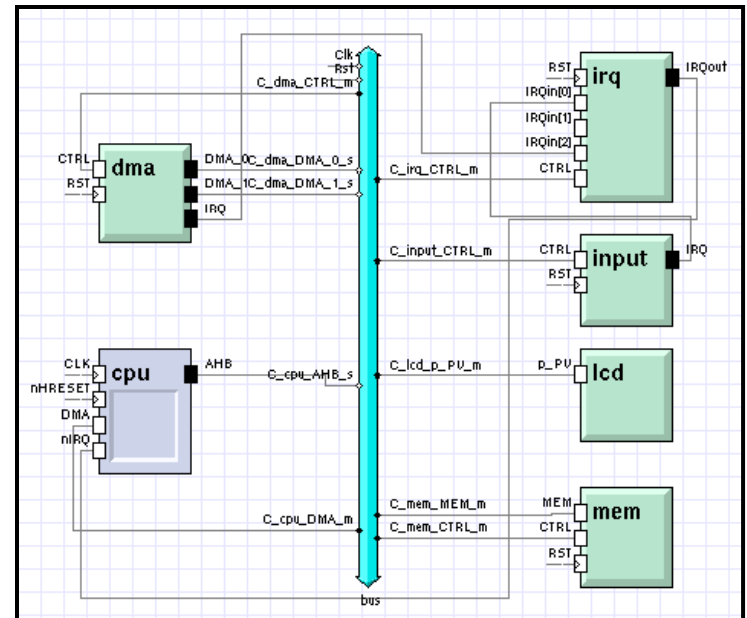
wait swi=5
...
0 read 0xcf8
wait response=-1
0 write 0xc000044 0x100
wait response=-1
0 read 0xcfc
...
raise swi=124
    
```

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

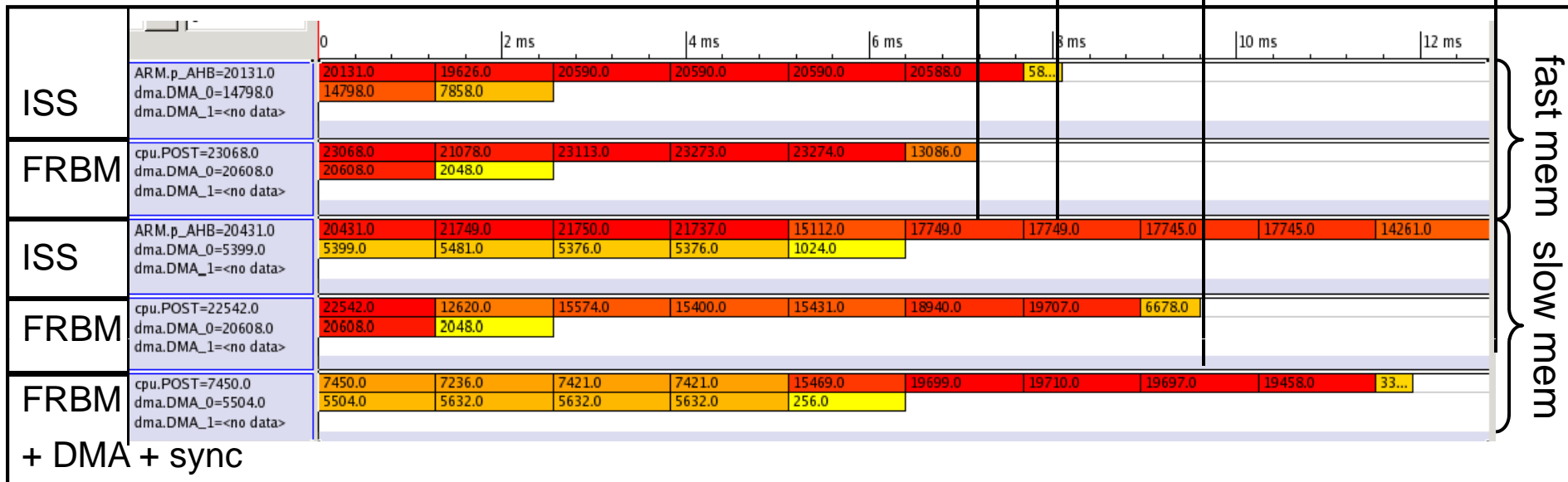
- Goal
 - Validate expressiveness of performance results by comparing FRBM and ISS based virtual prototypes
- Scenario
 - CPU programs DMA
 - DMA loads picture
 - CPU filters and stores picture
- Experiment
 - Record STL trace with ARM 968 LT ISS and AT bus
 - Replace AT bus with cycle accurate AHB bus and modify memory latency
 - Replace ISS and DMA with 3 GFRBMs
 - Compare analysis results and measure accuracy of traces



AHB Results

~11.5%

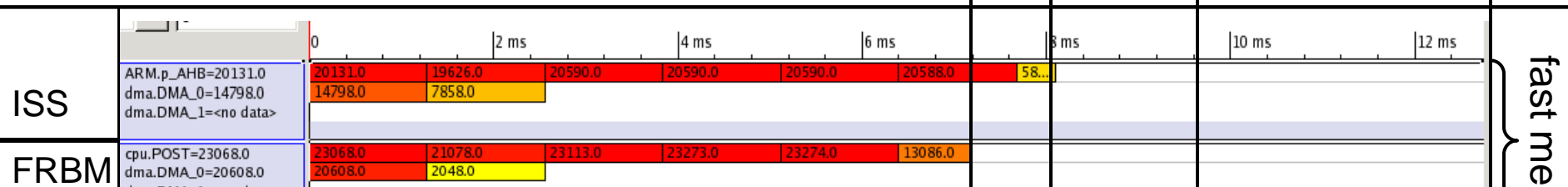
~25%



AHB Results

~11.5%

~25%



fast mem slow mem

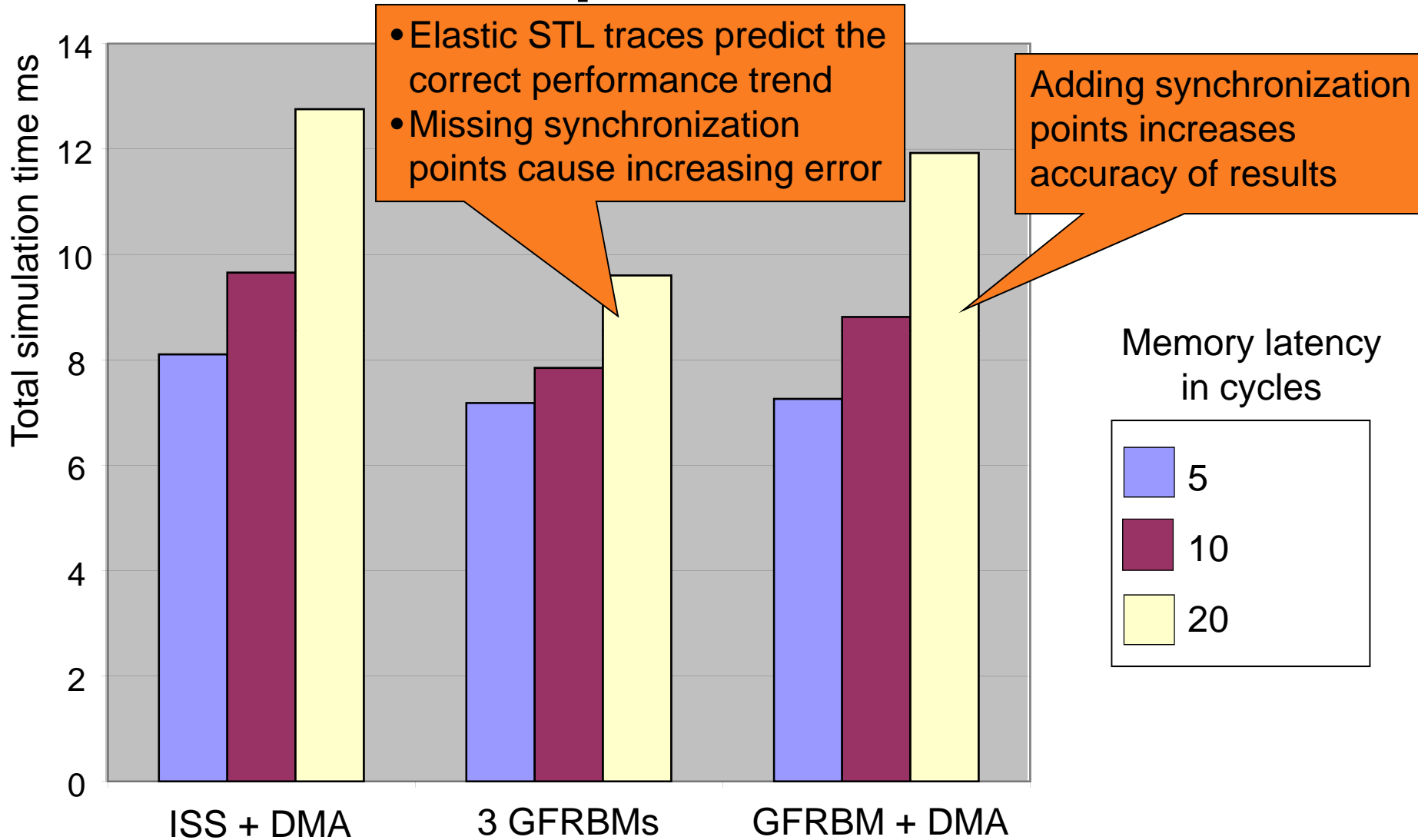
DMA finishes, then CPU starts next DMA

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

Use functional DMA controller

Add waiting for HW interrupt to CPU trace

Architecture Exploration Results



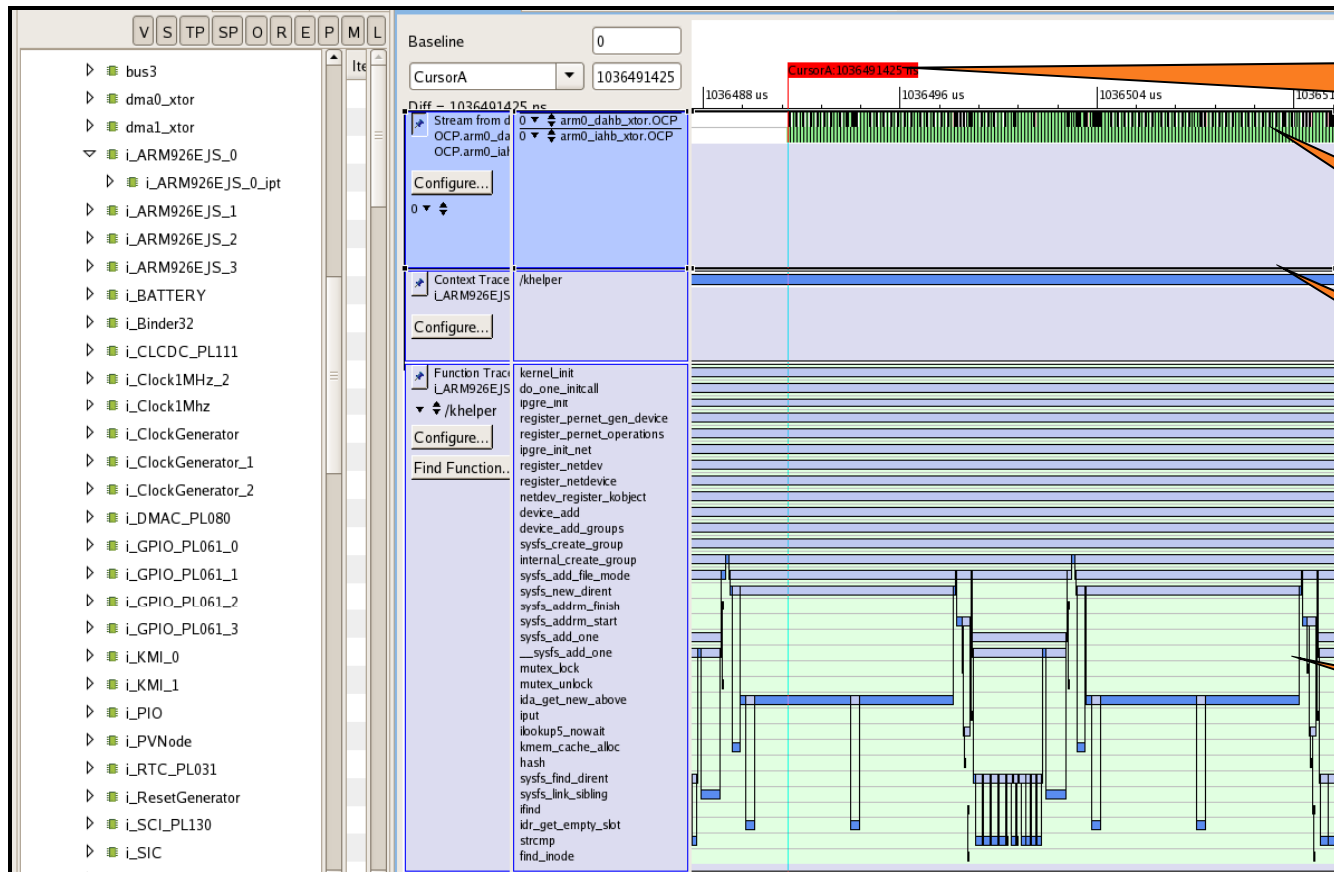
Case Study Results

- 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)